WEBSITE BOOK EDITION

Pixels → Perception

從像素到理解：視覺、語言與學習

基礎・影像・語言・學習・理論

第 3 冊（共 4 冊）

機器學習與電腦視覺深度學習

繁體中文版・第 3 冊

2026-11-05

https://blog.shyandram.dev/

本書說明

本冊是 Pixels → Perception 網站完整課程分成的第 3 冊（共 4 冊），主題為「機器學習與電腦視覺深度學習」，收錄 44 篇已發布教學與 0 篇部落格文章，依網站的學習單元順序編排。這是網站內容的書籍排版版本，並非另經同儕審查的出版品。

正文、程式碼、公式與參考文獻沿用網站內容。練習提示在本書中直接展開；靜態影像保留，互動示範則提供網站連結。原本的影像比較器改列為輸入與輸出影像。篇末參考文獻的編號僅適用於該篇教學。

本站內容仍在審閱中。製作過程使用 AI 協助，內容可能仍有錯誤或疏漏；請搭配原始文獻閱讀，並以網站的後續修訂為準。

兩種語言、各冊分別提供 PDF。連到同冊教學的連結會在本 PDF 內跳轉；連到其他冊的教學、特定小節、互動操作、筆記本與外部資料仍連至網站。內容快照: 2026-11-05

來源版本: `2b3c30d`

https://blog.shyandram.dev/

目錄

I 機器學習 **1**

ML 01・機器學習入門 **2**

ML 01A・學習所需的線性代數 29

ML 01B・機率與資訊理論 49

ML 01C・數值計算與梯度檢查 70

ML 02・類神經網路：從感知器到深度網路 **92**

ML 03・卷積、影像濾波與 CNN **115**

ML 04・循環與遞迴神經網路 **137**

ML 05・注意力與視覺 Transformer **157**

ML 06・表示學習與自監督 **176**

ML 06A・線性因子模型與自編碼器 197

ML 07・生成模型：自編碼器、VAE、GAN 與正規化流 **218**

ML 08・擴散模型與流匹配 **239**

ML 09・視覺自迴歸：像素、Tokens 與尺度 **271**

ML 10・基礎模型與視覺語言學習 **291**

ML 11・評估、效率與部署 **311**

II 電腦視覺深度學習 **336**

DLCV 01・深度學習的影像處理與電腦視覺 **337**

DLCV 01A・深度視覺模型的訓練實務 360

DLCV 01B・網路的積木與組合：分類與影像復原網路 387
DLCV 01C・模型壓縮與高效推論 414

DLCV 02・深度影像分類 **440**
DLCV 02A・臉部辨識與度量學習 472

DLCV 03・物件偵測：從 R-CNN 到 DETR **498**
DLCV 03A・以 YOLOv4 讀懂單階段偵測 524
DLCV 03B・三維物件偵測：光達、相機與鳥瞰視角 557

DLCV 04・影像分割：語意、實例與全景分割 **586**

DLCV 05・人體姿態與關鍵點估計 **612**

DLCV 06・影像中的文字：OCR 與文件理解 **638**

DLCV 07・物件追蹤：從單一物件到多物件 **666**

DLCV 08・影片理解與動作辨識 **693**

DLCV 09・深度影像復原：低階視覺 **717**
DLCV 09A・非盲影像超解析度 757
DLCV 09A.1・盲與真實世界影像超解析度 781
DLCV 09A.2・單步擴散超解析度 801
DLCV 09A.3・影片超解析度：用時間找回細節 811
DLCV 09B・生成式影像復原：取樣出一個合理的答案 836
DLCV 09C・低光影像增強 860

DLCV 10・影像融合：互補來源與有用輸出 **890**

DLCV 11・三維重建：從多視角幾何到 NeRF 與 Gaussian Splatting **921**
DLCV 11A・深度估計與光流 949

DLCV 12・影像生成：從 GAN 到文字生成影像系統 **975**
DLCV 12A・影片生成 997
DLCV 12B・單步生成 1022

DLCV 13・世界模型：影片、潛在動態與規劃 **1035**

DLCV 14・穩健性：分布偏移、對抗樣本與異常 **1055**

篇 I

機器學習

ML 01・機器學習入門

電腦如何從資料中學習：學習範式、線性迴歸與邏輯迴歸、成本函數、梯度下降、過擬合與模型評估，全部用 NumPy 從頭實作。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/introduction/

你將學到

- 定義輸入、特徵、目標值、參數與損失。
- 依使用的回饋區分學習範式。
- 以梯度下降與特徵縮放擬合線性迴歸。
- 用邏輯迴歸分類，並以驗證資料診斷擬合情況。
- 選擇符合任務的指標與基準。

先看全貌

大多數程式都像食譜：有人把問題想清楚，再把每一個步驟寫下來。機器學習把這件事反過來。我們寫一個**有彈性**、裡面有很多可調整數字的程式，給它看大量「正確完成任務」的例子，再讓一個演算法去調整這些數字，直到程式把任務做好。範例取代了食譜。

> **用最簡單的話說**
>
> 想想你小時候怎麼學會分辨貓和狗。沒有人給你一本規則手冊（「尖耳朵加鬍鬚、不會汪汪叫，就是貓」）。大人只是一次又一次指著說「貓」、「狗」、「貓」，你猜錯了就被糾正，直到你「就是知道」。機器學習也是這樣：我們給電腦看很多附上正確答案的例子，它先猜，我們告訴它錯了多少，它每次把猜測稍微修正一點，直到很少出錯。

為什麼重要：幾乎所有現代電腦視覺系統，從手機的臉部解鎖到醫學影像分析，都是學習出來的模型，而不是手寫的程式。本篇的幾個概念：帶有參數的模型、衡量誤差的成本、讓成本下降的梯度下降，以及用另外的資料檢查結果，正是今天訓練最大型神經網路所用的同一套想法，只是模型的規模不同。把這幾個概念弄清楚之後，本系列接下來的內容（ML 02 與各篇深度學習教學）主要就是在談「用**哪一種**模型」與「用**哪一種**成本」。

> **關於本篇**
>
> 本篇是由我先前的筆記〈Introduction to Machine Learning〉[1] 擴寫而成的正式教學（原文為英文）。那份筆記是跟著吳恩達（Andrew Ng）的 Machine Learning Specialization [4] 與《Deep Learning》一書 [3] 整理的。

什麼是機器學習？

白話版。機器學習系統藉由觀察資料來把任務做得更好，而不是由程式設計師寫下每一個決定。

用學習取代寫程式

這個想法由來已久。1959 年，Arthur Samuel 描述了一個學會下西洋跳棋（checkers）、而且下得比他本人還好的程式：它下了很多盤棋，並據此調整自己評估棋局的方式 [2]。他的重點是：一台能從經驗中學習的電腦，可以省去這項任務原本需要的大量細部手寫程式。那句著名的一行定義，把機器學習說成「讓電腦不需明確寫程式就能學習的研究領域」，其實是後人對這個想法的轉述，通常被歸於 Samuel，但並不是他論文裡的原句。

教科書中常見一個更具操作性的定義，包含三個部分：一個**任務**（例如辨識數字）、一個**效能指標**（辨識正確的比例），以及**經驗**（一批有標註的數字影像）。如果程式在這個任務上的效能隨著經驗增加而提升，我們就說它在**學習** [3]、[7]。

影像的例子：規則與學習

我們貫穿全篇的例子是 scikit-learn 的 digits 資料集 [5]、[6]：1,797 張手寫數字的灰階影像，每張只有 8×8 個像素，數值從 0 到 16。就像〈Images as Arrays〉介紹的，每張影像只是一個小小的數字陣列，攤平之後就是 64 個數字組成的向量。

假設我們只需要分辨 **0** 和 **1**。程式設計師可能會寫一條規則：「1 很細、0 很寬，所以數一數有幾欄有墨跡；不超過五欄就是 1。」這條規則容易理解，對大多數直立的 1 也是對的。但人們寫 1 的方式各不相同，有的傾斜、有的帶襯線或底座，而傾斜的 1 佔的欄數和 0 一樣多（圖 1）。這條規則大約只對了 79%。想修好它就得加更多規則（「除非它是斜的 ……」），而每條新規則又會帶來新的例外。

機器學習的做法則**不寫任何**針對數字的規則。它挑一個有彈性的函數家族（這裡是邏輯迴歸，本篇後面會從頭實作），再讓演算法從範例中選出其中的 65 個數字。在沒看過的影像上，它把 0 和 1 完全分對；而且一模一樣的程式碼，不加任何新規則就能處理全部十個數字：

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

digits = load_digits()                      # 1,797 images, 8x8 pixels, values 0..16
X, y = digits.data, digits.target           # X: (1797, 64) flattened pixels; y: 0..9

# Hand-written rule for "0 versus 1": a 1 is thin, a 0 is wide.
keep = (y == 0) | (y == 1)
imgs, labels = digits.images[keep], y[keep]
width = (imgs.max(axis=1) > 4).sum(axis=1)   # how many columns contain ink
rule = np.where(width <= 5, 1, 0)
print("hand rule, 0 vs 1:", round((rule == labels).mean(), 3))          # 0.789

# Learned rule for the same task, judged on images it never saw.
```

```
X01 = imgs.reshape(len(imgs), -1)
Xa, Xb, ya, yb = train_test_split(X01, labels, test_size=0.3, random_state=0)
print("learned, 0 vs 1:", LogisticRegression(max_iter=1000).fit(Xa, ya).score(Xb, yb))   #
 ↪  1.0

# The same recipe scales to all ten digits with no new rules.
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.25, random_state=0)
print("learned, 10 digits:",
      round(LogisticRegression(max_iter=5000).fit(X_tr, y_tr).score(X_te, y_te), 3))  # 0.953
```

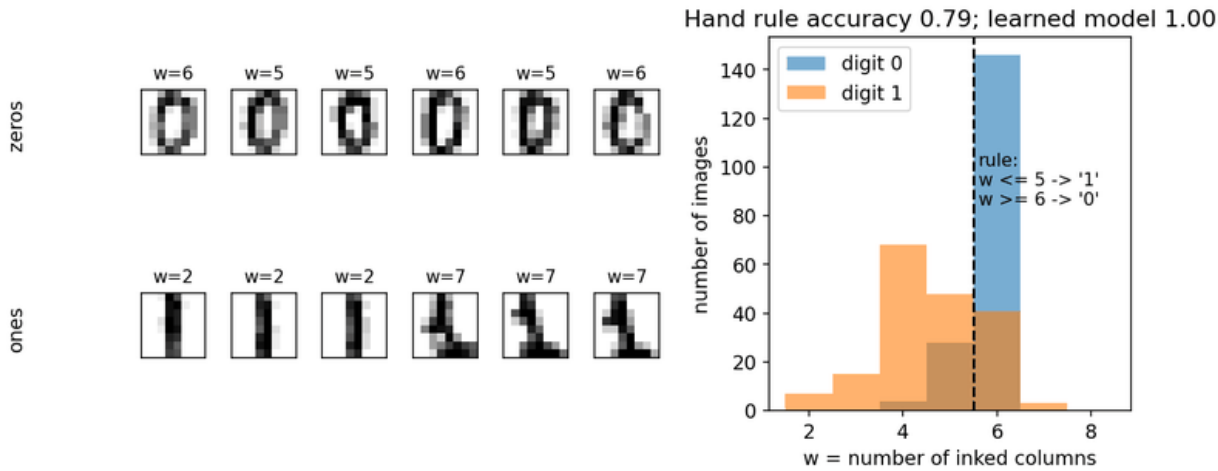


圖 *1* — 為什麼手寫規則很脆弱。左：*digits* 資料集中的 *0* 與 *1*；傾斜的 *1* 和 *0* 一樣寬。右：「數有墨跡的欄數」這個特徵在兩類之間大量重疊，所以任何門檻都會犯很多錯。學習出來的模型則同時使用全部 *64* 個像素。

這個小實驗包含了支持機器學習的全部理由：當規則難以用文字說清楚、範例卻很容易取得時，就從範例中學出規則。它也顯示了代價：學到的模型是一串 65 個數字，比「數欄數」難讀得多。

學習範式

白話版。先問回饋來自哪裡：外加標籤、未標記資料的結構、資料本身產生的目標，或動作帶來的獎勵。

範式	資料長什麼樣子	學到什麼	典型例子
監督式	成對的 (x, y)：輸入與「正確答案」	從 x 到 y 的函數	數字影像 → 數字標籤
非監督式	只有輸入 x	結構：群組、方向、離群值	把相似的數字分在一起
自監督式	只有輸入 x，但把 x 的一部分藏起來當作目標	通用的表示（representation）	預測被遮住的字詞或影像區塊
強化學習	代理人的動作，以及動作換來的獎勵	策略：在每種情況下該採取哪個動作	玩遊戲、調校聊天模型

我原本的筆記列了第一、第二和第四種；自監督式學習後來已成為預訓練大型模型的主要方式，所以這裡讓它單獨佔一列 [9]、[10]。

監督式學習

在監督式學習（supervised learning）中，學習者會拿到正確答案。資料是一組範例 $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$，其中 $x^{(i)}$ 是輸入，$y^{(i)}$ 是期望的輸出（稱為**標籤**或**目標值**），m 是範例數。目標是找到一個函數 f，使 $f(x)$ 接近 y，**而且對資料中沒有的新輸入也一樣**。最後這一點稱為**泛化**（generalization），它才是真正的目標；把範例背起來很容易。

非監督式學習

在非監督式學習（unsupervised learning）中沒有答案，只有輸入 $\{x^{(i)}\}$。目標是從資料中找出有意思的東西：相似範例組成的群組、精簡的描述，或是格格不入的範例。由於沒有唯一「正確」的輸出，這類方法很適合用來探索資料，但也比較難評估。

自監督式學習

自監督式學習（self-supervised learning）使用沒有標註的資料，但把一部分藏起來、要模型預測被藏的部分，藉此把問題變成監督式問題。在文字上，遮住一個字詞，從句子的其餘部分預測它。在影像上，遮住大部分的區塊再把它們重建出來：遮罩自編碼器（masked autoencoder）會遮住一張影像 75% 的區塊，訓練網路把它們補回來 [12]。LeCun 與 Misra 認為，自監督式學習讓模型能從原始觀察中累積廣泛的背景知識，因為標籤是免費得到的 [9]。

重點不在「填空」本身，而在於：一個很會填空的模型，必定已經學到很多關於資料結構的知識。這個學到的內部**表示**可以再用很少的標籤，調整去做許多其他任務。這正是**基礎模型**（foundation model）背後的做法：在大規模、廣泛的資料上訓練（通常是自監督式），再調整去做各式各樣的下游任務 [10]。GPT-3 只是被訓練來在大量文字上預測下一個字，卻能憑提示中給的幾個例子完成許多語言任務，不需要再訓練 [11]。

強化學習

在強化學習（reinforcement learning, RL）中，一個**代理人**（agent）與**環境**互動。每一步它觀察一個**狀態**、選一個**動作**，然後得到一個**獎勵**（一個數字）和新的狀態。沒有人告訴它正確的動作；它必須自己發現哪些動作能在長期累積較高的總獎勵，而獎勵常常在造成它的動作之後很久才出現。學到的是一個**策略**（policy），也就是在每個狀態下選擇動作的規則 [8]。Samuel 的跳棋程式就是這種情境的早期例子 [2]。

今天，強化學習最為人所知的用途是調校大型語言模型。在 Ouyang 等人描述的**基於人類回饋的強化學習**（reinforcement learning from human feedback, RLHF）[13] 中，先用人寫的示範答案對預訓練語言模型做微調（監督式學習）。接著由人對模型針對同一個提示所產生的幾個答案排序，再訓練一個獨立的**獎勵模型**來預測這些排序。最後用強化學習最佳化語言模型，讓它產生獎勵模型評分高的答案。作者回報，標註人員偏好他們 13 億參數的調校模型，勝過 1,750 億參數的

GPT-3 [13]。更近期，DeepSeek-R1 用大規模強化學習，搭配可自動檢查的獎勵（例如數學答案是否正確），來提升逐步推理的能力 [14]。

注意 RLHF 串起了三種範式：自監督式預訓練、監督式微調，以及強化學習。真實的系統會自由混用各種範式；上面那張表是一張地圖，而不是一道道牆。

監督式學習：迴歸與分類

白話版。監督式學習主要有兩種。**迴歸**預測一個數字；**分類**預測一個類別。

迴歸

在迴歸（regression）中，目標值 y 是一個實數，所以可能的輸出有無限多個。例如：預測股價、用之前的考試成績預測學生的期末成績，或從臉部影像預測一個人的年齡。最後一個例子說明：輸入可以是一張影像，輸出仍然只是一個數字。

分類

在分類（classification）中，目標值是一小組固定類別中的某一個。例如：垃圾郵件或非垃圾郵件（兩類，二**元分類**）、臉型的類別，或是腫瘤類型，如良性、惡性第 1 型、惡性第 2 型（**多類別分類**）。類別是名稱，不是數量：「第 2 類」並不是「第 1 類」的兩倍。分類器通常會對每個類別輸出一個機率，再挑最可能的那一個。

```
from sklearn.datasets import make_regression, make_classification
from sklearn.linear_model import LinearRegression, LogisticRegression

# Regression: the target is a real number.
X_r, y_r = make_regression(n_samples=100, n_features=1, noise=10.0, random_state=0)
reg = LinearRegression().fit(X_r, y_r)
print(reg.predict(X_r[:3]).round(1), y_r[:3].round(1))   # numbers vs numbers

# Classification: the target is one of a few labels.
X_c, y_c = make_classification(n_samples=100, n_features=2, n_redundant=0,
                               n_clusters_per_class=1, random_state=0)
clf = LogisticRegression().fit(X_c, y_c)
print(clf.predict(X_c[:5]), y_c[:5])                     # labels vs labels
print(clf.predict_proba(X_c[:2]).round(3))               # class probabilities
```

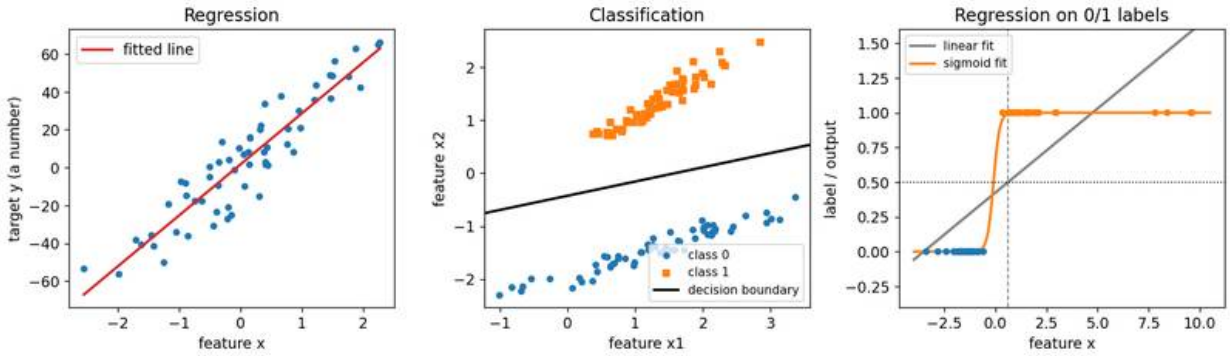


圖 2 — 左：迴歸讓曲線穿過數值目標。中：分類用決策邊界分開有標籤的點。右：對 *0/1* 標籤擬合一條迴歸直線（灰色），輸出會超出 $[0, 1]$，而且遠處的點會拉動它跨過 *0.5* 的位置；*sigmoid*（邏輯迴歸，橘色）才是合適的工具。

為什麼不用迴歸來做分類？

把類別編成 0 和 1、擬合一條迴歸直線、大於 0.5 就判為 1，這個做法很誘人。在簡單的資料上它也許可行，但通常表現很差，原因有兩個，都能在圖 2（右）看到。第一，直線會輸出 −0.4 或 1.7 這類數值，它們不是機率。第二，平方誤差會懲罰那些**已經分對**、只是離邊界很遠的點，所以少數極端但容易的範例會把直線拉過去，讓它跨過 0.5 的位置移到錯誤的地方。本篇後面的邏輯迴歸同時解決了這兩個問題。

非監督式學習

白話版。即使沒有答案，我們仍然可以問：哪些範例屬於同一群？怎樣用更少的數字描述每個範例？哪些範例很奇怪？

分群

分群（clustering）把相似的範例歸在一起。經典的演算法是 **k-means** [15]。先決定群數 K，從 K 個中心點出發，然後重複兩個步驟直到不再變化：把每個範例指派給最近的中心，再把每個中心移到被指派給它的範例的平均位置。每一步都只會讓範例到其中心的總平方距離

$$\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

變小或不變，其中 μ_k 是第 k 群的中心，$c^{(i)}$ 是範例 i 被指派到的群。所以演算法一定會停下來，但可能停在不好的局部解，這也是為什麼要從好幾個隨機起點各跑一次。Jain 回顧五十年分群研究的文章 [15] 強調，一般而言沒有「最好」的分群演算法：「正確」的群組取決於「相似」對這份資料代表什麼意思。

一個常見的混淆：**k 近鄰法**（k-nearest neighbors, KNN）**不是**分群方法。KNN 對一個新範例分類的方式，是讓 k 個最相似的**有標籤**訓練範例投票，所以它是監督式方法 [5]。k-means 的 k 是群數；KNN 的 k 是鄰居數。

降維

降維（dimensionality reduction）用更少的數字描述每個範例，同時盡量保留有用的資訊。**主成分分析**（principal component analysis, PCA）找出資料變化最大的幾個方向，再把資料投影到其中前幾個方向上；新的座標彼此不相關，並依序盡可能捕捉最多的變異 [16]。兩個 PCA 座標就能把 64 個像素的數字影像變成平面上的點，讓我們看得到（圖 3 中）。

異常偵測

異常偵測（anomaly detection）找出與其他資料模式不符的範例：詐騙交易、瑕疵零件、損壞的影像。Chandola、Banerjee 與 Kumar 的綜述 [17] 依照各技術對「正常長什麼樣子」所做的假設，把眾多技術加以分類。一個簡單的版本使用 PCA：在資料上擬合幾個主成分，用它們重建每個範例，再標出重建誤差大的範例，因為主成分只描述了常見的部分。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

X, y = load_digits(return_X_y=True)             # y is used only to check, never to fit

# Clustering: ten groups, found without looking at y.
km = KMeans(n_clusters=10, n_init=10, random_state=0).fit(X)
purity = sum(np.bincount(y[km.labels_ == k]).max() for k in range(10)) / len(y)
print("cluster purity:", round(purity, 3))      # 0.792: most clusters are one digit

# Dimensionality reduction: 64 numbers per image -> 2 numbers.
Z = PCA(n_components=2).fit_transform(X)        # shape (1797, 2)

# Anomaly detection: images that a 20-component PCA rebuilds badly are unusual.
pca20 = PCA(n_components=20).fit(X)
err = ((X - pca20.inverse_transform(pca20.transform(X))) ** 2).sum(axis=1)
print("most unusual images:", np.argsort(err)[-5:])
```

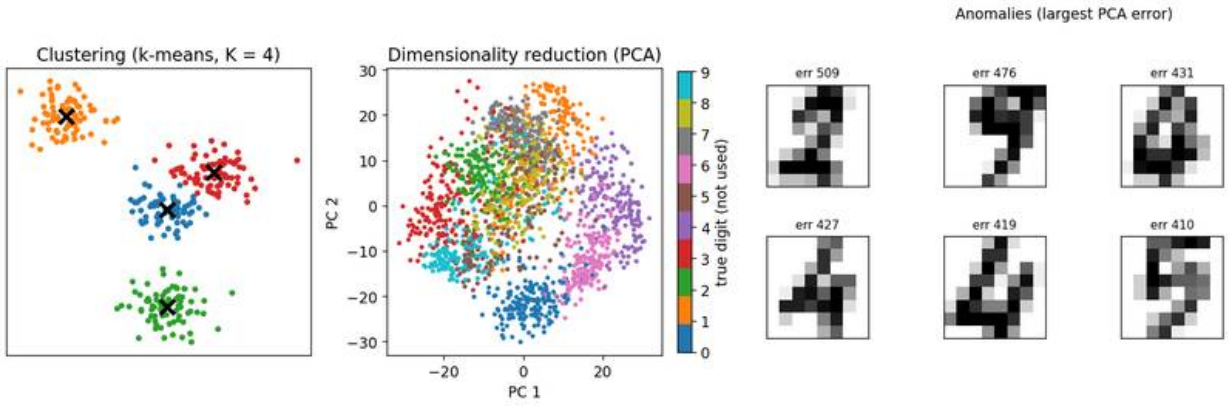


圖 3 — 非監督式學習。左：*k-means* 把點聚集在 K 個中心周圍。中：*PCA* 把每個 *64* 像素的數字壓縮成兩個數字（顏色是 *PCA* 從沒看過的真實數字）。右：*PCA* 重建誤差最大的數字，正是看起來最奇怪的那些。

程式中的純度（purity）檢查只在分群**之後**才用到標籤，用來看找出的群組是否和數字吻合。大約 79% 的影像落在以它自己的數字為主的群裡，即使 k-means 從沒看過任何標籤。

學習問題的詞彙

白話版。在談數學之前，我們需要替各部分取名字：什麼進去、什麼出來、正確答案是什麼、模型是什麼，以及怎麼衡量它的錯誤。

輸入、輸出與目標值

- **特徵**（features，輸入）x：描述一個範例的數字。對數字影像來說是 64 個像素值；對一棟房子來說是坪數和房間數。有 n 個特徵時寫成 $x = (x_1, \dots, x_n)$，x_j 是第 j 個特徵。
- **目標值**（target）y：正確的輸出，也叫標籤或**真值**（ground truth）。只有監督式學習有目標值。
- **預測值**（prediction）$\hat{y}$（讀作 y-hat）：模型對某個輸入的輸出。帽子代表「估計的」。
- **訓練範例** i：成對的 $(x^{(i)}, y^{(i)})$。括號中的上標是索引，不是次方。$x_j^{(i)}$ 是範例 i 的第 j 個特徵。

模型與參數

模型是把特徵對應到預測值的函數。我們寫成 $f_{w,b}$，表示它取決於稱為**參數**的可調整數字，這裡是 w（**權重**，weights）與 b（**偏差**，bias）：

$$\hat{y} = f_{w,b}(x).$$

選擇 f 的**形式**（直線、曲線、神經網路）是設計上的決定；選擇 w 和 b 的**數值**則是學習要做的事。由我們手動設定的數字，例如學習率或多項式的次數，稱為**超參數**（hyperparameters），以和學習出來的參數區分。

成本函數

損失（loss）$L(\hat{y}, y)$ 衡量一個預測錯了多少。**成本函數**（cost function）$J(w, b)$ 把損失對整個訓練集取平均，因此它是一個數字，說明在某組參數下整個模型錯了多少。模型是 f；成本是 J。把兩者分清楚可以避免很多混淆：$f_{w,b}(x)$ 吃一個輸入、吐出預測值；$J(w, b)$ 吃一組參數、吐出誤差。

於是訓練的**目標**就很明確：找出讓 J 盡可能小的參數，使所有訓練範例的 $\hat{y}^{(i)}$ 都接近 $y^{(i)}$。

訓練集、驗證集與測試集

資料會切成三份，各有不同的工作：

- **訓練集**用來擬合參數；
- **驗證集**在開發過程中用來比較模型、選擇超參數；
- **測試集**只在最後使用一次，用來估計最終模型在新資料上的表現。

測試集絕不能影響任何決定，否則估計值會過於樂觀。我們會在過擬合那一節回來說明**為什麼**要這樣切。

僅線上版：*Domingos*：機器學習實務中的關鍵觀念 — *https://blog.shyandram.dev/zh-tw/learn/ml/introduction/papers/domingos* 機器學習實務中的關鍵觀念/

線性迴歸

白話版。在資料中畫一條直線。這條線有兩個旋鈕：斜率和高度。學習就是轉動旋鈕，直到直線盡可能貼近所有的點。

模型

只有一個特徵時，線性迴歸（linear regression）模型是

$$f_{w,b}(x) = wx + b,$$

其中 x 是特徵，w 是斜率（x 增加 1 時預測值改變多少），b 是截距（$x = 0$ 時的預測值）。改變 w 會讓直線傾斜；改變 b 會讓直線上下移動（圖 4 左）。

有 n 個特徵時，同一個模型是加權總和：

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b = \sum_{j=1}^{n} w_j x_j + b,$$

其中 $\mathbf{w} = (w_1, \ldots, w_n)$ 為每個特徵各存一個權重。常見的技巧是加上一個常數特徵 $x_0 = 1$，偏差就變成另一個權重 w_0，模型縮寫成 $\mathbf{w} \cdot \mathbf{x}$。無論哪種寫法，「線性」指的都是對**參數**是線性的；之後我們會餵給它平方或三次方的特徵，它仍然是線性迴歸。

成本函數：均方誤差

對每個範例，**誤差**（殘差）是 $f_{w,b}(x^{(i)}) - y^{(i)}$。取平方讓誤差變成正數，並讓大誤差比小誤差受到更重的懲罰。對 m 個訓練範例取平均，就得到**均方誤差**（mean squared error, MSE）成本：

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2,$$

其中 m 是訓練範例數，$(x^{(i)}, y^{(i)})$ 是第 i 個範例。多出來的 $\frac{1}{2}$ 只是為了方便：微分平方項時出現的 2 會和它抵消，而且它不會改變最小值的位置。

把成本想像成參數上的一片地形。如果固定 b、只改變 w，J 是一條拋物線：太平或太陡的直線誤差都很大，介於中間的某個斜率最好（圖 4 中）。同時考慮 w 和 b 時，J 是一個碗，我們把它畫成由橢圓組成的等高線圖（圖 4 右）。最好的直線就在碗底。

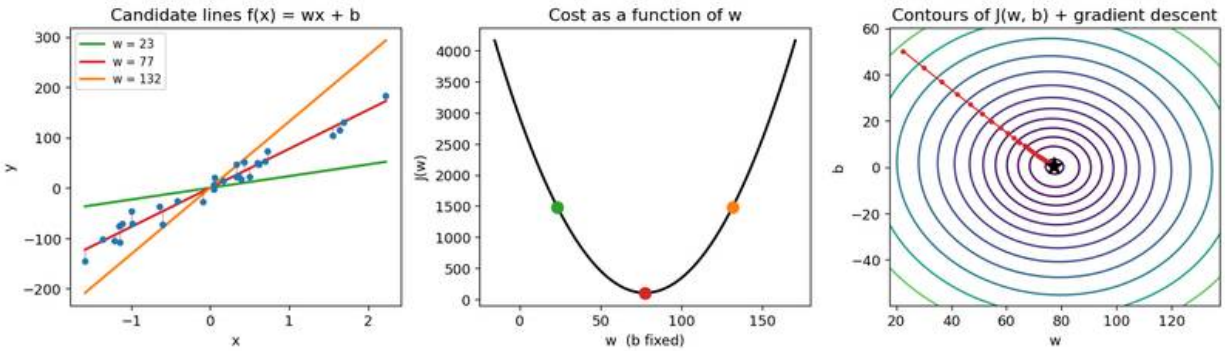


圖 4 — 從直線到成本。左：三條候選直線；垂直線段是最佳直線的殘差。中：固定 b 時，$J(w)$ 是拋物線，每條直線都是上面的一個點。右：$J(w,b)$ 是一個碗（等高線），梯度下降沿著下坡走到碗底。

為什麼這個碗只有一個底

對線性迴歸來說，MSE 成本是 (w,b) 的**凸函數**（convex）：曲面上任兩點之間的直線段都位於曲面上方或曲面上。每個局部最小值都是全域最小值，但只有在設計矩陣（含偏差欄）的各欄線性獨立時，最佳參數才唯一。重複特徵可能形成一整片同樣好的平坦解集合。對 ML 02 的神經網路而言，小梯度也可能出現在鞍點；不能直接假設目標是凸的，或已找到全域最小值。

線性最小平方法也可以直接求解，不必反覆執行梯度下降。正規方程式描述其駐點條件，但數值求解器不一定直接建立它：`np.polyfit` 使用最小平方求解，並可回報縮放後設計矩陣的奇異值 [27]。ML 01A 會解釋這與條件數有什麼關係。這裡介紹梯度下降，是因為它也能用於沒有這類直接解法的模型。

梯度下降

白話版。梯度下降像在霧中走下坡，沿局部斜率的反方向小步前進；地面平坦不代表已到谷底。

更新規則

導數 $\partial J/\partial w$ 告訴我們 w 稍微增加時 J 會怎麼變：正值表示「往右是上坡」，負值表示「往左是上坡」。它的正負號給出方向，大小說明坡有多陡。梯度下降（gradient descent）讓每個參數**逆著**它的導數走一小步：

$$w \leftarrow w - \alpha \frac{\partial J(w,b)}{\partial w}, \qquad b \leftarrow b - \alpha \frac{\partial J(w,b)}{\partial b},$$

其中 $\alpha > 0$ 是**學習率**（步長），$\leftarrow$ 表示「以右邊取代」。兩個導數都要在**目前**的 (w,b) 計算完之後才更新任何一個參數，這叫做**同步更新**。所有偏導數組成的向量就是**梯度** ∇J，它指向上升最快的方向，所以這個更新是朝下降最快的方向移動 [3]。

對線性迴歸的 MSE 成本，連鎖律給出

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)x^{(i)}, \qquad \frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right).$$

用文字讀：w 的梯度是「平均誤差乘以特徵」；b 的梯度是「平均誤差」。如果預測值平均而言太高，b 就往下調。如果預測值主要在 x 很大的地方太高，w 就往下調。有 n 個特徵時，$\partial J/\partial w_j$ 用 $x_j^{(i)}$ 取代 $x^{(i)}$。

從頭實作

```
import numpy as np
from sklearn.datasets import make_regression

X, y = make_regression(n_samples=100, n_features=1, noise=10.0, random_state=0)
x = X[:, 0]                                  # one feature, shape (m,)
m = len(x)

def cost(w, b):
    return ((w * x + b - y) ** 2).sum() / (2 * m)

w, b, alpha = 0.0, 0.0, 0.1
for step in range(200):
    err = w * x + b - y                      # f_wb(x_i) - y_i for every i
    dj_dw = (err * x).sum() / m
    dj_db = err.sum() / m
    w, b = w - alpha * dj_dw, b - alpha * dj_db   # simultaneous update
    if step % 50 == 0:
        print(f"step {step:3d}  J = {cost(w, b):8.2f}")

print("gradient descent:", round(w, 3), round(b, 3))   # 42.619 -0.814
print("closed form     :", np.polyfit(x, y, 1).round(3))  # [42.619 -0.814]
```

成本在 50 步內從大約 800 降到 57，之後就不再變化。梯度下降和精確公式的結果在小數點後三

位都一致。剩下的成本不是零，因為資料中含有任何直線都無法解釋的雜訊。

收斂：導數為零的地方

「重複直到收斂」需要一個定義。在最小值處地形是平的，所以那裡的**導數為零**，更新 $w - \alpha \cdot 0$ 不會改變 w。要注意變成零的是**導數**，不是成本：最小值處的成本只是能達到的最小數值，在上面的例子中是 57。實務上，當兩次迭代之間成本下降得比一個小容許值還少、梯度的大小低於某個門檻，或是用完固定的步數時，就停止。

一個有用的推論：越接近最小值，導數越小，所以即使 α **固定**，步伐 $\alpha\,\partial J/\partial w$ 也會自動變小。梯度下降在接近谷底時會自然放慢。

選擇學習率

學習率是梯度下降最重要的超參數（圖 5）：

- **太小：**每一步都很小。演算法確實在下坡，但可能需要非常多步。
- **剛好：**成本快速下降並穩定下來。
- **太大：**一步跨過谷底跳到山谷另一側，甚至可能比起點還高。參數來回彈跳，可能永遠停不下來（**越過頭**，overshoot），甚至一路飛向無限大（**發散**，divergence）。

對這個固定、使用完整批次的二次成本，每次更新後都畫出同一個 J；足夠小的固定學習率會使它下降。若上升，就值得檢查步長或梯度。但這**不代表每種訓練的損失都必須逐步下降**：小批次取樣、資料增強與隨機層會改變當次觀察的量，有用的隨機更新也可能提高完整資料上的損失 [18]。應用一致的評估方式查看多次更新的趨勢。對這個確定性範例，可用大約 3 倍的間隔測試學習率，選擇成本平穩下降的設定。

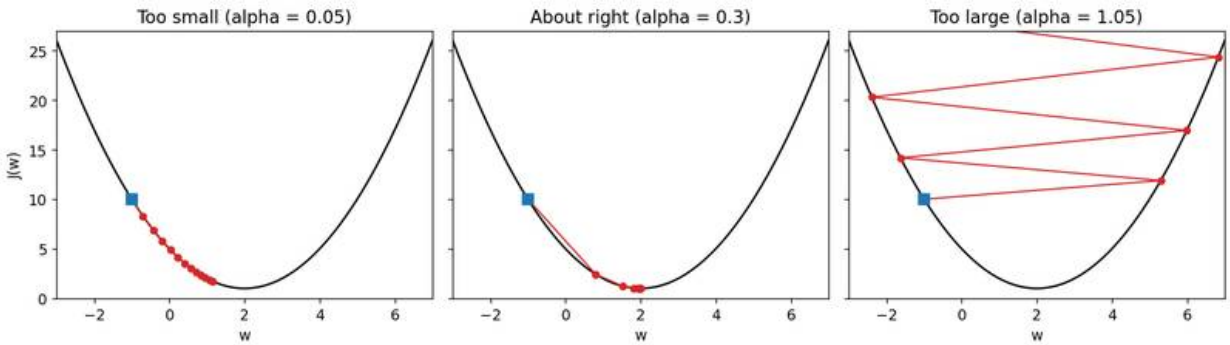


圖 5 — 同一個成本 $J(w)$、同一個起點、三種學習率。太小：進展緩慢。剛好：快速收斂。太大：每一步越過得更遠，迭代發散。

許多最佳化方法會在訓練過程中調整步長：用一個**排程**（schedule）隨時間降低 α，或像 Adam 那樣，用每個參數過去梯度的移動統計量來縮放它的步伐 [19]。這讓「接近最小值時步伐變小」成為明確的設計，而不只是副作用。

批次、小批次與隨機梯度下降

梯度公式要對全部 m 個範例加總。100 個範例沒問題，一億個就是麻煩了。關鍵的洞見是：梯度是一個**平均值**，也就是對資料的期望值，而平均值可以從樣本估計。[3] 對此有清楚的說明，Bottou、Curtis 與 Nocedal [18] 則做了深入的分析。

- **批次梯度下降**（batch gradient descent）每次更新都使用**整個**訓練集。每一步都精確，但也昂貴。
- **小批次梯度下降**（mini-batch gradient descent）先打亂資料，每次更新只用一個隨機的小子集（**小批次**），大小為 B 個範例，通常是 32 到幾百。每一步是對完整梯度帶雜訊但不偏的估計，成本只有 B/m。
- **隨機梯度下降**（stochastic gradient descent, SGD）在嚴格意義上每次更新只用一個範例（$B = 1$）。實務上「SGD」也常指小批次的版本。

完整走過訓練資料一遍稱為一個 **epoch**。注意程式中的命名：深度學習函式庫中的 `batch_size` 參數指的是**小批次**的大小，而不是批次梯度下降的「整個資料集」。

小批次梯度的雜訊不只可以容忍，往往還有幫助：在相同的 epoch 數內，小批次下降比批次下降多做了非常多次更新。Bottou 等人認為，這正是隨機方法主導大規模學習的原因：它們用遠少於精確方法的計算量，就能達到有用的準確度 [18]。

```
import numpy as np
from sklearn.datasets import make_regression

X, y = make_regression(n_samples=10_000, n_features=5, noise=10.0, random_state=0)
m, n = X.shape
rng = np.random.default_rng(0)

def minibatch_gd(batch_size, alpha=0.05, epochs=5):
    w, b = np.zeros(n), 0.0
    for epoch in range(epochs):
        order = rng.permutation(m)                    # shuffle once per epoch
        for start in range(0, m, batch_size):
            idx = order[start:start + batch_size]
            err = X[idx] @ w + b - y[idx]
            w -= alpha * X[idx].T @ err / len(idx)    # gradient estimated on the batch
            b -= alpha * err.mean()
    return ((X @ w + b - y) ** 2).mean() / 2

for bs in [m, 64, 1]:                                 # batch, mini-batch, stochastic
    print(f"batch size {bs:5d}: J = {minibatch_gd(bs, alpha=0.05 if bs > 1 else 0.005):.2f}")
# batch size 10000: J = 6127.00   (only 5 updates in 5 epochs)
# batch size    64: J = 49.93     (785 updates)
# batch size     1: J = 50.64     (50,000 noisy updates)
```

同樣走過資料五遍，批次下降只走了五步，離谷底還很遠；小批次下降則基本上已經收斂（這份資料的雜訊程度讓可達到的最佳 J 大約是 50）。單一範例的 SGD 也能到達，但因為梯度雜訊較大，需要較小的學習率，而且無法有效利用向量化運算。

特徵縮放

白話版。特徵尺度差異可能讓梯度下降來回震盪；縮放能改善地形，但不保證損失曲面變成圓碗。

為什麼尺度很重要

假設特徵 x_1（例如以平方英尺計的房屋面積）的範圍是好幾千，特徵 x_2（房間數）的範圍是 1 到 5。w_1 稍微改變一點，預測值就變很多；w_2 做同樣的改變卻幾乎沒影響。所以 J 沿著 w_1 方向非常陡，沿著 w_2 方向非常平：等高線是細長的橢圓。學習率若小到不會在陡的方向發散，在平的方向上就幾乎沒有進展（圖 6 左）。

標準化與最小–最大縮放

最常見的解法是**標準化**（standardization，z 分數正規化）：

$$x'_j = \frac{x_j - \mu_j}{\sigma_j},$$

其中 μ_j 與 σ_j 是特徵 j **在訓練集上計算**的平均值與標準差。縮放後每個特徵的平均為 0、標準差為 1。另一個做法是**最小–最大縮放**（min–max scaling），$x'_j = (x_j - \min_j)/(\max_j - \min_j)$，把訓練集的範圍對應到 $[0, 1]$，這也是像素強度的常用選擇（8 位元影像除以 255）。兩者在 scikit-learn 中分別是 `StandardScaler` 與 `MinMaxScaler` [5]。

兩條規則可以避免隱微的錯誤。只在訓練集上計算 μ_j, σ_j，再把**同一組**數字套用到驗證、測試和未來的資料。並且記得縮放會改變學到的權重的意義：權重現在代表特徵每增加一個標準差時預測值改變多少。

```
import numpy as np
from sklearn.datasets import make_regression

X, y = make_regression(n_samples=200, n_features=2, noise=5.0, random_state=1)
X = X * np.array([1000.0, 1.0])               # feature 1 now has a ~1000x wider range

def gd(X, y, alpha, steps=500):
    m, n = X.shape
    w, b = np.zeros(n), 0.0
    for _ in range(steps):
        err = X @ w + b - y
        w -= alpha * X.T @ err / m
        b -= alpha * err.mean()
    return ((X @ w + b - y) ** 2).mean() / 2

print("raw,    alpha=1e-7:", round(gd(X, y, 1e-7), 2))    # 158.01; alpha=3e-6 diverges
mu, sigma = X.mean(axis=0), X.std(axis=0)
Xs = (X - mu) / sigma                          # z-score standardization
print("scaled, alpha=0.1 :", round(gd(Xs, y, 0.1), 2))    # 13.47
```

在原始特徵上，任何大於約 2×10^{-6} 的學習率都會發散；用安全的學習率跑 500 步，成本仍卡在 158 左右（跑 5,000 步也還是 158 左右），因為第二個權重幾乎不動。標準化之後，$\alpha = 0.1$ 輕鬆到達最小值（大約 13，也就是雜訊的程度）。

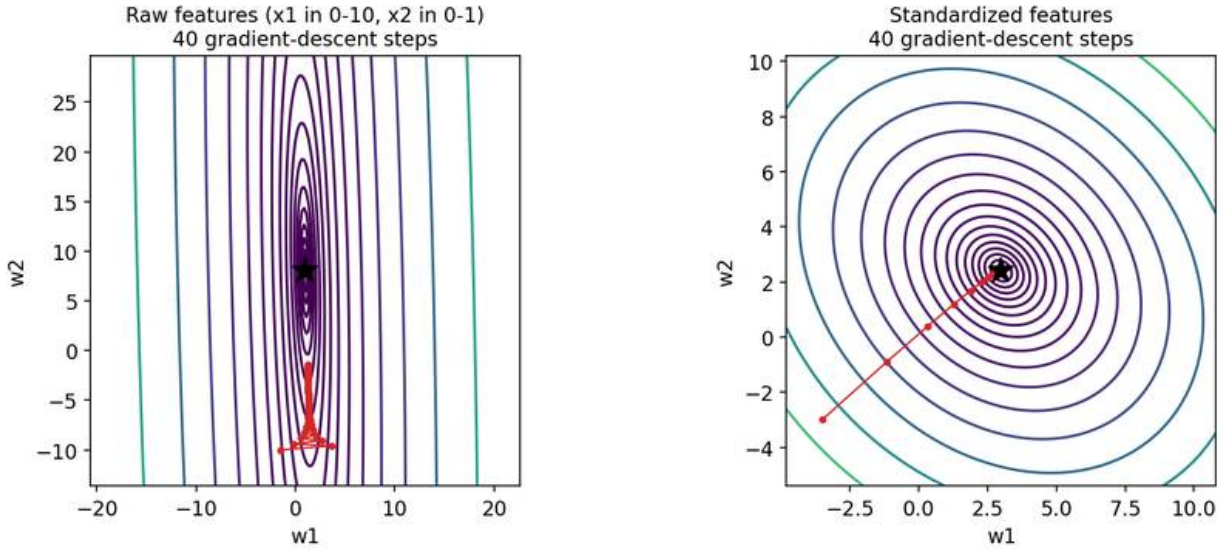


圖 *6* — 特徵縮放重塑了成本地形。左：尺度差異很大的特徵形成狹窄的山谷，梯度下降呈鋸齒狀前進。右：標準化之後，等高線接近圓形，同樣的步數就能到達最小值。

用於分類的邏輯迴歸

白話版。保留線性迴歸的加權總和，但把輸出壓進 0 到 1 之間，讓它可以被解讀成機率。然後選擇權重，讓真實類別的機率越高越好。

Sigmoid 函數

Sigmoid（logistic）函數是

$$g(z) = \frac{1}{1 + e^{-z}},$$

其中 z 是任意實數。它把 $-\infty$ 對應到 0、$+\infty$ 對應到 1、0 恰好對應到 0.5，中間是一條 S 形曲線（圖 7 左）。它還有對稱性 $g(-z) = 1 - g(z)$，以及方便的導數 $g'(z) = g(z)\,(1 - g(z))$。

模型與決策邊界

邏輯迴歸（logistic regression）把線性模型送進 sigmoid：

$$f_{\mathbf{w},b}(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}},$$

並把輸出解讀為標籤為 1 的估計機率，$f_{\mathbf{w},b}(\mathbf{x}) = P(y = 1 \mid \mathbf{x})$。名字雖然叫「迴歸」，它其實是分類方法。做決定時，只要這個機率至少 0.5 就預測為 1。由於 $g(z) \geq 0.5$ 恰好在 $z \geq 0$ 時成立，規則就是

$$\hat{y} = 1 \quad \text{if} \quad \mathbf{w} \cdot \mathbf{x} + b \geq 0, \qquad \hat{y} = 0 \text{ otherwise.}$$

滿足 $\mathbf{w} \cdot \mathbf{x} + b = 0$ 的集合就是**決策邊界**（decision boundary）。有兩個特徵時它是一條直線；一般而言它是一個平的超平面，而 $\mathbf{w}$ 與它垂直。離邊界很遠的點，機率接近 0 或 1；靠近邊界的點，機率接近 0.5。若想得到彎曲的邊界，就餵給模型非線性的特徵，例如 x_1^2 或 $x_1 x_2$；模型對參數仍然是線性的。

成本：交叉熵（對數損失）

為什麼不沿用平方誤差？和 sigmoid 結合之後，平方誤差成本對 $(\mathbf{w}, b)$ **不是**凸的：它會出現平坦的高原和多個山谷。所以邏輯迴歸改用**對數損失**（log loss），也稱為二**元交叉熵**（binary cross-entropy）：

$$L(f, y) = -y \log f \; - \; (1 - y) \log(1 - f),$$

其中 $f = f_{\mathbf{w},b}(\mathbf{x})$ 是預測機率，$y \in \{0, 1\}$ 是真實標籤。每個範例只會用到兩項中的一項。若 $y = 1$，損失是 $-\log f$：$f = 1$ 時為零，$f \to 0$ 時無限增大。若 $y = 0$，損失是 $-\log(1 - f)$，正好是鏡像（圖 7 中）。信心滿滿卻答錯會受到很重的懲罰；信心滿滿且答對則幾乎沒有成本。

成本是整個訓練集上的平均：

$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^{m} \Big[-\, y^{(i)} \log f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - (1 - y^{(i)}) \log \left(1 - f_{\mathbf{w},b}(\mathbf{x}^{(i)})\right) \Big].$$

這個選擇並不隨意。如果把每個標籤看成一次擲硬幣，出現 1 的機率是 $f_{\mathbf{w},b}(\mathbf{x}^{(i)})$，那麼 J 正好是訓練標籤的負對數概似（negative log-likelihood）除以 m。最小化它就是**最大概似估計**（maximum likelihood estimation）[3]。而且搭配 sigmoid 時，這個成本是凸的，所以梯度下降又能找到唯一的全域最小值。

再一次梯度下降

利用 $g' = g(1 - g)$，對數損失的導數會化簡得非常漂亮：

$$\frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}, \qquad \frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)} \right).$$

它和線性迴歸的梯度**形式完全相同**：平均誤差乘以特徵。唯一的差別在 f 的內部。這不是巧合；只要輸出函數和它相配的概似函數成對使用，整個模型家族都有這個性質，這也是梯度下降程式碼這麼容易重複使用的原因之一。

```
import numpy as np
from sklearn.datasets import make_classification
```

```
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, class_sep=1.5, random_state=3)

def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))

def log_loss(p, y, eps=1e-12):
    p = np.clip(p, eps, 1 - eps)              # avoid log(0)
    return -(y * np.log(p) + (1 - y) * np.log(1 - p)).mean()

w, b, alpha = np.zeros(2), 0.0, 0.5
for step in range(1000):
    p = sigmoid(X @ w + b)                    # predicted P(y = 1 | x)
    w -= alpha * X.T @ (p - y) / len(y)       # same form as linear regression
    b -= alpha * (p - y).mean()

p = sigmoid(X @ w + b)
print("log loss:", round(log_loss(p, y), 3))  # 0.06
print("accuracy:", ((p >= 0.5) == y).mean())  # 0.99
print("boundary: %.2f*x1 + %.2f*x2 + %.2f = 0" % (w[0], w[1], b))
```

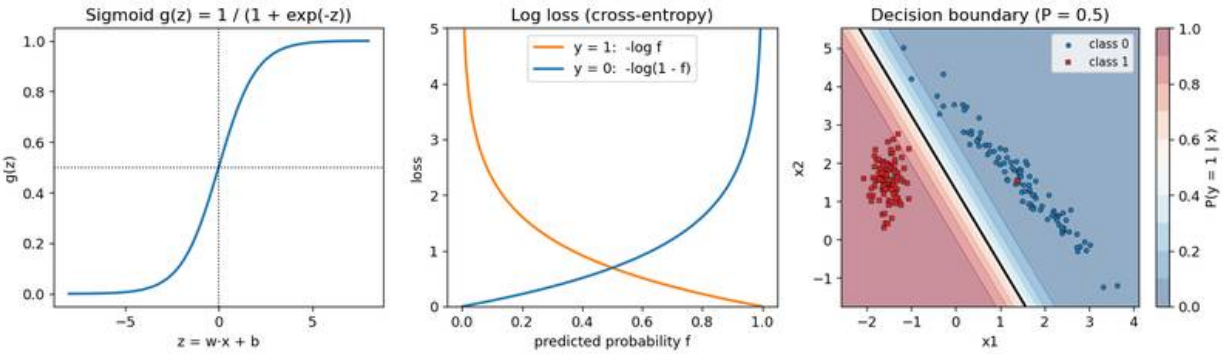


圖 7 — 邏輯迴歸。左：*sigmoid* 把任何分數壓進 $(0, 1)$。中：$y = 1$ 與 $y = 0$ 的對數損失，對信心滿滿的錯誤懲罰很重。右：從頭訓練的模型；底色是預測機率，黑線是決策邊界 $w{\cdot}x + b = 0$。

超過兩個類別

當類別數 $K > 2$ 時，標準的推廣方式是讓每個類別有自己的權重向量和分數 $z_k = \mathbf{w}_k \cdot \mathbf{x} + b_k$，再用 **softmax** 函數把分數轉成總和為 1 的機率：$P(y = k \mid \mathbf{x}) = e^{z_k} / \sum_{l=1}^{K} e^{z_l}$。成本同樣是真實類別的負對數機率的平均（類別交叉熵）[3]。本篇開頭 scikit-learn 的 `LogisticRegression` 處理十個數字時做的就是這件事，而它也正是 ML 02 中大多數分類網路的輸出層。

過擬合、欠擬合與正則化

白話版。訓練誤差高可能是欠擬合或最佳化問題；訓練誤差低但驗證誤差高，則可能過擬合，可嘗試正則化。

親眼看它發生

取 25 個帶雜訊的平滑曲線樣本，擬合不同次數的多項式，也就是在特徵 $x, x^2, \dots, x^d$ 上做線性迴歸（圖 8）：

- **1 次（欠擬合，高偏差）**。直線跟不上曲線。不論在訓練集**還是**新資料上，誤差都很大。
- **3 次（擬合良好）**。曲線跟著趨勢走，忽略雜訊。兩邊的誤差都很小。
- **15 次（過擬合，高變異）**。曲線幾乎精確地穿過每個訓練點，所以訓練誤差接近零。但在點與點之間和點的範圍之外，它劇烈擺盪，新資料上的誤差暴增。

光看訓練誤差無法分辨好模型和過擬合的模型；它甚至偏好過擬合的那個。只有模型沒看過的資料才能揭露差別。這就是我們保留驗證集的原因。

使用驗證集與測試集

流程是：

1. 在**訓練集**上擬合每個候選模型（每種次數、每種正則化強度）。
2. 在**驗證集**上比較候選模型，保留最好的那個。
3. 在**測試集**上回報所選模型的誤差，只做一次。

為什麼要三份而不是兩份？因為在驗證集上從眾多候選中挑出最好的，本身就是一種擬合：勝出者有一部分只是剛好在那些特定範例上運氣好。沒動過的測試集才能給出不偏的最終估計。資料很少時，可以用**交叉驗證**（cross-validation）取代單一的驗證集：把訓練資料切成 k 份，訓練 k 次，每次用不同的一份做驗證，再取平均。

要留意的模式：如果訓練誤差和驗證誤差都很高，模型欠擬合（換更有彈性的模型或更好的特徵）。如果訓練誤差很低、驗證誤差卻高得多，模型過擬合（取得更多資料、簡化模型，或加上正則化）。

L2 正則化

正則化（regularization）保留有彈性的模型，但不鼓勵極端的參數值。15 次多項式那條狂野的曲線，需要非常大、又彼此精細抵消的權重。**L2 正則化**在成本上加入權重平方大小的懲罰：

$$J_{\text{reg}}(\mathbf{w}, b) = J(\mathbf{w}, b) + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2,$$

其中 J 是原本的成本（MSE 或對數損失），$\lambda \geq 0$ 是**正則化強度**（一個超參數），加總的範圍是所有權重，但慣例上不包括偏差 b。$\lambda = 0$ 時回到原本的模型；λ 越大，權重越被拉向零，擬合出的函數越平滑；λ 太大則會欠擬合。對線性迴歸來說，這就是**脊迴歸**（ridge regression），由 Hoerl 與 Kennard 提出，用來在特徵高度相關時穩定最小平方估計 [20]。

從梯度下降的更新可以看出懲罰項做了什麼：

$$w_j \leftarrow w_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{\partial J}{\partial w_j}.$$

在每次一般的更新之前，每個權重都先乘上一個略小於 1 的數。這就是為什麼 L2 正則化在神經網路訓練中也被稱為**權重衰減**（weight decay）[3]。

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.metrics import mean_squared_error

rng = np.random.default_rng(0)
x = rng.uniform(0, 1, 25)
y = np.sin(2 * np.pi * x) + rng.normal(0, 0.25, 25)      # true curve + noise
X = x[:, None]

# 60% train, 20% validation, 20% test
X_tmp, X_te, y_tmp, y_te = train_test_split(X, y, test_size=0.2, random_state=0)
X_tr, X_va, y_tr, y_va = train_test_split(X_tmp, y_tmp, test_size=0.25, random_state=0)

def fit(degree, lam=0.0):
    reg = Ridge(alpha=lam) if lam > 0 else LinearRegression()   # sklearn calls lambda
    ↪  "alpha"
    model = make_pipeline(PolynomialFeatures(degree), StandardScaler(), reg)
    return model.fit(X_tr, y_tr)

for degree, lam in [(1, 0), (3, 0), (15, 0), (15, 1e-2)]:
    model = fit(degree, lam)
    tr = mean_squared_error(y_tr, model.predict(X_tr))
    va = mean_squared_error(y_va, model.predict(X_va))
    print(f"degree {degree:2d}, lambda {lam:g}: train MSE {tr:.3f}, val MSE {va:.3g}")

best = fit(15, 1e-2)                                    # chosen on validation only
print("test MSE:", round(mean_squared_error(y_te, best.predict(X_te)), 3))
```

輸出說明了一切：1 次的訓練／驗證 MSE 是 0.144 ／ 0.292（欠擬合），3 次是 0.031 ／ 0.052（良好），沒有正則化的 15 次訓練 MSE 只有 0.003，驗證 MSE 卻高達數百萬，因為有些驗證點落在訓練點之間很寬的空隙中（例如 $x \approx 0.27$ 到 $x \approx 0.61$ 之間），多項式在那裡劇烈擺盪。對同一個 15 次模型加上溫和的 L2 懲罰（$\lambda = 0.01$），得到 0.041 ／ 0.035，測試 MSE 為 0.075。

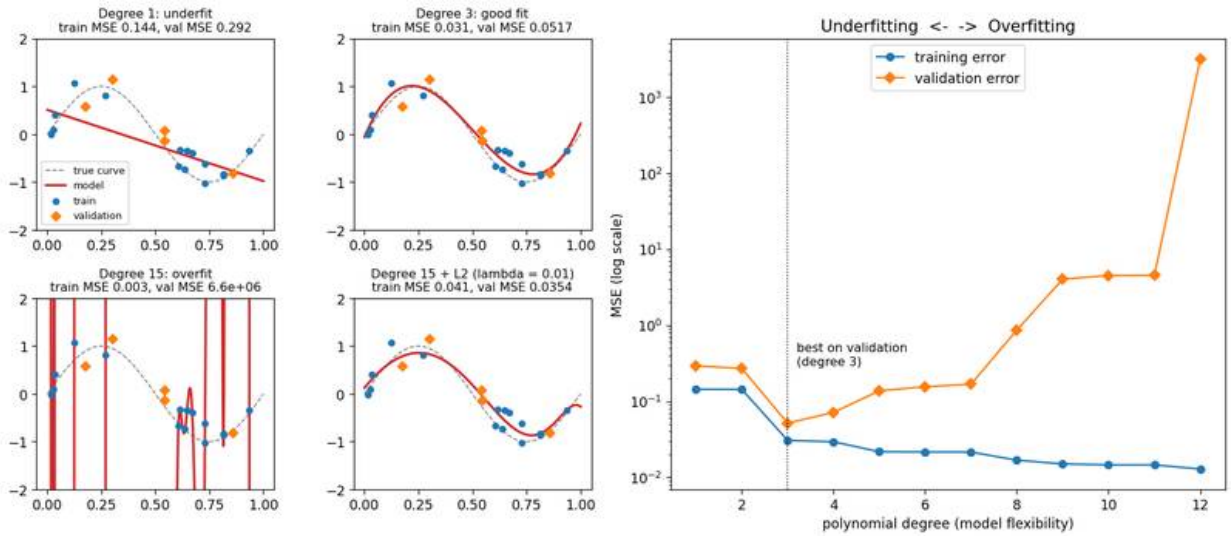


圖 *8* — 欠擬合、良好擬合、過擬合，以及被 *L2* 正則化馴服的過擬合（藍色為訓練點，橘色為驗證點）。右：次數增加時，訓練誤差持續下降，驗證誤差則先降後升；驗證曲線的最低點就是該選的模型。

評估模型

白話版。單一個數字永遠說不完整個故事。選一個符合「犯錯代價」的指標，而且一定要和簡單的基準比較。

迴歸指標

迴歸最常用的指標是 MSE，$\frac{1}{m}\sum_i(\hat{y}^{(i)}-y^{(i)})^2$，這裡不再有只是為了訓練方便的 $\frac{1}{2}$。它的平方根 **RMSE** 和 y 的單位相同，比較好解讀（「平均大約差 3 分」）。**平均絕對誤差**（mean absolute error），$\frac{1}{m}\sum_i|\hat{y}^{(i)}-y^{(i)}|$，對少數大誤差比較不敏感 [5]。

分類指標

分類中，**準確率**（accuracy）是預測正確的比例。它容易理解，也容易騙人。在 digits 資料中，大約 10% 的影像是 9。一個永遠回答「不是 9」的「模型」有 90% 的準確率，卻完全沒用。

混淆矩陣（confusion matrix）計算二元問題的四種結果，「陽性」代表我們在意的那一類：

	預測為陽性	預測為陰性
實際為陽性	真陽性（TP）	偽陰性（FN）
實際為陰性	偽陽性（FP）	真陰性（TN）

由此得到

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}, \qquad \text{precision} = \frac{TP}{TP + FP}, \qquad \text{recall} = \frac{TP}{TP + FN}.$$

精確率（precision）回答「模型說是陽性時，有多常是對的？」**召回率**（recall）回答「所有真正的陽性中，模型找到了多少？」兩者互相拉扯：把邏輯迴歸的 0.5 決策門檻調低，會找到更多陽性（召回率較高），但也會發出更多假警報（精確率較低）。哪個比較重要取決於每種錯誤的代價：癌症篩檢不該漏掉腫瘤（高召回率），垃圾郵件過濾器則不該把真正的郵件藏起來（高精確率）。**F1 分數**是精確率與召回率的調和平均，把兩者合成一個數字 [5]。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score

X, y = load_digits(return_X_y=True)
y = (y == 9).astype(int)                  # "is it a 9?": only about 10% positives
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)

lazy = np.zeros_like(y_te)                # always answers "not a 9"
model = LogisticRegression(max_iter=5000).fit(X_tr, y_tr)
pred = model.predict(X_te)

print("lazy accuracy :", round(accuracy_score(y_te, lazy), 3), " recall:", recall_score(y_te,
 ↪  lazy))
print("model accuracy:", round(accuracy_score(y_te, pred), 3),
      " precision:", round(precision_score(y_te, pred), 3),
      " recall:", round(recall_score(y_te, pred), 3))
print(confusion_matrix(y_te, pred))       # rows: true 0/1, columns: predicted 0/1
# lazy accuracy : 0.9   recall: 0.0
# model accuracy: 0.974  precision: 0.87  recall: 0.87
```

偷懶的基準有 90% 的準確率，召回率卻是零。真正的模型準確率 97.4%，聽起來只好一點點，但 0.87 的召回率顯示它確實找到了大部分的 9。一定要回報基準；類別不平衡時，一定要看準確率以外的指標。同樣的指標會在 DIP 的圖樣分類一章再次出現，並更仔細地處理門檻與多類別的問題。

現代觀點

本篇的機制（帶參數的模型、成本、以梯度為基礎的最佳化，以及保留資料的評估）幾十年來沒有改變，下面談到的每個系統也都還在使用它們。改變的是**特徵從哪裡來**、**標籤從哪裡來**，以及**模型有多大**。以下幾篇回顧文章把這些轉變描繪得很好。

機器學習這個領域。 Jordan 與 Mitchell 發表在《Science》的回顧 [7] 以一個問題來界定機器學習：如何打造能透過經驗自動進步的電腦；並把它定位在電腦科學與統計學的交會處、人工智慧與資料科學的核心。他們把機器學習在科學、醫療、製造、教育與金融等領域的快速擴散，歸因於三件事同時到來：新的學習演算法與理論、可取得資料量的爆炸性成長，以及便宜的運算。這個框架是對抗炒作的好解藥：方法的好壞，取決於它得到的資料與經驗。

從特徵工程到表示學習。很長一段時間，傳統模型的成敗主要取決於人工設計的特徵，也就是由專家來做圖 1 中「數有墨跡的欄數」這一步：邊緣直方圖、紋理描述子、關鍵點描述子（見 IP 12）。Bengio、Courville 與 Vincent [21] 認為這種依賴是一個弱點，因為正確的表示決定了任務的難易，並主張直接從資料學習表示本身。他們的回顧涵蓋了做到這件事的工具（機率模型、自編碼器、流形學習與深度網路），以及讓表示變好的通用先驗，例如把資料背後各自獨立的解釋因子分離開來。用本篇的語言來說，深度網路仍然是「特徵加上邏輯迴歸」，只是特徵由前面的層產生，而這些層的權重也是用同樣的梯度下降學出來的。這就是 ML 02 的主題。

自監督式學習。學習表示需要資料，而標籤是昂貴的部分。Jing 與 Tian 關於自監督式視覺特徵學習的綜述 [22] 把不需要標籤的「前置任務」（pretext task）分成四類：以生成為基礎（例如重建影像或為影像上色）、以上下文為基礎（利用空間或時間結構，例如影像區塊的排列）、以免費語意標籤為基礎（使用自動得到的標籤，例如來自繪圖引擎或傳統方法），以及跨模態（判斷兩個訊號，例如影片與聲音，是否屬於同一事件）。它也確立了至今仍是標準的評估流程：用表示遷移到分類、偵測、分割等下游任務的表現來評判它。Gui 等人較新的綜述 [23] 則把這個領域重新整理為：以上下文為基礎的方法、對比學習（利用負例、自蒸餾或特徵去相關）、以遮罩影像建模為主的生成式方法，以及結合對比與生成目標的混合方法，反映出遮罩自編碼器 [12] 之後轉向遮罩預測的趨勢。對實務工作者，Balestriero 等人的《A Cookbook of Self-Supervised Learning》[24] 把現代方法分成深度度量學習家族、自蒸餾家族、典型相關分析家族與遮罩影像建模，並坦白指出結果有多依賴資料擴增與訓練時使用的小型「投影器」（projector）網路等細節。

基礎模型。 Bommasani 等人的報告 [10] 為由此產生的範式命名：在大規模、廣泛的資料上訓練（通常是自監督式），再調整去做許多下游任務的模型。報告強調了兩個後果。**湧現**（emergence）：像是憑提示中的幾個例子就能遵循指示 [11] 這類能力，是隨著規模出現的，而不是被設計出來的。**同質化**（homogenization）：許多應用如今都由同樣少數幾個基礎模型調整而來，這帶來巨大的槓桿效果，但也代表基礎模型的缺陷會被建立在它之上的一切所繼承。報告也強調，我們對這些模型如何運作、何時會失敗的理解仍非常有限。

基於人類回饋的強化學習。強化學習已經從遊戲走到語言模型訓練的最後階段 [13]、[14]。Casper 等人 [25] 整理了可能出錯的地方。他們依照 RLHF 的三個步驟（收集人類回饋、擬合獎勵模型、最佳化策略），逐一列出每一步的問題：人的回饋帶有雜訊、前後不一致，而且可能被操弄；獎勵模型只是「人們真正想要的東西」的不完美替代；而針對這個替代品用力最佳化的策略，可能會鑽它的漏洞。他們區分了在 RLHF 框架內用更好的方法就能解決的問題，以及需要換一種方法才能克服的根本限制，並主張提高透明度、把 RLHF 與其他安全措施結合，而不是只依賴它。

大規模最佳化。 Bottou、Curtis 與 Nocedal [18] 解釋了為什麼成為機器學習主力的，是樸素的隨機梯度法，而不是傳統數值分析中更精巧的最佳化方法：資料充足時，便宜而帶雜訊的一步勝過昂貴而精確的一步。Adam [19] 等自適應方法則在其上加入每個參數各自的步長。本節提到的每個模型，無論多大，都是用本篇小批次迴圈的後代訓練出來的。

未解問題。儘管規模驚人，這篇入門教學中的幾個問題仍然是最困難的。**泛化**：非常大的模型幾乎完美地擬合訓練資料，卻仍能泛化，這和圖 8 中簡單的過擬合圖像不符，其原因尚未完全被理解。**評估**：當模型用了相當大比例的公開網路資料來訓練，就很難保證測試集沒有出現在訓練資料中，因此誠實評估的基礎，乾淨的訓練／測試分離，變得難以驗證。**目標**：成本函數是模型唯一會最佳化的東西，而 RLHF 顯示出要寫出一個真正反映人們需求的成本有多難。還有**資料**：模型能學到什麼，受限於它看到的資料與標籤，以及其中所有的缺口與偏差。

針對電腦視覺，傳統路線（人工特徵加上簡單分類器）在 DIP 的圖樣分類一章中有完整發展；學

習的路線則延續到 ML 02 與〈Deep Learning for Image Processing〉。

重點整理

- 模型把輸入對應到預測，訓練則調整參數以降低損失。
- 監督式學習使用目標值，自監督從資料建立目標，強化學習使用獎勵；非監督方法在沒有外加目標值時尋找結構。
- 梯度下降沿負梯度更新；梯度為零本身不足以判定最小值。
- 只用訓練資料估計縮放統計量，並依驗證結果調整學習率。
- 線性迴歸預測數值，二元邏輯迴歸把線性分數轉成機率。
- 比較訓練與驗證誤差，以驗證資料選擇正則化，測試集保留到最後回報。

練習

1. **手算導數**。對單一特徵的 MSE 成本 $J(w,b)=\frac{1}{2m}\sum_i (wx^{(i)}+b-y^{(i)})^2$，推導 $\partial J/\partial w$ 與 $\partial J/\partial b$。接著對三個點 $(1,2),(2,4),(3,6)$，從 $w=0, b=0$ 出發，以 $\alpha=0.1$ 計算一步梯度下降。

   提示

   對每個平方項用連鎖律得到 $2(wx^{(i)}+b-y^{(i)})\cdot x^{(i)}$，2 和 $\frac{1}{2}$ 抵消，就得到梯度下降一節中的公式。在 $w=b=0$ 時誤差是 $-2,-4,-6$。所以 $\partial J/\partial w=(-2\cdot 1-4\cdot 2-6\cdot 3)/3=-28/3$，$\partial J/\partial b=-12/3=-4$。一步之後 $w=0.1\cdot 28/3\approx 0.933$，$b=0.4$。

2. **找出發散的門檻**。對 $J(w)=\frac{1}{2m}\sum_i (wx^{(i)}-y^{(i)})^2$（沒有偏差），證明一步梯度下降會把到最佳值 w^* 的距離乘上 $1-\alpha s$，其中 $s=\frac{1}{m}\sum_i (x^{(i)})^2$。$\alpha$ 在什麼範圍內梯度下降會收斂？用特徵縮放一節中的 `gd` 函數檢查你的答案。

   提示

   因為 J 是曲率為 s、最小值在 w^* 的拋物線，$\partial J/\partial w=s\,(w-w^*)$。所以 $w_{\text{new}}-w^*=(1-\alpha s)(w-w^*)$。當 $|1-\alpha s|<1$，也就是 $0<\alpha<2/s$ 時收斂；$\alpha=1/s$ 時一步就落在最小值上。有多個特徵時，s 換成 $\frac{1}{m}X^{\top}X$ 的最大特徵值，這就是為什麼一個尺度巨大的特徵會逼得 α 必須非常小。

3. **用迴歸做分類**。用 `make_classification(n_features=1, n_informative=1, n_redundant=0, n_clusters_per_class=1)` 產生一維資料，再加上五個 x 非常大的第 1 類範例。分別擬合 `LinearRegression`（以 0.5 為門檻）與 `LogisticRegression`。比較加入額外點前後兩者的準確率，並解釋差異。

   提示

   額外的點很容易分類，但平方誤差仍然希望直線經過它們附近。直線的斜率下降，跨過 0.5 的位置往第 1 類移動，一些靠近邊界的普通第 1 類點因此被分錯。分類正確且離得很遠的點，對數損失已經接近 0，所以邏輯迴歸幾乎不受影響。

4. **縮放 digits**。用本篇從頭實作的邏輯迴歸，在 digits 資料上訓練「是 9 或不是 9」，一次用原始像素（0 到 16），一次用除以 16 的像素。使用相同的步數，找出各自可用的最大學習率。縮放改變的是最終準確率、速度，還是兩者都有？

   提示

   這裡所有像素的單位相同，所以效果比圖 6 溫和，但原始輸入大了 16 倍，梯度的大小和地形的曲率也隨之放大。原始像素的安全學習率大約小 $16^2 = 256$ 倍。學習率都調好時，兩者能達到相近的準確率；縮放過的版本只是比較容易調。

5. **選擇 lambda**。用多項式的例子，固定 15 次，嘗試 $\lambda \in \{10^{-6}, 10^{-4}, 10^{-2}, 1, 100\}$。以對數橫軸畫出訓練與驗證 MSE 對 λ 的變化。哪一端欠擬合、哪一端過擬合？你會選哪個 λ？為什麼不能用測試集來做這個選擇？

   提示

   很小的 λ 等於沒有正則化（過擬合：訓練誤差低、驗證誤差高）。很大的 λ 把曲線壓成接近常數（欠擬合：兩種誤差都高）。選驗證誤差最低的 λ。如果測試集參與了選擇，它就不再是新資料上效能的不偏估計。

6. **哪一種範式？**把每個問題歸類為監督式（迴歸或分類）、非監督式、自監督式或強化學習，並說出目標值或獎勵：(a) 用附有實際結果的使用紀錄，估計手機剩餘的電池時間；(b) 把一百萬張沒有標籤的商品照片分組以便瀏覽；(c) 藉由預測每張照片缺少的四分之一來預訓練影像模型；(d) 教機械手臂堆積木，只有在積木堆穩時才給分數；(e) 在沒有任何故障標註的情況下，標出工廠感測器資料流中的異常讀數。

   提示

   (a) 監督式迴歸；目標值是剩餘時數。(b) 非監督式分群。(c) 自監督式；目標值是被藏起來的像素，由資料本身產生。(d) 強化學習；獎勵是積木是否堆穩，要到最後才給。(e) 非監督式異常偵測。

接下來的機器學習／深度學習路線

若數學符號還不熟悉，先讀 線性代數、機率與資訊理論 和 數值計算，再進入 類神經網路。

後續從 影像濾波與 CNN 連接循環／遞迴網路、Transformer 和表示學習。接著由因子模型與自編碼器進入 VAE、GAN、正規化流、擴散／ flow matching、自迴歸與可重用的基礎模型，最後用 評估與部署 完成一個可靠實驗。統計決策與推論的觀點見 圖形識別，具體視覺任務則見 視覺深度學習。

第一次讀 ML 論文的路線

先讀章節正文。這 21 篇核心導讀是選讀檢查點，依問題而非發表年份分組，不增加課程先修要求。每篇都有六個可摺疊部分、原論文圖、閱讀問題與下一步。一次走一個階段；跨主題前，先確認所列章節先備知識。

1・建立問題與數學觀點

先備知識：ML 01、ML 01A、ML 01B：資料切分、向量、共變異數與機率。

Practical ML → PCA → Shannon

檢查點：哪些假設連起表示、目標與評估？

2・理解訓練與架構

先備知識：ML 01C → ML 02 → ML 03 → ML 04 → ML 05：數值範圍、梯度、卷積、遞迴與注意力。

Mixed Precision → Adam → ResNet → LSTM → ViT

檢查點：每個設計針對什麼失敗？哪個比較能隔離它的效果？

3・檢查表示保留什麼

先備知識：ML 06 與 ML 06A；CLIP 前先讀 ML 10。理解對比學習、高斯潛在模型與遷移評估協定。

Probabilistic PCA → SimCLR → CLIP

檢查點：學習目標從哪裡來？下游評估實際測試什麼？

4・比較生成目標與取樣

先備知識：ML 07–ML 09：潛在變數、KL 散度、雜訊排程、微分方程與 token 先驗。

VAE → GAN → VQ-VAE → DDPM → Flow Matching

檢查點：區分重建、密度或界限、樣本品質與生成成本；各由什麼證據量測？

4A・沿著 Flow Matching 繼續深入

先備知識： ML 08、瞬時與平均速度、全導數、引導及像素／潛在表示。先沿區間學習閱讀，再接到直接影像輸出。

MeanFlow → Improved MeanFlow → JiT → Pixel MeanFlow

檢查點：哪個方法改變時間區間、迴歸輸入、直接輸出或生成空間？為何單靠 JiT 不是一步模型？

5・回到可靠評估

先備知識：ML 11：logits、驗證與測試分離，以及可靠度圖。

Calibration

檢查點：信心能否改善卻不改準確度？資料偏移後又會如何？

參考文獻

1. Shyandram, “Introduction to Machine Learning,” Medium post, 2024; updated 2026. link
2. A. L. Samuel, “Some studies in machine learning using the game of checkers,” *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959. doi
3. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016. book site
4. A. Ng, E. Shyu, A. Bagul and G. Ladwig, “Machine Learning Specialization,” DeepLearning.AI and Stanford Online, Coursera. course page
5. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. link
6. E. Alpaydin and C. Kaynak, “Optical Recognition of Handwritten Digits,” UCI Machine Learning Repository, 1998. doi
7. M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, 2015. doi
8. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018. book page
9. Y. LeCun and I. Misra, “Self-supervised learning: The dark matter of intelligence,” Meta AI blog, 2021. link
10. R. Bommasani, D. A. Hudson, E. Adeli et al., “On the opportunities and risks of foundation models,” arXiv:2108.07258, 2021. arXiv
11. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan et al., “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 33, 2020. link
12. K. He, X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, “Masked autoencoders are scalable vision learners,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), pp. 15979–15988, 2022. arXiv
13. L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright et al., “Training language models to follow instructions with human feedback,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 35, 2022. link
14. DeepSeek-AI, “DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning,” *Nature*, vol. 645, pp. 633–638, 2025. arXiv
15. A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010. doi
16. I. T. Jolliffe and J. Cadima, “Principal component analysis: A review and recent developments,” *Philosophical Transactions of the Royal Society A*, vol. 374, no. 2065, 20150202, 2016. doi
17. V. Chandola, A. Banerjee and V. Kumar, “Anomaly detection: A survey,” *ACM Computing Surveys*, vol. 41, no. 3, 2009. doi
18. L. Bottou, F. E. Curtis and J. Nocedal, “Optimization methods for large-scale machine learning,” *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018. doi
19. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2015. arXiv
20. A. E. Hoerl and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal prob-

lems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. doi
21. Y. Bengio, A. Courville and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. doi
22. L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 4037–4058, 2021. doi
23. J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo and D. Tao, "A survey on self-supervised learning: Algorithms, applications, and future trends," arXiv:2301.05712, 2023. arXiv
24. R. Balestriero, M. Ibrahim, V. Sobal, A. Morcos, S. Shekhar et al., "A cookbook of self-supervised learning," arXiv:2304.12210, 2023. arXiv
25. S. Casper, X. Davies, C. Shi et al., "Open problems and fundamental limitations of reinforcement learning from human feedback," *Transactions on Machine Learning Research*, 2023. TMLR ·arXiv
26. P. Domingos, "A Few Useful Things to Know About Machine Learning," *Communications of the ACM*, 55(10), pp. 78–87, 2012. doi
27. NumPy developers, "numpy.polyfit," NumPy 文件，查閱日期 2026-11-05。最小平方求解與奇異值

ML 01A・學習所需的線性代數

向量、矩陣與張量、範數、特徵分解、奇異值分解、偽反矩陣、跡與行列式，並一步步推導主成分分析，附經過測試的 NumPy 程式碼。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/linear-algebra/

你將學到

- 追蹤陣列形狀，理解矩陣乘法。
- 用秩、零空間與範數理解線性系統。
- 從特徵值與奇異值理解幾何性質。
- 解最小平方問題，診斷對雜訊的敏感度。
- 推導 PCA，並連結矩陣恆等式與學習運算。

先看全貌

神經網路裡幾乎每個運算，都是「線性映射」接上一個簡單的非線性函數。全連接層是乘上權重矩陣；卷積是有結構的矩陣乘法；注意力機制先計算向量之間的內積，再用一個權重矩陣把向量混合起來。線性代數就是描述這一切的語言。它告訴我們一個映射保留了什麼、丟掉了什麼、拉長了什麼，以及怎樣計算才不會損失精確度。

> **用最簡單的話說**
>
> 向量是一串測量值，例如一個人的身高、體重和年齡。矩陣像一台機器：把一串數字放進去，它會混合出一串新的數字。有的機器只是把這串數字「轉個方向」（旋轉），有的把某些部分放大、某些部分縮小（拉伸），有的則把某些部分壓扁，資訊從此就找不回來。線性代數就是這些機器的說明書：它告訴你每台機器朝哪個方向拉、拉多少，以及能不能把動作復原。

本篇對應 Goodfellow、Bengio 與 Courville 所著《Deep Learning》的「應用數學」部分 [1]，以及《Mathematics for Machine Learning》的線性代數章節 [2]。想多做練習，Strang 的教科書 [3] 是最平易近人的完整課程；Golub 與 Van Loan [4]、Trefethen 與 Bau [5] 則是「如何在電腦上可靠地完成這些計算」的標準參考書。如果你還不熟悉矩陣符號，建議在 ML 02 之前先讀本篇。姊妹篇是 ML 01B・機率與資訊理論 與 ML 01C・數值計算。

本篇有一條主線。我們從物件與形狀開始，接著談乘法與解方程式，再用範數量測大小。特徵分解與 SVD 說明一個矩陣的「個性」；偽反矩陣、跡與行列式則是建立在它們之上的工具。最後，我們用上述每一項工具推導 PCA——這個經典的資料壓縮方法——推導方式與 [1] 相同。

純量、向量、矩陣與張量

白話版。機器學習裡的每個數字都住在一個有命名軸的陣列裡。做任何乘法之前，先說清楚每個軸代表什麼。

純量（scalar）是單一數字，寫成 $s \in \mathbb{R}$。**向量**（vector）是 n 個數字的有序清單：$x \in \mathbb{R}^n$，分量為 $x_1, \ldots, x_n$；依慣例，向量是一個行向量（直的）。**矩陣**（matrix）是二維的數字格子：$A \in \mathbb{R}^{m \times n}$ 有 m 列（row）與 n 行（column），A_{ij} 是第 i 列、第 j 行的元素。**張量**（tensor）則是任意多個軸的陣列。

這些物件可以直接對應到資料：

- 高 H、寬 W 的灰階影像是 $\mathbb{R}^{H \times W}$ 的矩陣；攤平後就是向量 $x \in \mathbb{R}^{HW}$（見影像即陣列）。
- n 筆樣本、每筆 d 個特徵，每列放一筆樣本，就得到**設計矩陣**（design matrix）$X \in \mathbb{R}^{n \times d}$。
- 權重 $W \in \mathbb{R}^{d \times m}$、偏差 $b \in \mathbb{R}^m$ 的全連接層一次處理整個批次：$Y = XW + \mathbf{1}b^\top \in \mathbb{R}^{n \times m}$，其中 $\mathbf{1} \in \mathbb{R}^n$ 是全為 1 的向量。
- PyTorch 的影像批次通常是形狀 $B \times C \times H \times W$（批次、通道、高、寬）的四維張量；Transformer 的 token 嵌入批次通常是 $B \times N \times D$（批次、序列長度、嵌入維度）。

轉置（transpose）$A^\top$ 把列與行對調：$(A^\top)_{ij} = A_{ji}$。它滿足 $(A^\top)^\top = A$ 與 $(AB)^\top = B^\top A^\top$——注意順序反過來了。

廣播

寫 $\mathbf{1}b^\top$ 在數學上很誠實，但很繁瑣，所以程式庫允許你直接把向量加到矩陣的每一列。這叫**廣播**（broadcasting）：兩個陣列形狀不同時，長度為 1 的軸（或缺少的前導軸）會被延展成相同大小。廣播很方便，也很危險，因為形狀錯了程式往往照樣執行。最常見的錯誤是把形狀 $(n, 1)$ 的行向量和形狀 $(n,)$ 的一維陣列相減，結果悄悄變成 $n \times n$ 的矩陣，而不是 n 個殘差。

```
import numpy as np

rng = np.random.default_rng(0)
X = rng.normal(size=(32, 64))      # batch of 32 examples, 64 features
W = rng.normal(size=(64, 10))      # linear map 64 -> 10
b = rng.normal(size=10)            # one bias per output
Y = X @ W + b                      # b broadcasts over the 32 rows
assert Y.shape == (32, 10)

# A classic broadcasting bug: (32,) - (32, 1) silently becomes (32, 32)
y_true = rng.normal(size=32)
y_pred = rng.normal(size=(32, 1))
print("buggy residual shape:", (y_true - y_pred).shape)
print("fixed residual shape:", (y_true - y_pred[:, 0]).shape)

# Four equivalent views of the same matrix product
A, B = rng.normal(size=(3, 4)), rng.normal(size=(4, 2))
entry = np.array([[A[i] @ B[:, j] for j in range(2)] for i in range(3)])
cols = np.stack([A @ B[:, j] for j in range(2)], axis=1)
outer = sum(np.outer(A[:, k], B[k]) for k in range(4))
assert np.allclose(A @ B, entry) and np.allclose(A @ B, cols) and np.allclose(A @ B, outer)
print("all views agree")
```

錯誤的那一行印出 `(32, 32)`，修正後是 `(32,)`。對 32×32 陣列取平均仍然只得到一個數字，所

以用這種方式算出的損失函數可以順利訓練，卻在最佳化錯誤的目標。凡是重要的程式碼，都替形狀加上 `assert`。

矩陣與向量的乘法

白話版。矩陣乘法把兩台線性機器串成一台。輸出的每個數字，是第一個矩陣的某一列與第二個矩陣的某一行的內積。

對 $A \in \mathbb{R}^{m\times n}$ 與 $B \in \mathbb{R}^{n\times p}$，**矩陣乘積** $C = AB \in \mathbb{R}^{m\times p}$ 為

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}.$$

其中 i 是 A（也是 C）的列索引，j 是 B（也是 C）的行索引，k 走遍共同的內部維度 n。內部維度必須相同——遇到形狀錯誤時，這是第一個要檢查的地方。

同一個乘積有四種讀法，各有用處：

1. **元素觀點**。C_{ij} 是 A 的第 i 列與 B 的第 j 行的內積。注意力分數 $QK^\top$ 就是這樣讀：每個 query 和每個 key 比較。
2. **行觀點**。C 的第 j 行等於 A 乘上 B 的第 j 行；C 的每一行都是 A 各行的線性組合。
3. **列觀點**。C 的第 i 列等於 A 的第 i 列乘上 B。在 $Y = XW$ 中，每筆輸出樣本就是它的輸入列通過同一組權重。
4. **外積觀點**。$C = \sum_k a_{:k} b_{k:}$，是 n 個秩一矩陣的總和，每個由 A 的第 k 行與 B 的第 k 列構成。這個觀點會在本篇後面解釋低秩近似與 LoRA。

上面的程式碼驗證了四種讀法結果相同。

矩陣乘法滿足**結合律** $(AB)C = A(BC)$ 與**分配律** $A(B + C) = AB + AC$，但**不滿足交換律**：一般而言 $AB \neq BA$，而且其中一個乘積常常根本沒有定義。**逐元素**（Hadamard）乘積 $A \odot B$ 則是把對應元素相乘，需要形狀相同。在 NumPy 與 PyTorch 中，`A @ B` 是矩陣乘法，`A * B` 是逐元素乘法。兩個向量 $x, y \in \mathbb{R}^n$ 的**內積**是特例 $x^\top y = \sum_i x_i y_i$，而且對稱：$x^\top y = y^\top x$。

單位矩陣、反矩陣與線性方程組

m 條方程式、n 個未知數的線性方程組可以簡寫成

$$Ax = b, \qquad A \in \mathbb{R}^{m\times n},\ x \in \mathbb{R}^n,\ b \in \mathbb{R}^m.$$

A 的每一列是一條方程式的係數，x 是未知數，b 是等號右邊的值。

單位矩陣 I_n 對角線為 1、其餘為 0，不改變任何向量：$I_n x = x$。若方陣 A 存在矩陣 A^{-1} 使 $A^{-1}A = AA^{-1} = I$，就稱 A **可逆**，此時方程組恰有一解 $x = A^{-1}b$。

這個公式很適合推導，卻不適合計算。直接算出 A^{-1}，比用 LU、Cholesky 或 QR 分解直接求解更費時、也更不精確 [4], [5]。在 NumPy 中請寫 `np.linalg.solve(A, b)`，而不是 `np.linalg.inv(A) @ b`。本篇中的 A^{-1} 是思考工具。

線性相依、生成空間與秩

白話版。秩計算一個矩陣真正能產生多少個不同方向。重複或多餘的行不會增加新東西。

向量 $v^{(1)}, \dots, v^{(k)}$ 的**線性組合**是 $\sum_i c_i v^{(i)}$，c_i 為純量。一組向量的**生成空間**（span）是它們所有線性組合構成的集合。由於 $Ax = \sum_j x_j a_{:j}$，向量 Ax 是 A 各行 $a_{:j}$ 的線性組合。因此 $Ax = b$ 有解，若且唯若 b 落在各行的生成空間裡，這個空間稱為 A 的**行空間**（column space，或值域）。

若一組向量中沒有任何一個是其他向量的線性組合——等價地說，$\sum_i c_i v^{(i)} = 0$ 只在所有 $c_i = 0$ 時成立——就稱它們**線性獨立**。A 的**秩**（rank）是線性獨立的行數，而它永遠等於線性獨立的列數。對 $A \in \mathbb{R}^{m \times n}$，$\operatorname{rank}(A) \le \min(m, n)$；等號成立時稱 A **滿秩**。

A 的**零空間**（null space）是所有滿足 $Az = 0$ 的向量 z，也就是被 A 摧毀的輸入方向。**秩–零度定理**說

$$\operatorname{rank}(A) + \dim \operatorname{null}(A) = n,$$

其中 n 是行數，dim 是零空間中獨立方向的個數。這一條等式就解釋了 $Ax = b$ 的三種情況：

形狀與秩	會發生什麼	典型的機器學習情境
方陣、滿秩	恰有一解	良好設定的座標轉換
高瘦（$m > n$）、行滿秩	通常沒有精確解；改求最小平方解	樣本數多於特徵數的迴歸
矮胖（$m < n$）或秩不足	若有解則有無限多解	參數多於限制條件，例如深度網路

對方陣來說，「可逆」「滿秩」「各行線性獨立」「零空間只有零向量」是同一件事；不滿足的方陣稱為**奇異**（singular）矩陣。

範數：量測大小

白話版。範數把一個向量變成一個非負數，表示它有多大。不同的範數對「大」有不同看法。

範數是滿足以下條件的函數 f：$f(x) = 0 \Rightarrow x = 0$；$f(x + y) \le f(x) + f(y)$（三角不等式）；對任何純量 α，$f(\alpha x) = |\alpha| f(x)$。最常見的是 L^p 範數：

$$\|x\|_p = \Big(\sum_i |x_i|^p \Big)^{1/p}, \qquad p \ge 1.$$

- $\|x\|_2$ 是**歐幾里得**範數，也就是一般的長度，常簡寫為 $\|x\|$。它的平方 $\|x\|_2^2 = x^\top x$ 更好微分，出現在均方誤差損失與權重衰減中。
- $\|x\|_1 = \sum_i |x_i|$ 在每個地方——包括靠近零的地方——都以相同速率成長。當「恰好為零」和「稍微不為零」的差別很重要時它特別有用，這也是 L^1 懲罰會鼓勵稀疏解的原因。
- $\|x\|_\infty = \max_i |x_i|$ 是**最大範數**，即最大元素的大小。對抗強健性的研究常用它限制擾動。
- 非零元素的個數有時被稱為「L^0 範數」，但它不是範數：把 x 乘以 2 並不會改變它。

內積透過夾角與歐幾里得範數相連：$x^\top y = \|x\|_2 \|y\|_2 \cos\theta$，$\theta$ 是 x 與 y 的夾角。**餘弦相似度**

$$\cos(x, y) = \frac{x^\top y}{\|x\|_2 \|y\|_2}$$

只對非零向量有定義，並且忽略尺度。這對某些嵌入很有用（對比學習正因如此會把特徵正規化），但當大小本身有意義時——例如影像亮度——就不適用。

矩陣範數

矩陣最常用兩種範數。**Frobenius 範數**把矩陣當成一條長向量：

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}.$$

譜範數（spectral norm，即算子 2-範數）量測 A 能把單位向量拉長的最大倍數：

$$\|A\|_2 = \max_{\|x\|_2 = 1} \|Ax\|_2 = \sigma_{\max}(A),$$

其中 $\sigma_{\max}(A)$ 是最大奇異值，定義見下方 SVD 一節。若以全部奇異值 $\sigma_1 \geq \sigma_2 \geq \dots$ 表示，則 $\|A\|_F^2 = \sum_i \sigma_i^2$，而 $\|A\|_2 = \sigma_1$。所以許多小奇異值可以讓 Frobenius 範數變大，卻不改變最大放大倍數。譜範數限制了一層網路最多能把輸入擾動放大多少，因此出現在強健性與 GAN 穩定性的研究中 [17]。

```
import numpy as np

x = np.array([3.0, -4.0, 0.0, 1.0])
for p in [1, 2, np.inf]:
    print(f"L{p} norm:", np.linalg.norm(x, p))
print("L0 'norm' (count of nonzeros):", np.count_nonzero(x))

A = np.array([[3.0, 0.0], [4.0, 5.0]])
s = np.linalg.svd(A, compute_uv=False)
print("singular values:", s.round(4))
print("Frobenius:", np.linalg.norm(A, "fro"), "=", np.sqrt((s**2).sum()))
print("spectral :", np.linalg.norm(A, 2), "=", s[0])
assert np.isclose(np.linalg.norm(A, "fro")**2, np.trace(A.T @ A))
```

對 $x = (3, -4, 0, 1)$，程式印出 $L^1 = 8$、$L^2 = \sqrt{26} \approx 5.10$、$L^\infty = 4$，以及三個非零元素。這個 2×2 矩陣的奇異值是 $3\sqrt{5} \approx 6.71$ 與 $\sqrt{5} \approx 2.24$，因此 $\|A\|_F = \sqrt{50} \approx 7.07$、$\|A\|_2 \approx 6.71$。

特殊的矩陣與向量

白話版。有些矩陣比其他矩陣好處理得多。學會辨認它們，可以省下計算量，也能理解許多設計選擇。

- **對角矩陣** $D = \operatorname{diag}(v)$ 只有主對角線上有非零元素。乘上它等於把每個座標分別縮放：$\operatorname{diag}(v)x = v \odot x$。當每個 $v_i \neq 0$ 時，它的反矩陣是 $\operatorname{diag}(1/v_1, \dots, 1/v_n)$。正規化層中逐通道的縮放，其實就是對角矩陣。長方形的對角矩陣也存在，會出現在 SVD 中。
- **對稱矩陣**滿足 $A = A^\top$。共變異數矩陣、Gram 矩陣 $X^\top X$、平滑函數的 Hessian 與距離矩陣都是對稱的。

- **單位向量**滿足 $\|x\|_2 = 1$。若 $x^\top y = 0$，兩向量**正交**；若它們同時也是單位向量，就是**單範正交**（orthonormal）。
- **正交矩陣** Q 是各列、各行皆單範正交的方陣，因此 $Q^\top Q = QQ^\top = I$ 且 $Q^{-1} = Q^\top$——反矩陣不花成本。它保持長度與角度不變，$\|Qx\|_2 = \|x\|_2$，幾何上是旋轉、鏡射或兩者的組合。

若 $Q \in \mathbb{R}^{n\times k}$ 有 $k < n$ 個單範正交的行，則 $Q^\top Q = I_k$，但 $QQ^\top \neq I_n$。矩陣 $P = QQ^\top$ 是投影到 Q 各行生成空間的**正交投影**。投影 Px 是該子空間中離 x 最近的點，殘差 $x - Px$ 與 Q 的每一行都垂直。投影兩次不會再改變任何東西：$P^2 = P$。下面的 PCA 正是在尋找最好的這種 Q。

特徵分解

白話版。實特徵向量變換後仍在同一直線上；特徵值決定縮放，也可能反轉方向。

方陣 $A \in \mathbb{R}^{n\times n}$ 的**特徵向量**是滿足

$$Av = \lambda v$$

的非零向量 v，純量 λ 是對應的**特徵值**。若 v 是特徵向量，任何 $c \neq 0$ 的 cv 也是，所以通常把特徵向量正規化成單位長度。

假設 A 有 n 個線性獨立的特徵向量 $v^{(1)}, \dots, v^{(n)}$，特徵值為 $\lambda_1, \dots, \lambda_n$。把特徵向量當成 V 的各行，把特徵值放進 $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$，就有 $AV = V\Lambda$，因此

$$A = V\Lambda V^{-1}.$$

這就是**特徵分解**。它的意思是：先換到特徵向量座標（V^{-1}），每個座標各自縮放（Λ），再換回來（V）。不是每個矩陣都有特徵分解。剪切矩陣 $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$ 只有一個獨立的特徵向量；旋轉 90° 的矩陣特徵值是 $\pm i$，是複數，因為沒有任何實數方向能保持不轉。

對稱的情形

對稱矩陣是好相處的情形，也是機器學習最常遇到的。**譜定理**指出，每個實對稱矩陣都可以寫成

$$A = Q\Lambda Q^\top,$$

其中 Q 是正交矩陣（各行是單範正交的特徵向量），Λ 是實數對角矩陣 [3], [4]。因為 $Q^{-1} = Q^\top$，完全不需要求反矩陣。幾何上，A 沿著方向 $q^{(i)}$（Q 的第 i 行）把空間拉伸 λ_i 倍。除了特徵值的排列順序，以及重複特徵值內部基底的選擇之外，這個分解是唯一的；慣例上把特徵值由大到小排列。

對對稱的 A，把二**次型** $f(x) = x^\top Ax$ 限制在單位向量上，其值被最大與最小特徵值夾住：

$$\lambda_{\min} \le x^\top Ax \le \lambda_{\max} \qquad \text{當} \|x\|_2 = 1,$$

最大值出現在最大特徵值的特徵向量，最小值出現在最小特徵值的特徵向量。比值 $x^\top Ax / x^\top x$ 稱為 **Rayleigh 商**。這個事實是後面 PCA 推導的引擎。

正定矩陣

特徵值的正負號可以為對稱矩陣分類：

名稱	特徵值	等價條件
正定（PD）	全部 $\lambda_i > 0$	對所有 $x \neq 0$，$x^\top A x > 0$
半正定（PSD）	全部 $\lambda_i \geq 0$	對所有 x，$x^\top A x \geq 0$
負定（半負定）	全部 $\lambda_i < 0$（或 ≤ 0）	不等式反向
不定	正負都有	$x^\top A x$ 正負皆可能

每個共變異數矩陣與每個 Gram 矩陣 $X^\top X$ 都是半正定的，因為 $v^\top X^\top X v = \|Xv\|_2^2 \geq 0$。高斯密度需要正定的共變異數矩陣才能求反。損失函數的 Hessian 在嚴格局部最小值處是正定的，在鞍點處則是不定的；ML 01C 會用這一點解釋最佳化為什麼可能很慢。正定矩陣有 **Cholesky 分解** $A = LL^\top$（L 為下三角矩陣），這是用它解方程組、以及從高斯分佈抽樣最快又穩定的方式 [4]。

```
import numpy as np

rng = np.random.default_rng(1)
M = rng.normal(size=(4, 4))
S = M @ M.T + 0.1 * np.eye(4)        # symmetric positive definite
lam, Q = np.linalg.eigh(S)           # ascending eigenvalues, orthonormal Q
assert np.allclose(Q @ np.diag(lam) @ Q.T, S)
assert np.allclose(Q.T @ Q, np.eye(4))
print("eigenvalues:", lam.round(3), "all positive:", bool((lam > 0).all()))

# The quadratic form x^T S x is bounded by the extreme eigenvalues
x = rng.normal(size=4); x /= np.linalg.norm(x)
print("Rayleigh quotient in range:", lam[0] <= x @ S @ x <= lam[-1])

# A non-symmetric matrix can have complex eigenvalues
R = np.array([[0.0, -1.0], [1.0, 0.0]])  # 90-degree rotation
print("rotation eigenvalues:", np.linalg.eigvals(R))
```

對稱矩陣請用 `np.linalg.eigh`：它比較快，回傳由小到大排列的實數特徵值與單範正交的特徵向量。`np.linalg.eig` 處理一般情形，可能回傳複數，如旋轉矩陣所示。

奇異值分解

白話版。SVD 把實矩陣分解成正交座標變換與逐軸縮放；座標變換也可能包含鏡射。

特徵分解需要方陣，而且不一定存在。**奇異值分解**（singular value decomposition, SVD）則對每個實矩陣 $A \in \mathbb{R}^{m \times n}$ 都存在：

$$A = U \Sigma V^\top.$$

- $U \in \mathbb{R}^{m \times m}$ 是正交矩陣，各行是**左奇異向量**，屬於輸出空間的方向。
- $\Sigma \in \mathbb{R}^{m \times n}$ 是長方形對角矩陣，對角元素 $\sigma_1 \geq \sigma_2 \geq \ldots \geq 0$ 是**奇異值**。

- $V \in \mathbb{R}^{n\times n}$ 是正交矩陣，各行是**右奇異向量**，屬於輸入空間的方向。

由右往左讀，$Ax = U\Sigma V^\top x$ 先把 x 表示成 V 座標（旋轉或鏡射），再把第 i 個座標乘上 σ_i，最後沿著 U 的方向擺放。圖 1 以 2×2 矩陣示範：單位圓變成一個橢圓，半軸長分別是 σ_1 與 σ_2。

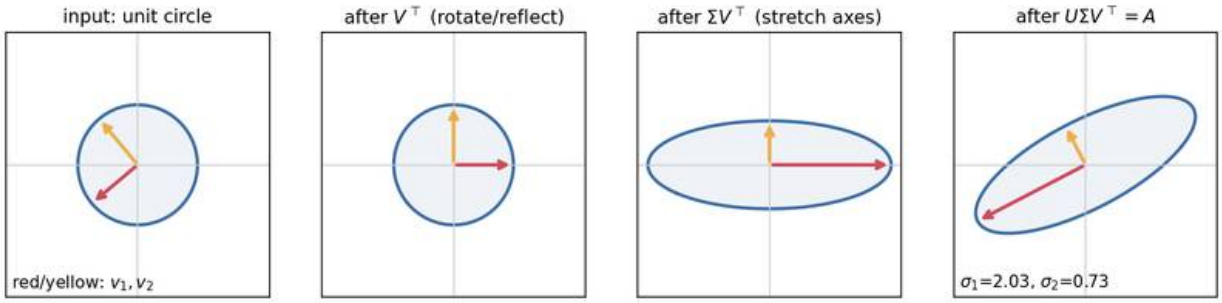


圖 *1 — SVD* 就是「旋轉、拉伸、旋轉」。右奇異向量 *v1*、*v2*（紅、黃）先被 V 的轉置轉到座標軸上，再依奇異值拉伸，最後由 U 旋轉。最終的橢圓就是單位圓經過 A 映射後的像。

SVD 與特徵分解關係密切。由於 $A^\top A = V\Sigma^\top\Sigma V^\top$ 且 $AA^\top = U\Sigma\Sigma^\top U^\top$，右奇異向量是 $A^\top A$ 的特徵向量，左奇異向量是 $AA^\top$ 的特徵向量，而 $\sigma_i = \sqrt{\lambda_i(A^\top A)}$。對於對稱半正定矩陣，SVD 與特徵分解一致。

非零奇異值的個數等於秩 r。只保留它們，就得到**精簡**（thin）SVD：$A = U_r\Sigma_r V_r^\top = \sum_{i=1}^{r}\sigma_i u^{(i)}v^{(i)\top}$，也就是 r 個秩一矩陣的總和——又是矩陣乘法的外積觀點。

最佳低秩近似

把這個總和截斷在第 k 項，得到 $A_k = \sum_{i=1}^{k}\sigma_i u^{(i)}v^{(i)\top}$。**Eckart–Young 定理** [6] 指出，A_k 是 A 在 Frobenius 範數下的最佳秩 k 近似：

$$\min_{\operatorname{rank}(B)\le k}\|A-B\|_F^2 = \|A-A_k\|_F^2 = \sum_{i>k}\sigma_i^2.$$

同一個 A_k 在譜範數下也是最佳的，誤差為 $\|A-A_k\|_2 = \sigma_{k+1}$ [4]。換句話說：丟掉最小的那些奇異值，損失的能量剛好就是丟掉的部分。儲存 A_k 只需要 $k(m+n+1)$ 個數字，而不是 mn 個。

圖 2 把這套方法用在一張 256×256 的灰階照片上。自然影像的奇異值衰減得很快，所以秩 50 的近似已經保留了主要結構。低秩時的條紋狀瑕疵，來自每個秩一項都是一個行樣式與一個列樣式的外積。

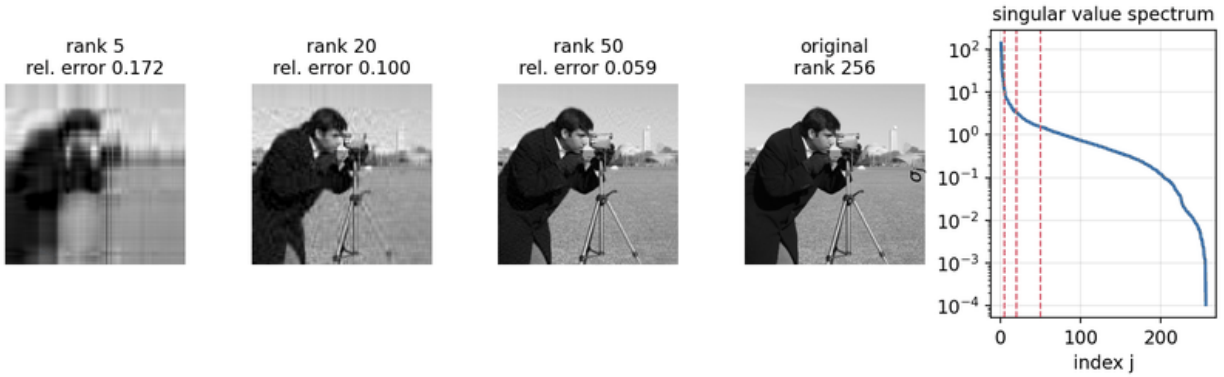


圖 2 — *scikit-image* 中 256×256「*camera*」影像的秩 k 近似，標題標示相對 *Frobenius* 誤差。右圖：對數尺度下的奇異值，虛線標出 $k = 5、20、50$。每個近似的誤差由該虛線右邊的奇異值決定。

```
import numpy as np

rng = np.random.default_rng(0)
A = rng.normal(size=(12, 8))
U, s, Vt = np.linalg.svd(A, full_matrices=False)
assert np.allclose(U @ np.diag(s) @ Vt, A)
k = 3
Ak = (U[:, :k] * s[:k]) @ Vt[:k]
err_F = np.linalg.norm(A - Ak, "fro")**2
err_2 = np.linalg.norm(A - Ak, 2)
assert np.isclose(err_F, (s[k:]**2).sum()) and np.isclose(err_2, s[k])
print("rank-k squared Frobenius error:", round(err_F, 3))
print("rank-k spectral error = sigma_{k+1}:", round(err_2, 3), round(s[k], 3))
# Eigenvalues of A^T A are squared singular values of A
assert np.allclose(np.sort(np.linalg.eigvalsh(A.T @ A))[::-1], s**2)
print("storage: full", A.size, "numbers; rank-3 factors", k * (12 + 8 + 1))
```

`full_matrices=False` 回傳精簡 SVD，這幾乎總是你要的。`(U[:, :k] * s[:k])` 用廣播縮放 U 的各行，不必建立對角矩陣。兩條 Eckart–Young 等式都在機器精確度內成立。

一個提醒：奇異值最大的方向承載最多能量，卻不一定承載最有用的資訊。對分類而言，變異數小的方向可能正是把類別分開的那個方向。PR 05 正是為此比較了 PCA 與判別式方法。

Moore–Penrose 偽反矩陣

白話版。矩陣沒有反矩陣時，偽反矩陣給出次佳的答案：盡可能貼近資料的解，而在同樣好的解之中，選最小的那一個。

只有滿秩方陣才有反矩陣。對任意 $A \in \mathbb{R}^{m\times n}$，**Moore–Penrose 偽反矩陣**（pseudo-inverse）$A^{+} \in \mathbb{R}^{n\times m}$ 是唯一滿足以下四個 Penrose 條件的矩陣 [7]：

$$AA^{+}A = A, \quad A^{+}AA^{+} = A^{+}, \quad (AA^{+})^{\top} = AA^{+}, \quad (A^{+}A)^{\top} = A^{+}A.$$

用 SVD 計算最容易。若 $A = U\Sigma V^{\top}$，則

$$A^{+} = V \Sigma^{+} U^{\top},$$

其中 $\Sigma^{+} \in \mathbb{R}^{n \times m}$ 是把 Σ 轉置、並把每個非零奇異值 σ_i 換成 $1/\sigma_i$（零維持為零）。Goodfellow 等人給出等價的極限形式 $A^{+} = \lim_{\alpha \searrow 0} (A^{\top}A + \alpha I)^{-1} A^{\top}$ [1]，這已經暗示了嶺迴歸。

在秩的表格的每一種情況下，偽反矩陣都給出正確答案：

- **高瘦、行滿秩**（$m > n$）。$Ax = b$ 通常沒有精確解。此時 $x = A^{+}b = (A^{\top}A)^{-1}A^{\top}b$ 是**最小平方**解：它使 $\|Ax - b\|_2^2$ 最小。
- **矮胖、列滿秩**（$m < n$）。解有無限多個。此時 $x = A^{+}b = A^{\top}(AA^{\top})^{-1}b$ 是 $\|x\|_2$ 最小的解，稱為**最小範數**解。其他任何解都多加了一個零空間向量，只會讓範數變大。
- **秩不足**。$A^{+}b$ 同時兼具兩者：在所有最小平方解之中，範數最小的那一個。

最小範數性質是通往一個現代問題的小窗口。對欠定的最小平方問題從零開始做梯度下降，會收斂到最小範數解，因為每個梯度 $A^{\top}(Ax - b)$ 都落在 A 的列空間中。過參數化的網路不是線性的，但思考「在眾多完美擬合中，訓練演算法會選哪一個」時，這個簡單情形通常是起點。

最小平方、正規方程式與條件數

對迴歸設計矩陣 $X \in \mathbb{R}^{n \times d}$ 與目標 $y \in \mathbb{R}^{n}$，最小平方法最小化 $\|Xw - y\|_2^2$。令梯度為零，得到**正規方程式**

$$X^{\top}X w = X^{\top} y.$$

它們是對的，但在數值上有風險。線性問題的敏感度用**條件數** $\kappa_2(A) = \sigma_{\max}/\sigma_{\min}$ 量測。粗略地說，資料中的相對誤差在解中最多可能被放大 κ 倍 [5]。形成 $X^{\top}X$ 會把條件數平方：$\kappa(X^{\top}X) = \kappa(X)^2$，於是輕微病態的問題變成嚴重病態。程式庫的最小平方求解器（`np.linalg.lstsq`、`torch.linalg.lstsq`）直接對 X 做 QR 或 SVD 分解，避免了這個平方 [4]。

當兩個特徵幾乎相同時，$\sigma_{\min}$ 非常小，即使預測沒問題，個別係數也會極度敏感。**嶺迴歸**（ridge regression）[8] 加上懲罰項 $\lambda\|w\|_2^2$，$\lambda > 0$：

$$w_{\text{ridge}} = (X^{\top}X + \lambda I)^{-1} X^{\top} y = \sum_{i} \frac{\sigma_i}{\sigma_i^2 + \lambda} \, (u^{(i)\top} y) \, v^{(i)}.$$

右邊的 SVD 形式說明了嶺迴歸在做什麼。一般最小平方使用係數 $1/\sigma_i$，在 σ_i 很小時爆炸；嶺迴歸使用 $\sigma_i/(\sigma_i^2 + \lambda)$，上限是 $1/(2\sqrt{\lambda})$。大 σ 的方向幾乎不受影響，小 σ 的方向被壓抑，代價是引入偏差。在神經網路中，同一個懲罰項稱為**權重衰減**（weight decay）。

```
import numpy as np

rng = np.random.default_rng(0)
# Over-determined: more equations than unknowns -> least squares
x = np.linspace(-1, 1, 40)
X = np.c_[np.ones_like(x), x]
y = 2 + 3*x + rng.normal(0, .05, len(x))
w, residuals, rank, sv = np.linalg.lstsq(X, y, rcond=None)
print("intercept and slope:", w.round(3), "rank:", rank)
assert np.allclose(w, np.linalg.pinv(X) @ y)
```

```
# Under-determined: fewer equations than unknowns -> minimum-norm solution
A = rng.normal(size=(2, 5)); b = rng.normal(size=2)
w_min = np.linalg.pinv(A) @ b
null = np.linalg.svd(A)[2][-1]             # a direction with A @ null = 0
w_other = w_min + 0.7 * null
print("both solve Aw=b:", np.allclose(A @ w_min, b), np.allclose(A @ w_other, b))
print("norms:", np.linalg.norm(w_min).round(3), "<", np.linalg.norm(w_other).round(3))

# Near-collinear features: condition number and ridge
z = rng.normal(size=200)
F = np.c_[z, z + 1e-4 * rng.normal(size=200)]
t = z + 0.01 * rng.normal(size=200)
print("cond(F) = %.1e, cond(F^T F) = %.1e" % (np.linalg.cond(F), np.linalg.cond(F.T @ F)))
w_ls = np.linalg.lstsq(F, t, rcond=None)[0]
lam = 1e-2
w_ridge = np.linalg.solve(F.T @ F + lam * np.eye(2), F.T @ t)
print("least squares:", w_ls.round(2), " ridge:", w_ridge.round(3))
```

直線擬合得到截距 ≈ 2.00、斜率 ≈ 3.01，而 `pinv` 與 `lstsq` 結果一致。欠定系統有兩個精確解，偽反矩陣給的那一個範數較小（0.885 對 1.128）。在共線的例子中，$\kappa(F) \approx 2 \times 10^4$，而 $\kappa(F^\top F) \approx 4 \times 10^8$，正好是它的平方。最小平方回傳誇張的係數 $(-2.63, 3.63)$，但兩者相加仍約為 1，所以預測沒有問題；嶺迴歸則回傳穩定的 $(0.5, 0.5)$。第一個例子中的截距行讓直線可以有非零偏移。在實際流程中，置中與縮放只能在訓練集上擬合，再原封不動套用到驗證與測試資料。

跡運算子

白話版。跡就是把對角線加起來。它看起來簡單到不值一提，卻能把麻煩的總和改寫成矩陣運算式，並讓你在式子裡搬動矩陣。

方陣的**跡**（trace）是

$$\operatorname{Tr}(A) = \sum_i A_{ii}.$$

它有這些好用的性質：

- **Frobenius 範數**。$\|A\|_F = \sqrt{\operatorname{Tr}(AA^\top)} = \sqrt{\operatorname{Tr}(A^\top A)}$。
- **轉置**。$\operatorname{Tr}(A) = \operatorname{Tr}(A^\top)$。
- **循環性質**。只要乘積有定義，$\operatorname{Tr}(ABC) = \operatorname{Tr}(CAB) = \operatorname{Tr}(BCA)$，即使矩陣不是方陣、中間乘積形狀不同也成立。特別地，對 $A \in \mathbb{R}^{m\times n}$、$B \in \mathbb{R}^{n\times m}$，$\operatorname{Tr}(AB) = \operatorname{Tr}(BA)$。可以循環，但不能任意重排：一般而言 $\operatorname{Tr}(ABC) \neq \operatorname{Tr}(ACB)$。
- **純量**。純量等於自己的跡，$a = \operatorname{Tr}(a)$。這能把二次型變成跡：$x^\top Ax = \operatorname{Tr}(x^\top Ax) = \operatorname{Tr}(Axx^\top)$。
- **特徵值**。$\operatorname{Tr}(A) = \sum_i \lambda_i$。對共變異數矩陣而言，這就是總變異數，也就是沿每個主方向的變異數總和。

許多期望值就是用這個純量技巧算出來的。若隨機向量 x 的平均為零、共變異數為 Σ，則 $\mathbb{E}[x^\top Ax] = \mathbb{E}[\operatorname{Tr}(Axx^\top)] = \operatorname{Tr}(A\Sigma)$，因為跡與期望值都是線性的。

行列式

白話版。行列式絕對值表示體積縮放，正負表示定向；行列式為零代表方陣壓縮了維度，因而不可逆。

方陣的**行列式** $\det(A)$ 是一個數字，具有以下性質：

- $\det(A) = \prod_i \lambda_i$，即特徵值的乘積。
- $|\det(A)|$ 是 A 縮放體積的倍數：單位立方體被映射成體積為 $|\det(A)|$ 的平行多面體。
- 正負號記錄方向（定向）。行列式為負表示 A 包含鏡射。
- $\det(A) = 0$ 恰好表示 A 是奇異的：某個方向被壓成零，資訊遺失。
- $\det(AB) = \det(A)\det(B)$、$\det(A^\top) = \det(A)$、$\det(A^{-1}) = 1/\det(A)$。

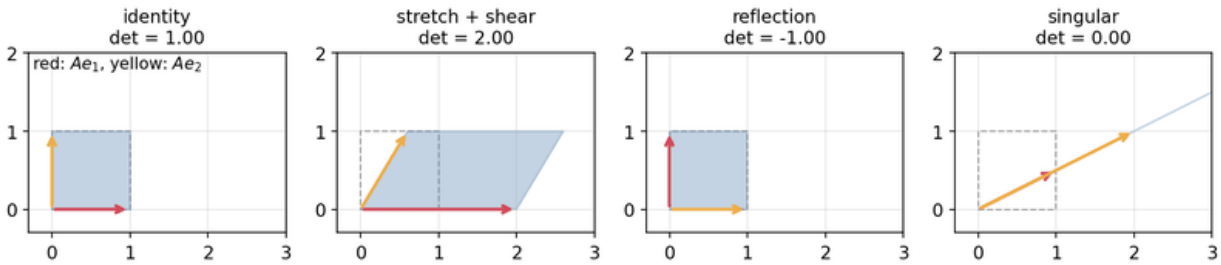


圖 3 — 行列式就是帶正負號的面積。虛線：單位正方形；陰影：它的像，由 *Ae1*（紅）與 *Ae2*（黃）張成。剪切不改變面積；鏡射讓正負號翻轉；奇異矩陣把正方形壓成一條線（$det = 0$）。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

行列式在機器學習中主要出現在兩個地方。多變量高斯密度中有 $\det(\Sigma)^{-1/2}$，是共變異數的體積正規化；ML 01B 會用到它。正規化流（normalizing flow，見 ML 07）需要 Jacobian 矩陣 J 的 $\log|\det J|$，來記錄可逆網路如何拉伸機率質量。這兩種情況都需要對數，絕不要直接算原始值：一個對角元素皆為 10^{-3} 的 500×500 對角矩陣，行列式是 10^{-1500}，在 float64 中會下溢成零。

```
import numpy as np

rng = np.random.default_rng(2)
A, B, C = rng.normal(size=(3, 4)), rng.normal(size=(4, 5)), rng.normal(size=(5, 3))
print("cyclic trace:", np.isclose(np.trace(A @ B @ C), np.trace(C @ A @ B)),
      np.isclose(np.trace(A @ B @ C), np.trace(B @ C @ A)))

S = rng.normal(size=(4, 4))
lam = np.linalg.eigvals(S)
print("trace = sum of eigenvalues:", np.isclose(np.trace(S), lam.sum().real))
print("det = product of eigenvalues:", np.isclose(np.linalg.det(S), np.prod(lam).real))

# log-determinant of a covariance without overflow/underflow
Sigma = np.diag(np.full(500, 1e-3))
print("det underflows:", np.linalg.det(Sigma))
sign, logdet = np.linalg.slogdet(Sigma)
print("slogdet:", sign, round(logdet, 2))
```

```
L = np.linalg.cholesky(Sigma)
print("via Cholesky:", round(2 * np.log(np.diag(L)).sum(), 2))
```

直接計算行列式印出 `0.0`，而 `slogdet` 與 Cholesky 兩種做法都得到 $\log\det = 500\log 10^{-3} \approx -3453.88$。對正定矩陣，若 $A = LL^\top$，則 $\det(A) = \prod_i L_{ii}^2$，所以 $\log\det A = 2\sum_i \log L_{ii}$。

實例：推導主成分分析

白話版。PCA 要找出資料點雲最貼近的那個平坦子空間。把資料投影上去，每個點就能壓縮成少數幾個數字，同時損失最少。

PCA 可追溯到 Pearson 的「最貼近點集的直線與平面」[9] 與 Hotelling 對「主成分」的分析 [10]；Jolliffe 與 Cadima 提供了現代的綜述 [11]。我們依照 [1] 的方式，只用上面介紹的工具來推導。

設定。有 n 個點 $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$，想讓每個點只用 $l < d$ 個數字儲存。先從每個點減去平均 $\mu = \frac{1}{n}\sum_i x^{(i)}$，因此可以假設資料已置中。把置中後的點當作 $X \in \mathbb{R}^{n\times d}$ 的各列。

步驟 1：選定編碼的形式。使用線性**解碼器** $g(c) = Dc$，$D \in \mathbb{R}^{d\times l}$。為了讓問題良好設定，要求 D 的各行單範正交：$D^\top D = I_l$。（否則可以無止境地縮小 c、放大 D。）

步驟 2：固定解碼器，找最佳編碼。對給定的 x，選擇 c 使重建誤差最小：

$$c^\ast = \arg\min_c \|x - Dc\|_2^2 = \arg\min_c \left(-2x^\top Dc + c^\top D^\top Dc\right) = \arg\min_c \left(-2x^\top Dc + c^\top c\right).$$

我們展開平方、丟掉與 c 無關的 $x^\top x$，並使用 $D^\top D = I$。對 c 的梯度是 $-2D^\top x + 2c$，令其為零就得到**編碼器**

$$f(x) = D^\top x.$$

所以編碼就是一次矩陣–向量乘法，而重建 $r(x) = g(f(x)) = DD^\top x$ 正是 x 到 D 生成空間的正交投影。

步驟 3：找最佳解碼器。現在選擇 D，使所有點的總重建誤差最小：

$$D^\ast = \arg\min_D \|X - XDD^\top\|_F^2 \quad \text{subject to } D^\top D = I_l.$$

先看 $l = 1$，此時 D 是單一單位向量 d。把 Frobenius 範數寫成跡並展開：

$$\begin{aligned}
\|X - Xdd^\top\|_F^2 &= \operatorname{Tr}\left((X - Xdd^\top)^\top (X - Xdd^\top)\right) \\
&= \operatorname{Tr}(X^\top X) - 2\operatorname{Tr}(X^\top Xdd^\top) + \operatorname{Tr}(dd^\top X^\top Xdd^\top) \\
&= \operatorname{Tr}(X^\top X) - 2\,d^\top X^\top Xd + d^\top X^\top Xd\,(d^\top d) \\
&= \operatorname{Tr}(X^\top X) - d^\top X^\top Xd.
\end{aligned}$$

第二項與第三項用循環性質把 $d^\top$ 移到最前面，最後一行用 $d^\top d = 1$ 化簡。第一項與 d 無關，所以最小化誤差等同於

$$d^\ast = \arg\max_d \; d^\top X^\top Xd \quad \text{subject to } d^\top d = 1.$$

這是對稱半正定矩陣 $X^\top X$ 的 Rayleigh 商。由特徵分解一節可知，它在 $X^\top X$ **最大特徵值對應的特徵向量**處達到最大。

步驟 4：多個主成分。對一般的 l 重複這個論證（例如用歸納法，要求每個新方向與先前的方向正交），答案是：D 的各行是 $X^\top X$ 前 l 大特徵值對應的特徵向量 [1]。最小誤差等於被捨棄的特徵值總和。

步驟 5：變異數觀點與 SVD 的連結。把 $X^\top X$ 除以 n，得到樣本共變異數 $S = \frac{1}{n} X^\top X$（有些教科書用 $n-1$）。$d^\top S d$ 是投影後資料 Xd 的變異數。所以「最小化重建誤差」與「最大化保留的變異數」選出的是同樣的方向——Pearson 與 Hotelling 的觀點一致。最後，若置中資料矩陣的精簡 SVD 為 $X = U\Sigma V^\top$，則 $X^\top X = V\Sigma^2 V^\top$：主方向就是右奇異向量 V，特徵值是 σ_i^2，編碼是 $XV_l = U_l\Sigma_l$。實務上 PCA 是用 X 的 SVD 計算，而不是先形成共變異數矩陣，理由與最小平方法中的條件數相同。

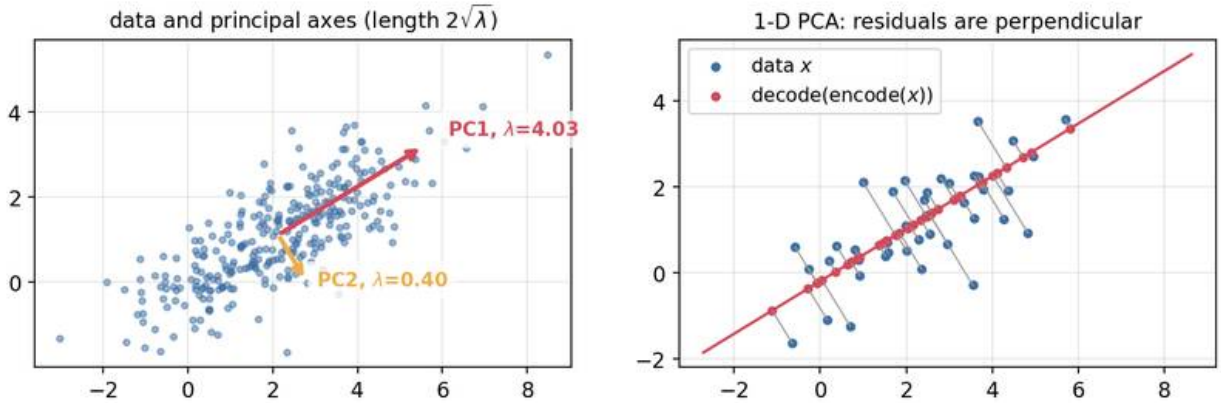


圖 *4* — 對 *300* 個相關二維資料點做 *PCA*。左：由樣本共變異數特徵分解得到的主軸，長度畫為 $2\sqrt{\lambda}$。右：一維 *PCA* 的重建 $decode(encode(x))$；每條灰色殘差都與紅線垂直，因為重建就是正交投影。

```
import numpy as np

rng = np.random.default_rng(0)
n, d, l = 500, 5, 2
Z = rng.normal(size=(n, l))
X = Z @ rng.normal(size=(l, d)) + 0.1 * rng.normal(size=(n, d)) + 3.0

mu = X.mean(axis=0)
Xc = X - mu                                  # centre first
S = Xc.T @ Xc / n                            # sample covariance (d x d)
lam, Q = np.linalg.eigh(S)
lam, Q = lam[::-1], Q[:, ::-1]               # descending
D = Q[:, :l]                                 # decoder matrix, orthonormal columns

codes = Xc @ D                               # encode: f(x) = D^T (x - mu)
X_hat = codes @ D.T + mu                     # decode: g(c) = D c + mu
mse = np.mean(np.sum((X - X_hat)**2, axis=1))
print("explained variance ratio:", (lam[:l].sum() / lam.sum()).round(4))
print("reconstruction MSE:", mse.round(5), " sum of discarded eigenvalues:",
 ↪  lam[l:].sum().round(5))
```

```
assert np.isclose(mse, lam[l:].sum())

# Same subspace from the SVD of the centred data matrix
U, s, Vt = np.linalg.svd(Xc, full_matrices=False)
assert np.allclose(s[:l]**2 / n, lam[:l])
assert np.allclose(np.abs(Vt[:l] @ D), np.eye(l), atol=1e-8)
print("SVD and eigendecomposition agree")
```

這份合成資料是五維的點，實際上落在一個二維平面附近，再加上小雜訊。兩個主成分解釋了 99.36% 的變異數，而平均平方重建誤差剛好等於被捨棄的三個特徵值總和，與步驟 4 的預測一致。SVD 的做法找到同樣的方向（差一個正負號，特徵向量的正負號本來就是任意的）。

三個實務提醒。一定要先置中，否則第一個「主成分」只會指向平均值。若特徵的單位無法比較，要先標準化，因為 PCA 不具尺度不變性。平均與方向都只能在訓練資料上擬合。PCA 是自編碼器的線性祖先；ML 06A 會發展它的機率版本與非線性後繼者。

僅線上版：*PCA：Jolliffe 與 Cadima 的綜述* — *https://blog.shyandram.dev/zh-tw/learn/ml/linear-algebra/papers/pcajolliffe-與-cadima-的綜述/*

你會遇到的矩陣微積分

白話版。訓練就是沿著梯度走，而機器學習中的大多數梯度，都是少數幾條矩陣公式反覆使用。

對向量 $x \in \mathbb{R}^n$ 的純量函數 $f(x)$，**梯度** $\nabla_x f \in \mathbb{R}^n$ 收集所有偏導數 $\partial f/\partial x_i$。對向量函數 $f : \mathbb{R}^n \to \mathbb{R}^m$，**Jacobian 矩陣** $J \in \mathbb{R}^{m\times n}$ 的元素為 $J_{ij} = \partial f_i/\partial x_j$。最常用的規則如下：

函數	梯度	備註
$a^\top x$	a	線性函數
$x^\top A x$	$(A + A^\top)x$	A 對稱時為 $2Ax$
$\tfrac{1}{2}\lVert Xw - y\rVert_2^2$	$X^\top(Xw - y)$	最小平方損失
$\lVert x\rVert_2^2$	$2x$	權重衰減
$\operatorname{Tr}(AB)$ 對 A	$B^\top$	搭配上面的跡技巧

每一條都可以把運算式寫成對索引的總和、逐項微分來驗證 [2]。向量函數的**連鎖律**是把 Jacobian 相乘：若 $y = g(x)$，而 $L = f(y)$ 是純量，則 $\nabla_x L = J_g^\top \nabla_y L$。反向傳播逐層套用這條規則，把轉置的局部 Jacobian 乘上傳進來的梯度，而不必真的建出完整的 Jacobian（那會大得驚人）。ML 02 會詳細建構這個過程，ML 01C 則加入 Hessian，也就是二階導數矩陣。

令最小平方梯度為零，就回到前一節的正規方程式。這個損失的 Hessian 是 $X^\top X$，為半正定，所以損失是凸函數，每個駐點都是全域最小值。用有限差分快速檢查公式：

```
import numpy as np

rng = np.random.default_rng(3)
X, y = rng.normal(size=(30, 4)), rng.normal(size=30)
w = rng.normal(size=4)
L = lambda w: 0.5 * np.sum((X @ w - y)**2)
```

```
grad = X.T @ (X @ w - y)                    # analytic gradient
eps = 1e-6
num = np.array([(L(w + eps*e) - L(w - eps*e)) / (2*eps) for e in np.eye(4)])
print("max gradient mismatch:", np.abs(grad - num).max())
# Setting the gradient to zero gives the normal equations
w_star = np.linalg.lstsq(X, y, rcond=None)[0]
print("gradient at solution ~ 0:", np.abs(X.T @ (X @ w_star - y)).max() < 1e-10)
```

解析梯度與數值梯度相差約 10^{-8}，正是有限差分捨入誤差預期的大小。

現代觀點

本篇的定義已有百年歷史，但它們在機器學習中的用法仍不斷改變。以下四條研究方向值得認識。

隨機化數值線性代數。大型矩陣的精確 SVD 很昂貴，大約需要 $O(mn\min(m,n))$ 次運算。Halko、Martinsson 與 Tropp 在 SIAM Review 的綜述 [12] 讓一個簡單的想法成為標準做法：用一個細長的隨機矩陣 Ω 乘上 A，「草描」出它的值域；把結果正交化得到基底 Q；再對小矩陣 $Q^{\top}A$ 做精確 SVD。加上少量過取樣與幾次冪迭代，對奇異值會衰減的矩陣就能得到準確的結果。Martinsson 與 Tropp 在 *Acta Numerica* 的綜述 [13] 涵蓋理論與大量演算法：範數估計、隨機嵌入、迴歸、低秩近似、半正定矩陣的 Nyström 近似、串流演算法，以及機器學習中的核矩陣。Murray 等人 [14]（LAPACK Working Note 299，2023）從軟體角度切入：他們指出 RandNLA 方法已進入 MATLAB、NAG Library、NVIDIA 的 cuSOLVER 與 scikit-learn，並主張建立標準的「RandBLAS」與「RandLAPACK」程式庫。scikit-learn 的隨機化 PCA 求解器就是直接的後代。核心演算法只要幾行：

```
import numpy as np

def randomized_svd(A, k, oversample=10, power_iters=2, seed=0):
    rng = np.random.default_rng(seed)
    Omega = rng.normal(size=(A.shape[1], k + oversample))  # random test matrix
    Y = A @ Omega                                          # sample the range of A
    for _ in range(power_iters):                           # sharpen a slow spectrum
        Y = A @ (A.T @ Y)
    Q, _ = np.linalg.qr(Y)                                 # orthonormal basis
    B = Q.T @ A                                            # small (k+p) x n matrix
    Ub, s, Vt = np.linalg.svd(B, full_matrices=False)
    return (Q @ Ub)[:, :k], s[:k], Vt[:k]

rng = np.random.default_rng(1)
m, n, r = 2000, 500, 20
A = rng.normal(size=(m, r)) @ rng.normal(size=(r, n)) + 0.01 * rng.normal(size=(m, n))
U, s, Vt = randomized_svd(A, k=r)
s_exact = np.linalg.svd(A, compute_uv=False)[:r]
print("max relative error in top singular values:", np.max(np.abs(s - s_exact) / s_exact))
rel = np.linalg.norm(A - (U * s) @ Vt) / np.linalg.norm(A)
print("relative reconstruction error:", round(rel, 5))
```

對這個近似秩 20 的矩陣，前 20 個奇異值與精確值的相對誤差約 10^{-15}，而秩 20 重建的誤差（約 0.2%）由加入的雜訊決定。冪迭代是奇異值衰減緩慢時仍能成功的關鍵；若矩陣的奇異值完全平坦，任何秩 k 的方法——隨機或非隨機——都做不好。

大型模型中的低秩結構。矩陣乘法的外積觀點，已經成為調整大型網路的實用工具。LoRA [15] 凍結預訓練權重 $W_0 \in \mathbb{R}^{d\times k}$，只學習更新量 $\Delta W = BA$，其中 $B \in \mathbb{R}^{d\times r}$、$A \in \mathbb{R}^{r\times k}$，且 $r \ll \min(d,k)$。可訓練的數字只有 $r(d+k)$ 個，而不是 dk 個；以 $d = k = 4096$、$r = 8$ 為例，每個矩陣是 65,536 個，而不是約 1,680 萬個。B 初始化為零，所以訓練恰好從預訓練模型出發。GaLore [16]（ICML 2024）則讓所有權重都可訓練，但把梯度投影到低秩子空間，以縮小最佳化器的記憶體用量；作者認為像 LoRA 那樣把權重更新本身限制在低秩子空間，可能不如全秩訓練。兩者都依賴 Eckart–Young 的直覺：矩陣中有用的內容大多集中在少數幾個奇異方向。這在特定任務上是否成立，是實證問題，不是定理。

控制譜範數。網路對輸入擾動的敏感度，上限是各層譜範數的乘積（在 ReLU 這類 1-Lipschitz 激勵函數下）。譜正規化（spectral normalization）[17] 把每個權重矩陣除以最大奇異值的估計值，而這個估計在每個訓練步驟只需一次冪迭代，成本很低。它原本是為了穩定 GAN 的判別器而提出（見 ML 07），凡是需要 Lipschitz 上界的地方都能看到同樣的想法。

超越矩陣。卷積核與影片特徵天生就是三軸以上的張量。Kolda 與 Bader 在 SIAM Review 的綜述 [18] 是張量分解的標準入門，涵蓋 CP 分解（秩一張量的總和，推廣了外積觀點）與 Tucker 分解（一個小的核心張量沿每個模態乘上一個矩陣，推廣了 SVD）。與矩陣不同，張量的秩一般而言很難計算，最佳低秩近似也不一定存在，Eckart–Young 那幅清楚的圖像無法直接沿用。這些分解被用來壓縮網路層，以及分析多向資料。

開放問題。為什麼訓練後的權重矩陣常有快速衰減的奇異值？這和泛化有什麼關係？調整用的秩可以多低才不損失品質？這取決於任務還是模型？隨機化演算法要如何與低精度硬體結合——那裡的捨入誤差遠大於 float64？這些問題都還沒有定論，而本篇的工具正是閱讀相關論文所需要的。若想找把線性代數與深度學習連起來的完整教科書，可參考 [1]、[2] 與 [19]。

重點整理

- 矩陣相乘或陣列廣播前，先說明各軸意義並檢查形狀。
- 秩與零空間限制可能的解；解線性系統時，使用求解器而非顯式計算反矩陣。
- 實對稱矩陣具有正交規一特徵基底；特徵值正負描述其二次型。
- 截斷 SVD 在譜範數或 Frobenius 誤差下給出最佳低秩近似。
- 偽反矩陣選出最小範數的最小平方解，小奇異值可能放大雜訊。
- 對置中資料，在固定子空間維度下，PCA 最大化保留變異並最小化平方重建誤差。

練習

1. **形狀與偏差**。一個批次形狀為 32×64，權重矩陣形狀為 64×10。偏差應該是什麼形狀？若改用形狀 $(32, 1)$ 的偏差，會出什麼問題？

提示

形狀 $(10,)$ 的偏差會沿 32 列廣播：每個輸出單元一個偏移量，由所有樣本共用。$(32,1)$ 的偏差則沿 10 行廣播，變成每個樣本一個偏移量。程式照樣能跑，但「偏差」現在取決於樣本在批次中的位置，這沒有意義，而且會隨批次大小改變。

2. **投影**。設 $Q \in \mathbb{R}^{n\times k}$ 的各行單範正交，$P = QQ^\top$。證明 $P^2 = P$、$P^\top = P$，且 $x - Px$ 與 Q 的每一行正交。P 的特徵值是什麼？

   提示

   $P^2 = Q(Q^\top Q)Q^\top = QIQ^\top = P$。正交性：計算 $Q^\top(x - Px) = Q^\top x - Q^\top QQ^\top x = 0$。由 $P^2 = P$，任何特徵值滿足 $\lambda^2 = \lambda$，所以 $\lambda \in \{0, 1\}$：特徵值 1 的重數為 k（子空間本身），0 的重數為 $n - k$（它的正交補空間）。因此 $\operatorname{Tr}(P) = k$。

3. **手算 SVD**。透過計算 $A^\top A$ 的特徵值，求 $A = \begin{bmatrix} 3 & 0 \\ 4 & 5 \end{bmatrix}$ 的奇異值。驗證 $\|A\|_F^2 = \sigma_1^2 + \sigma_2^2$ 以及 $|\det A| = \sigma_1\sigma_2$。

   提示

   $A^\top A = \begin{bmatrix} 25 & 20 \\ 20 & 25 \end{bmatrix}$ 的特徵值是 45 與 5，所以 $\sigma_1 = 3\sqrt{5}$、$\sigma_2 = \sqrt{5}$。於是 $\sigma_1^2 + \sigma_2^2 = 50 = 9 + 16 + 25$，且 $\sigma_1\sigma_2 = 15 = |\det A| = |3 \cdot 5 - 0 \cdot 4|$。一般而言，方陣的 $|\det A| = \prod_i \sigma_i$，因為 $|\det U| = |\det V| = 1$。

4. **條件數與嶺迴歸**。從最小平方的程式碼出發，把第二個特徵換成第一個特徵的完全複本。`np.linalg.lstsq` 會回傳什麼？為什麼它是最小範數解？接著推導嶺迴歸解的 SVD 形式，並證明每個係數 $\sigma/(\sigma^2 + \lambda)$ 最多是 $1/(2\sqrt{\lambda})$。

   提示

   兩行完全相同時秩為 1，解集合是一條直線 $w_1 + w_2 = c$。`lstsq` 使用 SVD，回傳偽反矩陣解，也就是平分權重 $w_1 = w_2 = c/2$，因為直線上離原點最近的就是這一點。至於上界，最大化 $g(\sigma) = \sigma/(\sigma^2 + \lambda)$：$g'(\sigma) = 0$ 發生在 $\sigma = \sqrt{\lambda}$，此時 $g = 1/(2\sqrt{\lambda})$。

5. **PCA 的誤差**。利用本篇的推導，證明 l 個主成分的 PCA 總重建誤差等於 $\sum_{i>l} \lambda_i(X^\top X)$，而保留的變異數比例是 $\sum_{i\le l}\lambda_i / \sum_i \lambda_i$。接著對 `sklearn.datasets.load_digits()` 做 PCA，找出保留 90% 變異數所需的最小 l。

   提示

   寫成 $\|X - XDD^\top\|_F^2 = \operatorname{Tr}(X^\top X) - \operatorname{Tr}(D^\top X^\top XD)$（使用 $D^\top D = I$ 與循環性質）。當 D 是前 l 個特徵向量時，第二個跡等於 $\sum_{i\le l}\lambda_i$，而 $\operatorname{Tr}(X^\top X) = \sum_i \lambda_i$。手寫數字資料：把 1797×64 的矩陣置中，計算奇異值，再對 σ_i^2 取累積和。請回報你自己算出的數字，不要相信別人引用的數字。

6. **行列式與體積**。某個 2×2 矩陣把單位正方形映射成頂點為 0、$(2,1)$、$(1,3)$、$(3,4)$ 的平行四邊形。寫出這樣的矩陣，計算它的行列式，並說明為什麼正規化流必須記錄每一層的 $\log|\det J|$。

提示

把兩條邊向量放進各行：$A = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}$，$\det A = 5$，所以面積放大 5 倍。若一個映射把體積放大 5 倍，機率密度就必須縮小同樣倍數，總機率才能維持為 1。疊加多層時這些倍數相乘，取對數後變成相加，這就是正規化流逐層加總對數行列式的原因。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 2 “Linear Algebra”. Book
2. M. P. Deisenroth, A. A. Faisal and C. S. Ong, *Mathematics for Machine Learning*, Cambridge University Press, 2020. Book
3. G. Strang, *Introduction to Linear Algebra*, 6th ed., Wellesley-Cambridge Press, 2023. Book site
4. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, 2013. DOI
5. L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, SIAM, 1997. DOI
6. C. Eckart and G. Young, “The Approximation of One Matrix by Another of Lower Rank,” *Psychometrika*, vol. 1, pp. 211–218, 1936. DOI
7. R. Penrose, “A Generalized Inverse for Matrices,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, pp. 406–413, 1955. DOI
8. A. E. Hoerl and R. W. Kennard, “Ridge Regression: Biased Estimation for Nonorthogonal Problems,” *Technometrics*, vol. 12, pp. 55–67, 1970. DOI
9. K. Pearson, “On Lines and Planes of Closest Fit to Systems of Points in Space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, pp. 559–572, 1901. DOI
10. H. Hotelling, “Analysis of a Complex of Statistical Variables into Principal Components,” *Journal of Educational Psychology*, vol. 24, pp. 417–441, 1933. DOI
11. I. T. Jolliffe and J. Cadima, “Principal Component Analysis: A Review and Recent Developments,” *Philosophical Transactions of the Royal Society A*, vol. 374, 20150202, 2016. DOI
12. N. Halko, P.-G. Martinsson and J. A. Tropp, “Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions,” *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011. DOI ·arXiv
13. P.-G. Martinsson and J. A. Tropp, “Randomized Numerical Linear Algebra: Foundations and Algorithms,” *Acta Numerica*, vol. 29, pp. 403–572, 2020. DOI ·arXiv
14. R. Murray, J. Demmel, M. W. Mahoney, N. B. Erichson, M. Melnichenko, O. A. Malik, L. Grigori, P. Luszczek, M. Dereziński, M. E. Lopes, T. Liang, H. Luo and J. Dongarra, “Randomized Numerical Linear Algebra: A Perspective on the Field With an Eye to Software,” LAPACK Working Note 299, 2023. arXiv
15. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang and W. Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” in *Proc. International Conference on Learning Representations (ICLR)*, 2022. arXiv
16. J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar and Y. Tian, “GaLore: Memory-Efficient LLM Training by Gradient Low-Rank Projection,” *ICML*, 2024. arXiv

17. T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida, “Spectral Normalization for Generative Adversarial Networks,” *ICLR*, 2018. arXiv
18. T. G. Kolda and B. W. Bader, “Tensor Decompositions and Applications,” *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009. DOI
19. C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024. Book

ML 01B・機率與資訊理論

隨機變數、常見分布、貝氏定理、期望值與共變異數、概似、熵、交叉熵、KL 散度與結構化模型，並連結到神經網路的訓練與評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/probability-information/

你將學到

- 區分機率、密度與分布。
- 連結聯合、邊際與條件機率。
- 描述變異與相依，避免混淆相關與獨立。
- 用貝氏定理、概似與先驗估計未知量。
- 連結熵、交叉熵、KL 散度與學習目標。

先看全貌

神經網路永遠看不到整個世界。它只看到有限的訓練集——由不完美的感測器量測、由忙碌的人標註——接著卻要回答從沒看過的影像。這個過程的每一步都牽涉**不確定性**。機率論就是描述不確定性的數學：它告訴我們如何表達自己不知道的事、如何結合證據，以及如何做出承認懷疑的預測。與它關係密切的資訊理論，則衡量不確定性有多少，以及使用錯誤分布要付出多少代價。兩者合起來，幾乎可以解釋本站 ML 與 DL 系列中的每一個損失函數。

> **用最簡單的話說**
>
> 想像一個看不到內部的彈珠袋。**機率**是你對下一顆彈珠的最佳猜測：「大概是紅色，也許是藍色，幾乎不可能是綠色。」每摸出一顆彈珠，你就更了解這個袋子，並更新你的猜測——這個更新就是**貝氏定理**。**熵**是你預期會有多驚訝：全是紅色彈珠的袋子永遠不會讓你驚訝；十種顏色各占一樣多的袋子則常常讓你驚訝。機器學習模型是一個很厲害的猜測者，而訓練就是讓它對真正摸出的彈珠越來越不驚訝。

本篇依循 Goodfellow、Bengio 與 Courville《Deep Learning》中的「應用數學」部分 [1]，並補充 Bishop 與 Bishop [2]、Murphy [3] 以及《Mathematics for Machine Learning》[4] 的內容。它位於 ML 01A・線性代數與 ML 01C・數值計算之間。如果你接下來要讀圖形識別系列，PR 01・貝氏決策會直接延伸這裡的貝氏定理一節。

為什麼需要機率？

白話版。機率讓學習系統能描述不確定的觀測、預測與模型假設。

Goodfellow 等人 [1] 列出三種不確定性的來源：

1. **系統本身的隨機性**。有些系統本來就是隨機的——光子抵達相機感測器的時間（散粒雜訊）、骰子的點數，或行人踏下人行道的確切時刻。

2. **觀測不完整**。即使系統是決定性的，只要我們看不到它所有的變數，它看起來就是隨機的。一張汽車照片無法告訴你煞車燈是不是快要亮起。
3. **建模不完整**。模型必須捨棄資訊才有用處。只看 32×32 縮圖的分類器無法找回降取樣時遺失的細節，所以它對細粒度類別的預測必然帶有不確定性。

使用帶有不確定性的簡單規則（「大多數鳥會飛」），往往比使用永遠正確但極其複雜的規則（「鳥會飛，但企鵝、鴕鳥、受傷的鳥……除外」）更實際。

解讀機率的兩種方式

機率可以有兩種解讀，而兩者遵守相同的數學規則。

- **頻率學派機率**描述可重複事件的長期頻率：「這枚硬幣擲很多次，大約 70% 會出現正面」。
- **貝氏機率**描述對某件不一定能重複的事的信念程度：「看了這張 X 光片，我有 40% 的把握病人罹病」。

只要要求信念程度滿足幾項符合常識的一致性性質，就能證明它們必須遵守與頻率完全相同的規則，因此同一套計算可以同時服務兩種解讀 [1], [3]。實務上，深度學習混用兩者：資料被視為從某個分布抽出的樣本（頻率觀點），而模型的輸出常被解讀成對標籤的信念（貝氏觀點）。

隨機變數與機率分布

白話版。隨機變數是事先不知道值的量；機率分布說明每個值出現的可能性有多大。

[1] 用正體字母（例如 x）表示隨機變數；本篇用 X 表示變數、x 表示它的一個取值。隨機變數可以是**離散的**（有限或可數個值，例如 $\{1, \dots, K\}$ 中的類別標籤），也可以是**連續的**（實數值，例如被視為實數的像素強度）。

機率質量函數

離散變數有**機率質量函數**（probability mass function, PMF）$P(X = x)$，常簡寫為 $P(x)$。它必須滿足

$$0 \le P(x) \le 1 \text{ for every } x, \qquad \sum_{x} P(x) = 1.$$

這裡的總和涵蓋 X 所有可能的值 x。第一個條件說沒有事件比「確定」更確定；第二個條件說總有某件事會發生。聯合 PMF $P(X = x, Y = y)$ 則是兩個事件同時發生的機率。

機率密度函數

連續變數有**機率密度函數**（probability density function, PDF）$p(x)$，必須滿足

$$p(x) \ge 0, \qquad \int p(x)\, dx = 1.$$

密度不是機率。X 落在區間 $[a, b]$ 的機率是密度曲線下的面積：

$$P(a \le X \le b) = \int_a^b p(x)\,dx.$$

任何單一精確值的機率都是零。因為只有面積必須不超過一，密度本身可以大於一：標準差為 0.25 的窄高斯，峰值約為 1.6（圖 1 右上）。把密度值當成「機率 1.6」回報，是常見錯誤。

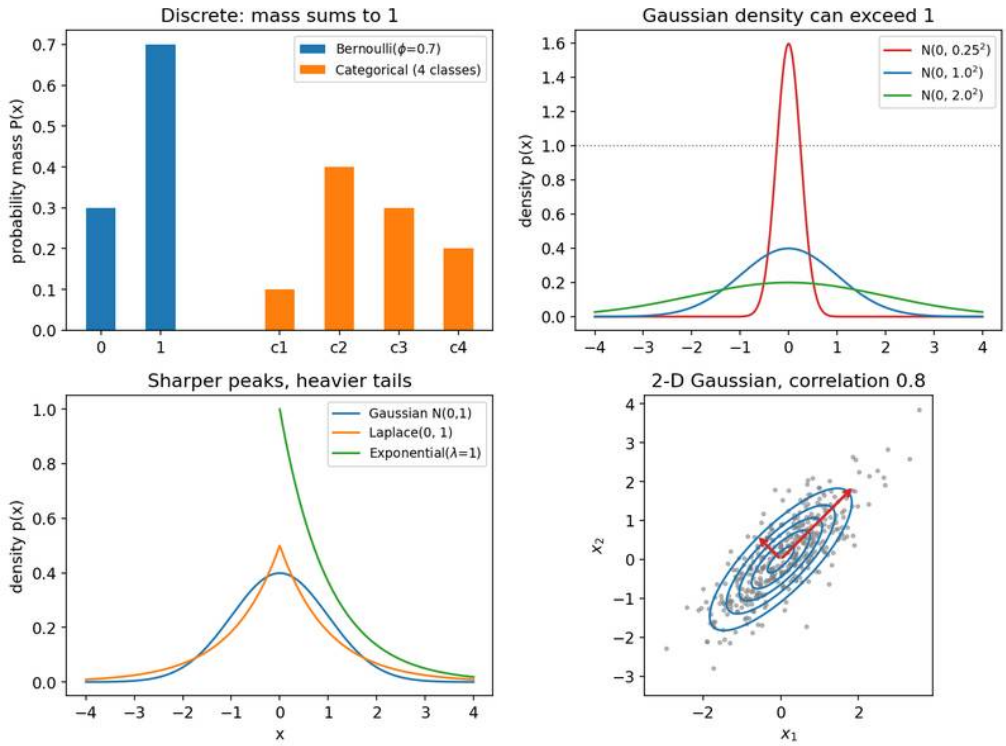


圖 *1* — 常見分布。左上：離散機率質量總和為一。右上：密度可以超過一，因為只有面積必須等於一。左下：*Laplace* 密度峰值比高斯尖、尾巴比高斯厚；指數分布只定義在 $x \ge 0$。右下：相關係數 *0.8* 的二維高斯；紅色箭頭是共變異數矩陣的特徵向量，長度為兩倍標準差。

邊際機率

白話版。如果知道兩件事如何一起變化，就可以把其中一件「加總掉」，得到另一件事單獨的變化方式。

給定 X 與 Y 的聯合分布，X 的**邊際**分布為

$$P(x) = \sum_y P(x, y) \quad \text{(discrete)}, \qquad p(x) = \int p(x, y)\,dy \quad \text{(continuous)}.$$

這就是**加法規則**（sum rule）。如果把聯合分布寫成表格，列代表 x、欄代表 y，那麼 x 的邊際分布就是每一列的總和——過去習慣寫在紙張的邊緣（margin），因此得名。

條件機率

白話版。條件機率回答的是：「在已經知道那件事的情況下，這件事有多可能？」

$$P(y \mid x) = \frac{P(x, y)}{P(x)}, \qquad P(x) > 0.$$

$P(y \mid x)$ 是觀察到 $X = x$ 之後 $Y = y$ 的機率，只有在條件真的可能發生時才有定義。移項後得到**乘法規則**（product rule）$P(x, y) = P(y \mid x) P(x)$。

不要把條件化和介入混為一談。「有人帶傘的情況下街道是濕的」機率很高，但強迫大家帶傘並不會讓街道變濕。關於介入的問題屬於因果推論，光有聯合分布是不夠的。

機率的連鎖規則

反覆套用乘法規則，可以把任何 n 個變數的聯合分布分解成

$$P(x^{(1)}, \dots, x^{(n)}) = P(x^{(1)}) \prod_{i=2}^{n} P(x^{(i)} \mid x^{(1)}, \dots, x^{(i-1)}).$$

這裡 $x^{(i)}$ 是第 i 個變數的值。這個恆等式是精確的——不需要任何假設。它是自迴歸模型的骨幹：語言模型或自迴歸影像模型，會根據前面所有的 token 或像素，一次預測一個（ML 09、NLP 01）。建模的選擇在於如何參數化每一個條件分布，而不在分解本身。

獨立與條件獨立

如果

$$p(x, y) = p(x)\, p(y) \quad \text{for all } x, y,$$

就稱兩個變數**獨立**，記為 $X \perp Y$。如果

$$p(x, y \mid z) = p(x \mid z)\, p(y \mid z) \quad \text{for all } x, y, z,$$

就稱它們在給定 Z 下**條件獨立**，記為 $X \perp Y \mid Z$。

冰淇淋銷量和曬傷人數彼此相關，但在給定天氣之後，兩者大致獨立。條件獨立是讓高維模型變得可計算的主要工具——單純貝氏（naive Bayes）假設特徵在給定類別下彼此獨立；隱藏式馬可夫模型假設給定目前狀態後，未來與過去獨立。這些都是假設 [3]：應該由問題本身來支持，而不是因為能讓代數變簡單就採用。

程式碼：聯合機率表、邊際與條件機率

下面的例子為天氣與帶傘建立一張小型聯合機率表，計算邊際與條件機率，並檢查是否獨立。它也展示了兩個變數明顯相依、共變異數卻（幾乎）為零的情況。

```
import numpy as np
```

```
# Joint table P(weather, umbrella): rows = sunny/rainy, columns = no/yes
P = np.array([[0.56, 0.14],
              [0.06, 0.24]])
assert np.isclose(P.sum(), 1)

p_weather = P.sum(axis=1)            # marginal over umbrella
p_umbrella = P.sum(axis=0)           # marginal over weather
p_umb_given_w = P / p_weather[:, None]   # each row sums to 1
print("P(weather):", p_weather)
print("P(umbrella):", p_umbrella)
print("P(umbrella | weather):\n", p_umb_given_w)
print("independent?", np.allclose(P, np.outer(p_weather, p_umbrella)))

# Zero covariance does not imply independence
rng = np.random.default_rng(0)
x = rng.uniform(-1, 1, 200_000)
y = x**2                              # y is a deterministic function of x
print("cov(x, y) ~", np.cov(x, y)[0, 1].round(4))
```

輸出：邊際分布為 $[0.7, 0.3]$ 與 $[0.62, 0.38]$；條件機率表為 $[[0.8, 0.2], [0.2, 0.8]]$；兩個變數不獨立；而 x 與 x^2 的樣本共變異數約為 -0.0007，幾乎是零——即使 y 完全由 x 決定。

期望值、變異數與共變異數

白話版。期望值是長期平均；變異數是數值通常偏離平均多遠；共變異數則說明兩個量是否傾向一起上升、一起下降。

函數 f 在某個分布下的**期望值**為

$$\mathbb{E}_{x\sim P}[f(x)] = \sum_x P(x) f(x), \qquad \mathbb{E}_{x\sim p}[f(x)] = \int p(x) f(x)\, dx.$$

下標 $x \sim P$ 指出對哪個分布取平均；意思明確時可以省略。期望值是**線性的**：對常數 α, β，$\mathbb{E}[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}[f(x)] + \beta \mathbb{E}[g(x)]$。幾乎每一個訓練損失都是對資料分布的期望值，並用小批次（mini-batch）的平均來估計。

變異數衡量數值在平均 $\mu = \mathbb{E}[f(x)]$ 周圍的分散程度：

$$\operatorname{Var}(f(x)) = \mathbb{E}\big[(f(x) - \mu)^2\big] = \mathbb{E}[f(x)^2] - \mu^2.$$

它的平方根是**標準差**，單位與 $f(x)$ 相同。

兩個量的**共變異數**衡量它們如何一起變化：

$$\operatorname{Cov}(f(x), g(y)) = \mathbb{E}\big[(f(x) - \mathbb{E}[f(x)])\,(g(y) - \mathbb{E}[g(y)])\big].$$

共變異數為正，代表兩者傾向同時高於（或低於）各自的平均；為負，代表一個高時另一個傾向低。它的大小取決於單位，所以我們常把它正規化成**相關係數**

$$\rho = \frac{\operatorname{Cov}(X, Y)}{\sqrt{\operatorname{Var}(X)\operatorname{Var}(Y)}} \in [-1, 1].$$

獨立意味著共變異數為零，但**共變異數為零不代表獨立**。共變異數只衡量線性相依。在上面的程式碼中，x 在 $[-1,1]$ 上均勻分布、$y = x^2$：知道 x 就能確切知道 y，但因為關係對稱且非線性，共變異數為零。稍後介紹的互資訊能捕捉任何形式的相依。

共變異數矩陣

對平均為 $\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}]$ 的隨機向量 $\mathbf{x} \in \mathbb{R}^n$，**共變異數矩陣**為

$$\Sigma = \mathrm{Cov}(\mathbf{x}) = \mathbb{E}\big[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\top\big], \qquad \Sigma_{ij} = \mathrm{Cov}(x_i, x_j).$$

對角線是各分量的變異數 $\Sigma_{ii} = \mathrm{Var}(x_i)$。$\Sigma$ 是對稱半正定矩陣，因此（見 ML 01A）具有非負實特徵值與正交特徵向量。特徵向量是資料雲的主軸，特徵值是沿主軸的變異數——也就是圖 1 中的紅色箭頭——這正是 PCA 所計算的東西。對仿射映射 $\mathbf{y} = A\mathbf{x} + \mathbf{b}$，有 $\mathbb{E}[\mathbf{y}] = A\boldsymbol{\mu} + \mathbf{b}$ 與 $\mathrm{Cov}(\mathbf{y}) = A\Sigma A^\top$。

常見機率分布

白話版。有一小群分布會一再出現。選擇其中一個，就是對資料或目標做出一個假設。

Bernoulli 分布

單一二元變數 $X \in \{0,1\}$，參數 $\phi \in [0,1]$：

$$P(X=1) = \phi, \quad P(X=0) = 1-\phi, \quad P(X=x) = \phi^x(1-\phi)^{1-x}.$$

平均為 ϕ，變異數為 $\phi(1-\phi)$。輸出層使用 sigmoid 的二元分類器，就是在預測標籤之 Bernoulli 分布的參數 ϕ。

類別分布（multinoulli）

具有 K 個狀態的單一離散變數，以機率向量 $\mathbf{p} \in [0,1]^K$（$\sum_k p_k = 1$）為參數：

$$P(X = k) = p_k.$$

若標籤以 one-hot 編碼為 $\mathbf{y} \in \{0,1\}^K$，則 $P(\mathbf{y}) = \prod_k p_k^{y_k}$。softmax 分類器輸出的正是這樣的向量。（多項分布計算多次抽取的結果次數；只抽一次時就退化成類別分布，Goodfellow 等人稱之為「multinoulli」[1]。）

高斯（常態）分布

最常用的連續分布：

$$\mathcal{N}(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\!\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big).$$

μ 是平均（也是峰值位置），σ^2 是變異數。有時改用**精確度**（precision）$\beta = 1/\sigma^2$，在變異數經常改變時計算起來比較省。

高斯之所以成為預設選擇有兩個原因。**中央極限定理**說，許多變異數有限的獨立變數相加，會趨近高斯分布，所以由許多微小因素組成的量測雜訊往往接近高斯。而在實數線上、平均與變異數固定的所有分布中，高斯的**熵最大**：它加入的額外結構最少。

在 n 維中，

$$\mathcal{N}(\mathbf{x};\boldsymbol{\mu},\Sigma) = \frac{1}{\sqrt{(2\pi)^n \det \Sigma}} \exp\Big(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big),$$

其中 $\boldsymbol{\mu} \in \mathbb{R}^n$ 是平均向量，$\Sigma \in \mathbb{R}^{n\times n}$ 是對稱正定的共變異數矩陣。**精確度矩陣** $\Lambda = \Sigma^{-1}$ 有個漂亮的意義：$\Lambda_{ij} = 0$ 恰好代表在給定其他所有座標下，x_i 與 x_j 條件獨立。**等向**（isotropic）高斯的 $\Sigma = \sigma^2 I$——這正是擴散模型使用的雜訊分布（ML 08）。

它和你會用到的損失函數有一個根本的連結：如果迴歸網路預測一個固定變異數之高斯的平均，那麼最大化概似就等於最小化**均方誤差**。

指數分布與 Laplace 分布

有時候我們想在特定位置放一個尖峰。

$$p(x;\lambda) = \lambda\, \mathbf{1}_{x\ge 0}\, e^{-\lambda x}, \qquad \mathrm{Laplace}(x;\mu,\gamma) = \frac{1}{2\gamma}\exp\Big(-\frac{|x-\mu|}{\gamma}\Big).$$

$\lambda > 0$ 是速率，$\mathbf{1}_{x\ge 0}$ 是指示函數（$x \ge 0$ 時為 1，否則為 0），μ 是位置，$\gamma > 0$ 是尺度。和高斯相比，Laplace 密度的峰值更尖、尾巴更厚（圖 1 左下）。用最大概似預測 Laplace 分布的位置，等同於最小化**絕對誤差**（L_1 損失），這就是 L_1 比 L_2 更能容忍離群值的原因；而對權重使用 Laplace 先驗，就得到 L_1 正則化（稀疏性）。

Dirac 分布與經驗分布

Dirac delta $\delta(x-\mu)$ 把所有質量放在點 μ：它在其他地方都是零，積分卻等於一。它是一種廣義函數，由它在積分中的作用來定義：$\int \delta(x-\mu) f(x)\,dx = f(\mu)$。

訓練集 $\{\mathbf{x}^{(1)},\dots,\mathbf{x}^{(m)}\}$ 定義了**經驗分布**

$$\hat{p}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} \delta(\mathbf{x}-\mathbf{x}^{(i)}),$$

在 m 個樣本上各放 $1/m$ 的質量。把損失對整個資料集取平均來訓練模型，就等於在 $\hat{p}$ 下取期望值。經驗分布也是讓訓練資料概似最大的分布——這正是只最小化訓練損失會過擬合（overfitting）的原因。

混合分布與潛在變數

白話版。混合分布先隨機選一個「類型」，再從那個類型的分布中抽樣。

$$P(\mathbf{x}) = \sum_{k=1}^{K} P(z = k)\, P(\mathbf{x} \mid z = k).$$

z 是一個類別型的**潛在變數**（latent variable）——我們從未直接觀察到的變數——以機率 $\pi_k = P(z = k)$ 選出，$P(\mathbf{x} \mid z = k)$ 是第 k 個成分。**高斯混合模型**（Gaussian mixture model, GMM）使用高斯成分 $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k)$；只要成分夠多，它可以把任何平滑密度近似到任意指定的非零誤差 [1]。給定一個觀測 $\mathbf{x}$，貝氏定理給出**後驗責任值** $P(z = k \mid \mathbf{x})$：該點由第 k 個成分產生的可能性。用期望最大化（EM）擬合 GMM 是 PR 07 的主題。

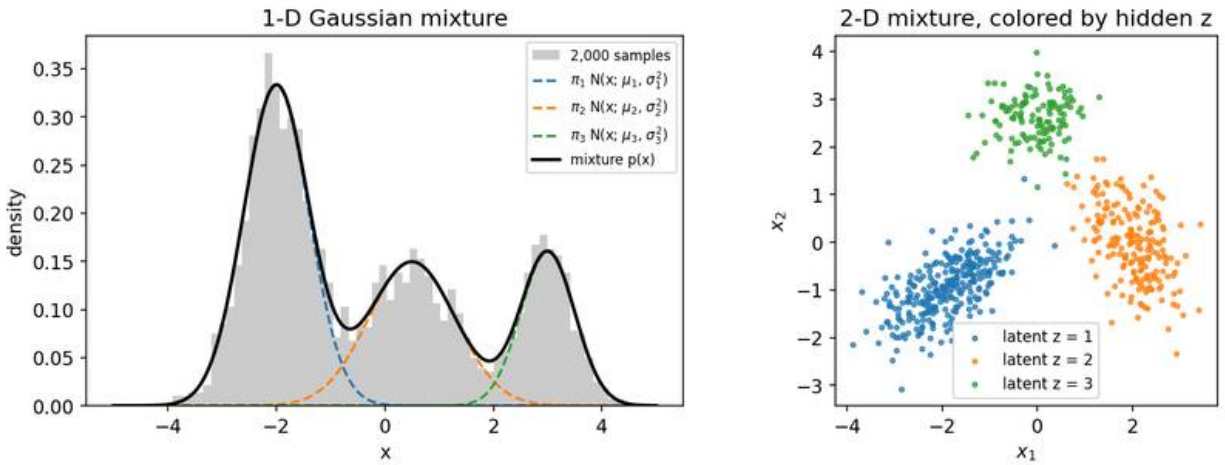


圖 2 — 高斯混合。左：黑色密度是三個成分的加權總和；灰色直方圖是先選成分、再從該成分抽樣得到的 *2,000* 個樣本。右：在二維中，顏色代表隱藏成分 z——真實資料集並不會告訴你這個資訊。

潛在變數是現代生成模型的核心。VAE（ML 07）粗略地說，就是潛在變數 $\mathbf{z}$ 為連續、且 $p(\mathbf{x} \mid \mathbf{z})$ 由神經網路表示的混合模型 [8]；此時邊際分布 $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})\, d\mathbf{z}$ 無法直接計算，只能近似。

```
import numpy as np

rng = np.random.default_rng(0)
pi = np.array([0.5, 0.3, 0.2])      # mixing weights P(z = k)
mu = np.array([-2.0, 0.5, 3.0])     # component means
sd = np.array([0.6, 0.8, 0.5])      # component standard deviations

# Ancestral sampling: first the hidden component z, then x given z
z = rng.choice(3, size=100_000, p=pi)
x = rng.normal(mu[z], sd[z])

def log_normal(x, m, s):
    return -0.5*((x - m)/s)**2 - np.log(s) - 0.5*np.log(2*np.pi)

def mixture_logpdf(x):
    # log sum_k pi_k N(x; mu_k, sd_k^2), computed with log-sum-exp
    a = np.log(pi) + log_normal(x[:, None], mu, sd)
    m = a.max(axis=1, keepdims=True)
```

```
    return (m + np.log(np.exp(a - m).sum(axis=1, keepdims=True))).ravel()

grid = np.linspace(-6, 6, 4001)
print("density integrates to",
 ↪  (np.exp(mixture_logpdf(grid)).sum()*(grid[1]-grid[0])).round(4))
print("sample mean", x.mean().round(3), "vs exact", (pi*mu).sum().round(3))
# Posterior responsibility P(z | x) for one observation
a = np.log(pi) + log_normal(np.array([0.0])[:, None], mu, sd)
resp = np.exp(a - a.max()); resp /= resp.sum()
print("P(z | x=0):", resp.round(3))
```

密度積分為 1.0，樣本平均（−0.255）與精確平均 $\sum_k \pi_k \mu_k = -0.25$ 相符，而點 $x = 0$ 以 0.99 的機率被歸給中間的成分。這裡用到的 log-sum-exp 技巧在 ML 01C 中說明。

實用函數：sigmoid 與 softplus

白話版。 sigmoid 把任何實數壓進 0 到 1 之間，變成一個機率；softplus 是 $\max(0, x)$ 的平滑版本，輸出永遠為正。

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \qquad \zeta(x) = \log\left(1 + e^{x}\right).$$

logistic sigmoid σ 產生 Bernoulli 分布的參數 ϕ；當 $|x|$ 很大時它會**飽和**（變平），使那裡的梯度很小。**softplus** ζ 產生正數，例如標準差 σ 或精確度 β；它的名字來自它是「正部」$x^{+} = \max(0, x)$ 的柔化版本。

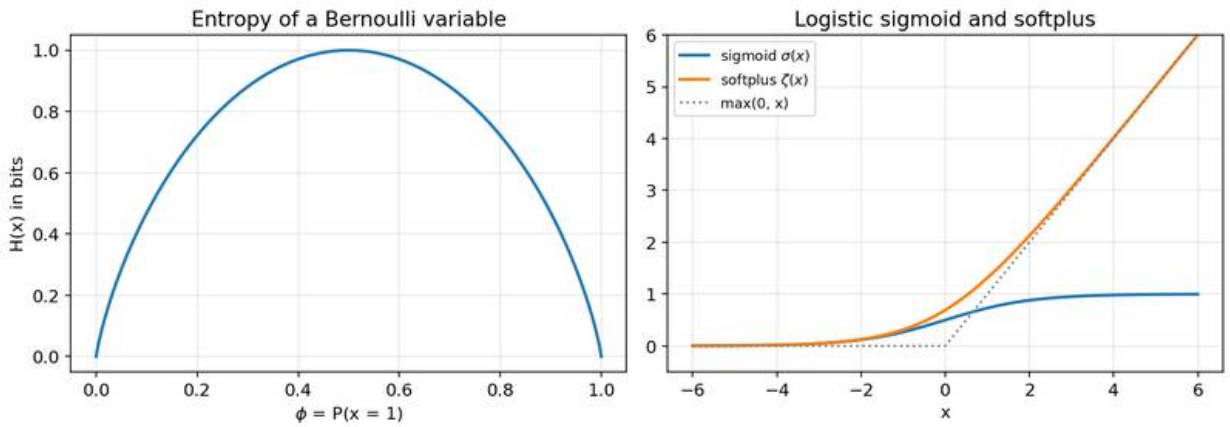


圖 3 — 左：*Bernoulli* 變數的熵，在 $\varphi = 0.5$ 時最大（1 位元），結果確定時為零。右：*logistic sigmoid* $\sigma(x)$ 與 *softplus* $\zeta(x)$，後者是 $max(0, x)$ 的平滑版本。

下列恆等式值得記住，因為每次化簡損失函數時都會用到 [1]：

$$1-\sigma(x)=\sigma(-x), \qquad \frac{d}{dx}\sigma(x)=\sigma(x)(1-\sigma(x)),$$
$$\log\sigma(x)=-\zeta(-x), \qquad \frac{d}{dx}\zeta(x)=\sigma(x),$$
$$\sigma^{-1}(p)=\log\frac{p}{1-p}, \qquad \zeta(x)-\zeta(-x)=x.$$

σ^{-1} 是 **logit**（對數勝算）函數。恆等式 $\log\sigma(x)=-\zeta(-x)$ 說明了為什麼二元交叉熵應該從 logit 直接計算：$-\log\sigma(x)=\zeta(-x)$，而 softplus 可以在不溢位的情況下求值。

```
import numpy as np

def sigmoid(x):
    return 1/(1 + np.exp(-x))

def softplus(x):
    return np.logaddexp(0, x)          # log(1 + exp(x)) without overflow

x = np.linspace(-30, 30, 13)
assert np.allclose(1 - sigmoid(x), sigmoid(-x))
assert np.allclose(np.log(sigmoid(x)), -softplus(-x))
assert np.allclose(softplus(x) - softplus(-x), x)
h = 1e-6
assert np.allclose((softplus(x+h) - softplus(x-h))/(2*h), sigmoid(x), atol=1e-6)
p = sigmoid(x[4:9])
assert np.allclose(np.log(p/(1-p)), x[4:9])   # logit inverts sigmoid
print("naive log(1+exp(1000)):", np.log1p(np.exp(np.float64(1000))))
print("stable softplus(1000):", softplus(1000.0))
```

所有恆等式都通過檢查。直接計算的寫法傳回 `inf`（並出現溢位警告），而 `np.logaddexp` 傳回 1000.0。

貝氏定理

白話版。貝氏定理結合假設對證據的解釋程度，以及假設原本的可信程度。

把乘法規則從兩個方向寫出來，$P(a,b)=P(a\mid b)P(b)=P(b\mid a)P(a)$，於是

$$P(a\mid b)=\frac{P(b\mid a)\,P(a)}{P(b)}, \qquad P(b)=\sum_{a'}P(b\mid a')\,P(a').$$

- $P(a)$ 是**先驗**（prior）：看到證據之前 a 的可信度。
- $P(b\mid a)$ 是在假設 a 下證據 b 的**概似**（likelihood）。
- $P(a\mid b)$ 是**後驗**（posterior）：更新後的可信度。
- $P(b)$ 是**證據**（evidence，或稱邊際概似），對所有假設 a' 加總得到的正規化常數。

實例：基本比率效應

考慮一個假想的瑕疵偵測器，**靈敏度** $P(+ \mid \text{defect}) = 0.9$、**偽陽性率** $P(+ \mid \text{no defect}) = 0.05$，而**盛行率**（基本比率）$P(\text{defect}) = 0.01$。被判為陽性的物品真的有瑕疵的機率是

$$P(\text{defect} \mid +) = \frac{0.9 \cdot 0.01}{0.9 \cdot 0.01 + 0.05 \cdot 0.99} \approx 0.154.$$

即使偵測器能抓到 90% 的瑕疵，警報中只有約 15% 是真的。用數量來想就很清楚：在 10,000 件物品中，有 100 件有瑕疵，其中 90 件觸發警報；但另外 9,900 件正常品中，也有 495 件觸發警報（圖 4 右）。同一個偵測器在 10% 盛行率下則得到 $0.09/(0.09 + 0.045) = 2/3$。這是數學示範，不是任何真實系統的效能宣稱。它的教訓適用於每一個部署中的分類器：**光有靈敏度與偽陽性率，無法決定陽性預測有多可信；部署環境的盛行率同樣關鍵**。 ML 11 在討論分布偏移下的精確率時會再回到這一點。

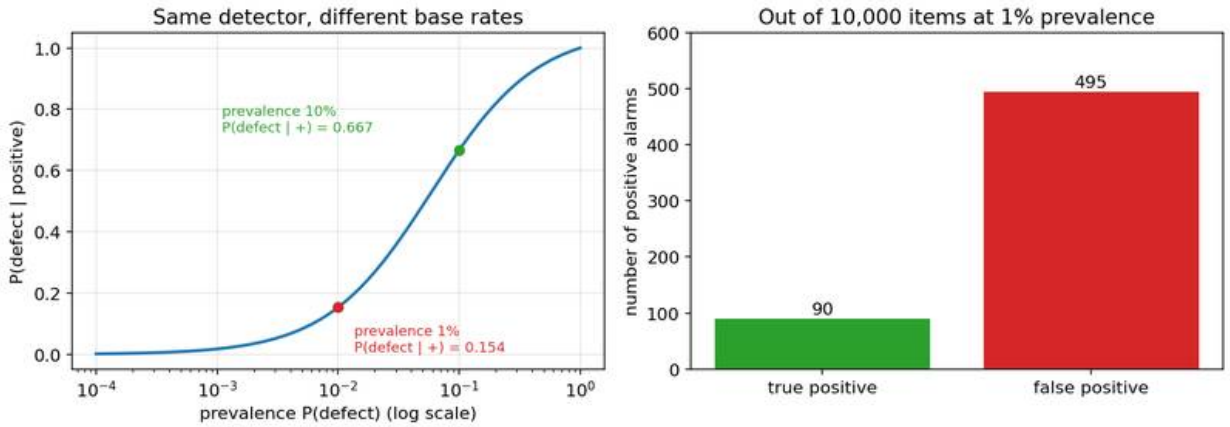


圖 *4* — 假想偵測器（靈敏度 *0.9*、偽陽性率 *0.05*）的基本比率效應。左：後驗 $P(\textit{defect} \mid +)$ 強烈依賴盛行率。右：在 *1%* 盛行率下，假警報比真警報多出五倍以上。

```
import numpy as np

def posterior_positive(prevalence, sensitivity=0.9, false_pos=0.05):
    evidence = sensitivity*prevalence + false_pos*(1 - prevalence)   # P(+)
    return sensitivity*prevalence / evidence                          # P(defect | +)

for prev in [0.001, 0.01, 0.1, 0.5]:
    print(f"prevalence {prev:>5}: P(defect | +) = {posterior_positive(prev):.3f}")

# Simulation check with 1,000,000 synthetic items
rng = np.random.default_rng(0)
defect = rng.random(1_000_000) < 0.01
alarm = np.where(defect, rng.random(defect.size) < 0.9, rng.random(defect.size) < 0.05)
print("simulated:", defect[alarm].mean().round(3))
```

公式對四種盛行率分別給出 0.018、0.154、0.667 與 0.947，而一百萬件物品的模擬重現了 0.154。

貝氏定理也是**貝氏分類器**的基礎：預測後驗 $P(y=k\mid \mathbf{x}) \propto p(\mathbf{x}\mid y=k)P(y=k)$ 最大的類別 k，可以讓期望錯誤率最小。PR 01 會深入發展這套決策理論。

密度的變數變換

白話版。拉伸一張橡皮布，上面的墨水會變淡。隨機變數經過變換後，它的密度也必須依照變換把空間拉伸了多少來重新縮放。

一個誘人的錯誤：若 $y=g(x)$、x 的密度為 p_x，就以為 $p_y(y)=p_x(g^{-1}(y))$。這是錯的，因為它的積分不等於一。對可逆且可微的 g，一維的正確規則是

$$p_y(y) = p_x(g^{-1}(y))\left|\frac{d\,g^{-1}(y)}{dy}\right|.$$

小區間內的機率必須守恆：$|p_y(y)\,dy| = |p_x(x)\,dx|$。在 n 維中，若 $\mathbf{y}=g(\mathbf{x})$，則

$$p_x(\mathbf{x}) = p_y(g(\mathbf{x}))\left|\det \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}\right|,$$

其中 $\partial g/\partial \mathbf{x}$ 是 g 的 $n\times n$ **Jacobian 矩陣**（第 (i,j) 個元素為 $\partial y_i/\partial x_j$），$|\det(\cdot)|$ 是其行列式的絕對值——也就是 g 放大小體積的倍率（行列式見 ML 01A）。這個公式是**正規化流**（normalizing flows）的全部基礎：它堆疊 Jacobian 行列式容易計算的可逆網路，得到精確的概似 [9]。

```
import numpy as np

rng = np.random.default_rng(0)
x = rng.normal(0, 1, 500_000)        # x ~ N(0, 1)
y = np.exp(x)                        # y = g(x), invertible on (0, inf)

def p_x(x):
    return np.exp(-0.5*x**2)/np.sqrt(2*np.pi)

def p_y(y):
    # p_y(y) = p_x(g^{-1}(y)) * |d g^{-1}(y) / dy| = p_x(log y) / y
    return p_x(np.log(y))/y

counts, edges = np.histogram(y, bins=np.linspace(0.05, 4, 80), density=False)
centers = 0.5*(edges[:-1] + edges[1:])
empirical = counts/(y.size*np.diff(edges))
print("max |histogram - formula|:", np.abs(empirical - p_y(centers)).max().round(3))
print("wrong formula (no Jacobian) error:", np.abs(empirical -
 ↪  p_x(np.log(centers))).max().round(3))
```

加上 Jacobian 因子 $1/y$ 後，直方圖與公式的差距在 0.011 以內（抽樣雜訊）；少了它，誤差達到 0.46。

概似、最大概似與 MAP

白話版。固定已觀察到的資料，問哪些參數值最能讓這些資料顯得合理。如果你對參數另有先驗知識，也把它一併考慮進去。

假設模型 $p_{\boldsymbol{\theta}}(\mathbf{x})$ 的參數為 $\boldsymbol{\theta}$，並觀察到由獨立同分布（i.i.d.）樣本組成的資料集 $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$。**概似**為

$$L(\boldsymbol{\theta}) = p_{\boldsymbol{\theta}}(\mathcal{D}) = \prod_{i=1}^{m} p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}).$$

作為 $\boldsymbol{\theta}$ 的函數，它不是參數上的機率分布：它對 $\boldsymbol{\theta}$ 的積分不必等於一。**最大概似估計**（maximum likelihood estimate, MLE）為

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \arg\min_{\boldsymbol{\theta}} \; \mathbb{E}_{\mathbf{x}\sim\hat{p}}\big[-\log p_{\boldsymbol{\theta}}(\mathbf{x})\big].$$

我們取對數，因為許多小於一的數相乘會下溢，而對數相加在數值上很安全，且最大值位置不變。右邊的形式說明 MLE 是在經驗分布 $\hat{p}$ 下**最小化平均負對數概似**（negative log-likelihood, NLL）。對分類器 $p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$，同樣的想法給出條件 NLL $-\log p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$，也就是下面會看到的交叉熵損失。

兩個經典結果：n 次擲硬幣得到 k 次正面時，$\phi_{\mathrm{ML}} = k/n$；對高斯而言，μ_{ML} 是樣本平均，$\sigma^2_{\mathrm{ML}} = \frac{1}{m}\sum_i (x^{(i)} - \mu_{\mathrm{ML}})^2$——這是略帶偏差的估計（改除以 $m-1$ 可以消除偏差）。

MAP 估計

貝氏推論把 $\boldsymbol{\theta}$ 本身視為隨機變數，給它一個**先驗** $p(\boldsymbol{\theta})$，再計算**後驗**

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\mathcal{D})} \;\propto\; p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}).$$

完整的後驗往往難以計算，常見的捷徑是取它的眾數，即**最大後驗**（maximum a posteriori, MAP）估計：

$$\boldsymbol{\theta}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{\theta}} \big[\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\big].$$

若使用高斯先驗 $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \frac{1}{\lambda} I)$，則 $-\log p(\boldsymbol{\theta})$ 等於 $\frac{\lambda}{2}\|\boldsymbol{\theta}\|_2^2$ 加上常數：**權重衰減就是使用高斯先驗的 MAP**。Laplace 先驗則給出 L_1 正則化。這把機率假設和 ML 02 中的正則化項連結起來。但 MAP 點估計仍然只是一個參數向量，無法告訴你參數有多不確定。近似整個後驗是 PR 08 與變分推論 [10] 的主題。

```
import numpy as np

rng = np.random.default_rng(0)
flips = rng.random(10) < 0.7          # 10 coin flips, true P(heads) = 0.7
k, n = flips.sum(), flips.size
```

```
phi = np.linspace(1e-3, 1 - 1e-3, 999)
log_lik = k*np.log(phi) + (n - k)*np.log(1 - phi)
a, b = 5.0, 5.0                          # Beta(5, 5) prior, centered on 0.5
log_prior = (a - 1)*np.log(phi) + (b - 1)*np.log(1 - phi)

print(f"{k} heads in {n} flips")
print("MLE  grid:", phi[log_lik.argmax()].round(3), " closed form:", k/n)
print("MAP  grid:", phi[(log_lik + log_prior).argmax()].round(3),
      " closed form:", round((k + a - 1)/(n + a + b - 2), 3))

# Gaussian MLE: sample mean and the 1/m (biased) variance
x = rng.normal(2.0, 3.0, 50)
print("Gaussian MLE mean, var:", x.mean().round(3), x.var().round(3),
      " unbiased var:", x.var(ddof=1).round(3))
```

在這個隨機種子下，10 次擲出 6 次正面。MLE 為 0.6；Beta(5, 5) 先驗把 MAP 估計拉向 0.5，得到 0.556，與閉合解 $(k+a-1)/(n+a+b-2)$ 一致。資料越多，概似就越占主導，兩種估計會逐漸接近。高斯的 MLE 變異數（7.71）小於不偏估計（7.87），符合預期。

資訊理論

白話版。資訊理論衡量驚訝程度。罕見的事件一旦發生，帶來大量資訊；確定的事件則毫無資訊。把驚訝程度取平均就是熵；使用錯誤分布所多出的驚訝，就是 KL 散度。

Claude Shannon 為了研究透過有雜訊的通道傳送訊息需要多少位元而創立了這個領域 [5]；標準教科書是 Cover 與 Thomas [7]。對機器學習而言，最有用的概念是量化兩個分布有多不同。

自資訊與熵

結果 x 的**自資訊**（self-information）為

$$I(x) = -\log P(x).$$

使用自然對數時單位為 **nat**；使用 $\log_2$ 時單位為**位元**（bit）（1 nat $= 1/\ln 2 \approx 1.443$ 位元）。確定事件（$P = 1$）的資訊量為零；不太可能的事件資訊量很大；而獨立事件的資訊可以相加，因為對數把乘積變成總和。

Shannon 熵是自資訊的期望值：

$$H(P) = \mathbb{E}_{x\sim P}[I(x)] = -\sum_{x} P(x)\log P(x).$$

我們採用 $0\log 0 = 0$ 的約定，理由是 $p \to 0$ 時 $p\log p \to 0$。熵是編碼從 P 抽出之結果所需的最少平均位元數（或 nat 數）。決定性結果的熵為零；K 個結果的均勻分布熵最大，為 $\log K$。參數為 ϕ 的 Bernoulli 變數，$H = -\phi\log\phi - (1-\phi)\log(1-\phi)$，在 $\phi = 0.5$ 時達到 1 位元的峰值（圖 3 左）。

對連續變數，$h(p) = -\int p(x)\log p(x)\,dx$ 稱為**微分熵**。使用時要小心：它可以是負的（$[0, \frac{1}{2}]$ 上的均勻分布 $h = -\log 2$），而且座標尺度一改它就跟著改。它並不是離散位元數的極限版本。

KL 散度

Kullback–Leibler 散度 [6] 衡量分布 Q 與參考分布 P 有多不同：

$$D_{\mathrm{KL}}(P\,\|\,Q) = \mathbb{E}_{x\sim P}\Big[\log \frac{P(x)}{Q(x)}\Big] = \sum_x P(x)\log\frac{P(x)}{Q(x)}.$$

用編碼的語言說，它是：當編碼是為 Q 設計、卻用來編碼 P 的樣本時，額外需要的 nat 數。它的主要性質：

- **非負**：$D_{\mathrm{KL}}(P\|Q) \geq 0$，等號成立若且唯若 $P = Q$（離散情況；連續情況為「幾乎處處」相等）。這來自 Jensen 不等式，因為 $-\log$ 是凸函數。
- **在無支撐處為無限大**：若某個 x 使 $P(x) > 0$ 但 $Q(x) = 0$，散度為無限大。
- **不對稱**：一般而言 $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$，而且不滿足三角不等式。因此 KL 不是距離度量，儘管它常被描述成「分布之間的距離」。

正向與反向 KL

當近似用的分布族太簡單時，這種不對稱會產生看得見的後果。圖 5 用單一高斯 q 去擬合有兩個分得很開之峰的目標 p。

- 最小化**正向 KL** $D_{\mathrm{KL}}(p\|q)$ 時，若 q 在 p 有質量的地方很小，就會受到重罰。因此最佳高斯會**涵蓋兩個峰**，把平均放在兩峰之間（$\mu = 0$、$\sigma \approx 2.62$），並把大量質量放在空蕩的谷底。對高斯 q 而言，正向 KL 的最佳解就是匹配 p 的平均與變異數。
- 最小化**反向 KL** $D_{\mathrm{KL}}(q\|p)$ 時，若 q 在 p 很小的地方很大，就會受到重罰。因此最佳高斯會**鎖定其中一個峰**（$\mu \approx 2.49$、$\sigma \approx 0.81$），忽略另一個。

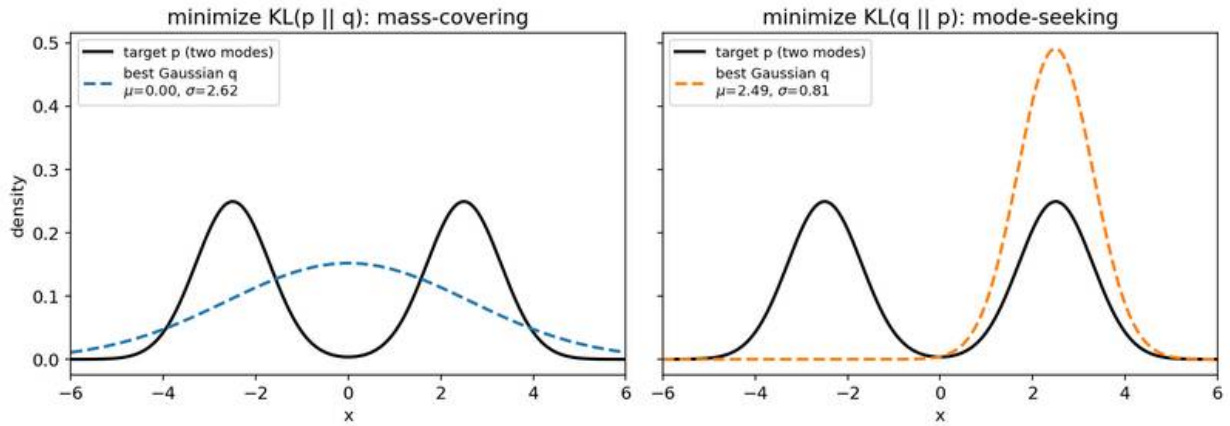


圖 5 — *KL* 不對稱。用單一高斯 q 擬合雙峰目標 p：正向 $KL(p\|q)$ 得到以兩峰中間為中心、涵蓋質量的擬合；反向 $KL(q\|p)$ 得到只落在一個峰上、尋找眾數的擬合（落在另一個峰上是同樣好的鏡像解）。

要最小化哪一個方向，是設計上的選擇。**最大概似最小化的是從資料分布到模型的正向 KL**，因此以概似訓練的模型（自迴歸模型、正規化流、VAE）傾向把機率分散到所有資料上，有時會產

生模糊的樣本。**變分推論通常最小化從近似分布到真實後驗的反向 KL** [10]，因此傾向低估後驗的不確定性。

交叉熵

Q 相對於 P 的**交叉熵**為

$$H(P,Q) = -\mathbb{E}_{x\sim P}[\log Q(x)] = H(P) + D_{\mathrm{KL}}(P\,\|\,Q).$$

因為 $H(P)$ 與 Q 無關，對 Q 最小化交叉熵就等於最小化正向 KL。在分類中，P 是經驗標籤分布——每個樣本是一個 one-hot 向量——而 Q 是模型的 softmax 輸出 $\hat{\mathbf{y}}$。對真實類別為 y 的單一樣本，

$$H(\mathbf{e}_y, \hat{\mathbf{y}}) = -\sum_k [\mathbf{e}_y]_k \log \hat{y}_k = -\log \hat{y}_y,$$

其中 $\mathbf{e}_y$ 是類別 y 的 one-hot 向量。對整個資料集取平均，這正是條件負對數概似。所以「交叉熵損失」、「對數損失」與「類別模型的負對數概似」是同一個目標的三個名字。任何由負對數概似構成的損失，都是經驗分布與模型之間的交叉熵——這也是為什麼均方誤差同樣是高斯模型的「交叉熵」[1]。

互資訊

互資訊衡量知道一個變數能減少多少對另一個變數的不確定性：

$$I(X;Y) = D_{\mathrm{KL}}(p(x,y)\,\|\,p(x)p(y)) = H(X) - H(X\mid Y) = H(X) + H(Y) - H(X,Y).$$

其中 $H(X\mid Y) = -\mathbb{E}_{x,y}[\log p(x\mid y)]$ 是條件熵，$H(X,Y)$ 是聯合熵。$I(X;Y) \geq 0$，等號成立若且唯若 X 與 Y 獨立——和共變異數不同，它能偵測任何形式的相依。它對 X 與 Y 是對稱的。

程式碼：資訊量

```
import numpy as np

p = np.array([.7, .2, .1])
q = np.array([.5, .3, .2])
entropy = -(p*np.log(p)).sum()
cross_entropy = -(p*np.log(q)).sum()
kl = (p*np.log(p/q)).sum()
reverse_kl = (q*np.log(q/p)).sum()
assert np.allclose(cross_entropy - entropy, kl) and kl >= 0
print("H(p), H(p,q), KL(p||q), KL(q||p) in nats:",
      np.round([entropy, cross_entropy, kl, reverse_kl], 4))
print("H(p) in bits:", (entropy/np.log(2)).round(4))

# Mutual information from a joint table (same weather/umbrella table)
P = np.array([[0.56, 0.14],
              [0.06, 0.24]])
```

```
px, py = P.sum(1), P.sum(0)
mi = (P*np.log(P/np.outer(px, py))).sum()
H = lambda t: -(t[t > 0]*np.log(t[t > 0])).sum()
assert np.isclose(mi, H(px) + H(py) - H(P.ravel()))
print("I(weather; umbrella) =", mi.round(4), "nats")

# Cross-entropy loss on one-hot labels equals mean -log q(y)
probs = np.array([[0.7, 0.2, 0.1], [0.1, 0.8, 0.1], [0.3, 0.3, 0.4]])
labels = np.array([0, 1, 2])
print("mean cross-entropy:", (-np.log(probs[np.arange(3), labels])).mean().round(4))
```

輸出：$H(p) = 0.8018$、$H(p,q) = 0.8869$、$D_{\mathrm{KL}}(p\|q) = 0.0851$、$D_{\mathrm{KL}}(q\|p) = 0.0920$ nats——兩個方向的值不同。$H(p)$ 為 1.1568 位元。天氣與帶傘的互資訊為 0.1637 nats，兩種公式算出的結果一致。三筆預測的平均交叉熵為 0.4987 nats。在真正的訓練程式中，請用融合函數直接從 logit 計算交叉熵，而不是先算機率（見 ML 01C）。

僅線上版：*Shannon*：通訊的數學理論 — *https://blog.shyandram.dev/zh-tw/learn/ml/probability-information/papers/shannon 通訊的數學理論/*

結構化機率模型

白話版。圖模型利用條件獨立結構，把聯合分布拆成較小部分；圖本身不足以確立因果。

100 個二元變數的聯合機率表需要 $2^{100} - 1$ 個數字。**結構化機率模型**，又稱**圖模型**（graphical models），利用條件獨立把聯合分布分解成小因子 [1], [3]。主要有兩類。

有向模型（貝氏網路）使用有向無環圖 $\mathcal{G}$。每個變數只依賴它的父節點 $Pa_{\mathcal{G}}(x_i)$：

$$p(\mathbf{x}) = \prod_i p(x_i \mid Pa_{\mathcal{G}}(x_i)).$$

例如在 $a \to b \to c$ 且 $a \to c$ 時，$p(a,b,c) = p(a)\,p(b \mid a)\,p(c \mid a,b)$；刪掉邊 $a \to c$ 就是宣稱 $c \perp a \mid b$，最後一個因子縮短為 $p(c \mid b)$。有向圖很適合描述生成故事（「先選類別，再產生像素」），而混合模型 $z \to \mathbf{x}$ 是最簡單的例子。

無向模型（馬可夫隨機場）使用無向圖。每個**團**（clique）$C^{(i)}$（彼此全部相連的節點集合）有一個非負的**因子** $\phi^{(i)}$，且

$$p(\mathbf{x}) = \frac{1}{Z} \prod_i \phi^{(i)}(C^{(i)}), \qquad Z = \sum_{\mathbf{x}} \prod_i \phi^{(i)}(C^{(i)}).$$

配分函數 Z 用來正規化乘積；計算它往往是最困難的部分。無向模型適合對稱的交互作用，例如影像先驗中相鄰像素的關係，或用於分割的條件隨機場。

圖是對分布的一種描述，不是另一種分布：任何分布都可以用兩種方式寫出，只是其中一種可能精簡得多。深度學習常隱含地使用這些概念——自迴歸模型是全連接的有向模型，VAE 則是條件分布由神經網路表示的雙節點有向模型 $\mathbf{z} \to \mathbf{x}$ [8]。

現代觀點

本篇的機率與資訊理論是古典內容。深度學習時代改變的是它如何被使用，以及實務上會在哪裡出錯。本節綜覽相關綜述文獻與近年方向；只整理它們的框架，不轉述基準數字。

校準：預測機率誠實嗎？

如果分類器在所有預測信心為 0.8 的輸入中，大約有 80% 是正確的，就稱它是**校準良好**的（calibrated）。Guo 等人 [11]（ICML 2017）發現現代深度網路常常**過度自信**，並指出深度、寬度、權重衰減與批次正規化都會影響校準。他們主要推薦的**溫度縮放**（temperature scaling）——把所有 logit 除以一個在驗證集上擬合的純量 T——是 Platt scaling 的單參數版本；因為它只改變信心、不改變預測類別，至今仍是強力的基準方法。Wang 的綜述 [12] 把校準方法分為四大類：事後校準、訓練期間的正則化、不確定性估計，以及這些方法的組合；其修訂版也涵蓋大型語言模型的校準。從機率的角度看，教訓是：softmax 輸出是 $P(y \mid \mathbf{x})$ 的一個模型，和任何模型一樣，它的不確定性也可能是錯的，而不只是 argmax 會錯。

不確定性量化

兩篇主要綜述描繪了深度學習中不確定性研究的全貌。Abdar 等人 [13]（*Information Fusion*，2021）回顧了貝氏近似、集成方法以及它們在許多領域的應用。Gawlikowski 等人 [14]（*Artificial Intelligence Review*，2023）區分了**可降低的模型（認知，epistemic）不確定性**與**不可降低的資料（偶然，aleatoric）不確定性**——分別對應本篇開頭所說的第三與第一種不確定性來源——並比較了確定性單一網路、貝氏方法、集成與測試時資料擴增，以及校準的量測方式。

有兩篇原始研究特別有影響力。**深度集成**（deep ensembles）[15] 從不同的隨機初始化訓練多個網路，再平均它們的預測分布；這個簡單、可平行化的方法與近似貝氏神經網路表現相當，並且在分布外（out-of-distribution）輸入上表現出較高的不確定性。Ovadia 等人 [16]（NeurIPS 2019）在**資料集偏移**下評測不確定性方法，發現以分布內驗證資料做的事後校準難以延續，而對多個模型做邊際化的方法（如集成）表現較穩定。

另一種假設較少的互補方法是**共形預測**（conformal prediction）。Angelopoulos 與 Bates [17] 提供了一份淺顯的入門：利用一組保留的校準資料，共形方法能把任何預訓練模型的分數轉成預測集合，並以使用者指定的機率包含真實標籤，前提是校準資料與測試資料可交換（exchangeable）。它不要求模型機率本身已經校準，因此對被當成黑盒子使用的視覺與語言系統很有吸引力。

表示學習中的互資訊

互資訊曾是自監督學習（ML 06）中熱門的目標。對比預測編碼（Contrastive Predictive Coding）[18] 提出了 **InfoNCE** 損失：它在多個負樣本中辨認出正樣本對，可以證明是在最大化互資訊的一個下界。Poole 等人 [19]（ICML 2019）統一了許多變分互資訊界，並指出當真實互資訊很大時這些界會退化，在高偏差與高變異數之間取捨。Tschannen 等人 [20]（ICLR 2020）主張這些方法的成功「不能只歸因於互資訊本身的性質」：編碼器架構與估計器的參數化至少同樣重要。現在的共識是把 InfoNCE 類的目標視為有資訊理論動機、行為良好的對比損失，而不是精確的互資訊估計器。

生成模型中的散度與潛在變數

生成建模就是應用機率。VAE [8] 最大化證據下界（evidence lower bound），它結合了重建概似與一個 KL 項；Blei 等人 [10] 回顧了變分推論，即在一個容易處理的分布族中最小化（反向）KL 來近似難解的後驗。正規化流 [9] 完全建立在變數變換公式上。擴散模型與流匹配模型（ML 08）結合了高斯雜訊、條件期望值與概似界。選擇哪一種散度（正向 KL、反向 KL 或其他）決定了模型是涵蓋全部資料，還是集中在少數眾數上——與圖 5 是同一幅圖像，只是規模大得多。

尚待解決的問題

- **分布偏移下的校準**。在驗證集上成立的校準，換到新的相機、醫院或提示詞時可能失效 [16]；如何偵測這種情況仍是開放問題。
- **區分不確定性類型**。把預測不確定性拆成認知與偶然兩部分，在概念上很清楚，但對大型網路很難可靠地做到 [14]。
- **高維資訊量的估計**。變分互資訊界有根本的偏差–變異數限制 [19]，因此對深度表示做資訊理論宣稱時需要格外謹慎。
- **基礎模型的不確定性**。把校準與共形方法應用到輸出開放式內容的大型視覺–語言模型與語言模型，是活躍的研究方向 [12], [17]。

重點整理

- 離散機率總和為一；連續密度積分為一，單點密度可以大於一。
- 邊際化與連鎖法則是恆等關係，獨立性則是額外假設。
- 共變異數為零，通常不足以推得獨立。
- 概似與先驗假設決定損失及正則化，應明確交代尺度參數。
- 貝氏定理以證據更新先驗信念；密度變換還需要體積修正。
- 交叉熵可拆成熵與 KL；KL 不對稱，涵蓋或集中於模式的行為取決於近似家族及問題。

練習

1. **單位與確定性**。 (a) 把 1 nat 的損失換算成位元。(b) 分別計算公平硬幣、確定結果與公平八面骰的熵（以位元為單位）。

   提示

   (a) 除以 $\ln 2$：約 1.443 位元。(b) 1 位元、0 位元與 $\log_2 8 = 3$ 位元。K 個結果的均勻分布熵為 $\log K$，是 K 個結果所能達到的最大值。

2. **盛行率偏移**。使用貝氏定理一節中的偵測器，找出讓陽性結果恰好像擲硬幣、即 $P(\text{defect} \mid +) = 0.5$ 的盛行率。

   提示

令盛行率為 π，解 $0.9\pi = 0.05(1-\pi)$，得 $\pi = 0.05/0.95 \approx 0.053$，約 5.3%。低於這個值時，大多數警報都是假的。

3. **相依但不相關**。令 X 以相同機率取 $-1, 0, 1$，且 $Y = X^2$。證明 $\mathrm{Cov}(X, Y) = 0$，並以位元計算 $I(X; Y)$。

   提示

   $\mathbb{E}[X] = 0$ 且 $\mathbb{E}[XY] = \mathbb{E}[X^3] = 0$，所以共變異數為零。因為 Y 是 X 的函數，$H(Y \mid X) = 0$，於是 $I(X;Y) = H(Y)$。$Y = 1$ 的機率為 $2/3$、$Y = 0$ 的機率為 $1/3$，所以 $I \approx 0.918$ 位元。

4. **高斯 MLE 等於平方誤差**。證明在 σ 固定時，對 $\boldsymbol{\theta}$ 最大化 $\prod_i \mathcal{N}(y^{(i)}; f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), \sigma^2)$ 等同於最小化均方誤差。如果 σ 也由網路預測，會有什麼改變？

   提示

   負對數概似為 $\sum_i \frac{(y^{(i)} - f_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^2}{2\sigma^2} + m \log \sigma + \text{const}$。$\sigma$ 固定時，只有平方誤差與 $\boldsymbol{\theta}$ 有關。如果網路也預測 $\sigma(\mathbf{x})$（例如透過 softplus），每個樣本的誤差會被 $1/\sigma^2$ 加權，而 $\log \sigma$ 懲罰項會防止變異數無限制地變大——這就是異質變異（heteroscedastic）迴歸模型。

5. **變數變換**。若 $X \sim \mathcal{U}(0, 1)$（均勻分布）且 $Y = -\frac{1}{\lambda} \log X$，$\lambda > 0$，$Y$ 的密度是什麼？用 NumPy 抽樣驗證你的答案。

   提示

   $g^{-1}(y) = e^{-\lambda y}$ 且 $|dg^{-1}/dy| = \lambda e^{-\lambda y}$，而在 $(0, 1)$ 上 $p_X = 1$。所以當 $y \geq 0$ 時 $p_Y(y) = \lambda e^{-\lambda y}$：指數分布。這就是反轉換抽樣（inverse-transform sampling）的技巧。

6. **KL 的方向**。對 $p = (0.5, 0.5, 0)$ 與 $q = (0.4, 0.4, 0.2)$，計算 $D_{\mathrm{KL}}(p \| q)$ 與 $D_{\mathrm{KL}}(q \| p)$。哪一個是無限大？這對以最大概似訓練模型有什麼啟示？

   提示

   $D_{\mathrm{KL}}(p \| q) = \log(0.5/0.4) \approx 0.223$ nats（第三項是 $0 \log 0 = 0$）。$D_{\mathrm{KL}}(q \| p) = \infty$，因為 q 在 p 沒有質量的地方放了質量。最大概似最小化的是 $D_{\mathrm{KL}}(p_{\text{data}} \| q_{\text{model}})$；只要模型對任何一筆觀察到的樣本給出零機率，它就會變成無限大——所以以概似訓練的模型必須在資料可能出現的每個地方都分配一些機率。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 3 “Probability and Information Theory”. Book
2. C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024. Book
3. K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022. Book

4. M. P. Deisenroth, A. A. Faisal and C. S. Ong, *Mathematics for Machine Learning*, Cambridge University Press, 2020. Book
5. C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, 1948. DOI
6. S. Kullback and R. A. Leibler, “On Information and Sufficiency,” *The Annals of Mathematical Statistics*, vol. 22, pp. 79–86, 1951. DOI
7. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley, 2006. DOI
8. D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” in *Proc. International Conference on Learning Representations (ICLR)*, 2014. arXiv
9. G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed and B. Lakshminarayanan, “Normalizing Flows for Probabilistic Modeling and Inference,” *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021. arXiv
10. D. M. Blei, A. Kucukelbir and J. D. McAuliffe, “Variational Inference: A Review for Statisticians,” *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017. DOI
11. C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, “On Calibration of Modern Neural Networks,” *ICML*, 2017. arXiv
12. C. Wang, “Calibration in Deep Learning: A Survey of the State-of-the-Art,” arXiv:2308.01222, 2023 (revised 2025). arXiv
13. M. Abdar et al., “A Review of Uncertainty Quantification in Deep Learning: Techniques, Applications and Challenges,” *Information Fusion*, vol. 76, pp. 243–297, 2021. DOI
14. J. Gawlikowski et al., “A Survey of Uncertainty in Deep Neural Networks,” *Artificial Intelligence Review*, vol. 56, pp. 1513–1589, 2023. DOI
15. B. Lakshminarayanan, A. Pritzel and C. Blundell, “Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles,” *NeurIPS (NIPS)*, 2017. arXiv
16. Y. Ovadia et al., “Can You Trust Your Model’s Uncertainty? Evaluating Predictive Uncertainty Under Dataset Shift,” *NeurIPS*, 2019. arXiv
17. A. N. Angelopoulos and S. Bates, “A Gentle Introduction to Conformal Prediction and Distribution-Free Uncertainty Quantification,” arXiv:2107.07511, 2021. arXiv
18. A. van den Oord, Y. Li and O. Vinyals, “Representation Learning with Contrastive Predictive Coding,” arXiv:1807.03748, 2018. arXiv
19. B. Poole, S. Ozair, A. van den Oord, A. A. Alemi and G. Tucker, “On Variational Bounds of Mutual Information,” *ICML*, 2019. arXiv
20. M. Tschannen, J. Djolonga, P. K. Rubenstein, S. Gelly and M. Lucic, “On Mutual Information Maximization for Representation Learning,” *ICLR*, 2020. arXiv

ML 01C・數值計算與梯度檢查

浮點數、溢位與下溢、穩定的 softmax、條件數、梯度、Jacobian 與 Hessian、牛頓類方法、KKT 條件與混合精度：讓數學模型在真實計算中站得住腳。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/numerical-computation/

你將學到

- 區分浮點數的範圍與精度。
- 改寫損失與正規化公式，避免數值失敗。
- 區分問題的條件性與演算法的穩定性。
- 用梯度與曲率理解最佳化及限制條件。
- 檢查導數，並驗證混合精度運算。

先看全貌

機器學習用實數的語言寫成，但電腦沒有真正的實數。它只有有限個浮點數，每一次算術運算都會把結果捨入到其中一個。大多數時候這種捨入看不見；有時卻是災難：指數溢位成無限大、極小的機率變成剛好 0 而它的對數變成負無限大，或是兩個幾乎相等的數相減，絕大部分有效位數就此消失。

故事的另一半是最佳化。訓練模型就是最小化一個有數百萬甚至數十億個參數的函數，而我們只能用數值方法做到：依照導數的指引走很多小步。這些步伐前進得多快、多可靠，取決於損失曲面的形狀（斜率與曲率），也取決於我們怎麼走。本篇以 Goodfellow、Bengio 與 Courville 的「Numerical Computation」一章 [1] 為骨架，補上現代低精度訓練所依賴的浮點細節，並用小型 NumPy 實驗串起每個概念。

> **用最簡單的話說**
>
> 想像一台只能顯示七位數的計算機。你問它一百萬加 0.1，它顯示一百萬：0.1 從螢幕上掉下去了。接著再減掉一百萬，你得到 0 而不是 0.1。好的數值程式會安排計算步驟，讓重要的位數永遠不會掉出螢幕。最佳化則是另一種遊戲：你蒙著眼站在起伏的田野上，想走到最低點。你能用腳感覺坡度（梯度），多花點力氣還能感覺地面怎麼彎（Hessian）。數值計算要做的，就是在計算機不騙你的前提下，安全地走下山。

如果 softmax 與交叉熵對你還很陌生，請先讀 ML 01B；特徵值與奇異值分解則可隨時回頭參考 ML 01A。之後的 ML 02 會用到本篇的所有內容來訓練神經網路。

浮點數

白話版。浮點數儲存符號、有效數與指數；正規數的相對間距大致固定，次正規範圍接近零時則惡化。

浮點數如何儲存

IEEE 754 標準 [3] 定義了幾乎所有 CPU 與 GPU 使用的二進位浮點格式。一個正規（非特殊值）數字儲存為

$$x = (-1)^s \times \left(1 + \sum_{i=1}^{t} b_i\, 2^{-i}\right) \times 2^{e},$$

其中 $s \in \{0,1\}$ 是正負號位元，$b_1, \dots, b_t \in \{0,1\}$ 是有效數（significand）中儲存的 t 個小數位元（開頭的 1 是隱含的），e 是限制在固定範圍內的整數指數。最重要的兩個數：

- **機器 epsilon** $\varepsilon_{\text{mach}} = 2^{-t}$：1 與下一個可表示的較大數之間的間距。
- **單位捨入誤差**（unit roundoff）$u = \varepsilon_{\text{mach}}/2$：「捨入到最近值」的相對誤差上界。對正規範圍內的任意實數 x，儲存值為 $\mathrm{fl}(x) = x(1+\delta)$，且 $|\delta| \le u$。

標準也要求每個基本運算（$+, -, \times, \div, \sqrt{\cdot}$）都像先精確計算再捨入一次那樣：$\mathrm{fl}(a \circ b) = (a \circ b)(1+\delta)$，$|\delta| \le u$。這條規則就是捨入誤差分析的基礎 [2]、[4]。

比最小正規數更小的範圍，格式改用**次正規數**（subnormal），犧牲前導精度、讓數值逐步降到 0。特殊值則補足整體：結果太大時為 $\pm\infty$；無效運算如 $0/0$、$\infty - \infty$、$\infty \times 0$ 則得到 **NaN**（not a number）。NaN 一旦出現，幾乎所有碰到它的運算都會回傳 NaN，這就是為什麼單一個壞值就能毀掉整次訓練。

你會遇到的四種格式

下表以 `numpy.finfo` 計算（bfloat16 因 NumPy 不支援，改用 `torch.finfo`）。「位元」欄列出正負號／指數／小數位元數。

格式	位元	$\varepsilon_{\text{mach}}$	最大有限值	最小正規數	最小次正規數
float16（IEEE 半精度）	1/5/10	9.77×10^{-4}	65504	6.10×10^{-5}	5.96×10^{-8}
bfloat16	1/8/7	7.81×10^{-3}	3.39×10^{38}	1.18×10^{-38}	9.18×10^{-41}
float32（單精度）	1/8/23	1.19×10^{-7}	3.40×10^{38}	1.18×10^{-38}	1.40×10^{-45}
float64（雙精度）	1/11/52	2.22×10^{-16}	1.80×10^{308}	2.23×10^{-308}	4.94×10^{-324}

這張表要從兩個方向讀。**指數位元**決定範圍：float16 超過 65504 就溢位，而 bfloat16 保留了 float32 的 8 個指數位元，所以範圍與 float32 相同。**小數位元**決定精度：float32 約有 7 位有效十進位數字，float64 約 16 位，float16 約 3 位，bfloat16 只有約 2 位。bfloat16「範圍寬、精度粗」與 float16「範圍窄、精度較細」的取捨，正是兩種半精度格式需要不同訓練做法的原因（見後文混合精度一節與 [5]、[12]）。

```
import numpy as np

for dtype in (np.float16, np.float32, np.float64):
    info = np.finfo(dtype)
    print(f"{info.dtype}: eps={float(info.eps):.3e}  max={float(info.max):.3e}  "
```

```
          f"tiny={float(info.tiny):.3e}  subnormal={float(info.smallest_subnormal):.3e}")

# Rounding: 1 + eps/2 rounds back to 1 in float32
e = np.finfo(np.float32).eps
print(np.float32(1) + np.float32(e/2) == np.float32(1))   # True
print(np.float32(1) + np.float32(e) == np.float32(1))     # False

# Addition is not associative
a, b, c = np.float32(1e8), np.float32(-1e8), np.float32(1.0)
print((a + b) + c, a + (b + c))                           # 1.0 0.0
```

最後一行值得停下來想。精確算術下兩種順序都得到 1；但在 float32 中，10^8 附近相鄰浮點數的間距是 8，所以 $-10^8 + 1$ 會捨入回 -10^8，那個 1 就不見了。實務上的後果是：用不同順序加總同一組數字（例如換了 GPU 數量，或換了歸約〔reduction〕核心），最後幾位數就可能不同。這是每次執行結果不完全一致的主要來源之一，而且不是你程式的錯 [13]。

溢位、下溢與消去

白話版。溢位是結果大到存不下；下溢是結果小到變成 0（或以次正規數形式失去精度）。消去（cancellation）是兩個幾乎相等的數相減：捨入誤差留了下來，真正的訊號卻被丟掉。

溢位與下溢

在機器學習中，最常見的元兇是指數函數。只要 $z > \ln(3.40\times10^{38}) \approx 88.7$，$\exp(z)$ 就超過 float32 的最大值，因此 logit 為 100 時在 float32 中會溢位（在 float16 中，logit 約 11.1 就會溢位，因為 $\ln 65504 \approx 11.09$）。另一方向，$\exp(-110) \approx 1.7 \times 10^{-48}$ 比 float32 最小的次正規數還小，結果剛好是 0。這兩種情況預設都不會報錯，只會產生 `inf` 與 `0`，而下一個運算就把它們變成 `nan` 或 $-\infty$。

```
import numpy as np

with np.errstate(over="ignore", under="ignore", invalid="ignore"):
    print(np.exp(np.float32(100.0)))        # inf: overflow
    print(np.exp(np.float32(-110.0)))       # 0.0: underflow
    big = np.exp(np.float32(100.0))
    print(big / big)                        # nan: inf / inf

# Catastrophic cancellation and its cure
x = 1e-10
print(np.log(1 + x), np.log1p(x))           # roundoff vs accurate
print(np.exp(x) - 1, np.expm1(x))

# Variance: the one-pass formula cancels badly when the mean is large
rng = np.random.default_rng(0)
data = (1e4 + rng.normal(0, 1, 100_000)).astype(np.float32)
one_pass = np.mean(data**2) - np.mean(data)**2
two_pass = np.mean((data - data.mean())**2)
print("one-pass:", one_pass, " two-pass:", two_pass)
```

在這台機器上，單趟（one-pass）變異數得到 −8.0——負的變異數——而雙趟（two-pass）公式給出正確的約 1.0。資料平均值為 10^4、變異數為 1，所以 $\mathbb{E}[X^2] \approx 10^8$、$(\mathbb{E}X)^2 \approx 10^8$；在這個量級上 float32 只能分辨約 8 的差異，真正的差值 1 完全淹沒在雜訊裡。

為什麼減法是危險的運算

假設 $\hat{a} = a(1+\delta_a)$、$\hat{b} = b(1+\delta_b)$ 的儲存相對誤差滿足 $|\delta_a|, |\delta_b| \le u$。即使減法本身是精確的，

$$\frac{|(\hat{a}-\hat{b}) - (a-b)|}{|a-b|} = \frac{|a\delta_a - b\delta_b|}{|a-b|} \le u\,\frac{|a|+|b|}{|a-b|}.$$

其中 a, b 是真實值，u 是單位捨入誤差。當 $a \approx b$ 時，因子 $(|a|+|b|)/|a-b|$ 非常大：減法並沒有製造誤差，而是把兩數共有的位數消掉，暴露出輸入原本就帶有的誤差。解法是代數上的：改寫公式，讓這個減法根本不發生。你會用到的例子：

- `log1p(x)` 計算 $\log(1+x)$、`expm1(x)` 計算 $e^x - 1$，在 x 很小時都很精確，天真寫法則幾乎失去所有位數（見上方輸出）。
- 變異數：先減去平均值（雙趟），或使用 Welford 的串流更新。
- Softmax 與 log-sum-exp：先減去最大值，下一節說明。
- 距離：$\|x-y\|^2 = \|x\|^2 + \|y\|^2 - 2x^\top y$ 很快，但在 $x \approx y$ 時嚴重消去；需要精確時，把結果截在 0 以上，或直接計算 $x - y$。

穩定的 softmax 與 log-sum-exp

白話版。取指數前先減去一個共同常數；對數機率要直接計算，而不是先產生極小的機率再取對數。

Softmax 把 logit 向量 $z \in \mathbb{R}^K$ 轉成機率：

$$\operatorname{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \qquad i = 1, \dots, K.$$

把同一個常數 c 加到每個 logit 上，softmax 不變，因為因子 $\exp(c)$ 在分子與分母間抵銷。取 $c = -m$，其中 $m = \max_i z_i$：

$$p_i = \frac{\exp(z_i - m)}{\sum_j \exp(z_j - m)}.$$

現在每個指數都 ≤ 0，不會溢位；最大的一項剛好是 $\exp(0) = 1$，所以分母至少為 1，也不會下溢成 0。同樣的技巧也能穩定 **log-sum-exp** 函數：

$$\operatorname{LSE}(z) = \log \sum_j \exp(z_j) = m + \log \sum_j \exp(z_j - m), \qquad \log p_i = z_i - \operatorname{LSE}(z).$$

LSE 是最大值的平滑版本：$\max_i z_i \le \operatorname{LSE}(z) \le \max_i z_i + \log K$；它的梯度恰好就是 softmax，$\partial \operatorname{LSE} / \partial z_i = p_i$。

對目標類別 y，交叉熵損失為 $-\log p_y = \mathrm{LSE}(z) - z_y$，它對 logit 的梯度是那個著名的簡單形式 $p - e_y$，其中 e_y 是目標的 one-hot 向量。先算 `softmax` 再取 `log` 仍可能失敗：若 p_y 下溢成 0，它的對數就是 $-\infty$，即使 $\mathrm{LSE}(z) - z_y$ 是個完全正常的有限數。這就是框架提供接受 logit 的融合函數 `log_softmax` 與 `cross_entropy` 的原因。

程式：會讓天真指數運算溢位的 logit

這個獨立的 NumPy 範例檢查穩定的機率、對數機率，以及平移 logit 的不變性。

```
import numpy as np

def logsumexp(z):
    m = np.max(z)
    return m + np.log(np.exp(z-m).sum())
def logsoftmax(z):
    return z-logsumexp(z)

z = np.array([1000., 1001., 999.], dtype=np.float64)
logp = logsoftmax(z)
p = np.exp(logp)
assert np.allclose(p.sum(), 1)
assert np.allclose(logsoftmax(z+12345), logp)
print("probabilities:", p)
print("cross-entropy for class 1:", -logp[1])
with np.errstate(over="ignore", invalid="ignore"):
    naive = np.exp(z) / np.exp(z).sum()
print("naive softmax:", naive)
```

穩定版輸出機率 $[0.245, 0.665, 0.090]$，類別 1 的交叉熵為 0.408 nats；天真版輸出 `[nan nan nan]`，因為 $\exp(1000)$ 即使在 float64 也會溢位（float64 的上限是 $\exp(709.78)$）。

穩定的 logistic（二元交叉熵）損失

二元分類以另一種形式遇到同樣的問題。設 logit 為 z、標籤 $y \in \{0, 1\}$、sigmoid 為 $\sigma(z) = 1/(1+e^{-z})$，損失 $-y \log \sigma(z) - (1-y)\log(1-\sigma(z))$ 可化簡為

$$\ell(z, y) = \mathrm{softplus}(z) - yz, \qquad \mathrm{softplus}(z) = \log(1 + e^{z}) = \max(z, 0) + \log\!\left(1 + e^{-|z|}\right).$$

右邊的寫法從不對正數取指數，`log1p` 則讓小修正項保持精確。請優先使用 `binary_cross_entropy_with_logits`，不要自己先算 sigmoid 再取對數。

```
import numpy as np

def bce_naive(z, y):
    s = 1 / (1 + np.exp(-z))
    return -(y*np.log(s) + (1-y)*np.log(1-s))
```

```
def bce_stable(z, y):
    # softplus(z) - y*z, with softplus(z) = max(z,0) + log1p(exp(-|z|))
    return np.maximum(z, 0) + np.log1p(np.exp(-np.abs(z))) - y*z

z = np.array([-800., -30., 0., 30., 800.])
y = np.array([1., 1., 0., 0., 0.])
with np.errstate(over="ignore", divide="ignore", invalid="ignore"):
    print("naive :", bce_naive(z, y))
print("stable:", bce_stable(z, y))
```

天真版對兩個「信心十足的錯誤」（$z = \pm 800$）回傳 `inf`，在 $z = 30$ 時回傳 30.001 而非 30.000，因為 $1 - \sigma(30)$ 是靠消去算出來的。穩定版回傳 800、30、$\log 2$、30、800，全部正確。

條件不良

白話版。有些問題天生敏感：輸入的微小改變會造成答案的巨大改變，不論你算得多小心。**條件數**（condition number）就是衡量這種敏感度的量。

線性系統的條件數

考慮對可逆方陣 A 解 $Ax = b$。若資料變成 $b + \delta b$，解變成 $x + \delta x$，則

$$\frac{\|\delta x\|}{\|x\|} \le \kappa(A)\,\frac{\|\delta b\|}{\|b\|}, \qquad \kappa(A) = \|A\|\,\|A^{-1}\| = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \ \text{(2-norm)}.$$

其中 $\|\cdot\|$ 是歐氏向量範數及其誘導矩陣範數，$\sigma_{\max}, \sigma_{\min}$ 是 A 的最大與最小奇異值。對稱正定矩陣的奇異值等於特徵值，因此 $\kappa = \lambda_{\max}/\lambda_{\min}$。一般可逆矩陣不能只靠特徵值比值判斷條件數；應使用奇異值，如 NumPy 的預設 `cond` 計算方式 [23]。大條件數表示誤差可能被強烈放大，實際誤差仍取決於擾動方向 [4]。

```
import numpy as np

A = np.array([[1.0, 1.0],
              [1.0, 1.0001]])
print("condition number:", np.linalg.cond(A))

b = np.array([2.0, 2.0001])
x = np.linalg.solve(A, b)
db = np.array([0.0, 0.0001])                # tiny change in the data
x2 = np.linalg.solve(A, b + db)
rel_in = np.linalg.norm(db) / np.linalg.norm(b)
rel_out = np.linalg.norm(x2 - x) / np.linalg.norm(x)
print("x:", x, " x after perturbation:", x2)
print(f"relative change in b: {rel_in:.1e}, in x: {rel_out:.1e}, "
      f"amplification: {rel_out/rel_in:.0f}")
```

A 的兩欄幾乎相同，$\kappa(A) \approx 4\times10^4$；$b$ 只有 3.5×10^{-5} 的相對改變，解卻從 $(1,1)$ 移到 $(0,2)$——百分之百的改變，放大約 28,000 倍，仍在 κ 的上界之內。沒有任何計算出錯，是這個問題本身條件不良。

條件不良的問題 vs. 不穩定的演算法

請把兩個概念分開：

- **條件不良的問題**會放大輸入的小變化。任何演算法都無法修正；你需要更好的資料、正則化（它改變了問題）或重新表述問題。
- **不穩定的演算法**會替一個可能條件良好的問題加入可避免的誤差。天真的 softmax 與單趟變異數就是不穩定的演算法，換演算法就能解決。

最小平方法同時展示兩者。如果特徵幾乎共線，係數本身就是條件不良的（問題）。解正規方程 $X^\top X w = X^\top y$ 會讓情況比必要的更糟，因為 $\kappa(X^\top X) = \kappa(X)^2$（演算法）；基於 QR 或 SVD 的求解器則直接處理 X。Ridge 迴歸加入 $\lambda\|w\|^2$，把問題改成 $(X^\top X + \lambda I)w = X^\top y$，其條件數 $(\sigma_{\max}^2 + \lambda)/(\sigma_{\min}^2 + \lambda)$ 較小——ML 01A 會進一步發展這個主題。

以梯度為基礎的最佳化

白話版。要最小化一個函數，就看它的斜率、往下坡走。梯度指向上升最陡的方向，所以往反方向走；步伐要夠小，讓你剛量到的斜率大致仍然有效。

導數、梯度與方向導數

我們要最小化**目標函數**（也稱成本、損失或誤差函數）$f:\mathbb{R}^n \to \mathbb{R}$。單變數時，導數 $f'(x)$ 給出一階近似 $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$，所以往 $f'(x)$ 的反號方向走一小步就能讓 f 下降。多變數時，**梯度** $\nabla_x f(x) \in \mathbb{R}^n$ 收集所有偏導數 $\partial f/\partial x_i$。單位方向 u 上的**方向導數**是沿 u 的斜率：

$$\frac{\partial}{\partial \alpha} f(x + \alpha u)\Big|_{\alpha=0} = u^\top \nabla_x f(x) = \|\nabla_x f(x)\|_2 \cos\theta,$$

其中 θ 是 u 與梯度的夾角。當 $\theta = \pi$，也就是 u 指向 $-\nabla_x f(x)$ 時，它最負。這就是**最陡下降法**，也就是**梯度下降**：

$$x \leftarrow x - \eta \nabla_x f(x),$$

其中 $\eta > 0$ 是**學習率**（步長）。η 可以取一個小常數、可以試幾個值並保留目標函數最低者（**線搜尋**，line search），也可以依照排程變化。當梯度的每個分量都（接近）0 時，梯度下降就收斂了。

臨界點：極小值、極大值與鞍點

滿足 $\nabla_x f(x) = 0$ 的點稱為**臨界點**或**駐點**。**局部極小值**比所有鄰近點都低，**局部極大值**比所有鄰近點都高，**鞍點**兩者皆非：它沿某些方向是極小、沿另一些方向是極大。**全域極小值**是整個定義域上的最低值；一個函數可能有好幾個全域極小值，也可能有許多不是全域極小的局部極小值。

對深度網路，我們很少找到真正的全域極小值，也不需要；只要參數讓損失夠低而且泛化良好就行。在高維空間中，對許多「類隨機」函數而言，鞍點比差勁的局部極小值更常見，而且因為梯度在鞍點附近變小，它們會拖慢以梯度為基礎的最佳化器 [8]。圖 1 顯示三種臨界點。

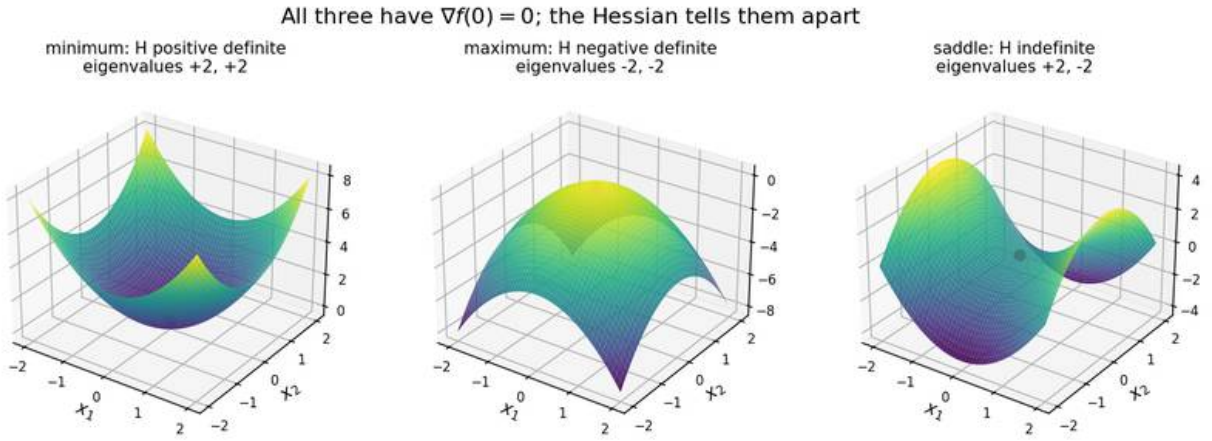


圖 *1* — 三個在原點梯度為零的函數。*Hessian* 的特徵值可以區分它們：全正（極小值）、全負（極大值）、正負皆有（鞍點）。

Jacobian 與 Hessian 矩陣

白話版。 Jacobian 收集向量值函數的所有一階導數。Hessian 收集純量函數的所有二階導數：它告訴你斜率本身如何變化，也就是曲率。

定義

對函數 $f : \mathbb{R}^m \to \mathbb{R}^n$，**Jacobian 矩陣** $J \in \mathbb{R}^{n \times m}$ 的元素為

$$J_{ij} = \frac{\partial f_i(x)}{\partial x_j}, \qquad i = 1, \dots, n,\ j = 1, \dots, m.$$

第 i 列是輸出 f_i 的梯度。反向傳播就是不斷乘上 Jacobian；反向模式自動微分計算向量–Jacobian 乘積 $v^\top J$，卻從不顯式建出 J，這就是為什麼一次梯度的成本只是一次函數求值的小常數倍 [11]。

對純量函數 $f : \mathbb{R}^n \to \mathbb{R}$，**Hessian 矩陣** $H(f)(x) \in \mathbb{R}^{n \times n}$ 是梯度的 Jacobian：

$$H(f)(x)_{ij} = \frac{\partial^2 f(x)}{\partial x_i \, \partial x_j}.$$

只要二階偏導數連續，微分順序就不影響結果，所以 $H_{ij} = H_{ji}$：Hessian 是對稱矩陣。實對稱矩陣有實數特徵值 $\lambda_1, \dots, \lambda_n$ 與一組正交歸一的特徵向量 $v_1, \dots, v_n$。沿單位方向 d 的**二階方向導數**是 $d^\top H d$。若 d 是特徵向量 v_i，它等於 λ_i；其他方向則是特徵值的加權平均 $\sum_i \lambda_i (d^\top v_i)^2$，權重介於 0 與 1 之間且總和為 1。所以 $\lambda_{\max}$ 是任一方向上的最大曲率，$\lambda_{\min}$ 是最小曲率。

一步梯度到底做了什麼：二階泰勒展開

令 $g = \nabla_x f(x^{(0)})$、$H = H(f)(x^{(0)})$。在 $x^{(0)}$ 附近的二階泰勒近似為

$$f(x) \approx f(x^{(0)}) + (x - x^{(0)})^\top g + \tfrac{1}{2}(x - x^{(0)})^\top H (x - x^{(0)}).$$

代入一步梯度 $x = x^{(0)} - \eta g$ 得到

$$f(x^{(0)} - \eta g) \approx f(x^{(0)}) - \eta\, g^\top g + \tfrac{1}{2}\eta^2\, g^\top H g.$$

三項分別是目前的值、斜率預測的改善量，以及曲率的修正。若 $g^\top H g \le 0$，η 越大，近似值就越小（直到近似本身失效為止）。若 $g^\top H g > 0$，曲率項最終會勝出，近似值在下式取得最小：

$$\eta^* = \frac{g^\top g}{g^\top H g}.$$

最壞情況下 g 與 $\lambda_{\max}$ 的特徵向量同向，此時 $\eta^* = 1/\lambda_{\max}$。曲率大，就只能走小步。

二階導數檢定

在臨界點（$g = 0$），泰勒展開只剩二次項，所以由 Hessian 決定這是哪一種點：

- H **正定**（所有 $\lambda_i > 0$）：局部極小值。
- H **負定**（所有 $\lambda_i < 0$）：局部極大值。
- H 至少有一個正特徵值與一個負特徵值：**鞍點**。
- 其他情況（有些 $\lambda_i = 0$，其餘同號）：檢定無法判斷。

一維中不可能有鞍點；但在 n 維中，只要有一個特徵值的符號「不對」就是鞍點，所以隨著 n 變大，鞍點變得非常普遍 [8]。

Hessian 的條件數：梯度下降為何鋸齒前進

在二次函數 $f(w) = \frac{1}{2} w^\top H w$（$H$ 對稱正定）上，梯度下降是線性的：$w \leftarrow (I - \eta H) w$。把 w 用 H 的特徵向量展開，沿 v_i 的分量每一步都乘上 $(1 - \eta\lambda_i)$。因此：

- 只有當每個 i 都滿足 $|1 - \eta\lambda_i| < 1$，也就是 $0 < \eta < 2/\lambda_{\max}$ 時，迭代才**穩定**。
- **最慢**的是平坦方向，每步只縮小 $|1 - \eta\lambda_{\min}|$ 倍。
- 最佳固定步長 $\eta = 2/(\lambda_{\max} + \lambda_{\min})$ 讓誤差每步縮小 $(\kappa - 1)/(\kappa + 1)$ 倍，其中 $\kappa = \lambda_{\max}/\lambda_{\min}$ 是 Hessian 的條件數。

所以條件數大，表示某個方向要求極小的步伐（否則發散），另一個方向卻需要很多步才有進展。梯度下降把時間浪費在狹窄山谷的陡峭兩壁間來回彈跳——圖 2 中的鋸齒——同時沿著谷底緩慢前進。輸入特徵尺度不一致正好造成這種情況，這也是標準化與正規化層能加快訓練的原因之一。

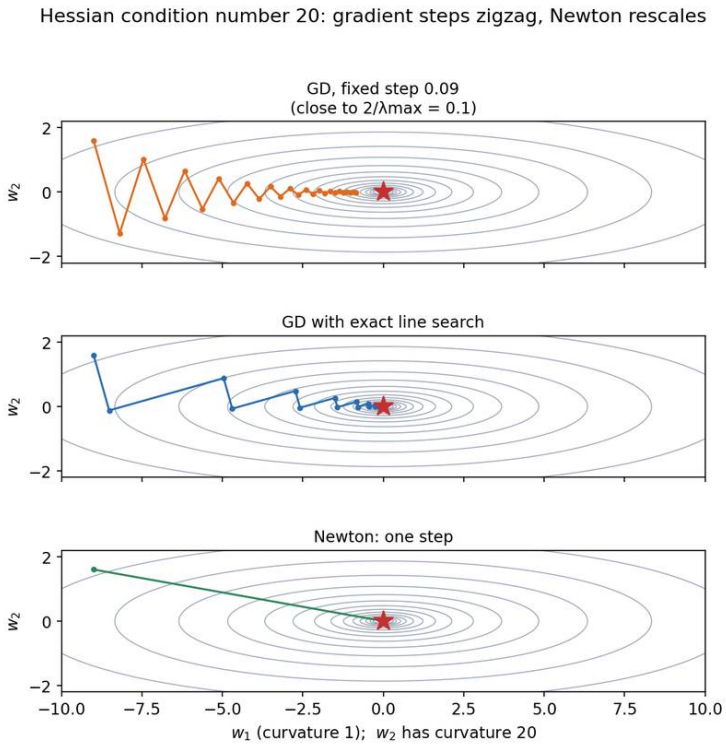


圖 2 — 最小化 *Hessian* 條件數為 *20* 的二次函數。步長接近穩定極限的梯度下降在陡峭方向來回振盪；精確線搜尋使相鄰兩步互相垂直；牛頓法以 *Hessian* 的反矩陣重新縮放，對二次函數一步就到達極小值。

圖 3 以數值呈現穩定極限。低於 $2/\lambda_{\max}$ 時損失下降；稍高於極限時，陡峭分量每步放大 $|1 - \eta\lambda_{\max}| > 1$ 倍，損失隨之爆炸。

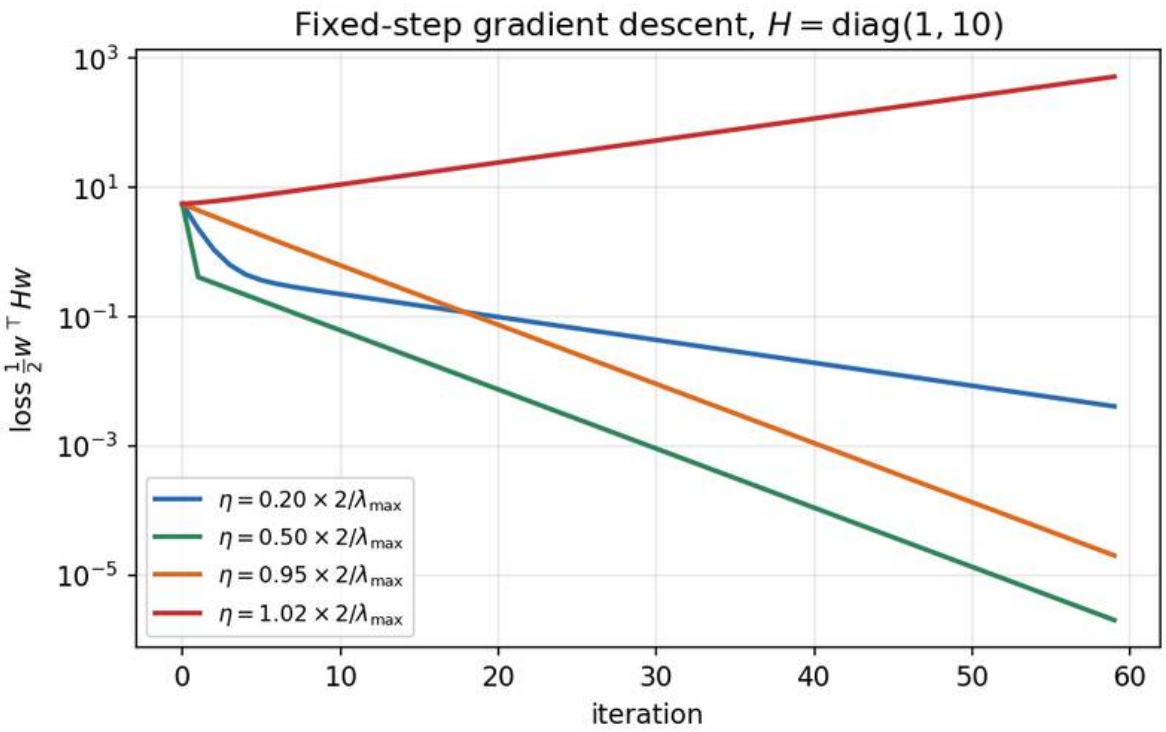


圖 3 — 在特徵值為 1 與 10 的二次函數上做固定步長梯度下降。小於 $2/\lambda max$ 的步長都會收斂（在 $2/(\lambda max+\lambda min)$ 附近最快）；只比極限大 2% 的步長就會發散。

```
import numpy as np

H = np.diag([1.0, 25.0])                    # Hessian: condition number 25
f = lambda w: 0.5 * w @ H @ w
grad = lambda w: H @ w

def run_gd(eta, steps=50):
    w = np.array([10.0, 1.0])
    for _ in range(steps):
        w = w - eta * grad(w)
    return f(w)

lam_max = np.linalg.eigvalsh(H).max()
for eta in (0.01, 0.07, 0.079, 0.081):       # stability limit 2/25 = 0.08
    print(f"eta={eta:<6} loss after 50 steps: {run_gd(eta):.3e}")

# Exact line search along -g for a quadratic: eps* = g'g / g'Hg
w = np.array([10.0, 1.0])
g = grad(w)
eps_star = (g @ g) / (g @ H @ g)
print("optimal step along the gradient:", round(eps_star, 4))

# One Newton step lands on the minimizer of a quadratic
w_newton = w - np.linalg.solve(H, g)
```

```
print("after one Newton step:", w_newton)
```

輸出說明了一切：$\eta = 0.01$ 穩定但慢（50 步後損失 18.3），$\eta = 0.07$ 降到 0.035，$\eta = 0.079$ 勉強穩定但陡峭分量仍在振盪（損失約 1.0），$\eta = 0.081$ 發散（損失 148）。沿第一個梯度的最佳步長是 0.046，而一步牛頓法正好落在 $(0, 0)$。

二階方法

白話版。牛頓法利用曲率重新縮放梯度，在條件良好的極小值附近可能有幫助；不定曲率則可能給出不良方向。

牛頓法

令二階泰勒近似的梯度為零，得到**牛頓步**：

$$x^{*} = x^{(0)} - H(f)(x^{(0)})^{-1}\,\nabla_x f(x^{(0)}).$$

在正定二次函數上，這一步直接跳到極小值（圖 2 下）。對局部接近正定二次函數的一般函數，反覆套用這個更新比梯度下降收斂快得多——在標準假設下，於解附近呈二次收斂 [6]。代價是 Hessian：儲存它需要 $O(n^2)$ 記憶體，用它求解需要 $O(n^3)$ 時間，對一個有 10^8 個參數的網路根本不可能。

對非凸目標還有更根本的問題。牛頓法找的是梯度為零的點，它對待鞍點和極小值一視同仁：在鞍點附近，牛頓步會指向鞍點。在充滿鞍點的地形上，單純的牛頓法正好被我們想逃離的點吸引。Dauphin 等人提出「saddle-free Newton」，改用 Hessian 特徵值的絕對值來縮放，以避免這個問題 [8]。相對地，梯度下降不會以同樣方式被鞍點吸引，雖然它在鞍點附近可能走得很慢。

擬牛頓法與 L-BFGS

擬牛頓法（quasi-Newton）只用梯度建立 $B_k \approx H$（或其反矩陣）的近似。走了一步 $s_k = x_{k+1} - x_k$、梯度變化為 $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$ 之後，它要求**割線條件** $B_{k+1} s_k = y_k$，意思是模型沿這一步的曲率要與觀察到的斜率變化一致。BFGS 是這類更新中最常用的；**L-BFGS**（有限記憶 BFGS）只保存最近 m 組 (s_k, y_k)，以 $O(mn)$ 時間隱式套用反 Hessian 的近似 [6]、[10]。L-BFGS 很適合平滑、確定性、中等規模的問題（logistic 迴歸、單張影像的風格轉換、擬合幾千個參數）；但當梯度是有雜訊的小批次估計時，它的曲率估計會受影響。

Hessian–向量乘積

許多利用曲率的方法其實不需要 H 本身，只需要乘積 Hv。這可以用大約幾次梯度計算的成本精確算出，因為 $Hv = \nabla_x(\nabla_x f(x)^{\top} v)$ 是某個純量的梯度；Pearlmutter 說明了如何用自動微分做到 [9]。有限差分版本 $Hv \approx [\nabla f(x + rv) - \nabla f(x - rv)]/(2r)$ 是很實用的檢查。以 Hessian–向量乘積做冪迭代（power iteration）可以估計 $\lambda_{\max}$，研究者正是用這個方法追蹤訓練中損失的尖銳度（sharpness）。

```
import numpy as np

rng = np.random.default_rng(0)
X = rng.normal(size=(50, 4))
y = (rng.random(50) < 0.5).astype(float)

def grad(w):                                    # logistic regression, mean loss
    p = 1 / (1 + np.exp(-(X @ w)))
    return X.T @ (p - y) / len(y)

def hessian(w):
    p = 1 / (1 + np.exp(-(X @ w)))
    return (X.T * (p * (1 - p))) @ X / len(y)

w = rng.normal(size=4)
v = rng.normal(size=4)
r = 1e-5
hvp_fd = (grad(w + r*v) - grad(w - r*v)) / (2*r)   # never forms H
print("Hv (finite diff):", hvp_fd.round(6))
print("Hv (explicit)   :", (hessian(w) @ v).round(6))
print("eigenvalues of H:", np.linalg.eigvalsh(hessian(w)).round(4))
```

兩種乘積在小數點後六位都相同，而且 Hessian 的特徵值全為正：logistic 迴歸是凸的，它的 Hessian 處處半正定。

Lipschitz 連續與凸性

深度學習的目標函數幾乎沒有什麼保證，但有兩個性質能解釋大部分理論。若對所有 x, y 都有 $|f(x) - f(y)| \le L\|x - y\|_2$，則函數是常數為 L 的 **Lipschitz 連續**：它的變化率有上界。若梯度是 L-Lipschitz（函數「L-平滑」），曲率就以 L 為上界，而步長 $\eta \le 1/L$ 的梯度步保證不會讓函數變大：

$$f(x - \eta \nabla f(x)) \le f(x) - \frac{\eta}{2}\|\nabla f(x)\|_2^2 \qquad (0 < \eta \le 1/L).$$

這是前面「$\eta < 2/\lambda_{\max}$」規則的一般形式，也是大多數收斂證明的起點 [7]、[14]。

若函數（二次可微時）的 Hessian 處處半正定，它就是**凸函數**。凸函數沒有鞍點，也沒有虛假的局部極小值：每個局部極小值都是全域極小值。最小平方、logistic 迴歸與支援向量機都是凸的；深度網路不是。凸最佳化是許多工具的來源，例如對偶性與下面的 KKT 條件，深度學習也借用了它們 [7]。

有限制條件的最佳化

白話版。限制條件界定可接受的解；投影把更新帶回可行域，拉格朗日乘數則描述限制如何進入最佳性條件。

投影梯度下降

我們要在可行集合 $\mathbb{S} \subseteq \mathbb{R}^n$ 上最小化 $f(x)$。如果投影到 $\mathbb{S}$ 很容易，就先走一般的梯度步，再回到最近的可行點：

$$x \leftarrow \operatorname{Proj}_{\mathbb{S}}\left(x - \eta \nabla f(x)\right), \qquad \operatorname{Proj}_{\mathbb{S}}(v) = \arg\min_{x \in \mathbb{S}} \|x - v\|_2 .$$

對歐氏球 $\{x : \|x\|_2 \le r\}$，投影為 $v \cdot \min(1, r/\|v\|_2)$：太長就縮短。權重的最大範數限制、把參數截在一個方盒內、投影到機率單形（simplex）都屬於這個模式。另一種做法是重新參數化，讓限制條件自動成立，例如用 $x = r\,(\cos\theta, \sin\theta)$ 表示圓上的向量，或用 softmax 產生機率。

廣義拉格朗日函數

更一般的做法是用 m 個等式限制 $g^{(i)}(x) = 0$ 與 k 個不等式限制 $h^{(j)}(x) \le 0$ 描述可行集合：

$$\mathbb{S} = \{x \mid g^{(i)}(x) = 0 \ \forall i,\ h^{(j)}(x) \le 0 \ \forall j\}.$$

為每個限制條件引入一個 **KKT 乘數**，等式用 λ_i、不等式用 α_j，並定義**廣義拉格朗日函數**

$$L(x, \lambda, \alpha) = f(x) + \sum_{i=1}^{m} \lambda_i\, g^{(i)}(x) + \sum_{j=1}^{k} \alpha_j\, h^{(j)}(x).$$

原本有限制的問題等價於沒有限制的 min–max 問題

$$\min_{x} \max_{\lambda} \max_{\alpha,\ \alpha \ge 0} L(x, \lambda, \alpha).$$

為什麼可行？若 x 違反某個限制，內層最大化可以選一個很大的乘數，把 L 推到 $+\infty$，所以不可行點永遠不會被選上。若 x 可行，內層最大化最多只能讓這些項為零（等式本來就是 0；對 $h^{(j)}(x) \le 0$，最好的非負 α_j 會使 $\alpha_j h^{(j)}(x) = 0$），因此值恰好是 $f(x)$ [1]。

KKT 條件

在解 x^* 處（在稱為「限制條件規範性」〔constraint qualification〕的溫和正則條件下），存在乘數滿足 **Karush–Kuhn–Tucker（KKT）條件** [1]、[7]：

1. **穩定性（stationarity）：** $\nabla_x L(x^*, \lambda^*, \alpha^*) = 0$。
2. **原始可行性：** x^* 滿足所有限制條件。
3. **對偶可行性：** $\alpha^* \ge 0$。
4. **互補鬆弛性：**對每個 j，$\alpha_j^*\, h^{(j)}(x^*) = 0$。

互補鬆弛性是說，每個不等式要嘛**有效**（active，$h^{(j)}(x^*) = 0$：解落在邊界上，限制條件以 $\alpha_j \ge 0$ 的力量把它推回來），要嘛**無效**（inactive，$h^{(j)}(x^*) < 0$：限制條件在局部不起作用，$\alpha_j = 0$）。從幾何上看，穩定性條件是說：在有效限制處，下坡方向 $-\nabla f$ 正好垂直指出可行集合之外，與 ∇h 平行；任何可行的移動都會往上坡走（圖 4）。對凸問題，KKT 條件也是最佳性的充分條件 [7]。

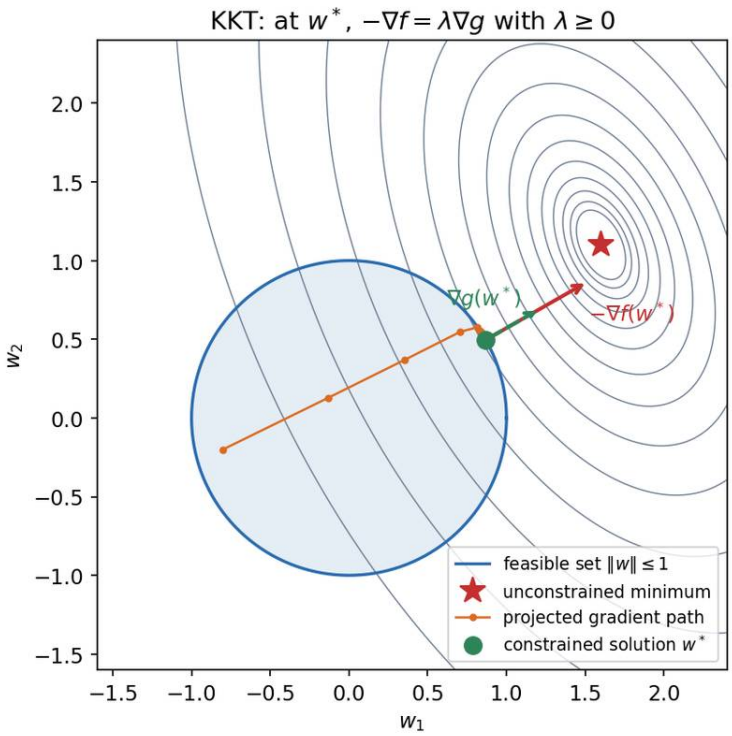


圖 4 — 在單位圓盤上最小化二次函數。無限制的極小值（星號）不可行。投影梯度下降停在邊界上的 w^*，此處 $-\nabla f$ 與限制條件的外法向量 ∇g 平行：這正是乘數為正時的 *KKT* 穩定性條件。

範例：帶範數限制的最小平方

依照 [1] 的例子，在限制 $h(w) = w^\top w - r^2 \leq 0$ 下最小化平方誤差 $f(w) = \|Aw - b\|_2^2$，其中 $A \in \mathbb{R}^{N \times n}$、$b \in \mathbb{R}^N$。拉格朗日函數為

$$L(w, \alpha) = \|Aw - b\|_2^2 + \alpha\,(w^\top w - r^2).$$

穩定性條件給出 $2A^\top(Aw - b) + 2\alpha w = 0$，所以

$$w(\alpha) = (A^\top A + \alpha I)^{-1} A^\top b.$$

這就是懲罰係數為 α 的 ridge 迴歸。若一般最小平方解 $w(0)$ 已經滿足 $\|w\| \leq r$，限制條件無效，$\alpha^* = 0$，問題解決。否則限制條件有效，就增大 α 直到 $\|w(\alpha)\| = r$；因為 $\|w(\alpha)\|$ 隨 α 單調遞減，用二分法就能找到。這也正是對 α 做內層最大化的行為：限制被違反時乘數變大，剛好滿足時停下。下面的程式用兩種方法求解，並檢查全部四個 KKT 條件。

```
import numpy as np

# minimize ||Aw - b||^2  subject to  ||w||^2 <= r^2
rng = np.random.default_rng(1)
A = rng.normal(size=(30, 5))
b = rng.normal(size=30)
```

```
r = 0.2

def w_of(lam):                                  # stationarity of the Lagrangian
    return np.linalg.solve(A.T @ A + lam * np.eye(5), A.T @ b)

w_ls = w_of(0.0)
print("unconstrained norm:", np.linalg.norm(w_ls).round(3))

lo, hi = 0.0, 1e6                               # find lambda with ||w|| = r
for _ in range(200):
    mid = 0.5 * (lo + hi)
    if np.linalg.norm(w_of(mid)) > r:
        lo = mid
    else:
        hi = mid
lam = hi
w = w_of(lam)
print("lambda:", round(lam, 3), " ||w||:", np.linalg.norm(w).round(6))

# KKT checks
stationarity = 2 * A.T @ (A @ w - b) + 2 * lam * w
print("stationarity residual:", np.abs(stationarity).max())
print("primal feasible:", w @ w <= r**2 + 1e-9, " dual feasible:", lam >= 0)
print("complementary slackness:", lam * (w @ w - r**2))

# Projected gradient descent reaches the same point
L = 2 * np.linalg.eigvalsh(A.T @ A).max()     # Lipschitz constant of the gradient
v = np.zeros(5)
for _ in range(5000):
    v = v - (1 / L) * 2 * A.T @ (A @ v - b)
    n = np.linalg.norm(v)
    if n > r:
        v = v * (r / n)                         # projection onto the ball
print("projected GD matches KKT solution:", np.allclose(v, w, atol=1e-6))
```

無限制解的範數為 0.219，大於 $r = 0.2$，所以限制條件有效。二分法找到 $\alpha^* \approx 1.81$，穩定性殘差約 10^{-15}，互補鬆弛性成立，而步長 $1/L$ 的投影梯度下降收斂到同一個向量。請注意，梯度的 Lipschitz 常數 L 決定了安全的步長，正如上一節所說。

檢查導數

白話版。檢查程式算出的梯度，是否與「輕推參數時損失實際的變化」一致。

中央有限差分估計偏導數：

$$\frac{\partial f}{\partial x_j} \approx \frac{f(x + \epsilon e_j) - f(x - \epsilon e_j)}{2\epsilon},$$

其中 e_j 是第 j 個標準基底向量，$\epsilon > 0$ 是小步長。它的**截斷誤差**（忽略高階泰勒項造成）是 $O(\epsilon^2)$，優於前向差分 $[f(x + \epsilon e_j) - f(x)]/\epsilon$ 的 $O(\epsilon)$。但分子是兩個幾乎相等的數相減，所以**捨入誤差**以 $u\,|f|/\epsilon$ 的速度增長。總誤差在中等步長時最小——當 f 與其導數為單位尺度時，中央差分約在

$\epsilon \approx u^{1/3}$、前向差分約在 $u^{1/2}$。圖 5 顯示這種 V 形曲線，也說明為什麼 float32 的梯度檢查遠不如 float64 精確。

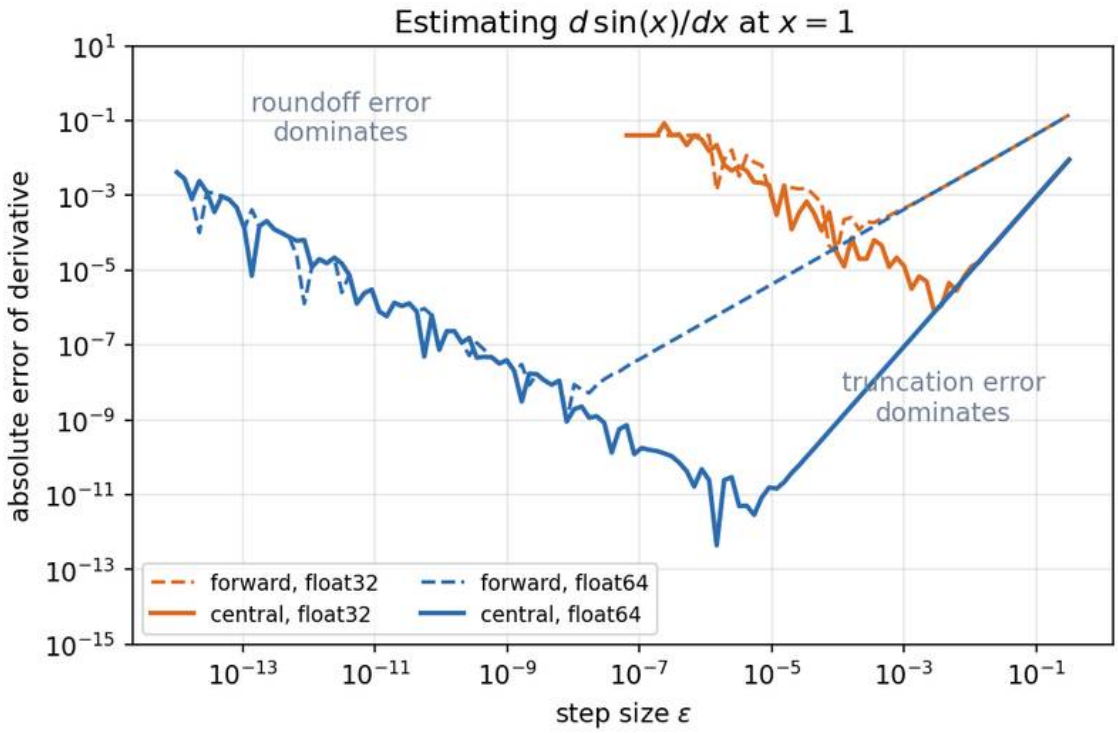


圖 5 — 在 *x=1* 處以有限差分估計 $sin(x)$ 導數的誤差。最低點右側由截斷誤差主導（前向差分斜率 *1*、中央差分斜率 *2*）；左側由捨入誤差主導。中央差分在 *float32* 最低約 *1e−6*，在 *float64* 可低到約 *1e−11* 或更小。

實用原則：用 float64 做檢查；嘗試多個步長；用相對誤差比較，例如 $|a-n|/\max(|a|,|n|,\text{tiny})$；並在平滑的點上測試。ReLU 在 0 處不可微，那裡沒有單一導數可以比對。參數很多時，**方向檢查**更省：選一個隨機單位向量 v，比較 $\nabla f(w)^\top v$ 與沿 v 的有限差分——只需要一對函數求值，而不是 $2n$ 次。

```
import numpy as np

rng = np.random.default_rng(0)
X = rng.normal(size=(20, 5))
y = rng.normal(size=20)
w = rng.normal(size=5)
v = rng.normal(size=5)
v /= np.linalg.norm(v)
def loss(w):
    return .5*np.mean((X@w-y)**2)
grad = X.T@(X@w-y)/len(y)
eps = 1e-5
numeric = (loss(w+eps*v)-loss(w-eps*v))/(2*eps)
analytic = grad@v
```

```
assert np.allclose(numeric, analytic, rtol=1e-6, atol=1e-8)
print("directional derivative:", numeric, analytic)
```

兩個數字大約有九位有效數字相同。通過檢查只確認導數，不代表任務寫對了。錯誤歸約、意外廣播或目標洩漏，都可能定義出可微卻錯誤的目標函數，讓梯度檢查通過 [11]。意外使用 `detach` 則不同：若前向數值仍隨輸入改變，但自動微分切斷了這條依賴，有限差分便可能抓出不一致 [24]。應在平滑點檢查相關輸入、固定隨機性，並以已知答案的案例核對損失值。

混合精度

白話版。大部分運算用 16 位元數字，可以讓記憶體傳輸量減半並使用高速硬體單元；但 16 位元不夠應付所有事情，少數敏感的量要保留在 32 位元。

Micikevicius 等人提出的混合精度訓練 [12] 結合三個想法：

1. 保留一份 **float32 主權重**（master weights）。當更新量小於所加權重的約 2^{-11} 倍時，在 float16 中加上去會被完全捨去（與上文 $1 + \varepsilon_{\text{mach}}/2 = 1$ 是同一個現象），而很多更新就是這麼小。
2. float16 要用**損失縮放**（loss scaling）：反向傳播前把損失乘上因子 S，讓小梯度不會下溢到 float16 最小次正規數（6×10^{-8}）以下，更新前再把梯度除以 S。動態損失縮放在出現溢位（inf 或 NaN）時降低 S，並跳過該步。
3. 即使輸入是 float16，內積與歸約也以 float32 **累加**。

bfloat16 的指數與 float32 一樣有 8 位元，因此很少需要損失縮放；Kalamkar 等人發現，bfloat16 訓練在多種模型上都能達到 float32 的結果，而且不需要調整超參數 [5]。代價是精度：$\varepsilon_{\text{mach}} \approx 0.0078$ 的 bfloat16 無法區分 $1 + 0.003$ 與 1，所以長串加總與權重更新仍需要 float32 累加。

損失縮放保護的是 float16 的小梯度；它無法修正錯誤的目標函數，也無法修正網路其他地方的溢位（例如注意力 logit 或正規化統計量）。改變精度也可能改變最佳化軌跡與輸出，特別是歸約運算與長時間訓練。請在目標硬體上驗證；較小的資料型別不保證每個運算都會變快 [13]。

僅線上版：*Mixed Precision Training*：精度該保留在哪裡 — *https://blog.shyandram.dev/zh-tw/learn/ml/numerical-computation/papers/mixed-precision-training 精度該保留在哪裡/*

非有限值的除錯順序

損失變成 `nan` 或 `inf` 時，別急著把所有東西都截斷。先找出第一個產生非有限值的運算，再判斷它屬於本篇哪一種失效模式。實用的順序：

1. **輸入：**檢查形狀、範圍、資料型別與缺值。一個 NaN 像素或一張沒有正規化的 0–255 影像，就會改變下游的一切。
2. **前向傳播：**檢查激活值與損失是否有限。留意 `exp`、`log`、除法、`sqrt`，以及變異數接近 0 的正規化層。
3. **損失：**使用以 logit 為輸入的損失（`cross_entropy`、`binary_cross_entropy_with_logits`），而不是先算機率再取 `log`。
4. **梯度：**在小模型上做梯度檢查；查看各層的梯度範數。

5. **更新：**比較更新量與權重的大小；學習率超過曲率極限 $2/\lambda_{\max}$ 時，即使算術完全精確，也會振盪甚至發散。
6. **精度：**先在 float32（或 float64）中重現，再加入混合精度或編譯。

先讓模型過擬合一個極小的批次，確認整條流程能學習。梯度範數截斷（gradient-norm clipping）限制的是傳給最佳化器的梯度，並不直接限制自適應最佳化器的更新量，也無法讓 NaN 梯度變得有意義。ML 02 會更詳細地介紹最佳化器與正規化。

現代觀點

本篇的核心概念已有數十年歷史，但因為模型更大、訓練更久、硬體不斷把精度往下壓，它們仍站在當前研究的中心。

大規模學習的最佳化理論。 Bottou、Curtis 與 Nocedal 發表在 SIAM Review 的綜述 [14] 是這個領域的標準地圖。它解釋為什麼隨機梯度法主導大規模機器學習（資料有冗餘時，便宜而有雜訊的步伐勝過昂貴而精確的步伐），在類似上文 Lipschitz 條件的平滑性假設下分析其收斂，並把改進方法分成兩大類：降低雜訊（更大或逐漸增大的批次、梯度聚合／變異數縮減、迭代平均）與二階方法（Hessian-free 牛頓法、Gauss–Newton、擬牛頓法、自然梯度與對角縮放）。Nocedal 與 Wright [6]、Boyd 與 Vandenberghe [7] 仍是確定性與凸理論的標準教科書；Baydin 等人 [11] 則綜述了讓梯度變得便宜的自動微分。

損失曲面實際長什麼樣子。 Dauphin 等人 [8] 主張，高維非凸最佳化的主要障礙是鞍點，而不是差勁的局部極小值。後來 Cohen 等人 [15] 觀察到，神經網路上的全批次梯度下降通常運作在「穩定性邊緣」（edge of stability）：Hessian 的最大特徵值會上升到剛好略高於 $2/\eta$——正是上文推導的古典穩定極限——而損失在長時間尺度上仍持續下降，只是不再單調。古典的二次分析預測此時會發散，所以如何解釋這個狀態仍是開放的理論問題。

考慮曲率的最佳化器。完整的牛頓法在大規模下不可行，但結構化的近似很實用。K-FAC [16] 以每層小矩陣的 Kronecker 乘積近似 Fisher 資訊矩陣（一種半正定的曲率矩陣，與 Hessian 的 Gauss–Newton 近似密切相關），讓求反矩陣變得便宜。Shampoo [17] 為張量的每個維度各維護一個預條件（preconditioning）矩陣。更近期的 Muon 對每個權重矩陣的更新量做近似正交化；Liu 等人 [18] 報告，在加入權重衰減並仔細調整每個參數的更新尺度後，它在其實驗中以約兩倍於所比較之自適應最佳化器基準的計算效率訓練大型語言模型。這些方法都在重新回答本篇用 Hessian 提出的問題：一步更新應該如何依方向重新縮放？

更低的精度。格式的演進從搭配損失縮放的 float16 [12]、bfloat16 [5]，一路走到 8 位元浮點數。Micikevicius 等人 [19] 提出兩種 FP8 編碼：E4M3（精度較高）與 E5M2（範圍較大），並在影像與語言模型上展示了與 16 位元基準相當的訓練結果。上方表格中的範圍–精度取捨，正是這些設計要平衡的東西，而它們放寬的是 IEEE 754 的規則 [3]。逐張量或逐區塊的縮放因子，扮演著 float16 時代損失縮放的角色。

大規模下的穩定 softmax。減去最大值的技巧有串流版本。Milakov 與 Gimelshein [20] 證明，只要維護一個執行中的最大值，並在最大值改變時重新縮放累加和，就能一趟算出 softmax 的正規化因子。FlashAttention [21] 以此為基礎，在高速的晶片上記憶體中逐區塊計算精確的注意力，從不實際建出完整的注意力矩陣；FlashAttention-3 [22] 進一步延伸到 FP8，並以區塊量化與「incoherent processing」降低量化誤差。這個線上技巧的核心只要幾行：

```
import numpy as np

def online_logsumexp(chunks):
    m, s = -np.inf, 0.0                 # running max and rescaled running sum
    for c in chunks:
        m_new = max(m, c.max())
        s = s * np.exp(m - m_new) + np.exp(c - m_new).sum()
        m = m_new
    return m + np.log(s)

rng = np.random.default_rng(0)
z = rng.normal(0, 50, size=10_000)
chunks = np.array_split(z, 37)          # seen one block at a time
m = z.max()
reference = m + np.log(np.exp(z - m).sum())
print(online_logsumexp(chunks), reference)
assert np.isclose(online_logsumexp(chunks), reference)
```

兩個值都印出 174.0918752579…，在顯示的位數內完全相同，雖然線上版本從未同時看到所有 logit。ML 05 介紹注意力本身，ML 11 介紹部署時的量化。

開放問題。梯度方法為何能在穩定性邊緣成功；如何讓低精度（8 位元及以下）訓練像 float32 一樣穩健，而不必手動調整縮放；如何讓考慮曲率的最佳化器夠便宜、夠穩健，足以取代一階方法成為預設；以及在平行歸約改變加總順序時，如何得到可重現的結果。

重點整理

- 指數位元決定範圍，尾數位元決定精度；次正規數的相對精度行為不同。
- 提高精度前，先使用 log-sum-exp 與以 logits 計算損失等穩定公式。
- 條件性衡量輸入敏感度；穩定演算法無法消除問題本身的病態敏感性。
- 曲率決定二次目標的安全步長；一般非凸損失需要額外考量。
- 投影維持可行性；KKT 條件需要正則性假設，並非通用的全域最佳證明。
- 在光滑點以合適精度與步長檢查梯度，並在實際運算中驗證混合精度。

練習

1. **Softmax 平移。**證明 softmax 在 $z_i \mapsto z_i + c$ 下不變，並說明取 $c = -\max_i z_i$ 時分母落在 $[1, K]$ 之間。

   提示

   共同因子 $\exp(c)$ 在分子與分母間抵銷。平移後每一項 $\exp(z_j - m)$ 都在 $(0, 1]$ 之間，最大的一項等於 1，所以 K 項的總和介於 1 與 K 之間，既不會溢位，也不會下溢成 0。

2. **有限差分掃描。**對 $f(x) = e^x$ 在 $x = 0$ 處，以 $\epsilon = 10^{-1}, \dots, 10^{-12}$ 分別在 float32 與 float64 中計算前向與中央差分。各自在哪個 ϵ 誤差最小？與 $u^{1/2}$、$u^{1/3}$ 比較。

提示

float64 的 $u \approx 1.1\times10^{-16}$，所以 $u^{1/2} \approx 10^{-8}$（前向）、$u^{1/3} \approx 5\times10^{-6}$（中央）。float32 的 $u \approx 6\times10^{-8}$，對應約 2×10^{-4} 與 4×10^{-3}。觀察到的最佳步長應與這些值相差不到一個數量級，最低誤差則大約分別是 $u^{1/2}$ 與 $u^{2/3}$。

3. **穩定極限**。對 $f(w) = \frac{1}{2}w^\top H w$，$H = \mathrm{diag}(2, 50)$，求最大的穩定固定學習率，以及使最壞情況收縮因子最小的步長。用這個最佳步長，需要幾次迭代才能把誤差縮小 10^6 倍？

提示

穩定需要 $\eta < 2/50 = 0.04$。最佳步長為 $2/(\lambda_{\max}+\lambda_{\min}) = 2/52 \approx 0.0385$，收縮因子為 $(\kappa-1)/(\kappa+1) = 24/26 \approx 0.923$。解 $0.923^k \le 10^{-6}$：$k \ge \ln 10^6/(-\ln 0.923) \approx 173$ 次。牛頓法只要一步。

4. **判斷臨界點**。找出 $f(x,y) = x^3 - 3x + y^2$ 的臨界點並加以分類。

提示

$\nabla f = (3x^2 - 3,\ 2y) = 0$ 給出 $(1,0)$ 與 $(-1,0)$。Hessian 為 $\mathrm{diag}(6x,\ 2)$：在 $(1,0)$ 是 $\mathrm{diag}(6, 2)$，正定，為局部極小值；在 $(-1,0)$ 是 $\mathrm{diag}(-6, 2)$，不定，為鞍點。兩者都不是全域極小值，因為 $x \to -\infty$ 時 $f \to -\infty$。從 $(-1,0)$ 附近開始的牛頓法會收斂到鞍點。

5. **單形上的 KKT**。在 $\sum_i x_i = 1$ 且 $x_i \ge 0$ 的限制下，對 $c = (0.8, 0.6, -0.4)$ 最小化 $f(x) = \frac{1}{2}\|x - c\|_2^2$。寫出拉格朗日函數與 KKT 條件，並求出解。

提示

$L = \frac{1}{2}\|x-c\|^2 + \lambda(\sum_i x_i - 1) - \sum_i \alpha_i x_i$，$\alpha_i \ge 0$。穩定性給出 $x_i = c_i - \lambda + \alpha_i$；互補鬆弛性給出 $x_i = \max(c_i - \lambda, 0)$。猜測只有前兩個為正：$(0.8-\lambda) + (0.6-\lambda) = 1$，所以 $\lambda = 0.2$、$x = (0.6, 0.4, 0)$。驗證：$c_3 - \lambda = -0.6 < 0$，所以 $x_3 = 0$，且 $\alpha_3 = 0.6 \ge 0$。這就是到機率單形的歐氏投影。

6. **錯誤損失的正確梯度**。在梯度檢查的範例中，把 mean 歸約改成 sum。檢查還會通過嗎？自動微分錯了嗎？訓練上還有什麼會改變？

提示

如果同時一致地修改損失與解析梯度，檢查會通過：對新的目標函數而言，導數是正確的。自動微分沒有錯。但梯度變成原來的 N 倍，所以最大的穩定學習率縮小為 $1/N$；在多項損失中，各項之間的平衡也會改變。梯度檢查無法告訴你目標函數選錯了。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 4 (Numerical Computation) and ch. 8. Book

2. D. Goldberg, “What Every Computer Scientist Should Know About Floating-Point Arithmetic,” *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991. DOI
3. IEEE, “IEEE Standard for Floating-Point Arithmetic,” IEEE Std 754-2019, 2019. DOI
4. N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, 2002. DOI
5. D. Kalamkar et al., “A Study of BFLOAT16 for Deep Learning Training,” arXiv:1905.12322, 2019. arXiv
6. J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., Springer, 2006. DOI
7. S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004. Book
8. Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli and Y. Bengio, “Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization,” *NeurIPS*, 2014. Paper
9. B. A. Pearlmutter, “Fast Exact Multiplication by the Hessian,” *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994. DOI
10. D. C. Liu and J. Nocedal, “On the Limited Memory BFGS Method for Large Scale Optimization,” *Mathematical Programming*, vol. 45, pp. 503–528, 1989. DOI
11. A. G. Baydin, B. A. Pearlmutter, A. A. Radul and J. M. Siskind, “Automatic Differentiation in Machine Learning: a Survey,” *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018. arXiv
12. P. Micikevicius et al., “Mixed Precision Training,” *ICLR*, 2018. arXiv
13. PyTorch documentation, “Numerical Accuracy.” Documentation
14. L. Bottou, F. E. Curtis and J. Nocedal, “Optimization Methods for Large-Scale Machine Learning,” *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018. DOI
15. J. M. Cohen, S. Kaur, Y. Li, J. Z. Kolter and A. Talwalkar, “Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability,” *ICLR*, 2021. arXiv
16. J. Martens and R. Grosse, “Optimizing Neural Networks with Kronecker-factored Approximate Curvature,” *ICML*, 2015. Paper
17. V. Gupta, T. Koren and Y. Singer, “Shampoo: Preconditioned Stochastic Tensor Optimization,” *ICML*, 2018. Paper
18. J. Liu, J. Su, X. Yao et al., “Muon is Scalable for LLM Training,” arXiv:2502.16982, 2025. arXiv
19. P. Micikevicius et al., “FP8 Formats for Deep Learning,” arXiv:2209.05433, 2022. arXiv
20. M. Milakov and N. Gimelshein, “Online Normalizer Calculation for Softmax,” arXiv:1805.02867, 2018. arXiv
21. T. Dao, D. Y. Fu, S. Ermon, A. Rudra and C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness,” *NeurIPS*, 2022. Paper
22. J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani and T. Dao, “FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision,” *NeurIPS*, 2024. Paper
23. NumPy developers, “numpy.linalg.cond,” NumPy documentation, accessed 2026-11-05. Documentation
24. PyTorch contributors, “torch.autograd.gradcheck.gradcheck,” PyTorch documentation, accessed 2026-11-05. Documentation

ML 02・類神經網路：從感知器到深度網路

從單一感知器到深度網路：激活函數、正規化層、輸出層與損失函數、反向傳播、最佳化器，以及讓網路訓練得起來又能泛化的實務習慣。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/neural-networks/

你將學到

- 區分感知器、邏輯迴歸與多層網路。
- 說明非線性激活與正規化的作用。
- 依預測任務搭配輸出層與損失。
- 推導反向傳播，並以數值方式檢查梯度。
- 由初始化、最佳化與正則化選擇診斷訓練。

先看全貌

類神經網路是一個帶有許多可調旋鈕（也就是**參數**）的函數。它接收輸入，例如影像的像素或平面上一個點的兩個座標，然後產生輸出，例如類別標籤。所謂訓練，就是每次把旋鈕轉動一點點，讓模型在已知範例上的輸出越來越接近正確答案。本篇的所有內容都在回答三個問題之一：這個函數長什麼樣子（架構）、怎麼衡量「更接近」（損失函數），以及怎麼把旋鈕轉好（最佳化）。

> **用最簡單的話說**
>
> 想像一排人在傳紙條。第一個人看著一張圖片，寫下幾個簡單的觀察（「這裡有一條弧線」、「這一塊很亮」）。下一個人讀了這些紙條，寫出稍微聰明一點的結論（「這看起來像一隻眼睛」）。最後一個人說「貓」或「狗」。答錯的時候，老師從隊伍尾巴往回走，告訴每個人他的紙條要負多少責任，每個人就把自己的習慣改一點點。往回走的那一趟就是**反向傳播**，重複幾千幾萬次就是**訓練**。

為什麼重要：今天在影像、語音與語言領域的幾乎所有系統，都是由本篇介紹的零件組成的。卷積網路、Transformer 和擴散模型（diffusion model）用的是同一套材料：線性層、非線性激活函數、正規化、殘差連接、損失函數、反向傳播，以及自適應最佳化器。把這些弄清楚之後，深度學習系列談的就只是如何為影像排列這些零件。

> **關於本篇**
>
> 本篇由作者先前的學習筆記〈機器學習及類神經網路筆記〉[1] 擴充改寫而成。閱讀前建議先熟悉 ML 01 的觀念：資料、特徵、損失函數與梯度下降。

感知器

白話版。感知器是最簡單的人工神經元。它把每個輸入乘上一個權重，加總後再加上偏差（bias），總和為正就回答「是」，否則回答「否」。它只在答錯時才微調權重，藉此學習。

模型

Rosenblatt 在 1958 年提出感知器，作為大腦如何儲存與辨識圖樣的模型 [4]。對輸入向量 $\mathbf{x} \in \mathbb{R}^d$，它計算

$$z = \mathbf{w}^\top \mathbf{x} + b, \qquad \hat{y} = \mathbf{1}[z > 0],$$

其中 $\mathbf{w} \in \mathbb{R}^d$ 是權重向量，$b \in \mathbb{R}$ 是偏差，z 是**激活前的值**（pre-activation，即加權和），$\mathbf{1}[\cdot]$ 是指示函數（條件成立時為 1，否則為 0），$\hat{y} \in \{0, 1\}$ 是預測類別。這個指示函數就是**階梯函數**（step function）。從幾何上看，$\mathbf{w}^\top \mathbf{x} + b = 0$ 是一個超平面（在二維中是一條直線），感知器依照點落在超平面的哪一側來分類。

感知器學習規則

把訓練範例 $(\mathbf{x}_i, y_i)$ 一個一個拿來看。每看一個，就算出 $\hat{y}_i$ 並更新

$$\mathbf{w} \leftarrow \mathbf{w} + \eta\,(y_i - \hat{y}_i)\,\mathbf{x}_i, \qquad b \leftarrow b + \eta\,(y_i - \hat{y}_i),$$

其中 $\eta > 0$ 是**學習率**（步長），$y_i \in \{0, 1\}$ 是真實標籤。預測正確時 $y_i - \hat{y}_i = 0$，什麼都不變。如果感知器答 0 但正確答案是 1，權重會往 $\mathbf{x}_i$ 的方向移動，下次遇到這個輸入時 z 就會變大；反方向的錯誤則讓權重遠離它。當兩個類別可以被一個超平面分開（資料**線性可分**）時，這個規則在有限次更新後就不會再犯錯；如果不可分，它永遠不會停下來。

```
import numpy as np

def perceptron_train(X, y, epochs=20, eta=1.0):
    """X: (N, d), y in {0, 1}. Returns w, b."""
    w, b = np.zeros(X.shape[1]), 0.0
    for _ in range(epochs):
        mistakes = 0
        for x_i, y_i in zip(X, y):
            y_hat = float(w @ x_i + b > 0)       # step function
            if y_hat != y_i:                     # update only on a mistake
                w += eta * (y_i - y_hat) * x_i
                b += eta * (y_i - y_hat)
                mistakes += 1
        if mistakes == 0:
            break
    return w, b

# AND is linearly separable; XOR is not
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=float)
for name, y in [("AND", [0, 0, 0, 1]), ("XOR", [0, 1, 1, 0])]:
```

```
    w, b = perceptron_train(X, np.array(y, float))
    print(name, (X @ w + b > 0).astype(int))
# AND [0 0 0 1]
# XOR [1 1 0 0]   <- wrong: no single line can do it
```

感知器與邏輯迴歸的差別

這兩個模型很容易混淆，因為第一步完全相同，都是 $z = \mathbf{w}^\top \mathbf{x} + b$。但它們是不同的方法：

	感知器	邏輯迴歸
輸出函數	階梯：$\hat{y} = \mathbf{1}[z > 0]$	sigmoid：$\hat{p} = \sigma(z) = \dfrac{1}{1+e^{-z}}$
輸出意義	0/1 的硬決策	機率 $P(y = 1 \mid \mathbf{x})$
訓練方式	感知器規則，只在答錯時更新	以梯度下降最小化交叉熵（最大概似）
資料不可分時	永不收斂	收斂到定義明確的最佳解

邏輯迴歸對單一範例的損失梯度是 $(\hat{p}_i - y_i)\,\mathbf{x}_i$，所以梯度下降的一步是 $\mathbf{w} \leftarrow \mathbf{w} + \eta\,(y_i - \hat{p}_i)\,\mathbf{x}_i$。這看起來幾乎和感知器規則一模一樣，但 $\hat{p}_i$ 是介於 0 與 1 之間的機率，因此**每一個**範例都有貢獻，而且非常有把握又答對的範例貢獻很小。正是這種平滑性，讓邏輯迴歸可以用梯度下降訓練，也讓它能被疊成更大的可微分網路。相對地，階梯函數幾乎處處導數為零，梯度根本無法通過它。

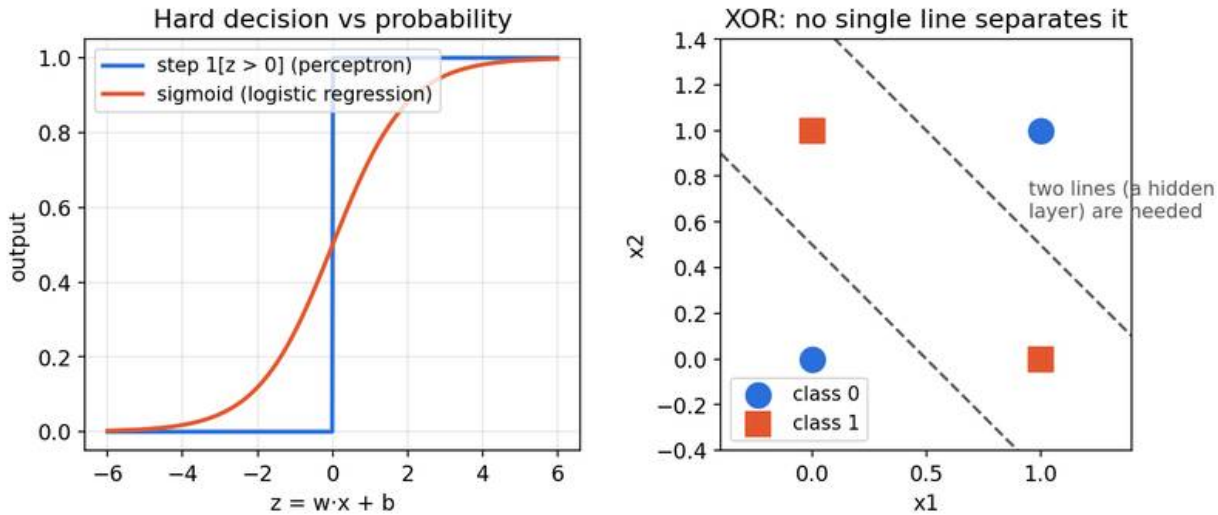


圖 *1* — 左：感知器的階梯輸出與邏輯迴歸的 *sigmoid* 機率。右：*XOR*，任何單一線性邊界都無法解決的經典問題；需要兩條邊界（也就是一個隱藏層）。

多層感知器（MLP）

白話版。各層結合加權總和與非線性激活，形成複雜決策區域；只堆疊線性映射，結果仍是線性映射。

把層疊起來

只有一個隱藏層的 MLP 計算

$$\mathbf{h} = f(W_1\mathbf{x} + \mathbf{b}_1), \\ \hat{\mathbf{y}} = g(W_2\mathbf{h} + \mathbf{b}_2),$$

其中 $\mathbf{x} \in \mathbb{R}^d$ 是輸入，$W_1 \in \mathbb{R}^{m\times d}$ 與 $\mathbf{b}_1 \in \mathbb{R}^m$ 是第一層的權重與偏差，m 是**寬度**（隱藏單元數），f 是逐元素作用的非線性**激活函數**（activation function），$\mathbf{h} \in \mathbb{R}^m$ 是隱藏表示，$W_2 \in \mathbb{R}^{K\times m}$ 與 $\mathbf{b}_2 \in \mathbb{R}^K$ 把它映射到 K 個輸出，g 則是依任務選擇的輸出函數（見後文）。每個隱藏單元都是一個「類似邏輯迴歸」的神經元；「多層感知器」這個名字是歷史沿用，現代 MLP 用的是平滑的激活函數，不是階梯函數。

非線性 f 是關鍵。少了它，$W_2(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = (W_2W_1)\mathbf{x} + (W_2\mathbf{b}_1 + \mathbf{b}_2)$ 又只是一個線性映射：再多層的線性層都會塌縮成一層，網路依然只能畫一個超平面。以 XOR 為例，兩個隱藏單元可以各畫出圖 1 中的一條虛線，輸出單元再回答「是否落在兩條線之間」。

通用近似

Cybenko 在 1989 年證明：只要隱藏層夠寬，一**個**由 sigmoid 型單元組成的隱藏層，就能在有界的方塊（單位超立方體）上以任意精度逼近任何連續函數 [5]。後續研究把結果推廣到其他激活函數，包括 ReLU。這個定理令人安心，但要仔細讀：

- 它講的是**存在性**。它只說合適的權重存在，完全沒說梯度下降找不找得到。
- 它沒說「夠寬」到底要**多寬**。對某些函數，所需的寬度會隨輸入維度極快地成長。
- 它沒談**泛化**：把訓練點擬合得很好，不代表能準確預測新的點。

所以這個定理說明了 MLP 在原理上夠有彈性，而本篇其餘部分談的，正是它留下來的那些實務問題。

深度神經網路：深度與寬度

白話版。「深」只是代表依序疊了很多層。並沒有一個官方的層數門檻，超過就算深。

深度神經網路（deep neural network, DNN）是具有多個隱藏層的 MLP 或其他層狀網路：$\mathbf{h}_\ell = f(W_\ell\mathbf{h}_{\ell-1} + \mathbf{b}_\ell)$，$\ell = 1, \dots, L$，且 $\mathbf{h}_0 = \mathbf{x}$。「深」沒有公認的門檻。常見的非正式說法是隱藏層超過一層就算深，但各教材計算總層數或隱藏層數的方式不同，所以「三層」在不同書裡指的東西也不一樣。重要的是概念而不是數字：深度網路是把很多簡單步驟**組合**（composition）起來得到答案。

增加模型能力有兩種方式：**寬度**（每層更多單元）與**深度**（更多層）。兩者並不等價。加寬是多放幾塊只組合一次的零件；加深則讓網路重複利用前面各層的特徵，建構出更抽象的特徵。LeCun、Bengio 與 Hinton 的回顧論文 [2] 正是以此解釋深度學習的成功：每一層都把前一層的表示轉換成稍微抽象一點的表示，疊很多這樣的層，網路就能直接從原始資料學到非常複雜的函數，而且特徵是學出來的，不是人工設計的。

從歷史來看，1990 年代以**淺層**方法為主流。支持向量機（SVM）是當時的主要工具，它是一種核方法（kernel method），並不是很寬的類神經網路；那時深度網路很難訓練。從 2000 年代中期開始，更多的資料、GPU 與更好的訓練技巧讓深度變得可行，深度網路開始在一個又一個基準測試中勝出 [2]。不過深度並不是全部：單純疊更多普通的層，最後反而會讓訓練**變差**，這個問題要靠殘差連接才得以解決（見最後一個主要章節）。

圖 2 以 scikit-learn 的雙月形（two moons）資料集展示寬度與深度的作用。一層只有 2 或 8 個隱藏單元時，邊界只由少數幾段直線組成，無法沿著月牙走。更寬或更深的網路能把邊界彎曲到貼合資料；最寬又最深的那個甚至繞著個別的雜訊點打轉，這是過擬合的第一個徵兆。

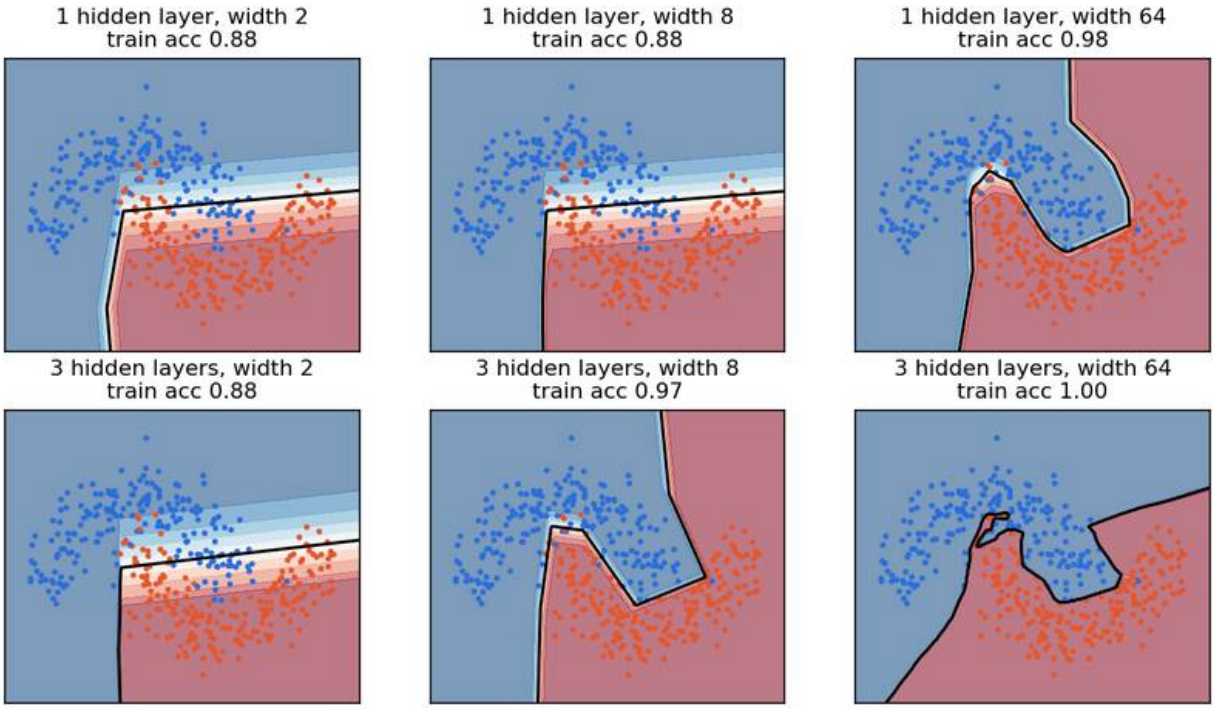


圖 *2 — ReLU MLP 在 make_moons（400 點、雜訊 0.2）上的決策邊界，欄為寬度、列為深度，每個模型都以 Adam 做 1,500 步全批次訓練。黑色曲線是機率 0.5 的等高線。*

激活函數

白話版。激活函數是每次加權和之後加上的「彎曲」。沒有它，整個網路就只是直線模型。不同的彎曲方式在平滑度、速度，以及梯度能否順利通過很多層之間取捨。

圖 3 畫出常見的選擇與它們的導數。導數和函數本身一樣重要，因為反向傳播會一層一層地把導數乘起來（見訓練章節）。

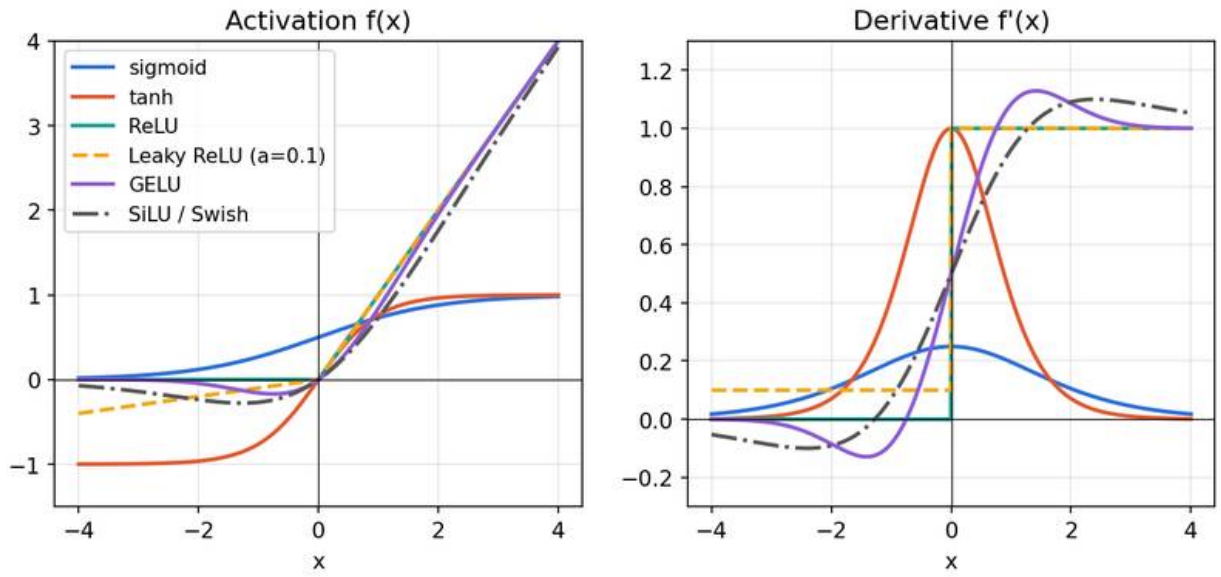


圖 3 — 常見激活函數（左）與其導數（右）。注意 *sigmoid* 的導數處處都很小（最多 *0.25*），而 *sigmoid* 和 *tanh* 在輸入很大時都變得非常平坦。

Sigmoid 與 tanh

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma'(x) = \sigma(x)(1-\sigma(x)); \qquad \tanh(x) = 2\sigma(2x) - 1, \quad \tanh'(x) = 1 - \tanh^2(x).$$

Sigmoid 把任何實數映射到 $(0,1)$；tanh 映射到 $(-1,1)$，並以零為中心。兩者都會**飽和**(saturate)：$|x|$ 很大時曲線很平，導數幾乎為零，梯度幾乎過不去。Sigmoid 的導數最大只有 $1/4$（在 $x=0$）。這兩件事讓深層 sigmoid 網路很難訓練，Glorot 與 Bengio 對此有詳細分析 [17]。今天 sigmoid 主要用在二元分類器的**輸出**與門控機制（例如下面的 SwiGLU），tanh 則出現在某些遞迴網路中。

Softmax

Softmax 把 K 個實數分數組成的向量 $\mathbf{z}$（稱為 **logits**）轉成機率分布：

$$p_k = \mathrm{softmax}(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad \frac{\partial p_k}{\partial z_j} = p_k(\delta_{kj} - p_j),$$

其中 $k = j$ 時 δ_{kj} 為 1，否則為 0。每個 p_k 都是正數，而且總和為 1。和本節其他函數不同，softmax 作用在**整個向量**上，而不是逐元素作用，所以幾乎只用在多類別分類器的輸出層（以及 attention 內部），不當作隱藏層的激活函數。寫程式時，先減去 $\max_j z_j$ 再取指數；結果不變，但可以避免溢位。

ReLU 與 Leaky ReLU

$$\mathrm{ReLU}(x) = \max(0, x), \quad \mathrm{ReLU}'(x) = \mathbf{1}[x > 0]; \qquad \mathrm{LeakyReLU}_a(x) = \max(ax, x), \quad 0 < a < 1.$$

整流線性單元（rectified linear unit, ReLU）讓正的輸入原封不動通過，其餘歸零。它對所有正輸入的導數恰好是 1，所以梯度通過活躍單元時不會縮小，而且計算成本極低。ReLU 是讓深層監督式網路不必靠非監督式預訓練也能訓練起來的改變之一 [2]。它的弱點是**死亡 ReLU**（dying ReLU）：如果某個單元對所有範例的輸入都是負的，它的梯度就是零，從此不再學習。Leaky ReLU 在負側保留一個小斜率 a（例如 0.01），讓一些梯度永遠能流過。Dubey、Singh 與 Chaudhuri 的激活函數綜述 [7] 把這些以及後來的許多變體歸為 ReLU 家族。

GELU 與 SiLU/Swish

這些是 ReLU 的平滑版本，會讓小的負值通過：

$$\mathrm{GELU}(x) = x\,\Phi(x), \qquad \mathrm{GELU}'(x) = \Phi(x) + x\,\varphi(x),$$

其中 Φ 是標準常態分布的累積分布函數（CDF），φ 是它的密度函數 [8]。GELU 不像 ReLU 依正負號硬切，而是依輸入的大小來加權：$\Phi(x)$ 是標準常態隨機變數小於 x 的機率。常用的快速近似是 $0.5x(1 + \tanh[\sqrt{2/\pi}\,(x + 0.044715x^3)])$ [8]。GELU 是許多 Transformer 模型的預設激活函數。

$$\mathrm{Swish}_\beta(x) = x\,\sigma(\beta x), \qquad \mathrm{SiLU}(x) = x\,\sigma(x), \qquad \mathrm{SiLU}'(x) = \sigma(x)(1 + x(1 - \sigma(x))).$$

sigmoid 加權線性單元（SiLU）由 Elfwing、Uchibe 與 Doya 為強化學習提出 [9]。Ramachandran、Zoph 與 Le 以自動搜尋找到同樣的形狀，β 可固定或可學習，並稱之為 Swish [10]；$\beta = 1$ 時 Swish 就是 SiLU。GELU 和 SiLU 長得非常像（圖 3）。兩者都不是單調函數：對中等大小的負輸入，它們會稍微低於零。

SwiGLU：帶門控的前饋區塊

門控線性單元（gated linear unit, GLU）把兩個線性投影逐元素相乘，其中一個先通過一個像閘門一樣的非線性函數。Shazeer 在 Transformer 的前饋（feed-forward）區塊中測試了幾種變體 [11]。SwiGLU 版本以論文的列向量寫法、且省略偏差，寫成

$$\mathrm{FFN}_{\mathrm{SwiGLU}}(\mathbf{x}) = (\mathrm{Swish}_1(\mathbf{x}W) \odot \mathbf{x}V)\,W_2,$$

其中 $\mathbf{x}$ 是大小為 d_{model} 的列向量，W 與 V 是 $d_{\text{model}} \times d_{\text{ff}}$ 矩陣，W_2 是 $d_{\text{ff}} \times d_{\text{model}}$ 矩陣，$\odot$ 是逐元素相乘。在 Shazeer 的實驗中，包括 SwiGLU 與 GEGLU 在內的 GLU 變體，表現都優於該區塊的 ReLU 與 GELU 版本。由於 SwiGLU 有三個權重矩陣而不是兩個，d_{ff} 要縮小為原本的 2/3，參數量才會相同 [11]。例如 LLaMA 就以 SwiGLU 取代 ReLU，隱藏維度用 $\frac{2}{3} \cdot 4d$ [12]；SwiGLU 已成為大型語言模型的常見預設。

激活函數	值域	平滑	目前的典型用途
sigmoid	$(0, 1)$	是	二元／多標籤輸出、門控
tanh	$(-1, 1)$	是	某些遞迴網路
softmax	機率向量	是	多類別輸出、attention 權重

激活函數	值域	平滑	目前的典型用途
ReLU / Leaky ReLU	$[0,\infty)$ / $\mathbb{R}$	否（0 處有折角）	CNN、MLP，安全的預設
GELU、SiLU	約 $[-0.17,\infty)$、$[-0.28,\infty)$	是	Transformer、現代 CNN
SwiGLU	$\mathbb{R}$	是	Transformer 的前饋區塊

正規化層

白話版。訓練進行中，網路內部流動的數值可能飄到很大或很小的尺度，讓學習變得不穩定。正規化層（normalization）在每一步把它們重新縮放到標準大小，再讓網路自己選擇偏好的最終尺度。

三種常見的正規化層共用同一個模板。對一組數值 $x_1, \dots, x_n$，

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2, \qquad y_i = \gamma\,\frac{x_i-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta,$$

其中 μ 與 σ^2 是平均與變異數，ϵ 是為了數值安全的小常數，γ, β 是可學習的縮放與平移（如果有幫助，網路可以藉此還原正規化）。三者的差別在於**哪些數值**被放在同一組（圖 4）。

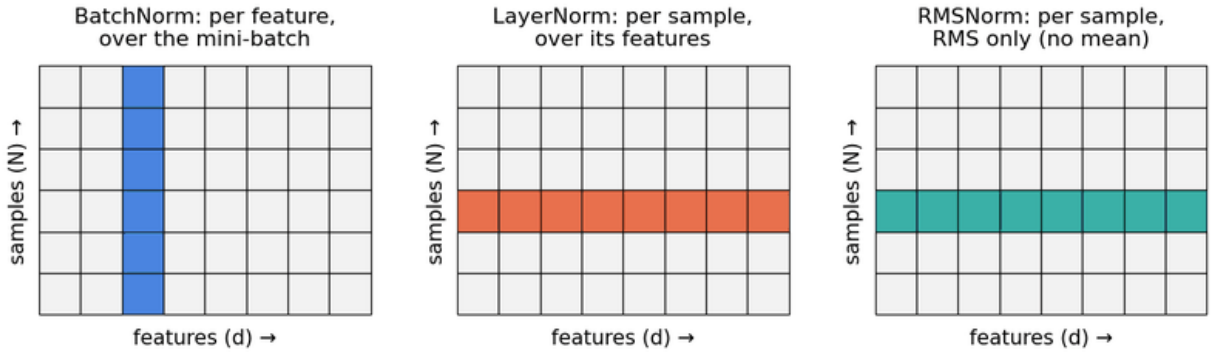


圖 4 — 哪些元素共用統計量。*BatchNorm* 對每個特徵在 *mini-batch* 上正規化；*LayerNorm* 與 *RMSNorm* 對每個樣本在它自己的特徵上正規化。

- **BatchNorm**（Ioffe 與 Szegedy [13]）對每個特徵（在 CNN 中是每個通道）**跨 mini-batch** 計算 μ, σ^2。它能加快訓練，成為 CNN 的標準配置。由於統計量取決於 batch，它在訓練與推論時的行為不同：推論時改用訓練期間累積的移動平均。batch 很小時統計量變得很吵，效果也會變差。
- **LayerNorm**（Ba、Kiros 與 Hinton [14]）**對單一樣本的所有特徵**計算 μ, σ^2。它與 batch 無關，訓練和測試時行為一致，也很自然地適用於序列。它是 Transformer 的標準正規化方式。
- **RMSNorm**（Zhang 與 Sennrich [15]）省略減去平均的步驟，只除以均方根：$y_i = \gamma_i\, x_i / \sqrt{\frac{1}{n}\sum_j x_j^2 + \epsilon}$。作者假設 LayerNorm 的重新置中（re-centering）是可有可無的，並報告品質相當而執行時間更短 [15]。LLaMA 採用 RMSNorm，並把它放在每個 Transformer 子層的**輸入端**（pre-normalization），而不是輸出端 [12]。

```
import torch

x = torch.randn(4, 8) * 5 + 3                       # 4 samples, 8 features
ln = torch.nn.LayerNorm(8, elementwise_affine=False)
bn = torch.nn.BatchNorm1d(8, affine=False)
rms = x / torch.sqrt(x.pow(2).mean(dim=1, keepdim=True) + 1e-6)

print(ln(x).mean(dim=1))     # ~0 for every sample (row)
print(bn(x).mean(dim=0))     # ~0 for every feature (column), in train mode
print(rms.pow(2).mean(dim=1))  # ~1 for every sample: unit RMS, mean not removed
```

Huang 等人 [16] 的綜述整理了這個家族，並把任何正規化方法拆成三個選擇：哪些數值放在一起（正規化區域）、做什麼運算（標準化、置中、縮放或白化），以及之後如何還原表示（可學習的仿射步驟）。BatchNorm、LayerNorm 與 RMSNorm 主要差在前兩項。

設計輸出層與選擇損失函數

白話版。最後一層必須說答案的語言：數量就輸出任意實數，是非題就輸出機率，「K 選一」就輸出機率分布。每種輸出都有一個天生搭配的損失函數，下表就是該用的組合。

任務	輸出層	輸出範圍	損失函數
迴歸	線性（不加激活函數）	$\mathbb{R}$	均方誤差（MSE，L2）或平均絕對誤差（MAE，L1）
二元分類	1 個單元 + sigmoid	$(0,1)$	二元交叉熵（binary cross-entropy）
多標籤（多個是非題）	K 個單元，各接 sigmoid	$(0,1)^K$	每個標籤各算二元交叉熵
多類別（K 選一）	K 個單元 + softmax	機率向量	交叉熵（cross-entropy）

常見的錯誤是在迴歸輸出接 sigmoid。sigmoid 的值域是開區間 $(0,1)$，它的工作是為二元或多標籤分類輸出機率。迴歸的輸出層通常是**線性**的，才能輸出任何實數。只有當目標值本身已知落在 0 到 1 之間（例如正規化後的像素強度或比例）時，迴歸輸出接 sigmoid 才合理。

對單一範例，損失函數為

$$\begin{aligned} \text{MSE：} \quad & \ell = \tfrac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2, \\ \text{二元交叉熵：} \quad & \ell = -[\, y \log \hat{p} + (1-y)\log(1-\hat{p}) \,], \\ \text{交叉熵：} \quad & \ell = -\sum_{k=1}^{K} y_k \log p_k = -\log p_c, \end{aligned}$$

其中 $\mathbf{y}$ 是目標，$\hat{p} = \sigma(z)$，$\mathbf{p} = \mathrm{softmax}(\mathbf{z})$，$y_k$ 是 one-hot 標籤，c 是正確類別。（MSE 的 $\frac{1}{2}$ 只是為了方便；PyTorch 的 `MSELoss` 取平均且不含這個因子。）MAE 即 $\|\hat{\mathbf{y}} - \mathbf{y}\|_1$，比 MSE 更不受離群值影響。每一組搭配都是最大概似估計：MSE 對應目標上的高斯雜訊，二元交叉熵對應 Bernoulli 標籤，交叉熵對應類別（categorical）標籤 [3]。

這些搭配還有一個漂亮的實務理由。三種情況下，損失對輸出層**激活前的值**的梯度都只是

$$\frac{\partial \ell}{\partial \mathbf{z}} = \hat{\mathbf{y}} - \mathbf{y} \quad \text{（預測減目標）},$$

這裡的 $\hat{\mathbf{y}}$ 分別指線性輸出、$\sigma(z)$ 或 softmax$(\mathbf{z})$。會飽和的 sigmoid 或 softmax 導數被抵消掉了，所以即使模型非常有把握地答錯，梯度仍然很大。（練習 3 請你為 softmax 驗證這件事。）這也是為什麼框架會把最後的激活函數併進損失函數：PyTorch 的 `nn.CrossEntropyLoss` 與 `nn.BCEWithLogitsLoss` 接收的是 **logits**，所以模型本身在訓練時不能以 softmax 或 sigmoid 結尾。

訓練（一）：前向傳遞與反向傳播

白話版。前向傳遞計算預測與損失；反向傳播重用中間值，透過連鎖律計算參數導數。

Rumelhart、Hinton 與 Williams 讓反向傳播成為訓練多層網路的普及方法 [6]。它其實就是微積分的連鎖律，只是安排得讓任何計算都不重複。圖 5 畫出我們要微分的網路的計算圖。

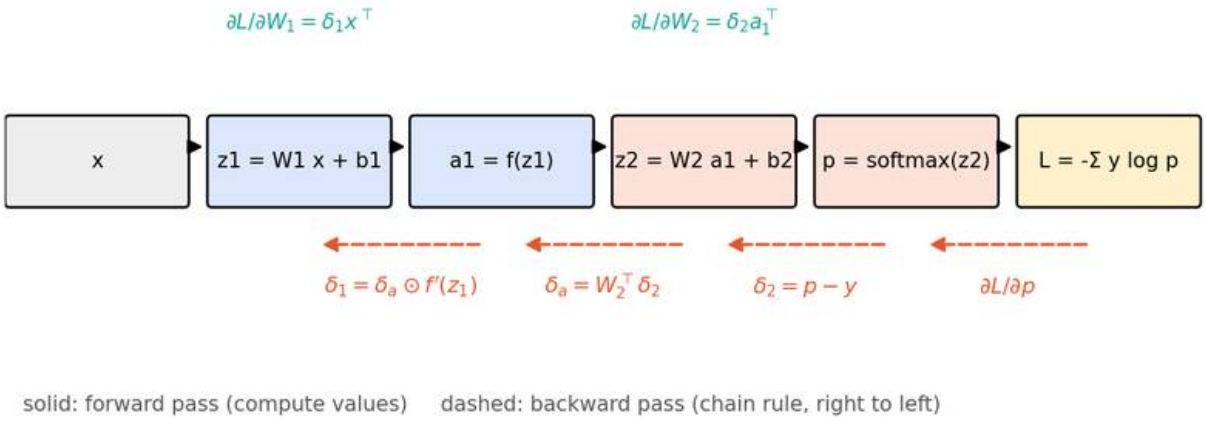


圖 5 — 搭配 *softmax* 交叉熵的兩層 *MLP* 的計算圖。實線：前向傳遞。虛線：反向傳遞，把誤差訊號 δ 從損失往輸入方向傳回去。

兩層 MLP 的推導

取一個範例 $(\mathbf{x}, \mathbf{y})$，$\mathbf{x} \in \mathbb{R}^d$，$\mathbf{y} \in \{0,1\}^K$ 是 one-hot 標籤。前向傳遞為

$$\begin{aligned} \mathbf{z}_1 &= W_1\mathbf{x} + \mathbf{b}_1 \in \mathbb{R}^m, \quad \mathbf{a}_1 = f(\mathbf{z}_1) \in \mathbb{R}^m, \\ \mathbf{z}_2 &= W_2\mathbf{a}_1 + \mathbf{b}_2 \in \mathbb{R}^K, \quad \mathbf{p} = \mathrm{softmax}(\mathbf{z}_2), \qquad L = -\textstyle\sum_k y_k \log p_k, \end{aligned}$$

其中 $W_1 \in \mathbb{R}^{m\times d}$、$W_2 \in \mathbb{R}^{K\times m}$，$f$ 是逐元素的激活函數。定義某一層的**誤差訊號**為損失對該層激活前的值的梯度：$\delta_\ell = \partial L/\partial \mathbf{z}_\ell$。

步驟 1：輸出層。由上一節可知，softmax 搭配交叉熵給出

$$\delta_2 = \frac{\partial L}{\partial \mathbf{z}_2} = \mathbf{p} - \mathbf{y} \in \mathbb{R}^K.$$

步驟 2：輸出層權重。因為 $z_{2,k} = \sum_j W_{2,kj}\, a_{1,j} + b_{2,k}$，所以 $\partial z_{2,k}/\partial W_{2,kj} = a_{1,j}$，於是

$$\frac{\partial L}{\partial W_2} = \delta_2\, \mathbf{a}_1^\top \in \mathbb{R}^{K\times m}, \qquad \frac{\partial L}{\partial \mathbf{b}_2} = \delta_2.$$

一個權重的梯度等於（它輸出端的誤差）乘以（它輸入端的激活值）。形狀和 W_2 相同，這是很好用的檢查。

步驟 3：往回穿過線性層。每個隱藏激活值 $a_{1,j}$ 都影響所有輸出，所以它的梯度要對所有輸出加總：$\partial L/\partial a_{1,j} = \sum_k \delta_{2,k} W_{2,kj}$，寫成矩陣形式為

$$\frac{\partial L}{\partial \mathbf{a}_1} = W_2^\top \delta_2 \in \mathbb{R}^m.$$

步驟 4：往回穿過激活函數。因為 f 逐元素作用，它的 Jacobian 是對角矩陣，連鎖律就變成逐元素相乘：

$$\delta_1 = \frac{\partial L}{\partial \mathbf{z}_1} = (W_2^\top \delta_2) \odot f'(\mathbf{z}_1).$$

步驟 5：第一層權重。和步驟 2 完全一樣，

$$\frac{\partial L}{\partial W_1} = \delta_1\, \mathbf{x}^\top \in \mathbb{R}^{m\times d}, \qquad \frac{\partial L}{\partial \mathbf{b}_1} = \delta_1.$$

更深的網路只是重複步驟 3 到 5：$\delta_\ell = (W_{\ell+1}^\top \delta_{\ell+1}) \odot f'(\mathbf{z}_\ell)$，以及 $\partial L/\partial W_\ell = \delta_\ell\, \mathbf{a}_{\ell-1}^\top$。對 N 個範例組成的 mini-batch，損失取平均，所以每個梯度也是各範例梯度的平均。

用 NumPy 從零實作，並做梯度檢查

下面的程式把範例存成**列**（`X` 的形狀是 $N \times d$），這是程式中常見的慣例。於是 `X @ W1`（`W1` 形狀為 $d\times m$）扮演數學式中 $W_1\mathbf{x}$ 的角色，上面每個公式在程式裡都以轉置的形式出現：例如 `dW2 = A1.T @ dZ2` 就是把 $\delta_2\mathbf{a}_1^\top$ 沿 batch 疊起來。**梯度檢查**把一個解析導數和中央差分 $(L(w+\epsilon)-L(w-\epsilon))/2\epsilon$ 比較；如果兩者在很多位數上都一致，反向傳遞就是對的。只要親手寫梯度，就一定要做這個檢查。

```
import numpy as np
from sklearn.datasets import make_moons

rng = np.random.default_rng(0)
X, y = make_moons(n_samples=200, noise=0.2, random_state=0)
N, d, h, K = X.shape[0], 2, 16, 2
Y = np.eye(K)[y]                                  # one-hot targets, (N, K)

# He-style initialization (std = sqrt(2 / fan_in))
W1 = rng.normal(0, np.sqrt(2 / d), (d, h)); b1 = np.zeros(h)
W2 = rng.normal(0, np.sqrt(2 / h), (h, K)); b2 = np.zeros(K)

def forward(X, W1, b1, W2, b2):
    Z1 = X @ W1 + b1                              # (N, h)
```

```
    A1 = np.maximum(Z1, 0)                          # ReLU
    Z2 = A1 @ W2 + b2                               # (N, K) logits
    Z2 = Z2 - Z2.max(axis=1, keepdims=True)         # numerical stability
    P = np.exp(Z2) / np.exp(Z2).sum(axis=1, keepdims=True)
    loss = -np.mean(np.sum(Y * np.log(P + 1e-12), axis=1))
    return Z1, A1, P, loss

def backward(X, Z1, A1, P, W2):
    dZ2 = (P - Y) / N                               # delta at the output
    dW2 = A1.T @ dZ2;  db2 = dZ2.sum(axis=0)
    dA1 = dZ2 @ W2.T
    dZ1 = dA1 * (Z1 > 0)                            # ReLU'(z) = 1[z > 0]
    dW1 = X.T @ dZ1;   db1 = dZ1.sum(axis=0)
    return dW1, db1, dW2, db2

# Gradient check on one entry of W1 (central difference)
Z1, A1, P, loss = forward(X, W1, b1, W2, b2)
dW1, db1, dW2, db2 = backward(X, Z1, A1, P, W2)
eps = 1e-6; E = np.zeros_like(W1); E[0, 3] = eps
num = (forward(X, W1 + E, b1, W2, b2)[3] - forward(X, W1 - E, b1, W2, b2)[3]) / (2 * eps)
print(f"analytic {dW1[0, 3]:.8f}  numeric {num:.8f}")

# Plain gradient descent
lr = 0.5
for step in range(2000):
    Z1, A1, P, loss = forward(X, W1, b1, W2, b2)
    dW1, db1, dW2, db2 = backward(X, Z1, A1, P, W2)
    W1 -= lr * dW1; b1 -= lr * db1; W2 -= lr * dW2; b2 -= lr * db2
acc = (P.argmax(axis=1) == y).mean()
print(f"final loss {loss:.3f}, train accuracy {acc:.3f}")
# analytic -0.01941581  numeric -0.01941581
# final loss 0.063, train accuracy 0.970
```

實務上你永遠不必自己寫反向傳遞：PyTorch 等框架會記錄前向計算，在你呼叫 `loss.backward()` 時自動執行完全相同的程序。但親手寫過一次，仍是理解這個呼叫在做什麼、以及為什麼計算圖中某處導數為零時它會失靈的最好方法。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

訓練（二）：讓最佳化真正可行

白話版。只知道梯度還不夠。你還需要一個合理的起點（初始化）、把梯度變成步伐的規則（最佳化器）、隨時間改變的步長（排程）、每一步看多少範例的選擇（batch size），以及防止偶發超大步伐的保護（裁剪）。

梯度消失與梯度爆炸

把遞迴式 $\delta_\ell = (W_{\ell+1}^\top \delta_{\ell+1}) \odot f'(\mathbf{z}_\ell)$ 沿 L 層展開，第一層的梯度會包含 L 個權重矩陣與 L 個激活函數導數的**連乘**。如果典型的因子小於 1，連乘會隨深度指數縮小，前面的層幾乎學不到東西：這

就是**梯度消失**（vanishing gradient）。用 sigmoid 時每個 f' 最多 0.25，十層就可能把訊號縮小到 $0.25^{10} \approx 10^{-6}$。如果典型因子大於 1，例如初始權重太大，連乘會指數成長，更新變得巨大，損失變成 `inf` 或 `NaN`：這就是**梯度爆炸**（exploding gradient）。

下面介紹的補救方法，從不同方向處理這個連乘：

- 在大範圍內導數為 1 的激活函數（ReLU 與它的平滑親戚）；
- 讓訊號大小逐層保持不變的初始化；
- 重新標準化激活值的正規化層；
- **殘差連接**，給梯度一條因子為 1 的通道（最後一個主要章節）；
- **梯度裁剪**，對付爆炸的標準緊急煞車。

有時也會列出其他做法，例如微調（fine-tuning）預訓練模型或改用較小的網路；它們讓整體訓練變得比較容易，但並不直接解決梯度問題。

權重初始化

如果所有權重一開始都相同，每個隱藏單元算的東西都一樣、拿到的梯度也一樣，它們就永遠一模一樣。所以權重要隨機初始化。尺度很重要：我們希望激活值（前向）與梯度（反向）的變異數在層與層之間大致維持不變。對有 n_{in} 個輸入、n_{out} 個輸出的線性層：

- **Xavier（Glorot）初始化** [17] 為 tanh 等對稱激活函數平衡前向與反向條件：$\mathrm{Var}(W_{ij}) = \dfrac{2}{n_{\text{in}} + n_{\text{out}}}$，例如在 $\left[-\sqrt{6/(n_{\text{in}}+n_{\text{out}})},\ \sqrt{6/(n_{\text{in}}+n_{\text{out}})}\right]$ 上均勻分布。
- **He（Kaiming）初始化** [18] 考慮到 ReLU 會把約一半的輸入歸零，使每層的變異數減半，因此把尺度加倍：$\mathrm{Var}(W_{ij}) = \dfrac{2}{n_{\text{in}}}$，例如 $W_{ij} \sim \mathcal{N}(0, 2/n_{\text{in}})$。

偏差通常從零開始。He 等人證明這種初始化能讓非常深的 ReLU 網路從零開始訓練 [18]。上面的 NumPy 範例就是用它。

最佳化器

下面每個最佳化器都是**隨機梯度下降**（stochastic gradient descent, SGD）的變體：在一個小的隨機 mini-batch 上估計梯度，往反方向走一步，然後重複。記 $\mathbf{g}_t = \nabla_\theta L_{\text{batch}}(\theta_t)$ 為第 t 步的 mini-batch 梯度，θ 為所有參數，η 為學習率。

SGD。$\theta_{t+1} = \theta_t - \eta\,\mathbf{g}_t$。簡單、不需額外記憶體，但在狹長的山谷中會來回鋸齒狀前進，在平坦方向上又走得很慢。

SGD with momentum。維持一個移動的速度：$\mathbf{v}_{t+1} = \mu\,\mathbf{v}_t + \mathbf{g}_t$，$\theta_{t+1} = \theta_t - \eta\,\mathbf{v}_{t+1}$，$\mu$ 約為 0.9。就像一顆沉重的球滾下山坡，它把鋸齒平均掉，並在方向一致時累積速度。

Adam（Kingma 與 Ba [19]）為每個參數維持兩個移動平均，分別是梯度與梯度平方：

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\,\mathbf{g}_t, \quad \mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\,\mathbf{g}_t^2,$$
$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}, \qquad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}, \qquad \theta_t = \theta_{t-1} - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t}+\epsilon},$$

其中平方、開根號與除法都是逐元素運算，$\beta_1 \approx 0.9$、$\beta_2 \approx 0.999$ 是衰減率，ϵ 是小常數。第一個平均就是 momentum。除以第二個平均的平方根，讓每個參數有自己的步長，這是 Adam 所借鏡的 RMSprop 的概念 [19]：梯度一直很大的參數走小步，很少被更新的參數走大步。**偏差修正**（bias correction）$1/(1-\beta^t)$ 是因為兩個平均都從零開始，若不修正，在最初幾步會太小。

AdamW（Loshchilov 與 Hutter [20]）。權重衰減（weight decay）在每一步把權重往零縮。對單純的 SGD，這等同於在損失中加入 L2 懲罰 $\frac{\lambda}{2}\|\theta\|^2$。對 Adam 則不然：L2 項會進入 $\mathbf{g}_t$，再被 $\sqrt{\hat{\mathbf{v}}_t}$ 除掉，於是梯度大的權重反而被正則化得**比較少**。AdamW 把衰減和自適應步伐**解耦**：

$$\theta_t = \theta_{t-1} - \eta\left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t}+\epsilon} + \lambda\,\theta_{t-1}\right).$$

作者證明這能改善 Adam 的泛化能力 [20]，AdamW 現在是訓練 Transformer 與大型語言模型最常用的最佳化器。在 PyTorch 中，`torch.optim.Adam(..., weight_decay=λ)` 是耦合的 L2 版本，`torch.optim.AdamW` 才是解耦版本。

較新的最佳化器（簡述）。有兩個近年的選項值得認識。**Lion** 是透過自動化的程式搜尋找到的 [21]：它只保留一個 momentum 緩衝區（所以比 Adam 省記憶體），並以內插後 momentum 的**正負號**（sign）更新每個參數，因此所有更新的大小都相同；它需要比 AdamW 更小的學習率。**Muon** [22] 作用在隱藏層的二維權重矩陣上：它取 momentum 更新，用幾次 Newton–Schulz 迭代把它近似正交化；嵌入層、輸出層、偏差以及其他純量或向量參數仍以 AdamW 訓練。Liu 等人報告，在加入權重衰減與逐參數更新尺度調整之後，Muon 在他們的運算最適（compute-optimal）大型語言模型訓練實驗中，達到約 AdamW 兩倍的運算效率 [23]。這些都還是活躍的研究主題；AdamW 仍是安全的預設。

學習率排程

學習率通常是最重要的超參數。訓練初期合適的學習率，到後期常常太大，因為模型那時需要小而謹慎的步伐，所以學習率會隨時間改變（圖 6）。

- **階梯衰減**（step decay）：在固定的 epoch 把學習率乘上 0.1 之類的因子。以 SGD 訓練 CNN 的傳統做法。
- **餘弦退火**（cosine annealing）：$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\pi t/T))$，在 T 步內從 $\eta_{\max}$ 平滑降到 $\eta_{\min}$；SGDR [24] 把它和週期性的熱重啟（warm restart）結合。
- **暖身**（warmup）：在最初幾百或幾千步內，把學習率從接近零線性升高。訓練初期 Adam 的二階動差估計很吵，權重也離任何好區域都很遠，直接用全尺寸的步伐可能造成不穩定。

以一個已發表的大型模型為具體例子：LLaMA 使用 AdamW（$\beta_1 = 0.9$、$\beta_2 = 0.95$）、權重衰減 0.1、梯度裁剪 1.0、2,000 步暖身，以及結束於峰值 10% 的餘弦排程 [12]。

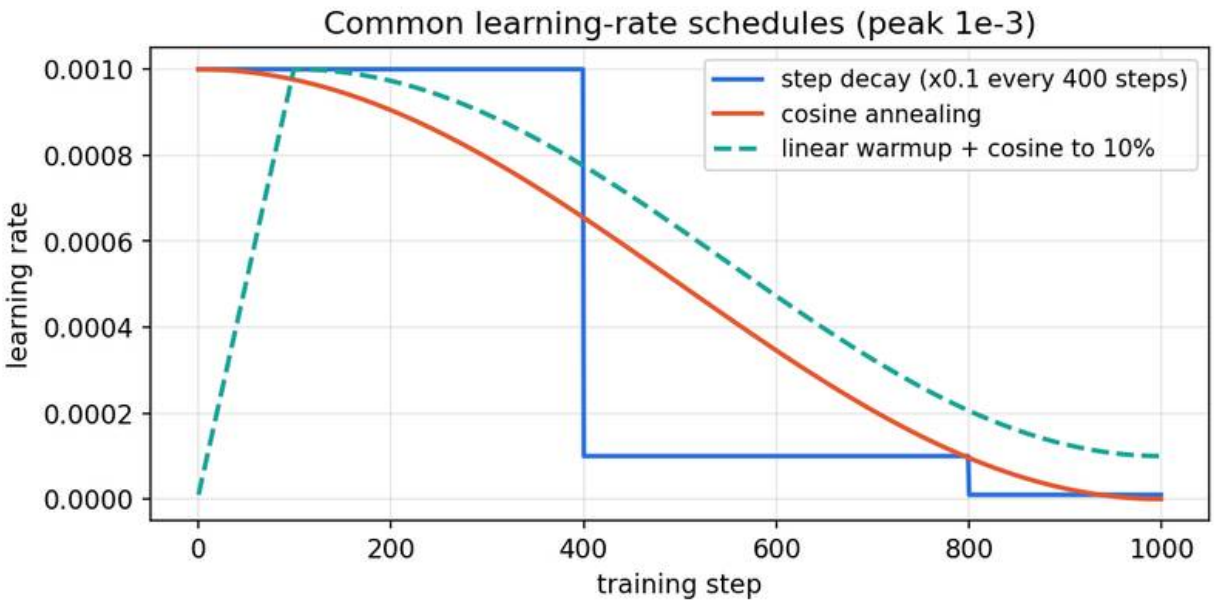


圖 *6* — 三種常見的學習率排程，共 *1,000* 步，峰值 *1e-3*。

Mini-batch、iteration 與 epoch

Mini-batch（常簡稱 batch）是用來做一次梯度估計的 B 個隨機範例。一次 **iteration**（迭代）是一次參數更新，使用一個 mini-batch。一個 **epoch** 是把整個訓練集完整走過一遍，對 N 個範例需要 $\lceil N/B \rceil$ 次 iteration。例如 $N = 50{,}000$、$B = 128$ 時，每個 epoch 有 391 次 iteration。

Batch size B 同時影響速度與結果，而且**並非越大越好**：

- **小 batch** 的梯度估計很吵（mini-batch 平均的變異數大約以 $1/B$ 下降），所以損失曲線跳動，每個 epoch 要走很多步，硬體利用率也差。
- **大 batch** 的梯度穩定，在 GPU 上平行效率高，但每個 epoch 的更新次數較少，而且在其他設定不變時往往泛化較差。Keskar 等人發現，大 batch 訓練傾向收斂到**尖銳極小值**（sharp minima），也就是解附近損失急遽上升的地方；小 batch 訓練則因為梯度雜訊，傾向找到泛化較好的**平坦極小值**（flat minima）[25]。

實務上，選一個能舒服放進記憶體、而且仍然訓練得好的最大 batch；改變 B 時要重新調整學習率，兩者交互作用很強。

梯度裁剪

即使網路設計得很好，偶爾某個 batch 也會產生巨大的梯度。**梯度範數裁剪**（gradient-norm clipping）在梯度範數超過門檻 c 時把整個梯度縮小：

$$\mathbf{g} \leftarrow \mathbf{g} \cdot \min\left(1, \frac{c}{\|\mathbf{g}\|}\right),$$

其中 $\|\mathbf{g}\|$ 是把所有參數梯度串接起來後的範數。它保留方向、限制步伐大小，是對付梯度爆炸的標準防線，在遞迴網路與 Transformer 中尤其常用 [3]。門檻 1.0 是常見的選擇。

在 PyTorch 中組合起來

下面的程式以 mini-batch、AdamW、餘弦排程與梯度裁剪，在 `make_moons` 上訓練一個小型 MLP。在 CPU 上幾秒鐘就跑完。

```
import torch, torch.nn as nn
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split

torch.manual_seed(0)
X, y = make_moons(n_samples=1000, noise=0.25, random_state=0)
Xtr, Xva, ytr, yva = train_test_split(X, y, test_size=0.3, random_state=0)
Xtr, Xva = torch.tensor(Xtr, dtype=torch.float32), torch.tensor(Xva, dtype=torch.float32)
ytr, yva = torch.tensor(ytr), torch.tensor(yva)

model = nn.Sequential(
    nn.Linear(2, 64), nn.ReLU(),
    nn.Linear(64, 64), nn.ReLU(),
    nn.Linear(64, 2),               # logits: no softmax here
)
loss_fn = nn.CrossEntropyLoss()     # applies log-softmax internally
opt = torch.optim.AdamW(model.parameters(), lr=1e-2, weight_decay=1e-2)
sched = torch.optim.lr_scheduler.CosineAnnealingLR(opt, T_max=200)

for epoch in range(200):
    model.train()
    perm = torch.randperm(len(Xtr))
    for i in range(0, len(Xtr), 64):                # mini-batches of 64
        idx = perm[i:i + 64]
        loss = loss_fn(model(Xtr[idx]), ytr[idx])
        opt.zero_grad()
        loss.backward()                              # backpropagation
        torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        opt.step()
    sched.step()

model.eval()
with torch.no_grad():
    acc = (model(Xva).argmax(dim=1) == yva).float().mean().item()
print(f"validation accuracy: {acc:.3f}")   # about 0.94
```

注意 `model.train()` 與 `model.eval()`：它們讓 dropout、BatchNorm 等層在訓練行為與推論行為之間切換。測試時忘了 `eval()` 是經典的 bug。

僅線上版：*Adam*：自適應更新與收斂論證的限制 — *https://blog.shyandram.dev/zh-tw/learn/ml/neural-networks/papers/adam 自適應更新與收斂論證的限制/*

泛化：欠擬合、過擬合與正則化

白話版。比較訓練與驗證表現；訓練誤差高可能源於容量或最佳化，驗證落差大則提示泛化不佳。

模型能力與兩種失敗模式

模型的**能力**（capacity）粗略地說就是它能擬合多複雜的函數；它隨參數量、深度、寬度與訓練時間增加。能力提高時，訓練誤差持續下降，驗證誤差則先降後升，因為模型開始擬合雜訊。兩者之間的差距稱為**泛化差距**（generalization gap）[3]。

- **欠擬合**：**訓練**誤差本身就很高。模型或訓練程序抓不到資料的規律。
- **過擬合**：訓練誤差很低，但驗證或測試誤差高得多。

處理欠擬合，依序是：

1. **先檢查程式與資料**。 bug、輸入與標籤沒對齊、前處理錯誤，或損失套在錯誤的張量上，看起來都像欠擬合。
2. **提高模型能力**：更寬或更深的模型，或更好的架構。
3. **改善最佳化**：訓練更久、調整學習率與排程、換最佳化器、檢查初始化與正規化。

把模型**變小**永遠不能解決欠擬合，那只會讓能力更低。在準確率已經足夠之後，為了速度或記憶體選擇較小的模型是合理的，但那是效率上的考量，不是解方。

處理過擬合：用更多或更多樣的資料、加入正則化（見下文）、降低能力，以及提早停止。也值得確認訓練集和驗證集真的來自同一個分布；兩者不一致時看起來也像過擬合，但需要的是修正資料，而不是正則化。

讀懂損失曲線

把訓練損失與驗證損失都對 iteration 或 epoch 作圖。下面四種情況涵蓋大部分狀況：

- **A. 兩條曲線一開始就是平的**。如果損失很高，模型根本沒在學：懷疑 bug、學習率太小或太大，或模型與資料不匹配。如果損失已經很低，可能是任務非常簡單，或每個記錄點之間隔了太多 iteration（一次記錄涵蓋的資料太多），把曲線的形狀藏起來了；請更頻繁地記錄。
- **B. 訓練損失持續下降，驗證損失轉而上升**。典型的過擬合。保留驗證損失最低時的參數（提早停止），並加入正則化。
- **C. 兩條曲線都下降，並在彼此接近處持平**。可能模型已接近最佳，這在訓練與驗證資料非常相似時很常見；或兩者都還很高，那就是欠擬合：訓練更久或提高能力。
- **D. 驗證損失高於訓練損失，但兩者都穩定下降**。正常、健康的情況。繼續訓練，並保留驗證損失最低的檢查點。

正則化工具

圖 7 刻意製造過擬合，並展示兩種解法。一個很寬的 MLP（兩個各 256 單元的隱藏層）只用 100 個有雜訊的雙月形資料點訓練。沒有正則化時，訓練損失降向零，驗證損失則很早就到達最低點，接著急遽上升。權重衰減與 dropout 都大幅減緩了這個上升。

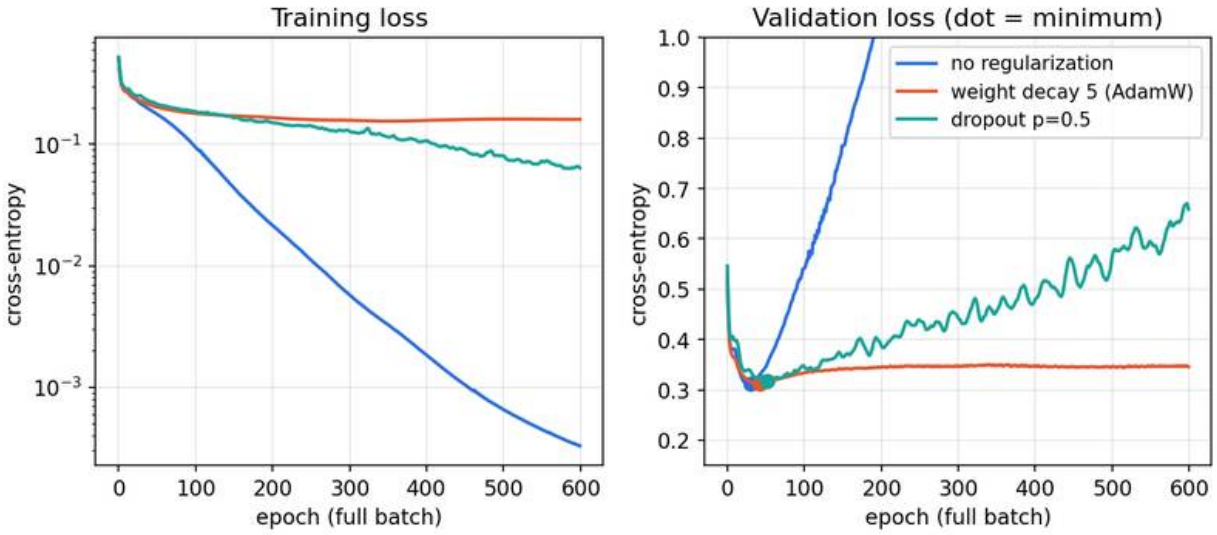


圖 *7* — 以 *make_moons*（雜訊 *0.35*）的 *100* 個訓練點製造過擬合，並以 *2,000* 個驗證點評估。全批次 *AdamW*，學習率 *3e-3*。圓點標示每次執行的最低驗證損失，也就是提早停止會保留模型的位置。

權重衰減（L2 正則化）。在損失中加入 $\frac{\lambda}{2}\|\theta\|^2$ 來懲罰大權重；使用 Adam 時則改用解耦的衰減（AdamW）。大權重讓網路能圍著個別資料點畫出尖銳、彎曲的邊界；讓權重保持小，就偏好較平滑的函數。在圖 7 中，$\lambda = 5$ 的解耦衰減讓驗證損失在最低點之後幾乎持平。

Dropout（Srivastava 等人 [27]）。訓練時，每個隱藏單元以機率 p 被設為零，每個範例、每一步都獨立抽樣，存活的單元再乘上 $1/(1-p)$，讓激活值的期望不變（「inverted dropout」）。測試時使用所有單元，不做縮放。於是每一步訓練的都是一個不同的「變瘦」子網路；單元無法依賴特定的夥伴，必須學出本身就有用的特徵，而測試時的完整網路就像許多瘦網路的平均 [27]。這也是 `model.eval()` 很重要的原因。

資料擴增（data augmentation）。用不改變標籤的轉換產生新的訓練範例：對影像做翻轉、裁切、小角度旋轉、色彩抖動與加雜訊。它常常是影像領域最有效的單一正則化方法，因為它直接教模型該有的不變性。但它必須符合任務：水平翻轉對貓沒問題，對辨識文字就不行。

提早停止（early stopping）。訓練時監看驗證損失，保留它最低那個 epoch 的參數；如果連續若干個 epoch（「patience」）都沒有改善就停止。它不花成本、不必修改模型，並藉由限制參數能離初始值多遠而發揮正則化作用 [3]。

更多資料與更好的策略。原始筆記列出的其他手段值得再強調：更多訓練資料是對付過擬合最可靠的方法；更強的架構（更深，或結構更好以利收斂）在模型欠擬合時有幫助；更好的訓練策略（最佳化器、學習率排程、batch size）則兩者都有幫助。

殘差連接與通往 Transformer 之路

白話版。疊很多層時，每一層不但要改進訊號，還得把它完整傳下去，這其實很難。殘差連接給每個區塊一條快速道路：區塊只需要學一個加在輸入上的修正量。

He 等人觀察到，在已經很深的網路上再加普通的層，連**訓練**誤差都會變差；這是最佳化的問題，不是過擬合 [28]。他們的**殘差區塊**計算

$$\mathbf{y} = \mathbf{x} + F(\mathbf{x}),$$

其中 $\mathbf{x}$ 是區塊的輸入，F 是一小疊層（在 ResNet 中是卷積加上 BatchNorm 與 ReLU）。如果一個區塊最好的做法是什麼都不做，它只需要把 F 推向零，這很容易。反向傳遞說明了為什麼它好訓練：

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = I + \frac{\partial F}{\partial \mathbf{x}},$$

其中 I 是單位矩陣。單位項讓梯度直接穿過每一個區塊，所以許多區塊的連乘不再必然縮到零。靠著殘差連接，He 等人在 ImageNet 上訓練了多達 152 層的網路，並贏得 ILSVRC 2015 的多個項目 [28]。

Transformer（Vaswani 等人 [29]）結合了本篇的每一個概念。每個區塊有兩個子層：多頭自注意力（multi-head self-attention，用 softmax 在不同位置之間混合資訊）與逐位置的 MLP，每個子層都包在殘差連接與 LayerNorm 裡。原始論文是在相加之後才正規化；許多近期模型（包括 LLaMA）改為正規化子層的**輸入**（pre-normalization），並使用 RMSNorm 與 SwiGLU [12]：

$$\mathbf{h} = \mathbf{x} + \mathrm{Attention}\,(\,\mathrm{Norm}(\mathbf{x})), \qquad \mathbf{y} = \mathbf{h} + \mathrm{FFN}\,(\,\mathrm{Norm}(\mathbf{h})).$$

自 2017 年起，這個設計從機器翻譯擴展到語言模型、影像、語音等領域，也是今日大型語言模型的基礎。它的訓練方法同樣來自本篇：AdamW、暖身加餘弦衰減、梯度裁剪與權重衰減 [12]。深度學習系列會從這裡接著談用於影像的卷積網路與 Transformer，第一個任務是影像分類。

現代觀點

上述的基本工具已經穩定好幾年了；研究持續探討它們為什麼有效，以及每個零件更好的版本。以下幾篇回顧從不同角度提供了很好的導覽。

全貌：LeCun、Bengio 與 Hinton [2]（Nature，2015）。這篇短回顧至今仍是「深度學習是什麼」最清楚的說明：以多層進行表示學習（representation learning），每一層把前一層的表示轉成更抽象的表示，特徵以反向傳播從資料中學得，而不是人工設計。它介紹了以 SGD 進行的監督式學習、ReLU 的角色、處理影像的卷積網路、處理序列的遞迴網路，以及分散式表示（distributed representation），最後預測非監督式學習，以及表示學習與推理的結合，在未來會更加重要。若要以教科書的方式學習本篇的每個主題（包括損失函數的最大概似觀點、正則化與最佳化），Goodfellow、Bengio 與 Courville 的書 [3] 仍是標準參考。

最佳化：Sun [26]。Sun 的綜述問的是「類神經網路何時、為什麼能被成功訓練？」，並把答案分成三部分：梯度的問題（爆炸、消失）以及透過謹慎的初始化與正規化來解決；實用的演算法（SGD、Adam 等自適應方法，以及分散式訓練）與它們的收斂理論；以及損失地形（loss

landscape）的整體圖像，包括不好的局部極小值、模式連通性（mode connectivity）、樂透彩券假說（lottery ticket hypothesis）與無限寬度極限。它是從上述實務走向理論的好橋樑。

正規化：Huang 等人 [16]（IEEE TPAMI，2023）。這篇綜述從最佳化的角度給出正規化方法的統一觀點，並提出一套分類法，把任何方法拆成正規化區域、正規化運算與表示還原三部分。它涵蓋從 BatchNorm 開始的發展史、正規化為何能加速訓練並幫助泛化的分析，以及在不同領域的應用。讀它可以把 BatchNorm、LayerNorm、分組式與以權重為主的正規化，看成同一個設計空間中的不同點。

激活函數：Dubey、Singh 與 Chaudhuri [7]（Neurocomputing，2022）。這篇綜述把激活函數分為 logistic sigmoid ／ tanh 類、ReLU 類、ELU 類與可學習類，依值域、單調性與平滑性等性質加以描述，並在不同網路架構與資料類型上評測其中 18 種。它的實務訊息是：沒有哪一種激活函數處處最好，選擇會和架構與資料互相影響。

現況（2023–2026）。從已發表的模型訓練配方可以看到幾個趨同的方向：

- **架構**。採用 pre-normalization 與殘差連接的 Transformer 區塊，是語言領域的主流架構，在影像領域也越來越普遍；RMSNorm 與 SwiGLU（如 LLaMA [12]）是大型語言模型的常見預設。
- **最佳化**。 AdamW [20] 搭配線性暖身、餘弦（或類似的）衰減與梯度裁剪是標準配方。不只做逐座標縮放的最佳化器是活躍的研究方向：Lion [21] 使用正負號更新、較省記憶體；Muon [22] 把矩陣更新正交化，並有在大規模下提升運算效率的報告 [23]。這些增益在不同任務與規模下有多穩健，仍在研究中，因此 AdamW 依然是安全的選擇。
- **泛化**。尖銳與平坦極小值的觀點 [25] 是有用的直覺，而不是完整的理論；高度過度參數化（over-parameterized）的網路為什麼還能泛化得這麼好，至今仍是開放問題，Sun 的綜述 [26] 整理了主要的研究方向。

開放問題。深度網路可靠的泛化理論；同時選擇學習率、batch size 與排程的原則性規則，特別是在放大模型規模時；正規化、殘差連接與自適應最佳化器之間究竟如何交互作用；以及不靠目前配方中那些大量小技巧，也能在極大規模下穩定訓練。

重點整理

- 感知器使用門檻決策與錯誤驅動更新；邏輯迴歸使用平滑機率與基於概似的損失。
- 非線性隱藏層擴大網路的表示能力；近似定理不保證訓練成功或泛化。
- 激活與正規化選擇改變梯度傳遞，須區分批次統計與逐樣本統計。
- 依任務選擇輸出與損失配對，推導梯度時檢查加總或平均的係數。
- 反向傳播透過連鎖律重用中間導數，數值檢查可找出許多實作錯誤。
- 一起看訓練與驗證曲線；調整最佳化與正則化後，再判斷是否只是模型大小造成失敗。

練習

1. **手算感知器**。對 AND 資料執行感知器規則，$\eta = 1$、$\mathbf{w} = \mathbf{0}$、$b = 0$，依 $(0,0), (0,1), (1,0), (1,1)$ 的順序拜訪各點。寫下每次更新後的 $\mathbf{w}$ 與 b，直到某個 epoch 完全沒有錯誤。接著用一句話解釋，為什麼同樣的程序用在 XOR 上永遠不會停。

提示

在嚴格判斷 $z > 0$ 下，第一個點 $(0,0)$ 的 $z = 0$，預測為 0，是正確的。第一個錯誤發生在 $(1,1)$，得到 $\mathbf{w} = (1,1)$、$b = 1$。繼續做下去：在標籤為 0 的點上犯錯就減去 $\mathbf{x}$ 和 1，在標籤為 1 的點上犯錯就加上它們。以這個順序，五個 epoch 內共更新十次，第六個 epoch 完全正確，此時 $\mathbf{w} = (2,1)$、$b = -2$，也就是直線 $2x_1 + x_2 = 2$。換一個拜訪順序會得到另一條同樣有效的直線。對 XOR 而言，沒有任何 $(\mathbf{w}, b)$ 能把四個點全部分對，所以永遠有某個點是錯的，也就永遠會更新。

2. **sigmoid 的梯度消失得多快？**證明 $\sigma'(x) = \sigma(x)(1 - \sigma(x)) \le 1/4$。接著考慮一條由 20 個 sigmoid 層組成、所有權重都等於 1 的鏈。梯度在往回傳的路上，最多會被乘上多大的因子？對活躍單元的 ReLU 重做一次推理。

提示

$s(1-s)$ 在 $s \in (0,1)$ 上是開口向下的拋物線，在 $s = 1/2$ 有最大值 $1/4$。二十個最多 $1/4$ 的因子相乘，最多是 $4^{-20} \approx 9 \times 10^{-13}$。對活躍的 ReLU 單元，$f' = 1$，所以激活函數完全不會造成縮小；只有權重會，這就是初始化重要的原因。

3.「**預測減目標**」**梯度**。利用 $\partial p_k / \partial z_j = p_k(\delta_{kj} - p_j)$，證明當 $\mathbf{y}$ 為 one-hot 時，$L = -\sum_k y_k \log p_k$ 滿足 $\partial L / \partial z_j = p_j - y_j$。再對 sigmoid 搭配二元交叉熵證明同樣的結果。

提示

$\partial L / \partial z_j = -\sum_k (y_k / p_k)\, p_k(\delta_{kj} - p_j) = -\sum_k y_k \delta_{kj} + p_j \sum_k y_k = -y_j + p_j$，因為 $\sum_k y_k = 1$。對 sigmoid，$\partial \ell / \partial \hat{p} = -y/\hat{p} + (1-y)/(1-\hat{p})$，且 $\partial \hat{p} / \partial z = \hat{p}(1-\hat{p})$；相乘並化簡即得 $\hat{p} - y$。

4. **擴充梯度檢查**。在 NumPy 範例中，把 `W2`、`b1`、`b2` 的每個元素（而不只是 `W1` 的一個元素）都和有限差分比較，並回報最大相對誤差 $|a - n| / \max(|a|, |n|, 10^{-12})$。然後故意放一個 bug，例如拿掉 `* (Z1 > 0)` 這個因子，看看誤差會變多大。

提示

用 `np.ndindex(param.shape)` 跑過每個元素，把它擾動 $\pm 10^{-6}$ 並重新計算損失。正確的實作相對誤差約在 10^{-7} 或更小（少數緊鄰 ReLU 折角的元素可能較大）。拿掉 ReLU 導數後，`W1` 與 `b1` 會出現量級為 1 的誤差，而 `W2` 與 `b2` 仍然正確，因為 bug 只在隱藏層以下的反向路徑上。

5. **Iteration、epoch 與 batch size**。某資料集有 60,000 張訓練影像。(a) $B = 32$ 與 $B = 1{,}024$ 時，一個 epoch 各有多少次 iteration？(b) 兩者都訓練 20 個 epoch，各做了多少次參數更新？(c) 根據 batch size 的討論，舉出兩個理由說明在相同學習率下，$B = 1{,}024$ 的執行可能得到較差的驗證準確率，並提出一個可能有幫助的調整。

提示

(a) $\lceil 60000/32 \rceil = 1875$ 與 $\lceil 60000/1024 \rceil = 59$。(b) 37,500 次對 1,180 次更新。(c) 同樣的 epoch 數下更新次數少很多；梯度雜訊也較少，而尖銳極小值的研究把這和較差的泛化連在一起。常見的調整是針對新的 batch size 重新調學習率（通常調大，並加上暖身），以及訓練更多 epoch。

6. **診斷曲線**。同事的網路訓練損失一開始下降很快，接著停在一個很高的值不動；驗證損失和它幾乎貼在一起。同事提議把隱藏單元數減半，「讓它收斂得快一點」。發生了什麼事？你會先檢查什麼？你會改提出什麼建議？

提示

兩條損失都很高而且彼此接近：這是欠擬合（數值偏高的情況 C），不是過擬合。先檢查 bug：標籤是否與輸入對齊、損失是否作用在 logits 上、資料是否已正規化、學習率是否合理。接著提高模型能力或改善最佳化（更寬或更深的模型、學習率排程、訓練更久）。把模型縮小會降低能力，只會讓欠擬合更嚴重。

參考文獻

1. Shyandram, “機器學習及類神經網路筆記,” Medium post, 2024; updated 2026. link
2. Y. LeCun, Y. Bengio and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. doi
3. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016. online book
4. F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, pp. 386–408, 1958. doi
5. G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989. doi
6. D. E. Rumelhart, G. E. Hinton and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986. doi
7. S. R. Dubey, S. K. Singh and B. B. Chaudhuri, “Activation functions in deep learning: A comprehensive survey and benchmark,” *Neurocomputing*, vol. 503, pp. 92–108, 2022. arXiv
8. D. Hendrycks and K. Gimpel, “Gaussian error linear units (GELUs),” arXiv:1606.08415, 2016. arXiv
9. S. Elfwing, E. Uchibe and K. Doya, “Sigmoid-weighted linear units for neural network function approximation in reinforcement learning,” *Neural Networks*, vol. 107, pp. 3–11, 2018. arXiv
10. P. Ramachandran, B. Zoph and Q. V. Le, “Searching for activation functions,” arXiv:1710.05941, 2017. arXiv
11. N. Shazeer, “GLU variants improve Transformer,” arXiv:2002.05202, 2020. arXiv
12. H. Touvron, T. Lavril, G. Izacard et al., “LLaMA: Open and efficient foundation language models,” arXiv:2302.13971, 2023. arXiv
13. S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. International Conference on Machine Learning (ICML)*, PMLR 37, pp. 448–456, 2015. PMLR
14. J. L. Ba, J. R. Kiros and G. E. Hinton, “Layer normalization,” arXiv:1607.06450, 2016. arXiv
15. B. Zhang and R. Sennrich, “Root mean square layer normalization,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv
16. L. Huang, J. Qin, Y. Zhou, F. Zhu, L. Liu and L. Shao, “Normalization techniques in training DNNs: Methodology, analysis and application,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 8, pp. 10173–10196, 2023. arXiv

17. X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proc. International Conference on Artificial Intelligence and Statistics* (*AISTATS*), PMLR 9, pp. 249–256, 2010. PMLR
18. K. He, X. Zhang, S. Ren and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in *Proc. IEEE International Conference on Computer Vision* (*ICCV*), pp. 1026–1034, 2015. arXiv
19. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2015. arXiv
20. I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2019. arXiv
21. X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu and Q. V. Le, “Symbolic discovery of optimization algorithms,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2023. arXiv
22. K. Jordan, “Muon: An optimizer for hidden layers in neural networks,” Medium post, December 2024. link
23. J. Liu, J. Su, X. Yao et al., “Muon is scalable for LLM training,” arXiv:2502.16982, 2025. arXiv
24. I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2017. arXiv
25. N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy and P. T. P. Tang, “On large-batch training for deep learning: Generalization gap and sharp minima,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2017. arXiv
26. R. Sun, “Optimization for deep learning: Theory and algorithms,” arXiv:1912.08957, 2019. arXiv
27. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. JMLR
28. K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pp. 770–778, 2016. arXiv
29. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2017. arXiv
30. S. J. Reddi, S. Kale and S. Kumar, “On the Convergence of Adam and Beyond,” *ICLR*, 2018. arXiv

ML 03・卷積、影像濾波與 CNN

從模糊與 Sobel 卷積核到學習式濾波器：卷積與互相關的差別、邊界處理、填補、步幅、膨脹、池化、感受野、卷積的反向傳播，以及經典 CNN 架構。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/convolutional-networks/

筆記本: https://github.com/Shyandram/cv-blog/blob/main/notebooks/image-processing/convolution.ipynb

你將學到

- 說明濾波核，區分互相關與卷積。
- 實作濾波，明確選擇資料型別與邊界規則。
- 追蹤通道、填補、步幅、膨脹與輸出形狀。
- 連結感受野、平移行為與反向傳播。
- 從結構與計算選擇比較 CNN 設計。

先看全貌

影像是一個數字網格（見 影像就是陣列）。我們想對影像做的許多事都是**局部的**：一個像素是在邊緣上、在平滑區域內，還是在雜訊斑塊裡，大多取決於它的鄰居。**卷積** 正是把「看一小塊鄰域、用固定權重組合、然後在每個位置重複」這件事形式化的運算。它是古典影像處理的主力工具，也是**卷積神經網路**（convolutional neural network, CNN）的基本積木；差別在於 CNN 從資料中學習權重，而不是由人手動設計。

> **用最簡單的話說**
>
> 想像你有一張開了九個小洞的硬紙板，每個洞後面寫著一個數字。把紙板放在圖畫上，把每個洞看到的顏色乘上洞後面的數字，全部加起來，把答案寫在一張新紙上。接著把紙板滑一格，再做一次，直到蓋過整張圖。如果九個數字都一樣，新的圖就是模糊的平均。如果左邊的數字是負的、右邊是正的，只有在圖從暗變亮的地方才會得到大答案：這就是邊緣偵測器。CNN 就是一台同時滑動成千上萬片這種紙板，並且**自己學會**洞後面該寫什麼數字的機器。

本篇分成兩半。前半（濾波）是古典工具箱，只需要 NumPy。後半（CNN）建立在 ML 02 之上，你在那裡已經認識了網路層、損失函數與反向傳播。若之後想更深入固定濾波器，IP 03 介紹空間濾波，IP 04 則透過卷積定理說明頻率域觀點。深度學習方面，DLCV 02 詳細探討現代分類骨幹網路，ML 04 轉向序列資料，ML 05 則介紹卷積的主要替代方案：注意力機制。標準教科書可參考 Goodfellow 等人 [1] 的第 9 章與 Prince [10] 的第 10 章。

第一部分・用固定卷積核濾波

滑動的加權和

白話版。每個輸出像素都是輸入影像在相同位置附近像素的加權和。那一小格權重稱為**卷積核**（kernel，也稱為濾波器 filter 或遮罩 mask）。

精確版。令 I 為灰階影像，$I(x,y)$ 是第 x 欄、第 y 列的強度。令 K 為大小 $(2a+1)\times(2b+1)$ 的卷積核，垂直索引從 $-a$ 到 a、水平索引從 $-b$ 到 b，因此 $K(0,0)$ 是它的中心。濾波後的影像 G 為

$$G(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j)\, I(x+j,\; y+i).$$

其中 i 索引卷積核的列、j 索引卷積核的行，$G(x,y)$ 是相同位置的輸出，雙重總和恰好走訪卷積核的每個係數一次。以 3×3 卷積核為例，$a=b=1$，每個輸出像素需要九次乘法與八次加法。

圖 1 手算了一個輸出值。橘色視窗蓋住填補後輸入左上角的 3×3 鄰域。與卷積核逐元素相乘再相加：$1\cdot1+0\cdot2+(-1)\cdot0+2\cdot0+0\cdot1+(-2)\cdot3+1\cdot2+0\cdot0+(-1)\cdot1=-4$。這個數字就成為 G 在視窗中心的值。把視窗往右滑一格再重複，就填出下一個輸出。

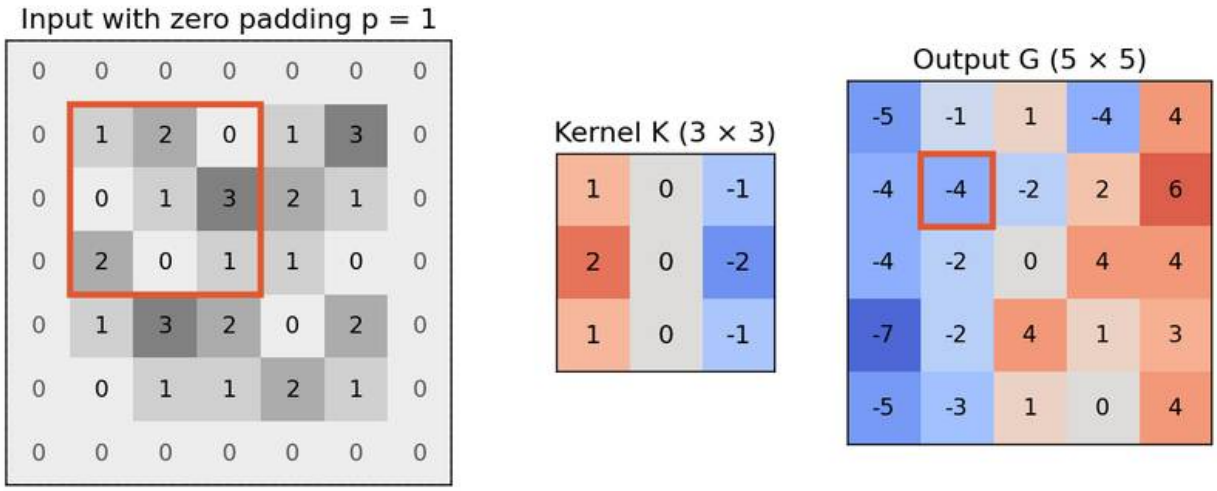


圖 *1* — 滑動加權和的一步。橘色 3×3 視窗與 K 逐元素相乘再加總，在對應的輸出位置得到 -4。零填補（灰色）讓輸出維持與輸入相同的大小。

有兩個觀察對後面所有內容都很重要。第一，同樣的九個權重用在每一個位置：這個運算具有**平移不變性**（shift-invariant，也稱為空間不變）。第二，這個運算是**線性的**：對 $\alpha I_1+\beta I_2$ 濾波會得到 $\alpha G_1+\beta G_2$。線性平移不變（linear shift-invariant, LSI）運算恰好就是卷積能表達的那一類，這也是卷積在訊號處理中如此核心的原因。

互相關與卷積

白話版。上面的公式在技術上是**互相關**（cross-correlation）。真正的**卷積**做同樣的事，只是卷積核先旋轉 180 度。對稱的卷積核兩者結果相同；不對稱的則不同。

精確版。數學上的卷積寫成 $K * I$：

$$(K * I)(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j)\, I(x-j,\, y-i) = \sum_{i,j} K(-i,-j)\, I(x+j,\, y+i).$$

第二種寫法顯示，卷積等於用翻轉後的卷積核 $K(-i,-j)$ 做互相關。看出差別最乾淨的方法，是對單一亮點（**脈衝**，impulse）濾波。互相關會留下一個旋轉 180 度的卷積核副本；卷積則留下卷積核的原樣副本。這個副本就是**脈衝響應**（impulse response），也是訊號處理偏好卷積的原因：任何影像的輸出，都是一堆經過平移、縮放的脈衝響應之和。

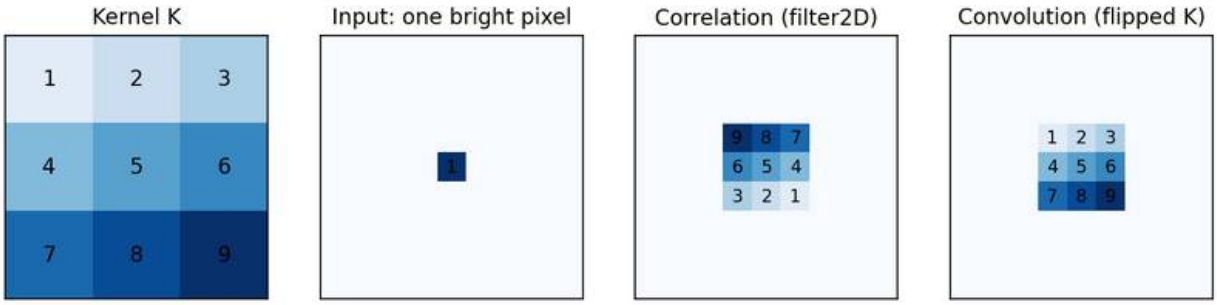


圖 2 — 對脈衝濾波。互相關（*cv2.filter2D* 的計算方式）印出旋轉 *180°* 的卷積核；卷積印出卷積核本身。

卷積具有互相關所沒有的代數性質。它滿足**交換律**（$K * I = I * K$）與**結合律**（$K_2 * (K_1 * I) = (K_2 * K_1) * I$），因此連續套用兩個濾波器，等於套用一個較大的濾波器。例如連續做兩次 3×3 方框模糊，就等於一個 5×5 卷積核。結合律也解釋了**可分離**（separable）卷積核：$k \times k$ 的高斯核等於一個 $k \times 1$ 的行濾波器再接一個 $1 \times k$ 的列濾波器，每個像素的運算量從 k^2 降到 $2k$。**卷積定理**指出，空間中的卷積等於傅立葉域中的乘法；IP 04 會發展這個觀點。

各函式庫計算的是哪一種？

函式庫呼叫	計算內容	預設邊界
`cv2.filter2D`	互相關 [11]	reflect-101（`BORDER_DEFAULT`）
`scipy.ndimage.correlate` / `convolve`	互相關 / 卷積	`reflect`（重複邊緣樣本）
`scipy.signal.convolve2d`	卷積	補零
`torch.nn.functional.conv2d`、`nn.Conv2d`	互相關 [31]	零填補

對於手工設計的不對稱卷積核（Sobel、浮雕、運動模糊），翻轉與否會改變結果的正負號或方向，所以需要真正的卷積時，要刻意翻轉，例如用 `cv2.flip(K, -1)`。對於學習出來的卷積核，這個差別就消失了：一個「本來應該」使用翻轉卷積核的網路，會直接把翻轉後的版本學起來。這就是為什麼深度學習框架把互相關稱為「卷積」也無傷大雅。

模糊、邊緣與銳化

白話版。總和為一的正權重會平均鄰居而產生模糊。總和為零的權重會抵消平坦區域、只留下變化，因此能找出邊緣。把一點點「影像減去其模糊版本」加回影像，就能銳化。

方框模糊（box blur）讓每個鄰居權重相等；3×3 版本有九個 $1/9$。**高斯模糊**取樣

$$K(i,j) = \frac{1}{Z} \exp\!\left(-\frac{i^2 + j^2}{2\sigma^2}\right),$$

其中 σ（以像素為單位）決定模糊寬度，Z 將權重正規化使總和為一。近的像素比遠的像素重要，因此能避免方框模糊的塊狀瑕疵。實務上常用約 3σ 的卷積核半徑。

權重總和告訴你在每個像素都等於 c 的平坦區域會發生什麼：輸出為 $c \sum K$。總和為一的模糊保持亮度；總和為二的卷積核會讓亮度加倍（並使 8 位元影像飽和）；總和為零的卷積核則輸出零。

Sobel 卷積核估計水平與垂直方向的導數，同時在垂直方向上做平滑：

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \qquad S_y = S_x^{\top}.$$

S_x 對沿 x 方向的強度變化有反應，所以它凸顯**垂直**邊界；S_y 凸顯**水平**邊界。輸出 G_x、G_y 帶有正負號：由暗到亮與由亮到暗的邊緣正負相反。**梯度強度**（gradient magnitude）$\sqrt{G_x^2 + G_y^2}$ 是非負的邊緣強度，$\operatorname{atan2}(G_y, G_x)$ 則是邊緣方向。上面的分解顯示 Sobel 是可分離的：一個 $[1,2,1]^{\top}$ 平滑行向量乘上一個 $[-1,0,1]$ 中央差分。

拉普拉斯（Laplacian）卷積核 $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ 近似 $\partial^2 I/\partial x^2 + \partial^2 I/\partial y^2$；它對細線以及邊緣的兩側都有反應。**反銳化遮罩**（unsharp masking）以下式銳化：

$$I_{\text{sharp}} = I + \lambda\,(I - G_\sigma * I),$$

其中 $G_\sigma * I$ 是高斯模糊後的副本，$I - G_\sigma * I$ 是高頻細節，$\lambda > 0$ 控制強度。λ 太大會放大雜訊，並在邊緣兩側產生亮暗光暈。

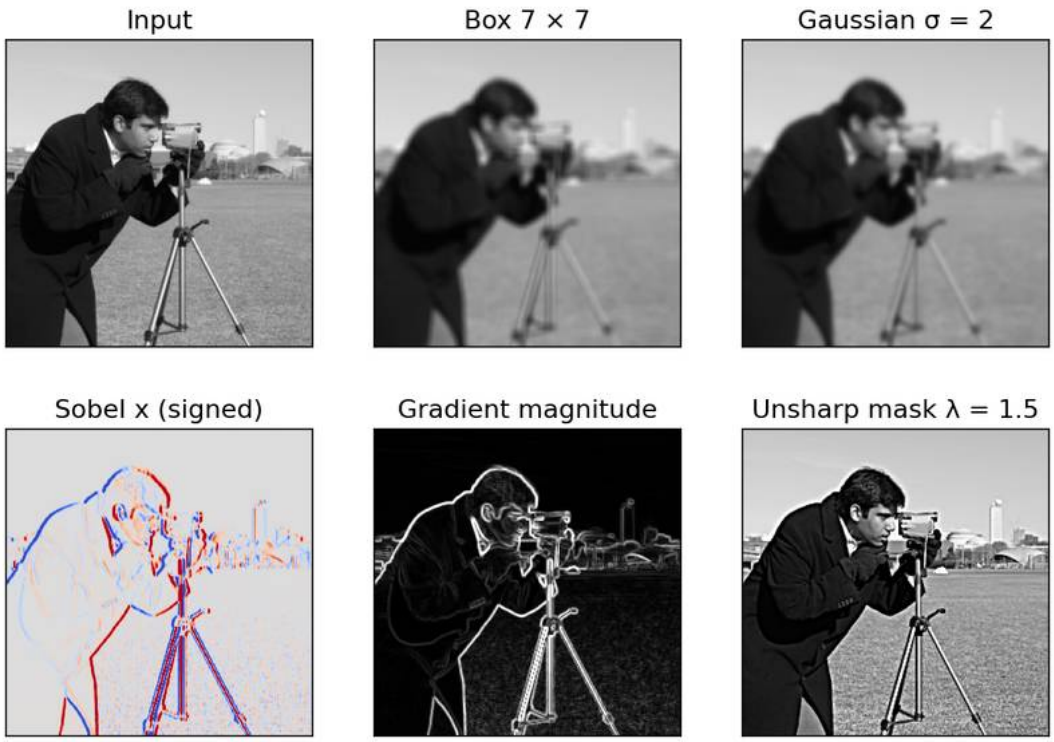


圖 3 — 同一張影像經過六種卷積核。方框與高斯模糊壓抑細節；*Sobel x* 帶有正負號（紅 = 由暗到亮，藍 = 由亮到暗）；梯度強度勾勒出物體輪廓；反銳化遮罩把放大的細節加回去。

動手試試：卷積核實驗室

編輯下方的卷積核格子，觀察輸出即時更新。先從 Sobel x 開始，然後把它轉置，改為偵測水平邊緣而非垂直邊緣。把模糊預設的每個權重都調大，你會看到平坦區域變亮，正如權重總和規則的預測。試試拉普拉斯與浮雕預設，並在看結果之前先預測它們的卷積核總和。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

下方的滑桿比較一張合成測試影像（圓盤、正方形、棋盤格與條紋）與它經過 5×5 高斯模糊（$\sigma = 1.4$）的結果。細條紋與棋盤格最先消失，因為模糊是一種**低通濾波器**：它移除快速的強度變化，保留緩慢的變化。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

程式碼：從零開始，再到 OpenCV、SciPy 與 PyTorch

下面的迴圈就是整個演算法。它不逐一走訪每個像素（在 Python 裡很慢），而是走訪卷積核的每個係數，每個係數加上一份平移過的填補影像，讓 NumPy 做向量化計算。接著程式檢查：只要

對齊定義（互相關或卷積）與邊界規則，四個函式庫的結果就會一致。需要 numpy、scipy、opencv-python 與 torch。

```
import numpy as np
import cv2
import torch
import torch.nn.functional as F
from scipy import ndimage, signal

def correlate2d(img, K, mode="edge"):
    """Cross-correlation of a 2-D float image with an odd-sized kernel.
    mode is any np.pad mode: "constant" (zeros), "edge", "reflect", "wrap"."""
    kh, kw = K.shape
    ph, pw = kh // 2, kw // 2
    padded = np.pad(img, ((ph, ph), (pw, pw)), mode=mode)
    H, W = img.shape
    out = np.zeros((H, W), dtype=np.float64)
    for i in range(kh):              # loop over kernel taps, not pixels:
        for j in range(kw):          # each step adds one shifted copy of the image
            out += K[i, j] * padded[i:i + H, j:j + W]
    return out

def convolve2d(img, K, mode="edge"):
    return correlate2d(img, K[::-1, ::-1], mode)     # convolution = correlation with flipped
     ↪  K

rng = np.random.default_rng(0)
img = rng.random((32, 40))
K = rng.standard_normal((3, 5))                      # deliberately asymmetric

ours_corr = correlate2d(img, K, "reflect")
ours_conv = convolve2d(img, K, "reflect")
# OpenCV: filter2D computes correlation; BORDER_REFLECT_101 is np.pad's "reflect"
cv_corr = cv2.filter2D(img, cv2.CV_64F, K, borderType=cv2.BORDER_REFLECT_101)
# SciPy: ndimage.correlate / ndimage.convolve with mode="mirror" (= reflect-101)
sp_corr = ndimage.correlate(img, K, mode="mirror")
sp_conv = ndimage.convolve(img, K, mode="mirror")
# PyTorch: conv2d is correlation; pad first with the same rule
t = torch.tensor(img)[None, None]
t = F.pad(t, (2, 2, 1, 1), mode="reflect")
pt_corr = F.conv2d(t, torch.tensor(K)[None, None])[0, 0].numpy()

for name, a in [("OpenCV", cv_corr), ("SciPy", sp_corr), ("PyTorch", pt_corr)]:
    print(f"{name:8s} correlation matches:", np.allclose(ours_corr, a))
print("SciPy    convolution matches:", np.allclose(ours_conv, sp_conv))
print("conv == corr for this kernel?", np.allclose(ours_conv, ours_corr))
# zero padding: our convolution equals signal.convolve2d with mode="same"
same = signal.convolve2d(img, K, mode="same", boundary="fill")
print("zero-padded 'same' convolution matches:", np.allclose(convolve2d(img, K, "constant"),
 ↪  same))
```

除了 conv == corr 因為隨機卷積核不對稱而印出 False 之外，所有檢查都印出 True。注意命名陷阱：NumPy 的 "reflect" 填補（dcb|abcd|cba，不重複邊緣樣本）對應 OpenCV 的 BORD

ER_REFLECT_101 與 SciPy 的 "mirror"；而 SciPy 的 "reflect" 會重複邊緣樣本（dcba|abcd|dcba）。

第二段程式碼展示了初學者處理真實影像時最常遇到的兩個錯誤：整數飽和與卷積核總和規則。

```
import numpy as np
import cv2
from skimage import data

img_u8 = data.camera()                                  # 512 x 512 uint8 test image
img = img_u8.astype(np.float32) / 255.0
sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)

gx = cv2.filter2D(img, -1, sobel_x)                     # float in, float out: negatives kept
gy = cv2.filter2D(img, -1, sobel_x.T)
mag = np.hypot(gx, gy)
print("float Sobel x range:", gx.min().round(3), gx.max().round(3))

gx_u8 = cv2.filter2D(img_u8, -1, sobel_x)               # uint8 in, uint8 out: saturated
print("uint8 Sobel x range:", gx_u8.min(), gx_u8.max(),
      "| pixels clipped to 0:", f"{(gx_u8 == 0).mean():.0%}")

blur = cv2.GaussianBlur(img, (5, 5), 1.4)               # 5 x 5 Gaussian, sigma 1.4
print("mean brightness kept by blur:", np.isclose(img.mean(), blur.mean(), atol=1e-3))
K2 = 2 * np.ones((3, 3), np.float32) / 9                # weights sum to 2
print("mean after sum-2 kernel:", round(float(cv2.filter2D(img, -1, K2).mean() / img.mean()),
 ↪  3))
```

在浮點數下，Sobel 響應大約介於 −3.4 到 3.3。若輸入與輸出都是 uint8，所有負的響應都會被截成零（這裡超過一半的像素），因此所有由亮到暗的邊緣都消失了。請一律用 float32 濾波，只在顯示時才轉回整數。

邊界處理

白話版。在影像邊緣附近，卷積核有一部分會懸在影像外面。你必須決定要假裝那裡有什麼值。

常見選擇有：**零填補**（常數填補），假裝外面的世界是黑色；**複製**（replicate），重複邊緣像素；**反射**（reflect），在邊界處鏡射影像；**環繞**（wrap），把影像視為週期性的，離散傅立葉轉換隱含的就是這種假設。OpenCV 的預設 BORDER_REFLECT_101 會鏡射但不重複邊緣像素 [11]。PyTorch 的 nn.Conv2d 預設使用零填補，除非你把 padding_mode 設為 'reflect'、'replicate' 或 'circular' [31]。

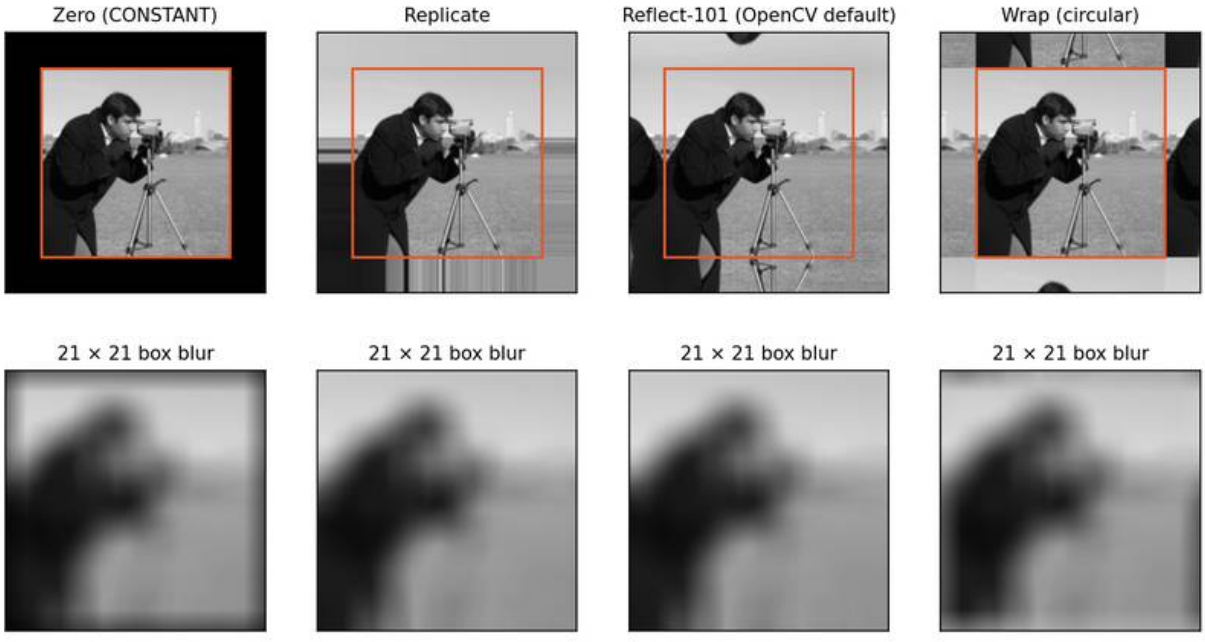


圖 *4* — 四種邊界規則及其對大型 *21 × 21* 方框模糊的影響。零填補讓每個邊框變暗；環繞混合了相對的邊；複製與反射讓平坦的邊框維持平坦。

零填補會讓模糊影像的邊框變暗，因為黑色像素被平均進去了。環繞則把影像頂部混進底部。對照片而言，複製與反射通常最不顯眼。卷積核小的時候，差別只影響一圈細框；但在深層 CNN 中，每一層都會加上自己的一圈：經過許多零填補層之後，特徵圖中很大一部分都「看過」人工邊界，這讓網路能從填補中推斷出絕對位置。

第二部分・卷積神經網路

從固定濾波器到學習式網路層

白話版。標準密集卷積學習涵蓋全部輸入通道的濾波器，各自產生一張輸出特徵圖；分組卷積則限制通道連接。

精確版。令輸入為 $X \in \mathbb{R}^{C_{\text{in}} \times H \times W}$（通道、高、寬），權重為 $K \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k \times k}$，偏差為 $b \in \mathbb{R}^{C_{\text{out}}}$。步幅為一、零填補為 p 的網路層計算

$$Y_{o,u,v} = b_o + \sum_{c=1}^{C_{\text{in}}} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} K_{o,c,i,j}\, X_{c,\, u+i-p,\, v+j-p}.$$

o 索引輸出通道（濾波器），c 索引輸入通道，(u, v) 索引輸出位置，(i, j) 索引卷積核係數；影像外的位置讀取填補值。接著逐元素套用 ReLU 等非線性函數。一個 RGB 濾波器有三片 $k \times k$，每

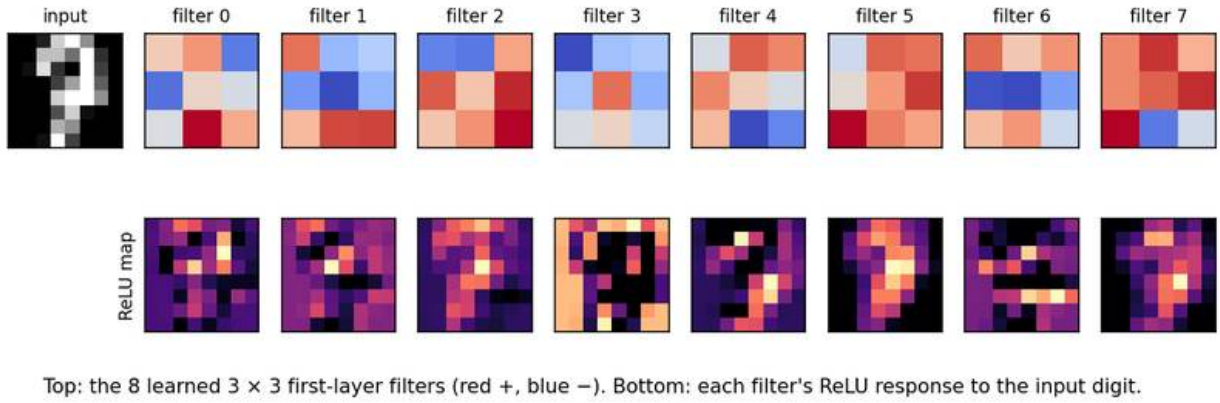
個顏色通道一片，它們的響應會**加總**成每個位置的一個數字：一個濾波器不會產生三張分開的影像。

這個公式內建了兩個結構假設：

- **局部連接**（local connectivity）：每個輸出只取決於一個 $k \times k$ 鄰域，而不是整張影像。
- **權重共享**（weight sharing，又稱參數共享）：同一個 K 用在每個位置 (u, v)。

兩者合起來就是 CNN 的**歸納偏差**（inductive bias），也就是對哪些函數比較可能的假設。它的回報是效率。一個 $C_{\text{in}} = 3$、$C_{\text{out}} = 32$、$k = 3$ 的網路層有 $32 \times (3 \cdot 9 + 1) = 896$ 個參數，與影像大小無關。若改用全連接層，把攤平的 $224 \times 224 \times 3$ 影像映射到區區 32 個數字，就需要 $150{,}528 \times 32 \approx 4.8$ 百萬個權重，而且還得在每個位置分別學會同一個邊緣偵測器。

Goodfellow 等人 [1] 把動機總結為稀疏交互作用、參數共享與等變性。圖 5 展示一個小型 CNN 在後面分類器程式碼所用的 8×8 數字資料集上訓練後，第一層學到了什麼。學到的 3×3 濾波器看起來像有方向的導數與斑點偵測器：網路自己重新發現了類似 Sobel 的卷積核，因為它們有用，而不是因為有人告訴它。Zeiler 與 Fergus [30] 透過視覺化大型 ImageNet 網路的濾波器與特徵圖，在大尺度上展示了同樣的現象。



Top: the 8 learned 3 × 3 first-layer filters (red +, blue −). Bottom: each filter's ReLU response to the input digit.

圖 *5* — 在 *8 × 8* 數字上學到的第一層濾波器（上）以及它們對一個測試數字的 *ReLU* 響應（下）。許多看起來像有方向的邊緣或筆畫偵測器。

填補、步幅與膨脹

白話版。填補（padding）控制邊界處的行為以及輸出是否維持大小。步幅（stride）跳過一些位置以縮小輸出。膨脹（dilation）把卷積核係數分散開來，不增加權重就能看得更遠。

對輸入大小 n（高或寬）、卷積核大小 k、每邊填補 p、步幅 s 與膨脹 d，輸出大小為 [12]

$$n_{\text{out}} = \left\lfloor \frac{n + 2p - d(k-1) - 1}{s} \right\rfloor + 1.$$

$d(k-1)+1$ 是**有效卷積核大小**：膨脹 d 會在係數之間插入 $d-1$ 個空隙。三種填補慣例有名稱：**valid**（$p = 0$，輸出縮小 $k-1$）、**same**（奇數 k 且 $s = 1$ 時 $p = (k-1)/2$，輸出維持大小）、**full**

($p = k - 1$，計入所有部分重疊，輸出增加 $k - 1$）。步幅二大致把每個空間維度減半，是池化之外常見的下採樣方式。

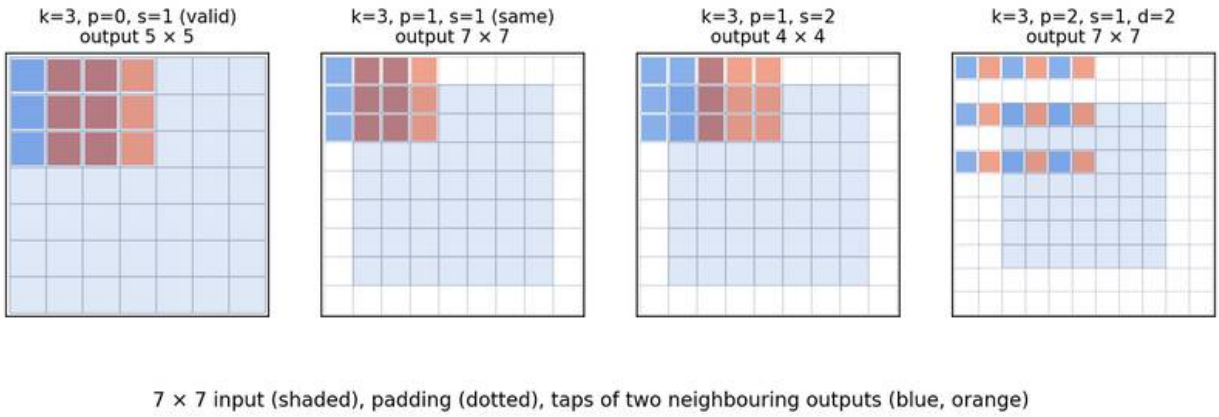


圖 *6* — 四種設定下兩個相鄰輸出（藍與橘；重疊處呈褐色）的係數位置。步幅讓視窗移動 s；膨脹讓係數相隔 d。

膨脹（atrous）卷積由 Yu 與 Koltun [18] 推廣到密集預測：堆疊膨脹率為 1、2、4、… 的網路層，可以在維持完整解析度的同時，讓視野呈指數成長。單一膨脹層只看到稀疏的輸入格點，因此混合膨脹率時要小心，以免產生「網格」（gridding）瑕疵。**轉置卷積**（transposed convolution）把帶步幅卷積的前向映射反過來執行以上採樣；它會在下面的反向傳播小節再次出現。Dumoulin 與 Visin 的卷積算術指南 [12] 為這些設定的每種組合都畫了圖解。

下面的程式碼用 PyTorch 檢查公式、計算參數量、說明 1×1 卷積就是逐像素的線性層，並量測平移等變性：

```
import torch
from torch import nn

def out_size(n, k, p=0, s=1, d=1):
    return (n + 2 * p - d * (k - 1) - 1) // s + 1

x = torch.randn(1, 3, 64, 64)                          # one RGB 64 x 64 image
for k, p, s, d in [(3, 1, 1, 1), (5, 2, 2, 1), (3, 2, 1, 2), (1, 0, 1, 1)]:
    conv = nn.Conv2d(3, 16, k, stride=s, padding=p, dilation=d)
    y = conv(x)
    n_params = sum(q.numel() for q in conv.parameters())
    assert y.shape[-1] == out_size(64, k, p, s, d)
    assert n_params == 16 * (3 * k * k + 1)
    print(f"k={k} p={p} s={s} d={d}: output {tuple(y.shape[1:])}, parameters {n_params}")

# A 1x1 convolution is the same linear layer applied at every pixel
conv1 = nn.Conv2d(16, 4, kernel_size=1)
lin = nn.Linear(16, 4)
lin.weight.data = conv1.weight.data[:, :, 0, 0].clone()
lin.bias.data = conv1.bias.data.clone()
z = torch.randn(2, 16, 8, 8)
same = torch.allclose(conv1(z), lin(z.permute(0, 2, 3, 1)).permute(0, 3, 1, 2), atol=1e-6)
print("1x1 conv == per-pixel Linear:", same)
```

```
# Translation equivariance away from borders (stride 1, no padding effects in the centre)
conv = nn.Conv2d(1, 4, 3, padding=1, bias=False)
img = torch.zeros(1, 1, 32, 32); img[..., 10:14, 8:12] = 1
shift = lambda t: torch.roll(t, shifts=(3, 5), dims=(-2, -1))
with torch.no_grad():
    a, b = conv(shift(img)), shift(conv(img))
print("shift then conv == conv then shift:", torch.allclose(a, b, atol=1e-6))
conv2 = nn.Conv2d(1, 4, 3, padding=1, stride=2, bias=False)
with torch.no_grad():
    y0, y1 = conv2(img), conv2(torch.roll(img, 1, dims=-1))   # shift by one pixel
print("stride 2, 1-pixel shift: max output change =", round((y0 - y1).abs().max().item(), 3))
```

它印出的輸出形狀為 16 × 64 × 64、16 × 32 × 32、16 × 64 × 64 與 16 × 64 × 64，參數量分別為 448、1216、448 與 64；1 × 1 與等變性檢查都通過；而平移一個像素會讓步幅二的輸出產生明顯非零的變化（確切數值取決於隨機初始化的權重）。

通道與 1×1 卷積

白話版。通道是特徵圖的「是什麼」維度；高與寬是「在哪裡」維度。1 × 1 卷積在每個像素混合通道，不看鄰居。

具有 C_{in} 個輸入與 C_{out} 個輸出的 1 × 1 卷積，會把每個像素的通道向量乘上同一個 $C_{\text{out}} \times C_{\text{in}}$ 矩陣，正如上面程式碼所驗證。Lin 等人的 Network in Network [17] 把這個想法當作迷你的逐像素 MLP，並提出**全域平均池化**（global average pooling）作為分類器頭部。GoogLeNet [16] 在昂貴的 3 × 3 與 5 × 5 分支之前用 1 × 1 網路層減少通道數，ResNet 的瓶頸區塊 [5] 則把 3 × 3 層夾在 1 × 1 縮減層與 1 × 1 擴張層之間。動機都相同：空間濾波的成本是通道混合的 k^2 倍，所以便宜的部分交給 1 × 1 網路層，昂貴的部分保持窄小。

池化與下採樣

白話版。池化（pooling）用一個數字（通常是最大值或平均值）概括特徵圖的每個小區塊。它縮小特徵圖，並讓表示對小幅平移較不敏感。

視窗 2 × 2、步幅二的**最大池化**保留每個區塊中的最大值：「這附近有沒有出現這個特徵？」**平均池化**保留平均值。**全域平均池化**把每個通道在所有位置上平均，不論輸入大小，每個通道都產生一個數字，這就是現代分類器能接受不同影像大小的原因。池化沒有可學習的參數。它的梯度很簡單：最大池化把整個上游梯度交給每個視窗中勝出的輸入；平均池化則把梯度平均分給視窗內所有輸入。

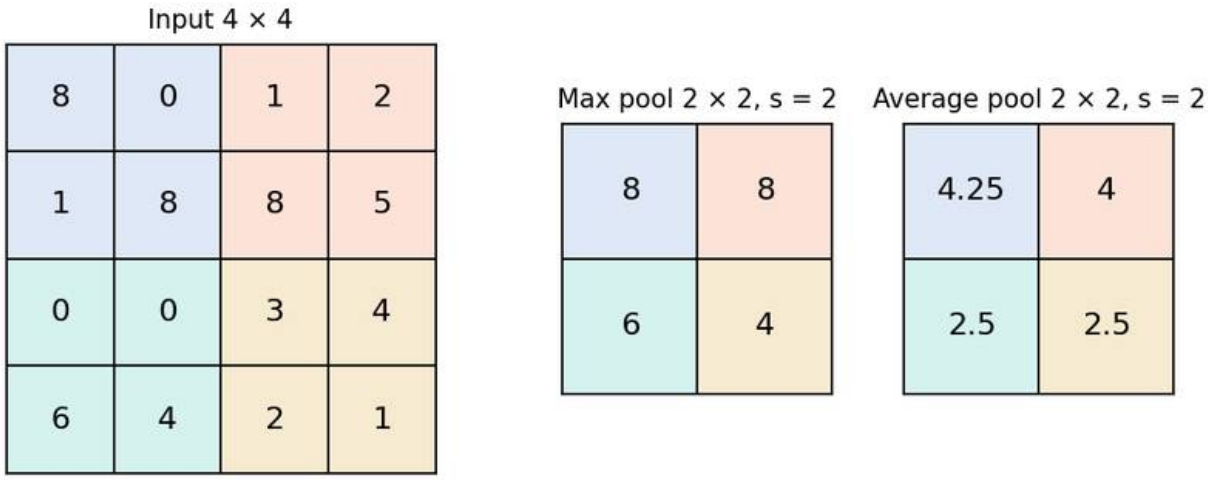


圖 7 — 視窗 2 × 2、步幅 2 的最大池化與平均池化。每個彩色區塊變成一個數字。

無論用池化或帶步幅的卷積，下採樣都能節省運算並擴大後續的感受野，但它也是一種沒有適當低通濾波的**次取樣**（subsampling）。高頻會產生混疊（aliasing），輸入只移動一個像素，輸出就可能明顯改變，正如上面程式碼所量測。Zhang [20] 證明在次取樣前插入一個小模糊（「抗混疊」或「blur pooling」），能讓標準 CNN 的平移不變性顯著提升：這是古典訊號處理規則修好深度網路的漂亮例子。

等變性不等於不變性

白話版。排除邊界影響時，步幅為一的卷積會隨輸入平移；這種等變性不代表預測具有平移不變性。

令 T_Δ 把影像平移 Δ 個像素。在遠離邊界處，步幅為一的卷積具有**平移等變性**（translation-equivariant）：$f(T_\Delta X) = T_\Delta f(X)$。分類器要的是**不變性**（invariance）：$g(T_\Delta X) = g(X)$。CNN 藉由堆疊等變的網路層再加上池化（特別是會捨棄位置資訊的全域平均池化），得到近似的不變性。填補、步幅、裁切與有限的影像大小都會破壞精確的等變性。旋轉與尺度等變性**並不是**內建的；必須靠資料擴增或特別設計的群等變（group-equivariant）網路層來提供。

這個區別在實務上很重要。偵測器必須是等變的：人移動了，預測框也要跟著移動。分類器應該是不變的：人移動了，標籤不該改變。同一個骨幹網路可以服務兩者，只是頭部不同。

感受野

白話版。前面的網路層看到影像的小碎片；較深的網路層把這些碎片組合起來，看到較大的區域。

一個單元的**感受野**（receptive field）是能影響它的輸入像素集合。令 r_ℓ 為第 ℓ 層之後的感受野寬度，j_ℓ 為**跳距**（jump），即第 ℓ 層相鄰單元之間相隔的輸入像素數。從 $r_0 = j_0 = 1$ 開始，卷積核大小 k_ℓ、膨脹 d_ℓ、步幅 s_ℓ 的網路層給出

$$r_\ell = r_{\ell-1} + (d_\ell(k_\ell - 1))\, j_{\ell-1}, \qquad j_\ell = s_\ell\, j_{\ell-1}.$$

三個堆疊的步幅一 3×3 網路層看到 7×7 個像素，面積與單一 7×7 網路層相同，但權重是 $3 \cdot 9C^2 = 27C^2$ 而不是 $49C^2$，並且有三個非線性而非一個；這正是 VGG [15] 的核心設計論點。步幅二的網路層會讓跳距加倍，因此之後每一層的感受野都以兩倍速度成長。

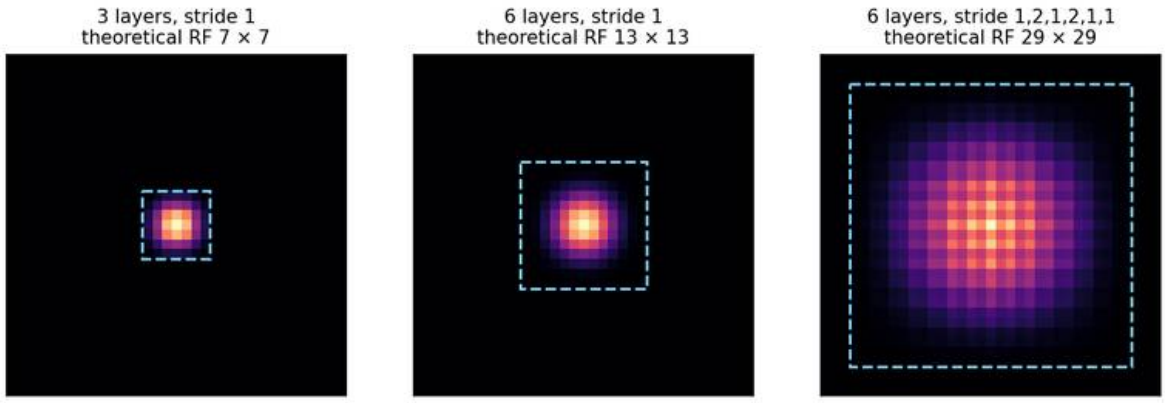


圖 *8* — 理論感受野（虛線）與實際影響力的比較，實際影響力以一個中央輸出穿過 3×3 平均網路層堆疊後的輸入梯度量測。影響力呈類高斯分布並集中在中央；步幅則加上棋盤狀圖樣。

理論感受野說明哪些像素**可能**有影響。Luo 等人 [3] 透過輸入梯度量測**有效感受野**（effective receptive field），如圖 8 所示，發現它大致呈高斯形狀，只占理論感受野的一小部分，而且對均勻的網路層堆疊而言，大約隨深度的平方根成長。由此得到兩個實務教訓：網路可能需要比理論感受野所暗示的更多上下文；而下採樣、膨脹與大型卷積核，都是不必無限加深就能擴大有效感受野的方法。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

卷積的反向傳播

白話版。卷積核梯度結合輸入與上游梯度；輸入梯度使用前向線性運算子的轉置，並須配合步幅及填補規約。

精確版。取單一通道、無填補、步幅一：$Y_{u,v} = \sum_{i,j} K_{i,j} X_{u+i,v+j}$。令 L 為損失，$G_{u,v} = \partial L / \partial Y_{u,v}$ 為反向傳播傳來的**上游梯度**，如 ML 02 所述。每個權重 $K_{i,j}$ 用在每個輸出位置，所以依連鎖律，它的梯度是對這些位置的加總：

$$\frac{\partial L}{\partial K_{i,j}} = \sum_{u,v} G_{u,v}\, X_{u+i,\,v+j}.$$

這是輸入 X 與 G 的 valid 互相關，也說明了為什麼權重共享不會讓訓練變複雜：共享的權重只是把每個位置的梯度累加起來。每個輸入像素 $X_{m,n}$ 會貢獻給所有視窗涵蓋它的輸出，對應係數為 $(i,j) = (m-u, n-v)$：

$$\frac{\partial L}{\partial X_{m,n}} = \sum_{i,j} G_{m-i,\,n-j}\, K_{i,j},$$

其中 G 在其範圍外視為零。這是 G 與 K 的 **full 卷積**，等價於把零填補後的 G 與翻轉的卷積核做互相關。這正是**轉置卷積**所計算的東西，轉置卷積也因此得名。偏差的梯度就是每個輸出通道的 $\sum_{u,v} G_{u,v}$。若有多個通道，輸入梯度還要對輸出通道加總，卷積核梯度還要對批次加總。

```
import torch
import torch.nn.functional as F

torch.manual_seed(0)
X = torch.randn(1, 1, 6, 7, requires_grad=True)     # input
K = torch.randn(1, 1, 3, 3, requires_grad=True)     # kernel
Y = F.conv2d(X, K)                                   # 'valid' correlation: 4 x 5
G = torch.randn_like(Y)                              # pretend upstream gradient dL/dY
(Y * G).sum().backward()                             # so that dL/dY = G exactly

# 1) dL/dK: correlate the input with the upstream gradient
dK = F.conv2d(X.detach(), G)
# 2) dL/dX: 'full' convolution of G with K = correlation of zero-padded G with flipped K
dX = F.conv2d(F.pad(G, (2, 2, 2, 2)), K.detach().flip(-1, -2))
# 3) the same thing is what a transposed convolution computes
dX_t = F.conv_transpose2d(G, K.detach())

print("dL/dK matches autograd:", torch.allclose(dK, K.grad, atol=1e-5))
print("dL/dX matches autograd:", torch.allclose(dX, X.grad, atol=1e-5))
print("conv_transpose2d gives the same:", torch.allclose(dX_t, X.grad, atol=1e-5))
```

三行都印出 `True`。`(Y * G).sum().backward()` 這個技巧讓 autograd 使用我們指定的上游梯度 G，是測試任何手推反向傳播的好方法。

經典架構速覽

白話版。每個知名的 CNN 都是卷積、非線性與下採樣的堆疊。這些年改變的是深度、如何讓深層堆疊可以訓練，以及如何聰明地分配運算量。

模型（年份）	關鍵想法	帶來的啟示
LeNet-5（1998）[13]	卷積 → 次取樣 → 卷積 → 次取樣 → 全連接，端到端訓練	學習出來的卷積特徵在數字辨識上勝過手工特徵
AlexNet（2012）[14]	更深更寬、ReLU、dropout、資料擴增、GPU 訓練	大量標註資料加上 GPU 讓 CNN 贏下 ImageNet
VGG（2014）[15]	只用 3×3 卷積，16–19 層	以小卷積核加深既簡單又有效
GoogLeNet / Inception（2014）[16]	平行的 1×1、3×3、5×5 分支，搭配 1×1 縮減	以受控成本取得多尺度特徵
ResNet（2015）[5]	殘差連接 $y = x + F(x)$、BatchNorm	非常深的網路（100 層以上）變得可以訓練
U-Net（2015）[4]	編碼器–解碼器，在每個解析度加上跳接	用少量影像得到密集、像素對齊的輸出

模型（年份）	關鍵想法	帶來的啟示
MobileNetV2（2018）[7]	深度可分離卷積、反轉殘差、線性瓶頸	適合手機的高準確率模型
EfficientNet（2019）[8]	深度、寬度、解析度的複合縮放	各維度要一起縮放，而不是一次一個
ConvNeXt（2022）[9]	7×7 深度卷積核、LayerNorm、反轉瓶頸、Transformer 時代的訓練方法	現代化的純 CNN 可與視覺 Transformer 競爭

從 LeNet 到 ResNet 的模式是深度穩定增加。AlexNet [14] 證明，用 GPU、ReLU 與 dropout 在 ImageNet 上訓練的深層 CNN 能大幅勝過傳統流程。VGG [15] 主張使用一致的 3×3 網路層堆疊，如感受野小節所計算。GoogLeNet [16] 用 1×1 縮減讓寬度變便宜。接著深度撞上了最佳化之牆：層數更多的普通網路反而有**更高的訓練誤差**，這不是過擬合（overfitting），而是最佳化失敗。ResNet [5] 用恆等捷徑解決了這個問題。DLCV 02 會沿著這條路線繼續探討訓練方法、遷移學習與評估。

殘差區塊與正規化

白話版。不要求一個區塊計算整個輸出，而是要它計算對輸入的修正量。

殘差區塊計算 $y = x + F_\theta(x)$，其中 F_θ 通常是兩到三個卷積，搭配正規化與 ReLU。它的雅可比矩陣為 $\partial y/\partial x = I + \partial F_\theta/\partial x$：即使 F_θ 貢獻很小，恆等路徑仍能把訊號往前送、把梯度往回傳。若 $F_\theta \approx 0$ 就是一個區塊所能做到最好的，它也能輕易學會，所以增加區塊不再傷害最佳化。當形狀改變（更多通道、步幅二）時，捷徑變成投影 $P(x)$，通常是帶步幅的 1×1 卷積。

批次正規化（batch normalization）[19] 在訓練時用批次平均與變異數標準化每個通道，再套用學習出來的縮放與平移；評估時則使用移動平均。它讓訓練更快、對初始化較不敏感，但在批次非常小或相關性高時表現不佳，此時**群組正規化**（GroupNorm）[6]（在每個樣本內對通道群組計算統計量）較可靠。驗證之前一定要呼叫 `model.eval()`，讓 BatchNorm 使用移動統計量。

```
import torch
from torch import nn

count = lambda m: sum(p.numel() for p in m.parameters())

C = 64
dense = nn.Conv2d(C, C, 3, padding=1, bias=False)
separable = nn.Sequential(
    nn.Conv2d(C, C, 3, padding=1, groups=C, bias=False),   # depthwise: one 3x3 per channel
    nn.Conv2d(C, C, 1, bias=False))                        # pointwise: mix channels
print("dense:", count(dense), "| depthwise+pointwise:", count(separable),
      "| ratio:", round(count(separable) / count(dense), 4), "vs 1/C + 1/9 =",
      round(1 / C + 1 / 9, 4))

class BasicBlock(nn.Module):
    """ResNet-style block: y = relu(x + F(x)), with a 1x1 projection when shapes change."""
    def __init__(self, c_in, c_out, stride=1):
        super().__init__()
        self.f = nn.Sequential(
            nn.Conv2d(c_in, c_out, 3, stride, 1, bias=False), nn.BatchNorm2d(c_out),
            ↪  nn.ReLU(),
```

```
            nn.Conv2d(c_out, c_out, 3, 1, 1, bias=False), nn.BatchNorm2d(c_out))
        self.proj = (nn.Identity() if stride == 1 and c_in == c_out else
                     nn.Sequential(nn.Conv2d(c_in, c_out, 1, stride, bias=False),
                                   nn.BatchNorm2d(c_out)))
    def forward(self, x):
        return torch.relu(self.proj(x) + self.f(x))

x = torch.randn(2, 64, 32, 32)
print("same shape :", tuple(BasicBlock(64, 64)(x).shape))
print("downsample :", tuple(BasicBlock(64, 128, stride=2)(x).shape))
```

輸出顯示密集網路層有 36,864 個權重，可分離組合則有 4,672 個，比例 0.1267 與下面的公式一致；區塊輸出形狀分別為 $64 \times 32 \times 32$ 與 $128 \times 16 \times 16$。

高效率 CNN：把空間濾波與通道混合分開

白話版。「看周圍」（空間濾波）與「組合特徵」（通道混合）不必在同一個昂貴的運算中完成。

密集的 $k \times k$ 卷積有 $k^2 C_{\text{in}} C_{\text{out}}$ 個權重。**深度**（depthwise）卷積分別濾波每個通道（$k^2 C_{\text{in}}$ 個權重，PyTorch 中為 `groups=C`）；接著的**逐點**（pointwise）1×1 卷積混合通道（$C_{\text{in}} C_{\text{out}}$ 個權重）。當 $C_{\text{in}} = C_{\text{out}} = C$ 時，

$$\frac{\text{depthwise} + \text{pointwise}}{\text{dense}} = \frac{k^2 C + C^2}{k^2 C^2} = \frac{1}{C} + \frac{1}{k^2}.$$

當 $k = 3$ 且 C 很大時，這個比例趨近 $1/9$。

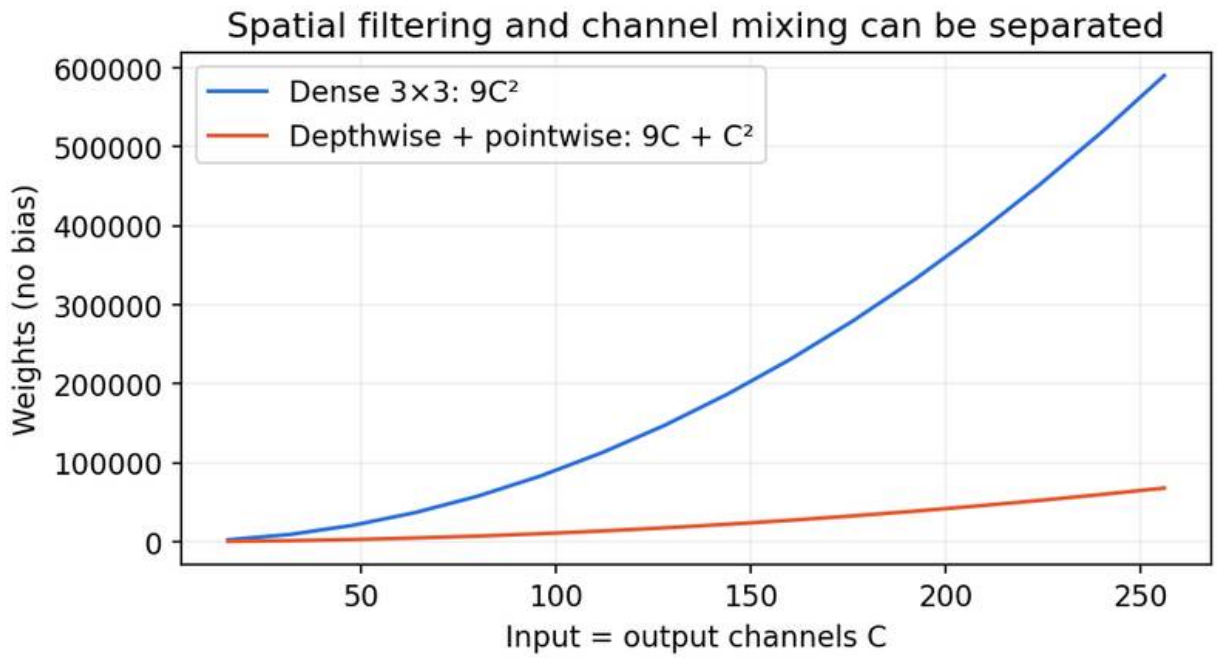


圖 *9* — 輸入與輸出通道數相同時的權重數解析計算。權重較少不代表延遲一定較低：深度卷積的運算強度低，在某些硬體上可能跑得很慢。

MobileNetV2 [7] 的區塊先用 1×1 網路層擴張通道，以深度 3×3 濾波，再以線性（無 ReLU）1×1 瓶頸投影回去，並在兩個窄端之間加上殘差連接。EfficientNet [8] 用單一複合係數同時縮放深度、寬度與輸入解析度。ConvNeXt [9] 從 ResNet 出發，一次一項地套用與視覺 Transformer 相關的設計選擇：patchify 起始層、7×7 深度卷積核、反轉瓶頸、較少的激勵函數、LayerNorm，以及搭配大量資料擴增的較長訓練。結果與相近大小的 Swin Transformer 相當，顯示 Transformer 的優勢有很大一部分來自訓練方法與巨觀設計，而不只是注意力本身。

僅線上版：*ResNet：學習殘差而非整個映射* — *https://blog.shyandram.dev/zh-tw/learn/ml/convolutional-networks/papers/resnet 學習殘差而非整個映射/*

程式碼：完整的小影像分類器

這個範例需要 `torch`、`numpy` 與 `scikit-learn`。它使用內建的 8×8 數字資料集，因此不需要下載或 GPU。分層切分得到彼此獨立的訓練、驗證與測試集：驗證集用來挑選最佳的訓練週期，測試集只在最後評估一次。這是教學用的基準，而不是 ImageNet 的訓練方法。

```
import copy
import numpy as np
import torch
from torch import nn
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

torch.manual_seed(0)
torch.set_num_threads(1)
d = load_digits()
X = torch.tensor(d.images[:, None] / 16.0, dtype=torch.float32)
y = torch.tensor(d.target, dtype=torch.long)
train, test = train_test_split(np.arange(len(y)), test_size=.2,
                               stratify=d.target, random_state=0)
train, val = train_test_split(train, test_size=.2,
                              stratify=d.target[train], random_state=1)
model = nn.Sequential(
    nn.Conv2d(1, 16, 3, padding=1), nn.ReLU(),
    nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(),
    nn.MaxPool2d(2),                       # B × 32 × 4 × 4
    nn.Flatten(), nn.Linear(32 * 4 * 4, 10),
)
opt = torch.optim.AdamW(model.parameters(), lr=1e-3, weight_decay=1e-4)
best_loss, best_state = float("inf"), None
for epoch in range(12):
    model.train()
    order = torch.tensor(train)[torch.randperm(len(train))]
    for idx in order.split(64):
        loss = nn.functional.cross_entropy(model(X[idx]), y[idx])
        opt.zero_grad()
        loss.backward()
        opt.step()
```

```
    model.eval()
    with torch.no_grad():
        val_loss = nn.functional.cross_entropy(model(X[val]), y[val]).item()
    if val_loss < best_loss:
        best_loss = val_loss
        best_state = copy.deepcopy(model.state_dict())
model.load_state_dict(best_state)
model.eval()
with torch.no_grad():
    accuracy = (model(X[test]).argmax(1) == y[test]).float().mean().item()
print("parameters:", sum(p.numel() for p in model.parameters()))
print("best validation loss:", round(best_loss, 4))
print("held-out accuracy:", round(accuracy, 4))
```

在我們的 CPU 上執行，它回報 9,930 個參數，保留測試集準確率約 96%（確切數字會因 PyTorch 版本略有差異）。交叉熵接收的是 **logits**，前面不接 softmax。頭部把 $32 \times 4 \times 4$ 特徵圖攤平，而不是做全域平均池化，因為在很小的數字影像上，筆畫的位置本身就帶有資訊；對於大小不一的影像，自適應或全域池化能提供固定大小的頭部。

要檢查錯誤，而不只看分數。哪些數字配對容易混淆？把每個測試數字平移一個像素會不會變差？在把所有進步都歸功於卷積之前，先與參數量相當的 MLP 以及古典 SVM 比較，如 IP 13 所示。

現代觀點

綜述文獻。有幾篇綜述描繪了 CNN 的全貌，各有不同重點。Gu 等人 [23]（*Pattern Recognition*，2018）依元件整理 CNN 的進展：卷積變體、池化、激勵函數、損失函數、正規化與最佳化，接著是應用。Khan 等人 [22]（*Artificial Intelligence Review*，2020）依主要創新提出架構分類法：空間利用（卷積核大小）、深度、多路徑設計（捷徑、Inception 分支）、寬度、特徵圖利用（通道注意力）以及基於注意力的設計。Li 等人 [21]（*IEEE TNNLS*，2022）涵蓋從 LeNet 開始的歷史，比較一維、二維與多維卷積，並綜覽輕量化網路與應用，給出選擇超參數的經驗法則。Alzubaidi 等人 [2]（*Journal of Big Data*，2021）以 CNN 架構為中心，對深度學習做了廣泛、應用導向的回顧，並討論資料稀少、遷移學習與硬體。這些綜述都早於 ConvNeXt，因此請把它們當作 2012–2020 年代的地圖來讀。

Transformer 之後的 CNN（2022–2026）。視覺 Transformer（ML 05）促使學界重新檢視卷積，主要發現是兩者的差距比早期比較所顯示的要小：

- **現代化的 CNN**。 ConvNeXt [9] 證明經過細心現代化的 ResNet 能與 Swin 相當。ConvNeXt V2 [24]（CVPR 2023）加入以稀疏卷積實作的遮罩自編碼器預訓練，以及全域響應正規化層，把自監督預訓練（見 ML 06）帶進純 CNN。
- **大型卷積核**。 RepLKNet [26]（CVPR 2022）使用大到 31×31 的深度卷積核，並以重新參數化的小卷積核降低最佳化難度，藉此擴大有效感受野。UniRepLKNet [27]（CVPR 2024）把大卷積核設計延伸到音訊、影片、點雲與時間序列。
- **動態與可變形卷積**。 InternImage [25]（CVPR 2023）把取樣位置由輸入預測的可變形卷積，擴展成用於偵測與分割的大型基礎骨幹網路。

- **規模**。Smith 等人 [29]（2023）在大型 JFT-4B 影像資料集上預訓練 NFNet CNN，發現在可比較的預算下，其效能遵循與視覺 Transformer 相似的運算量縮放趨勢，主張在大規模下主導結果的是運算量與資料，而不是架構。
- **邊緣裝置**。MobileNetV4 [28]（ECCV 2024）設計了在 CPU、GPU、DSP 與加速器上都有良好延遲的區塊與神經架構搜尋，承認光看參數量無法預測速度。

CNN 在 2026 年的定位。卷積仍是下列情境的預設選擇：受延遲限制與裝置端的推論、高解析度的密集預測（復原、分割、醫學影像，見 DLCV 09）、局部性偏差有幫助的小資料集，以及 CNN–Transformer 混合模型的起始層或前段。大型視覺基礎模型多半以 Transformer 為主，但它們的 patch 嵌入、上採樣解碼器與擴散 U-Net 中經常包含卷積。

開放問題。下採樣下的精確平移等變性（混疊 [20]）；以低成本內建旋轉與尺度對稱性；理解一個任務真正需要多大的有效感受野 [3]；以及架構之間的公平比較，這需要先對齊資料、資料擴增、解析度、訓練排程與運算量，才能把差異歸因於卷積或注意力。

重點整理

- 卷積會翻轉卷積核，互相關不會；重現濾波器時須檢查函式庫規約。
- 卷積核總和決定對常數輸入的響應，但總和為一不代表一定平滑。
- 學得的濾波器在空間上共用權重，通道分組決定各濾波器讀取哪些輸入。
- 填補、步幅與膨脹決定形狀及取樣，下採樣可能引入混疊。
- 平移等變性不同於不變性；邊界、步幅與池化會影響實際行為。
- 比較架構時，分開看空間濾波、通道混合與跳接，並在相同預算下量測效率。

練習

1. **設計並驗證**。寫出一個只偵測水平邊緣的 3×3 卷積核，以及另一個只偵測從左上到右下對角邊緣的卷積核。在卷積核實驗室中，或用 `cv2.filter2D` 在一張含有水平線、垂直線與對角線的合成影像上驗證兩者。

   提示

   水平邊緣是沿 y 方向的變化，所以使用 $S_y = S_x^\top$（各列為 $-1,-2,-1$ / $0,0,0$ / $1,2,1$）。對角邊緣則把導數放在跨越對角線的方向，例如 $\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$。它的權重總和為零，所以平坦區域輸出為零。

2. **卷積核總和**。為什麼模糊卷積核的權重總和應該為一？先預測再檢查：一張中灰色的 `uint8` 影像經過權重都等於 $2/9$ 的 3×3 卷積核濾波後會怎樣？若改用 `float32` 濾波又有什麼不同？

   提示

   數值為 c 的平坦區域會映射到 $c \sum K$。總和為二時，灰階 128 會變成 256，在 `uint8` 中飽和為 255；所有比中灰亮的區域都變成白色。在 `float32` 中數值加倍但不會被截斷，之後可以再重新縮放。

3. **組合**。證明連續做兩次 3×3 方框模糊等於一個 5×5 卷積核，並寫出這個卷積核。在遠離邊界處以數值驗證，並解釋為什麼在邊界處等式可能不成立。

   提示

   依結合律，組合後的卷積核是方框核與自己的卷積：$[1,2,3,2,1]$ 與自己的外積再除以 81。在邊界處，第一次濾波已經編造了填補值，第二次再對那個結果填補；而單次 5×5 濾波填補的是原始影像，所以兩者在寬度為二的外框中不同。

4. **網路層算術**。對 64×64 的 RGB 輸入，以及一個有 16 個濾波器、$k=5$、$p=2$、$s=2$ 的網路層：寫出輸出形狀與參數數量。接著計算三個步幅分別為 1、2、1 的 3×3 網路層之後的 (r,j)，並與所有步幅都為一的情況比較。

   提示

   $\lfloor(64+4-4-1)/2\rfloor+1=32$，所以輸出為 $16\times32\times32$，參數為 $16(3\cdot25+1)=1216$。感受野：$(3,1)\to(5,2)\to(9,2)$。步幅全為一則得到 $(7,1)$。

5. **手算梯度**。對 1×4 的輸入 $X=[x_1,x_2,x_3,x_4]$ 與卷積核 $K=[k_1,k_2]$ 做 valid 互相關，寫出 Y，再以 $G=\partial L/\partial Y$ 推導 $\partial L/\partial K$ 與 $\partial L/\partial X$。用反向傳播程式碼中的 autograd 技巧檢查你的公式。

   提示

   $Y=[k_1x_1+k_2x_2,\ k_1x_2+k_2x_3,\ k_1x_3+k_2x_4]$。於是 $\partial L/\partial k_1=G_1x_1+G_2x_2+G_3x_3$、$\partial L/\partial k_2=G_1x_2+G_2x_3+G_3x_4$（$X$ 與 G 的互相關）。$\partial L/\partial X=[G_1k_1,\ G_1k_2+G_2k_1,\ G_2k_2+G_3k_1,\ G_3k_2]$，即 G 與 K 的 full 卷積。

6. **數字分類的消融實驗**。在分類器中，把 `Flatten + Linear(512, 10)` 換成全域平均池化 + `Linear(32, 10)`。比較參數量、驗證損失與測試準確率，再用平移一個像素的測試數字評估兩個模型。解釋你觀察到的取捨。

   提示

   使用 `nn.AdaptiveAvgPool2d(1)` 再接 `nn.Flatten()`。池化移除了大部分頭部參數與位置資訊，可能有助於對平移的穩健性，但在筆畫位置很重要的 8×8 數字上可能變差。其他設定保持不變，只用驗證集做選擇，並同時回報準確率與參數量。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 9. Book
2. L. Alzubaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of Big Data*, 2021. Review
3. W. Luo, Y. Li, R. Urtasun and R. Zemel, “Understanding the Effective Receptive Field in Deep Convolutional Neural Networks,” NeurIPS, 2016. Paper

4. O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” MICCAI, 2015. Paper
5. K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” CVPR, 2016. Paper
6. Y. Wu and K. He, “Group Normalization,” ECCV, 2018. Paper
7. M. Sandler et al., “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” CVPR, 2018. Paper
8. M. Tan and Q. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” ICML, 2019. Paper
9. Z. Liu et al., “A ConvNet for the 2020s,” CVPR, 2022. Paper
10. S. J. D. Prince, *Understanding Deep Learning*, MIT Press, 2023, ch. 10. Book and notebooks
11. OpenCV, “Image Filtering” (`filter2D`, border types), documentation, version 4.x. Documentation
12. V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” arXiv:1603.07285, 2016. arXiv
13. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, 1998. DOI
14. A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” NeurIPS, 2012. Paper
15. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” ICLR, 2015. arXiv
16. C. Szegedy et al., “Going Deeper with Convolutions,” CVPR, 2015. arXiv
17. M. Lin, Q. Chen and S. Yan, “Network In Network,” ICLR, 2014. arXiv
18. F. Yu and V. Koltun, “Multi-Scale Context Aggregation by Dilated Convolutions,” ICLR, 2016. arXiv
19. S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” ICML, 2015. arXiv
20. R. Zhang, “Making Convolutional Networks Shift-Invariant Again,” ICML, 2019. arXiv
21. Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE Transactions on Neural Networks and Learning Systems*, 2022. DOI
22. A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” *Artificial Intelligence Review*, 2020. arXiv
23. J. Gu et al., “Recent Advances in Convolutional Neural Networks,” *Pattern Recognition*, 2018. arXiv
24. S. Woo et al., “ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders,” CVPR, 2023. arXiv
25. W. Wang et al., “InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions,” CVPR, 2023. arXiv
26. X. Ding et al., “Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs” (RepLKNet), CVPR, 2022. arXiv
27. X. Ding et al., “UniRepLKNet: A Universal Perception Large-Kernel ConvNet for Audio, Video, Point Cloud, Time-Series and Image Recognition,” CVPR, 2024. arXiv
28. D. Qin et al., “MobileNetV4: Universal Models for the Mobile Ecosystem,” ECCV, 2024. arXiv

29. S. L. Smith, A. Brock, L. Berrada and S. De, “ConvNets Match Vision Transformers at Scale,” arXiv:2310.16764, 2023. arXiv
30. M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” ECCV, 2014. arXiv
31. PyTorch, `torch.nn.Conv2d` documentation. Documentation

ML 04・循環與遞迴神經網路

從序列與樹結構學習：共享的循環狀態、時間反向傳播、梯度消失與爆炸、LSTM 與 GRU 閘門、雙向與編碼器–解碼器模型、作為通往 Transformer 橋梁的注意力機制、遞迴網路，以及它們在電腦視覺中的應用。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/recurrent-recursive-networks/

你將學到

- 展開循環更新，追蹤共用參數。
- 說明時間反向傳播與長距離梯度困難。
- 比較 LSTM 與 GRU 的記憶路徑及閘門。
- 區分雙向、序列到序列與樹狀模型。
- 從上下文、串流與記憶體需求評估注意力及狀態空間方法。

先看全貌

ML 03 對影像只用了一個假設：相鄰像素彼此相關，而且同一個圖樣可能出現在任何位置，所以同一個濾波器可以在**空間**上重複使用。序列也適用同樣的技巧，只是換成在**時間**上重複使用。影片是一連串影格，圖說是一連串字詞，筆跡是一連串座標。**循環神經網路**（recurrent neural network, RNN）一次讀入序列中的一個元素，把一份稱為**隱藏狀態**（hidden state）的摘要從這一步帶到下一步，並在每一步使用相同的權重 [1]。**遞迴神經網路**（recursive neural network）則把這個想法從鏈推廣到樹：它反覆套用同一條共享的組合規則，把局部拼成整體。

中文文獻常把這兩個英文名詞混用；本篇固定以「循環神經網路」指 recurrent neural network、以「遞迴神經網路」指 recursive neural network，以免混淆。

> **用最簡單的話說**
>
> 聽故事時，你不會記住每一個字，而是在腦中保留一幅「到目前為止發生了什麼」的小圖畫，每聽到新的一句就更新這幅圖。循環網路就是這樣做的：它有一本小筆記本（隱藏狀態），一次讀一個字或一張影格，每讀完一個就用同一套規則改寫筆記。LSTM 是一本附了橡皮擦和筆的筆記本，網路會學著小心使用它們，讓開頭的重要筆記到了最後也不會被塗糊。遞迴網路則像玩 LEGO：把兩塊小積木扣成一塊大的，再把大的扣成更大的，每次都用同一種扣法。

既然 Transformer 已經主宰大規模的語言與視覺任務，為什麼還要學 RNN？有三個理由。第一，現代序列建模的每一個概念——教師強制、編碼器–解碼器、注意力、遮罩、因果式生成——都是在 RNN 上發展出來的，而把 Transformer 理解成「對 RNN 局限的回答」是最容易的理解方式。第二，循環結構很適合**串流**處理：每一步只需要固定大小的狀態，適合機器人、感測器和裝置端的影片。第三，循環結構回來了：Mamba 等現代狀態空間模型本質上就是參數化方式很巧妙的循環式，而 ML 05 會把它們應用到影像上。

循環狀態

白話版。在每一步，把目前的輸入和前一步的狀態結合成新的狀態，每次都使用相同的權重。需要輸出時，就從狀態讀出來。

精確版。設輸入序列為 $x_1, \dots, x_T$，其中 $x_t \in \mathbb{R}^d$；隱藏狀態為 $h_t \in \mathbb{R}^m$，起始狀態 h_0 固定（通常是零）。簡單（Elman [9]）RNN 為

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b), \qquad \hat{y}_t = g\big(W_y h_t + c\big).$$

$W_x \in \mathbb{R}^{m\times d}$ 把輸入映射到狀態空間，$W_h \in \mathbb{R}^{m\times m}$ 是把前一步資訊帶過來的**循環**權重矩陣，b 與 c 是偏差，W_y 把狀態映射成輸出，g 是輸出函數，例如 softmax。關鍵在於這些參數都不依賴 t。這就是**跨時間的參數共享**，相當於 CNN 權重共享在時間上的版本：不論序列有 5 步還是 5,000 步，模型的參數數量都相同，而在第 3 步學到的圖樣也能在第 300 步被辨認出來。

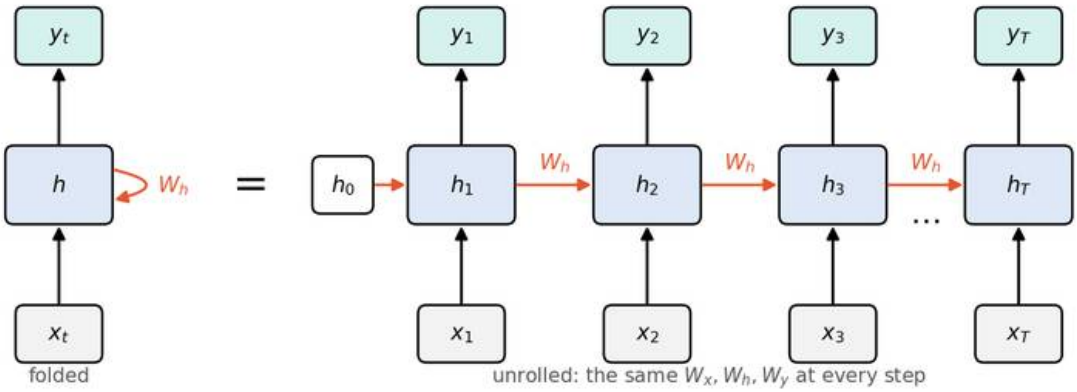


圖 *1 — RNN* 的折疊與展開兩種畫法。展開後，迴圈變成一個很深的前饋網路，而且每一層共用同一組權重。

展開（unrolling，圖 1）是最關鍵的思考步驟。把迴圈寫開 T 步，就得到一個 T 層深的普通前饋網路，而每一層的權重都相同。你在 ML 02 學到的反向傳播，全部都能套用在這張展開後的計算圖上。

同一個循環式可以支援好幾種輸入–輸出型態：

型態	例子	從哪裡讀取輸出
多對一	分類一段影片；判斷一句話的情感	最後的狀態 h_T（或池化後的狀態）
多對多、對齊	為每張影格貼標籤；為每個字詞標註詞性	每一個 h_t
一對多	從一個影像向量生成圖說	一個把自己的輸出再餵回去的解碼器
多對多、不對齊	翻譯一句話；描述一段影片	**編碼器–解碼器**（稍後介紹）

由 $x_{\le t}$ 預測 x_{t+1} 的 RNN 是一個**自迴歸**（autoregressive）模型，從中取樣就能生成新的序列；但當作編碼器使用的 RNN 並不會自動變成生成器。決定任務的是輸出頭與損失函數。

程式碼：用 NumPy 寫出 RNN 與它的反向傳播

下面的程式碼手動實作前向傳遞與時間反向傳播，損失為 $L = v^\top h_T$，再用有限差分檢查其中一個梯度元素。請注意 `dWh += …`這一行：因為 W_h 在每一步都被使用，它的梯度是所有步貢獻的**總和**。

```
import numpy as np

rng = np.random.default_rng(0)
d, m, T = 3, 5, 7                                  # input size, state size, length
Wx = rng.normal(0, 0.5, (m, d))
Wh = rng.normal(0, 0.5, (m, m))
b = np.zeros(m)

def rnn_forward(xs, h0):
    hs = [h0]
    for x in xs:                                    # the SAME Wx, Wh, b at every step
        hs.append(np.tanh(Wx @ x + Wh @ hs[-1] + b))
    return hs

def rnn_backward(xs, hs, dhT):
    """BPTT for the loss L = dhT . h_T; returns dL/dWh and dL/dx_t for every t."""
    dWh, dxs, dh = np.zeros_like(Wh), [None] * len(xs), dhT
    for t in reversed(range(len(xs))):
        da = dh * (1 - hs[t + 1] ** 2)              # through tanh
        dWh += np.outer(da, hs[t])                  # shared weight: gradients ADD up
        dxs[t] = Wx.T @ da
        dh = Wh.T @ da                              # pass the gradient one step back
    return dWh, dxs

xs = rng.normal(size=(T, d))
hs = rnn_forward(xs, np.zeros(m))
dhT = rng.normal(size=m)
dWh, dxs = rnn_backward(xs, hs, dhT)

# finite-difference check of one entry of dL/dWh
eps, (i, j) = 1e-6, (2, 3)
Wh[i, j] += eps; up = dhT @ rnn_forward(xs, np.zeros(m))[-1]
Wh[i, j] -= 2 * eps; down = dhT @ rnn_forward(xs, np.zeros(m))[-1]
Wh[i, j] += eps
print("BPTT vs finite difference:", np.isclose(dWh[i, j], (up - down) / (2 * eps)))
print("|dL/dx_t| from first to last step:", np.round([np.linalg.norm(g) for g in dxs], 3))
```

它會印出 `True`，而輸入梯度的範數為 $[0.009, 0.015, 0.006, 0.029, 0.383, 0.212, 0.433]$：即使序列只有七步，早期輸入收到的梯度也比後期小了大約十到五十倍。這就是縮小版的梯度消失問題。

時間反向傳播

白話版。把網路展開，在展開後的計算圖上執行普通的反向傳播，再把每個共享權重的梯度加總起來。

精確版。把激活前的值寫成 $a_t = W_x x_t + W_h h_{t-1} + b$，因此 $h_t = \tanh(a_t)$。假設損失是各步的總和 $L = \sum_t L_t(h_t)$。從 $t = T$ 往回推，抵達 h_t 的總梯度為

$$\frac{\partial L}{\partial h_t} = \frac{\partial L_t}{\partial h_t} + \left(\frac{\partial h_{t+1}}{\partial h_t}\right)^{\top} \frac{\partial L}{\partial h_{t+1}}, \qquad \frac{\partial h_{t+1}}{\partial h_t} = \operatorname{diag}(1 - h_{t+1}^2)\, W_h.$$

第一項是第 t 步的局部損失；第二項是 h_t 影響到的所有後續部分，往回傳了一步。令 $\delta_t = (1 - h_t^2) \odot \partial L / \partial h_t$（激活前的梯度），參數梯度就是對時間的總和：

$$\frac{\partial L}{\partial W_h} = \sum_{t=1}^{T} \delta_t\, h_{t-1}^{\top}, \qquad \frac{\partial L}{\partial W_x} = \sum_{t=1}^{T} \delta_t\, x_t^{\top}, \qquad \frac{\partial L}{\partial b} = \sum_{t=1}^{T} \delta_t.$$

這個演算法稱為**時間反向傳播（BPTT）**，Werbos [8] 對它有清楚的闡述。它的計算量與 T 成線性，但記憶體也一樣：每個 h_t 都必須保存下來，供反向傳遞使用。

截斷式 BPTT

對非常長的資料流（一小時的影片、連續的感測紀錄）而言，保存所有狀態是不可能的。**截斷式 BPTT**（truncated BPTT）把資料流切成每段 k 步的視窗來處理。它會把隱藏狀態從一個視窗帶到下一個視窗，所以模型仍然「看得到」完整的過去；但它在每個視窗邊界把狀態從計算圖上**切斷**（detach），所以梯度最多只往回傳 k 步。因此，模型只有在視窗內的梯度訊號促使它學會保存長距離資訊時，才能利用這些資訊。

```
import torch
from torch import nn

torch.manual_seed(0)
rnn, head = nn.GRU(1, 16, batch_first=True), nn.Linear(16, 1)
params = list(rnn.parameters()) + list(head.parameters())
opt = torch.optim.Adam(params, lr=1e-2)
t = torch.arange(0, 2000) * 0.05
stream = torch.sin(t) + 0.1 * torch.randn_like(t)   # one long noisy signal
x, y = stream[:-1, None][None], stream[1:, None][None]   # predict the next sample

h, window = None, 50
for start in range(0, x.shape[1] - window, window):
    out, h = rnn(x[:, start:start + window], h)
    loss = nn.functional.mse_loss(head(out), y[:, start:start + window])
    opt.zero_grad()
    loss.backward()                       # gradients flow back at most `window` steps
    nn.utils.clip_grad_norm_(params, 1.0)
    opt.step()
    h = h.detach()                        # keep the state, cut the graph
print("final window MSE:", round(loss.item(), 4), "(noise variance is 0.01)")
```

把資料流走過一遍之後，最後一個視窗的均方誤差約為 0.014，接近加入雜訊所設定的 0.01 下限。如果忘了 `h.detach()`，第二個視窗會試圖反向傳播進一張已經被釋放的計算圖，PyTorch 會丟出錯誤；反過來，若在每個視窗都把 `h = None` 重設，就等於把上下文丟掉了。

梯度消失與梯度爆炸

白話版。反向傳播跨時間連乘狀態的 Jacobian；乘積可能快速縮小或放大，使遠距離依賴難以學習。

精確版。把每一步的 Jacobian 串起來，得到

$$\frac{\partial h_t}{\partial h_s} = \prod_{k=s+1}^{t} \frac{\partial h_k}{\partial h_{k-1}} = \prod_{k=s+1}^{t} \operatorname{diag}(1 - h_k^2)\, W_h,$$

這是 $t-s$ 個矩陣依序相乘的乘積（矩陣乘法不可交換，所以順序很重要）。因為 $|\tanh'| \le 1$，

$$\left\| \frac{\partial h_t}{\partial h_s} \right\| \le \left(\sigma_{\max}(W_h) \right)^{t-s},$$

其中 $\sigma_{\max}$ 是最大奇異值。若 $\sigma_{\max}(W_h) < 1$，來自遠處步驟的梯度會以指數速度**消失**。若 W_h 有很大的奇異值、而且神經元沒有飽和，梯度就可能**爆炸**。Bengio、Simard 與 Frasconi [6] 在 1994 年證明，這不是某種特定訓練方法的缺陷：一個能在雜訊下穩健保存資訊的系統，必然處於長距離依賴梯度會縮小的狀態。Pascanu、Mikolov 與 Bengio [7] 用動態系統的工具重新分析這個問題，並提出處理梯度爆炸的實用方法。

用純量線性 RNN $h_t = \lambda h_{t-1} + x_t$ 最容易看出直覺：此時 $\partial h_t / \partial h_s = \lambda^{t-s}$；當 $\lambda = 0.9$ 時是 $0.9^{80} \approx 2 \times 10^{-4}$，當 $\lambda = 1.1$ 時是 $1.1^{80} \approx 2 \times 10^{3}$。圖 2 顯示在真實網路中量測到的相同行為。

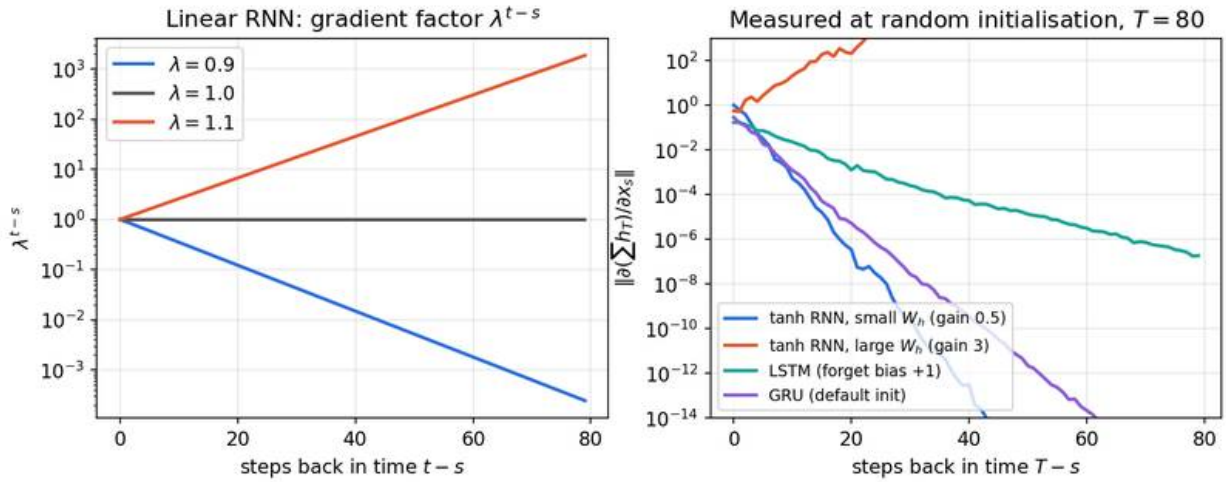


圖 2 — 左：純量因子 λ^(t−s)。右：在隨機初始化下量測最後狀態對每個輸入的梯度（$T = 80$）。單純的 *tanh RNN* 不是消失就是爆炸；*LSTM* 的加法記憶路徑衰減得慢得多。

對策。梯度爆炸用**梯度裁剪**（gradient clipping）[7] 處理：若梯度範數 $\|g\|$ 超過門檻 τ，就把它縮放成 $g \leftarrow \tau\, g / \|g\|$。這樣能保留更新方向、同時限制步長。PyTorch 的 `clip_grad_norm_` 做的正是這件事。梯度消失就比較難處理：需要換一種架構，讓狀態更新接近恆等映射，梯度才能穿過許多步。這種架構就是 LSTM。

長短期記憶（LSTM）

白話版。給網路一條獨立的記憶通道，也就是**細胞狀態**（cell state），它透過加法而不是反覆的矩陣乘法來更新，並讓學習得到的**閘門**（gate）決定要擦掉什麼、寫入什麼、顯露什麼。

精確版。以 $[x_t, h_{t-1}]$ 表示串接、σ 表示 logistic sigmoid，標準的 LSTM 一步計算為

$$
\begin{aligned}
f_t &= \sigma\big(W_f [x_t, h_{t-1}] + b_f\big) && \text{forget gate} \\
i_t &= \sigma(W_i [x_t, h_{t-1}] + b_i) && \text{input gate} \\
o_t &= \sigma(W_o [x_t, h_{t-1}] + b_o) && \text{output gate} \\
\tilde{c}_t &= \tanh(W_c [x_t, h_{t-1}] + b_c) && \text{candidate memory} \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t && \text{cell update} \\
h_t &= o_t \odot \tanh(c_t) && \text{exposed state}
\end{aligned}
$$

每個閘門都是一個介於 0 到 1 之間的數值向量，而 $\odot$ 是逐元素相乘，所以每個閘門就像一排調光開關，每個記憶單元各一個。遺忘閘（forget gate）f_t 決定保留多少舊記憶，輸入閘（input gate）i_t 決定寫入多少候選記憶 $\tilde{c}_t$，輸出閘（output gate）o_t 決定把多少記憶顯露成 h_t。

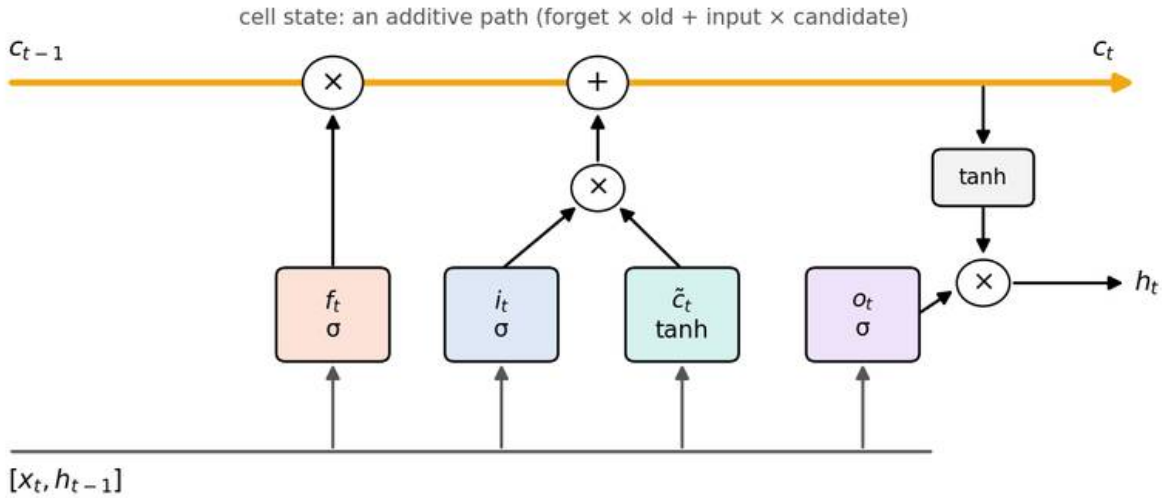


圖 3 — *LSTM* 細胞的內部。細胞狀態（金色）只經過逐元素縮放與加法，這給了梯度一條穿越時間的受保護路徑。

為什麼有效。沿著細胞狀態路徑，$\partial c_t / \partial c_{t-1} = \mathrm{diag}(f_t)$（忽略經過閘門的較小間接路徑）。這條路徑上沒有反覆出現的權重矩陣，也沒有壓縮型的非線性函數。只要遺忘閘維持在接近 1，梯度就能幾乎不變地穿過許多步。這就是圖 2 中 LSTM 曲線衰減得這麼慢的原因，也是常見的初始化會把遺忘閘偏差設成正值（例如 +1）的原因：讓網路一開始就傾向於記住。但這並不保證無限長的記憶：遺忘閘能夠、也確實會學著關閉，而長距離的功勞分配（credit assignment）仍然會隨距離變得更困難。

歷史與出處。 Hochreiter 與 Schmidhuber 在 1997 年提出 LSTM [2]，當時用的是「恆定誤差旋轉木馬」(constant-error carousel)：細胞狀態原封不動地往下傳，**沒有遺忘閘**。Gers、Schmidhuber

與 Cummins 在 2000 年加入遺忘閘 [5]，讓細胞能學會重設記憶，例如在連續資料流中相互獨立的片段之間。上面這組含遺忘閘的方程式，才是實務上與所有主流框架使用的版本。Greff 等人 [19] 在語音、手寫與音樂任務上比較了八種 LSTM 變體，發現沒有一種顯著勝過標準架構，而遺忘閘與輸出激活函數是其中最關鍵的元件。

參數量。四個閘門區塊各有一個 $m \times (d + m)$ 的權重矩陣和一個偏差，所以一層 LSTM 大約有 $4m(d + m + 1)$ 個參數，是相同狀態大小的簡單 RNN 的四倍。（PyTorch 把輸入偏差和循環偏差分開存，所以是 $4m(d + m + 2)$。）

程式碼：從零寫出一步 LSTM，並與 PyTorch 比對

PyTorch 依照輸入閘、遺忘閘、候選細胞（g）、輸出閘的順序把四個閘門區塊堆疊在一起。親手實作一步，再和 `nn.LSTMCell` 比較，是確認自己沒讀錯方程式的最好方法：

```
import torch
from torch import nn

torch.manual_seed(0)
d, m = 4, 6
cell = nn.LSTMCell(d, m)                # PyTorch stacks gate weights in order i, f, g, o

def lstm_step(x, h, c):
    z = x @ cell.weight_ih.T + cell.bias_ih + h @ cell.weight_hh.T + cell.bias_hh
    i, f, g, o = z.chunk(4, dim=-1)
    i, f, o = torch.sigmoid(i), torch.sigmoid(f), torch.sigmoid(o)
    g = torch.tanh(g)                   # candidate memory, often written c~_t
    c = f * c + i * g                   # additive cell update
    h = o * torch.tanh(c)
    return h, c

x = torch.randn(2, d)
h0, c0 = torch.randn(2, m), torch.randn(2, m)
h_ref, c_ref = cell(x, (h0, c0))
h_ours, c_ours = lstm_step(x, h0, c0)
print("matches nn.LSTMCell:", torch.allclose(h_ours, h_ref, atol=1e-6),
      torch.allclose(c_ours, c_ref, atol=1e-6))
print("parameters:", sum(p.numel() for p in cell.parameters()), "= 4 * m * (d + m + 2) =",
      4 * m * (d + m + 2))
```

它會印出 `True True` 以及 `288 = … = 288`。

僅線上版：*LSTM：跨越長時間間隔保存狀態* — *https://blog.shyandram.dev/zh-tw/learn/ml/recurrent-recursive-networks/papers/lstm 跨越長時間間隔保存狀態/*

閘控循環單元（GRU）

白話版。 GRU 是精簡版的 LSTM：它把細胞狀態和隱藏狀態合而為一，並用兩個閘門取代三個。

精確版。依照 Cho 等人 [3] 以及 PyTorch 的 `nn.GRU`：

$$
\begin{aligned}
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) && \text{reset gate} \\
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) && \text{update gate} \\
\tilde{h}_t &= \tanh(W x_t + r_t \odot (U h_{t-1}) + b) && \text{candidate} \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t && \text{interpolation}
\end{aligned}
$$

重設閘（reset gate）r_t 決定在形成候選狀態時使用多少舊狀態；更新閘（update gate）z_t 在保留舊狀態（$z_t \approx 1$）與替換成新狀態（$z_t \approx 0$）之間內插。這個內插扮演了 LSTM 加法細胞路徑的角色。有兩點值得注意。有些論文與教科書會把 z_t 與 $1 - z_t$ 的角色對調；兩種模型等價，但閘門的意義正好相反。此外，原始論文把重設閘放在循環矩陣之前（$U(r_t \odot h_{t-1})$），而 PyTorch 與 cuDNN 則像上式一樣放在之後。在替閘門賦予意義之前，務必先查閱框架的文件。

GRU 有三個閘門區塊而不是四個，所以在相同寬度下參數比 LSTM 少約 25%。兩者都沒有在所有任務上穩定勝出；當資料或運算資源有限時，GRU 是合理的預設選擇。

程式碼：能推廣到新序列的循環分類器

這個完整的 CPU 範例學習判斷序列總和的正負號，必須把全部十二步的資訊累積起來才辦得到。一組全新的測試資料用來確認模型不是只把訓練序列背下來。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
rnn = nn.GRU(input_size=1, hidden_size=24, batch_first=True)
head = nn.Linear(24, 2)
opt = torch.optim.Adam(list(rnn.parameters())+list(head.parameters()), lr=.003)
def batch(n):
    x = torch.randn(n, 12, 1)
    y = (x.sum(dim=(1, 2)) > 0).long()
    return x, y
for step in range(400):
    x, y = batch(64)
    _, h = rnn(x)
    loss = F.cross_entropy(head(h[-1]), y)
    opt.zero_grad(); loss.backward()
    nn.utils.clip_grad_norm_(list(rnn.parameters())+list(head.parameters()), 1.)
    opt.step()
rnn.eval(); head.eval()
with torch.no_grad():
    x, y = batch(512)
    _, h = rnn(x)
    acc = (head(h[-1]).argmax(1) == y).float().mean()
    print("fresh-sequence accuracy:", acc.item())
```

它在長度 12 的新序列上達到約 99% 的準確率。即使設定了 `batch_first=True`，回傳的隱藏狀態張量形狀仍是 `layers × batch × hidden`；這個旗標只影響輸入與輸出序列的排列方式。用

更長的序列測試是在問另一個問題——長度推廣（length generalization）——模型可能在長度 12 成功、卻在長度 100 失敗（見練習 2）。

深層、雙向與串流 RNN

堆疊。深層 RNN 把某一層循環網路輸出的隱藏狀態序列，餵給另一層循環網路（PyTorch 中的 `num_layers`）。較低的層可以捕捉短距離的圖樣，較高的層則捕捉較長距離的圖樣。常見的是二到四層；更深的堆疊通常需要在層與層之間加入殘差連接。

雙向。 Schuster 與 Paliwal [10] 提出讓一個 RNN 沿時間往前跑、另一個往後跑，再把兩者的狀態串接起來：$h_t = [\vec{h}_t; \overleftarrow{h}_t]$。如此每個位置都同時總結了它的過去與未來。這非常適合整段輸入都已取得的**離線**任務（標註錄好的影片、編碼一個句子），但它**不具因果性**：不能用在必須在未來影格抵達之前就做出預測的即時任務。把離線雙向模型的準確率拿來報告線上應用，會高估部署後系統實際能掌握的資訊。

變動長度。一個批次裡的序列長度不一，會被填補（padding）到相同長度。若不小心處理，填補值會滲入狀態與損失。請使用每條序列的實際長度（PyTorch 的 `pack_padded_sequence`）或遮罩，讓填補的步驟既不更新狀態、也不計入損失。

串流。對即時資料流，要在多次呼叫之間保留狀態，只在彼此獨立的資料流之間重設。不小心把某位使用者、某份文件或某段影片的狀態帶到下一個，會洩漏資訊並造成無法重現的結果。

編碼器–解碼器模型與教師強制

白話版。基本編碼器—解碼器先把輸入序列讀成狀態，再據此生成輸出序列；兩者長度可以不同。

精確版。在 Sutskever、Vinyals 與 Le [11] 的**序列到序列**（sequence-to-sequence, seq2seq）模型，以及 Cho 等人 [3] 密切相關的編碼器–解碼器中，**編碼器** RNN 讀入 $x_1, \dots, x_T$，其最終狀態成為上下文向量 $c = h_T$。以 c 初始化的**解碼器** RNN 建模

$$p(y_1, \dots, y_U \mid x) = \prod_{u=1}^{U} p(y_u \mid y_{<u}, c), \qquad s_u = \mathrm{RNN}(s_{u-1}, y_{u-1}),$$

其中 s_u 是解碼器狀態，$p(y_u \mid \cdot)$ 是在詞彙表上的 softmax。當解碼器輸出序列結束符記時生成就停止，所以輸入與輸出的長度互不相關。

教師強制（teacher forcing）。訓練時，解碼器在第 u 步的輸入是**真實答案**中的前一個符記 y_{u-1}，而不是模型自己的預測。這讓所有步驟都能針對已知目標平行訓練，大幅穩定學習。然而在測試時，模型必須吃進自己的預測，而這些預測可能含有訓練時從未見過的錯誤。這種訓練與測試之間的落差稱為**曝光偏差**（exposure bias）。排程取樣（scheduled sampling）[25] 在訓練過程中逐步用模型的取樣取代真實輸入，以減輕這個問題。解碼時通常使用貪婪搜尋或集束搜尋（beam search），而不是隨機取樣。

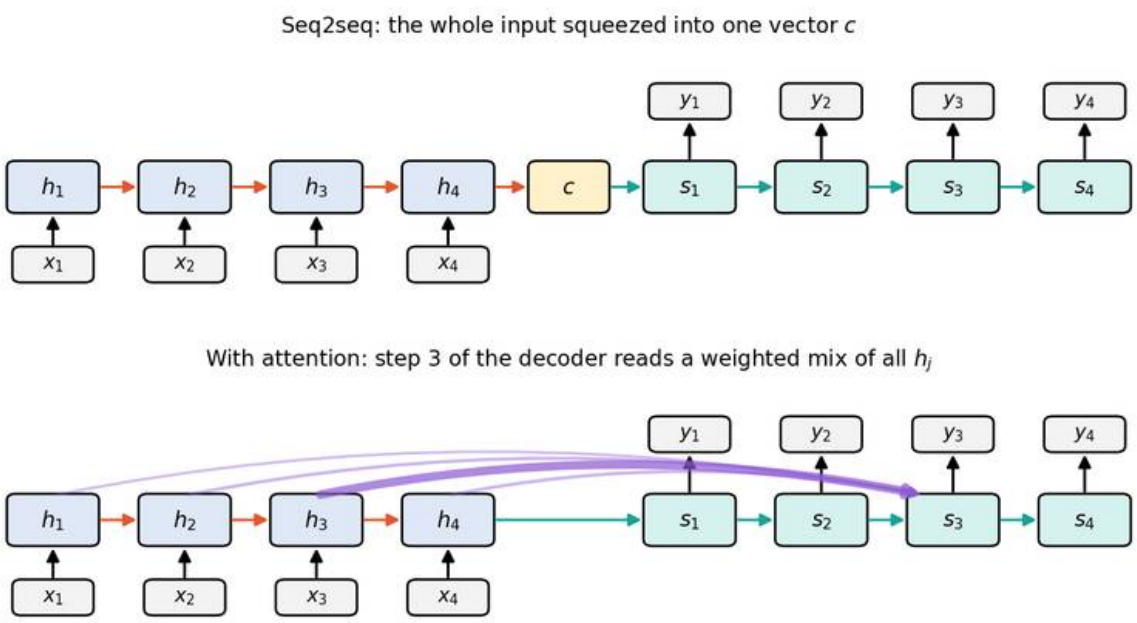


圖 4 — 上：單純的 *seq2seq* 把整段輸入擠進一個向量 c。下：加上注意力後，解碼器的每一步都讀取所有編碼器狀態經學習得到的加權組合。

圖 4（上）清楚顯示了弱點：解碼器對輸入所知道的一切，都必須通過一個固定大小的向量 c，不論輸入有多長。單純 seq2seq 模型的翻譯品質在長句子上會下降。Sutskever 等人發現，單純把**來源句子反轉**就有幫助，因為這縮短了對應字詞之間的距離——這暗示問題出在瓶頸與過長的路徑 [11]。

注意力：通往 Transformer 的橋梁

白話版。注意力讓解碼器在每一步選取相關的編碼器狀態，不必只依賴固定摘要。

精確版。 Bahdanau、Cho 與 Bengio [12] 保留了每一個編碼器狀態 $h_1, \dots, h_T$（來自雙向編碼器），並在解碼器的每一步 u 計算

$$e_{u,j} = v^\top \tanh\!\left(W s_{u-1} + U h_j\right), \qquad \alpha_{u,j} = \frac{\exp(e_{u,j})}{\sum_{k=1}^{T} \exp(e_{u,k})}, \qquad c_u = \sum_{j=1}^{T} \alpha_{u,j}\, h_j .$$

$e_{u,j}$ 是解碼器當下需求（s_{u-1}）與編碼器位置 j 之間經學習得到的**對齊分數**（alignment score）；W、U 與 v 是可學習參數。softmax 把分數轉成皆為正、總和為一的權重 $\alpha_{u,j}$，而**上下文向量** c_u 就是用這些權重對編碼器狀態做的加權平均。解碼器接著把 c_u 與自己的狀態一起使用。這種形式稱為**加法注意力**（additive attention）。因為 c_u 在每一步都會改變，而且能直接存取每一個輸入位置，任何輸入到任何輸出之間的路徑長度都變成一步，與序列長度無關。

```
import torch
from torch import nn
```

```
class AdditiveAttention(nn.Module):
    """Bahdanau-style scores e_j = v^T tanh(W s + U h_j), then softmax over j."""
    def __init__(self, d_dec, d_enc, d_att):
        super().__init__()
        self.W = nn.Linear(d_dec, d_att, bias=False)
        self.U = nn.Linear(d_enc, d_att, bias=False)
        self.v = nn.Linear(d_att, 1, bias=False)

    def forward(self, s, H, mask=None):
        # s: B x d_dec (decoder state), H: B x T x d_enc (all encoder states)
        e = self.v(torch.tanh(self.W(s)[:, None] + self.U(H))).squeeze(-1)   # B x T
        if mask is not None:                        # True where the input is padding
            e = e.masked_fill(mask, float("-inf"))
        alpha = e.softmax(dim=-1)                   # attention weights, rows sum to 1
        context = (alpha[:, :, None] * H).sum(1)    # B x d_enc weighted average
        return context, alpha

torch.manual_seed(0)
att = AdditiveAttention(d_dec=16, d_enc=32, d_att=8)
H = torch.randn(2, 5, 32)
s = torch.randn(2, 16)
pad = torch.tensor([[False] * 5, [False, False, False, True, True]])   # 2nd sequence has
 ↪  length 3
ctx, alpha = att(s, H, pad)
print("context:", tuple(ctx.shape), "| weights sum to 1:",
      torch.allclose(alpha.sum(-1), torch.ones(2)))
print("weights on padding:", alpha[1, 3:].tolist())
```

它會印出形狀為 `(2, 32)` 的上下文，確認權重總和為一，並顯示填補位置的權重恰好為零：在 softmax 之前用 $-\infty$ 遮罩，是讓注意力忽略填補（以及在 ML 05 中忽略未來）的標準做法。

注意力權重也可以解讀成一種軟性的**對齊**（alignment）。圖 5 顯示一個帶有這種注意力的小型 GRU 編碼器–解碼器，在 CPU 上訓練不到一分鐘，學習把隨機數字序列反轉（在教師強制下，對新序列的逐符記準確率約 96%）。沒有人告訴它，它卻學會了在寫第 u 個輸出時去看第 $T+1-u$ 個輸入位置：也就是反對角線。

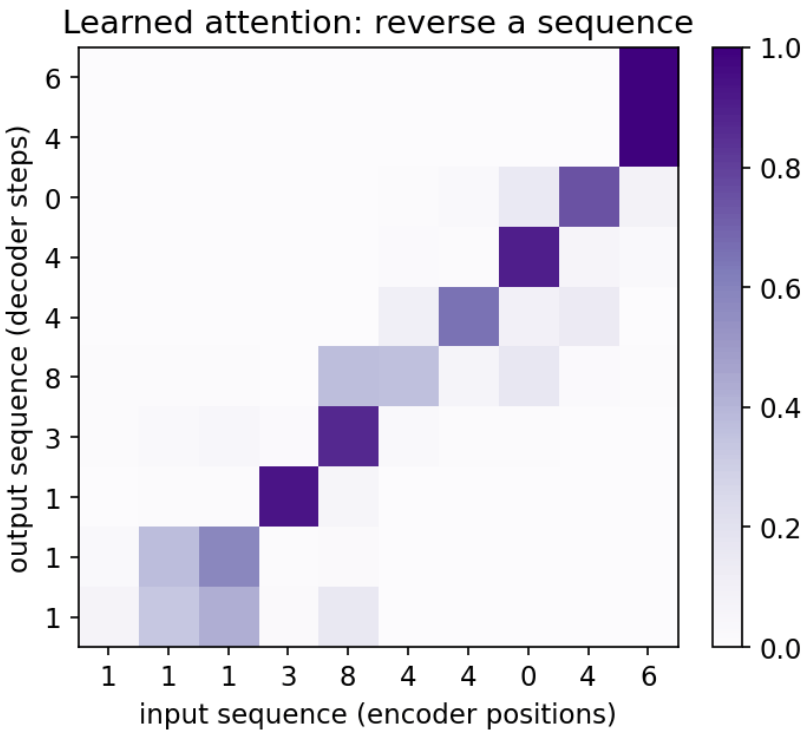


圖 5 — 訓練來反轉序列的玩具編碼器–解碼器的注意力權重（由圖片腳本產生）。每個輸出步驟都集中在鏡射位置的輸入上。

從這裡走向 Transformer。注意力解決了瓶頸，但編碼器與解碼器仍然是循環式的，所以處理長度 T 的序列需要 T 個無法在訓練時平行化的循序步驟。Vaswani 等人 [28] 問：如果完全拿掉循環，**在編碼器與解碼器內部也使用注意力**（自注意力，self-attention），再用位置編碼恢復順序，會怎麼樣？答案就是 Transformer：它能在所有位置上平行訓練，成為語言領域的主流架構，並且隨著 ViT 成為視覺領域的主流架構。ML 05 會從縮放點積注意力（scaled dot-product attention）接續這個故事，它以 $q^\top k/\sqrt{d_k}$ 取代加法分數。

循環網路在電腦視覺中的應用

影像描述（image captioning）。 Show and Tell [13]（Vinyals 等人）是一個直接的影像到序列模型：一個 CNN（ML 03）把影像編碼成一個向量，用它初始化一個 LSTM 語言模型，再由這個語言模型逐字生成圖說，並以教師強制在影像–圖說配對上訓練。Show, Attend and Tell [14]（Xu 等人）保留 CNN 的空間特徵網格而不是單一向量，讓 LSTM 在生成每個字時對影像區域做注意力，所產生的注意力圖常常會凸顯正在被說出名稱的物體。這是最早廣為人知的視覺注意力展示之一。

影片理解。長期循環卷積網路（Long-term Recurrent Convolutional Networks, LRCN）[15]（Donahue 等人）對每張影格套用 CNN、跨影格套用 LSTM，並端到端訓練，用於動作辨識、影像描述與影片描述。CNN 處理空間，RNN 處理時間：這是處理任何逐影格特徵序列的常見模式，例如姿態軌跡，或物件追蹤中的逐影格偵測結果。

時空預測。當狀態本身就應該是一張影像時，全連接的閘門會浪費空間結構。Shi 等人的 **ConvLSTM** [16] 把 LSTM 閘門中的矩陣乘法換成卷積，$i_t = \sigma(W_{xi} * X_t + W_{hi} * H_{t-1} + \dots)$，所以輸入、狀態與閘門都是三維張量（通道 × 高 × 寬）。它最初是為了從雷達圖做降雨即時預報而提出，如今廣泛用於影片預測與其他網格上的預測任務。

發生了什麼改變。在大規模的影像描述與影片分類上，Transformer 與視覺–語言模型（見 ML 10）已大致取代 LSTM，因為它們能平行訓練，而且隨著資料增加擴展得更好。循環與卷積–循環模型仍常見於影格逐張抵達、記憶體必須維持固定的場合：線上影片處理、追蹤、機器人與低功耗裝置。

遞迴網路：沿著樹組合

白話版。遞迴網路沿著一棵樹，套用同一條共享的組合規則，把兩個子節點的表示合併成父節點，直到抵達根節點。

精確版。對一棵葉節點帶有向量（詞嵌入、區域特徵）的二元樹，每個內部節點計算

$$h_{\text{parent}} = \tanh\!\left(W\,[h_{\text{left}}; h_{\text{right}}] + b\right),$$

每個節點都使用**同一個** $W \in \mathbb{R}^{D \times 2D}$ 與 b。根節點向量代表整個結構，可以拿來分類；每個節點也可以有自己的標籤。訓練使用**結構反向傳播**（backpropagation through structure）：和 BPTT 是同一個想法，只是沿著樹的邊進行。長度為 n 的鏈是完全不平衡的樹這個特例；平衡樹把任一葉節點到根節點的路徑從 $n-1$ 步縮短到大約 $\log_2 n$ 步（圖 6），縮短了梯度要走的路。

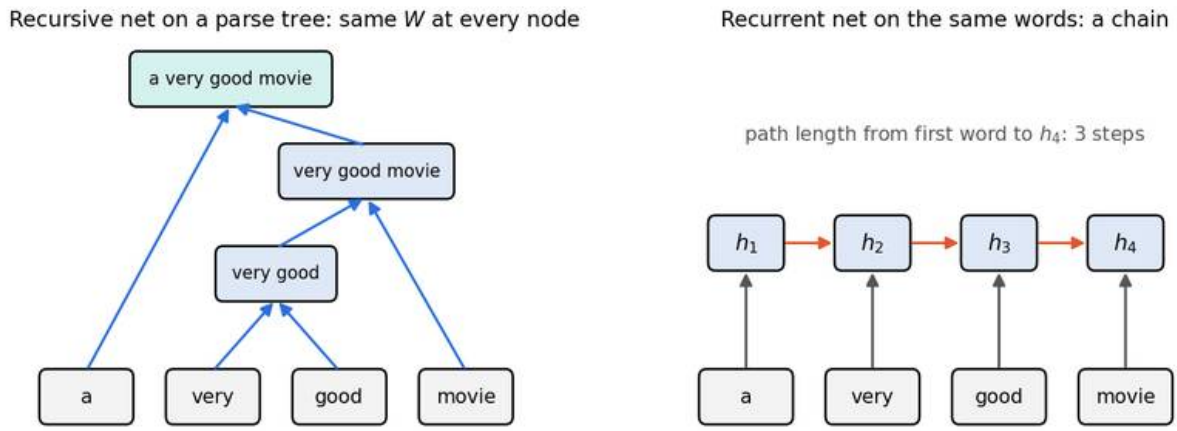


圖 6 — 遞迴網路沿著樹前進（左）；循環網路沿著鏈前進（右）。兩者都共享權重，但樹反映了輸入本身的結構。

Socher 等人 [18] 最早是在視覺情境中使用遞迴網路：他們把影像切成超像素，接著同時學習把相鄰區域合併成場景剖析樹、以及計算區域表示，並用同一套方法剖析句子。他們後來的遞迴神經張量網路（Recursive Neural Tensor Network）與 Stanford Sentiment Treebank [17] 顯示，沿著剖析樹組合能捕捉否定（“not very good”）之類的現象。Tai、Socher 與 Manning 的 **Tree-LSTM** [4] 為樹狀組合加入 LSTM 式的閘門，每個子節點各有一個遺忘閘，讓父節點能有選擇地保留來自每棵子樹的資訊。

```
import torch
from torch import nn

torch.manual_seed(0)
D = 8
vocab = {"a": 0, "very": 1, "good": 2, "movie": 3}
emb = nn.Embedding(len(vocab), D)
compose = nn.Linear(2 * D, D)            # ONE weight matrix shared by every internal node
classify = nn.Linear(D, 2)               # e.g. negative / positive sentiment

def encode(tree):
    """tree is a word (leaf) or a pair (left, right)."""
    if isinstance(tree, str):
        return emb(torch.tensor(vocab[tree]))
    left, right = tree
    return torch.tanh(compose(torch.cat([encode(left), encode(right)])))

tree = ("a", (("very", "good"), "movie"))          # (a ((very good) movie))
root = encode(tree)
loss = nn.functional.cross_entropy(classify(root)[None], torch.tensor([1]))
loss.backward()                                     # backpropagation through structure
print("root vector:", tuple(root.shape), "| loss:", round(loss.item(), 4))
print("gradient reached the shared W:", compose.weight.grad.abs().sum().item() > 0)
swapped = encode((("movie", ("good", "very")), "a"))
print("swapping children changes the root:", not torch.allclose(root, swapped))
```

遞迴就是一般的 Python 遞迴呼叫，而 autograd 會即時建立樹狀的計算圖。交換子節點會改變結果，因為串接讓左右兩個位置扮演不同的角色。

實務上的主要限制在於樹本身：得有人提供它（剖析器、分割階層），而且要有效率地把形狀各異的樹組成批次並不容易。同時學習結構與表示是可行的，但更困難。在目前的實務中，自注意力能在沒有明確樹結構的情況下，學到柔性、隨輸入而變的分組，這也是遞迴網路如今少見的原因之一；不過階層式組合的想法仍延續在階層式視覺骨幹網路與場景圖（scene graph）中。

Mamba：選擇性狀態空間與掃描

有用的記憶應該能在干擾之間保留重要觀察，並在情境改變時覆寫舊資訊。Mamba [21] 讓記憶更新取決於目前輸入，同時維持更新對前一個狀態的仿射形式。這個差別讓模型保有循環的解讀方式，又能用平行前綴掃描（parallel prefix scan）計算整段序列。

從連續動態到選擇性更新

先從小型的連續時間記憶開始。穩定的轉移使儲存的資訊逐漸衰減；輸入投影寫入新資訊；輸出投影讀出記憶：

$$\frac{dh(\tau)}{d\tau} = Ah(\tau) + Bu(\tau), \qquad y(\tau) = Ch(\tau).$$

其中 τ 是連續時間，$u(\tau) \in \mathbb{R}$ 是單一輸入通道，$h(\tau) \in \mathbb{R}^N$ 是該通道的 N 維狀態，$A \in \mathbb{R}^{N\times N}$ 是轉移矩陣，$B \in \mathbb{R}^{N\times 1}$ 負責寫入輸入，$C \in \mathbb{R}^{1\times N}$ 負責讀取狀態，$y(\tau) \in \mathbb{R}$ 是輸出。完整的神經網路區塊會以可學習投影混合多個通道，並加入局部卷積與閘控 [21]。這個方程式只抽出記憶機制。

在每個區間內保持輸入與寫入矩陣不變。精確的零階保持（zero-order hold）離散化得到

$$\bar{A}_t = \exp(\Delta_t A), \qquad \bar{B}_t = \left(\int_0^{\Delta_t} \exp(sA)\, ds \right) B_t,$$
$$h_t = \bar{A}_t h_{t-1} + \bar{B}_t u_t, \qquad y_t = C_t h_t.$$

其中 t 是離散 token 的索引，$\Delta_t > 0$ 是區間長度，s 是積分變數，u_t 是固定的輸入，B_t, C_t 是目前的寫入與讀取矩陣，上橫線表示離散係數。即使 A 是奇異矩陣，積分形式仍然成立；使用反矩陣的公式則需要額外假設。若 A 是對角矩陣，這些運算可以逐元素執行。

選擇性表示從目前 token 預測 Δ_t, B_t, C_t。例如，可學習的仿射投影接上 $\mathrm{softplus}(z) = \log(1 + e^z)$，就能產生正的區間長度。底層可學習的 A 可以保持固定，但 $\bar{A}_t$ 仍會透過 Δ_t 改變。當對角元素為負時，較長的區間會降低舊狀態的保留比例。選擇性的寫入與讀取也決定哪些內容進入或離開記憶。因此，即使每次狀態更新都是仿射的，整體映射對輸入仍是非線性的。圖 7 使用合成係數單獨呈現保留效果，並未量測訓練好的模型。

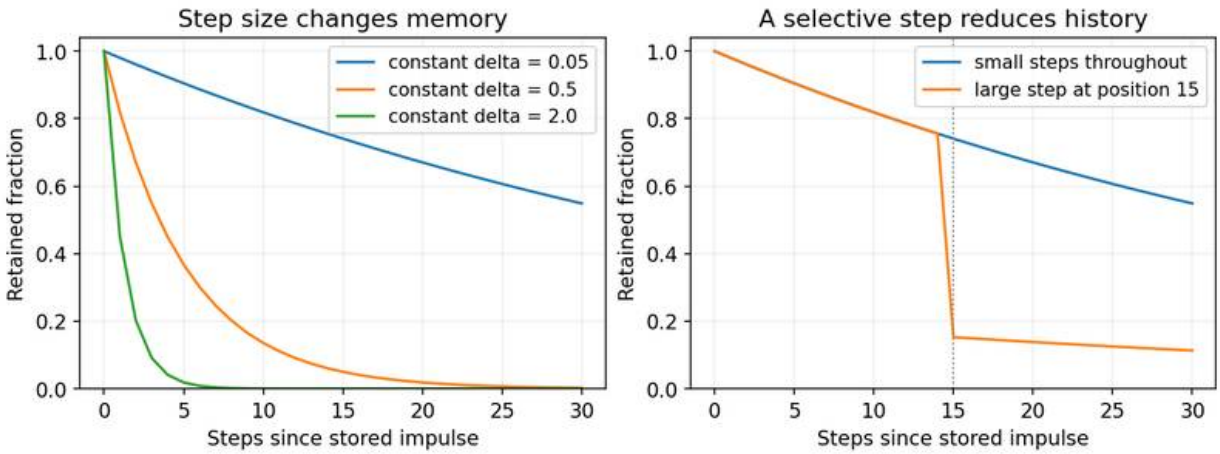


圖 7 — 純量轉移 $A = -0.4$ 時，歷史保留比例是 $exp(-0.4\ delta)$ 的連乘。選定的大區間會明顯降低舊記憶，小區間則能保留得更久。

為什麼這種循環可以掃描

輸入相依的係數算好後，組合兩次仿射更新仍然只需要一次仿射更新：

$$(a_2, b_2) \circ (a_1, b_1) = (a_2 a_1,\ a_2 b_1 + b_2).$$

其中配對 (a_i, b_i) 表示映射 $h \mapsto a_i h + b_i$，$i = 1, 2$ 標示連續的兩次更新，$\circ$ 表示先套用第一次，再套用第二次。組合具有結合律；對角轉移的乘法是逐元素運算。因此，前綴掃描可以把運算重新分組成樹狀結構，卻保留時間順序造成的效果。一般 LSTM ／ GRU 的閘門取決於前一個隱藏狀態，通常無法在狀態尚未算出之前先準備這些係數。

以下可獨立執行的 CPU 範例使用精確的對角離散化，以及手動指定的選擇性係數，檢查因果性與分塊之間的狀態連續性。這是教學用的循環，並非預訓練 Mamba 實作或速度基準。

```
import numpy as np

rng = np.random.default_rng(44)
u = rng.normal(size=24)
A = -np.array([0.1, 0.8])  # two stable diagonal entries

def selective_scan(values, state=None):
    h = np.zeros(2) if state is None else state.copy()
    outputs = []
    for x in values:
        delta = np.logaddexp(0.0, 0.5 * x)
        B = np.array([1.0, np.tanh(x)])
        C = np.array([1.0, 0.5 + 0.1 * np.tanh(x)])
        a = np.exp(delta * A)
        b = np.expm1(delta * A) / A * B * x
        h = a * h + b
        outputs.append(C @ h)
    return np.array(outputs), h

y, final_state = selective_scan(u)
changed = u.copy()
changed[12:] += 10.0
changed_y, _ = selective_scan(changed)
assert np.allclose(y[:12], changed_y[:12])
left, boundary = selective_scan(u[:12])
right, resumed_state = selective_scan(u[12:], boundary)
assert np.allclose(y, np.r_[left, right])
assert np.allclose(final_state, resumed_state)
reset_right, _ = selective_scan(u[12:])
assert not np.allclose(right, reset_right)
print('causal prefix and carried-state chunk checks passed')
```

輸出確認兩項檢查都通過。在批次邊界把狀態歸零會改變計算；反過來，在獨立範例之間共用狀態則會洩漏資訊。串流實作必須追蹤每個狀態屬於哪段序列，並在真正的序列邊界重設。

Mamba-2 與注意力的連結

Mamba-2 [29] 同時改變內部序列混合器與外圍區塊。其結構化狀態空間對偶性（structured state-space duality）在每個頭內使用「純量乘上單位矩陣」的轉移，並把計算切成適合加速器硬體執行密集矩陣乘法的區塊。它也重新安排參數投影與正規化。這比在原始 Mamba 區塊中加入注意力更為具體。

初始狀態為零時，純量轉移的循環可以展開為

$$y_t = \sum_{j=1}^{t} \left[C_t \left(\prod_{k=j+1}^{t} a_k \right) B_j \right] u_j.$$

其中 j 索引先前輸入，k 索引中間的轉移，a_k 是乘在單位矩陣上的離散純量轉移，B_j 是離散寫入列向量，C_t 是讀取行向量，空的連乘等於一。括號內是具有結構的因果混合係數。這提供類似注意力的矩陣觀點，卻不需要一般 softmax 注意力，也不必儲存完整矩陣。Hu 等人 [30]（ICML 2026）把對偶性延伸到對角轉移，並釐清為何無法直接延伸到標準 softmax 注意力。

在固定寬度與狀態維度下，循環推論只攜帶固定大小的狀態；一般具有快取的因果注意力則保留 key 與 value，儲存量隨上下文長度增加。壓縮能省下記憶體，但精確回想許多互不相關細節的能力也受到考驗。序列成本呈線性並不代表訓練記憶體固定：活化值、梯度與重新計算策略都會影響結果。應一起評估檢索品質、上下文長度與裝置實測延遲。影像特有的排序問題接著在 ML 05 討論。

現代觀點

綜述文獻。 Goodfellow 等人書中的第 10 章 [1] 至今仍是循環與遞迴網路、回聲狀態（echo-state）想法與長期依賴問題最清楚的統一論述。Lipton、Berkowitz 與 Elkan [20] 在 2015 年寫了一篇 RNN 序列學習的批判性回顧：以時間順序、自成一體地交代三十年來通往 LSTM 與雙向 RNN 的研究，統一了互相衝突的記號與名稱，並涵蓋影像描述、翻譯與手寫辨識等應用。Greff 等人 [19] 與其說是綜述，不如說是大型實證研究，但它是「LSTM 哪些元件重要」這個問題的參考依據。Tiezzi 等人近年的兩篇回顧，則為 Transformer 時代重新定位循環結構：一篇詳盡的綜述 [26] 把 Transformer–RNN 混合模型、具循環形式的注意力變體，以及深度狀態空間模型（一類特殊的線性 RNN）統一放在「循環」這個大傘之下，並強調從可能無限長的資料流中線上學習、而非處理固定長度序列所帶來的研究機會；另一篇刊登於 *Nature Machine Intelligence* 的較短觀點文章 [27]，則重新檢視位於 Transformer 與狀態空間模型交會處的循環處理。

循環的回歸（2023–2026）。 Transformer 處理所有位置兩兩之間的關係，記憶體與運算量隨長度呈平方成長。一系列模型重新採用循環結構，以固定大小的狀態取得線性時間的推論：

- **線性循環單元**。 Orvieto 等人 [24]（ICML 2023）證明，一個經過精心參數化的**線性**循環式——對角的複數狀態矩陣、穩定的初始化與正規化——在長距離基準測試上可媲美深度狀態空間模型，而且非線性可以放在循環式之外。
- **選擇性狀態空間**。 Mamba [21] 讓狀態空間的參數成為輸入的函數，所以模型能像 LSTM 的閘門那樣選擇要儲存或忽略什麼，同時以考慮硬體特性的平行掃描（parallel scan）維持訓練效率。視覺上的變體，以及它們對影像是否真有必要的爭論，會在 ML 05 討論。
- **復興的 LSTM 與 GRU**。 xLSTM [22]（NeurIPS 2024）為 LSTM 加入指數型閘門與矩陣形式的記憶，並把它擴展到語言模型的規模。Feng 等人 [23]（2024）移除 LSTM 與 GRU 閘門對隱藏狀態的依賴（“minLSTM”、“minGRU”），使訓練時能使用平行掃描，並報告了與近期序列模型相當的結果。

共同的教訓是：為 LSTM 發明的**閘控、加法式狀態更新**是對的；限制經典 RNN 的主要是循序訓練。讓循環式對狀態呈線性（因此能用平行掃描計算），就能恢復平行訓練，同時保留便宜的串流推論。

開放問題。超出訓練長度範圍的長度推廣；在非常長的資料流上、不用完整 BPTT 的功勞分配；固定大小的狀態相較於注意力不斷成長的快取究竟能記住多少——對需要精確回想遠處細節的任務而言，這是實際存在的限制；以及為二維影像與三維影片選擇掃描順序，因為它們沒有唯一自

然的序列順序。

重點整理

- RNN 跨時間重用同一狀態更新規則，共用參數不保證有用的長期記憶。
- BPTT 加總共用參數的梯度；截斷限制梯度回溯範圍，但可繼續傳遞狀態。
- 裁剪限制梯度爆炸，加性閘控路徑有助梯度傳遞，但無法消除所有長距離學習困難。
- 雙向編碼器使用未來輸入；因果串流需要不同的存取與狀態處理。
- 注意力讓解碼器讀取多個編碼器狀態，遞迴網路則沿樹狀結構組合表示。
- 選擇性狀態空間以精簡串流狀態壓縮歷史；須依實際工作負載比較回憶能力與計算。

練習

1. **因果性**。你的團隊報告一個使用雙向 LSTM 的動作辨識模型，逐影格準確率達 92%，並提議把它部署成即時警報系統。這個計畫有什麼問題？你要如何公平地評估可部署的替代方案？執行選擇性掃描範例，再加上反向掃描：哪個因果性檢查應該失敗？

   提示

   第 t 張影格上的反向 LSTM 已經讀過 t 之後的影格，而即時系統還沒看到這些影格。請重新訓練一個單向（因果）模型，或允許固定的前瞻延遲，並以完全相同的延遲做評估。把準確率表示成延遲的函數來報告。反向掃描讀過被改動的未來，因此即使正向前綴不變，合併後的前綴通常仍會改變。

2. **長度推廣**。用長度 12 訓練 GRU 總和分類器，接著在不重新訓練的情況下，用長度 6、24、100 與 400 的新序列評估。畫出準確率對長度的圖。再用 LSTM、以及用長度 6–24 混合訓練，重複一次。

   提示

   每種長度都要產生新的資料。總和的尺度隨 $\sqrt{T}$ 成長，所以即使邏輯正確，狀態在長輸入上也可能飽和。混合長度訓練通常比更換細胞種類更有幫助。在長度 12 上成功，本身並不能說明長度 400 的表現。

3. **梯度上界**。對純量 RNN $h_t = \tanh(w h_{t-1} + x_t)$，證明 $|\partial h_t / \partial h_s| \le |w|^{t-s}$。梯度在 w 取哪些值時可能爆炸？為什麼 tanh 的飽和會抑制爆炸？

   提示

   每個因子都是 $w(1 - h_k^2)$，且 $0 \le 1 - h_k^2 \le 1$。爆炸需要 $|w| > 1$，但很大的 $|w|$ 也會把 h_k 推向 ± 1，使 $1 - h_k^2$ 變小。梯度只會在停留於非飽和區域的軌跡上爆炸，這就是實務上爆炸往往是突發性的原因。

4. **LSTM 的記憶**。在 LSTM 方程式中，假設從 s 到 t 的每一步都有 $f_t = 1$ 且 $i_t = 0$。c_t 用 c_s 表示是多少？沿著細胞路徑的 $\partial c_t / \partial c_s$ 是多少？如果像 1997 年原始設計那樣，把遺忘閘永遠固定為 1，在一個永不結束的輸入流上會發生什麼事？

提示

$c_t = c_s$ 且 $\partial c_t / \partial c_s = I$：完美的記憶與完美的梯度流動。若 $f_t \equiv 1$ 而寫入不為零，細胞狀態會在無止盡的資料流上無限累積，這正是 Gers 等人讓網路學會遺忘所解決的問題。

5. **注意力的形狀與遮罩**。在加法注意力的程式碼中，一個批次的編碼器狀態形狀為 $8 \times 40 \times 256$（其中有些序列經過填補），解碼器狀態形狀為 8×512。請給出 e、α 與上下文的形狀，並說明若用 0 而不是 $-\infty$ 遮罩填補位置，會出什麼問題。

提示

e 與 α 是 8×40；上下文是 8×256。分數 0 只是一個普通的分數：$e^0 = 1$ 仍會給填補位置正的權重，上下文就會混入沒有意義的填補狀態。

6. **遞迴與循環**。對一棵有 $n = 1024$ 個葉節點的平衡二元樹，葉節點與根節點之間有幾個組合步驟？與一個由左到右讀取同樣 1024 個輸入的 RNN 比較，並把答案與梯度消失的上界連結起來。

提示

$\log_2 1024 = 10$ 步，鏈則最多 1023 步。梯度上界的衰減方式是（因子）$^{\text{path length}}$，所以從每個輸入到輸出的路徑越短，越有利於最佳化。注意力把這一點推到極致：路徑長度為一。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 10. Book
2. S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, 1997. DOI
3. K. Cho et al., “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation,” EMNLP, 2014. arXiv
4. K. S. Tai, R. Socher and C. D. Manning, “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks,” ACL, 2015. arXiv
5. F. A. Gers, J. Schmidhuber and F. Cummins, “Learning to Forget: Continual Prediction with LSTM,” *Neural Computation*, 2000. DOI
6. Y. Bengio, P. Simard and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, 1994. DOI
7. R. Pascanu, T. Mikolov and Y. Bengio, “On the difficulty of training Recurrent Neural Networks,” ICML, 2013. arXiv
8. P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, 1990. DOI
9. J. L. Elman, “Finding Structure in Time,” *Cognitive Science*, 1990. DOI
10. M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, 1997. DOI
11. I. Sutskever, O. Vinyals and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” NeurIPS, 2014. arXiv

12. D. Bahdanau, K. Cho and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” ICLR, 2015. arXiv
13. O. Vinyals, A. Toshev, S. Bengio and D. Erhan, “Show and Tell: A Neural Image Caption Generator,” CVPR, 2015. arXiv
14. K. Xu et al., “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention,” ICML, 2015. arXiv
15. J. Donahue et al., “Long-term Recurrent Convolutional Networks for Visual Recognition and Description,” CVPR, 2015. arXiv
16. X. Shi et al., “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting,” NeurIPS, 2015. arXiv
17. R. Socher et al., “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank,” EMNLP, 2013. Paper
18. R. Socher, C. C.-Y. Lin, A. Y. Ng and C. D. Manning, “Parsing Natural Scenes and Natural Language with Recursive Neural Networks,” ICML, 2011. Paper
19. K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink and J. Schmidhuber, “LSTM: A Search Space Odyssey,” *IEEE Transactions on Neural Networks and Learning Systems*, 2017. arXiv
20. Z. C. Lipton, J. Berkowitz and C. Elkan, “A Critical Review of Recurrent Neural Networks for Sequence Learning,” arXiv:1506.00019, 2015. arXiv
21. A. Gu and T. Dao, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces,” COLM, 2024. arXiv
22. M. Beck et al., “xLSTM: Extended Long Short-Term Memory,” NeurIPS, 2024. Paper
23. L. Feng et al., “Were RNNs All We Needed?,” arXiv:2410.01201, 2024. arXiv
24. A. Orvieto et al., “Resurrecting Recurrent Neural Networks for Long Sequences,” ICML, 2023. arXiv
25. S. Bengio, O. Vinyals, N. Jaitly and N. Shazeer, “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks,” NeurIPS, 2015. arXiv
26. M. Tiezzi, M. Casoni, A. Betti, T. Guidi, M. Gori and S. Melacci, “On the Resurgence of Recurrent Models for Long Sequences: Survey and Research Opportunities in the Transformer Era,” arXiv:2402.08132, 2024. arXiv
27. M. Tiezzi, M. Casoni, A. Betti, T. Guidi, M. Gori and S. Melacci, “Back to recurrent processing at the crossroad of transformers and state-space models,” *Nature Machine Intelligence*, 2025. DOI
28. A. Vaswani et al., “Attention Is All You Need,” NeurIPS, 2017. arXiv
29. T. Dao and A. Gu, “Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality,” ICML, PMLR 235, 2024. Paper
30. J. Y.-C. Hu, X. Zhang, A. Elsheikh, W. Wu and H. Liu, “On Structured State-Space Duality,” ICML, PMLR 306, 2026. Paper

ML 05・注意力與視覺 Transformer

從 query、key、value 建立自注意力；加入多頭、位置編碼與 Transformer 區塊；用 ViT 把影像切成 patch token；再認識 DeiT、Swin、CNN 與 Transformer 混合模型、高效率注意力與狀態空間模型。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/vision-transformers/

你將學到

- 追蹤 query、key、value 與注意力權重。
- 建構多頭、遮罩與交叉注意力。
- 說明位置資訊與 Transformer 區塊。
- 比較區塊式、視窗式與混合視覺模型。
- 區分注意力的運算量、記憶體流量與上下文取捨。

先看全貌

CNN（ML 03）讓資訊透過固定的局部鄰域傳遞：一個像素要經過很多層，才能「聽到」遠處像素的消息。**注意力**（attention）則讓每個位置根據自己目前的內容，自行決定要讀取哪些其他位置，而且一步就到。要辨認一隻鳥，含有鳥頭的 patch 可以直接參考遠處含有翅膀的 patch。

> **用最簡單的話說**
>
> 想像一間教室，每個小朋友手上拿著一片拼圖。每個人都對自己那一片提出一個問題（「我這片有藍色的邊，誰有天空？」），其他人則各自舉起一張卡片，描述自己手上那一片。每個小朋友會最專心聽那些卡片最符合自己問題的同學，然後更新自己對這片拼圖的想法。接著大家用更新後的想法再來一輪。注意力就是這個「聽誰說話」的規則；Transformer 則是重複很多輪聆聽，中間再穿插一點自己思考的時間。

注意力的想法來自用循環網路做的機器翻譯：ML 04 說明了它如何移除編碼器–解碼器 RNN 的單一向量瓶頸。Vaswani 等人的 Transformer [1] 走得更遠，完全拿掉循環結構，編碼器與解碼器都只由注意力和逐 token 的小型網路組成。Dosovitskiy 等人的 Vision Transformer（ViT）[2] 則幾乎原封不動地把 Transformer 編碼器套用到影像 patch 序列上。Bishop 與 Bishop [10] 提供了現代的教科書式推導，而現代觀點一節討論的綜述 [3]、[11]、[12] 則整理了眾多視覺變體。讀完本篇之後，DLCV 02 會在實務上比較 CNN 與 Transformer 骨幹在分類上的表現，ML 06 則說明這些骨幹如何在沒有標註的情況下預訓練。

自注意力：query、key 與 value

白話版。 query 為可存取的 key 評分，正規化後的分數用來加權 value，讓各 token 選取可讀取的內容。

精確版。把 N 個 token 疊成矩陣 $X \in \mathbb{R}^{N\times D}$ 的各列，其中 D 是模型寬度。三個可學習的矩陣將它們投影：

$$Q = XW_Q, \qquad K = XW_K, \qquad V = XW_V,$$

其中 $W_Q, W_K \in \mathbb{R}^{D\times d_k}$、$W_V \in \mathbb{R}^{D\times d_v}$。**縮放點積注意力**（scaled dot-product attention）[1] 為

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}_{\mathrm{row}}\!\left(\frac{QK^\top}{\sqrt{d_k}}\right)V = AV.$$

$QK^\top$ 是一個 $N \times N$ 的**分數**矩陣：第 (i,j) 項是 query i 與 key j 的點積，衡量 token j 對 token i 有多相關。逐列的 softmax 把每一列轉成非負且總和為一的權重，得到**注意力矩陣** A。輸出 $Y = AV \in \mathbb{R}^{N\times d_v}$ 每個 token 一列：第 i 列是 $\sum_j A_{ij} v_j$，也就是 value 向量的加權平均。

為什麼要三個不同的投影？如果 query 和 key 是同一個向量，每個 token 都會和自己最吻合，而且相關性被迫是對稱的。分開的 W_Q 與 W_K 讓「我要找的東西」可以不同於「我能提供的東西」（眼睛的 patch 可以去找鼻子，而不必自己就是鼻子）；分開的 W_V 則讓「我能提供的東西」不同於「我實際傳出的內容」。

為什麼要除以 $\sqrt{d_k}$？若 query 與 key 的各分量互相獨立、平均為零、變異數為一，則點積 $q\cdot k = \sum_{l=1}^{d_k} q_l k_l$ 的平均為零、變異數為 d_k。不縮放的話，頭越寬分數就越大，softmax 會飽和（幾乎所有權重集中在一個 token 上），梯度也會變得極小。除以 $\sqrt{d_k}$ 讓變異數回到一。這是一種尺度慣例，並不是把每個 query 正規化成單位長度。

決定誰跟誰交談的是內容，而不是位置。卷積的權重綁定在相對位置上；注意力的權重則是在執行時由資料算出來的。由此衍生一個性質：自注意力是**排列等變**（permutation-equivariant）的。把輸入 token 打亂，輸出也會以相同方式被打亂，因為公式裡沒有任何東西涉及 token 的順序。下面的程式碼同時檢查形狀與這個對稱性。

```
import torch

torch.manual_seed(0)
X = torch.randn(5, 8)                    # N=5, D=8
Wq, Wk, Wv = [torch.randn(8, 4) for _ in range(3)]
def attention(x):
    q, k, v = x @ Wq, x @ Wk, x @ Wv
    a = (q @ k.T / q.shape[-1]**.5).softmax(dim=-1)
    return a @ v, a
y, a = attention(X)
perm = torch.tensor([3, 0, 4, 1, 2])
yp, _ = attention(X[perm])
assert torch.allclose(a.sum(-1), torch.ones(5), atol=1e-6)
assert torch.allclose(yp, y[perm], atol=1e-5)
print("output shape:", tuple(y.shape))
```

這個對稱性很適合處理集合，但影像不是一堆 patch 的集合：眼睛長在嘴巴下面的臉，就不是一張臉。下面介紹的位置資訊會打破這個對稱性。

多頭注意力

白話版。平行執行好幾個注意力運算，每個都有自己較小的投影，讓不同的頭可以追蹤不同的關係（鄰近紋理、同一個物體、對稱的部位）。最後把它們的輸出串接起來再混合。

精確版。有 h 個頭時，第 r 個頭有自己的 $W_Q^{(r)}, W_K^{(r)}, W_V^{(r)} \in \mathbb{R}^{D\times d_k}$，通常取 $d_k = d_v = D/h$：

$$\text{head}_r = \text{Attention}\big(XW_Q^{(r)}, XW_K^{(r)}, XW_V^{(r)}\big), \qquad \text{MHA}(X) = [\,\text{head}_1, \dots, \text{head}_h\,]\,W_O,$$

其中方括號把 h 個寬度為 D/h 的輸出串接回寬度 D，$W_O \in \mathbb{R}^{D\times D}$ 再把它們混合。由於每個頭寬度只有 D/h，多頭注意力的成本與一個全寬度的頭差不多，所有投影合計 $4D^2$ 個權重（另加偏差）。各個頭不保證會學到人類看得懂的角色；有些頭是多餘的，可以被剪除。

下面的實作把投影切成多個頭、逐頭計算分數、串接後再投影，結果與 `nn.MultiheadAttention` 完全一致。它也示範了**因果遮罩**：在 softmax 之前把禁止的分數設為 $-\infty$，這些位置的權重就會恰好為零。

```
import torch
from torch import nn

torch.manual_seed(0)
B, N, D, h = 2, 6, 16, 4
mha = nn.MultiheadAttention(D, num_heads=h, batch_first=True, bias=False)
X = torch.randn(B, N, D)

def multi_head(x):
    Wq, Wk, Wv = mha.in_proj_weight.chunk(3)          # each D x D (rows = outputs)
    q, k, v = x @ Wq.T, x @ Wk.T, x @ Wv.T              # B x N x D
    split = lambda t: t.view(B, N, h, D // h).transpose(1, 2)   # B x h x N x d_k
    q, k, v = split(q), split(k), split(v)
    scores = q @ k.transpose(-2, -1) / (D // h) ** 0.5  # B x h x N x N
    A = scores.softmax(dim=-1)
    heads = A @ v                                       # B x h x N x d_k
    concat = heads.transpose(1, 2).reshape(B, N, D)     # put the heads side by side
    return concat @ mha.out_proj.weight.T, A

ours, A = multi_head(X)
ref, _ = mha(X, X, X)
print("matches nn.MultiheadAttention:", torch.allclose(ours, ref, atol=1e-5))
print("attention tensor:", tuple(A.shape), "| parameters:",
      sum(p.numel() for p in mha.parameters()), "= 4 * D^2 =", 4 * D * D)

# Causal mask: query i may only read keys j <= i
causal = torch.triu(torch.ones(N, N, dtype=torch.bool), diagonal=1)   # True = forbidden
out_c, w_c = mha(X, X, X, attn_mask=causal)
print("no weight on the future:", torch.all(w_c[:, causal] == 0).item())
```

輸出為 `True`、形狀 `(2, 4, 6, 6)` 的注意力張量（批次、頭、query、key）、`1024 = 4 * D^2 = 1024`，以及因果檢查的 `True`。正式程式可使用 `torch.nn.functional.scaled_dot_product_attention`，它會在支援的條件下使用融合核心 [7]。**不能直接照抄布林遮罩：** `nn.MultiheadAttention` 的 `True` 表示禁止讀取，`scaled_dot_product_attention` 的 `True` 卻表示允許讀取。評估時也要明確傳入 `dropout_p=0.0`，因為函式不會查看模組的訓練旗標。替換教學實作時，應分別確認遮罩語意與 dropout 設定。

連接 token 的三種方式

白話版。一個 query 可以讀取哪些 token，是模型設計的一部分，以分數矩陣上的遮罩來表達。

機制	query 可讀取	典型用途
雙向自注意力	同一序列中的所有 token	ViT 及其他影像編碼器
因果（遮罩）自注意力	只有自己與更早的位置	下一個 token 的生成、自迴歸影像模型
交叉注意力	來自另一個序列的 token	解碼器讀取編碼器；影像 token 讀取文字

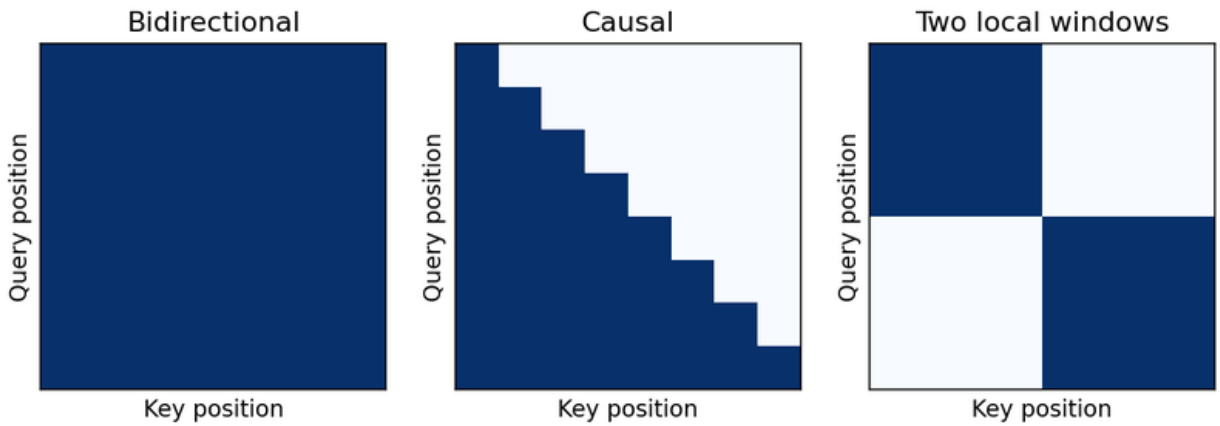


圖 *1* — 注意力遮罩：藍色表示允許，白色表示禁止。因果生成還會把輸入平移一個位置，讓目標無法被直接複製。

在**交叉注意力**中，query 來自一個序列 X，key 與 value 則來自另一個序列 Z：$Q = XW_Q$、$K = ZW_K$、$V = ZW_V$。分數矩陣的形狀是 $N_X \times N_Z$，輸出每個 query 一列。原始 Transformer 的解碼器就是這樣讀取編碼器；影像描述模型這樣讀取影像特徵；文字生成影像的擴散模型也這樣讀取文字提示（ML 08），而不需要在影像 patch 上做自迴歸。

注意力權重是計算的一部分，並不自動等於一種解釋。很大的權重可能乘上一個毫無資訊的 value，而殘差與 MLP 路徑也會影響輸出。

位置資訊

白話版。純內容自注意力沒有內建位置標記；位置嵌入、相對位置資訊或與位置相關的遮罩提供結構。

正弦位置編碼。原始 Transformer [1] 在位置 pos 的 token 上加一個固定向量 $p_{pos} \in \mathbb{R}^D$：

$$p_{pos,2i} = \sin\!\left(\frac{pos}{10000^{2i/D}}\right), \qquad p_{pos,2i+1} = \cos\!\left(\frac{pos}{10000^{2i/D}}\right), \qquad i = 0, \dots, \tfrac{D}{2} - 1.$$

每一對維度 $(2i, 2i+1)$ 都是一個以角頻率 $\omega_i = 10000^{-2i/D}$ 繞圓旋轉的點，頻率從很快（每個位置一弧度）到非常慢都有。由三角函數的和角公式可得兩個性質。第一，對任一固定位移 k，都有 $p_{pos+k} = p_{pos} R_k$，其中 R_k 是一個與 pos 無關的區塊對角旋轉矩陣；因此一個線性層就能學會「往後看 k 個位置」。第二，$p_m \cdot p_n = \sum_i \cos\left(\omega_i (m-n)\right)$ 只與位移 $m-n$ 有關（圖 2 中）。

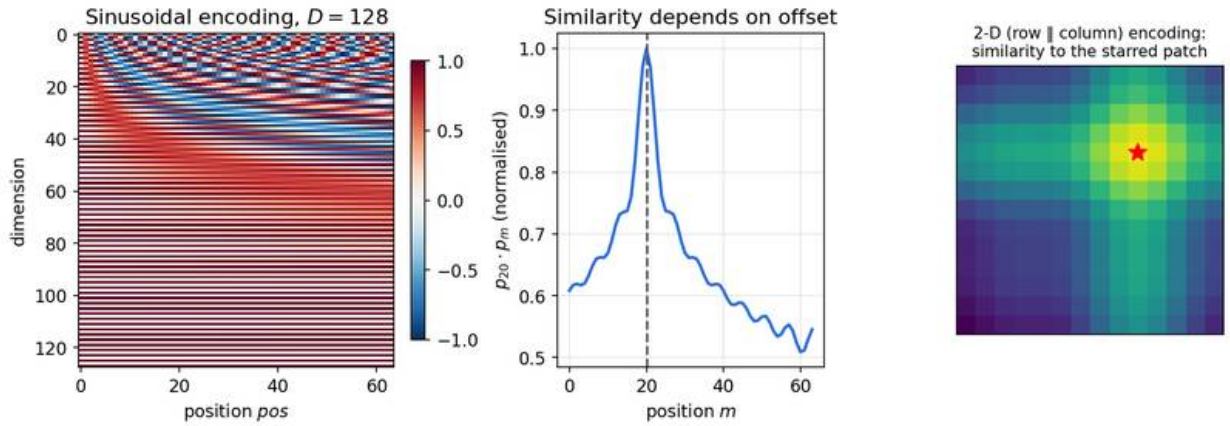


圖 2 — 正弦位置編碼。左：每一對維度以各自的頻率振盪。中：編碼之間的相似度只取決於位移。右：把列與行的編碼串接起來，就得到 *14 × 14 patch* 網格的二維編碼。

```
import numpy as np

def sinusoidal(n_pos, D):
    pos = np.arange(n_pos)[:, None]
    omega = 1.0 / 10000 ** (np.arange(0, D, 2) / D)    # one frequency per sin/cos pair
    pe = np.zeros((n_pos, D))
    pe[:, 0::2] = np.sin(pos * omega)
    pe[:, 1::2] = np.cos(pos * omega)
    return pe, omega

pe, omega = sinusoidal(100, 16)
k = 7                                                  # a fixed offset
# block-diagonal rotation that maps p_pos to p_{pos+k}, the same for every pos
R = np.zeros((16, 16))
for i, w in enumerate(omega):
    c, s = np.cos(w * k), np.sin(w * k)
    R[2 * i:2 * i + 2, 2 * i:2 * i + 2] = [[c, -s], [s, c]]
print("p[pos+k] == p[pos] @ R for all pos:", np.allclose(pe[:-k] @ R, pe[k:]))
```

```
print("dot products depend only on offset:",
      np.isclose(pe[10] @ pe[17], pe[50] @ pe[57]))
```

兩項檢查都印出 `True`。

其他做法。

- **可學習的絕對位置嵌入**：每個位置一個可訓練的向量，ViT [2] 就是如此。ViT 依光柵（raster）順序編號的一維可學習嵌入，效果與明確的二維嵌入差不多；訓練後，同一列或同一行的 patch 嵌入會變得相似，代表模型自己學會了網格結構。換到新的解析度時，先用二維內插調整嵌入網格的大小，再進行微調。
- **相對位置表示**（Shaw 等人 [17]）在 token i 與 j 的分數上，加一個取決於位移 $j-i$ 的可學習項。Swin [5] 則在每個視窗的分數上加一張可學習的**相對位置偏差**（relative position bias）表。
- **旋轉位置嵌入**（rotary position embedding, RoPE；Su 等人 [16]）把每個 query 與 key 依其位置旋轉一個成比例的角度，使用的正是上面程式碼中的那種旋轉。於是分數 $q_m^\top k_n$ 只透過 $m-n$ 依賴位置。RoPE 已是大型語言模型的標準配備，在視覺中也越來越常見，做法是對影像的兩個軸分別旋轉。
- **不加明確編碼**：網路內部的卷積（如 CvT [19]），或邊界的零填補，本身就可能洩漏足夠的位置資訊。

Transformer 區塊

白話版。一個區塊交替做兩件事：注意力讓 token 互相交換資訊；接著一個小型 MLP 分別處理每個 token。殘差連接與正規化讓整個堆疊保持穩定。

精確版。 ViT 與大多數現代模型使用的**前置正規化**（pre-norm）區塊為

$$U = X + \mathrm{MHA}(\mathrm{LN}(X)), \qquad Y = U + \mathrm{MLP}(\mathrm{LN}(U)), \qquad \mathrm{MLP}(u) = W_2\,\mathrm{GELU}(W_1 u + b_1) + b_2.$$

LN 是**層正規化**（layer normalization）[14]：它把每個 token 的 D 個特徵標準化為平均零、變異數一，再套用可學習的縮放與平移；與 BatchNorm 不同，它不依賴同一批次中的其他樣本。MLP 把每個 token 從 D 擴展到通常 $4D$ 個隱藏單元，套用 GELU 非線性 [26]（$\mathrm{GELU}(z) = z\,\Phi(z)$，$\Phi$ 為標準常態分布的累積分布函數），再投影回去。殘差相加代表每個子層只需學習一個修正量，與 ResNet 完全相同。

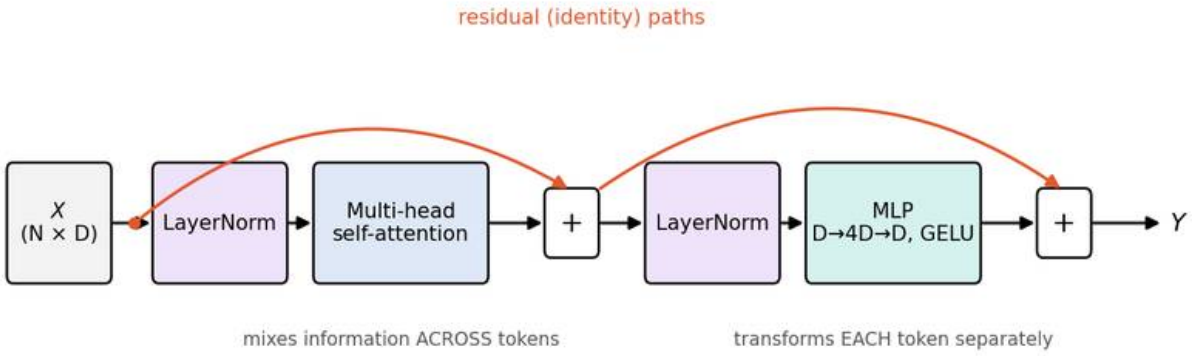


圖 3 — 前置正規化的 *Transformer* 區塊。注意力在 *token* 之間混合資訊；*MLP* 分別轉換每個 *token*；殘差路徑（橘色）把輸入繞過每個分支。

原始 Transformer 把 LayerNorm 放在每次殘差相加**之後**（後置正規化，post-norm）。Xiong 等人 [15] 證明，使用後置正規化時，初始化階段靠近輸出的梯度很大，這正是它需要學習率暖身（warm-up）的原因；前置正規化則讓梯度保持穩定，可以不用暖身就訓練。所以只說「一個 Transformer」並不完整，還得說明正規化放在哪裡。每個區塊的權重數：注意力 $4D^2$、MLP $8D^2$，合計約 $12D^2$，其中三分之二在 MLP。

Vision Transformer（ViT）

白話版。把影像切成方形 patch，把每個 patch 變成向量，加上位置資訊，然後送進標準的 Transformer 編碼器。答案從一個額外的特殊 token 讀出。

切成 patch。對一張 $H \times W$ 像素、C 個通道的影像，若 patch 大小 P 能整除 H 與 W，就有 $N = HW/P^2$ 個 patch。把每個 patch 攤平成 $x_i^{\text{patch}} \in \mathbb{R}^{P^2C}$，再做線性投影：

$$z_0 = [\, x_{\text{cls}};\ x_1^{\text{patch}}E;\ \ldots;\ x_N^{\text{patch}}E \,] + E_{\text{pos}},$$

其中 $E \in \mathbb{R}^{P^2C \times D}$ 是可學習的 **patch 嵌入**，$x_{\text{cls}} \in \mathbb{R}^D$ 是加在序列最前面的可學習**類別 token**（class token），$E_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$ 存放可學習的位置嵌入。經過 L 個區塊後，對類別 token 的輸出套用一個 LayerNorm 與線性分類頭，就得到 logits。把所有 patch 的輸出取平均（mean pooling）也是同樣常見的替代做法。

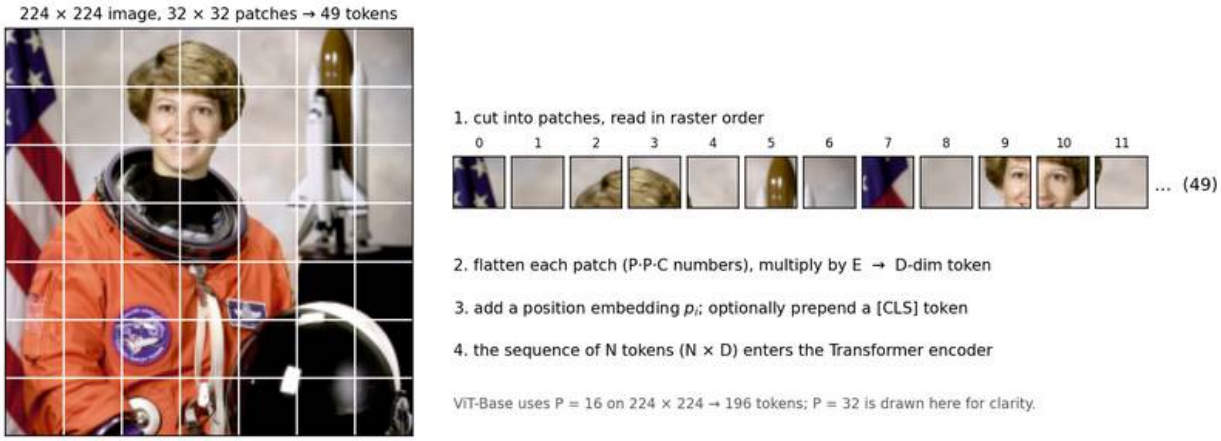


圖 *4* — 切成 *patch*。為了清楚起見這裡畫的是 *32* 像素的格子；*ViT-Base* 使用 *16 × 16* 的 *patch*，把 *224 × 224* 的影像變成 *196* 個 *token*。

對不重疊的 $P \times P$ patch 套用線性映射，恰好等於核大小與步幅都是 P 的卷積，實作上也正是這樣寫：

```
import torch
from torch import nn

torch.manual_seed(0)
B, C, H, W, P, D = 2, 3, 32, 32, 8, 48
x = torch.randn(B, C, H, W)

# 1) explicit: cut into P x P patches, flatten each, multiply by E
patches = x.unfold(2, P, P).unfold(3, P, P)                  # B, C, H/P, W/P, P, P
patches = patches.permute(0, 2, 3, 1, 4, 5).reshape(B, -1, C * P * P)   # B, N, P*P*C
E = torch.randn(C * P * P, D) * 0.02
tokens_explicit = patches @ E                                 # B, N, D

# 2) the same linear map written as a convolution with kernel = stride = P
conv = nn.Conv2d(C, D, kernel_size=P, stride=P, bias=False)
conv.weight.data = E.T.reshape(D, C, P, P)
tokens_conv = conv(x).flatten(2).transpose(1, 2)              # B, N, D

print("tokens:", tuple(tokens_conv.shape), "| identical:",
      torch.allclose(tokens_explicit, tokens_conv, atol=1e-5))
```

輸出為 16 個寬度 48 的 token，以及 `True`。

模型大小。標準設定有 ViT-Base（12 個區塊、$D = 768$、12 個頭、約 8,600 萬個參數）、ViT-Large（24 個區塊、$D = 1024$、16 個頭）與 ViT-Huge（32 個區塊、$D = 1280$、16 個頭）[2]。「ViT-B/16」代表使用 16×16 patch 的 Base 模型。**patch 大小是最主要的運算量旋鈕**：P 減半，N 變成四倍，注意力分數的數量變成十六倍。小 patch 保留細節；大 patch 便宜，但比 patch 還細的資訊都會遺失，而且後面任何一層都救不回來。

資料的教訓。 Dosovitskiy 等人 [2] 發現，只在 ImageNet-1k（約 130 萬張影像）上訓練的 ViT，

表現略遜於規模相當的 ResNet；但若先在 ImageNet-21k（1,400 萬張）或 JFT-300M（3 億張）上預訓練再微調，它就能追平甚至超越 ResNet，而且預訓練成本更低。他們的解釋是**歸納偏置**（inductive bias）：CNN 在每一層都假設局部性與平移等變性，而 ViT 只在 patch 嵌入內是局部的，空間關係必須從資料中**學出來**。Raghu 等人 [18] 比較了 ViT 與 ResNet 的內部表徵，發現 ViT 各層的表徵更加一致、較低層同時兼顧局部與全域的注意力，而且要在早期層學到局部注意力，需要大規模的預訓練。

程式碼：在手寫數字上訓練一個小型 ViT

這個完整範例在與 ML 03 CNN 相同的 8×8 手寫數字資料上訓練一個小型 ViT，使用 2×2 的 patch（16 個 token）、一個類別 token、可學習的位置嵌入，以及四個前置正規化區塊。在 CPU 上約半分鐘即可跑完。

```
import copy
import numpy as np
import torch
from torch import nn
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

torch.manual_seed(0)
torch.set_num_threads(1)

class TinyViT(nn.Module):
    def __init__(self, img=8, patch=2, chans=1, D=64, depth=4, heads=4, classes=10):
        super().__init__()
        N = (img // patch) ** 2
        self.patch = nn.Conv2d(chans, D, kernel_size=patch, stride=patch)   # patchify + E
        self.cls = nn.Parameter(torch.zeros(1, 1, D))                       # [CLS] token
        self.pos = nn.Parameter(torch.randn(1, N + 1, D) * .02)             # learned
        ↪  positions
        layer = nn.TransformerEncoderLayer(D, heads, dim_feedforward=4 * D, dropout=0.1,
                                           activation="gelu", batch_first=True,
                                           norm_first=True)                 # pre-norm block
        self.blocks = nn.TransformerEncoder(layer, depth, enable_nested_tensor=False)
        self.norm, self.head = nn.LayerNorm(D), nn.Linear(D, classes)

    def forward(self, x):
        t = self.patch(x).flatten(2).transpose(1, 2)                         # B, N, D
        t = torch.cat([self.cls.expand(len(t), -1, -1), t], dim=1) + self.pos
        return self.head(self.norm(self.blocks(t))[:, 0])                   # read [CLS]

d = load_digits()
X = torch.tensor(d.images[:, None] / 16.0, dtype=torch.float32)
y = torch.tensor(d.target)
train, test = train_test_split(np.arange(len(y)), test_size=.2, stratify=d.target,
                               random_state=0)
train, val = train_test_split(train, test_size=.2, stratify=d.target[train], random_state=1)
model = TinyViT()
opt = torch.optim.AdamW(model.parameters(), lr=1e-3, weight_decay=0.05)
best = (float("inf"), None)
```

```
for epoch in range(30):
    model.train()
    for idx in torch.tensor(train)[torch.randperm(len(train))].split(64):
        loss = nn.functional.cross_entropy(model(X[idx]), y[idx])
        opt.zero_grad(); loss.backward(); opt.step()
    model.eval()
    with torch.no_grad():
        v = nn.functional.cross_entropy(model(X[val]), y[val]).item()
    if v < best[0]:
        best = (v, copy.deepcopy(model.state_dict()))
model.load_state_dict(best[1]); model.eval()
with torch.no_grad():
    acc = (model(X[test]).argmax(1) == y[test]).float().mean().item()
print("parameters:", sum(p.numel() for p in model.parameters()))
print("best validation loss:", round(best[0], 4), "| held-out accuracy:", round(acc, 4))
```

在我們的執行中，它約有 202,000 個參數，保留測試集準確率約 95%；而 ML 03 的 CNN 在相同的資料切分下，只用不到 10,000 個參數就達到約 96%。在小型資料集上，CNN 內建的局部性大約值二十倍的參數量。這是「資料的教訓」的縮影，而不是對大規模效能的定論。

僅線上版：*ViT：影像區塊與預訓練規模的角色* — *https://blog.shyandram.dev/zh-tw/learn/ml/vision-transformers/papers/vit 影像區塊與預訓練規模的角色/*

訓練配方：DeiT 及其後續

白話版。早期 ViT 與 CNN 在 ImageNet 規模資料上的差距，很大一部分其實來自訓練方式，而不是架構。

DeiT（Touvron 等人 [4]）**只用 ImageNet-1k** 就把 ViT 訓練到具競爭力，靠的是強力的資料擴增（RandAugment、Mixup、CutMix、隨機抹除）、大量正則化（stochastic depth、label smoothing）以及較長的訓練排程。它還引入了**蒸餾 token**（distillation token）：一個類似類別 token 的額外可學習 token，其輸出被訓練去符合一個預訓練 CNN 教師模型的預測。CNN 的歸納偏置因此透過它的預測轉移過來。後來的 DeiT III [27] 證明，一個更簡單但仔細調校的監督式配方（輕量的「3-Augment」擴增、簡單的裁切，以及二元交叉熵損失函數）能讓純 ViT 更強。Steiner 等人 [25] 對資料、擴增與正則化做了大規模研究（「AugReg」），發現擴增、正則化與運算量的適當組合，可以追上用大約十倍資料訓練的模型。

一般性的教訓與 ML 03 討論 ConvNeXt 時相同：只有在資料、擴增、解析度、訓練排程與運算量都對齊的情況下，比較架構才有意義。

Swin 與階層式 Transformer

白話版。對數千個 token 做全域注意力很昂貴，而且純 ViT 從頭到尾維持同一個粗解析度，不利於偵測與分割。Swin 把注意力限制在局部視窗內，在層與層之間平移視窗讓資訊仍能擴散，並合併 patch 來建立類似 CNN 的金字塔。

視窗。 Swin [5] 把 token 網格切成互不重疊、每個 $M \times M$ 個 token 的視窗（通常 $M = 7$），只在每個視窗內部計算自注意力。對 $h \times w$ 的 token 網格、C 個通道，論文給出的成本為

$$\Omega(\text{global MSA}) = 4hwC^2 + 2(hw)^2C, \qquad \Omega(\text{window MSA}) = 4hwC^2 + 2M^2hwC,$$

其中 $4hwC^2$ 是四個投影的成本，第二項則是計算分數與加權的乘法。全域成本對 token 數 hw 是二次方；固定 M 時，視窗成本是線性的。

平移視窗。若視窗固定不動，不同視窗的 token 永遠不會互動。Swin 讓一般視窗與在兩個方向各平移 $\lfloor M/2 \rfloor$ 個 token 的視窗交替出現。在第 ℓ 層位於視窗邊界附近的 token，到第 $\ell + 1$ 層會與相鄰視窗的 token 同在一個視窗，資訊因此得以跨越邊界（圖 5）。高效率的實作會把特徵圖做循環平移，並用遮罩避免從對側邊緣繞過來的 token 互相注意。

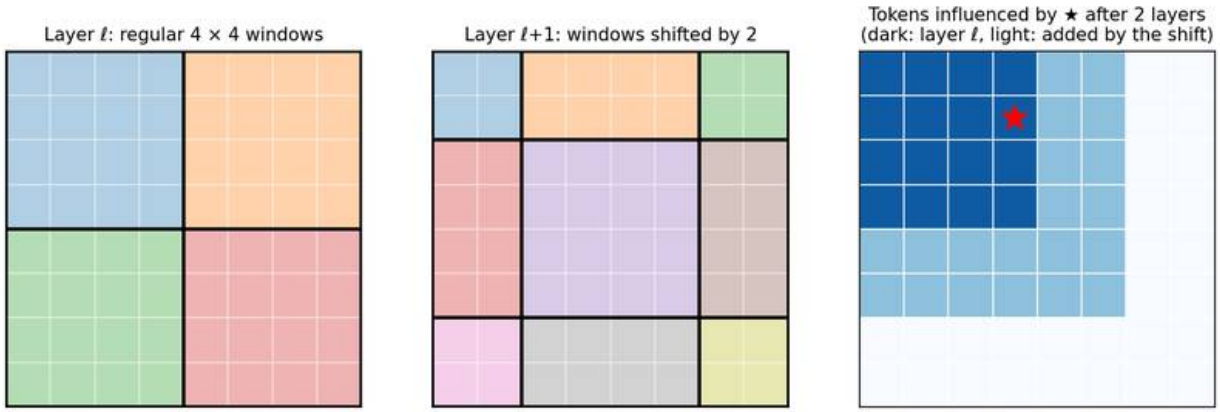


圖 *5 — Swin 在 8 × 8 token* 網格上的一般視窗與平移視窗。兩層之後，星號 *token* 的資訊會到達深色視窗（第 ℓ 層），再加上平移視窗帶來的淺色帶狀區域（第 ℓ*+1* 層）。

階層結構。在各階段之間，**patch 合併**（patch merging）把每 2×2 組相鄰 token 串接後做線性投影，使解析度減半、通道數加倍，就像步幅卷積一樣。四個階段分別產生輸入解析度 1/4、1/8、1/16、1/32 的特徵圖，與 ResNet 的金字塔相同，因此 Swin 可以在偵測與分割框架中直接取代 CNN 骨幹。每個視窗的分數也會加上可學習的**相對位置偏差**。Swin V2 [32] 用殘差後置正規化、縮放餘弦注意力（scaled cosine attention），以及連續、對數間隔的位置偏差來擴大規模，後者在不同視窗大小與解析度之間的遷移效果更好。

CNN–Transformer 混合模型

白話版。卷積擅長在資料少時處理局部細節；注意力擅長長距離互動。混合模型讓兩者各自在適合的地方發揮。

- **卷積起始層（stem）**。原始 ViT 論文就已測試過一種「混合」版本，其 patch 取自 ResNet 的特徵圖而不是原始像素 [2]；在較小的運算預算下有幫助。
- **在注意力裡放卷積**。 CvT [19] 在每個階段使用卷積式 token 嵌入，並用深度可分離卷積產生 query、key、value 的投影，藉此內建局部性，也因此可以省去明確的位置嵌入。
- **依階段堆疊**。 CoAtNet [20] 在前期放深度卷積階段（此時解析度高、局部模式為主），後期放相對注意力階段；它在各種資料規模下都報告了強勁的結果，因為卷積階段在資料有限時提供

了歸納偏置。

- **行動裝置混合模型**。 MobileViT [28] 在 MobileNet 風格的網路中插入小型 Transformer 區塊，以行動裝置等級的成本加入全域脈絡。
- **反方向**。 ConvNeXt（見 ML 03）把 Transformer 時代的設計選擇引進純 CNN，並追平了 Swin。

成本與高效率注意力

白話版。全域注意力比較 token 配對，配對運算量隨 token 數呈平方成長；效率方法減少記憶體流量、互動或 token 數。

對 N 個寬度為 D 的 token，每層分數矩陣約需 N^2D 次乘加，投影與 MLP 約需 $12ND^2$ 次。單純的實作會為每個頭儲存一個 $N \times N$ 矩陣。固定 patch 大小時，影像高與寬各加倍，N 變成四倍，成對分數變成十六倍。

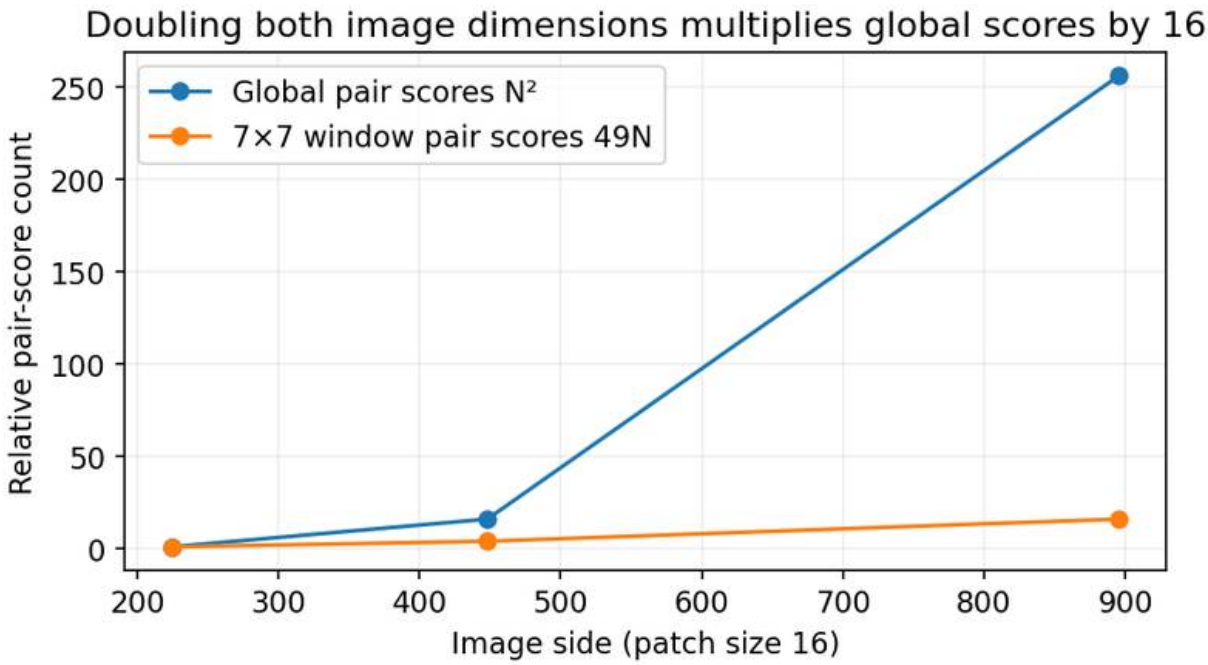


圖 *6* — 成對分數的成長趨勢，並非實測的執行時間。投影、*MLP*、記憶體存取與核心實作品質也會影響延遲。

主要的方法家族：

- **精確但更快**。 FlashAttention [6] 把計算切成區塊（tiling），讓 query、key、value 的區塊留在快速的晶片內記憶體中，並且從不把完整的 $N \times N$ 矩陣寫到較慢的 GPU 記憶體，藉此計算**精確**的注意力；FlashAttention-2 [29] 進一步改善了工作切分方式。算術量仍是二次方，但記憶體用量變成線性，實際執行時間也大幅下降。PyTorch 的 `scaled_dot_product_attention` 在可行時會自動選用這類核心 [7]。
- **限制連接方式**。視窗（Swin）或其他稀疏模式減少了成對的數量，代價是需要好幾層才能達成全域溝通。

- **減少 token**。 Token Merging（ToMe）[21] 在預訓練好的 ViT 內，以快速的配對步驟逐步合併相似的 token，不需重新訓練就能提高吞吐量。較大的 patch、token 剪枝與池化階段則以更粗略的方式達到相同效果。
- **更小的模型**。 Papa 等人的綜述 [13] 把高效率 ViT 分為精簡架構、剪枝、知識蒸餾與量化四類，並提出「Efficient Error Rate」指標，用來在參數量、位元數、FLOPs 與模型大小之間比較模型。

狀態空間模型與混合模型

白話版。不必比較每一對 token，而是像循環網路（ML 04）一樣，讓一個精簡的狀態沿著序列傳遞。成本隨 token 數線性成長。

對 token 序列，一個簡化的狀態空間遞迴為

$$h_t = \bar{A}_t\, h_{t-1} + \bar{B}_t\, x_t, \qquad y_t = C_t\, h_t,$$

其中 h_t 是固定大小的隱藏狀態，$\bar{A}_t$、$\bar{B}_t$、C_t 是（離散化後的）狀態空間矩陣。在 Mamba 這類**選擇性**（selective）模型中，寫入／讀取係數與步長取決於輸入 x_t；底層連續轉移可以保持固定，離散化後的係數卻會改變。輸入選擇與仿射狀態更新使平行掃描（parallel scan）成為可能 [33]。影像沒有天然的一維順序，因此視覺版本必須選擇**掃描順序**：Vision Mamba（Vim）[30] 加上位置嵌入，對 patch 序列做正向與反向掃描；VMamba [31] 則沿著二維網格的四個方向掃描。Visual Mamba 綜述 [8] 回顧了 200 多篇論文，並指出掃描方式是視覺應用中最關鍵的設計決策。

MambaOut [9]（CVPR 2025）是一個有用的反例。作者主張 Mamba 適合長序列與因果（自迴歸）結構的任務，並在保留閘控卷積區塊的同時拿掉狀態空間元件，建立了 MambaOut 模型。他們的模型在 ImageNet 分類上勝過視覺 Mamba 模型，顯示 Mamba 在這裡並非必要；而對於涉及長 token 序列的偵測與分割，他們認為狀態空間模型仍值得探索。更廣泛的重點在於方法論：要把某個結果歸功於某種機制（注意力、掃描、卷積）之前，必須先把它與閘控、正規化、訓練配方和參數預算分開來看。

影像掃描是一種歸納偏置

Mamba [33] 提供高效率的序列運算，但不會替我們決定影像的幾何結構。Vim [30] 使用 patch 序列、位置嵌入與正向／反向處理；VMamba [31] 則把四條路徑的選擇性掃描放進階層式視覺骨幹中。單靠模型名稱，無法判斷感受野的影響強度、解析度行為或執行時間。狀態維度、掃描路徑、局部運算與降取樣階段都會影響結果。

在固定掃描順序下，影像中相鄰的兩個像素，可能在序列中相隔許多步。反過來，一列最後的像素與下一列第一個像素，雖然在空間上相距甚遠，卻在序列中連續。局部卷積可以先提供鄰近證據，再由掃描把上下文傳得更遠。多條路徑提供額外的傳遞方式，但不會自動恢復所有幾何對稱性。位置嵌入也能編碼有用的位置，卻讓移轉到不同網格成為另一個需要處理的設計問題。

可以用數學明確呈現這個選擇：

$$Y^{(r)} = P_r^{-1}\, \mathcal{S}_r(P_r X), \qquad Y = \frac{1}{m}\sum_{r=1}^{m} Y^{(r)}.$$

其中 X 包含展平後的影像 token，r 索引 m 條路徑中的一條，P_r 把 token 排列成該路徑的順序，$\mathcal{S}_r$ 執行該路徑的序列循環，P_r^{-1} 還原影像座標，Y 對產生的特徵取平均。這是概念性的路徑融合模型。可學習的視覺區塊可以使用不同投影或融合規則；這裡選擇平均，是為了讓實驗容易解讀。

CPU 實驗：量測方向偏差

下一個範例使用單一純量狀態，每條路徑的衰減係數都相同。先比較正向／反向的逐列掃描，以及加入逐欄掃描的四條路徑；接著把輸入旋轉 90 度、處理後再把輸出轉回來。全部使用合成的 6×6 脈衝，不下載權重或影像。這個實驗單獨檢查排序，並不重現訓練好的 Vim 或 VMamba。

```
import numpy as np

def scan_image(image, four=False):
    ids = np.arange(image.size).reshape(image.shape)
    row, col = ids.ravel(), ids.T.ravel()
    orders = [row, row[::-1]]
    if four:
        orders += [col, col[::-1]]
    outputs = []
    for order in orders:
        h = 0.0
        out = np.zeros(image.size)
        for i in order:
            h = 0.8 * h + 0.2 * image.ravel()[i]
            out[i] = h
        outputs.append(out.reshape(image.shape))
    return np.mean(outputs, axis=0)

image = np.zeros((6, 6))
image[1, 4] = 1.0
errors = []
for four in [False, True]:
    original = scan_image(image, four=four)
    rotated_back = np.rot90(scan_image(np.rot90(image), four=four), -1)
    error = np.max(np.abs(original - rotated_back))
    errors.append(error)
    print('four routes:', four, 'rotation discrepancy:', round(error, 6))
assert 0.0 < errors[1] < errors[0]
```

兩條路徑的最大差異為 `0.063622`，四條路徑則為 `0.026738`。在這個範例中，增加路徑減少方向偏差，但剩餘差異並非零。旋轉逐列路徑時，其中一個遍歷軸會反向；變換後的路徑未必屬於原本四條路徑的集合。因此，即使係數共用且以平均融合，也不保證旋轉等變性。可學習的方向投影、位置資訊與局部運算又引入更多選擇。應直接測試需要的性質，而不是從「二維掃描」推定它存在。

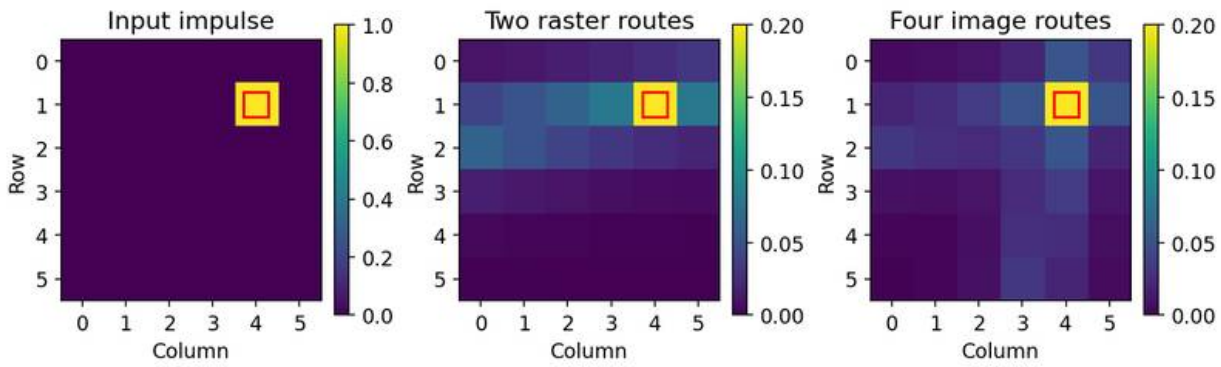


圖 7 — 紅色外框標示輸入脈衝。兩條逐列路徑不均勻地傳遞上下文；加入逐欄路徑後，影響圖會改變。這些合成響應使用範例中的相同循環，並非學得的注意力圖或訓練好的 *VMamba* 活化值。

資訊能沿著完整掃描傳遞，因此循環可以具有全域感受野。這表示依賴關係可能存在，並不代表它在所有距離都保持強大影響。純量衰減為 0.8 時，攜帶的貢獻每經過一步就再乘上 0.8。ML 04 推導的選擇性轉移可以改變這種保留方式，但訓練仍須學會哪些資訊值得保存。即使可以讀取遠處上下文，狀態壓縮或降取樣仍可能遺失細緻邊界與小物件。

Mamba-2 改變了什麼，又該如何評估

Mamba-2 [34] 以結構化狀態空間對偶性與高效率的分塊計算，重新安排選擇性序列混合器。它改變轉移結構與區塊參數投影，並沒有讓任意影像 patch 序列都等同於 Transformer。具有結構的因果係數矩陣可以用類似注意力的方式解讀，但標準 softmax 注意力仍保有不同的表達與計算性質。應用到影像時，仍須指定方向與融合方式。因此，對偶性回答的是計算問題，並未消除幾何問題。

公平的視覺比較應區分三種實驗：在共用區塊中只替換混合器；在對齊的訓練預算下比較完整骨幹；以及在目標硬體上量測部署模型。固定資料、資料增強、token 解析度與下游預測頭。除了參數量，也記錄狀態大小，並在不同影像解析度與批次大小下量測延遲及記憶體峰值。token 數的線性項可以同時伴隨昂貴的投影、路徑排列與小型運算核心的額外成本。上面的 Python 迴圈無法支持 GPU 吞吐量的宣稱。

對密集任務，檢查邊界附近、小型結構與不同長寬比下的錯誤；對影片，區分離線雙向編碼器與因果即時處理。除了分布內分類，也應納入旋轉／裁切敏感度與更長網格的評估。MambaOut [9] 提供重要的移除混合器消融實驗。結果支持依任務比較，並不是證明所有影像問題都不需要掃描的定理。Xu 等人 [8] 依骨幹、掃描機制，以及影像／影片／點雲／多模態應用整理視覺 Mamba。在已核實的紀錄中，這篇綜述仍是 2024 年預印本。其分類架構有助於選擇控制變因的實驗；提出的變體數量並不能確立普遍的贏家。

現代觀點

綜述文獻。 Khan 等人 [3]（*ACM Computing Surveys*，2022）從基礎（自注意力、大規模預訓練、雙向編碼）談起，接著綜覽 Transformer 在辨識、生成模型、視覺問答等多模態任務、影

片、低階視覺與三維分析上的應用，最後列出開放問題。Han 等人 [11]（*IEEE TPAMI*，2023）依任務整理視覺 Transformer（骨幹、偵測與分割等中高階視覺、低階視覺、影片），也涵蓋高效率 Transformer 方法。Liu 等人 [12]（*IEEE TNNLS*，2024）回顧了一百多個用於影像、點雲與視覺–語言資料之分類、偵測與分割的視覺 Transformer，並提出三個未來方向。在效率方面，Papa 等人 [13]（*IEEE TPAMI*，2024）最有系統。狀態空間視覺模型請參考 Xu 等人 [8]。教科書式的推導則見 Bishop 與 Bishop [10]。

2023–2026 年的最新發展。純 ViT 已成為預設的**基礎模型**（foundation model）骨幹。ViT-22B [24]（ICML 2023）把 ViT 擴大到 220 億個參數，並需要一些架構修改（例如對 query 與 key 做正規化）才能在這個規模下穩定訓練。DINOv2 [23]（TMLR 2024）以自監督方式在大型精選影像集合上訓練 ViT，得到的通用特徵在凍結狀態下就能用於分類、分割與深度估計（ML 06）。Darcet 等人 [22]（ICLR 2024）發現大型 ViT 會把資訊量低的背景 patch 挪作內部暫存空間，在注意力圖中產生高範數的假影 token；加入少數額外的**暫存器 token**（register tokens）就能消除這些假影，並改善密集預測。視覺–語言基礎模型（ML 10）與擴散 Transformer（ML 08）都建立在同樣的區塊上。與此同時，階層式與混合設計在高解析度的密集任務上仍很常見；在預算對齊時，現代化的 CNN 依然具有競爭力（ML 03）；而狀態空間與線性注意力模型仍持續被探索，用於影片這類非常長的 token 序列。

開放問題。高解析度與長影片下的二次方成本；純 ViT 對資料與運算量的龐大需求；能在不同解析度與長寬比之間泛化的位置編碼；如何解讀注意力而不過度詮釋；以及在注意力、卷積與狀態空間機制之間，做出運算量對齊的公平比較。

重點整理

- query 與 key 決定權重，value 提供注意力混合的內容。
- 分數縮放在變異數假設下控制量級，不保證任意學得輸入都有單位變異數。
- 遮罩限制資訊存取，位置訊號則區分純內容注意力無法辨別的排列。
- Transformer 區塊交替進行 token 混合與逐 token 變換，並搭配殘差路徑及正規化。
- ViT、位移視窗與卷積混合模型，對局部性、解析度及訓練需求採取不同選擇。
- FlashAttention 降低精確注意力的記憶體流量；視窗、token 縮減與狀態空間掃描則改變運算或上下文存取。

練習

1. **Token 預算**。對一張 512×512 的影像，分別計算 patch 大小為 16 與 8 時的 token 數，以及每個頭的注意力分數個數。patch 大小減半時，分數個數增加幾倍？

   提示

   $N = (512/16)^2 = 1024$ 與 $(512/8)^2 = 4096$ 個 token；$N^2 = 1{,}048{,}576$ 與 $16{,}777{,}216$ 個分數。patch 大小減半，N 乘以 4，分數乘以 16。

2. **縮放檢查**。對 $d_k = 4, 64, 1024$，隨機產生各分量為獨立標準常態的 query 與 key。分別量測有無除以 $\sqrt{d_k}$ 時 $q \cdot k$ 的變異數，以及在 100 個 key 上 softmax 的最大權重。不縮放時，softmax 會發生什麼事？

   提示

   原始點積的變異數接近 d_k；縮放後接近 1。不縮放時，在 $d_k = 1024$ 下 softmax 幾乎把所有權重都放在單一個 key 上（近乎 one-hot），流經其他 key 的梯度因此消失。

3. **移除位置資訊**。在 TinyViT 程式碼中，把位置嵌入設為零、拿掉類別 token 改用平均池化，然後在送進區塊之前打亂 patch token 的順序，並在評估模式下比較 logits。解釋結果，並說明這對一個沒有位置資訊的影像分類器意味著什麼。

   提示

   logits 在浮點誤差範圍內相同，因為每個區塊都是排列等變的，而平均池化是排列不變的。記得關掉 dropout（`eval()`），並打亂 token 而不是 patch 內的像素。這樣的模型無法分辨一張臉和用相同 patch 拼湊出來的亂序臉。

4. **交叉注意力的形狀**。在一個文字生成影像的模型中，256 個寬度 320 的影像 token 讀取 77 個寬度 768 的文字 token，使用 8 個寬度 40 的頭。寫出 W_Q、W_K、W_V、單一個頭的分數矩陣以及輸出的形狀。

   提示

   $W_Q \in \mathbb{R}^{320\times 320}$（8 個頭 × 40），$W_K, W_V \in \mathbb{R}^{768\times 320}$。每個頭的分數為 256×77。輸出跟隨 query：經過輸出投影後為 256×320。key 與 value 的長度必須彼此相同，而不必與 query 長度相同。

5. **Swin 的算術**。利用成本公式，比較 56×56 的 token 網格、$C = 96$、$M = 7$ 時的全域注意力與視窗注意力。兩種情況各由哪一項主導？

   提示

   $hw = 3136$。全域：$4 \cdot 3136 \cdot 96^2 \approx 1.16 \times 10^8$ 加上 $2 \cdot 3136^2 \cdot 96 \approx 1.89 \times 10^9$，由二次方項主導。視窗：同樣的 1.16×10^8 加上 $2 \cdot 49 \cdot 3136 \cdot 96 \approx 2.95 \times 10^7$，由投影主導。此例中視窗版本約便宜 14 倍。

6. **公平比較**。你有 2,000 張標註影像，以及預訓練好的 CNN、ViT 與視覺 Mamba 骨幹。設計一個實驗來決定要部署哪一個骨幹，並列出必須固定的條件。納入移除混合器的消融實驗，以及解析度／旋轉測試。

   提示

   先在凍結的特徵上比較線性探測（linear probe），再在驗證集上以對齊的預算、為各模型分別調校學習率來做微調。記錄預訓練資料、輸入解析度、參數量、FLOPs，以及在目標裝置上實測的延遲。拿預訓練的 ViT 去比從頭訓練的 CNN，測到的是預訓練，而不是架構。對掃描骨幹，記錄狀態大小，並測試保留閘控與局部卷積、移除混合器後，優勢是否仍存在。解析度與旋轉測試要使用一致的前處理。

參考文獻

1. A. Vaswani et al., “Attention Is All You Need,” NeurIPS, 2017. arXiv
2. A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” ICLR, 2021. arXiv
3. S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan and M. Shah, “Transformers in Vision: A Survey,” *ACM Computing Surveys*, 2022. arXiv
4. H. Touvron et al., “Training data-efficient image transformers & distillation through attention,” ICML, 2021. arXiv
5. Z. Liu et al., “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” ICCV, 2021. arXiv
6. T. Dao, D. Y. Fu, S. Ermon, A. Rudra and C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness,” NeurIPS, 2022. arXiv
7. PyTorch, `scaled_dot_product_attention` and `TransformerEncoderLayer` documentation. Attention, Encoder layer MultiheadAttention mask semantics
8. R. Xu et al., “Visual Mamba: A Survey and New Outlooks,” arXiv:2404.18861, 2024. arXiv
9. W. Yu and X. Wang, “MambaOut: Do We Really Need Mamba for Vision?,” CVPR, 2025. Paper
10. C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024, ch. 12. Book
11. K. Han et al., “A Survey on Vision Transformer,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. DOI
12. Y. Liu et al., “A Survey of Visual Transformers,” *IEEE Transactions on Neural Networks and Learning Systems*, 2024. DOI
13. L. Papa, P. Russo, I. Amerini and L. Zhou, “A Survey on Efficient Vision Transformers: Algorithms, Techniques, and Performance Benchmarking,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. DOI
14. J. L. Ba, J. R. Kiros and G. E. Hinton, “Layer Normalization,” arXiv:1607.06450, 2016. arXiv
15. R. Xiong et al., “On Layer Normalization in the Transformer Architecture,” ICML, 2020. arXiv
16. J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo and Y. Liu, “RoFormer: Enhanced Transformer with Rotary Position Embedding,” *Neurocomputing*, 2024. DOI
17. P. Shaw, J. Uszkoreit and A. Vaswani, “Self-Attention with Relative Position Representations,” NAACL, 2018. arXiv
18. M. Raghu, T. Unterthiner, S. Kornblith, C. Zhang and A. Dosovitskiy, “Do Vision Transformers See Like Convolutional Neural Networks?,” NeurIPS, 2021. arXiv
19. H. Wu et al., “CvT: Introducing Convolutions to Vision Transformers,” ICCV, 2021. arXiv
20. Z. Dai, H. Liu, Q. V. Le and M. Tan, “CoAtNet: Marrying Convolution and Attention for All Data Sizes,” NeurIPS, 2021. arXiv
21. D. Bolya et al., “Token Merging: Your ViT But Faster,” ICLR, 2023. arXiv
22. T. Darcet, M. Oquab, J. Mairal and P. Bojanowski, “Vision Transformers Need Registers,” ICLR, 2024. arXiv
23. M. Oquab et al., “DINOv2: Learning Robust Visual Features without Supervision,” *Transactions on Machine Learning Research*, 2024. arXiv
24. M. Dehghani et al., “Scaling Vision Transformers to 22 Billion Parameters,” ICML, 2023. arXiv
25. A. Steiner et al., “How to train your ViT? Data, Augmentation, and Regularization in Vision

Transformers," *Transactions on Machine Learning Research*, 2022. arXiv
26. D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," arXiv:1606.08415, 2016. arXiv
27. H. Touvron, M. Cord and H. Jégou, "DeiT III: Revenge of the ViT," ECCV, 2022. arXiv
28. S. Mehta and M. Rastegari, "MobileViT: Light-weight, General-purpose, and Mobile-friendly Vision Transformer," ICLR, 2022. arXiv
29. T. Dao, "FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning," ICLR, 2024. arXiv
30. L. Zhu et al., "Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model," ICML, 2024. arXiv
31. Y. Liu et al., "VMamba: Visual State Space Model," NeurIPS, 2024. arXiv
32. Z. Liu et al., "Swin Transformer V2: Scaling Up Capacity and Resolution," CVPR, 2022. arXiv
33. A. Gu and T. Dao, "Mamba: Linear-Time Sequence Modeling with Selective State Spaces," COLM, 2024. Author version
34. T. Dao and A. Gu, "Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality," ICML, PMLR 235, 2024. Paper

ML 06・表示學習與自監督

什麼是表示、為什麼分散式特徵能遷移，以及 SimCLR、MoCo、BYOL、SimSiam、DINO、MAE、JEPA 與 CLIP 如何在沒有標記的情況下學到特徵，並用線性探測與微調誠實評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/representation-learning/

你將學到

- 定義有用的表示與依任務而定的不變性。
- 透過最近鄰、線性探測或微調重用特徵。
- 比較對比、教師—學生、遮罩與特徵預測目標。
- 說明表示崩塌，以及抑制崩塌的機制。
- 在明確協定下評估遷移與影像—文字對齊。

先看全貌

假設你有一百萬張沒有標記的影像，但真正關心的任務只有 500 個標記樣本。只用這 500 個樣本從頭訓練一個大型網路，等於把幾乎所有可用的資料都丟掉了。**表示學習**把工作拆成兩步。第一步，從一個不需要你的標記的廣泛訊號學出編碼器 $h = f_\theta(x)$。第二步，用手上少量的標記，在 h 之上訓練一個小型的任務頭（head）。

> **用最簡單的話說**
>
> 小孩在學會每種鳥的名字之前，就已經會注意翅膀、鳥嘴、顏色和動作。後來爸媽指著三隻知更鳥和兩隻烏鴉，這樣就夠了：從此小孩就能分辨知更鳥和烏鴉。「注意」這件事是免費學來的，只是靠著看這個世界。表示就是這種可以重複使用的觀察方式，而自監督學習就是「只是看」的那個部分。

為什麼重要：幾乎所有現代視覺系統都從一個預訓練的編碼器出發。影像分類器、物件偵測器、分割模型、影像–文字模型，以及生成模型裡的影像編碼器，都建立在用這種方式學到的表示之上。起點的好壞，往往比任務頭的細節更重要。

本章先談什麼是表示、為什麼遷移有效，接著逐一介紹實務上主流的自監督目標，最後討論如何衡量一個表示到底好不好。本章假設你已經熟悉神經網路（ML 02）、CNN（ML 03）與 Vision Transformer（ML 05）。線性因子模型與自編碼器是學習表示的古典路線，另有專章：ML 06A。

什麼是表示？

白話版。表示就是模型用來描述一個輸入的那串數字。好的表示讓你關心的東西容易讀出來，並把你不關心的東西藏起來。

精確版。編碼器是一個函數 $f_\theta : \mathcal{X} \to \mathbb{R}^D$，把輸入 x（這裡是影像）對應到特徵向量 $h = f_\theta(x)$，其中 θ 是可學習的參數，D 是特徵維度。任務頭 g 把 h 對應到預測，所以完整模型是 $g \circ f_\theta$。如果一個簡單的任務頭（例如線性分類器）只用少量標記樣本就能從 h 解決任務，我們就說 h 是這個任務的好表示。Bengio、Courville 與 Vincent 的綜述 [1] 是這個觀點的經典陳述；它列出讓表示有用的「先驗」（priors），例如平滑性、多個解釋因子、由淺到深逐漸抽象的特徵階層，以及跨任務共享的因子。

局部表示與分散式表示

類別最簡單的表示是 **one-hot**：每個概念一個單元，且恰好只有一個單元是開啟的。它容易讀，卻無法泛化。知道「紅色汽車」完全不會告訴你任何關於「紅色卡車」的事，因為它們沒有共同的開啟單元。**分散式**表示則用許多特徵來描述每個輸入，而每個特徵又被許多輸入共享：「紅」、「藍」、「汽車」、「卡車」。於是「紅色卡車」就能重複使用從紅色汽車學來的「紅」這個單元。

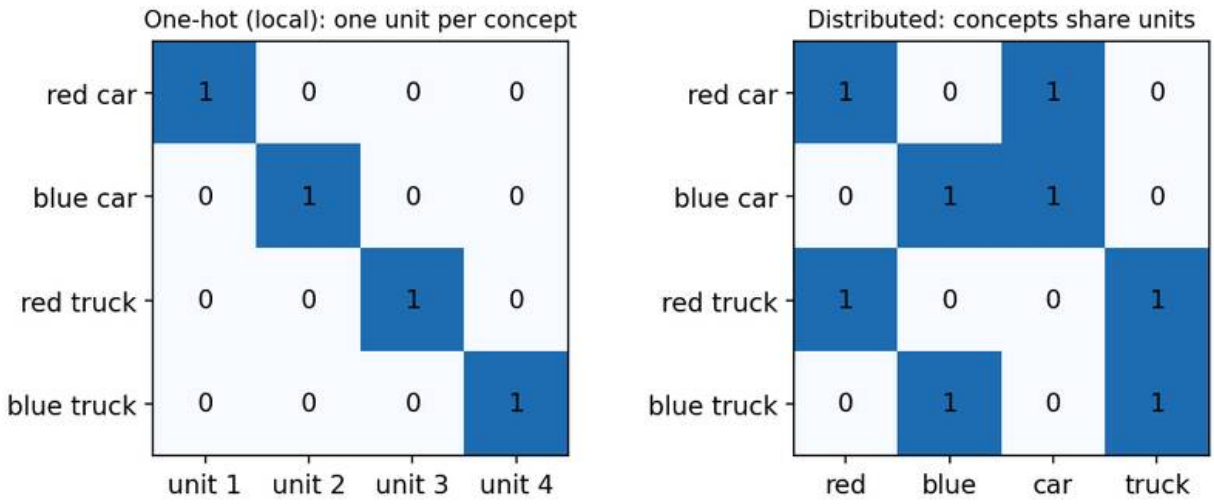


圖 1 — 四個概念的局部（*one-hot*）編碼與分散式編碼。在分散式編碼下，像紅色卡車這樣的新組合可以重複使用從其他樣本學到的特徵。

計數論證很驚人。用 D 個二元特徵，one-hot 編碼只能分辨 D 個概念，而分散式編碼原則上能分辨 2^D 種組態 [1]。深度網路會自動學出分散式編碼：每個隱藏單元對許多輸入有反應，每個輸入也會啟動許多單元。表示學習大部分的實際價值都來自這種共享。

不變性與等變性

有用的表示會保留與任務相關的差異，並壓抑會造成干擾的差異。

- 若 $f(T(x)) = f(x)$，特徵就對轉換 T 具有**不變性**（invariance）：把一隻貓平移兩個像素，不應該改變「這是一隻貓」。

- 若 $f(T(x)) = T'(f(x))$，特徵就具有**等變性**（equivariance），也就是以可預測的方式跟著改變：平移那隻貓，牠的分割遮罩也應該平移同樣的距離。

「有用」永遠取決於下游任務。顏色不變性有助於物體辨識，卻會傷害水果成熟度的估計。旋轉不變性對紋理有幫助，卻會搞混 6 和 9。密集預測（分割、深度）需要空間上等變的區塊（patch）特徵，而不是每張影像一個不變的向量。後面會看到，自監督目標主要是透過所使用的**資料增強**（data augmentation）來選擇不變性，所以這個選擇是建模決策，而不是細節。

學習距離：度量學習

PCA 在 ML 01 介紹過，並在 ML 06A 深入討論，它保留變異最大的方向。但變異大的方向可能來自光照，而不是物體身分。監督學習依照標記來塑造特徵；**度量學習**（metric learning）則依照應該配對的樣本來塑造距離。給定錨點（anchor）a、正例 p（同一身分）與負例 n（不同身分），三元組損失（triplet loss）是

$$\mathcal{L}_{\text{triplet}} = \max\left(0,\ \|h_a - h_p\|^2 - \|h_a - h_n\|^2 + m\right),$$

其中 h_a, h_p, h_n 是三個特徵向量，$m > 0$ 是邊界（margin）。當正例比負例近至少 m 時，損失為零。太容易的負例損失為零，什麼也學不到，所以挑選有資訊量的（「困難的」）負例很重要。下面的對比式自監督學習，就是一種用資料增強而不是標記來產生正例的度量學習。

遷移學習：預訓練一次，重複使用很多次

白話版。在一個大任務上訓練網路，保留它的特徵擷取器，再拿去用在小任務上。你可以凍結特徵、只訓練新的任務頭，也可以讓整個網路繼續學習。

精確版。預訓練在來源資料集或目標上找出編碼器參數 $\theta^\star$。遷移到一個有標記集合 $\{(x_i, y_i)\}_{i=1}^{n}$ 的目標任務時，有三種常見做法：

1. k **最近鄰（k-NN）：**不訓練。用測試特徵 $f_{\theta^\star}(x)$ 最相似的 k 個訓練特徵的標記來分類。
2. **線性探測：**凍結 $\theta^\star$，在 $h = f_{\theta^\star}(x)$ 上擬合線性分類器 $\hat{y} = \text{softmax}(Wh + b)$，其中 W 與 b 是唯一被訓練的參數。
3. **微調：**從 $\theta^\star$ 出發，連同任務頭一起更新編碼器，通常使用較小的學習率。

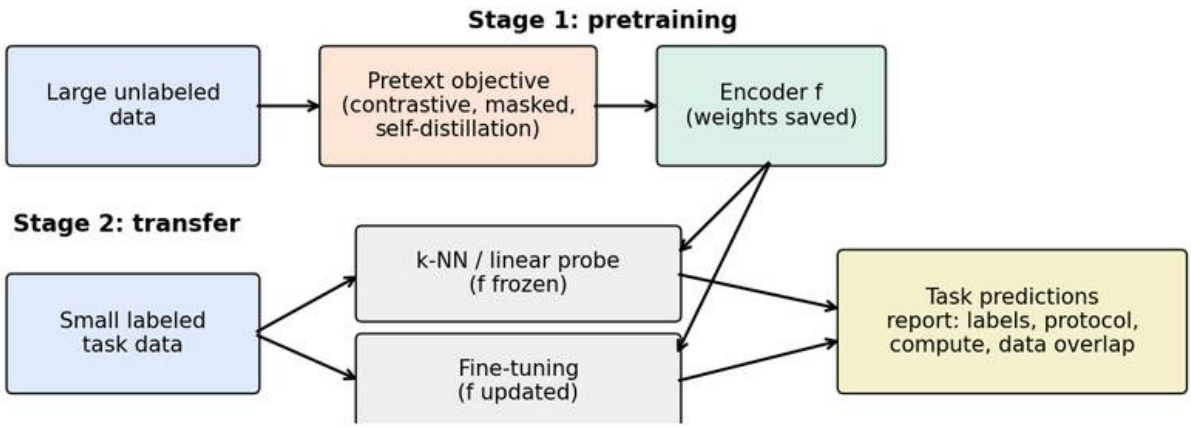


圖 2 — 表示學習的兩階段模式。預訓練昂貴但只做一次；遷移便宜且每個任務各做一次。

遷移為什麼會有效？Yosinski 等人 [2] 在 ImageNet 訓練的 CNN 上逐層量測。第一層特徵（很像 Gabor 濾波器的邊緣與顏色偵測器）是通用的：在任務之間遷移得很好。越高層的特徵越專屬於來源任務，而且來源任務與目標任務差異越大，可遷移性就越低。他們也發現，從遷移來的特徵初始化再微調，即使在目標任務上，也比從隨機初始化訓練有更好的泛化。

多年來，預訓練的預設來源是有監督的 ImageNet 分類。可是標記很昂貴，而且像「狗」這樣的標記，完全沒有告訴網路那隻狗的姿勢、草地或光線。自監督學習讓資料自己監督自己，移除了標記的瓶頸。

自監督學習：讓資料自己產生標記

白話版。從資料建立預測目標，例如匹配兩個視圖或恢復遮罩內容；學得的特徵能否遷移，仍須測試。

精確版。自監督目標是只從無標記輸入（或像影像–圖說這種天然配對的資料）計算出來的損失 $\mathcal{L}(\theta; x)$。Liu 等人 [3] 與 Gui 等人 [4] 的綜述，以及 Balestriero 等人實用的「食譜」[5]，把這個領域整理成下表的幾個家族：

家族	訓練訊號	代表方法	主要風險
對比式	拉近正配對，推開其他樣本	SimCLR、MoCo	需要大量負例；假負例
自蒸餾（無負例）	學生在另一個視圖上模仿教師	BYOL、SimSiam、DINO	缺少不對稱或中心化時會塌縮
冗餘抑制	一致性加上去相關、非常數的特徵維度	Barlow Twins、VICReg	正則項帶來額外超參數
遮罩建模	預測被遮住的像素或符元	BEiT、MAE	特徵可能偏重低階細節
特徵預測	預測被遮住區域的特徵	I-JEPA、V-JEPA 2	需要像蒸餾一樣的防塌縮設計
語言監督	讓影像與其圖說配對	CLIP、SigLIP	繼承網路資料的偏誤與雜訊

這些家族是學習目標，而不是架構：每一種都可以用來訓練 CNN 或 Transformer。現代系統常常結合好幾種。以下逐一介紹。

對比式學習與 InfoNCE 損失

白話版。讓同一張影像的兩個增強版本彼此一致，並且和批次中其他所有影像不一致。這是一道選擇題：「在這 N 個候選中，哪一個是我的搭檔？」

InfoNCE

van den Oord、Li 與 Vinyals 在對比預測編碼（Contrastive Predictive Coding, CPC）[6] 中提出 **InfoNCE** 損失。取一個查詢 q、一個「屬於」它的正例鍵 k^+，以及 $N-1$ 個負例鍵 $k_1^-, \dots, k_{N-1}^-$。給定相似度分數 $s(\cdot,\cdot)$ 與溫度 $\tau > 0$，

$$\mathcal{L}_{\text{InfoNCE}} = -\log \frac{\exp\left(s(q,k^+)/\tau\right)}{\exp\left(s(q,k^+)/\tau\right) + \sum_{j=1}^{N-1} \exp\left(s(q,k_j^-)/\tau\right)}.$$

這正是一個 N 類分類問題的 softmax 交叉熵，正確類別就是正例。由此得到兩個結果。

- **互資訊的下界**。CPC 證明 $I(q;k^+) \geq \log N - \mathcal{L}_{\text{InfoNCE}}$，其中 I 是互資訊（mutual information）。最小化損失會提高這個下界，而負例越多（N 越大）下界可以越高。實務上這個界很鬆，特徵品質也高度依賴編碼器與資料增強，所以請把它當成動機，而不是特徵為什麼好的解釋。
- **溫度控制難度**。除以較小的 τ 會讓 softmax 更尖銳，使損失集中在與查詢最相似的負例（困難負例）上。

SimCLR

SimCLR [7] 是最簡單形式的 InfoNCE。取一批 B 張影像，各產生兩個增強視圖（隨機裁切並縮放、顏色扭曲、模糊），共 $2B$ 個視圖。骨幹網路 f 把每個視圖對應到 h，一個小型 MLP **投影頭**（projection head）g 再把 h 對應到 $z = g(h)$，相似度用餘弦相似度 $s(z_i, z_j) = z_i^\top z_j / (\|z_i\|\|z_j\|)$。對於搭檔為 j 的視圖 i，

$$\ell_{i,j} = -\log \frac{\exp(z_i^\top z_j/\tau)}{\sum_{k \neq i} \exp(z_i^\top z_k/\tau)},$$

其中 z 已經做過 ℓ_2 正規化，總和遍及其他所有 $2B-1$ 個視圖。批次裡其他每張影像都是負例，因此不需要記憶庫，但這個方法受益於大批次。總損失對每個正配對的兩種順序取 $\ell_{i,j}$ 的平均；論文稱之為 NT-Xent 損失（normalized temperature-scaled cross-entropy）。

SimCLR 論文有三個值得記住的發現 [7]。資料增強的組合非常重要，其中隨機裁切加上顏色扭曲特別關鍵。投影頭有幫助：投影頭之前的特徵（h）比之後的（z）更能遷移，所以預訓練後會丟掉投影頭。而且對比式學習比監督學習更受益於大批次與更長的訓練。

程式：檢查損失與塌縮時的值

這段獨立的 PyTorch 範例用合成的配對向量驗證損失、自身遮罩與梯度。它不訓練編碼器。

```
import torch
from torch.nn import functional as F

torch.manual_seed(0)
B, D = 8, 16
base = torch.randn(B, D)
views = torch.cat([base + .1*torch.randn_like(base),
                   base + .1*torch.randn_like(base)]).requires_grad_()
z = F.normalize(views, dim=-1)
logits = z @ z.T / .2
self_mask = torch.eye(2*B, dtype=torch.bool)
logits = logits.masked_fill(self_mask, float("-inf"))
positive = (torch.arange(2*B) + B) % (2*B)
loss = F.cross_entropy(logits, positive)
loss.backward()
assert torch.isfinite(views.grad).all()
print("paired loss:", round(loss.item(), 4))

collapsed = F.normalize(torch.ones(2*B, D), dim=-1)
scores = (collapsed @ collapsed.T / .2).masked_fill(self_mask, float("-inf"))
collapsed_loss = F.cross_entropy(scores, positive)
print("collapsed loss:", round(collapsed_loss.item(), 4))
print("expected:", round(torch.log(torch.tensor(2*B-1.)).item(), 4))
```

它印出配對損失 0.1423 與塌縮損失 2.7081，正好等於 $\log 15$。如果所有表示都相同，正例得到的機率是 $1/(2B-1)$，損失就是 $\log(2B-1)$，也就是均勻亂猜的最差值。因此對比式的鑑別讓完全塌縮成為一個很差的解，雖然最佳化與負例選擇仍然可能很困難。

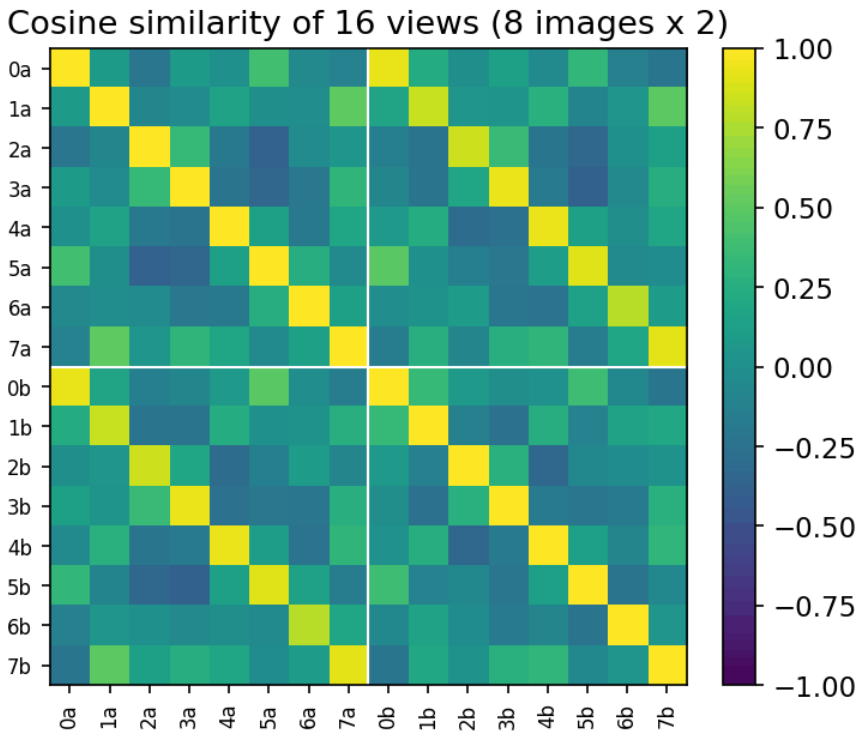


圖 *3* — 用本章稍後的小型 *SimCLR* 編碼器訓練後，*16* 個視圖（*8* 張數字影像，各兩個增強）之間的餘弦相似度。明亮的副對角線就是正配對；*InfoNCE* 把它們往上推，把其他所有元素往下壓。

MoCo：負例佇列與動量編碼器

InfoNCE 偏好大量負例，但 SimCLR 只能從當前批次取得負例。**動量對比**（Momentum Contrast, MoCo）[8] 用兩個想法把負例數量和批次大小脫鉤。

1. **佇列**。最近幾個批次編碼出來的鍵被存進長度為 K 的先進先出佇列，重複當作負例使用。無論批次多小，查詢都能看到 K 個負例。
2. **動量編碼器**。佇列裡的鍵是由稍早版本的網路產生的。如果鍵編碼器變化太快，新舊鍵就會不一致。因此 MoCo 用另一個編碼器計算鍵，它的參數 θ_k 緩慢地跟隨查詢編碼器的參數 θ_q：

$$\theta_k \leftarrow m\,\theta_k + (1-m)\,\theta_q,$$

其中 $m \in [0,1)$ 是動量，接近 1（論文預設 0.999）。只有 θ_q 接收梯度。這種**指數移動平均**（exponential moving average, EMA）教師的想法，在 BYOL、DINO 與 JEPA 中都會再次出現。

```
import torch
from torch.nn import functional as F

torch.manual_seed(0)
B, D, K = 8, 16, 32          # batch, embedding size, queue length
```

```
# --- MoCo-style InfoNCE: one positive key per query, negatives from a queue
q = F.normalize(torch.randn(B, D), dim=1)                 # queries (online encoder)
k = F.normalize(q + 0.1 * torch.randn(B, D), dim=1)       # keys (momentum encoder)
queue = F.normalize(torch.randn(K, D), dim=1)             # keys from past batches
l_pos = (q * k).sum(1, keepdim=True)                      # B x 1
l_neg = q @ queue.T                                       # B x K
logits = torch.cat([l_pos, l_neg], dim=1) / 0.07
loss_moco = F.cross_entropy(logits, torch.zeros(B, dtype=torch.long))
queue = torch.cat([k, queue])[:K]                         # enqueue new, drop oldest
print("MoCo logits:", tuple(logits.shape), "loss:", round(loss_moco.item(), 3))

# momentum (EMA) update of the key encoder's parameters
theta_q, theta_k, m = torch.randn(5), torch.zeros(5), 0.999
theta_k = m * theta_k + (1 - m) * theta_q
print("key params moved by:", round((theta_k.norm() / theta_q.norm()).item(), 4))
```

每個查詢面對一個 $1 + K$ 類的分類問題，正確答案永遠在第 0 欄，所以目標全為零。輸出顯示 logits 的形狀是 `(8, 33)`，而 EMA 一步只讓鍵編碼器往查詢編碼器移動了 0.1%（從零開始、走一步）。實際實作中，q 與 k 來自兩個編碼器分別作用在同一批影像的兩個增強版本上。

對齊性與均勻性

Wang 與 Isola [9] 從幾何角度清楚說明對比式損失在單位球面上做了什麼。在正規化特徵下，損失在漸近意義上最佳化兩個性質：

$$\mathcal{L}_{\text{align}} = \mathbb{E}_{(x,x^+)} \|f(x) - f(x^+)\|_2^2, \qquad \mathcal{L}_{\text{unif}} = \log \mathbb{E}_{x,y}\, e^{-t\|f(x)-f(y)\|_2^2},$$

其中 (x, x^+) 是正配對，x, y 是獨立樣本，$t > 0$（常用 $t = 2$）。正配對被映射到相近的點時，**對齊性**（alignment）就低。特徵在球面上分散開來時，**均勻性**（uniformity）就低，也就是盡可能保留輸入的資訊。塌縮的編碼器有完美的對齊性（0）和最差的均勻性（0）；有用的編碼器兩者都要小。

```
import torch
from torch.nn import functional as F

def alignment(z1, z2):            # mean squared distance between positive pairs
    return (z1 - z2).square().sum(1).mean()

def uniformity(z, t=2.0):         # log of mean Gaussian potential over all pairs
    return torch.pdist(z).square().mul(-t).exp().mean().log()

torch.manual_seed(0)
n, D = 512, 16
collapsed = F.normalize(torch.ones(n, D), dim=1)
spread = F.normalize(torch.randn(n, D), dim=1)
noisy_view = F.normalize(spread + 0.1 * torch.randn(n, D), dim=1)
print("collapsed: align", alignment(collapsed, collapsed).item(),
      "uniform", round(uniformity(collapsed).item(), 3))
print("spread:    align", round(alignment(spread, noisy_view).item(), 3),
```

```
      "uniform", round(uniformity(spread).item(), 3))
```

塌縮的集合得到對齊性 0.0、均勻性 0.0；分散的集合大約是 0.138 與 −3.51。圖 4 追蹤下面訓練數字編碼器時的這兩個量。在第 0 步，隨機網路把每個視圖都映射到幾乎同一點，所以對齊性接近零只是因為一切都塌縮了。訓練先把特徵分散開來（均勻性急遽下降），再逐步把正例拉回彼此身邊。

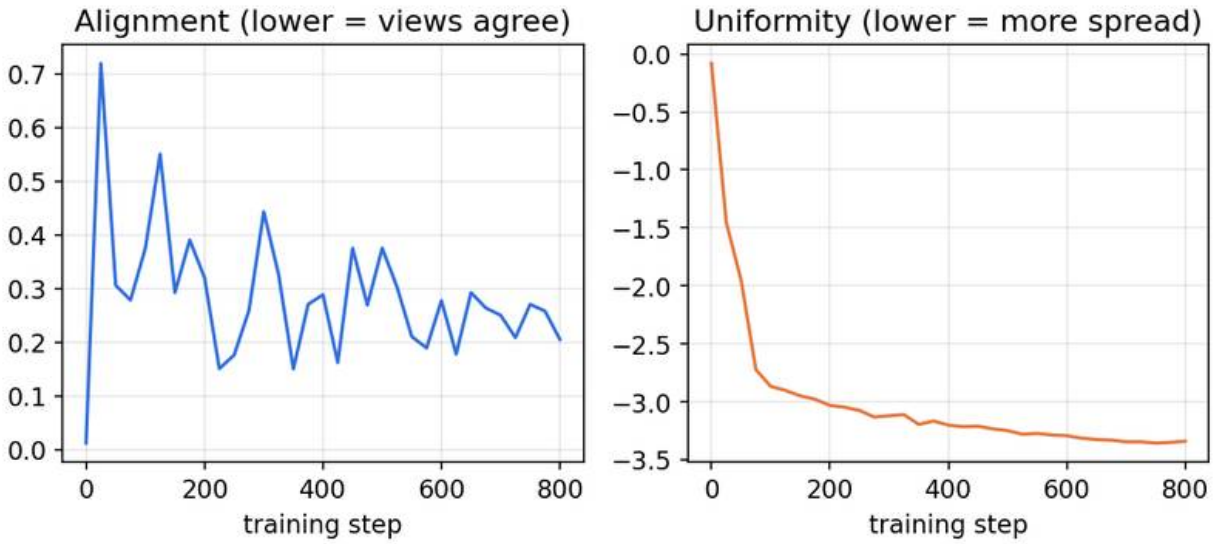


圖 *4* — 在 *8×8* 數字上以 *SimCLR* 訓練時，投影特徵的對齊性與均勻性。單獨的低對齊性沒有意義（常數編碼器也有）；有用的狀態是低對齊性與低均勻性同時成立。

實作範例：在 8×8 數字上跑 SimCLR

白話版。完全不使用標記，在 scikit-learn 的數字資料集上訓練一個迷你 SimCLR 編碼器，再檢查它的特徵能不能讓最近鄰分類器更穩健。

這裡的資料增強是最多一個像素的隨機平移、隨機亮度變化與像素雜訊。因此我們預期學到的特徵對小平移大致不變，這是原始像素做不到的。評估時，每一種特徵都使用同一組有標記的訓練集，並分別在乾淨的數字與最多平移一個像素的數字上測試。

```
import copy
import numpy as np
import torch
from torch import nn
from torch.nn import functional as F
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
torch.manual_seed(0); np.random.seed(0); torch.set_num_threads(1)
d = load_digits()
Xtr, Xte, ytr, yte = train_test_split(d.data / 16., d.target, test_size=.3,
                                      random_state=0, stratify=d.target)
Xtr_t = torch.tensor(Xtr, dtype=torch.float32)

def augment(x):                       # the augmentations define the invariances
    img = F.pad(x.view(-1, 1, 8, 8), (1, 1, 1, 1))
    i, j = np.random.randint(0, 3, size=2)               # shift by up to 1 px
    img = img[:, :, i:i + 8, j:j + 8]
    img = img * (0.7 + 0.6 * torch.rand(len(img), 1, 1, 1))  # brightness
    return (img + 0.1 * torch.randn_like(img)).reshape(len(x), 64)  # noise

backbone = nn.Sequential(nn.Unflatten(1, (1, 8, 8)),
                         nn.Conv2d(1, 32, 3, padding=1), nn.ReLU(),
                         nn.Conv2d(32, 64, 3, padding=1), nn.ReLU(),
                         nn.AdaptiveAvgPool2d(2), nn.Flatten())  # h: 256-d
random_backbone = copy.deepcopy(backbone)                    # baseline
head = nn.Sequential(nn.ReLU(), nn.Linear(256, 64))          # projection g
opt = torch.optim.Adam([*backbone.parameters(), *head.parameters()], lr=1e-3)

def nt_xent(z1, z2, tau=0.2):
    z = F.normalize(torch.cat([z1, z2]), dim=-1)
    n = len(z1)
    logits = (z @ z.T / tau).fill_diagonal_(float("-inf"))
    return F.cross_entropy(logits, (torch.arange(2 * n) + n) % (2 * n))

for step in range(800):                                  # about a minute on CPU
    x = Xtr_t[torch.randint(len(Xtr_t), (256,))]
    loss = nt_xent(head(backbone(augment(x))), head(backbone(augment(x))))
    opt.zero_grad(); loss.backward(); opt.step()

def shift(x, dy, dx):                    # move one 8x8 image, fill with zeros
    img = np.zeros((10, 10)); img[1:9, 1:9] = x.reshape(8, 8)
    return img[1 - dy:9 - dy, 1 - dx:9 - dx].ravel()

moves = np.random.default_rng(0).integers(-1, 2, size=(len(Xte), 2))
Xte_shift = np.stack([shift(x, *m) for x, m in zip(Xte, moves)])
enc = lambda net: (lambda A: net(torch.tensor(A, dtype=torch.float32)).detach().numpy())
pca = PCA(32, random_state=0).fit(Xtr)
for name, f in [("raw pixels", lambda A: A), ("PCA-32", pca.transform),
                ("random CNN", enc(random_backbone)), ("SimCLR CNN", enc(backbone))]:
    knn = KNeighborsClassifier(5, metric="cosine").fit(f(Xtr), ytr)
    print(f"{name:11s} clean {knn.score(f(Xte), yte):.3f}  "
          f"shifted {knn.score(f(Xte_shift), yte):.3f}")
```

在我們的機器上，輸出為：

```
raw pixels  clean 0.978  shifted 0.524
PCA-32      clean 0.978  shifted 0.507
random CNN  clean 0.974  shifted 0.524
SimCLR CNN  clean 0.987  shifted 0.919
```

確切數字會因 PyTorch 版本與硬體略有不同，重點是模式。在乾淨的數字上，原始像素已經非常好：這個資料集很簡單，花俏的編碼器幫助不大。在平移過的數字上，原始像素、PCA 以及同架構但未訓練的 CNN 都掉到 50% 左右，而自監督特徵保住了大部分的準確率。網路學到了我們透過資料增強要求的不變性，而且僅止於此：如果改用旋轉過的數字測試（資料增強從未出現過旋轉），就不應期待同樣的穩健性。

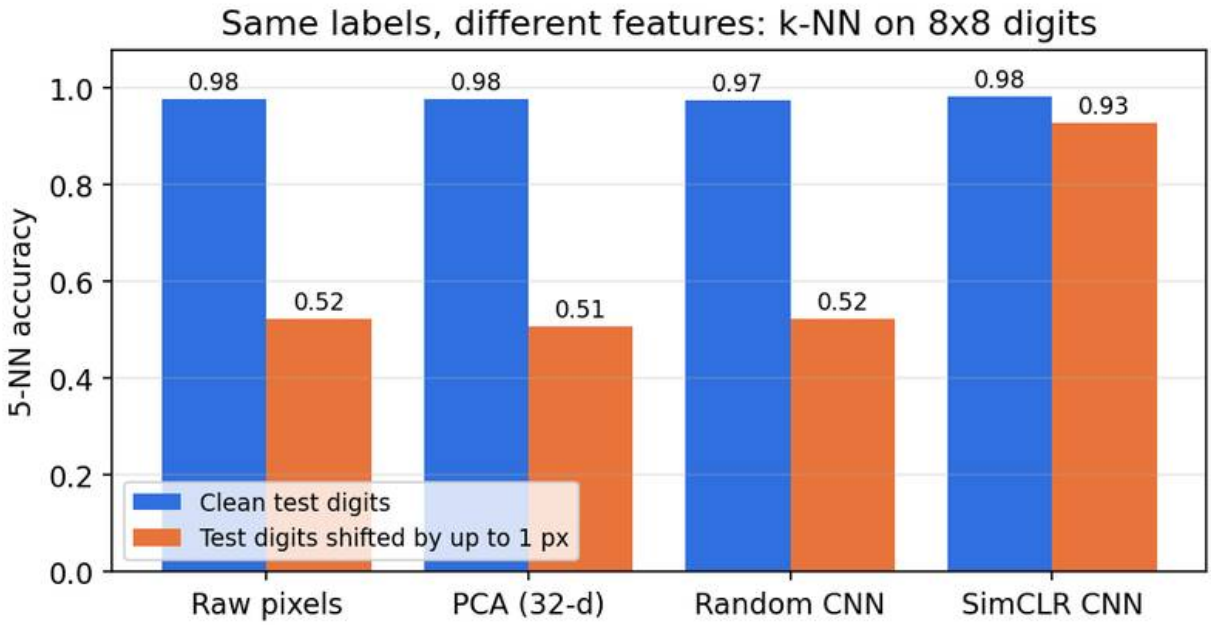


圖 5 — 相同的標記、相同的分類器、不同的特徵。繪圖腳本使用另一串亂數，所以數字與上面印出的結果最多相差 *0.01*。隨機 *CNN* 排除了「光靠卷積就能帶來穩健性」這個解釋。

僅線上版：*SimCLR*：正樣本對如何塑造表示 — *https://blog.shyandram.dev/zh-tw/learn/ml/representation-learning/papers/simclr 正樣本對如何塑造表示/*

沒有負例：為什麼只求一致會塌縮

白話版。如果唯一的指令是「讓兩個輸出一致」，那麼對每張影像都輸出同一個常數就完美達成了。沒有負例的方法必須靠別的機制阻止這件事。

對兩個視圖 v_1, v_2 最小化 $\|f_\theta(v_1) - f_\theta(v_2)\|^2$，有一個平凡解 $f_\theta(x) = c$，其中 c 是常數。光是一致，並不要求特徵攜帶任何資訊。**完全塌縮**把所有東西映射到同一點；**維度塌縮**（dimensional collapse）比較隱微：特徵會變化，但只沿著少數幾個方向，浪費了 D 維中的大部分。

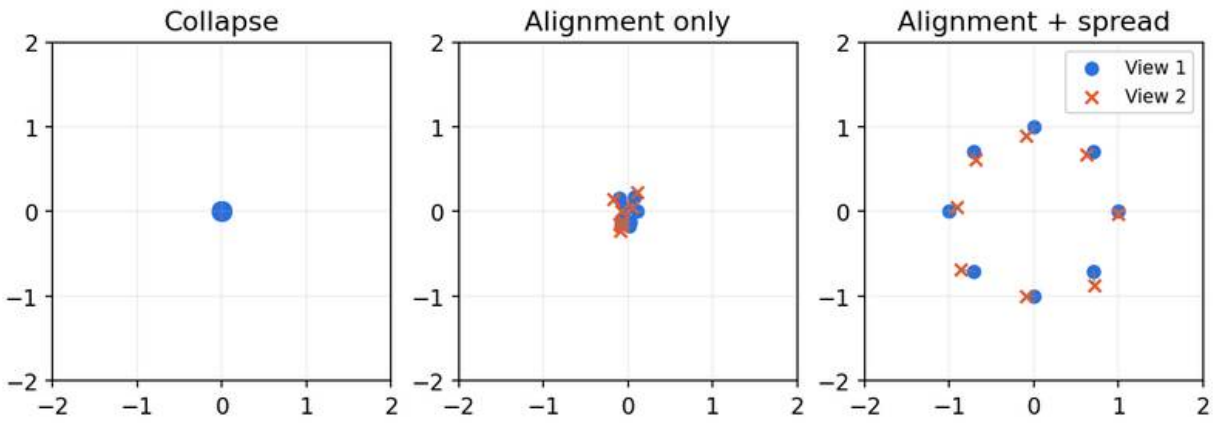


圖 6 — 塌縮、只有對齊沒有分散，以及對齊且分散的示意圖。這些點是刻意構造的說明，不是已發表模型學到的嵌入。

BYOL：線上網路追逐緩慢的目標

BYOL [10] 使用兩個網路。**線上**（online）網路（編碼器、投影器，再加一個**預測器** q_ϕ）被訓練去預測**目標**（target）網路對同一張影像另一個視圖的投影。目標網路不靠梯度訓練，而是線上權重的 EMA，和 MoCo 一樣：

$$\theta_{\text{target}} \leftarrow m\,\theta_{\text{target}} + (1-m)\,\theta_{\text{online}}.$$

損失是正規化後的預測與正規化後的目標投影之間的平方距離，並對兩個視圖做對稱化。目標輸出被當作常數（**停止梯度**，stop-gradient）。沒有負例，BYOL 實務上卻不會塌縮。不對稱性（一邊有預測器，另一邊有停止梯度與 EMA）以及完整的訓練配方都很重要；光有 EMA 並不能證明不可能塌縮。

SimSiam：停止梯度是關鍵成分

Chen 與 He 的 SimSiam [11] 把 BYOL 再進一步精簡：兩個視圖共用一個編碼器，一個分支上有預測器 h，**沒有**動量編碼器、**沒有**負例，也不需要大批次。令 $p_1 = h(f(v_1))$、$z_2 = f(v_2)$，

$$\mathcal{L} = \tfrac{1}{2}\,\mathcal{D}(p_1, \text{sg}(z_2)) + \tfrac{1}{2}\,\mathcal{D}(p_2, \text{sg}(z_1)), \qquad \mathcal{D}(p, z) = -\frac{p^\top z}{\|p\|\,\|z\|},$$

其中 sg 是停止梯度，$\mathcal{D}$ 是負的餘弦相似度。論文實驗顯示，拿掉停止梯度會讓模型很快塌縮，而預測器也是必要的。作者把這個方法詮釋為一種交替式、類似期望最大化（EM）的最佳化。實務上的教訓是：非常小的架構不對稱，就決定了這些方法是在學習還是在塌縮。

DINO：帶中心化與銳化的自蒸餾

DINO [12] 把一致性問題變成帶軟標記的分類。教師與學生都透過帶溫度的 softmax，在 K 個「原型」維度上輸出一個機率向量。看到視圖 x' 的學生，被訓練去符合教師在另一個視圖 x 上的

分布：

$$\mathcal{L} = -\sum_{k=1}^{K} P_t(x)_k \, \log P_s(x')_k, \qquad P_t(x) = \mathrm{softmax}\Big(\frac{g_t(x) - c}{\tau_t}\Big),$$

其中 g_t 是教師網路（學生的 EMA），c 是教師輸出的移動平均（**中心化**，centering），τ_t 是較低的教師溫度（**銳化**，sharpening）。中心化防止某一維獨大；銳化防止輸出變成均勻分布。兩者各自單獨使用都會以不同方式塌縮；合在一起才會平衡。DINO 也使用**多重裁切**（multi-crop）：教師只看大的全域裁切，學生還會看到小的局部裁切，因此學生必須從局部證據推斷全域內容。搭配 Vision Transformer 時，DINO 的注意力圖在完全沒有分割標記的情況下就能凸顯物體邊界，這點很受矚目。

Barlow Twins 與 VICReg：直接規範統計量

另一條路是直接在損失中禁止塌縮。**Barlow Twins** [13] 計算兩個視圖（在批次上標準化後）嵌入之間的 $D \times D$ 互相關矩陣 C，並把它推向單位矩陣：

$$\mathcal{L}_{\mathrm{BT}} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2,$$

第一項讓每一維在兩個視圖間一致，第二項（權重 $\lambda > 0$）讓不同維度去相關，避免攜帶重複的資訊。**VICReg** [14] 把同樣的想法拆成三個有名字的項：**不變性**（兩個嵌入間的均方距離）、**變異數**（用 hinge 損失讓每一維的標準差保持在門檻之上，直接禁止常數輸出）與**共變異數**（懲罰非對角的共變異數元素）。「沒有負例」不代表「沒有防塌縮設計」；這些方法只是把設計放進損失，而不是放進架構。

遮罩建模：從填空中學習

白話版。把輸入的一部分藏起來，預測缺了什麼。BERT 就是這樣學會語言的；只要選對遮住的比例，這招對影像也有效。

BEiT [15] 把 BERT 式的預訓練帶進 Vision Transformer。影像被切成區塊，部分區塊被遮住，網路為被遮住的位置預測離散的**視覺符元**（visual token），這些符元來自另外訓練好的影像符元化器（tokenizer）。

MAE [16] 則直接預測原始像素。它遮住很大比例的區塊，只把可見的區塊送進編碼器，再用一個輕量的解碼器重建缺失的像素。令被遮住的區塊索引集合為 $\mathcal{M}$，

$$\mathcal{L}_{\mathrm{mask}} = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \|\hat{x}_i - x_i\|^2,$$

其中 x_i 是區塊 i 的真實像素向量，$\hat{x}_i$ 是預測值。MAE 的設計是刻意的。高遮罩比例（預設 75%）讓任務夠難，網路無法只靠鄰居內插。編碼器只處理可見區塊，降低預訓練成本。用每個目標區塊自身的平均與變異數做正規化，可以改善特徵。預訓練完成後，解碼器就被丟掉。

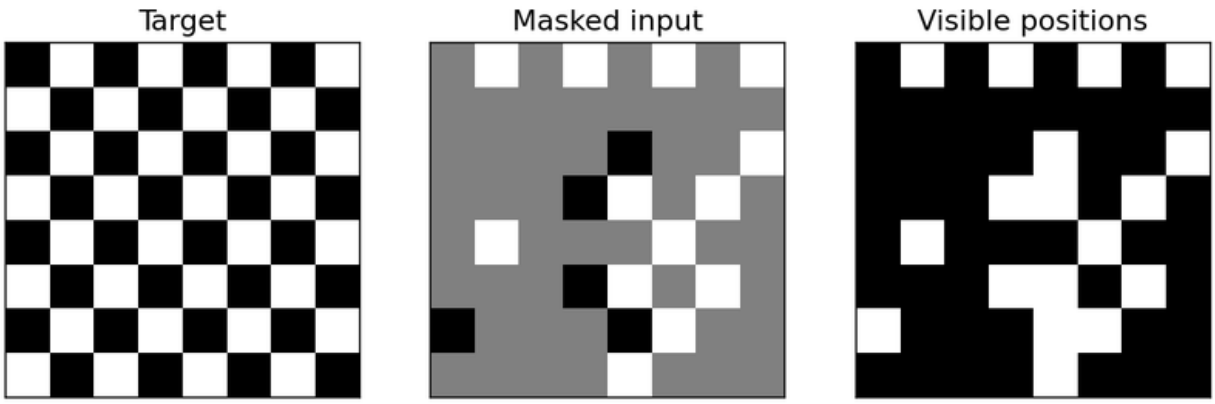


圖 7 — 遮罩預測從輸入本身造出目標。棋盤格只是說明遮罩操作，並不代表語意理解。

重建不等於生成。從可見像素預測被遮住像素的模型，並沒有被訓練成從零開始取樣出完整影像。低重建損失也不保證得到最好的語意特徵：背景紋理可能主導像素誤差。遮罩建模的特徵在線性探測下通常不如對比式或蒸餾式特徵，但微調之後很強，這也是為什麼評估方式必須講清楚。

預測特徵而不是像素：JEPA

白話版。與其把每個不確定的像素都畫出來，不如預測被遮住區域的描述。

I-JEPA [17]（影像聯合嵌入預測架構，Image Joint-Embedding Predictive Architecture）編碼一個可見的**脈絡**（context）區塊，並從中預測數個**目標**（target）區塊的編碼器特徵。目標編碼器是脈絡編碼器的 EMA，預測器則接收脈絡特徵以及目標的位置。簡化後的目標函數是

$$\mathcal{L}_{\text{pred}} = \left\| g_\phi\big(f_\theta(x_{\text{context}}), \text{positions}\big) - \text{sg}\left(f_{\bar{\theta}}(x_{\text{target}})\right) \right\|^2,$$

其中 f_θ 是脈絡編碼器，$f_{\bar{\theta}}$ 是 EMA 目標編碼器，g_ϕ 是預測器，sg 是停止梯度。這個式子只是概括想法；遮罩策略（大的、語意尺度的目標區塊）與架構都是方法的一部分，不是可有可無的細節。

V-JEPA 2 [18] 把特徵預測大規模延伸到影片。它的報告把無動作的影片預訓練，與之後用少量機器人資料訓練、用於規劃的動作條件模型區分開來。預測未來的潛在狀態、辨識動作、生成逼真的未來畫面，是不同的能力；一個「世界模型」的主張應該說清楚它指的是哪一種。

以語言作為監督：CLIP

白話版。對齊影像與圖說嵌入，再把影像與候選類別的文字描述比較；提示措辭會影響結果。

CLIP [19] 用從網路收集的影像–文字配對來訓練。在一批 N 個配對中，影像編碼器產生正規化嵌入 $u_1, \dots, u_N$，文字編碼器產生 $v_1, \dots, v_N$。$N \times N$ 的相似度矩陣 $S_{ij} = u_i^\top v_j / \tau$（$\tau$ 是可學習的溫度）的對角線就是正確配對。損失是兩個 InfoNCE 損失的平均，一個沿著列（哪句圖說配這張影像？），一個沿著行（哪張影像配這句圖說？）：

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}\Big[\frac{1}{N}\sum_i -\log\frac{e^{S_{ii}}}{\sum_j e^{S_{ij}}} + \frac{1}{N}\sum_j -\log\frac{e^{S_{jj}}}{\sum_i e^{S_{ij}}}\Big].$$

零樣本分類隨之而來。為每個類別嵌入一句提示，例如「a photo of a dog」，再把影像分到提示嵌入最相似的類別。不需要任何任務專屬的標記，不過提示的措辭會影響準確率。

SigLIP [20] 把 softmax 換成對每一個影像–文字配對各自獨立的 **sigmoid** 損失：矩陣中的每個元素都變成一個是非題：「這兩個配對嗎？」這樣就不需要對整個批次做正規化，簡化了大規模分散式訓練。

```
import torch
from torch.nn import functional as F

torch.manual_seed(0)
D = 16

# --- CLIP-style symmetric loss on N matched image-text pairs
N = 6
img = F.normalize(torch.randn(N, D), dim=1)
txt = F.normalize(img + 0.3 * torch.randn(N, D), dim=1)   # caption i matches image i
logit_scale = torch.tensor(1 / 0.07)
sim = logit_scale * img @ txt.T                           # N x N, diagonal = matches
labels = torch.arange(N)
loss_clip = (F.cross_entropy(sim, labels) + F.cross_entropy(sim.T, labels)) / 2
print("CLIP loss:", round(loss_clip.item(), 3))

# zero-shot classification: embed one prompt per class, pick the most similar
class_prompts = txt[:3]                                   # stand-ins for "a photo of a
 ↪ {class}"
pred = (img @ class_prompts.T).argmax(1)
print("zero-shot predictions for images 0-2:", pred[:3].tolist())

# SigLIP-style alternative: an independent yes/no decision for every pair
t, b = 10.0, -10.0                                        # learnable in practice
signs = 2 * torch.eye(N) - 1                              # +1 on matches, -1 elsewhere
loss_sig = -F.logsigmoid(signs * (t * img @ txt.T + b)).sum() / N
print("sigmoid loss:", round(loss_sig.item(), 3))
```

合成的圖說是影像嵌入加上雜訊的副本，所以 CLIP 損失很小（0.054），零樣本步驟也找回了正確的「類別」`[0, 1, 2]`。在真正的 CLIP 中，兩個編碼器分別是 Vision Transformer 或 CNN 與文字 Transformer，而困難的部分在資料：圖說描述了什麼、遺漏了什麼，決定了表示知道什麼。sigmoid 損失的值（3.473）不能和 softmax 損失直接比較；它的單位不同，還多了偏置項 b。

DINOv2 與 DINOv3：把特徵當成產品

白話版。只要有足夠的精選資料與算力，預訓練就能產出一個可以凍結、並重複用在許多不同任務上的編碼器，從分類到深度估計都行。

DINOv2 [21] 結合影像層級的自蒸餾（如 DINO）與區塊層級的遮罩目標，並且非常重視**資料策展**（data curation）：從未經整理的影像中，透過與精選影像比對檢索，建出一個大型、多樣且去除重複的預訓練集。成果是一系列凍結後就能搭配簡單任務頭、全域與區塊特徵都好用的編碼器。2025 年發表的 DINOv3 [22] 進一步擴大規模，並加入 **Gram anchoring**，以對抗長時間訓練中密集（區塊）特徵品質下降的問題。它不只是讓每個特徵各自匹配，而是約束區塊特徵之間的相似度矩陣，使其接近較早期模型的矩陣。報告也把一個非常大的教師模型蒸餾成較小的模型以便部署。

實務上的問題是：凍結的全域或密集特徵能不能解決**你的**任務。一個擅長語意檢索的模型，可能會漏掉製造零件上的一道小刮痕。提高輸入解析度並使用區塊特徵可能有幫助，但這會改變運算預算與比較的方式。

誠實地評估遷移

白話版。以明確調適協定，在保留的任務或資料上測試特徵；評估須與特徵及超參數選擇分開。

評估方式	訓練什麼	衡量什麼
k-NN	不訓練；與有標記參考特徵的相似度	表示的局部結構
線性探測	線性任務頭；編碼器凍結	簡單讀出就能取得的資訊
微調	任務頭加上部分或全部編碼器層	在給定標記與運算預算下的可調適性
密集遷移	在區塊特徵圖上的空間任務頭	是否適合像素或區域層級的輸出

線性探測與微調

微調的自由度較高，所以當目標資料充足、且與預訓練資料相似時，它通常會贏。但它不一定比較好。Kumar 等人 [23] 從理論與實驗上證明，完整微調可能會**扭曲**好的預訓練特徵：當隨機初始化的任務頭還是錯的時候，它的大梯度流進編碼器，把特徵改往適合訓練分布、卻傷害**分布外**（out-of-distribution, OOD）輸入的方向。線性探測不會扭曲特徵。他們的對策 **LP-FT** 分兩步：先凍結編碼器訓練線性任務頭，再從這個任務頭出發微調全部參數。

我們在數字範例上重現了這個定性效應。編碼器是上面的 SimCLR CNN；有標記的訓練數字是乾淨的；測試數字則是平移過的，屬於輕微的分布偏移。每種標記預算都對三次隨機抽取的標記樣本取平均。

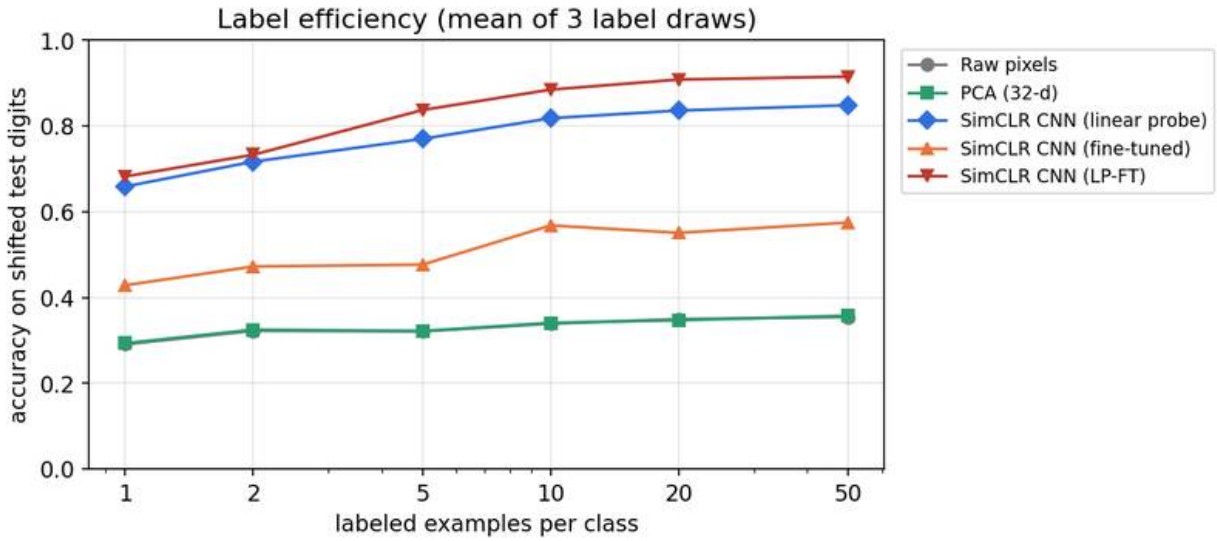


圖 *8* — 平移測試數字上的標記效率。自監督特徵比像素或 *PCA* 更省標記。在固定的一套訓練設定下，從隨機任務頭開始的一般微調降低了平移測試的準確率，而 *LP-FT* 則比探測更好。這是玩具實驗的結果：調好學習率或有更多標記時，一般微調可能表現更好。

請看圖 8 的形狀，而不是確切數值。凍結的自監督特徵搭配線性任務頭，每類只要少數幾個標記，就能勝過每類 50 個標記的原始像素。在這裡，從隨機初始化任務頭開始的一般微調輸給了凍結的探測，而 LP-FT 兩者都贏，這和 Kumar 等人在分布偏移下回報的順序相同。微調需要自己的超參數搜尋，任務頭的初始化也很重要。

防止資料洩漏的規則

- 做相似度比較時，特徵要一致地正規化（餘弦 *k*-NN 需要正規化向量）。
- 標準化、PCA 與分類器的超參數，只能在訓練與驗證資料上擬合。
- 凍結的編碼器仍然有訓練歷史：預訓練資料與評估資料的重疊可能灌水結果，網路規模的模型尤其如此。
- 回報標記數量、骨幹網路、檢查點、輸入解析度、資料增強、任務頭、調參預算與評估方式。在測試集上從許多編碼器中挑最好的，就是測試資料洩漏，即使沒有任何一個被微調。

現代觀點

綜述與回顧論文。四篇綜述從互補的角度描繪這個領域。

- **Bengio、Courville 與 Vincent（TPAMI 2013）[1]** 為整件事定下框架。他們問一個表示憑什麼比另一個好，並以一組通用先驗回答：分散式編碼、深度與抽象、解釋因子的解糾纏（disentangling）、流形與自然分群。他們回顧了機率模型、自編碼器與流形學習。這篇早於現代自監督學習，但仍是「我們為什麼要學表示」最好的陳述。

- **Liu 等人（TKDE 2023）[3]** 把橫跨視覺、語言與圖的自監督方法分成**生成式**（自編碼與自迴歸重建）、**對比式**（脈絡–實例與實例–實例）與**生成–對比式**（對抗式）三組，並討論當時對比式目標為何在視覺基準上勝出。
- **Gui 等人（TPAMI 2024）[4]** 對演算法做了較新的整理，涵蓋對比式、自蒸餾、冗餘抑制與遮罩影像建模方法，並討論應用與開放問題，例如理論、代理任務的選擇與多模態自監督學習。
- **Balestriero 等人（2023）[5]** 是由領域內許多作者合寫的實務指南。它把方法分成深度度量學習家族（SimCLR 及其相關方法）、自蒸餾家族（BYOL、SimSiam、DINO）、典型相關分析家族（VICReg、Barlow Twins）以及遮罩影像建模，並收集了論文常常埋在細節裡的「訣竅」（資料增強、投影器、EMA 排程、批次大小的影響）。Prince 的教科書 [24] 則以平易、圖文並茂的方式並附上 notebook 介紹同樣的概念。

實務現況（2023–2026）。從上面引用的論文中可以看到以下趨勢：

- **結合多種目標**。領先的影像編碼器把影像層級的蒸餾與區塊層級的遮罩預測結合起來（DINOv2 [21]、DINOv3 [22]），而不是只依賴一種代理任務。
- **資料是一級設計選擇**。 DINOv2 的策展流程，以及 CLIP [19] 與 SigLIP [20] 背後網路規模的配對收集，都顯示預訓練吃進什麼資料和損失函數一樣重要。
- **密集特徵**。分割、深度與對應關係需要好的區塊特徵，這促成了像 Gram anchoring [22] 這樣的目標。
- **在潛在空間中預測，並延伸到影片**。 I-JEPA [17] 與 V-JEPA 2 [18] 預測特徵而不是像素，並延伸到影片與動作條件預測。
- **與語言對齊的編碼器成為多模態模型的預設骨幹**（見 ML 10）。

開放問題。我們仍然缺少一套能預測「哪種代理任務適合哪種下游任務」的理論；防止塌縮靠的多半是配方而不是保證；評估結果高度依賴評估方式，以及預訓練與測試資料之間的重疊；網路規模的預訓練也繼承了資料中的偏誤與缺口。我們對 [3]、[4]、[5]、[21]、[22] 的解讀是：資料與目標的設計值得和架構一樣多的關注。這是對文獻的詮釋，不是「某個目標到處都贏」的定理。

接下來的章節換一個問題：與其描述一張影像，模型要如何描述並**取樣**整個分布？自編碼器見 ML 06A，生成模型見 ML 07。

重點整理

- 有用特徵讓目標任務更容易解決，分散式編碼本身不保證泛化。
- 資料擴增決定忽略哪些差異，應保留下游任務需要的差異。
- 對比目標區分正配對與其他候選；只要求一致，會允許常數輸出的崩塌解。
- 教師不對稱、變異數限制與冗餘懲罰，以不同方式處理崩塌。
- 重建像素、預測特徵與對齊圖說，訓練的是不同能力。
- 最近鄰測試、線性探測與微調回答不同問題；應在預期分布轉移下評估調適。

練習

1. **溫度與塌縮**。在第一段程式中，把溫度從 0.2 改成 0.05 和 1.0。塌縮損失會變嗎？配對損失呢？

提示

塌縮損失仍是 $\log 15$：所有非自身的 logits 都相等，所以除以任何 τ 之後 softmax 仍是均勻的。配對損失會改變，因為溫度縮放的是分數之間的差距；有真正的差距時，τ 越小 softmax 越尖銳。

2. **資料增強就是假設**。你想要的特徵用於 (a) 顯微影像中的細胞計數、(b) 鳥種辨識、(c) 讀取路標文字。各舉一個你會拿掉的標準 SimCLR 資料增強，並說明原因。

提示

(a) 隨機裁切可能改變視野中的細胞數量，所以「相同」的標記其實不再相同。(b) 強烈的顏色抖動可能抹去羽色，而羽色是重要的鳥種線索。(c) 水平翻轉會把文字變成鏡像文字。先決定什麼必須保留，再選擇不變性。

3. **MAE 的運算量**。一張影像有 196 個區塊，MAE 遮住 75%。有多少個區塊進入編碼器？編碼器中兩兩注意力分數的數量縮小為幾分之幾？

提示

49 個可見區塊進入編碼器。注意力分數隨符元數量的平方成長：$(49/196)^2 = 1/16$。解碼器仍然要處理全部 196 個位置，但它很小。

4. **破壞數字實驗**。修改 SimCLR 數字程式，讓 `augment` 不做平移。重新訓練，並比較平移測試的 k-NN 準確率與原本的結果。這告訴你什麼？

提示

把隨機的 `i, j` 換成 `i, j = 1, 1`。在我們的機器上，平移測試的準確率從 0.919 掉到 0.743，而乾淨測試的準確率維持在 0.981。少了平移增強，目標中就沒有任何東西要求平移不變性；剩下的穩健性來自訓練過的卷積加池化編碼器本來就會學到的東西（主範例中未訓練的那個完全沒有）。決定學到什麼的主要是資料增強，而不是「自監督」這個名詞。

5. **紙上的 SimSiam 消融實驗**。在 SimSiam 中，假設你拿掉停止梯度，讓兩個分支都接收梯度。用兩句話說明為什麼常數輸出會變成很容易達到的最小值，以及為什麼光有預測器不足以阻止它。

提示

兩邊都有梯度時，只要 p_1 和 z_2 指向同一方向，損失 $-\cos(p_1, z_2)$ 就達到最小，而一個常數編碼器加上一個輸出同樣常數方向的預測器，對每個輸入都做得到。停止梯度改變了最佳化動態，讓目標在每一步中保持固定；SimSiam 論文回報，沒有它就會迅速塌縮。

6. **公平的探測**。設計一個實驗，在每類 10、50、100 個標記下，比較凍結的預訓練編碼器與 PCA 加邏輯斯迴歸。列出三件必須固定的事，以及兩件必須回報的事。

提示

必須固定：有標記的子集（兩者使用相同樣本）、保留的測試切分，以及選擇正則化強度的程序（只用驗證資料）。必須回報：預訓練編碼器的訓練資料（以及與你測試集的任何重疊），以及隨機抽取標記的次數與其變異範圍。

參考文獻

1. Y. Bengio, A. Courville and P. Vincent, “Representation Learning: A Review and New Perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, 2013. Paper ·arXiv
2. J. Yosinski, J. Clune, Y. Bengio and H. Lipson, “How transferable are features in deep neural networks?” NeurIPS, 2014. Paper
3. X. Liu et al., “Self-supervised Learning: Generative or Contrastive,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, 2023 (online 2021). Paper ·arXiv
4. J. Gui et al., “A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024. Paper ·arXiv
5. R. Balestriero et al., “A Cookbook of Self-Supervised Learning,” arXiv:2304.12210, 2023. Paper
6. A. van den Oord, Y. Li and O. Vinyals, “Representation Learning with Contrastive Predictive Coding,” arXiv:1807.03748, 2018. Paper
7. T. Chen, S. Kornblith, M. Norouzi and G. Hinton, “A Simple Framework for Contrastive Learning of Visual Representations,” ICML, 2020. Paper
8. K. He, H. Fan, Y. Wu, S. Xie and R. Girshick, “Momentum Contrast for Unsupervised Visual Representation Learning,” CVPR, 2020. Paper
9. T. Wang and P. Isola, “Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere,” ICML, 2020. Paper
10. J.-B. Grill et al., “Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning,” NeurIPS, 2020. Paper
11. X. Chen and K. He, “Exploring Simple Siamese Representation Learning,” CVPR, 2021. Paper
12. M. Caron et al., “Emerging Properties in Self-Supervised Vision Transformers,” ICCV, 2021. Paper
13. J. Zbontar, L. Jing, I. Misra, Y. LeCun and S. Deny, “Barlow Twins: Self-Supervised Learning via Redundancy Reduction,” ICML, 2021. Paper
14. A. Bardes, J. Ponce and Y. LeCun, “VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning,” ICLR, 2022. Paper
15. H. Bao, L. Dong, S. Piao and F. Wei, “BEiT: BERT Pre-Training of Image Transformers,” ICLR, 2022. Paper
16. K. He et al., “Masked Autoencoders Are Scalable Vision Learners,” CVPR, 2022. Paper
17. M. Assran et al., “Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture,” CVPR, 2023. Paper
18. M. Assran et al., “V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning,” technical report, 2025. Report
19. A. Radford et al., “Learning Transferable Visual Models From Natural Language Supervision,” ICML, 2021. Paper
20. X. Zhai, B. Mustafa, A. Kolesnikov and L. Beyer, “Sigmoid Loss for Language Image Pre-Training,” ICCV, 2023. Paper
21. M. Oquab et al., “DINOv2: Learning Robust Visual Features without Supervision,” *Transactions on Machine Learning Research*, 2024. Paper
22. O. Siméoni et al., “DINOv3,” technical report, 2025. Report
23. A. Kumar, A. Raghunathan, R. Jones, T. Ma and P. Liang, “Fine-Tuning can Distort Pretrained

Features and Underperform Out-of-Distribution," ICLR, 2022. Paper
24. S. J. D. Prince, *Understanding Deep Learning*, MIT Press, 2023. Book and notebooks

ML 06A・線性因子模型與自編碼器

PCA、機率 PCA、因子分析、ICA、稀疏編碼、NMF 與自編碼器（欠完備、稀疏、去噪、收縮）：壓縮、雜訊假設與流形觀點如何一路通往 VAE。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/factor-models-autoencoders/

你將學到

- 說明潛在因素如何描述相關量測。
- 比較 PCA、機率式 PCA 與因素分析的雜訊假設。
- 區分獨立性、稀疏性與非負限制。
- 連結線性及非線性自編碼器與重建目標。
- 理解流形假設，以及為何只有重建不足以定義取樣。

先看全貌

一張 8×8 的數字影像有 64 個像素值，一張手機照片則有數百萬個。然而，一批這類影像中真正有意義的變化，只取決於少得多的潛在量：是哪個數字、筆畫粗細、傾斜角度、位置、光照。**因子模型**試圖用少數幾個隱藏（潛在）座標來解釋每一筆觀測。**自編碼器**則是一種神經網路，它學著把輸入擠進這樣的座標，再展開還原。

> **用最簡單的話說**
>
> 想像你在電話裡向別人描述朋友的長相。你不會逐一唸出每顆雀斑，而是說「圓臉、短髮、笑容很大、光從左邊打過來」。這幾個詞就是因子：聽的人靠它們就能在腦中拼出一張不錯的畫面。因子模型要找的，是能同時解釋最多張臉的那幾個詞。自編碼器則是一對小幫手：一個把圖片變成幾個詞，另一個把詞變回圖片。兩人一起練習，直到還原的圖片看起來夠像為止。

為什麼要用一整章談這個？因為現代表示學習與生成模型裡幾乎每個想法，都能在這裡找到祖先。用於視覺化的降維、去噪、壓縮、異常偵測、把影像轉成緊湊編碼供擴散模型使用的「tokenizer」，以及 VAE 的機率架構，全都從同一個問題出發：哪一小組原因可能產生了這些觀測？從線性因子模型到自編碼器的脈絡，依循 Goodfellow、Bengio 與 Courville [1, 第 13–14 章] 以及 Murphy [2]；Bengio、Courville 與 Vincent 的回顧論文 [3] 則說明了學習到的表示為何重要。

一個範本，多種模型

本章每一個方法都套用同一個範本。**編碼器**（encoder）把觀測 $x \in \mathbb{R}^d$ 映射成編碼 $z \in \mathbb{R}^k$，**解碼器**（decoder）再把編碼映回重建結果 $\hat{x}$。各模型的差別只在三個選擇：

選擇	本章出現的選項
編碼器／解碼器的形式	線性（PCA、因子分析、ICA、NMF、稀疏編碼）或非線性（深度自編碼器）
編碼的大小	**欠完備**（undercomplete，$k < d$：瓶頸迫使模型壓縮）或**過完備**（overcomplete，$k \geq d$：必須靠其他機制阻止模型直接複製）
額外假設或懲罰	高斯雜訊（PPCA、FA）、獨立且非高斯（ICA）、稀疏（稀疏編碼、稀疏自編碼器）、非負（NMF）、對破壞具穩健性（去噪）、對微小輸入變化不敏感（收縮）

請記住這張表。兩個方法結論不同時，幾乎總是因為它們在某一列做了不同的選擇，而不是其中一個「錯了」。

PCA：最佳的線性重建子空間

白話版。找幾個彼此正交的方向，使得每筆資料只保留在這些方向上的座標時，損失的資訊越少越好。

令 $X \in \mathbb{R}^{n \times d}$ 的每一列是一筆已中心化（減去各欄平均）的資料，共 n 筆。PCA 選擇一個欄向量互相正交且長度為 1 的矩陣 $W \in \mathbb{R}^{d \times k}$，使平方重建誤差最小：

$$\min_{W} \ \|X - XWW^{\top}\|_F^2 \qquad \text{subject to} \qquad W^{\top}W = I_k.$$

其中 $\|\cdot\|_F$ 是 Frobenius 範數（所有元素平方和的平方根），$Z = XW \in \mathbb{R}^{n \times k}$ 是**編碼**（也稱分數），$\hat{X} = ZW^{\top}$ 是重建結果。編碼器是 $x \mapsto W^{\top}x$，解碼器是 $z \mapsto Wz$：PCA 本身就是一個自編碼器，只是線性且權重共用。

解從哪裡來

兩種等價的觀點會得到同一個答案。

最大變異數。取一個單位向量 w。投影後資料 Xw 的變異數是 $w^{\top}Sw$，其中 $S = \frac{1}{n-1}X^{\top}X$ 是樣本共變異數矩陣。用拉格朗日乘數 λ，在 $w^{\top}w = 1$ 的限制下最大化 $w^{\top}Sw$：

$$\nabla_w (w^{\top}Sw - \lambda(w^{\top}w - 1)) = 0 \quad \Longrightarrow \quad Sw = \lambda w.$$

所以 w 必須是 S 的特徵向量，而它捕捉到的變異數就是對應的特徵值 λ。最好的方向是特徵值最大的特徵向量；在與第一個方向正交的限制下，次好的方向是第二個特徵向量，依此類推。

最小重建誤差。由於 W 的欄向量正交歸一，誤差可以拆成「總變異數減去保留的變異數」。因此最小化誤差等同於最大化保留的變異數，而殘差恰好是被捨棄的特徵值總和（乘上 $n-1$）。Eckart–Young 定理用矩陣語言說了同一件事：把奇異值分解 $X = U\Sigma V^{\top}$ 截斷到最大的 k 個奇異值，就得到 Frobenius 範數下最佳的秩 k 近似。PCA 方向就是前 k 個右奇異向量（V 的欄），而共變異數的特徵值為 $\sigma_j^2/(n-1)$。

程式：用 SVD 實作 PCA

下面的程式在 scikit-learn 內建 digits 資料集的訓練部分上擬合 PCA，在保留的測試影像上評估重建，並與 scikit-learn 的實作互相核對。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split

X = load_digits().data / 16.
train, test = train_test_split(X, test_size=.2, random_state=0)

mean = train.mean(axis=0)
U, s, Vt = np.linalg.svd(train - mean, full_matrices=False)
var = s**2 / (len(train) - 1)              # eigenvalues of the covariance
ratio = np.cumsum(var) / var.sum()
for k in (2, 10, 20):
    W = Vt[:k].T                           # d x k, orthonormal columns
    rec = (test - mean) @ W @ W.T + mean
    print(f"k={k:2d}  explained={ratio[k-1]:.3f}  "
          f"test MSE={np.mean((test - rec)**2):.4f}")

sk = PCA(n_components=10).fit(train)
print("matches sklearn:", np.allclose(np.abs(sk.components_), np.abs(Vt[:10])))
```

輸出：

```
k= 2  explained=0.291  test MSE=0.0546
k=10  explained=0.740  test MSE=0.0202
k=20  explained=0.896  test MSE=0.0084
matches sklearn: True
```

最後一行的 `np.abs` 並不是投機取巧：每個主方向本來就只確定到正負號，因為 w 與 $-w$ 張成同一條直線。也請注意，平均值只在訓練資料上估計，再套用到測試資料；若用測試資料估計平均值，就會洩漏資訊。

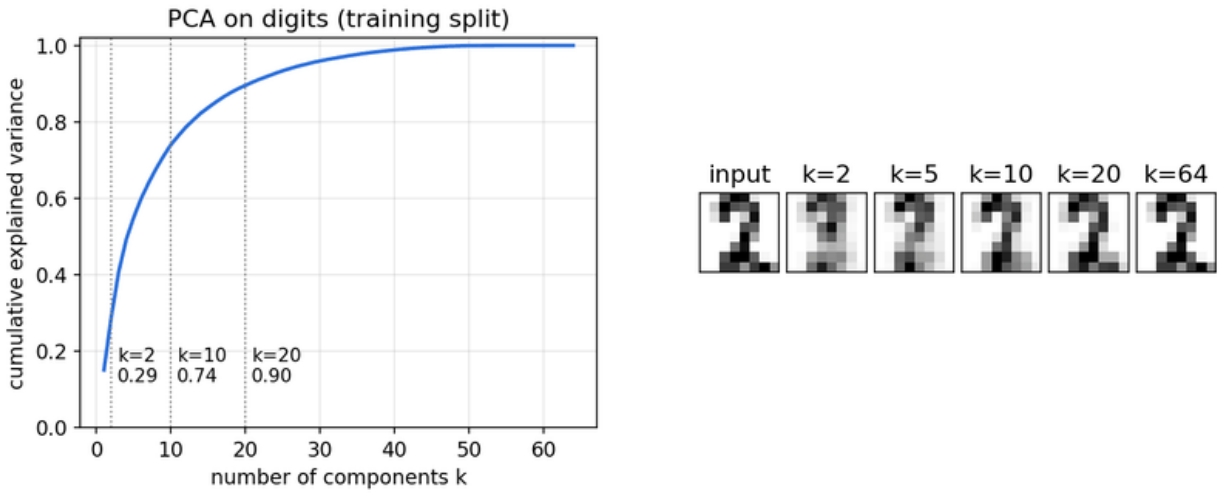


圖 *1* — 在內建 *digits* 資料集上的 *PCA*。左：累積解釋變異數（在訓練資料上擬合）。右：一張保留的測試數字用 k 個成分重建；$k = 64$ 時可完整還原輸入。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

PCA 沒有承諾的事

從目標函數可以直接得出三個提醒。

- **中心化與尺度會改變答案**。沒有中心化時，第一個「成分」大多指向平均值。若把每個特徵標準化為單位變異數，每個像素的權重就相同，因此原始像素 PCA 與標準化 PCA 解的是不同問題。
- **變異數不等於相關性**。變異數最大的方向可能是光照或背景，而不是你關心的類別。PCA 是非監督式的；它最大化保留的變異數，而不是類別分離度（那是線性判別分析等方法的工作）。
- **PCA 是線性的**。若資料落在彎曲的曲面上，平坦的子空間可能需要很多維度，才能描述其實是低維的形狀。我們會在流形一節回到這一點。

機率 PCA 與因子分析

白話版。與其只找一個子空間，不如寫下一個資料如何產生的故事：先抽出幾個隱藏因子，把它們線性混合，再加上雜訊。接著用最大概似法擬合這個故事。

兩個模型共用同一條生成方程式：

$$x = \mu + Wz + \epsilon, \qquad z \sim \mathcal{N}(0, I_k),$$

其中 $\mu \in \mathbb{R}^d$ 是平均值，$W \in \mathbb{R}^{d \times k}$ 是**負荷矩陣**（loading matrix，描述每個因子如何出現在每個量測座標上），z 是潛在因子向量，ϵ 是與 z 獨立的雜訊。兩者只差在雜訊。

機率 PCA（PPCA）

在 PPCA [4] 中，雜訊是等向的，$\epsilon \sim \mathcal{N}(0, \sigma^2 I_d)$：每個座標的雜訊變異數都相同。把 z 積分掉，得到高斯邊際分布

$$x \sim \mathcal{N}(\mu,\, C), \qquad C = WW^\top + \sigma^2 I_d.$$

這個共變異數有清楚的解讀：低秩部分 $WW^\top$ 由因子解釋，再加上一個因子解釋不了的固定「底噪」σ^2。Tipping 與 Bishop 證明最大概似解有封閉形式 [4]：

$$W_{\mathrm{ML}} = U_k(\Lambda_k - \sigma^2 I)^{1/2} R, \qquad \sigma^2_{\mathrm{ML}} = \frac{1}{d-k}\sum_{j=k+1}^{d} \lambda_j,$$

其中 U_k 是樣本共變異數前 k 個特徵向量，Λ_k 是對應的特徵值 $\lambda_1, \dots, \lambda_k$，$R$ 是任意 $k \times k$ 正交矩陣。換句話說：負荷矩陣恰好張成 PCA 子空間，而雜訊變異數就是 PCA 丟掉的變異數的平均。因此 PPCA 就是 PCA 再加上一個概似函數。這個概似函數很有用：你可以在保留資料上比較不同 k 的模型、用期望最大化（EM）演算法處理缺值，也能建立 PPCA 的混合模型。

給定一筆觀測，其因子的後驗分布是高斯分布，平均為

$$\mathbb{E}[z \mid x] = M^{-1} W^\top (x - \mu), \qquad M = W^\top W + \sigma^2 I_k.$$

這就是機率式的編碼器。當 $\sigma^2 \to 0$ 時，它趨近一般的 PCA 投影；當 $\sigma^2 > 0$ 時，它會把編碼往零收縮，因為看到的東西有一部分被歸因於雜訊。

因子分析（FA）

因子分析（factor analysis）保留相同結構，但允許對角的雜訊共變異數，$\epsilon \sim \mathcal{N}(0, \Psi)$，$\Psi = \mathrm{diag}(\psi_1, \dots, \psi_d)$。每個觀測座標各有自己的殘差雜訊水準 ψ_i，而潛在因子負責解釋座標之間**共享**的共變異數。邊際分布為 $x \sim \mathcal{N}(\mu, WW^\top + \Psi)$，因子的後驗平均為 $\mathbb{E}[z \mid x] = W^\top (WW^\top + \Psi)^{-1}(x - \mu)$。

FA 沒有封閉解，因此用 EM 或直接最佳化概似函數來擬合。當量測來自雜訊程度不一的感測器，或某些像素幾乎恆定（例如 digits 影像邊框上永遠空白的像素）時，這份額外的彈性就很重要。PCA 會毫不猶豫地把成分花在任何高變異數的座標上；FA 則可以把個別雜訊很大的座標解釋成「雜訊」，把因子留給座標之間的相關性。

旋轉問題

兩個模型都有同一種不確定性。對任意正交矩陣 R，把 W 換成 WR、z 換成 $R^\top z$，$WW^\top$ 不變，因此 x 的分布也不變，因為 $R^\top z$ 仍是標準常態。子空間可以被確定，個別的座標軸卻不行。這就是因子分析文獻使用 varimax 等旋轉來挑選較好解讀的座標軸的原因，也說明了因子不會自動等同於物理上的成因。若要確定個別因子，我們必須在高斯以外再假設些什麼，而 ICA 正是這麼做的。

僅線上版：*Probabilistic PCA*：子空間背後的機率模型 — *https://blog.shyandram.dev/zh-tw/learn/ml/factor-models-autoencoders/papers/probabilistic-pca 子空間背後的機率模型/*

獨立成分分析（ICA）

白話版。一個房間裡好幾個人同時講話，幾支麥克風各自錄到不同比例的混音。ICA 試著找出統計上盡可能獨立、而且盡可能不像高斯分布的組合，藉此還原每個人的聲音。

無雜訊的 ICA 模型為

$$x = As, \qquad s = (s_1, \dots, s_k) \text{ with independent components,}$$

其中 $x \in \mathbb{R}^d$ 是觀測到的混合訊號，$A \in \mathbb{R}^{d \times k}$ 是未知的**混合矩陣**，來源 s_j 彼此獨立且**非高斯**。目標是找一個解混矩陣 $B \approx A^{-1}$（或其偽逆），使 $\hat{s} = Bx$ 還原出來源 [5]。

為什麼關鍵在於非高斯

上一節的旋轉問題說明了 ICA 為何需要非高斯來源。先把資料**白化**（whitening）：轉換 x 使其共變異數為單位矩陣（先做 PCA，再把每個成分縮放到單位變異數即可）。白化之後，剩下的未知量只是一個旋轉。如果來源是高斯，獨立單位高斯的任何旋轉仍然是獨立單位高斯，於是沒有任何旋轉比較好。ICA 的可識別性定理說的是反方向：只要**至多一個**來源是高斯，來源就能被還原，只剩三種無傷大雅的不確定性 [5]：

- **順序**（我們無法得知哪個來源是「第一個」），
- **正負號**，以及
- **尺度**（大聲的來源配上較小的混合係數，看起來一模一樣）。

中央極限定理提供了找到正確旋轉的直覺：獨立變數的總和往往比各部分更像高斯。所以 x 的投影中最不像高斯的那些，最可能是單一來源而不是混合。因此 ICA 演算法會最大化某種非高斯程度的量度。一個有理論依據的量度是**負熵**（negentropy）：

$$J(y) = H(y_{\text{gauss}}) - H(y) \;\geq 0,$$

其中 H 是微分熵，y_{gauss} 是與 y 變異數相同的高斯變數。在變異數固定的所有分布中，高斯分布的熵最大，所以 $J(y) = 0$ 恰好發生在 y 為高斯時。由於熵很難估計，**FastICA** 演算法用 $G(u) = \log\cosh u$ 等對比函數近似負熵，並以快速的不動點迭代找出每個解混方向 [5]。最大化非高斯程度，與最小化各估計成分之間的互資訊、以及 ICA 模型的最大概似估計密切相關。

程式：把三個訊號解混

這個玩具問題用了一個正弦波、一個方波與一個鋸齒波，全都是非高斯，並以固定矩陣混合。

```
import numpy as np
from sklearn.decomposition import FastICA, PCA

rng = np.random.default_rng(0)
t = np.linspace(0, 8, 2000)
S = np.c_[np.sin(2*t),                  # smooth sine
          np.sign(np.sin(3*t)),         # square wave
          2*(t % 1) - 1]                # sawtooth
```

```
S += 0.05 * rng.normal(size=S.shape)
A = np.array([[1.0, 1.0, 1.0],
              [0.5, 2.0, 1.0],
              [1.5, 1.0, 2.0]])        # unknown mixing matrix
X = S @ A.T                            # three "microphones"

def best_match(est):
    C = np.abs(np.corrcoef(S.T, est.T)[:3, 3:])
    return np.round(C.max(axis=1), 3)

ica = FastICA(n_components=3, whiten="unit-variance", random_state=0)
print("ICA |corr| with each source:", best_match(ica.fit_transform(X)))
print("PCA |corr| with each source:", best_match(PCA(3).fit_transform(X)))
```

輸出：

```
ICA |corr| with each source: [0.997 0.999 0.997]
PCA |corr| with each source: [0.689 0.811 0.735]
```

我們用相關係數的絕對值並取最佳配對來比較，因為 ICA 回傳的來源順序、正負號與尺度都是任意的。PCA 只做去相關：它找的是高變異數的正交方向，在這裡仍然是混合訊號。

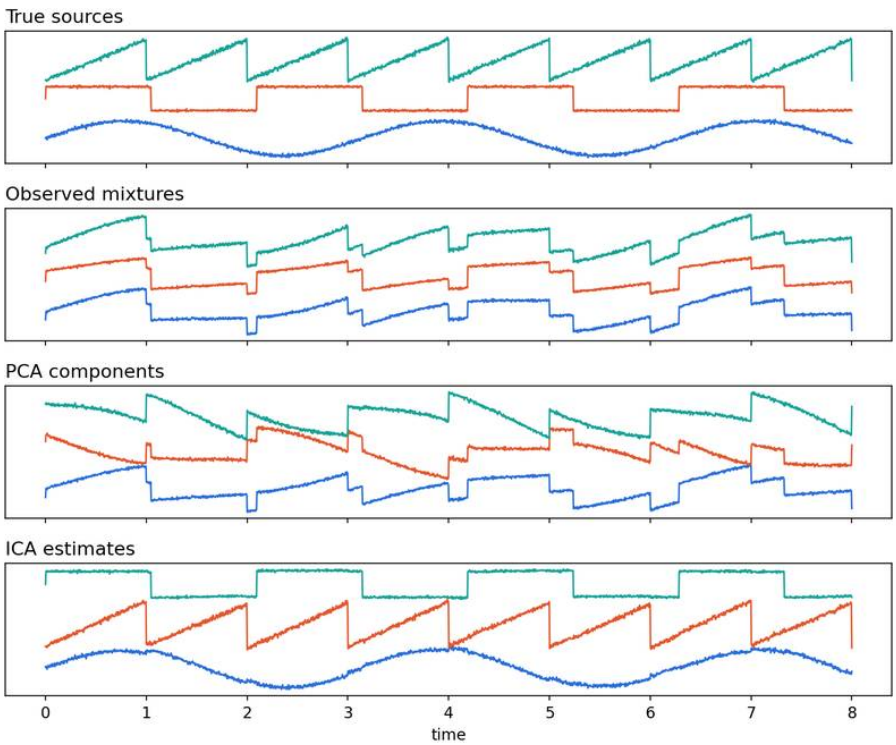


圖 2 — 盲訊號分離。*PCA* 成分仍是混合訊號；*FastICA* 在順序、正負號與尺度的不確定性之內，還原出正弦波、方波與鋸齒波。

ICA 廣泛用於分離腦電圖（EEG）等感測紀錄中的雜訊與偽影。它的假設也劃出了它的極限：它需要真正獨立、非高斯的來源，以及接近線性且瞬時的混合過程。在影像區塊（patch）上，ICA 往往學到局部化、有方向性的邊緣狀濾波器，這也引出一個密切相關的想法。

稀疏編碼

白話版。準備一本很大的小圖樣字典（「原子」，atom），每次描述一個輸入時只用其中少數幾個。稀疏編碼學習一個字典 $D = [d_1, \dots, d_m] \in \mathbb{R}^{d \times m}$，並為每筆資料 x_i 學一個稀疏編碼 $\alpha_i \in \mathbb{R}^m$，求解

$$\min_{D,\,\{\alpha_i\}} \sum_{i=1}^{n} \left(\tfrac{1}{2} \|x_i - D\alpha_i\|_2^2 + \lambda \|\alpha_i\|_1 \right) \qquad \text{subject to} \qquad \|d_j\|_2 \le 1.$$

第一項要求準確重建；ℓ_1 範數 $\|\alpha\|_1 = \sum_j |\alpha_j|$ 把大多數編碼元素推成恰好為零；$\lambda > 0$ 決定兩者的取捨。原子的範數限制防止模型作弊，也就是把原子放得很大、編碼縮得很小。字典常是**過完備**的（$m > d$）：原子比像素維度多，但對任何一個輸入只有少數幾個會啟動。

Olshausen 與 Field [6] 在自然影像的區塊上訓練這類模型，發現學到的原子是局部化、有方向性、帶通的，非常像初級視覺皮質中簡單細胞的感受野。這個結果影響深遠，因為它顯示：只靠稀疏性、不需要任何標籤，就能解釋生物視覺的一項已知特性。

有兩個實務重點讓稀疏編碼不同於本章稍後的自編碼器：

- **推論本身是一個最佳化問題**。給定 D，要為新輸入找出 α，就得解一個 lasso 問題。這裡沒有固定的編碼器網路；「編碼器」是一個演算法。訓練時交替更新編碼（固定 D）與字典（固定編碼）。
- **稀疏性是一種先驗**。ℓ_1 懲罰對應於編碼上的 Laplace 先驗，所以稀疏編碼可以看成：在一個具有稀疏、非高斯先驗的線性因子模型中做最大後驗（MAP）推論，而這正是 ICA 所屬的同一個家族。

程式：稀疏程度與重建精確度

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import DictionaryLearning
from sklearn.model_selection import train_test_split

X = load_digits().data / 16.
train, test = train_test_split(X, test_size=.2, random_state=0)
for alpha in (0.1, 1.0):
    dl = DictionaryLearning(n_components=32, alpha=alpha, max_iter=30,
                            transform_algorithm="lasso_lars",
                            transform_alpha=alpha, random_state=0)
    dl.fit(train[:600])
    codes = dl.transform(test)
    rec = codes @ dl.components_
    active = np.mean(np.abs(codes) > 1e-8, axis=1).mean()
    print(f"alpha={alpha}: active atoms per image={active*32:.1f}/32, "
```

```
          f"test MSE={np.mean((test - rec)**2):.4f}")
```

輸出（在筆電 CPU 上約需半分鐘）：

```
alpha=0.1: active atoms per image=17.9/32, test MSE=0.0075
alpha=1.0: active atoms per image=3.0/32, test MSE=0.0401
```

提高 λ（這裡的 `alpha`）可以換到稀疏得多的描述，每張影像約啟動 3 個原子而不是 18 個，代價是重建誤差變高。兩種設定都沒有「對錯」：壓縮編解碼器、去噪器與可解釋性工具各自會做不同的選擇。請注意，這個字典沒有減去平均值，所以它的誤差不能直接與上面的 PCA 數字比較。

非負矩陣分解（NMF）

白話版。把每筆資料描述成非負部件的加法組合：你可以加上筆畫，但永遠不能減掉筆畫。

對非負資料 $X \geq 0$（像素強度、詞頻、光譜），NMF 尋找 $Z \geq 0$ 與 $H \geq 0$，使下列重建準則最小：

$$\min_{Z \geq 0,\, H \geq 0} \|X - ZH\|_F^2,$$

其中 $H \in \mathbb{R}^{k \times d}$ 的每一列是一個非負部件，$Z \in \mathbb{R}^{n \times k}$ 的每一列是非負的混合權重。因為沒有東西可以互相抵消，這些部件往往看起來像物體的局部，而不是 PCA 那種有正有負的「幽靈」圖樣。Lee 與 Seung [7] 在人臉與文字上展示了這種以部件為基礎的行為，並推廣了簡單的乘法更新規則來擬合模型。

兩點提醒。第一，以部件為基礎是一種傾向，而不是保證：在某些資料集上，因子並不是乾淨的部件。第二，目標函數對 Z 與 H 聯合而言是非凸的，所以初始化很重要，不同次執行可能得到不同的因子。scikit-learn 的 `init="nndsvda"` 用以 SVD 為基礎的起點讓結果更穩定。

程式：比較各因子模型的重建

這個 CPU 範例只在訓練用的數字上擬合各模型，並在保留的影像上評估重建。它比較的是不同的模型假設，並不是宣稱某個方法在所有任務上都勝出。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA, FactorAnalysis, NMF

X = load_digits().data / 16.
train, test = train_test_split(X, test_size=.2, random_state=0)
models = {
    "PCA": PCA(n_components=12),
    "factor analysis": FactorAnalysis(n_components=12, random_state=0),
    "NMF": NMF(n_components=12, init="nndsvda", max_iter=1000,
               random_state=0),
```

```
}
for name, model in models.items():
    model.fit(train)
    z = model.transform(test)
    if name == "factor analysis":
        reconstructed = z @ model.components_ + model.mean_
    else:
        reconstructed = model.inverse_transform(z)
    mse = np.mean((test-reconstructed)**2)
    print(name, "held-out reconstruction MSE:", round(mse, 5))
```

輸出：

```
PCA held-out reconstruction MSE: 0.01652
factor analysis held-out reconstruction MSE: 0.02501
NMF held-out reconstruction MSE: 0.02207
```

這次執行中，PCA 的保留測試重建 MSE 最低。但最佳性定理針對的是擬合子空間所用、已中心化的**訓練矩陣**，不保證在每一份新測試集上誤差都最低。其他模型的假設若更適合目標母體，也可能有較好的泛化表現。這裡因子分析的重建是後驗平均，會把每個像素的一部分歸因於雜訊；FA 的完整機率模型也不能只用 MSE 概括。NMF 則加入非負、可加的因素限制。保留測試誤差應靠實驗比較，不能把這次排名當成定理。

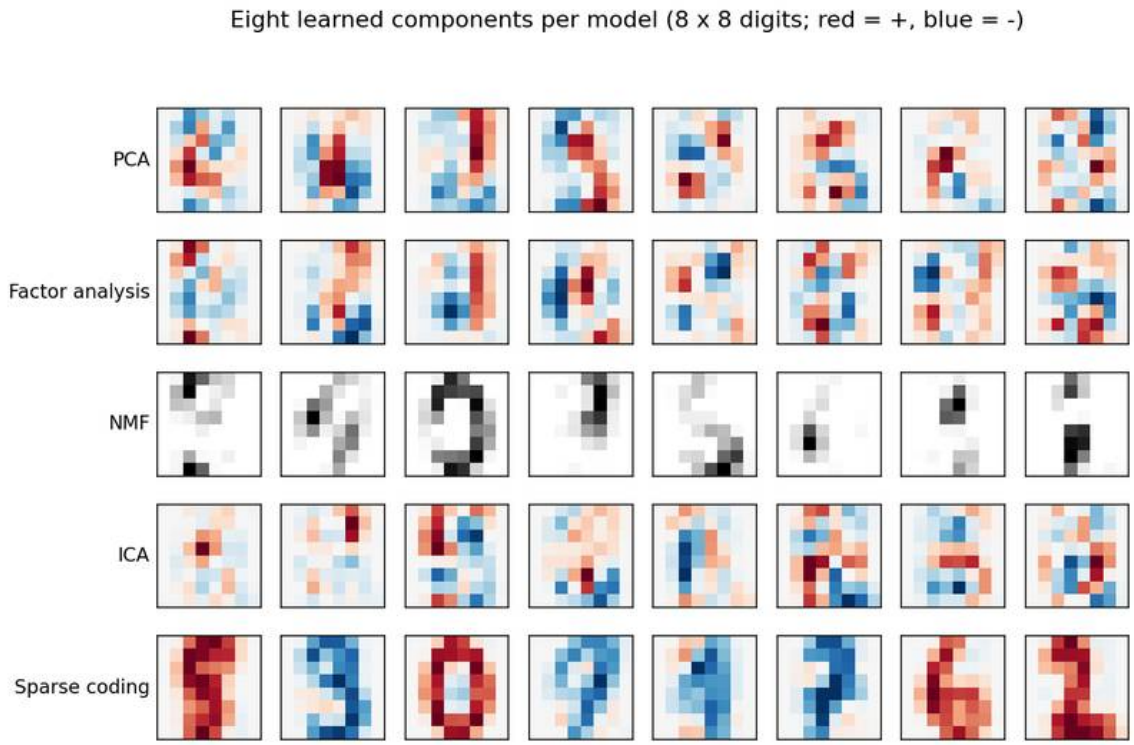


圖 3 — 各線性因子模型在同一批數字上學到的八個成分。*PCA* 與 *FA* 給出有正負號的全域圖樣；*NMF* 給出非負的筆畫；*ICA* 給出有正負號但較局部的圖樣；稀疏編碼的原子像是原型，每次只組合少數幾個。

線性自編碼器與 PCA

白話版。對置中資料使用平方重建誤差時，線性瓶頸自編碼器可在全域最佳解恢復 PCA 子空間，但訓練未必到達該解。

對中心化的輸入，線性自編碼器計算

$$z = W_e x, \qquad \hat{x} = W_d z, \qquad \mathcal{L}(W_e, W_d) = \frac{1}{n}\sum_{i} \|x_i - W_d W_e x_i\|^2,$$

其中編碼器 $W_e \in \mathbb{R}^{k\times d}$、解碼器 $W_d \in \mathbb{R}^{d\times k}$，且 $k < d$。乘積 $W_d W_e$ 的秩至多為 k，所以由 Eckart–Young 定理，最好的乘積就是投影到前 k 個主成分子空間。Baldi 與 Hornik [8] 證明了更多：對這種平方誤差的線性網路，每個局部極小值都是全域極小值，而在全域極小值處，網路投影到主成分子空間。然而個別的權重矩陣並不唯一。對任意可逆的 $k \times k$ 矩陣 G，把 W_e 換成 GW_e、W_d 換成 $W_d G^{-1}$，乘積不變，所以編碼座標可以是 PCA 分數的任何可逆線性轉換。

程式：線性自編碼器重新找到 PCA 子空間

```
import numpy as np, torch
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from scipy.linalg import subspace_angles

torch.manual_seed(0)
X = load_digits().data / 16.
X = X - X.mean(axis=0)
Xt = torch.tensor(X, dtype=torch.float32)
k = 5
enc = torch.nn.Linear(64, k, bias=False)
dec = torch.nn.Linear(k, 64, bias=False)
opt = torch.optim.Adam([*enc.parameters(), *dec.parameters()], lr=1e-2)
for step in range(3000):
    loss = (dec(enc(Xt)) - Xt).square().mean()
    opt.zero_grad(); loss.backward(); opt.step()

Wd = dec.weight.detach().numpy()            # 64 x k decoder columns
P = PCA(n_components=k).fit(X).components_.T
angles = np.degrees(subspace_angles(Wd, P))
print("largest angle to PCA subspace (deg):", round(angles.max(), 3))
print("decoder columns orthonormal?",
      np.allclose(Wd.T @ Wd, np.eye(k), atol=1e-2))
```

輸出：

```
largest angle to PCA subspace (deg): 0.003
decoder columns orthonormal? False
```

子空間夾角（兩個 k 維子空間之間的最大夾角）基本上是零：網路找到了 PCA 子空間。但它的解碼器欄向量並不正交歸一，所以它們不是主成分本身。這就是座標不確定性的實際表現，也是本章反覆出現的主題的第一個線索：光靠重建，只能確定保留了什麼，無法確定如何標記它。

非線性自編碼器

白話版。把線性映射換成神經網路，編碼就能描述任何平坦子空間都無法捕捉的彎曲結構。

一般的自編碼器有編碼器 f_ϕ 與解碼器 g_θ，兩者都是神經網路：

$$z = f_\phi(x), \qquad \hat{x} = g_\theta(z), \qquad \min_{\phi,\theta} \frac{1}{n} \sum_i \ell(x_i,\, g_\theta(f_\phi(x_i))),$$

其中 ℓ 是重建損失函數，例如實數值用平方誤差，$[0,1]$ 之間的值用二元交叉熵。ℓ 的選擇本身就是一個假設：平方誤差對應以 $\hat{x}$ 為中心的高斯概似，交叉熵對應 Bernoulli 概似。

Hinton 與 Salakhutdinov [9] 在 2006 年展示：先以逐層貪婪預訓練初始化、再端到端微調的深度自編碼器，在相同維度下找到的低維編碼，重建影像的效果比 PCA 好得多。今天，更好的初始化、正規化層與最佳化器已讓預訓練步驟變得不必要，但核心教訓依然成立：當資料是彎曲的，非線性編碼器可以比線性編碼器壓縮得更徹底。

欠完備還不夠，過完備需要規則

欠完備自編碼器（$k < d$）被迫壓縮。這很有用，但並不保證能學到好特徵。夠強大的編碼器與解碼器在原理上可以把每個訓練樣本映到它自己的任意編碼（例如它的索引），再把解碼記下來，在沒學到任何可重用結構的情況下達到很低的訓練誤差 [1, 第 14 章]。**過完備**自編碼器（$k \geq d$）則可以直接學成恆等函數。兩種情況下，讓表示有用的都不是瓶頸本身，而是歸納偏誤（inductive bias）：有限的容量、架構（例如卷積），以及明確的懲罰項。下一節介紹三種經典懲罰，合稱**正則化自編碼器**（regularized autoencoders）。

正則化自編碼器：稀疏、去噪與收縮

每一種懲罰都用不同方式回答「編碼應該忽略什麼？」這個問題。

稀疏自編碼器

白話版。允許有很多編碼單元，但要求對任何一個輸入只有少數幾個啟動。

損失函數在編碼 $h = f_\phi(x)$ 上加一個稀疏懲罰：

$$\mathcal{L} = \ell(x,\, g_\theta(h)) + \lambda\,\Omega(h), \qquad \Omega(h) = \sum_j |h_j|.$$

其他做法包括：懲罰每個單元平均啟動率偏離一個小目標值的程度，或只保留最大的 k 個啟動值（「top-k」或 k-稀疏自編碼器）。稀疏自編碼器是稀疏編碼的「攤銷式」（amortized）表親：編碼器網路一次前向傳遞就預測出稀疏編碼，而不必為每個輸入解一次 lasso。Goodfellow 等人把這個懲罰解讀為編碼上的近似先驗，而不是一般的權重正則化 [1, 第 14 章]。正如現代觀點一節會說明的，稀疏自編碼器近年已成為解讀大型神經網路內部的關鍵工具。

去噪自編碼器

白話版。給網路看一個受損的輸入，要它重建出乾淨的版本。要做到這點，它必須學會乾淨的資料長什麼樣子。

去噪自編碼器（denoising autoencoder，DAE）先抽出一個受破壞的副本 $\tilde{x} \sim C(\tilde{x} \mid x)$，再最小化

$$\mathcal{L} = \mathbb{E}_x\, \mathbb{E}_{\tilde{x}\sim C(\cdot\mid x)} \big[\, \ell(x,\, g_\theta(f_\phi(\tilde{x}))) \,\big],$$

其中 C 是事先選定的破壞過程：加性高斯雜訊、遮罩雜訊（把隨機一部分輸入設為零），或椒鹽雜訊 [10], [11]。Vincent 等人在 2008 年提出這個想法 [10]，並在 2010 年證明把 DAE 層堆疊起來，能為分類器提供有用的特徵 [11]。由於輸入與目標不同，即使編碼是過完備的，直接複製也不再是解。

破壞過程決定了模型學會對什麼具有穩健性。用高斯雜訊訓練的 DAE 沒有理由去補齊缺失的區塊，而用遮罩雜訊訓練的 DAE 則會學到補全。現代的遮罩影像建模，例如 ML 06 中的遮罩自編碼器（MAE），就是遮罩雜訊 DAE 的直系後代。

這裡還有更深一層的幾何解讀。對微小的高斯破壞，向量 $g(f(\tilde{x})) - \tilde{x}$ 會從受損的點指回資料密度較高的區域；它近似估計了**分數**（score）$\nabla_x \log p(x)$，也就是對數密度的梯度 [1, 第 14 章]。這個關聯是分數式與擴散生成模型的根源之一，詳見 ML 08。

程式：在數字上訓練去噪自編碼器

這個獨立的 PyTorch 範例在訓練用的數字上訓練一個小型非線性 DAE，並評估從加了雜訊的保留輸入重建乾淨影像的效果。

```
import torch
from torch import nn
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

torch.manual_seed(0); torch.set_num_threads(1)
train, test = train_test_split(load_digits().data/16., test_size=.2, random_state=0)
X = torch.tensor(train, dtype=torch.float32)
Xt = torch.tensor(test, dtype=torch.float32)
encoder = nn.Sequential(nn.Linear(64, 32), nn.ReLU(), nn.Linear(32, 12))
decoder = nn.Sequential(nn.Linear(12, 32), nn.ReLU(), nn.Linear(32, 64), nn.Sigmoid())
model = nn.Sequential(encoder, decoder)
opt = torch.optim.Adam(model.parameters(), lr=.003)
```

```
for step in range(2000):
    clean = X[torch.randint(len(X), (64,))]
    noisy = (clean + .2*torch.randn_like(clean)).clamp(0, 1)
    loss = (model(noisy)-clean).square().mean()
    opt.zero_grad(); loss.backward(); opt.step()
model.eval()
with torch.no_grad():
    noisy = (Xt + .2*torch.randn_like(Xt)).clamp(0, 1)
    print("noisy-input MSE:", (noisy-Xt).square().mean().item())
    print("reconstructed MSE:", (model(noisy)-Xt).square().mean().item())
```

輸出（含雜訊輸入約 0.0248，經過 DAE 後約 0.0197；最後幾位數依 PyTorch 版本而異）：

```
noisy-input MSE: 0.024781296029686928
reconstructed MSE: 0.01968955807387829
```

改善是真實的，但幅度不大，圖 4 說明了原因：在 12 維的編碼下，DAE 去除了斑點雜訊，但也把一些筆畫細節抹平了。去噪與忠實壓縮往不同方向拉扯；更寬的編碼或卷積架構會改變兩者的平衡。

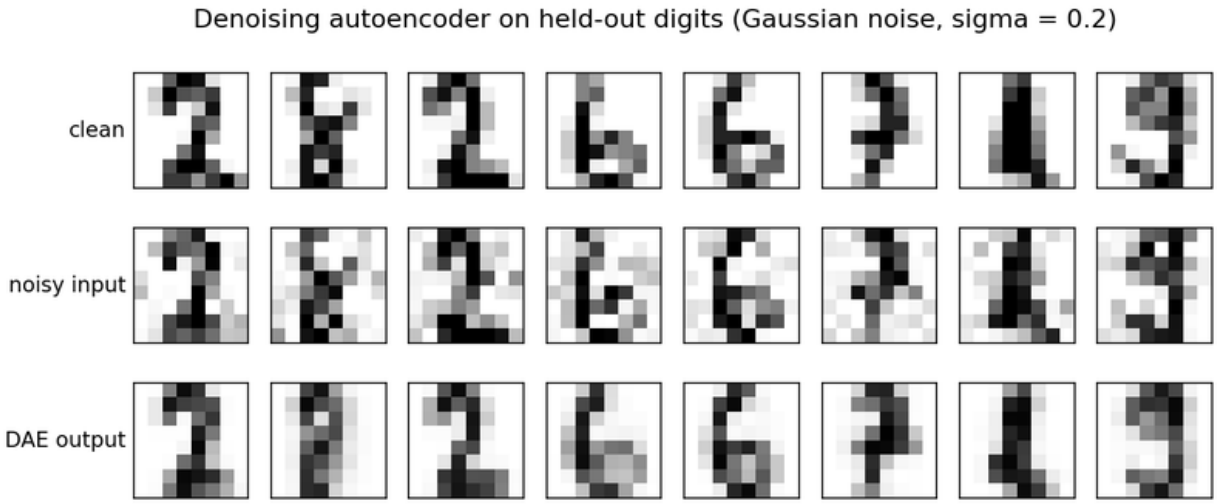


圖 *4* — 去噪自編碼器在保留數字上的輸出。背景斑點被去除；部分細微筆畫細節因 *12* 個單元的瓶頸而遺失。

收縮自編碼器

白話版。收縮懲罰抑制小輸入擾動造成的大編碼變化，但本身不知道哪些變化對任務重要。

Rifai 等人 [12] 加上編碼器 Jacobian 矩陣的 Frobenius 範數平方：

$$\mathcal{L} = \ell(x,\, g_\theta(f_\phi(x))) + \lambda \left\| \frac{\partial f_\phi(x)}{\partial x} \right\|_F^2, \qquad \left\| \frac{\partial f_\phi(x)}{\partial x} \right\|_F^2 = \sum_{j,i} \left(\frac{\partial h_j}{\partial x_i} \right)^2 .$$

其中 $h = f_\phi(x)$，Jacobian 元素 $\partial h_j / \partial x_i$ 衡量輸入座標 i 變動時，編碼單元 j 移動多少。對單層 sigmoid 編碼器 $h = \sigma(Wx + b)$，該元素為 $h_j(1 - h_j)W_{ji}$，所以懲罰項很容易計算：

$$\left\| \frac{\partial h}{\partial x} \right\|_F^2 = \sum_j \left(h_j(1 - h_j) \right)^2 \sum_i W_{ji}^2.$$

重建項希望編碼隨輸入改變（否則無法重建不同的輸入），懲罰項則希望編碼盡量少變。折衷的結果是：編碼只沿著訓練資料實際變化的少數方向保持敏感。DAE 以隱含方式達到相關的效果：對微小高斯雜訊，去噪訓練懲罰的是整個重建函數的敏感度 [1, 第 14 章]；而 CAE 則明確地只懲罰編碼器。

程式：收縮懲罰的實際效果

```
import torch
from torch import nn
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

train, test = train_test_split(load_digits().data/16., test_size=.2, random_state=0)
X = torch.tensor(train, dtype=torch.float32)
Xt = torch.tensor(test, dtype=torch.float32)

def fit(lam):
    torch.manual_seed(0)
    W = nn.Linear(64, 16)                    # encoder h = sigmoid(Wx + b)
    dec = nn.Linear(16, 64)
    opt = torch.optim.Adam([*W.parameters(), *dec.parameters()], lr=3e-3)
    for step in range(1500):
        x = X[torch.randint(len(X), (64,))]
        h = torch.sigmoid(W(x))
        rec = (dec(h) - x).square().sum(1).mean()
        # ||J||_F^2 = sum_j (h_j(1-h_j))^2 * ||W_j||^2 for a sigmoid layer
        jac = ((h*(1-h))**2 * W.weight.square().sum(1)).sum(1).mean()
        loss = rec + lam*jac
        opt.zero_grad(); loss.backward(); opt.step()
    with torch.no_grad():
        h = torch.sigmoid(W(Xt))
        mse = (dec(h) - Xt).square().mean().item()
        jac = ((h*(1-h))**2 * W.weight.square().sum(1)).sum(1).mean().item()
    return mse, jac

for lam in (0.0, 0.5):
    mse, jac = fit(lam)
    print(f"lambda={lam}: test MSE={mse:.4f}, mean ||J||_F^2={jac:.2f}")
```

輸出：

```
lambda=0.0: test MSE=0.0141, mean ||J||_F^2=4.79
lambda=0.5: test MSE=0.0194, mean ||J||_F^2=0.70
```

懲罰項把編碼器的平均敏感度降低了將近七倍，代價是重建誤差略為上升。這筆交易值不值得，取決於下游用途：較平滑的編碼通常有助於最近鄰檢索或少標籤分類，而壓縮編解碼器則會偏好未加懲罰的模型。

流形觀點

白話版。流形假說把資料建模為接近低維曲面，如房間中捲起的紙張；有用座標描述沿曲面的位置。

流形假說（manifold hypothesis）認為，影像等自然資料集中在嵌入高維輸入空間中的低維流形附近 [3]。一個隨機的 64 像素陣列幾乎不可能像一個數字；合理數字的集合是一個極小而有結構的區域。在任何資料點附近，流形局部看起來像一個平面，即**切空間**（tangent space），由少數幾個能讓你保持「真實」的移動方向張成（筆畫變粗、稍微傾斜、小幅平移）。與切空間正交的方向則會離開流形、進入雜訊。

這個觀點為本章所有內容提供了單一的視角：

- **PCA** 擬合一張全域的平面紙。流形接近平坦時有效，彎曲時就失效。
- **PPCA 或 FA 的混合模型**擬合許多張局部平面紙。
- **非線性自編碼器**學到流形的彎曲座標圖：編碼器給出紙上的座標，解碼器再把座標映回房間裡。
- **去噪**教模型把掉出流形的點移回流形上。
- **收縮**懲罰讓編碼器對垂直於流形的方向不敏感，而重建項讓它沿著切方向保持敏感 [1, 第 14 章], [12]。

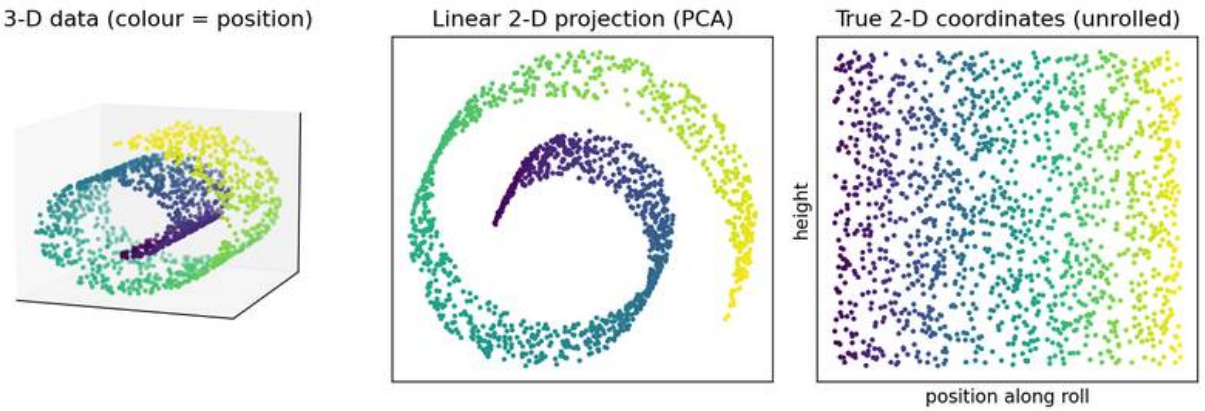


圖 *5* — 瑞士捲是一張捲在三維空間裡的二維紙。線性投影（*PCA*）把它壓成螺旋，無法保留沿紙面的位置；真實的內在座標則是一個平坦的矩形。

圖 5 讓線性方法的限制變得具體。瑞士捲有兩個內在維度——沿捲方向的位置與高度——但沒有任何二維線性投影能把它攤開：PCA 的視圖保留了螺旋，紙面上相距很遠的點並沒有依序排開。非線性編碼器原則上可以學到攤開後的座標。實務上能否做到，取決於容量、正則化與資料量；高維空間中的流形學習至今仍然困難。

從自編碼器到 VAE

白話版。一般的自編碼器學會壓縮與解壓縮，卻沒有提供憑空創造新編碼的規則。VAE 加上了這條規則。

假設訓練好的自編碼器把每個訓練用的數字映射到 $\mathbb{R}^{12}$ 中的一個編碼。為了生成新數字，我們想隨機挑一個編碼再解碼。但要挑哪個隨機編碼？訓練編碼可能占據一個奇怪、成團的區域，而它們之間的隨機點解碼後可能是一團亂。重建損失函數從來沒有要求編碼空間平滑、填滿，或符合任何已知分布。所以極佳的重建並不代表良好的生成。

變分自編碼器（variational autoencoder）[13] 的解法是回到 PPCA 與 FA 的機率因子模型故事，只是改用神經網路解碼器：

$$z \sim p(z) = \mathcal{N}(0, I), \qquad x \sim p_\theta(x \mid z),$$

其中 $p_\theta(x \mid z)$ 是一個分布，其參數由解碼器網路產生。由於真正的後驗 $p_\theta(z \mid x)$ 此時已無法計算，就由第二個網路——編碼器——輸出近似後驗 $q_\phi(z \mid x)$，通常是對角高斯分布。訓練時最大化 $\log p_\theta(x)$ 的**證據下界**（evidence lower bound，ELBO）：

$$\log p_\theta(x) \;\geq\; \underbrace{\mathbb{E}_{q_\phi(z|x)}\big[\log p_\theta(x \mid z)\big]}_{\text{reconstruction}} \;-\; \underbrace{D_{\mathrm{KL}}\big(q_\phi(z \mid x) \,\|\, p(z)\big)}_{\text{keep codes near the prior}}.$$

對照前面所有內容來讀這兩項：

- **重建**項（reconstruction）正是自編碼器的損失，只是現在在編碼中注入了雜訊。高斯解碼器下它變成平方誤差，Bernoulli 解碼器下則是二元交叉熵。
- **KL** 項（讓編碼靠近先驗）是一個正則化項，就像稀疏與收縮懲罰，但它的設計讓所有訓練樣本的編碼合起來填滿先驗分布。這正是從 $z \sim \mathcal{N}(0, I)$ 取樣之所以有意義的原因。
- 若使用**線性**高斯解碼器，這個模型就是 PPCA。它的精確後驗是高斯分布，可以被線性高斯編碼器完全匹配，所以下界變成等號，最佳解與 PPCA 的最大概似解一致。VAE 是我們一開始那些因子模型的非線性推廣。

Kingma 與 Welling 的導論 [14] 詳細發展了這個關聯。把 KL 項乘上 $\beta > 1$ 就得到 **β-VAE** [15]，它的提出是為了以重建品質為代價、鼓勵學到解耦（disentangled）的因子；請注意當 $\beta \neq 1$ 時，目標函數就不再是 ELBO。ELBO 的完整推導、重參數化技巧，以及後驗崩塌（posterior collapse）等失效模式，請見 ML 07。

重建誤差能告訴你什麼、不能告訴你什麼

離開自編碼器之前，先給一個實務上的警告。重建誤差常被當作異常分數：用正常資料訓練的自編碼器，應該能把正常輸入重建得很好、把異常輸入重建得很差。這有時行得通，但當自編碼器泛化得太好、連異常輸入也重建得不錯，或異常只存在於瓶頸本來就會丟掉的細節時，就會失敗。同樣地，重建得好的編碼可能不利於分類，而分類效果好的編碼可能丟掉了影像復原任務所需的像素細節。壓縮、去噪、下游效用與生成，請分開評估。

現代觀點

白話版。本章的古典模型並沒有消失，而是變成了零件：生成模型的壓縮階段、視覺的預訓練目標，以及觀察大型網路內部的顯微鏡。

值得一讀的綜述

- **Bengio、Courville 與 Vincent（2013）** [3] 是表示學習的標準回顧。它用一串通用先驗來界定好的表示（平滑性、多重解釋因子、概念的階層、跨任務共享的因子、流形假說、稀疏性等），並在這個框架下串起自編碼器、機率模型與流形學習。想知道本章那些懲罰項為什麼存在，它是很好的地圖。
- **Bank、Koenigstein 與 Giryes（2020）** [16] 是一篇精簡的自編碼器綜述章節。它依序介紹正則化變體（稀疏、去噪、收縮）、VAE 與解耦變體、各種應用（生成建模、分類、分群、異常偵測、推薦系統與降維），以及結合對抗式訓練的混合方法，是本篇很好的補充讀物。
- **Kingma 與 Welling（2019）** [14] 是作者親自撰寫的 VAE 延伸導論，涵蓋 ELBO、重參數化技巧、更豐富的後驗分布與階層模型，並把 VAE 放進其他深度潛在變數模型的脈絡中。
- **Goodfellow、Bengio 與 Courville** [1, 第 13–14 章] 與 **Murphy** [2] 仍是線性因子模型、DAE 與分數的關聯，以及流形詮釋最清楚的教科書說明。

解耦：一個冷靜的結果

一個反覆出現的期望是：非監督式因子模型能找回那組真正的潛在因子，每個編碼單元對應一個（「解耦」，disentanglement）。Locatello 等人 [17] 從理論上證明，若對模型與資料都沒有歸納偏誤，非監督式地學習解耦表示在根本上是不可能的，這在更大的尺度上呼應了因子分析的旋轉不確定性。他們在一個大規模、可重現的研究中也發現：沒有監督就無法辨認出解耦良好的模型，而且更高的解耦程度並不會可靠地降低下游任務的樣本複雜度。實務教訓與 ICA 相同：可識別性永遠來自額外的假設，所以請把假設講清楚。

自編碼器作為 tokenizer

自編碼器在現代最重要的角色，是大型生成模型的第一階段。**VQ-VAE** [18] 把連續編碼換成一本學習出來的離散向量碼本，使影像變成離散 token 的網格，供自迴歸模型預測（見 ML 09）。**潛在擴散**（latent diffusion）[19] 先訓練一個做感知壓縮的自編碼器，再在它的潛在空間、而不是像素空間中執行擴散模型，大幅降低運算量。兩者都受到自編碼器重建品質的上限限制：生成器無法還原編碼器已經丟掉的細節。如何設計這些 tokenizer——在壓縮、重建，以及潛在空間是否容易被生成器建模之間取得平衡——在 2026 年仍是活躍的研究領域。

用於可解釋性的稀疏自編碼器（2023–2026）

稀疏自編碼器找到了一個嶄新且非常活躍的應用：大型語言與視覺模型的**機制可解釋性**（mechanistic interpretability）。出發點是一個觀察：單一神經元往往是多義的（polysemantic），會對好幾個不相關的概念產生反應。Cunningham 等人 [20] 訓練過完備的稀疏自編碼器來重建語言模型

的內部啟動值，找到一些稀疏啟動的特徵，自動化評估認為它們比神經元或其他分解方式更具可解釋性、更單義。Gao 等人 [21] 使用 k-稀疏（top-k）自編碼器直接控制稀疏程度，報告了自編碼器大小與稀疏程度的乾淨縮放定律，提出評估特徵品質的新指標，並在 GPT-4 的啟動值上訓練了一個有 1,600 萬個潛在單元的自編碼器。概念上，這就是把稀疏編碼（Olshausen 與 Field 的想法）套用到網路的啟動值，而不是影像區塊。

開放問題

- **評估**。重建誤差、稀疏程度與下游準確率彼此不一致，而且對於學到的特徵是否「正確」——無論是解耦還是可解釋性——都沒有公認的衡量方式。
- **可識別性**。什麼時候可以相信學到的因子對應真實的成因？非線性 ICA 與因果表示學習研究的是哪些額外假設（時間結構、輔助變數、介入）能讓這件事成立。
- **Tokenizer 設計**。要壓縮多少、編碼該是離散還是連續、如何讓潛在表示既利於重建又容易建模，仍是開放的工程與理論問題。

重點整理

- 因素模型透過潛在變數解釋觀測，假設決定哪些表示可識別。
- PCA 最佳化線性重建子空間；PPCA 與因素分析加入不同雜訊模型。
- ICA、稀疏編碼與 NMF 分別利用獨立性、稀疏性及非負性，偏好不同分解。
- 平方誤差線性自編碼器在合適的全域最佳解可恢復 PCA 子空間，但權重不必是 PCA 軸。
- 瓶頸與稀疏、去雜訊或收縮目標，以不同方式約束編碼；重建品質不等於語意品質。
- VAE 加入機率式潛在模型與近似推論；線性高斯選擇使其與 PPCA 相連。

練習

1. **雜訊假設**。你記錄了 20 個感測器，其中 5 個的雜訊遠大於其他感測器，但它們全都反映同樣幾個潛在成因。你會從 PPCA 還是因子分析開始？PCA 會怎麼處理那些雜訊大的感測器？

   提示

   選因子分析：它的對角 Ψ 可以把較大的殘差變異數分配給那 5 個雜訊大的感測器，而不必為它們花費因子。PCA（以及只有單一共享 σ^2 的 PPCA）可能會把成分花在高變異數的雜訊感測器上，因為 PCA 眼中只有變異數。

2. **潛在不確定性**。對可逆的 $k \times k$ 矩陣 G，證明把 W_e 換成 GW_e、W_d 換成 $W_d G^{-1}$，線性自編碼器的重建結果不變。哪些矩陣 G 也能保持 PPCA 的概似函數不變？

   提示

   乘積為 $W_d G^{-1} G W_e = W_d W_e$。對 PPCA，概似函數取決於 $WW^\top$；把 W 換成 WG 得到 $WGG^\top W^\top$，當 $GG^\top = I$，也就是 G 為正交矩陣時不變。

3. **為什麼 ICA 對高斯來源失效**。產生兩個獨立的標準高斯來源，用任意可逆的 2×2 矩陣混合，再用不同隨機種子執行 FastICA 數次。你觀察到什麼？為什麼？

   提示

   白化之後，兩個獨立單位高斯的任何旋轉仍是兩個獨立單位高斯，所以沒有「正確」的旋轉可找。不同種子會給出不同但同樣合理的解混方向，與真實來源的相關係數每次都不一樣。

4. **稀疏編碼與 PCA 編碼**。使用稀疏編碼的程式，比較每張測試影像的非零編碼元素數量，與相同維度（32）的 PCA 編碼。接著把 `alpha` 設為 0.05、0.1、0.3、1.0，畫出稀疏編碼的測試 MSE。

   提示

   PCA 編碼是稠密的：32 個元素基本上全都非零。隨著 `alpha` 增大，啟動的原子數下降、誤差上升，畫出一條稀疏程度與精確度的取捨曲線。記得字典沒有平均值項，所以請比較曲線，而不是拿單一數字與 PCA 比。

5. **去噪的不匹配**。用高斯雜訊訓練本篇的 DAE，再拿隨機一個 3×3 區塊被設為零的輸入來測試。接著用這種遮罩破壞訓練第二個 DAE，並把兩個模型在兩種破壞下都測一遍。

   提示

   這是兩種不同的破壞過程。用高斯雜訊訓練的模型從沒被要求補齊缺失的結構，所以通常會留下空洞；用遮罩訓練的模型則學會補全。請列出一張 2×2 的表（訓練破壞 × 測試破壞），而不是從其中一種的成功推論出對另一種的穩健性。

6. **手算收縮懲罰**。對單一 sigmoid 單元 $h = \sigma(w^\top x + b)$，推導 $\|\partial h / \partial x\|^2$。它何時最大？這說明收縮自編碼器允許編碼在什麼地方保持敏感？

   提示

   $\partial h / \partial x = h(1-h)w$，所以平方範數為 $h^2(1-h)^2\|w\|^2$。它在 $h = 1/2$（sigmoid 中段最陡處）最大，單元飽和時則幾乎為零。這個懲罰把單元推向對大多數輸入都飽和的狀態，只在資料確實需要變化的地方留在敏感區間。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 13–14. Book
2. K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022. Book
3. Y. Bengio, A. Courville and P. Vincent, “Representation Learning: A Review and New Perspectives,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, 2013. Paper
4. M. E. Tipping and C. M. Bishop, “Probabilistic Principal Component Analysis,” *Journal of the Royal Statistical Society: Series B*, vol. 61, no. 3, 1999. Paper

5. A. Hyvärinen and E. Oja, “Independent component analysis: algorithms and applications,” *Neural Networks*, vol. 13, no. 4–5, 2000. Paper
6. B. A. Olshausen and D. J. Field, “Emergence of simple-cell receptive field properties by learning a sparse code for natural images,” *Nature*, vol. 381, 1996. Paper
7. D. D. Lee and H. S. Seung, “Learning the parts of objects by non-negative matrix factorization,” *Nature*, vol. 401, 1999. Paper
8. P. Baldi and K. Hornik, “Neural networks and principal component analysis: Learning from examples without local minima,” *Neural Networks*, vol. 2, no. 1, 1989. Paper
9. G. E. Hinton and R. R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks,” *Science*, vol. 313, 2006. Paper
10. P. Vincent, H. Larochelle, Y. Bengio and P.-A. Manzagol, “Extracting and composing robust features with denoising autoencoders,” ICML, 2008. Paper
11. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio and P.-A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” *JMLR*, vol. 11, 2010. Paper
12. S. Rifai, P. Vincent, X. Muller, X. Glorot and Y. Bengio, “Contractive Auto-Encoders: Explicit Invariance During Feature Extraction,” ICML, 2011. Paper
13. D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” ICLR, 2014. Paper
14. D. P. Kingma and M. Welling, “An Introduction to Variational Autoencoders,” *Foundations and Trends in Machine Learning*, vol. 12, no. 4, 2019. Paper
15. I. Higgins et al., “β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework,” ICLR, 2017. Paper
16. D. Bank, N. Koenigstein and R. Giryes, “Autoencoders,” survey chapter, arXiv:2003.05991, 2020. Survey
17. F. Locatello et al., “Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations,” ICML, 2019. Paper
18. A. van den Oord, O. Vinyals and K. Kavukcuoglu, “Neural Discrete Representation Learning,” NeurIPS, 2017. Paper
19. R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” CVPR, 2022. Paper
20. H. Cunningham, A. Ewart, L. Riggs, R. Huben and L. Sharkey, “Sparse Autoencoders Find Highly Interpretable Features in Language Models,” arXiv:2309.08600, 2023. Paper
21. L. Gao et al., “Scaling and evaluating sparse autoencoders,” arXiv:2406.04093, 2024. Paper

ML 07・生成模型：自編碼器、VAE、GAN 與正規化流

從學習標籤走向學習分布：概似式與隱式模型、ELBO 與重參數化、對抗訓練與 Wasserstein GAN、正規化流，以及生成模型該如何評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/generative-models/

你將學到

- 區分分布建模與標籤預測。
- 比較基於概似的模型與隱式取樣器。
- 推導 VAE 下界，理解重參數化。
- 說明對抗訓練與可逆密度變換。
- 分別評估概似、真實感、多樣性與條件遵循。

先看全貌

分類器問：「這張影像裡有什麼？」**生成模型（generative model）**問：「可能出現哪些影像？」如果它學到有用的分布，我們就能取樣出新的例子、補上缺失區域、壓縮資料、偵測異常輸入，或依文字、類別標籤、另一張影像來條件式地生成。

> **用最簡單的話說**
>
> 會認房子的小朋友，看到圖可以說「房子」。懂得房子怎麼畫的小朋友，則能自己畫出一棟有門、有窗、有屋頂的新房子。生成模型想學的是例子背後的「食譜」，好煮出新的菜——而且好廚師每次端出來的菜都不一樣，但都好吃，而不是同一道菜一直重複。

學分布比學標籤難得多。標籤只是一個數字；一張影像有成千上萬個像素，彼此以複雜的方式相互依賴。本章的每個家族，都是對同一個工程問題的不同回答：要怎麼表示一個極高維資料上的分布，才能既從例子**訓練**它，之後又能從它**取樣**？

本章採用 Bishop 與 Bishop [1] 以及 Prince [2] 發展的機率觀點。Bond-Taylor 等人的比較綜述 [3] 依照我們將遇到的取捨來整理這些家族——訓練速度、樣本品質、多樣性，以及是否能得到概似；Papamakarios 等人的正規化流綜述 [4] 則特別有助於區分「學到的分布」與「方便的取樣程序」。擴散模型與自迴歸模型各有專章（ML 08 與 ML 09），因為它們的訓練與取樣值得仔細討論；這裡先把它們放上地圖。

用分布描述資料

白話版。找一個機率模型，讓觀察到的例子在它之下看起來「很可能」，再用這個模型產生看起來像出自同一來源的新例子。

最大概似

假設資料 $x_1, \dots, x_n$ 是從未知分布 p_{data} 獨立抽出的樣本，p_θ 是參數為 θ 的模型。最大概似估計最小化平均**負對數概似（negative log-likelihood，NLL）**：

$$\mathcal{L}_{\text{NLL}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(x_i).$$

其中 n 是訓練樣本數，$\log p_\theta(x_i)$ 是模型給第 i 個樣本的對數密度。

為什麼這是合理的目標？當 n 變大，平均值會趨近於對資料分布的期望值，而且

$$\mathbb{E}_{x\sim p_{\text{data}}}[-\log p_\theta(x)] = D_{\text{KL}}(p_{\text{data}} \,\|\, p_\theta) + H(p_{\text{data}}).$$

$D_{\text{KL}}(p\|q) = \mathbb{E}_p[\log p - \log q]$ 是 Kullback–Leibler 散度，只有兩個分布相同時才為零；$H(p_{\text{data}})$ 是資料的熵，與 θ 無關。因此**最大概似就等於最小化從資料到模型的前向 KL 散度**。

KL 的方向很重要。前向 KL 會重罰「在真實資料所在之處給出接近零的密度」的模型，所以概似訓練是「涵蓋質量（mass-covering）」的：它寧可把機率攤到每一個模態（mode）上，即使因此在模態之間也放了一些機率。請記住這點；稍後會看到 GAN 的行為不同。

密度、離散像素與每維位元數

對連續值而言，$p_\theta(x)$ 是**密度**，不是「恰好觀察到 x」的機率；密度可以大於 1。然而真實影像存的是 0–255 這類整數。直接用連續密度去擬合整數是病態問題：模型可以在整數格點上放無限高的尖峰，讓概似趨於無窮大。標準解法是**去量化（dequantization）**——在擬合連續密度之前，對每個整數值加上 $[0, 1)$ 的均勻雜訊。Theis 等人 [5] 證明，連續模型在去量化資料上的對數概似，會是對應離散模型對數概似的下界，這讓不同論文的數字可以相互比較。另一種作法是直接使用離散概似（對 256 個值做 softmax，或像下方程式碼一樣對二值像素用 Bernoulli 分布）。

影像的概似通常以**每維位元數（bits per dimension）**回報：

$$\text{bpd} = \frac{\mathcal{L}_{\text{NLL}}}{D \ln 2},$$

其中 D 是儲存數值的個數（高 × 寬 × 通道數）。對離散模型，$\mathcal{L}_{\text{NLL}}$ 是每張影像以 nat 表示的負對數機率，bpd 對應每個儲存值的理想編碼成本。連續密度還需要交代去量化慣例：若 8 位元整數 x 加上獨立均勻雜訊 $u \in [0,1)^D$ 後縮放為 $y = (x + u)/256$，其每維位元數的去量化界限為 $8 - \mathbb{E}_u[\log p_\theta(y)]/(D \ln 2)$。額外的 8 來自縮放後量化格的體積。這是離散負對數概似的每維上界，不是精確的離散概似 [5]。比較分數前，要交代單位、雜訊分布，以及其他變數變換所需的修正。

條件式與無條件生成

判別式分類器建模 $p(y \mid x)$。條件式生成器建模 $p(x \mid c)$，其中 c 可以是文字、類別、低解析度影像或已觀察到的像素。貝氏定理把相關的量連在一起，但分類器本身並不提供影像取樣器。

條件化也解釋了一個常見的失敗。最小化像素 MSE 會預測**條件平均** $\mathbb{E}[x \mid c]$。如果一個模糊的輸入有好幾個合理輸出——例如一張灰階照片裡的襯衫可能是紅色也可能是藍色——平均值會把它們混成褐色。生成模型則嘗試表示整個分布 $p(x \mid c)$，一次取樣出一個合理的答案。

兩大家族：概似式與隱式

以下每個模型都需要兩種能力：從資料**訓練**的方法，以及**取樣**的方法。它們的差別在於能不能計算 $p_\theta(x)$。

- **概似式（顯式）模型**寫下密度或其界限，再加以最大化。自迴歸模型與正規化流給出精確概似；VAE（以及從變分角度看的擴散模型）最佳化一個下界；能量模型只定義到一個未知正規化常數為止的密度。
- **隱式模型**只定義一個取樣器——「把雜訊推過一個網路」——而沒有 $p_\theta(x)$ 的公式。GAN 是主要例子。因為沒有概似可以最大化，它們必須用別的方式比較樣本與資料來訓練。

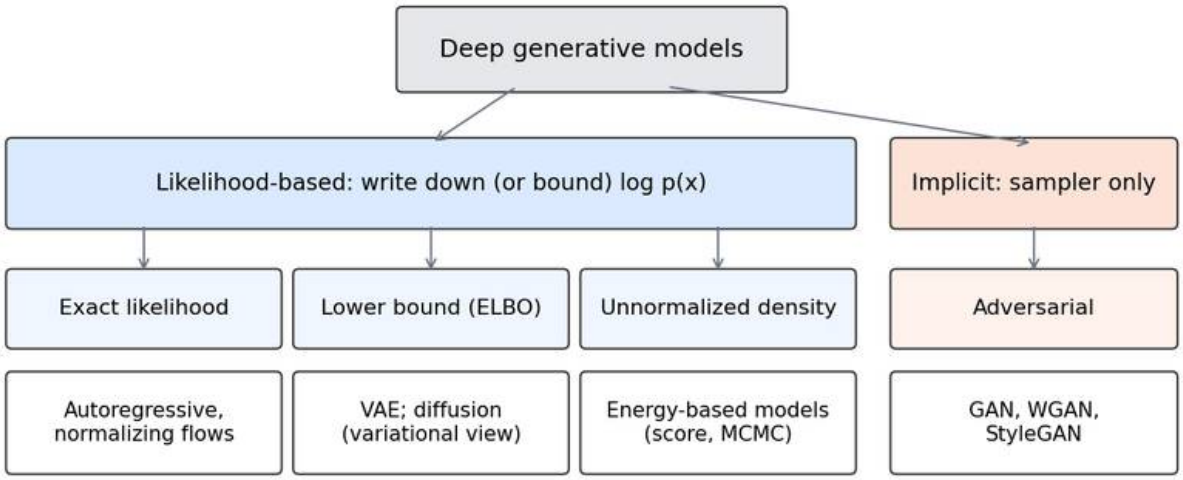


圖 *1* — 生成模型的教學用分類，採用比較綜述 [3] 的顯式／隱式劃分。實際系統常結合多個分支。

一般自編碼器：有壓縮、沒有先驗

白話版。把一個例子編碼成精簡的潛在碼，再解碼回來。這對壓縮與表示學習非常好，但它本身並不告訴你怎麼「發明」新的潛在碼。

編碼器 $z = f_\phi(x)$ 與解碼器 $\hat{x} = g_\theta(z)$ 最小化 $\|x - \hat{x}\|^2$ 之類的重建損失。自編碼器、它們的線性親戚（PCA、因子分析）及各種正則化變體，在 ML 06A 中有完整介紹。本章要強調的只有一點：一般自編碼器並不要求潛在碼**落在哪裡**。訓練資料的潛在碼可能只佔據潛在空間中一塊又薄又怪的區域。若你接著取樣 $z \sim \mathcal{N}(0, I)$ 再解碼，可能落在解碼器從沒見過的空白區域，得到一團垃圾。因此重建再好，也不保證無條件生成的品質。

VAE 透過「給潛在碼一個先驗分布」，把自編碼器變成機率模型來解決這個問題。

VAE：潛在變數機率模型

白話版。 VAE 透過隨機潛在編碼與解碼器建模觀測；編碼器近似各觀測對應的編碼後驗。

模型

選一個潛在向量 $z \in \mathbb{R}^k$ 的先驗 $p(z)$，通常是標準高斯 $\mathcal{N}(0, I)$，以及解碼器 $p_\theta(x \mid z)$——一個輸出影像分布參數的神經網路（二值像素用 Bernoulli 機率，或高斯平均值）。模型對影像的密度是邊際分布

$$p_\theta(x) = \int p_\theta(x \mid z)\, p(z)\, dz.$$

取樣很容易：先抽 $z \sim p(z)$，再抽 $x \sim p_\theta(x \mid z)$。計算 $p_\theta(x)$ 則很難：積分要跑過所有可能的潛在碼，而解碼器是非線性網路，沒有封閉解。真正的後驗 $p_\theta(z \mid x) = p_\theta(x \mid z)p(z)/p_\theta(x)$ 也因此無法計算。

推導證據下界

Kingma 與 Welling [6] 引入一個近似後驗的**編碼器**（或稱推論網路）$q_\phi(z \mid x)$。以下逐步推導。因為 $\log p_\theta(x)$ 與 z 無關，我們可以對任何 z 的分布取期望而不改變它的值：

$$\begin{aligned}
\log p_\theta(x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x)] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{p_\theta(z \mid x)}\right] \\
&= \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z \mid x)}\right] + \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\right] \\
&= \mathrm{ELBO}(x) + D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)).
\end{aligned}$$

第二行用了 $p_\theta(x) = p_\theta(x,z)/p_\theta(z \mid x)$；第三行在對數裡同乘、同除 $q_\phi(z \mid x)$。因為 KL 散度永遠不為負，第一項是 $\log p_\theta(x)$ 的下界：**證據下界（evidence lower bound，ELBO）**。下界與真實對數概似之間的**差距**，恰好是近似後驗與真實後驗之間的 KL 散度。編碼器越好，下界越緊；完美的編碼器會讓差距歸零。

把 $p_\theta(x,z) = p_\theta(x \mid z)p(z)$ 展開，得到程式中使用的形式：

$$\mathrm{ELBO}(x) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z))}_{\text{stay close to the prior}}.$$

第一項（重建）獎勵能讓解碼器重現 x 的潛在碼；第二項（貼近先驗）懲罰分布偏離先驗太遠的潛在碼。用 Jensen 不等式（$\log \mathbb{E} \geq \mathbb{E} \log$）一行就能得到同樣的下界，但上面的分解更有用，因為它明確指出了差距是什麼。訓練時對 θ 與 ϕ 同時最大化資料上的平均 ELBO，也就是最小化**負 ELBO**。

當編碼器是對角高斯 $q_\phi = \mathcal{N}(\mu, \mathrm{diag}(\sigma^2))$、先驗是標準高斯時，KL 項有封閉解：

$$D_{\mathrm{KL}}(q_\phi \| p) = \frac{1}{2} \sum_{j=1}^{k} \left(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1 \right),$$

其中 μ_j 與 σ_j^2 是編碼器在第 j 個潛在座標上的平均值與變異數。每一項只有在 $\mu_j = 0$ 且 $\sigma_j^2 = 1$ 時才為零。

重參數化

重建項是對 $z \sim q_\phi(z \mid x)$ 的期望值，而我們需要它對 ϕ 的梯度。照字面寫的取樣無法微分：隨機抽樣是個黑盒子。**重參數化技巧（reparameterization trick）** [6] 把隨機性移到一個固定的雜訊變數上：

$$\epsilon \sim \mathcal{N}(0, I), \qquad z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon.$$

$\odot$ 是逐元素相乘。現在 z 是 ϕ 與獨立雜訊 ϵ 的確定性、可微函數，一般的反向傳播就能透過 μ 與 σ 給出低變異的梯度估計。程式中編碼器預測 `logvar` $= \log \sigma^2$，再用 `exp(.5 * logvar)` 當標準差，不需任何限制就能保證為正。

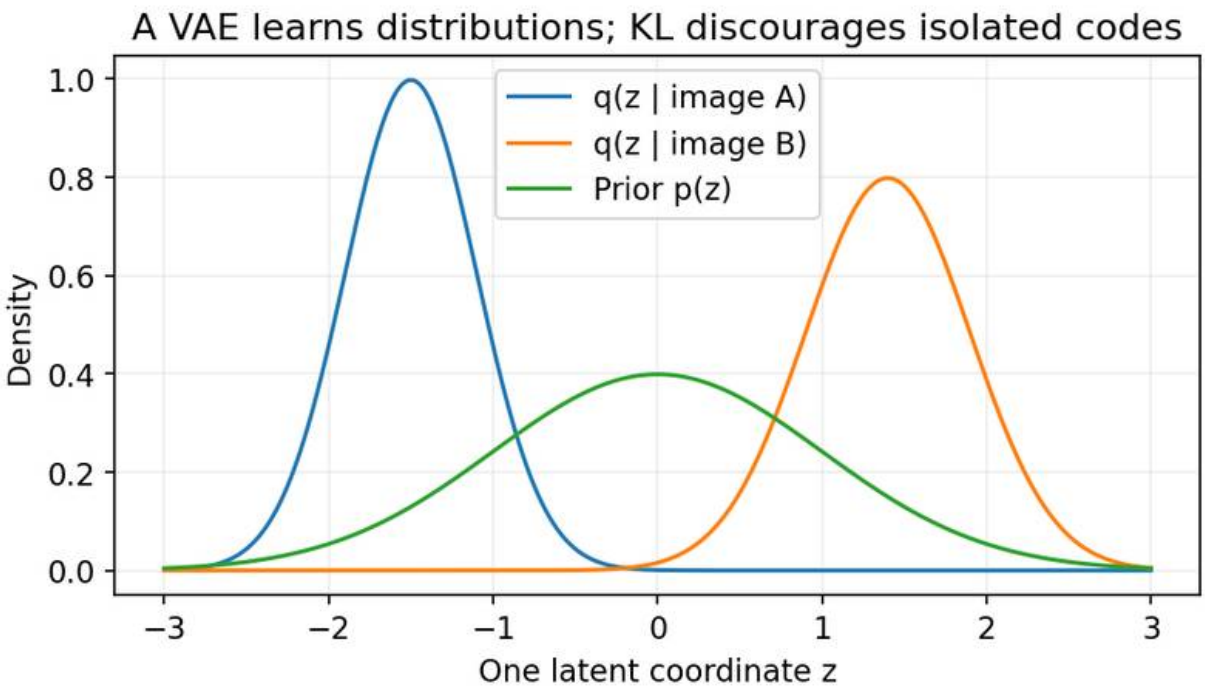


圖 2 — *VAE* 以分布表示潛在碼的不確定性，並鼓勵它與共用的先驗相容。曲線為示意。

程式：二值手寫數字的小型 VAE

這個完整的 CPU 範例使用 scikit-learn 內建的 8×8 手寫數字，並先二值化，讓 Bernoulli 解碼器成為明確的概似模型。重建項對像素加總、KL 對潛在維度加總，然後兩者都對樣本取平均；若只偷偷對其中一項取平均，就會改變兩者的比重。

```
import torch
from torch import nn
from torch.nn import functional as F
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

torch.manual_seed(0)
torch.set_num_threads(1)
d = load_digits()
train, test = train_test_split(d.data, test_size=.2, random_state=0,
                               stratify=d.target)
X = torch.tensor((train > 8).astype("float32"))
Xt = torch.tensor((test > 8).astype("float32"))
enc = nn.Sequential(nn.Linear(64, 64), nn.ReLU(), nn.Linear(64, 16))
dec = nn.Sequential(nn.Linear(8, 64), nn.ReLU(), nn.Linear(64, 64))
opt = torch.optim.Adam(list(enc.parameters()) + list(dec.parameters()), lr=.002)

def terms(x):
    mu, logvar = enc(x).chunk(2, dim=-1)
    z = mu + (.5*logvar).exp() * torch.randn_like(mu)
    recon = F.binary_cross_entropy_with_logits(dec(z), x, reduction="none").sum(1)
    kl = .5*(mu.square() + logvar.exp() - logvar - 1).sum(1)
    return recon, kl

for step in range(400):
    x = X[torch.randint(len(X), (64,))]
    recon, kl = terms(x)
    loss = (recon + kl).mean()
    opt.zero_grad()
    loss.backward()
    opt.step()
enc.eval(); dec.eval()
with torch.no_grad():
    recon, kl = terms(Xt)
    samples = dec(torch.randn(4, 8)).sigmoid().reshape(4, 8, 8)
    print("test negative ELBO estimate:", round((recon + kl).mean().item(), 3))
    print("reconstruction / KL:", round(recon.mean().item(), 3),
          round(kl.mean().item(), 3))
    print("sample probability maps:", tuple(samples.shape))
```

在我們的機器上，它印出測試集負 ELBO 約為每張影像 20.5 nats，其中重建約 17.0 nats、KL 約 3.5 nats。因為 ELBO 是對數概似的下界，20.5 nats 就是測試 NLL 的**上界**。輸出是 Bernoulli 機率圖；若要得到二值樣本，就依這些機率對每個像素取樣。測試估計帶有隨機性，多取幾次潛在樣本再平均會更穩定。這裡沒有用測試集挑選檢查點。

圖 3 顯示同樣資料上、潛在空間只有**兩維**的 VAE（由圖檔腳本訓練更久）學到了什麼。即使從沒

用到標籤，相同類別的數字仍落在彼此附近；對一個規則網格上的 z 解碼，數字形狀會平滑地漸變。這正是一般自編碼器缺少的性質：先驗主體附近的每一點，解碼出來都像個數字。

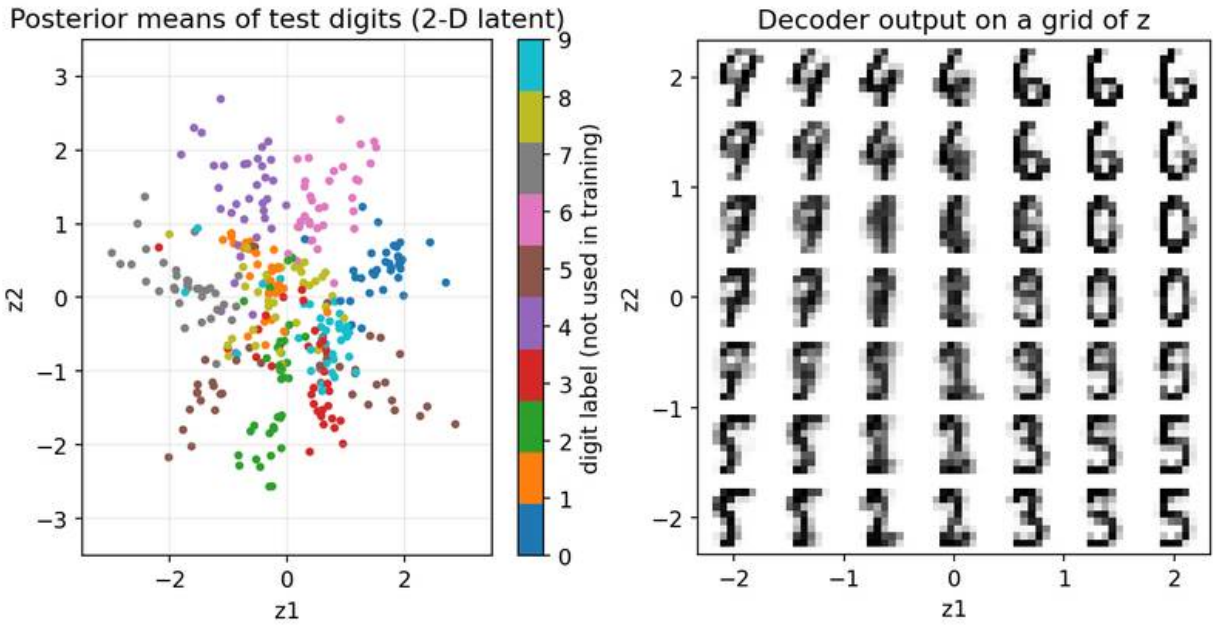


圖 3 — 在二值化 8×8 數字上訓練、潛在空間為 2 維的 *VAE*。左：測試影像的編碼器平均值（標籤只用於著色）。右：在潛在值網格上的解碼器輸出。

後驗崩塌、β-VAE 與更深的 VAE

如果解碼器非常強大（例如本身就能建模像素的自迴歸解碼器），提高 ELBO 最便宜的方式可能是完全忽略 z：令 $q_\phi(z \mid x) = p(z)$ 使 KL 項為零，讓解碼器包辦一切。這就是**後驗崩塌（posterior collapse）**。Bowman 等人 [7] 在文字 VAE 上觀察到這個現象，並提出 **KL 退火（KL annealing）**：一開始給 KL 項很小的權重，訓練中逐步升到 1，讓模型在懲罰生效前先學會使用潛在碼。監看每個潛在維度的 KL 是最簡單的診斷——KL 接近零的維度不攜帶任何資訊。

Higgins 等人 [8] 則反其道而行：**β-VAE** 把 KL 項乘上 $\beta > 1$，以犧牲重建為代價，迫使模型學出更分解、更「解纏（disentangled）」的潛在碼。要精確理解這件事：當 $\beta \neq 1$，目標函數就不再是 ELBO，它的值也不再是 $\log p_\theta(x)$ 的下界；Kingma 與 Welling 的教學綜述 [9] 討論了這點以及許多其他變體。最後，單層、因子化高斯的 VAE 容易產生模糊影像，部分原因是高斯解碼器會對不確定性取平均。**階層式 VAE** 疊了好幾層潛在變數；NVAE [10] 顯示，經過精心設計、搭配穩定化正則化的深層階層，可以在高達 256×256 像素的自然影像上產生清晰樣本。

僅線上版：*VAE*：透過可計算的下界學習 — *https://blog.shyandram.dev/zh-tw/learn/ml/generative-models/papers/vae 透過可計算的下界學習/*

GAN：與對手一起學習

白話版。生成器學習欺騙區分生成樣本與資料的判別器；這種競爭目標能改善樣本，但訓練可能不穩定。

極小極大遊戲

令 G 把雜訊 $z \sim p(z)$ 映到樣本 $G(z)$，其分布記為 p_g。令 $D(x) \in (0,1)$ 是判別器估計 x 為真的機率。Goodfellow 等人 [11] 的原始 GAN 目標是

$$\min_G \max_D \; V(D,G) = \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))].$$

判別器最大化 V：這就是一般的二元交叉熵，真資料標 1、假資料標 0。生成器最小化 V：它希望 $D(G(z))$ 越大越好。

這個遊戲在最小化什麼

固定生成器，把 V 寫成對 x 的積分：$V = \int [p_{\text{data}}(x)\log D(x) + p_g(x)\log(1 - D(x))]dx$。對每個 x 分別來看，當 $a, b > 0$，函數 $a\log D + b\log(1 - D)$ 在 $D = a/(a + b)$ 有最大值（令導數 $a/D - b/(1 - D)$ 為零即可）。所以**最佳判別器**是

$$D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}.$$

它就是「若兩者事前機率相同，一個點來自資料而非生成器」的機率。把 D^* 代回並加減 $\log 4$，得到

$$V(D^*, G) = -\log 4 + 2\,\text{JSD}(p_{\text{data}} \,\|\, p_g),$$

其中 $\text{JSD}(p\|q) = \frac{1}{2}D_{\text{KL}}(p\|m) + \frac{1}{2}D_{\text{KL}}(q\|m)$、$m = \frac{1}{2}(p + q)$，稱為 **Jensen–Shannon 散度**。因此在完美判別器之下，生成器是在最小化資料與模型間的一個對稱散度，全域最佳解是 $p_g = p_{\text{data}}$，此時處處 $D^* = 1/2$ [11]。注意整個過程從沒計算過概似：判別器扮演一個學出來、會自我調整的「這看起來像資料嗎？」檢定。

非飽和損失與實作方式

訓練初期生成器很差，$D(G(z)) \approx 0$，$\log(1 - D(G(z)))$ 幾乎是平的，生成器的梯度消失。實務上生成器改為最小化 $-\mathbb{E}\log D(G(z))$，也就是同一篇論文 [11] 提出的**非飽和（non-saturating）**損失：正好在生成器落後時給出強梯度。它的固定點相同，但動態不同。

兩者交替更新。更新 D 時要把生成樣本 **detach**，讓梯度不會流到 G；更新 G 時讓梯度穿過 D 流進 G，但只執行 G 的最佳化器。用 logits 搭配 `binary_cross_entropy_with_logits`，在數值上比先套 sigmoid 再取 log 更安全。

程式：在八個高斯上訓練 GAN

測試模態涵蓋的經典場景，是二維平面上排成一圈的八個緊密高斯群。它小到能在 CPU 上一分鐘內訓練完，而且你可以**親眼看到**每個模態有沒有被涵蓋。

```
import math
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
angles = torch.arange(8) * (2 * math.pi / 8)
centers = 2.0 * torch.stack([angles.cos(), angles.sin()], dim=1)

def sample_data(n):
    idx = torch.randint(8, (n,))
    return centers[idx] + 0.05 * torch.randn(n, 2)

def mlp(i, o):
    return nn.Sequential(nn.Linear(i, 128), nn.ReLU(),
                         nn.Linear(128, 128), nn.ReLU(), nn.Linear(128, o))

G, D = mlp(8, 2), mlp(2, 1)
opt_g = torch.optim.Adam(G.parameters(), lr=1e-3, betas=(.5, .999))
opt_d = torch.optim.Adam(D.parameters(), lr=1e-3, betas=(.5, .999))
for step in range(3000):
    real = sample_data(256)
    fake = G(torch.randn(256, 8))
    # Discriminator: logits, real -> 1, fake -> 0; detach so G is untouched.
    d_loss = (F.binary_cross_entropy_with_logits(D(real), torch.ones(256, 1))
              + F.binary_cross_entropy_with_logits(D(fake.detach()),
                                                   torch.zeros(256, 1)))
    opt_d.zero_grad(); d_loss.backward(); opt_d.step()
    # Generator: non-saturating loss -log D(G(z)).
    g_loss = F.binary_cross_entropy_with_logits(D(fake), torch.ones(256, 1))
    opt_g.zero_grad(); g_loss.backward(); opt_g.step()

with torch.no_grad():
    x = G(torch.randn(5000, 8))
    dist = torch.cdist(x, centers)
    nearest = dist.min(1)
    good = nearest.values < 0.15          # within 3 std of some mode
    counts = torch.bincount(nearest.indices[good], minlength=8)
print("fraction near a mode:", round(good.float().mean().item(), 3))
print("modes with >= 1% of samples:", int((counts >= 50).sum()))
print("samples per mode:", counts.tolist())
```

輸出為：

```
fraction near a mode: 0.896
modes with >= 1% of samples: 7
```

```
samples per mode: [590, 14, 710, 276, 651, 731, 863, 646]
```

約 90% 的樣本落在某個模態上，但 45° 方向的模態在 5,000 個樣本中只分到 14 個，另一個模態也明顯偏少。這就是當場抓到的**部分模式崩塌**。圖 4 呈現生成器的演變：早期是一團沒有結構的點雲，接著是一個涵蓋所有模態、但也涵蓋模態之間空隙的圓環，最後是更銳利的群聚，卻幾乎放棄了一個模態。精確數字會隨亂數種子與套件版本改變，但這個定性現象很典型。

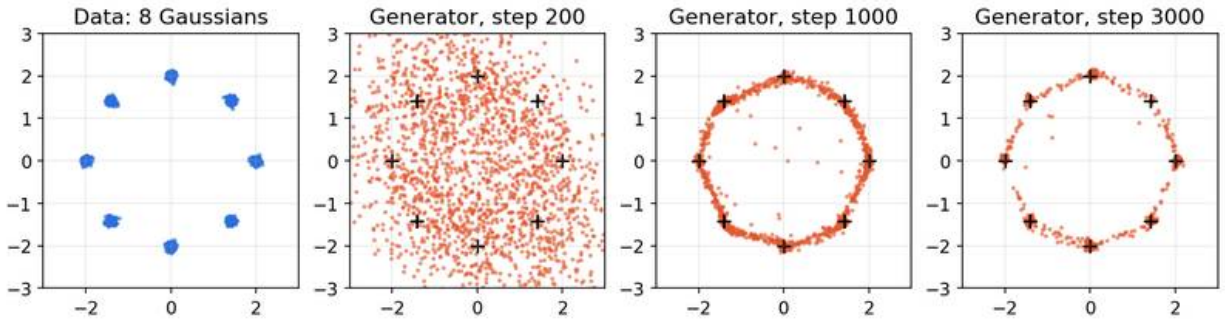


圖 *4* — 小型 *GAN* 學習八個高斯組成的圓環（十字標記真實模態）。到第 *3000* 步時樣本很清晰，但有一個模態幾乎被丟掉：清晰度與涵蓋度是兩回事。

模式崩塌及其成因

當生成器產生的輸出種類太少，就發生了**模式崩塌**。生成器得到的獎勵來自「目前的判別器接受的樣本」，而不是多樣性。如果它找到一塊判別器喜歡的區域，就可以把樣本都堆在那裡；判別器接著學會拒絕那塊區域，生成器再跳到另一塊，雙方可能互相追逐而不收斂。與最大概似不同，生成器的逐樣本損失中，沒有任何一項會直接懲罰**漏掉**某個模態。Salimans 等人 [12] 提出了特徵匹配（feature matching）與小批次判別（minibatch discrimination）等早期對策，讓判別器看整批樣本的統計量，於是多樣性不足就能被察覺。

所以一組清晰、逼真的影像，仍可能只涵蓋資料分布的一小部分。品質與涵蓋度一定要分開量測——我們會在評估一節回到這點。

Wasserstein GAN

當 p_{data} 與 p_g 分布在又薄、又互不重疊的集合上（影像在像素空間中正是如此），JSD 會卡在最大值 $\log 2$，無法提供「兩者**相距多遠**」的有用梯度。Arjovsky 等人 [13] 改用 **Wasserstein-1（推土機）距離**：把一個分布的機率質量搬到另一個分布的最小成本。根據 Kantorovich–Rubinstein 定理，它的對偶形式是

$$W(p_{\text{data}}, p_g) = \sup_{\|f\|_L \le 1} \mathbb{E}_{x\sim p_{\text{data}}}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)],$$

上確界取遍所有 1-Lipschitz 函數 f（輸入從 x' 變到 x 時，輸出變化不超過 $\|x - x'\|$）。網路 f 這時改稱為**評論器（critic）**：它輸出無上下界的分數，而不是機率。即使兩個支撐集不重疊，生成器樣本越靠近資料，Wasserstein 距離也會平滑地縮小。

難處在於如何落實 Lipschitz 限制。原始 WGAN 把評論器的每個權重裁剪到一個小區間，作法粗糙，可能讓評論器過於簡單。**WGAN-GP** [14] 改為加上梯度懲罰

$$\lambda\,\mathbb{E}_{\hat{x}}\big[(\|\nabla_{\hat{x}} f(\hat{x})\|_2 - 1)^2\big],$$

其中 $\hat{x}$ 是在真實樣本與生成樣本配對之間的直線上均勻抽取，λ 是權重（論文使用 10）。另外兩個廣泛使用的穩定技巧直接作用在判別器上：**頻譜正規化（spectral normalization）** [15] 把每個權重矩陣除以它最大的奇異值，逐層限制 Lipschitz 常數；Mescheder 等人 [16] 分析的 **R1 懲罰**則在真實資料上懲罰 $\frac{\gamma}{2}\|\nabla_x D(x)\|^2$，他們證明這能帶來局部收斂，而未正則化的 GAN 訓練可能會震盪。

架構，以及 GAN 現在用在哪裡

StyleGAN [17] 說明架構與損失同樣重要：它把雜訊映到一個中介潛在空間，再把這個「風格」注入每一個解析度，得到可控制、高品質的人臉合成。GAN 只需一次生成器前向傳遞就能取樣，推論很快。對抗損失也常作為其他系統的**元件**出現——例如在影像 tokenizer 中用來重建清晰細節。但這不代表每個使用這種 tokenizer 的模型都是 GAN 生成器；分布可能完全由別的東西建模。

僅線上版：*GAN：分布賽局與理想條件下的保證* — *https://blog.shyandram.dev/zh-tw/learn/ml/generative-models/papers/gan 分布賽局與理想條件下的保證/*

正規化流：可逆轉換

白話版。正規化流透過可逆映射變換簡單密度；追蹤各映射的體積變化，即可計算模型密度。

變數變換

令 $z \sim p_0(z)$ 是簡單的基底分布，$x = f_\theta(z)$ 是同維度、可逆且可微的映射。則

$$\log p_\theta(x) = \log p_0\big(f_\theta^{-1}(x)\big) - \log\left|\det \frac{\partial f_\theta}{\partial z}\right|_{z=f_\theta^{-1}(x)}.$$

$\partial f_\theta/\partial z$ 是映射的 Jacobian 矩陣，它的行列式衡量映射把 z 附近的小體積放大了多少。機率質量守恆，所以若一塊區域被拉伸成兩倍體積，它的密度就必須減半。在一維中，$x = 2z$ 讓每個區間加倍，對數密度加上 $-\log 2$。組合映射 $f = f_K \circ \cdots \circ f_1$ 只要把各步的對數行列式加起來。

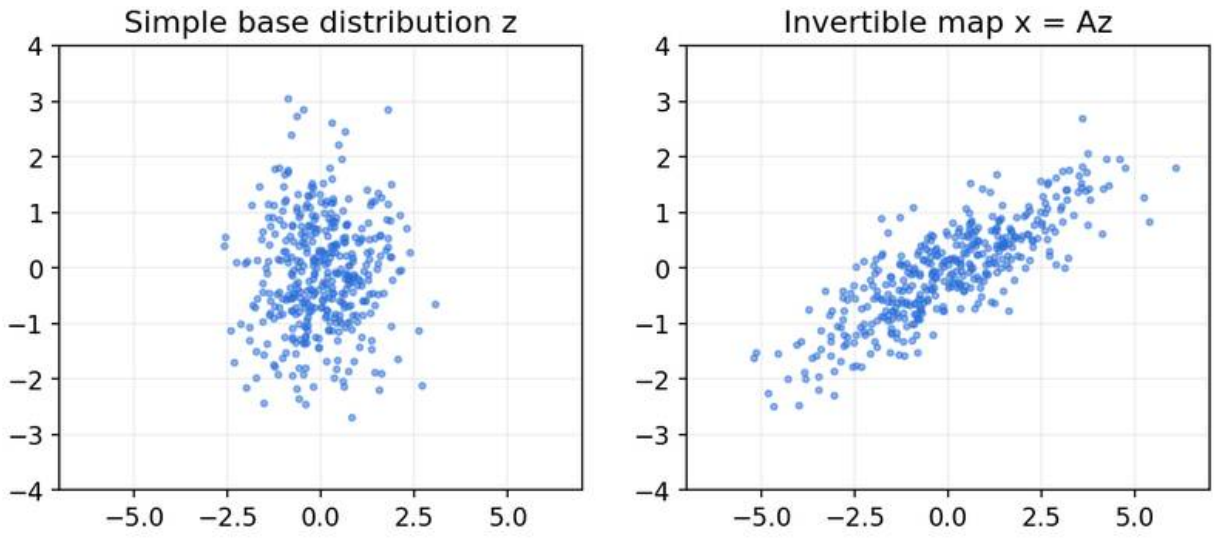


圖 5 — 變數變換保持總機率不變，同時改變密度。這個仿射映射的行列式可以解析計算。

耦合層

公式是精確的，但一般 $D \times D$ 行列式的成本是 $O(D^3)$——對影像來說毫無希望。流的層因此被設計成 Jacobian 為三**角矩陣**，行列式就只是對角線的乘積。RealNVP [18] 的仿射耦合層把 x 分成兩半 (x_1, x_2)，計算

$$y_1 = x_1, \qquad y_2 = x_2 \odot \exp\big(s(x_1)\big) + t(x_1),$$

其中 s 與 t 是任意神經網路。Jacobian 是區塊三角矩陣：第一塊是單位矩陣，第二塊是 $\operatorname{diag}(\exp s(x_1))$，所以 $\log|\det J| = \sum_j s_j(x_1)$。反函數一樣便宜：$x_2 = (y_2 - t(y_1)) \odot \exp(-s(y_1))$。關鍵在於 s 與 t 永遠不需要求逆，所以可以任意複雜。交替切分方式堆疊多層，就能讓每個座標都被轉換。Glow [19] 則在耦合層之間加入學習得到的可逆 1×1 卷積，作為廣義的通道排列。

程式：驗證耦合層的對數行列式

下面的程式建立一個仿射耦合層，並把它便宜算出的對數行列式，與 autograd 算出的完整 Jacobian 做比對。

```
import torch
from torch import nn

torch.manual_seed(0)

class AffineCoupling(nn.Module):
    """RealNVP-style layer: keep x1, scale-and-shift x2 using x1."""
    def __init__(self, dim, hidden=32):
        super().__init__()
```

```
        self.d = dim // 2
        self.net = nn.Sequential(nn.Linear(self.d, hidden), nn.Tanh(),
                                 nn.Linear(hidden, 2 * (dim - self.d)))

    def forward(self, x):
        x1, x2 = x[..., :self.d], x[..., self.d:]
        s, t = self.net(x1).chunk(2, dim=-1)
        s = torch.tanh(s)                       # keep scales moderate
        y2 = x2 * s.exp() + t
        return torch.cat([x1, y2], -1), s.sum(-1)   # output, log|det J|

    def inverse(self, y):
        y1, y2 = y[..., :self.d], y[..., self.d:]
        s, t = self.net(y1).chunk(2, dim=-1)
        s = torch.tanh(s)
        return torch.cat([y1, (y2 - t) * (-s).exp()], -1)

layer = AffineCoupling(4)
x = torch.randn(4)
y, logdet = layer(x)
J = torch.autograd.functional.jacobian(lambda v: layer(v)[0], x)
print("log|det J| from layer:   ", round(logdet.item(), 5))
print("log|det J| from autograd:", round(torch.linalg.slogdet(J)[1].item(), 5))
print("upper-right block is zero:", bool(torch.allclose(J[:2, 2:], torch.zeros(2, 2))))
print("inverse error:", (layer.inverse(y) - x).abs().max().item() < 1e-6)

# Log density of y under a standard-normal base after one layer:
base = torch.distributions.Normal(0., 1.)
z = layer.inverse(y)
log_px = base.log_prob(z).sum() - layer(z)[1]
print("log p(y):", round(log_px.item(), 4))
```

兩個對數行列式都印出 0.14867，Jacobian 右上區塊恰好為零（三角結構），反函數也還原了輸入。最後一行就是變數變換公式的實際運作。訓練一個流模型，就是用一疊這樣的層，在資料上最大化這個 `log p`。

優點、限制與連續流

流模型提供精確概似、精確反演（適合影像編輯，也能把資料映到性質良好的潛在空間），以及一次前向傳遞的取樣。代價在於架構：每一層都必須可逆且行列式便宜，而且潛在空間必須與資料**同維度**，這會把容量浪費在表示所有像素方向上的雜訊。Rezende 與 Mohamed [20] 最初提出流，是為了讓 VAE 的近似後驗更有彈性，這說明同一個工具既能用於推論，也能用於生成。

連續正規化流把離散的層疊換成常微分方程 $\mathrm{d}x/\mathrm{d}t = v_\theta(x, t)$。Neural ODE [21] 指出，這時對數密度是透過 v_θ 的**散度**（Jacobian 的跡）而非行列式來演化，於是解除了架構限制，但計算概似需要解一次 ODE。ML 08 介紹的**流匹配（flow matching）** [22]，則用簡單的迴歸而非最大概似來訓練這種向量場。所以「流」這個詞指的是相關的轉換，但訓練目標不一定是概似。

能量模型

白話版。與其描述怎麼產生影像，不如給每一張可能的影像打一個分數——「能量」——能量越低代表越「像真的」。能量低的地方，機率就高。

能量模型（energy-based model，EBM）定義

$$p_\theta(x) = \frac{\exp(-E_\theta(x))}{Z_\theta}, \qquad Z_\theta = \int \exp(-E_\theta(x))\, dx,$$

其中 E_θ 是任何輸出純量的網路，Z_θ 是讓密度積分為一的**配分函數（partition function）**。EBM 極有彈性——任何架構都行——但在高維中 Z_θ 無法計算，所以概似與精確樣本都無法直接取得。Goodfellow、Bengio 與 Courville [23] 涵蓋了經典工具（配分函數與近似推論等章節），Song 與 Kingma [24] 則整理了現代的訓練途徑：搭配 MCMC 樣本（例如 Langevin 動力學）的最大概似；**分數匹配（score matching）**，它擬合 $\nabla_x \log p_\theta(x) = -\nabla_x E_\theta(x)$，因而完全不需要 Z_θ；以及雜訊對比類方法。分數匹配正是通往擴散模型的橋樑。

自迴歸模型（簡介）

白話版。一次產生影像的一小塊，每一塊都以目前已產生的全部內容為條件，就像一個字一個字地寫句子。

對 x 的 D 個元素採用任何順序，機率的連鎖律都是精確的：

$$p_\theta(x) = \prod_{i=1}^{D} p_\theta(x_i \mid x_1, \ldots, x_{i-1}).$$

每個因子都是一般的分類或迴歸問題，所以訓練就是簡單、穩定、具精確概似的最大概似。PixelRNN 與 PixelCNN [25] 就是這樣對原始像素建模，每個子像素用一個 256 類的 softmax。缺點是取樣必須循序進行：D 個元素要 D 次網路計算。現代系統改為預測壓縮後的 token、一組組的 token 或整個尺度。ML 09 會詳細介紹這個家族。

擴散模型（簡介）

白話版。教一個網路從帶雜訊的影像中去掉一點點雜訊。然後從純雜訊出發，反覆套用這個去雜訊器，慢慢雕出一張影像。

擴散模型定義一個固定的前向過程，逐步對資料加入高斯雜訊，直到只剩雜訊；再學習一個逐步還原的反向過程。Ho 等人 [26] 證明，訓練（在權重之外）可以化約成預測被加入的雜訊——一個簡單的迴歸——而且這個目標是重新加權的變分下界，這也是圖 1 把擴散模型放在 VAE 旁邊的原因。與分數匹配的連結（見能量模型一節）提供了第二個等價觀點。Yang 等人的綜述 [27] 整理了眾多變體。擴散模型與流匹配如今主導高品質影像生成，是 ML 08 的主題。

家族地圖

家族	基本訓練訊號	取樣機制	概似	主要限制
VAE	變分下界（ELBO）	取樣潛在變數再解碼（一次傳遞）	下界	下界取決於近似推論；樣本可能模糊
GAN	對抗判別	生成器前向傳遞（一次）	無	遊戲不穩定；模式崩塌
正規化流	變數變換概似	可逆轉換（一次）	精確	可逆性與同維度限制
自迴歸	條件概似	循序條件取樣	精確	生成時的相依性可能代價高
擴散	去雜訊／分數目標	反向雜訊過程（多步）	下界或基於 ODE	品質取決於模型與取樣器
流匹配	速度場迴歸	對學到的向量場積分	需額外計算	積分成本
能量模型	能量學習（MCMC、分數匹配）	常需迭代取樣	未正規化	正規化與混合可能困難

Bond-Taylor 等人 [3] 把這些取捨總結成一個很難同時滿足的三角：快速取樣、高樣本品質、良好的模態涵蓋。GAN 快又清晰，但可能漏掉模態；概似式模型涵蓋模態，但過去較慢或較模糊；擴散模型以多步取樣為代價，達到高品質與高涵蓋。這張表是教學用的整理，不是互斥的分類。實際模型常結合多個家族：VAE 壓縮影像、Transformer 建模它的潛在分布，再用擴散目標訓練那個 Transformer。

評估生成模型

白話版。生成器可以在不同面向上「好」：給真實資料高機率、產生個別看來逼真的樣本、涵蓋資料中的所有多樣性，或遵循它的條件。沒有任何單一數字能涵蓋全部，所以要用多個指標，並清楚每個指標漏掉了什麼。

Theis 等人 [5] 把這點講得很透徹：在高維中，平均對數概似、樣本品質，以及 Parzen 窗估計之類的舊代理指標，彼此可以**大致獨立**。模型可能概似很好但樣本很差，反之亦然。請依你實際的用途來評估。

概似

對概似式模型而言，保留資料上的 NLL（或每維位元數）有理論根據，在去量化與前處理一致時也可以相互比較。GAN 則無法計算它。而且高概似不代表能可靠地偵測分布外資料：Nalisnick 等人 [28] 發現，在 CIFAR-10 上訓練的流模型、VAE 與 PixelCNN，可能給不相關的 SVHN 門牌數字影像**更高**的概似，因為概似主要受平滑度這類低階統計量主導，而非語意內容。

Inception Score

Salimans 等人 [12] 提出 **Inception Score（IS）**。把每張生成影像 x 送進預訓練的 ImageNet 分類器（Inception-v3），得到標籤分布 $p(y \mid x)$，並令 $p(y) = \mathbb{E}_x[p(y \mid x)]$ 為對樣本的平均。則

$$\mathrm{IS} = \exp\Big(\mathbb{E}_{x \sim p_g} D_{\mathrm{KL}}(p(y \mid x) \,\|\, p(y))\Big).$$

當每張影像都被有信心地分類（$p(y \mid x)$ 尖銳）、整組樣本又涵蓋許多類別（$p(y)$ 分散）時，IS 就高。它從不看真實資料，所以無法判斷樣本是否像**你的**資料集，而且只對類似 ImageNet 的內容有意義。

Fréchet Inception Distance

Heusel 等人 [29] 提出的 **Fréchet Inception Distance（FID）**在特徵空間中比較真實與生成影像。對兩組影像擷取 Inception-v3 特徵，各自擬合一個高斯——平均值 μ_r, μ_g 與共變異數 Σ_r, Σ_g——再計算兩個高斯之間的 Fréchet 距離：

$$\mathrm{FID} = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\right).$$

越低越好。FID 使用真實資料，對品質與多樣性都敏感，因此成為標準指標。但它不是逐張影像的真實度量：它取決於特徵擷取器、影像縮放方式，特別是**樣本數**（有限樣本下它有偏差，所以只能比較樣本數相同的 FID），而且它把品質與涵蓋度混成一個數字。

精確率與召回率

為了把兩者分開，Sajjadi 等人 [30] 定義了**分布的精確率與召回率**：精確率問的是生成樣本中有多少比例落在真實資料所在之處（品質），召回率問的是真實分布有多少比例被生成器涵蓋（多樣性）。Kynkäänniemi 等人 [31] 提出一個簡單又穩健的估計方法，以下就來實作。用一組球的聯集近似每組樣本的流形，每顆球的半徑等於該點到同組中第 k 個最近鄰的距離。於是

- **精確率** = 落在某個真實樣本的球內的生成樣本比例；
- **召回率** = 落在某個生成樣本的球內的真實樣本比例。

在影像上，這些點是深度特徵而非像素。在我們的二維圓環上，可以直接用座標：

```
import numpy as np

rng = np.random.default_rng(0)
angles = np.arange(8) * 2 * np.pi / 8
centers = 2 * np.stack([np.cos(angles), np.sin(angles)], 1)

def ring(n, modes):
    idx = rng.choice(modes, n)
    return centers[idx] + .05 * rng.standard_normal((n, 2))

def knn_radius(a, k):
    d = np.linalg.norm(a[:, None] - a[None], axis=-1)
    return np.sort(d, axis=1)[:, k]          # column 0 is the point itself

def coverage(query, ref, k=3):
    """Fraction of query points inside some reference point's k-NN ball."""
    r = knn_radius(ref, k)
    d = np.linalg.norm(query[:, None] - ref[None], axis=-1)
    return np.mean((d <= r[None]).any(axis=1))
```

```
real = ring(1000, range(8))
fakes = {
    "all 8 modes": ring(1000, range(8)),
    "only 2 modes": ring(1000, [0, 4]),
    "8 modes, blurry": ring(1000, range(8)) + .4 * rng.standard_normal((1000, 2)),
}
for name, fake in fakes.items():
    precision = coverage(fake, real)   # are fake samples on the real manifold?
    recall = coverage(real, fake)      # is the real manifold covered by fakes?
    print(f"{name:16s} precision={precision:.2f} recall={recall:.2f}")
```

```
all 8 modes      precision=0.97 recall=0.96
only 2 modes     precision=0.97 recall=0.26
8 modes, blurry  precision=0.07 recall=0.99
```

模式崩塌的生成器維持很高的精確率——它的每個樣本都很完美——但召回率掉到約 2/8。模糊的生成器恰好相反：它涵蓋每個模態，但大多數樣本偏離資料。像 FID 這樣的單一純量會把兩者都判為「比較差」，卻說不出**為什麼**。圖 6 呈現這三種情況。

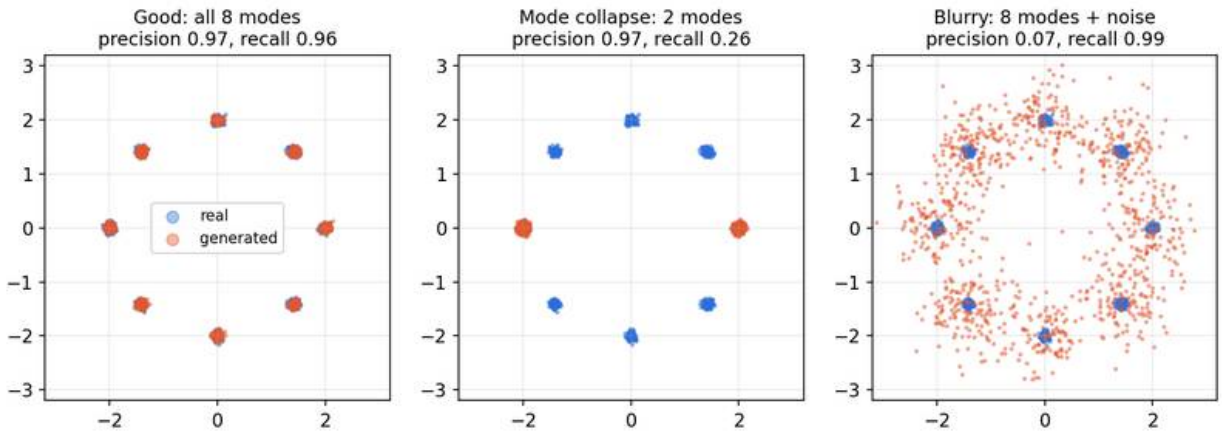


圖 *6* — 玩具圓環上的 *k* 近鄰精確率與召回率：模式崩塌降低召回率，模糊降低精確率。

現行指標的已知限制

評估本身就是活躍的研究領域。Stein 等人 [32] 進行了大規模人類實驗，發現沒有任何現有自動指標與人類的真實感判斷有強相關；特別是以 Inception 特徵計算的 FID，低估了擴散模型的感知真實度。他們建議 FID 與精確率／召回率之類的指標改用更豐富的自監督特徵空間（DINOv2 ViT-L/14）。Jayasumana 等人 [33] 主張 FID 的常態假設與 Inception 特徵不適合現代文字生成影像模型，並提出 **CMMD**：在 CLIP 嵌入上計算的最大平均差異（maximum mean discrepancy），不需要高斯假設。實務建議：回報多個互補指標並附上確切協定（特徵擷取器、樣本數、解析度、參考集）、展示未經挑選的樣本，而當主張涉及「看起來的品質」時，進行人類評估。

條件式任務還要加上**對齊度**的量測（影像是否符合文字或低解析度輸入？）。在影像復原中，一個

漂亮的生成紋理可能沒有觀測資料支持；「看起來合理的樣本」與「量測到的證據」之間的區別，是 DLCV 09 的核心議題。

現代觀點

值得一讀的綜述。四篇綜述從不同角度涵蓋本章內容。

- **Bond-Taylor 等人 [3]**（IEEE TPAMI 2022）是 VAE、GAN、正規化流、能量模型與自迴歸模型最好的單篇比較綜述。它依核心技術與已知病徵整理每個家族，並從訓練速度、取樣速度、樣本品質、多樣性與概似加以比較，結論是每個家族都在某處用弱點換來優點，而混合方法是自然的前進方向。
- **Papamakarios 等人 [4]**（JMLR 2021）是正規化流的統一論述：表達能力、耦合式／自迴歸式／殘差式／連續式流的設計空間，以及在密度估計、推論與模擬上的用途。
- **Kingma 與 Welling [9]**（Foundations and Trends in ML 2019）是 VAE 發明者撰寫的權威教學綜述：推導、重參數化估計量、更緊的下界、更有彈性的後驗（包括流）、階層式模型，以及加權 β 的變體。
- **Yang 等人 [27]**（ACM Computing Surveys 2023）綜述擴散模型——取樣加速、概似改進、結構化資料，以及與 VAE、GAN、流和 EBM 的關聯——是通往下一章的橋樑。

Bishop 與 Bishop [1]、Prince [2] 的教科書對所有家族都有一致的推導，Prince 的 notebook 更提供可執行的配套程式。

領域現況（2023–2026）。擴散與流匹配模型——通常在學到的潛在空間中運作——如今產出了大部分最先進的影像與影片生成結果。這讓**自編碼器**再次成為承重元件：VQ-VAE [34] 展示了如何學習離散潛在碼，讓另一個先驗模型去建模；潛在擴散 [35] 則證明，把生成模型放在壓縮後的自編碼器潛在空間中執行，能在保留細節的同時大幅降低成本。tokenizer 必須在壓縮率、重建品質，以及潛在空間是否容易建模之間取得平衡；請先評估它的重建，因為下游先驗無法還原編碼器已經丟掉的細節。

GAN 並沒有消失。GigaGAN [36] 把 GAN 擴展到文字生成影像，並凸顯 GAN 仍然最擅長的事：快到足以互動使用的單次傳遞取樣、高解析度上取樣，以及平滑、可編輯的潛在空間。Huang 等人 [37] 重新檢視訓練目標本身，推導出具局部收斂保證的正則化相對式（relativistic）GAN 損失，並顯示一個不依賴傳統技巧大全的極簡現代基準（R3GAN）在標準基準上能超越 StyleGAN2。對抗損失也繼續存在於其他系統中，例如用來讓 tokenizer 的重建更銳利。

在評估方面，2023–2024 年對 FID 的批評 [32]、[33] 推動領域轉向更豐富的特徵擷取器、分開的精確率／召回率式指標與人類評估，但至今仍沒有公認的單一指標。

開放問題。（1）與人類一致、且能區分品質、多樣性與記憶的可靠評估。（2）偵測模型何時在複製訓練影像而非泛化。（3）真正能用於分布外偵測的校準概似 [28]。（4）速度、品質與涵蓋度的三角取捨：在不失去模態的前提下，以 GAN 的速度達到擴散模型的品質。（5）學出同時具備壓縮性、忠實性又容易建模的潛在空間。

重點整理

- 生成器需要明確分布或取樣程序，只有重建並不足夠。
- 最大概似擬合觀測資料的機率；高概似與逼真樣本是不同目標。
- ELBO 是對數概似的下界，差距取決於近似後驗的失配。
- 對抗目標依賴判別器行為，訓練改進不保證收斂或避免模式崩塌。
- 可逆流透過 Jacobian 行列式追蹤密度；可計算的變換限制其設計。
- 回報互補指標，並交代特徵擷取器、樣本數與條件設定協定。

練習

1. **KL 健全性檢查**。把 $\mu = 0$、$\sigma^2 = 1$ 代入高斯 KL。接著計算一維中 $\mu = 1, \sigma^2 = 1$ 的 KL。

   提示

   $\mu = 0, \sigma^2 = 1$ 時每一項都是零，因為後驗與先驗是同一個分布。$\mu = 1, \sigma^2 = 1$ 時得到 $\frac{1}{2}(1+1-0-1) = 0.5$ nats：把平均值移動一個標準差，代價是半個 nat。

2. **損失縮放**。把 VAE 的重建項改成對像素取平均而不是加總。若 KL 保持不變，會發生什麼事？

   提示

   相對於 KL，重建項變小了 64 倍。這等同於對加總損失使用 $\beta = 64$ 的 β-VAE：它實際上改變了正則化強度（往往導向後驗崩塌），而不只是改變回報的單位。

3. **最佳判別器**。在 $p_{\text{data}}(x) = 0.3$、$p_g(x) = 0.1$ 的點 x 上，$D^*(x)$ 是多少？當 $p_g = p_{\text{data}}$ 時，D^* 處處是多少？此時遊戲的值是多少？

   提示

   $D^*(x) = 0.3/(0.3 + 0.1) = 0.75$。兩個分布相同時，處處 $D^* = 1/2$，$V = \log \frac{1}{2} + \log \frac{1}{2} = -\log 4$，與 JSD $= 0$ 一致。

4. **仿射流**。對 $x = Az + b$、$z \sim \mathcal{N}(0, I)$，推導密度修正項，並說明為何奇異的 A 會使它失效。再用已知的高斯 $\mathcal{N}(b, AA^\top)$ 檢查你的答案。

   提示

   減去 $\log|\det A|$。因為 $\det(AA^\top)^{1/2} = |\det A|$，這與 $\mathcal{N}(b, AA^\top)$ 的正規化常數一致。奇異映射不可逆，會把體積壓扁，所以同維度的變數變換公式不適用。

5. **辨認崩塌**。分別為 VAE 後驗崩塌與 GAN 模式崩塌提出不同的檢查方法。接著把 GAN 程式改成 WGAN-GP（評論器不加 sigmoid、梯度懲罰 $\lambda = 10$、每次生成器更新前做多次評論器更新），並比較各模態的樣本數。

提示

VAE：檢查每個潛在維度的 KL，以及改變 z 時解碼輸出是否跟著變。GAN：統計每個模態的樣本數（如我們的程式）、計算召回率，並在固定條件下找重複的樣本。這是兩種不同的失敗。WGAN-GP：取樣 $\hat{x} = \alpha x_{\text{real}} + (1-\alpha)x_{\text{fake}}$、$\alpha \sim U(0,1)$，呼叫 `torch.autograd.grad` 並設 `create_graph=True`；若訓練震盪就降低學習率。下結論前請比較多個亂數種子。

6. **指標的盲點**。利用精確率／召回率程式，建構一組生成樣本，使它的精確率與召回率和好的生成器差不多，但一半的樣本只集中在四個模態。這告訴你這個指標有什麼限制？

提示

模態權重不均仍可能得到高召回率，因為召回率只問每個真實點附近是否有**某個**生成點，而不問模態的頻率是否相符。精確率／召回率量測的是支撐集的重疊，而不是密度；請搭配 FID 這類分布層級的指標或逐模態直方圖。

參考文獻

1. C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024. Book
2. S. J. D. Prince, *Understanding Deep Learning*, MIT Press, 2023. Book and notebooks
3. S. Bond-Taylor, A. Leach, Y. Long and C. G. Willcocks, "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. Paper
4. G. Papamakarios et al., "Normalizing Flows for Probabilistic Modeling and Inference," *JMLR*, 2021. Review
5. L. Theis, A. van den Oord and M. Bethge, "A note on the evaluation of generative models," ICLR, 2016. Paper
6. D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," ICLR, 2014. Paper
7. S. R. Bowman et al., "Generating Sentences from a Continuous Space," CoNLL, 2016. Paper
8. I. Higgins et al., "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," ICLR, 2017. Paper
9. D. P. Kingma and M. Welling, "An Introduction to Variational Autoencoders," *Foundations and Trends in Machine Learning*, 2019. Review
10. A. Vahdat and J. Kautz, "NVAE: A Deep Hierarchical Variational Autoencoder," NeurIPS, 2020. Paper
11. I. Goodfellow et al., "Generative Adversarial Nets," NeurIPS, 2014. Paper
12. T. Salimans et al., "Improved Techniques for Training GANs," NeurIPS, 2016. Paper
13. M. Arjovsky, S. Chintala and L. Bottou, "Wasserstein GAN," ICML, 2017. Paper
14. I. Gulrajani et al., "Improved Training of Wasserstein GANs," NeurIPS, 2017. Paper
15. T. Miyato et al., "Spectral Normalization for Generative Adversarial Networks," ICLR, 2018. Paper
16. L. Mescheder, A. Geiger and S. Nowozin, "Which Training Methods for GANs do actually Converge?" ICML, 2018. Paper
17. T. Karras, S. Laine and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," CVPR, 2019. Paper

18. L. Dinh, J. Sohl-Dickstein and S. Bengio, “Density estimation using Real NVP,” ICLR, 2017. Paper
19. D. P. Kingma and P. Dhariwal, “Glow: Generative Flow with Invertible 1x1 Convolutions,” NeurIPS, 2018. Paper
20. D. J. Rezende and S. Mohamed, “Variational Inference with Normalizing Flows,” ICML, 2015. Paper
21. R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud, “Neural Ordinary Differential Equations,” NeurIPS, 2018. Paper
22. Y. Lipman et al., “Flow Matching for Generative Modeling,” ICLR, 2023. Paper
23. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 18–20. Book
24. Y. Song and D. P. Kingma, “How to Train Your Energy-Based Models,” arXiv, 2021. Paper
25. A. van den Oord, N. Kalchbrenner and K. Kavukcuoglu, “Pixel Recurrent Neural Networks,” ICML, 2016. Paper
26. J. Ho, A. Jain and P. Abbeel, “Denoising Diffusion Probabilistic Models,” NeurIPS, 2020. Paper
27. L. Yang et al., “Diffusion Models: A Comprehensive Survey of Methods and Applications,” *ACM Computing Surveys*, 2023. Survey
28. E. Nalisnick et al., “Do Deep Generative Models Know What They Don’t Know?” ICLR, 2019. Paper
29. M. Heusel et al., “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium,” NeurIPS, 2017. Paper
30. M. S. M. Sajjadi, O. Bachem, M. Lucic, O. Bousquet and S. Gelly, “Assessing Generative Models via Precision and Recall,” NeurIPS, 2018. Paper
31. T. Kynkäänniemi, T. Karras, S. Laine, J. Lehtinen and T. Aila, “Improved Precision and Recall Metric for Assessing Generative Models,” NeurIPS, 2019. Paper
32. G. Stein et al., “Exposing flaws of generative model evaluation metrics and their unfair treatment of diffusion models,” NeurIPS, 2023. Paper
33. S. Jayasumana et al., “Rethinking FID: Towards a Better Evaluation Metric for Image Generation,” CVPR, 2024. Paper
34. A. van den Oord, O. Vinyals and K. Kavukcuoglu, “Neural Discrete Representation Learning,” NeurIPS, 2017. Paper
35. R. Rombach et al., “High-Resolution Image Synthesis with Latent Diffusion Models,” CVPR, 2022. Paper
36. M. Kang et al., “Scaling up GANs for Text-to-Image Synthesis,” CVPR, 2023. Paper
37. Y. Huang, A. Gokaslan, V. Kuleshov and J. Tompkin, “The GAN is dead; long live the GAN! A Modern GAN Baseline,” NeurIPS, 2024. Paper

ML 08・擴散模型與流匹配

推導雜訊、乾淨影像與速度預測，以 JiT 和表徵自編碼器比較像素與潛在生成，並理解預訓練影像／影片的擴散與流生成流程。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/

你將學到

- 建立前向加雜訊程序與反向預測任務。
- 連結變分訓練與雜訊、乾淨影像及速度目標。
- 區分模型預測與數值取樣方法。
- 比較條件化、生成表示與流匹配。
- 評估完整影像／影片流程與少步方法。

先看全貌

擴散模型學的是「把受控的破壞復原」。訓練時，我們拿一個真實樣本，加上已知強度的 Gaussian 雜訊，然後請神經網路告訴我們加了什麼雜訊。因為雜訊是我們自己加的，正確答案永遠拿得到：不需要標籤，也不需要對手，就是迴歸而已。生成時，我們從純雜訊出發，反覆套用學到的「去掉一點雜訊」步驟，直到一個樣本浮現出來。

> **用最簡單的話說**
>
> 想像你慢慢用鉛筆在一張畫上亂塗。你把畫在各種塗鴉程度下都留一份：幾條線、很多線、多到原本的畫完全看不見。模型只練習一件小事：看一張被塗過的紙，猜出哪些線是後來加上去的。練習夠多之後，你給它一張「只有」塗鴉的紙。它擦掉一點、再看一次、再擦掉一點，一張全新的畫就慢慢出現了——而且這張畫從來不在那疊紙裡。

為什麼重要？從 2021 年起，擴散類模型在影像、影片與音訊合成上拿下了許多最先進的結果，同一套機制也用於影像修補、超解析度、影像編輯、3D 生成與科學資料。擴散模型也解決了 ML 07 中較早模型家族的兩個痛點：GAN 需要脆弱的雙人賽局，VAE 的樣本則容易模糊。擴散模型的訓練是穩定的迴歸，而因為生成被拆成許多小而容易的步驟，樣本也很清晰。

代價是取樣成本：經典取樣器每產生一張影像要呼叫網路數百次。2021 年以來的大量研究——DDIM、快速 ODE 求解器、蒸餾、一致性模型、流匹配、整流流——都是在維持品質的同時減少網路呼叫次數。

本章依照幾篇主要綜述的架構安排。Yang 等人 [1] 把領域分成高效取樣、改進概似（likelihood）與特殊結構資料三個方向；Croitoru 等人 [27] 依公式化方式（DDPM、雜訊條件分數網路、SDE）分類；Lipman 等人的流匹配指南 [2] 與 Lai 等人的專書 [29] 則把擴散與流視為同一族「學習出來的傳輸」。Prince 的教科書 [14] 是推導時很好的入門參考。全文**每一節只使用一種明確的時間方向慣例**，換慣例時會特別說明——混用慣例是這個領域最常見的正負號錯誤來源。

前向擴散有封閉解

白話版。前向程序逐漸減弱影像訊號並加入高斯雜訊；適當的終點排程使分布近似標準高斯。

這個想法可追溯到 Sohl-Dickstein 等人 [15]：用緩慢的前向擴散破壞結構，再學習把它反轉；DDPM [3] 讓它變得實用。令 $x_0 \in \mathbb{R}^d$ 為一筆資料（把影像攤平成 d 個數字，大致縮放到 $[-1, 1]$）。選一組**雜訊排程（noise schedule）** $\beta_1, \dots, \beta_T$，其中 $0 < \beta_t < 1$，並記 $\alpha_t = 1 - \beta_t$。前向（加噪）過程是一條馬可夫鏈

$$q(x_t \mid x_{t-1}) = \mathcal{N}\big(\sqrt{\alpha_t}\, x_{t-1},\ \beta_t I\big), \qquad t = 1, \dots, T,$$

其中 $\mathcal{N}(\mu, \Sigma)$ 表示平均為 μ、共變異數為 Σ 的 Gaussian，I 是 $d \times d$ 單位矩陣。係數 $\sqrt{\alpha_t}$ 把訊號稍微縮小，讓總變異數維持有界而不會一路變大；這也是這一族被稱為**變異數保持（variance preserving, VP）**的原因。

推導一次到位的公式

把一步寫成 $x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1-\alpha_t}\,\epsilon_t$，其中 $\epsilon_t \sim \mathcal{N}(0, I)$。代入前一步，

$$\begin{aligned} x_t &= \sqrt{\alpha_t}\left(\sqrt{\alpha_{t-1}} x_{t-2} + \sqrt{1-\alpha_{t-1}}\,\epsilon_{t-1}\right) + \sqrt{1-\alpha_t}\,\epsilon_t \\ &= \sqrt{\alpha_t \alpha_{t-1}}\, x_{t-2} + \underbrace{\sqrt{\alpha_t(1-\alpha_{t-1})}\,\epsilon_{t-1} + \sqrt{1-\alpha_t}\,\epsilon_t}_{\text{獨立 Gaussian 之和}}. \end{aligned}$$

兩個雜訊項是獨立、平均為零的 Gaussian，所以相加後仍是 Gaussian，變異數為 $\alpha_t(1-\alpha_{t-1}) + (1-\alpha_t) = 1 - \alpha_t\alpha_{t-1}$。把同樣的論證一路推回 x_0，令 $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$，得到

$$q(x_t \mid x_0) = \mathcal{N}\big(\sqrt{\bar{\alpha}_t}\, x_0,\ (1-\bar{\alpha}_t) I\big) \quad \Longleftrightarrow \quad x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0, I).$$

這裡 $\bar{\alpha}_t$ 是到第 t 步仍保留的訊號變異數比例，ϵ 是一個標準 Gaussian 向量。這條公式讓訓練很便宜：我們可以直接跳到任何雜訊程度，不必模擬前面的每一步。

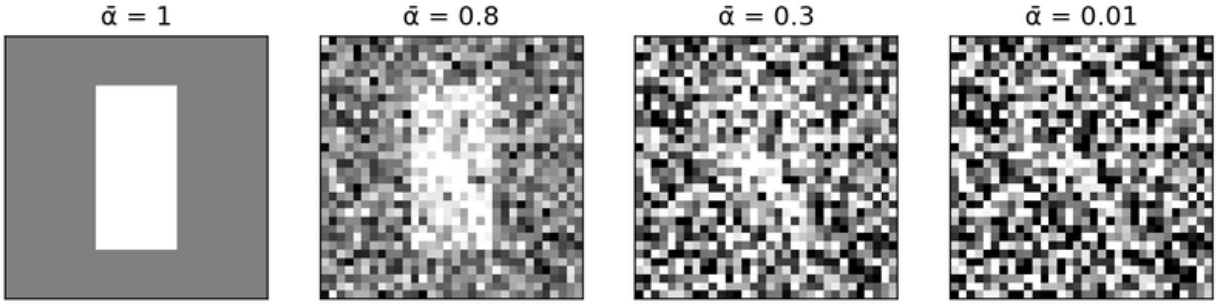


圖 *1* — 封閉形式前向過程在幾個訊號程度下的樣子。這裡共用同一組雜訊以便看出效果；實際訓練時每筆資料、每一步都抽新的雜訊。

雜訊排程與訊雜比

第 t 步的**訊雜比（signal-to-noise ratio, SNR）**為

$$\mathrm{SNR}(t) = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t},$$

即訊號變異數與雜訊變異數之比（假設資料變異數為 1）。它從 $t=1$ 時非常大，單調下降到 $t=T$ 時接近零。排程有兩個要求：$\bar{\alpha}_T$ 必須接近 0，讓 x_T 實際上就是標準 Gaussian（取樣器從 $\mathcal{N}(0, I)$ 出發）；中間各步則應把訓練資源花在網路真的有東西可學的雜訊程度上。

DDPM [3] 使用 $T = 1000$ 與**線性排程**，β_t 從 10^{-4} 線性增加到 0.02。Nichol 與 Dhariwal [19] 觀察到這樣會太快摧毀資訊：鏈的相當一大段裡，影像幾乎已經是純雜訊。他們的**餘弦排程**定義 $\bar{\alpha}_t = f(t)/f(0)$，其中 $f(t) = \cos^2\!\left(\frac{t/T+s}{1+s}\cdot\frac{\pi}{2}\right)$，小偏移量 $s = 0.008$，並把每個 β_t 截在 0.999。圖 2 比較兩者。Karras 等人 [20] 後來主張：排程、網路輸入輸出的縮放，以及訓練時雜訊程度的分布，是應該分開調整的獨立設計選擇。

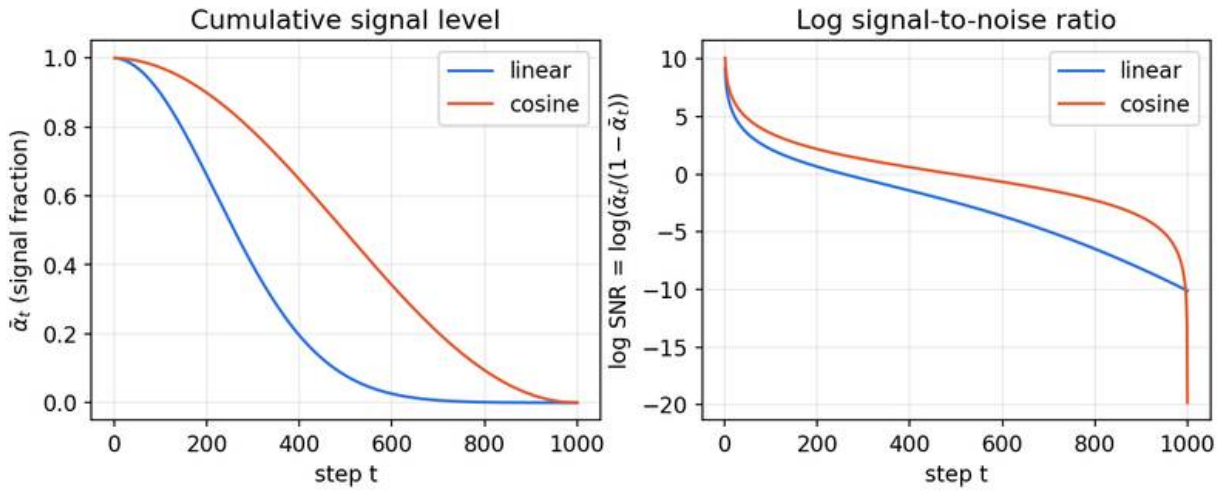


圖 2 — *$T = 1000$ 時的線性排程與餘弦排程。餘弦排程讓訊號保留得更久，對數訊雜比在各步之間分布得更平均。*

反向過程與變分下界

白話版。要生成，就需要反向的一步：「給一個有雜訊的樣本，稍微乾淨一點的樣本長什麼樣？」對真實資料我們無法直接算出來，但如果同時知道乾淨樣本，就算得出來——而這正好給出一個簡單的訓練損失。

模型與下界

生成模型是一條反向馬可夫鏈，從 $p(x_T) = \mathcal{N}(0, I)$ 出發，使用學習出來的 Gaussian 轉移

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}\big(\mu_\theta(x_t, t),\ \sigma_t^2 I\big),$$

其中 μ_θ 是參數為 θ 的神經網路，σ_t^2 是我們稍後要選的變異數。這是一個以 $x_{1:T}$ 為潛在變數的模型，就像一個編碼器 q 固定而非學習的深層 VAE（見 ML 07）。同樣的 Jensen 不等式論證給出證據下界（ELBO）。依照 [3]、[15]，負的下界可以整理成一串 KL 散度的和：

$$-\log p_\theta(x_0) \;\le\; \mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(x_T\mid x_0)\,\|\,p(x_T))}_{L_T} + \sum_{t=2}^{T}\underbrace{D_{\mathrm{KL}}(q(x_{t-1}\mid x_t,x_0)\,\|\,p_\theta(x_{t-1}\mid x_t))}_{L_{t-1}} \underbrace{-\log p_\theta(x_0\mid x_1)}_{L_0}\Big].$$

D_{KL} 是 Kullback–Leibler 散度。L_T 沒有可學參數（當 $\bar\alpha_T \approx 0$ 時很小）；L_0 是最後一步的解碼項；每個 L_{t-1} 則把模型的反向一步與**前向後驗分布** $q(x_{t-1}\mid x_t,x_0)$ 相比——也就是「如果知道乾淨樣本 x_0，反向一步會是什麼」的精確答案。

前向後驗是 Gaussian

由貝氏定理，$q(x_{t-1}\mid x_t,x_0) \propto q(x_t\mid x_{t-1})\,q(x_{t-1}\mid x_0)$。兩個因子對 x_{t-1} 都是 Gaussian，所以乘積也是 Gaussian。把 x_{t-1} 的二次項與一次項收集起來（配方），得到

$$q(x_{t-1}\mid x_t,x_0) = \mathcal{N}\big(\tilde\mu_t(x_t,x_0),\,\tilde\beta_t I\big),$$

$$\tilde\mu_t = \frac{\sqrt{\bar\alpha_{t-1}}\,\beta_t}{1-\bar\alpha_t}\,x_0 + \frac{\sqrt{\alpha_t}\,(1-\bar\alpha_{t-1})}{1-\bar\alpha_t}\,x_t, \qquad \tilde\beta_t = \frac{1-\bar\alpha_{t-1}}{1-\bar\alpha_t}\,\beta_t.$$

後驗平均是乾淨樣本與目前雜訊樣本的加權平均；後驗變異數 $\tilde\beta_t$ 比 β_t 略小。你不必相信代數：下面的程式抽樣前向鏈，並以 x_t 對 x_{t-1} 做迴歸。

```
import numpy as np

rng = np.random.default_rng(0)
beta = np.linspace(1e-4, 0.02, 1000)
alpha = 1 - beta
abar = np.cumprod(alpha)
t, x0, n = 300, 1.3, 400_000           # check q(x_{t-1} | x_t, x_0) at step t

# Sample the forward chain: x_{t-1} from q(x_{t-1}|x_0), then one more step.
x_prev = np.sqrt(abar[t-1])*x0 + np.sqrt(1-abar[t-1])*rng.normal(size=n)
x_t = np.sqrt(alpha[t])*x_prev + np.sqrt(beta[t])*rng.normal(size=n)

# Empirical regression of x_{t-1} on x_t (the posterior mean is linear in x_t).
slope, intercept = np.polyfit(x_t, x_prev, 1)
resid_var = np.var(x_prev - (slope*x_t + intercept))

# Closed-form posterior from the derivation.
c_x0 = np.sqrt(abar[t-1])*beta[t] / (1-abar[t])
c_xt = np.sqrt(alpha[t])*(1-abar[t-1]) / (1-abar[t])
var = beta[t]*(1-abar[t-1]) / (1-abar[t])
print(f"slope     {slope:.4f}  vs  {c_xt:.4f}")
print(f"intercept {intercept:.4f}  vs  {c_x0*x0:.4f}")
print(f"variance  {resid_var:.6f}  vs  {var:.6f}")
```

輸出在小數點後三到四位吻合（斜率 0.9934 對 0.9930，變異數 0.006050 對 0.006052）；微小差距是蒙地卡羅誤差。

從下界到簡化損失

現在把前向公式反過來用：$x_0 = (x_t - \sqrt{1-\bar\alpha_t}\,\epsilon)/\sqrt{\bar\alpha_t}$。代入 $\tilde\mu_t$，再用 $\bar\alpha_t = \alpha_t\bar\alpha_{t-1}$ 化簡，

$$\tilde\mu_t = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar\alpha_t}}\,\epsilon\right).$$

取樣時網路看得到 x_t，卻看不到 ϵ。所以 DDPM 讓模型平均採用完全相同的形式，只是把未知的雜訊換成預測值 $\epsilon_\theta(x_t, t)$：

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar\alpha_t}}\,\epsilon_\theta(x_t, t)\right).$$

兩個共變異數同為固定 $\sigma_t^2 I$ 的 Gaussian，其 KL 散度就是兩個平均之間平方距離的縮放，因此

$$L_{t-1} = \frac{1}{2\sigma_t^2}\|\tilde\mu_t - \mu_\theta\|^2 + C = \frac{\beta_t^2}{2\sigma_t^2\,\alpha_t\,(1-\bar\alpha_t)}\,\|\epsilon - \epsilon_\theta(x_t, t)\|^2 + C,$$

其中 C 與 θ 無關。下界的每一項因此都是**加權的雜訊預測誤差**。Ho 等人 [3] 發現，拿掉權重並均勻抽樣 t，訓練效果反而更好：

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{x_0,\,t\sim\mathcal{U}\{1,\dots,T\},\,\epsilon\sim\mathcal{N}(0,I)}\left\|\epsilon - \epsilon_\theta\!\left(\sqrt{\bar\alpha_t}x_0 + \sqrt{1-\bar\alpha_t}\,\epsilon,\ t\right)\right\|^2.$$

把它當成演算法來讀：挑一筆訓練資料、挑一個隨機步數、挑隨機雜訊，用一行算出 x_t，然後迴歸這個雜訊。時間 t 會餵給網路（通常透過正弦嵌入），因為同樣的像素值在不同雜訊程度下代表不同意義。

有兩點容易弄錯。第一，$\mathcal{L}_{\text{simple}}$ 不是概似下界：拿掉權重相對降低了低雜訊步的比重，這有助於樣本品質，卻會讓報告的對數概似變差。Improved DDPM [19] 用混合損失學習變異數 σ_t^2（在 β_t 與 $\tilde\beta_t$ 之間內插），找回了好的概似。第二，同一個網路也給出**乾淨估計**

$$\hat x_0(x_t, t) = \frac{x_t - \sqrt{1-\bar\alpha_t}\,\epsilon_\theta(x_t, t)}{\sqrt{\bar\alpha_t}},$$

每個現代取樣器內部都會用到它。當 $\bar\alpha_t$ 非常小時，這個除法會放大預測誤差，所以高雜訊時的 $\hat x_0$ 是一個模糊的平均，而不是一個樣本。

僅線上版：*DDPM：雜訊預測與品質、概似之間的取捨 — https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/ddpm 雜訊預測與品質概似之間的取捨/*

雜訊、乾淨資料、速度與分數

白話版。預測雜訊、預測乾淨影像、預測「往哪個方向走會讓這個樣本更典型」，是表達同一份知識的三種方式。差別在縮放，而縮放會改變訓練損失對不同雜訊程度的加權。

三種參數化

因為 $x_t = \sqrt{\bar\alpha_t}x_0 + \sqrt{1-\bar\alpha_t}\epsilon$ 是固定的線性關係，網路可以輸出下列任何一個：

- 雜訊 ϵ（DDPM [3]），
- 乾淨樣本 x_0，
- **速度** $v = \sqrt{\bar\alpha_t}\,\epsilon - \sqrt{1-\bar\alpha_t}\,x_0$，由 Salimans 與 Ho [22] 為了蒸餾而提出，在排程兩端都維持良好的尺度。

給定 x_t 與 t，三者可以彼此精確換算。但是對 ϵ、x_0 或 v 做不加權的 MSE，對各雜訊程度的加權並不相同，所以這個選擇會影響訓練。Karras 等人 [20] 用輸入輸出的「預條件（preconditioning）」係數把這件事講清楚：讓網路在每個雜訊程度下的輸入與目標都維持單位變異數。注意，這裡擴散模型的「速度」是訊號與雜訊的一種重新參數化；它與本章後面流匹配的傳輸速度相關，但不相同。

精確轉換不代表有限容量網路等價

對保留變異數的路徑，令 $a = \sqrt{\bar\alpha_t}$ 與 $b = \sqrt{1-\bar\alpha_t}$，因此 $a^2 + b^2 = 1$，而 $x_t = ax_0 + b\epsilon$。擴散速度 $v_{\mathrm{VP}} = a\epsilon - bx_0$ 與前向狀態共同構成一個旋轉：

$$x_0 = ax_t - bv_{\mathrm{VP}}, \qquad \epsilon = bx_t + av_{\mathrm{VP}},$$
$$\hat x_0^{(\epsilon)} = (x_t - b\hat\epsilon)/a, \quad \hat\epsilon^{(x)} = (x_t - a\hat x_0)/b.$$

這裡 a, b 是已知的排程係數；v_{VP} 是 [22] 使用的目標；帽號表示估計值；上標指出網路直接輸出的量。最後兩種轉換的除數必須非零。即使內部時間的代數精確，端點附近仍有數值與統計上的不穩定問題。

雜訊誤差 δ_ϵ 會變成乾淨影像誤差 $-(b/a)\delta_\epsilon$。因此，固定雜訊程度下互相對應的預測滿足：

$$\|\hat x_0 - x_0\|^2 = \frac{b^2}{a^2}\|\hat\epsilon - \epsilon\|^2 = \frac{1}{\mathrm{SNR}(t)}\|\hat\epsilon - \epsilon\|^2, \qquad \|\hat x_0^{(v)} - x_0\|^2 = b^2\|\hat v_{\mathrm{VP}} - v_{\mathrm{VP}}\|^2.$$

$\mathrm{SNR}(t) = a^2/b^2$ 是訊雜比；第二個恆等式由前述旋轉得到。相同形式的無加權損失因此會強調不同時間。改變目標卻未記錄時間抽樣與損失權重，就混入了其他實驗變因 [20]、[22]。

另外還有容量問題。若網路直接輸出乾淨影像，最後的表徵可以捨棄干擾性的雜訊。直接輸出雜訊的網路可能必須保留輸入中更多座標，或透過跳接取得那些座標。事後轉換某個網路的輸出，不會讓它的架構等同於直接學習另一種目標的網路。對大像素區塊與高維潛在表徵，這個差異尤其值得檢查；後面的 JiT 討論提供近期證據 [42]。這不代表雜訊預測普遍失敗：前面的 DDPM 範例與密集影像去雜訊網路就是反例。

```
import numpy as np

rng = np.random.default_rng(7)
x0 = rng.normal(size=32)
eps = rng.normal(size=32)
a, b = np.sqrt(.2), np.sqrt(.8)
xt = a*x0 + b*eps
```

```
v_vp = a*eps - b*x0
assert np.allclose(a*xt - b*v_vp, x0)
assert np.allclose(b*xt + a*v_vp, eps)

# A deliberately imperfect noise estimate: compare equivalent losses.
eps_hat = eps + .03*rng.normal(size=32)
x_hat = (xt - b*eps_hat)/a
noise_mse = np.mean((eps_hat - eps)**2)
clean_mse = np.mean((x_hat - x0)**2)
assert np.allclose(clean_mse/noise_mse, b*b/(a*a))
print("Clean/noise MSE ratio:", round(clean_mse/noise_mse, 3))

# A straight flow uses a different path and a different velocity.
t = .35
zt = (1-t)*eps + t*x0
u = x0 - eps
x_from_flow = zt + (1-t)*u
u_from_clean = (x0 - zt)/(1-t)
assert np.allclose(x_from_flow, x0)
assert np.allclose(u_from_clean, u)
print("VP velocity equals straight-flow target:", np.allclose(v_vp, u))
```

範例印出損失比例 4.0，並以 `False` 指出兩種速度目標不同。文獻有時都用字母 v；檢查點的排程器必須指出真正的路徑、正規化與時間方向。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

什麼是分數？為什麼去雜訊能估計它？

密度 p 的**分數（score）**是 $\nabla_x \log p(x)$：一個指向更高機率方向的向量場。如果知道分數，就能用 **Langevin 動力學**取樣：沿著分數走一小步、加一點雜訊，反覆進行。Song 與 Ermon [17] 在多個雜訊程度上訓練「雜訊條件分數網路（NCSN）」，並以退火 Langevin 動力學取樣——這是一條與擴散平行的路線，後來證明是同一個想法。

連接兩者的橋梁是**去雜訊分數匹配（denoising score matching）**（Vincent [16]）。對 Gaussian 破壞而言，（在均方誤差意義下）最好的雜訊預測器是一個條件期望值 $\epsilon^*(x_t, t) = \mathbb{E}[\epsilon \mid x_t]$，而 **Tweedie 公式** [40] 把它連到雜訊邊際分布 p_t 的分數：

$$\nabla_{x_t} \log p_t(x_t) = -\frac{\mathbb{E}[\epsilon \mid x_t]}{\sqrt{1-\bar{\alpha}_t}} \quad \Longrightarrow \quad s_\theta(x_t, t) \approx -\frac{\epsilon_\theta(x_t, t)}{\sqrt{1-\bar{\alpha}_t}},$$

其中 s_θ 是學到的分數。同一個公式用「乾淨」單位寫成 $\mathbb{E}[x_0 \mid x_t] = (x_t + (1-\bar{\alpha}_t)\nabla \log p_t(x_t))/\sqrt{\bar{\alpha}_t}$。兩個提醒：這是第 t 層加噪後的邊際分布的分數，不是任何單張訓練影像的梯度；而且它對最佳預測器成立——訓練出來的網路只是近似。

你可以在一切都已知的情況下驗證 Tweedie 公式：一維 Gaussian 資料，此時最佳雜訊預測器對 x_t 是線性的。

```
import numpy as np

rng = np.random.default_rng(0)
s, abar, n = 0.7, 0.4, 500_000     # data x0 ~ N(0, s^2); one noise level
x0 = s*rng.normal(size=n)
eps = rng.normal(size=n)
xt = np.sqrt(abar)*x0 + np.sqrt(1-abar)*eps

# "Train" the best linear noise predictor eps_hat = w * x_t by least squares.
w = (xt @ eps) / (xt @ xt)
score_from_eps = -w / np.sqrt(1-abar)       # s(x_t) = -eps_hat / sqrt(1-abar)

# Exact: x_t ~ N(0, abar s^2 + 1 - abar), so grad log p_t(x) = -x / var.
exact = -1 / (abar*s**2 + 1 - abar)
print(f"score slope from noise predictor: {score_from_eps:.4f}")
print(f"exact score slope:                {exact:.4f}")
```

它印出 -1.2560 對 -1.2563：對雜訊做迴歸就能找回分數。

SDE 觀點

Song 等人 [4] 讓步數趨於無窮。用連續時間 $t \in [0,1]$ 與速率 $\beta(t)$，VP 前向過程變成**隨機微分方程（SDE）**

$$dx = -\tfrac{1}{2}\beta(t)\,x\,dt + \sqrt{\beta(t)}\,dw,$$

其中 w 是標準 Wiener 過程（布朗運動）。一個經典結果指出：這個過程倒著時間跑，仍然是一條 SDE，而其中唯一未知的量就是分數：

$$dx = \Big[-\tfrac{1}{2}\beta(t)\,x - \beta(t)\,\nabla_x \log p_t(x)\Big]dt + \sqrt{\beta(t)}\,d\bar{w},$$

此處 dt 為負，$\bar{w}$ 是反向時間的 Wiener 過程。此外，還存在一條確定性的**機率流 ODE（probability-flow ODE）**，其解在每個時間點的邊際分布 p_t 都相同：

$$\frac{dx}{dt} = -\tfrac{1}{2}\beta(t)\Big[x + \nabla_x \log p_t(x)\Big].$$

這把整個領域統一起來。DDPM 的祖先取樣與反向 SDE 的離散化相連；NCSN [17] 對應到「變異數爆炸（variance exploding）」SDE。機率流 ODE 定義一個連續正規化流，其對數密度遵循瞬時變數變換恆等式 [4]。恆等式是精確的，但實際計算仍有 ODE 積分誤差；若以隨機方式估計散度，還有估計誤差。使用近似的學得分數時，ODE 與反向 SDE 的邊際分布也不一定完全相同。ODE 觀點同時說明了確定性取樣與一般數值求解器的用途。

取樣是一個反向過程

白話版。在單一隨機雜訊程度上算出的損失並不是生成器。要生成，必須從純雜訊出發，以一致的排程串起許多反向步驟——而你可以選擇要走幾步、每步注入多少新的隨機性。

DDPM 祖先取樣

DDPM 沿著學到的反向鏈取樣：

$$x_{t-1} = \mu_\theta(x_t, t) + \sigma_t z, \qquad z \sim \mathcal{N}(0, I), \qquad \sigma_t^2 \in \{\beta_t,\ \tilde{\beta}_t\},$$

最後一步不再加入雜訊。Ho 等人 [3] 報告 $\sigma_t^2 = \beta_t$ 與 $\sigma_t^2 = \tilde{\beta}_t$ 的結果相近；學習出的變異數 [19] 在步數較少時幫助最大。

程式：二維分布上的 DDPM

這個完整的 CPU 範例（約 10 秒）使用兩個中心在 $x = \pm 1.5$、標準差 0.25 的 Gaussian 群集。它包含影像模型的每個運算——排程、隨機時間訓練、雜訊預測、反向平均——卻不需要影像等級的計算量。在 100 步與這個排程下，到 $t = T$ 時幾乎沒有訊號殘留。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
T = 100
beta = torch.linspace(.001, .15, T)
alpha = 1-beta
abar = alpha.cumprod(0)
previous = torch.cat([torch.ones(1), abar[:-1]])
post_var = beta * (1-previous) / (1-abar)
net = nn.Sequential(nn.Linear(3, 64), nn.SiLU(),
                    nn.Linear(64, 64), nn.SiLU(), nn.Linear(64, 2))
opt = torch.optim.Adam(net.parameters(), lr=.002)
def data(n):
    side = torch.randint(2, (n, 1))*2-1
    return torch.cat([side*1.5, torch.zeros(n, 1)], 1) + .25*torch.randn(n, 2)
for step in range(3000):
    x0 = data(128)
    t = torch.randint(T, (len(x0),))
    eps = torch.randn_like(x0)
    a = abar[t, None]
    xt = a.sqrt()*x0 + (1-a).sqrt()*eps
    predicted = net(torch.cat([xt, t[:, None].float()/(T-1)], 1))
    loss = F.mse_loss(predicted, eps)
    opt.zero_grad(); loss.backward(); opt.step()
net.eval()
with torch.no_grad():
    x = torch.randn(512, 2)
    for t in reversed(range(T)):
        time = torch.full((len(x), 1), t/(T-1))
        eps = net(torch.cat([x, time], 1))
        mean = (x-beta[t]/(1-abar[t]).sqrt()*eps)/alpha[t].sqrt()
        x = mean + post_var[t].sqrt()*torch.randn_like(x) if t else mean
    assert torch.isfinite(x).all()
    print("terminal signal:", round(abar[-1].item(), 6))
```

```
    left = x[x[:, 0] < 0]
    print("fraction in left cluster:", round((x[:, 0] < 0).float().mean().item(), 3))
    print("left cluster: mean x", round(left[:, 0].mean().item(), 2),
          "std", round(left[:, 0].std().item(), 2), "(data: -1.5 and 0.25)")
```

在我們的機器上，它印出終端訊號 0.000349、左群集比例 0.441，左群集平均與標準差為 −1.47 與 0.25。群集的形狀是對的；44/56 的分配則顯示這個小網路沒有精確學到各模態的權重。這就是為什麼單一摘要數字永遠不夠：一定要把樣本畫出來，並對照新抽的資料檢查模態涵蓋、模態內離散程度與離群值。

DDIM：同樣的訓練，不同的取樣器

Song、Meng 與 Ermon [5] 注意到 $\mathcal{L}_{\text{simple}}$ 只依賴邊際分布 $q(x_t \mid x_0)$，而不依賴產生它們的馬可夫鏈。許多非馬可夫的前向過程擁有相同的邊際分布，每一個都給出不同的反向取樣器——而且全部都能用同一個訓練好的 ϵ_θ。對任何遞減的步數子序列 $\tau_1 > \tau_2 > \cdots$，**DDIM** 的更新是

$$x_{\tau_{i+1}} = \sqrt{\bar\alpha_{\tau_{i+1}}}\,\hat{x}_0 + \sqrt{1-\bar\alpha_{\tau_{i+1}}-\sigma^2}\,\epsilon_\theta(x_{\tau_i},\tau_i) + \sigma z, \qquad \sigma = \eta\sqrt{\frac{1-\bar\alpha_{\tau_{i+1}}}{1-\bar\alpha_{\tau_i}}\Big(1-\frac{\bar\alpha_{\tau_i}}{\bar\alpha_{\tau_{i+1}}}\Big)},$$

其中 $\hat{x}_0$ 是在 x_{τ_i} 的乾淨估計，$\eta \ge 0$ 控制隨機性。讀法是：「先估計乾淨樣本，再沿著預測的雜訊方向，把它重新加噪到下一個較低的雜訊程度。」當 $\eta = 1$ 且步數連續時，它還原 DDPM 的後驗變異數；當 $\eta = 0$ 時，取樣器是**確定性**的，也是機率流 ODE 的一種離散化。確定性有實際影響：同樣的起始雜訊永遠得到同樣的輸出，所以可以在雜訊空間內插，或把真實影像反推回它的雜訊（DDIM inversion），這是許多影像編輯方法的基礎。

以下獨立範例重複相同的小型訓練設定，讓程式能單獨執行。在同一工作階段實驗時，可直接重用已訓練的 `net`、`abar` 與 `T`：

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
T = 100
beta = torch.linspace(.001, .15, T)
alpha = 1-beta
abar = alpha.cumprod(0)
previous = torch.cat([torch.ones(1), abar[:-1]])
post_var = beta * (1-previous) / (1-abar)
net = nn.Sequential(nn.Linear(3, 64), nn.SiLU(),
                    nn.Linear(64, 64), nn.SiLU(), nn.Linear(64, 2))
opt = torch.optim.Adam(net.parameters(), lr=.002)
def data(n):
    side = torch.randint(2, (n, 1))*2-1
    return torch.cat([side*1.5, torch.zeros(n, 1)], 1) + .25*torch.randn(n, 2)
for step in range(3000):
    x0 = data(128)
```

```
    t = torch.randint(T, (len(x0),))
    eps = torch.randn_like(x0)
    a = abar[t, None]
    xt = a.sqrt()*x0 + (1-a).sqrt()*eps
    predicted = net(torch.cat([xt, t[:, None].float()/(T-1)], 1))
    loss = F.mse_loss(predicted, eps)
    opt.zero_grad(); loss.backward(); opt.step()
net.eval()

def ddim_sample(n, steps, eta=0.0, seed=1):
    g = torch.Generator().manual_seed(seed)
    taus = torch.linspace(T-1, 0, steps).round().long()
    x = torch.randn(n, 2, generator=g)
    with torch.no_grad():
        for i, t in enumerate(taus):
            time = torch.full((n, 1), t.item()/(T-1))
            eps = net(torch.cat([x, time], 1))
            a_t = abar[t]
            a_prev = abar[taus[i+1]] if i+1 < steps else torch.tensor(1.)
            x0_hat = (x - (1-a_t).sqrt()*eps) / a_t.sqrt()
            sigma = eta*((1-a_prev)/(1-a_t)*(1-a_t/a_prev)).sqrt()
            x = (a_prev.sqrt()*x0_hat + (1-a_prev-sigma**2).sqrt()*eps
                 + sigma*torch.randn(n, 2, generator=g))
    return x

for steps in [5, 20, 100]:
    s = ddim_sample(2000, steps)
    left = s[s[:, 0] < 0]
    print(f"{steps:3d} steps: left fraction {(s[:, 0] < 0).float().mean():.3f}, "
          f"left-cluster mean x {left[:, 0].mean():.2f}, std {left[:, 0].std():.2f}")
```

輸出呈現典型的規律。用 5 步時，左群集中心在 −1.11、標準差 0.39——樣本被拉向中間並被抹開，因為大步長是用直線去走一條彎曲的軌跡。用 20 步時是 −1.37 與 0.30；100 步時是 −1.42 與 0.24，接近資料。改變步數完全不需要重新訓練。

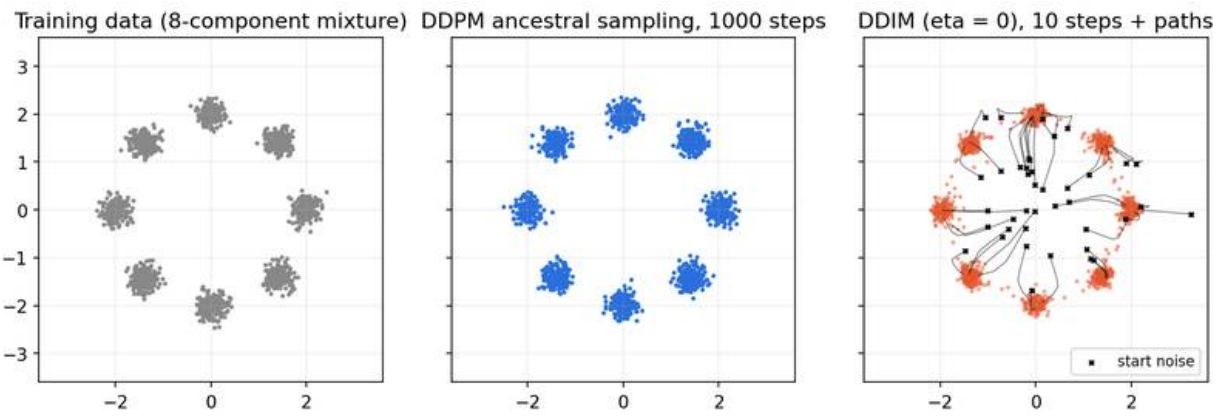


圖 3 — 取樣器都由八成分混合的精確去雜訊器驅動，所以只有取樣規則不同。*1000* 步 *DDPM* 重現了資料；*10* 步確定性 *DDIM* 落在模態附近，但群集略寬、被抹開。黑線是從起始雜訊（×）出發的 *DDIM* 軌跡。

更快的求解器

一旦取樣等同於「解一條 ODE」，數值分析就派得上用場。Karras 等人（EDM）[20] 使用二階 Heun 求解器搭配仔細選擇的雜訊程度間距，並證明取樣器、排程、預條件與訓練雜訊分布可以分開調整。DPM-Solver [21] 利用擴散 ODE 的半線性結構——線性部分可以精確積分——在大約 10 次網路評估內達到良好品質。隨機取樣器在每一步加一點新雜訊；這需要更多步，但可以修正累積的誤差。無論用哪種求解器，都要報告**函數評估次數（number of function evaluations, NFE）**：網路被呼叫的次數才是真正的取樣成本，而不是「步數」。

操控生成：條件與引導

白話版。條件指定所需輸出，引導調整取樣偏向條件的強度；更強引導不一定產生更好影像。

條件化

條件模型學習 $\epsilon_\theta(x_t, t, c)$，其中 c 是條件。類別標籤透過嵌入加到時間嵌入上；文字透過**交叉注意力（cross-attention）**，讓影像特徵去關注文字編碼器的 token [7]；影像（用於超解析度、修補、編輯）則串接到輸入通道，或經由額外的編碼器送入。

分類器引導

對加噪後的邊際分布套用貝氏定理，得到 $\nabla \log p_t(x_t \mid y) = \nabla \log p_t(x_t) + \nabla \log p_t(y \mid x_t)$。Dhariwal 與 Nichol [18] 在加噪影像上另外訓練一個分類器 $p_\phi(y \mid x_t)$，把它的梯度乘上 w 加到無條件模型上。以雜訊單位表示，

$$\hat{\epsilon} = \epsilon_\theta(x_t, t) - w\,\sqrt{1-\bar{\alpha}_t}\,\nabla_{x_t} \log p_\phi(y \mid x_t).$$

尺度 $w > 1$ 大致是從 $p_t(x_t)\,p_t(y \mid x_t)^w$ 取樣——一個被銳化的分布，用多樣性換取擬真度。加上架構改進，這篇論文展示了擴散模型在 ImageNet [35] 的樣本品質上勝過 GAN [18]。缺點是需要額外的分類器，而且必須在每個雜訊程度的雜訊輸入上訓練。

無分類器引導（CFG）

Ho 與 Salimans [6] 拿掉了分類器。訓練時，隨機一部分樣本的條件被換成「空」標記，於是同一個網路同時學到 $\epsilon_\theta(x_t, t, c)$ 與 $\epsilon_\theta(x_t, t, \varnothing)$。貝氏定理告訴我們，分類器梯度等於條件分數與無條件分數之差，所以上式的「隱式分類器」版本是

$$\epsilon_{\text{guided}} = \epsilon_\theta(x_t, t, \varnothing) + w\big(\epsilon_\theta(x_t, t, c) - \epsilon_\theta(x_t, t, \varnothing)\big).$$

在這個慣例下，$w = 0$ 是無條件，$w = 1$ 是普通的條件模型，$w > 1$ 則外插超過它。（有些論文把同一件事寫成 $(1 + w')\epsilon_c - w'\epsilon_u$，因此他們的「尺度」是我們記號中的 $w - 1$；比較數字之前一定要先確認慣例。）

下面的程式使用四團混合分布的精確雜訊預測器，因此任何奇怪的行為都來自引導本身，而不是網路沒訓練好。類別 1 是 $x > 0$ 的那兩團。

```
import numpy as np

# Data: four Gaussian blobs at the corners of a square; class c is the sign of x.
MU = np.array([[-1.5, -1.5], [-1.5, 1.5], [1.5, -1.5], [1.5, 1.5]])
SIG = 0.2
CLASS = np.array([0, 0, 1, 1])

def eps_exact(x, abar, comps):
    """Exact E[eps | x_t] for a Gaussian mixture restricted to `comps`."""
    m = np.sqrt(abar)*MU[comps]
    var = abar*SIG**2 + 1 - abar
    logw = -((x[:, None] - m[None])**2).sum(-1) / (2*var)
    w = np.exp(logw - logw.max(1, keepdims=True))
    w /= w.sum(1, keepdims=True)
    score = -(w[..., None]*(x[:, None] - m[None])).sum(1) / var
    return -np.sqrt(1-abar)*score

abar = np.cumprod(1 - np.linspace(1e-4, 0.02, 1000))
taus = np.linspace(999, 0, 50).round().astype(int)
everything, class1 = np.arange(4), np.where(CLASS == 1)[0]
print("data: mean distance to its mode", round(SIG*np.sqrt(np.pi/2), 3))
for w in [0, 1, 3, 8]:
    x = np.random.default_rng(0).normal(size=(4000, 2))
    for i, t in enumerate(taus):                    # deterministic DDIM
        e_u, e_c = eps_exact(x, abar[t], everything), eps_exact(x, abar[t], class1)
        e = e_u + w*(e_c - e_u)                     # classifier-free guidance
        x0 = (x - np.sqrt(1-abar[t])*e) / np.sqrt(abar[t])
        a_prev = abar[taus[i+1]] if i+1 < len(taus) else 1.0
        x = np.sqrt(a_prev)*x0 + np.sqrt(1-a_prev)*e
    dist = np.linalg.norm(x[:, None] - MU[None], axis=-1).min(1)
    print(f"w={w}: class-1 fraction {(x[:, 0] > 0).mean():.3f}, "
          f"mean distance to nearest mode {dist.mean():.3f}")
```

資料到其模態的平均距離是 0.251。$w = 0$ 時約一半樣本（0.490）屬於類別 1；$w = 1$ 時全部屬於類別 1，距離為 0.225（50 步 DDIM 略微低估離散程度）。$w = 3$ 時距離為 0.259，$w = 8$ 時增加到 0.446：強引導不只是「讓它更像類別 1」——它把樣本推超過資料，這是影像模型在非常高引導下出現過飽和色彩與變形的玩具版本。圖 4 在八模態混合上呈現同樣的效應。

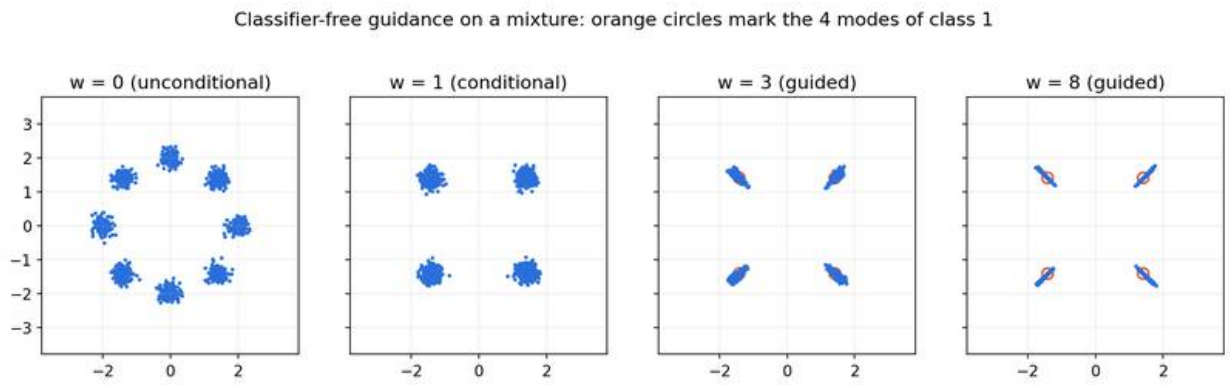


圖 *4* — 在八模態混合上的精確無分類器引導，其中類別 *1* 有 *4* 個模態（橘色圓圈）。$w = 1$ 已經把樣本限制在類別 *1*；更大的 w 讓群集在某些方向收縮、在其他方向拉長——樣本不再服從任何資料分布。

引導也有計算成本：每一步都需要一次條件預測和一次無條件預測，所以 30 步的引導取樣器需要 60 次網路評估，除非兩者被合併計算。而且 w 是取樣的旋鈕，不是影像與提示詞相符程度的校準指標。

潛在擴散與擴散 Transformer

白話版。像素很昂貴。先用自編碼器把影像壓縮成較小的特徵網格，在這個壓縮空間裡做擴散，最後再解碼。另外，去雜訊網路本身也可以是 Transformer，而不是卷積式 U-Net [37]。

潛在擴散

Rombach 等人 [7] 先訓練一個自編碼器：編碼器 $z = \mathcal{E}(x)$ 把 $H \times W \times 3$ 影像映射成 $h \times w \times c$ 的潛在表示，空間降取樣倍率為 $f = H/h$；解碼器 $\mathcal{D}$ 再映射回來。自編碼器以重建、感知與對抗損失，加上輕量的正則化（朝向 Gaussian 的小 KL 項，或向量量化）訓練，讓潛在空間保持良好性質。接著在 z 上照原樣訓練擴散模型，生成最後以 $\hat{x} = \mathcal{D}(z_0)$ 結束。當 $f = 8$ 時，512×512 影像變成 64×64 的潛在表示：每次網路呼叫的空間位置少了 64 倍。

取捨是重建瓶頸：編碼器丟棄的資訊無法由該代碼忠實找回，雖然解碼器可能合成合理細節。潛在表示的尺度也必須與擴散模型訓練時一致；實作上會把潛在表示重新縮放到大約單位變異數。[7] 的文字條件化在 U-Net 中使用交叉注意力層，這也是 Stable Diffusion 系列背後的設計。

擴散 Transformer

DiT（Peebles 與 Xie [8]）把 U-Net 換成在潛在 patch 上運作的 Vision Transformer [41]（見 ML 05）。大小為 $h \times w$ 的潛在表示被切成 $p \times p$ 的 patch，得到 hw/p^2 個 token；p 越小，token 越多、計算量越大、品質越好。時間步與類別標籤透過**自適應層正規化（adaptive layer norm）**為每個區塊提供條件：它們預測每個 LayerNorm 的縮放與平移；在「adaLN-Zero」變體中還預測一個初始化為零的閘控係數，讓每個殘差區塊一開始等於恆等映射。這篇論文的主要發現是一個縮放

趨勢：在不同模型大小與 patch 大小之間，Transformer 計算量（GFLOPs）越多，FID 就穩定地越低 [8]。

把三個選擇分開來看很有幫助。**架構**（U-Net 或 Transformer）是函數近似器。**目標**（雜訊、x_0、v 或流速度）是它要迴歸的對象。**取樣器**（DDPM、DDIM、ODE 求解器）則決定預測如何變成影像。Stable Diffusion 3 [9] 是一種新組合的例子：一個潛在 Transformer，影像與文字 token 串流使用各自的權重但共同做注意力（「MM-DiT」），以整流流目標訓練，並讓訓練時間步的分布偏向中間雜訊程度。

對影片而言，潛在表示與注意力必須同時涵蓋時間與空間，因此壓縮與注意力模式成為主要的工程問題。Melnik 等人的綜述 [31] 回顧了影片擴散的架構、時間一致性機制與評估。在影片上，逐格品質不夠：物體恆存、動作合理性與長時間一致性都需要各自的評估。

像素空間、潛在空間與表徵自編碼器

白話版。生成器需要一個反覆更新的空間。它可以是影像本身、緊湊的重建代碼，也可以是視覺表徵模型學到的特徵。加入潛在階段，是選擇特定的計算與統計取捨，而不是生成模型必備的條件。

為什麼像素要變成潛在表徵，再變回像素？

訓練影像以像素形式進來，輸出也必須是可觀看的像素。在兩端之間，編碼器可以去除冗餘，產生適合反覆更新、成本較低的網格 [7]。先用重建訓練編碼器與解碼器，再對編碼後的訓練影像學習生成分布。兩項工作不同：重建保留某個指定樣本；生成學習哪些編碼樣本合理。

訓練與取樣路徑如下：

$$\text{training: } x \xrightarrow{\mathcal{E}} z_{\text{data}} \xrightarrow{\text{corrupt}} z_t \xrightarrow{\text{prediction loss}} \theta,$$

$$\text{generation: } z_{\text{noise}} \xrightarrow{\text{learned sampler}} \hat{z} \xrightarrow{\mathcal{D}} \hat{x},$$

$$\text{reconstruction check: } x \xrightarrow{\mathcal{E}} \mathcal{E}(x) \xrightarrow{\mathcal{D}} \tilde{x}.$$

這裡 x 是訓練影像；$\mathcal{E}$ 與 $\mathcal{D}$ 是編碼器與解碼器；z_{data} 是編碼後的影像；z_t 是其受污染狀態；θ 是生成預測器的參數；$\hat{z}$ 是生成的代碼；$\hat{x}$ 是生成影像；$\tilde{x}$ 是重建影像。純文字到影像取樣**不需要**先編碼一張輸入影像。編輯與影像到影像生成則可以編碼實際輸入，從部分污染的代碼開始。

主要節省來自去雜訊器或流預測器會反覆執行，而編碼訓練資料與解碼樣本的次數相對少。空間縮減因子 f 使網格位置減少 f^2 倍，但潛在通道數同樣重要。三通道影像與 c 通道潛在表徵的純量數量縮減比例是 $3f^2/c$。不能把網格縮減比例直接說成所有浮點數量或總執行時間的縮減比例。

對 Transformer，還必須區分詞元數與詞元寬度。完整注意力儲存的矩陣元素數與 N^2 成正比，其中 N 是詞元數；特徵投影也受詞元寬度影響 [41]。像素與潛在空間都能以區塊大小降低 N。這些數量本身不能預測實際速度，因為運算核心、記憶體傳輸、解碼器與條件化也有成本。

```
# Synthetic shape bookkeeping, not a benchmark or a specific checkpoint.
H = W = 512
```

```
channels, spatial_factor, patch = 4, 8, 2
pixel_values = H*W*3
h = H//spatial_factor
w = W//spatial_factor
latent_values = h*w*channels
pixel_tokens = (H//patch)*(W//patch)
latent_tokens = (h//patch)*(w//patch)
print("Spatial positions reduction:", (H*W)//(h*w))
print("Scalar values reduction:", pixel_values//latent_values)
print("Dense attention entries reduction:", pixel_tokens**2//latent_tokens**2)
assert pixel_values//latent_values == 48
assert pixel_tokens//latent_tokens == 64
assert pixel_tokens**2//latent_tokens**2 == 4096
```

此人工例子得到空間位置少 64 倍、純量少 48 倍、注意力矩陣元素少 4096 倍，條件是**兩個座標空間都採用數值相同的區塊大小**。改變區塊大小，比較結果就會改變。像素 Transformer 可以使用大得多的區塊，得到與潛在 Transformer 相同的詞元數。

代價是資訊流失。若兩張影像具有相同代碼，確定性的解碼器無法還原各自不同的精確細節。解碼器可以補出合理紋理，但那沒有找回原始影像的資訊。因此，生成訓練之前應獨立評估重建。檢查細線、文字與跨影格變化，而不只看平均影像誤差。

下例的簡單編碼器對區塊取平均，解碼器重複這些平均值。它示範瓶頸，不宣稱重現學習式 VAE 或 RAE。

```
import numpy as np

y, x = np.mgrid[:64, :64]
checker = ((x+y) % 2).astype(float)
smooth = x/63
image = .6*smooth + .4*checker
f = 4
latent = image.reshape(64//f, f, 64//f, f).mean(axis=(1, 3))
reconstruction = np.repeat(np.repeat(latent, f, axis=0), f, axis=1)
print("Synthetic reconstruction MSE:", round(np.mean((image-reconstruction)**2), 6))
assert np.mean((image-reconstruction)**2) > .04
# Recovering the exact latent does not recover the missing checkerboard.
assert np.allclose(reconstruction[0, :f], reconstruction[0, 0])
```

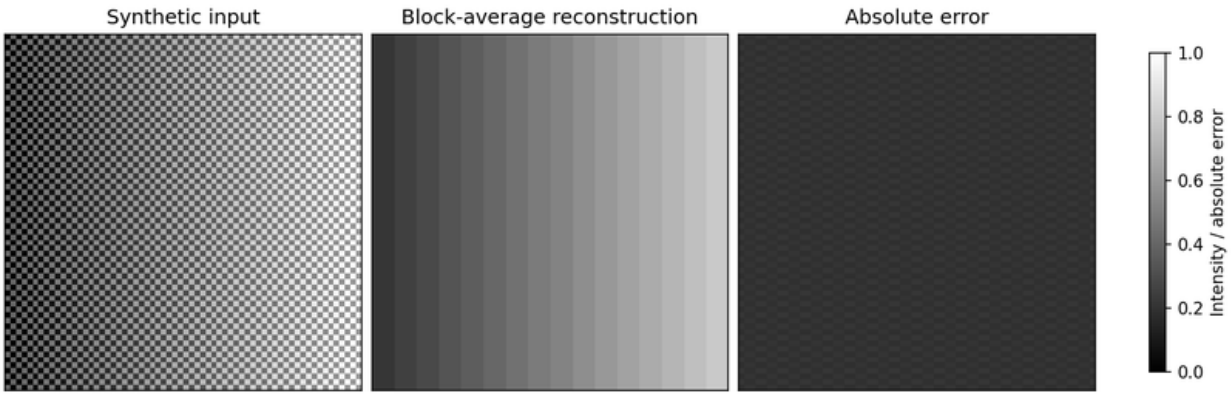


圖 5 — 潛在瓶頸可能在生成模型訓練前就移除細節。區塊平均範例用來解釋資訊流失，並非學習式自編碼器的評估。

在編輯時，反覆編碼與解碼結果可能累積失真。若流程允許，可將中間編輯保留在相容的潛在空間，並檢查最後的像素輸出。優良的無條件生成器仍可能不擅長重建。自編碼器的上限指的是對編碼資訊的忠實還原，並不表示生成解碼器永遠無法合成新的細緻紋理。

重建代碼與語意表徵代碼

VAE 編碼器以重建目標與分布正則化訓練，其代碼設計是為了支援解碼與取樣 [7]。預訓練辨識編碼器則學習有用的視覺特徵。兩者目標可以一致，但辨識所需的不變性也可能捨棄忠實重建需要的細節。解碼器與保留下來的特徵結構，決定表徵是否適合成為生成空間。

Zheng、Ma、Tong 與 Xie 的**表徵自編碼器（Representation Autoencoders，RAEs）**[43]，把凍結的預訓練表徵編碼器搭配可訓練解碼器。其 ICLR 2026 研究以語意視覺特徵作為生成模型的空間，檢查高維詞元特徵、依維度調整的雜訊排程、解碼器穩健性，以及淺而寬的擴散輸出頭。這改變了模型生成的表徵，而不只是對傳統潛在預測器加入特徵對齊損失。

不要把「潛在」等同於「每一維都更小」。空間稀疏的特徵網格仍可能有很多通道。RAE 也不會自動成為變分模型：凍結特徵擷取器加上解碼器，本身並未定義 Gaussian 後驗、KL 正則化或概似下界。應分別測量重建、表徵品質與生成品質。

Tong 等人 [44] 在 2026 年將此方法延伸到自由形式的文字到影像生成。研究在更大規模比較 RAE 與 VAE 空間，並檢查解碼器資料組成。結果顯示，依維度調整的雜訊排程仍重要，而某些小規模架構與解碼器調整在其較大規模設定下幫助有限。這提醒我們在規模與資料變化時重新測試配方，不能把 ImageNet 結果當成普遍設計規則。

對新應用，檢查所選編碼器是否保留重要內容：文件的細小文字、檢測影像的小缺陷，或影片的影格間差異。用保留的真實特徵與生成特徵分別評估解碼器。只用精確訓練代碼學習的解碼器，可能對取樣代碼的小幅分布外誤差敏感。依訓練配方正規化特徵，並將編碼器版本與生成檢查點一起記錄。

JiT：在像素上直接預測乾淨影像

Li 與 He 的 **Just image Transformers（JiT）**[42] 在 2025 年發布預印本，並於 CVPR 2026 發表，重新檢查直接操作像素的大區塊 Transformer。網路預測乾淨影像，再由該預測推得取樣用的流速度。其控制實驗支持以流形解釋的觀點：瓶頸能保留有結構的乾淨影像資訊，卻可能讓直接預測高維雜訊或速度變困難。該實驗設定不使用預訓練潛在詞元化器或額外表徵損失。這是特定架構與訓練條件的證據，不是像素模型總能勝過潛在生成器的證明。

關鍵是區分**網路直接輸出**與**損失或求解器使用的量**。對後面定義的直線路徑，使用 $y_t = (1-t)\epsilon + tx$，$t = 0$ 是雜訊，$t = 1$ 是乾淨資料。直接預測 $\hat{x}_\theta(y_t, t)$ 的網路可以提供：

$$\hat{u}_\theta(y_t, t) = \frac{\hat{x}_\theta(y_t, t) - y_t}{1-t}, \qquad \|\hat{u}_\theta - (x-\epsilon)\|^2 = \frac{\|\hat{x}_\theta - x\|^2}{(1-t)^2}.$$

這裡 x 是乾淨影像；ϵ 是抽出的雜訊；y_t 是兩者內插；$\hat{x}_\theta$ 是直接預測；$\hat{u}_\theta$ 是對應的傳輸速度。恆等式適用於 $t < 1$ 且尚未進行數值截限的情況。因此，輸出 x 的網路可以用速度空間的損失訓練。稱為「乾淨預測」不代表損失是無加權的乾淨影像 MSE。JiT 的釋出實作清楚呈現此差異 [42]。

$t = 1$ 附近，除以 $1 - t$ 必須有端點慣例與數值保護。就內部時間的代數而言，固定乾淨影像誤差越靠近端點，放大程度越高。取樣時應使用檢查點的時間分布、雜訊縮放與求解器配方。只改直接輸出的標籤，無法重現方法。

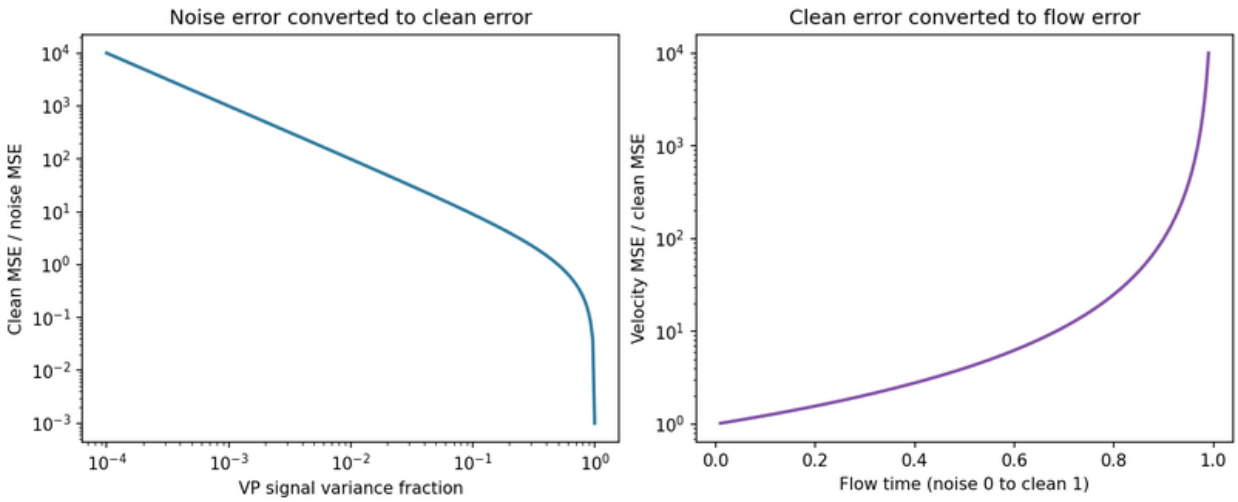


圖 6 — 等價預測使用不同單位。左圖是低訊號下的雜訊誤差放大；右圖是接近乾淨端點時的速度誤差放大。曲線是代數恆等式，而非模型效能測量。

有意義的比較應固定訓練資料、計算量、解析度、詞元數與取樣器，再逐一改變表徵或目標。依重建需求、生成品質與總成本，比較像素 Transformer、緊湊 VAE 潛在 Transformer 與表徵潛在 Transformer。記錄編碼器／解碼器成本與表徵預訓練。即使生成階段從零開始，使用預訓練視覺編碼器的模型仍承接了先前的訓練。

僅線上版：*JiT：在像素空間直接預測乾淨影像* — *https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/jit 在像素空間直接預測乾淨影像/*

流匹配：學習一個傳輸速度

白話版。對直線條件路徑，配對雜訊與資料並迴歸速度；生成沿學得的速度場前進，軌跡未必筆直。

時間慣例改變。本節的時間從 $t = 0$ **的雜訊走到** $t = 1$ **的資料**，與前面擴散各節相反（前面 t 越大代表雜訊越多）。

條件流匹配目標

連續正規化流用一條 ODE $dx_t/dt = v(x_t, t)$ 移動點。如果速度場 v 正確，它會把雜訊分布 $p_0 = \mathcal{N}(0, I)$ 傳輸到資料分布 p_1。Lipman 等人 [10]（以及同時期的 Liu 等人 [11]、Albergo 與 Vanden-Eijnden 的隨機內插 [34]）展示了如何在完全不模擬 ODE 的情況下訓練 v。

把雜訊 $z \sim \mathcal{N}(0, I)$ 與資料 $x \sim p_1$ 配對，定義直線的**條件路徑**

$$x_t = (1 - t)\, z + t\, x, \qquad \frac{dx_t}{dt} = u_t(x_t \mid z, x) = x - z.$$

每一對都以等速沿直線移動。我們真正想要的邊際速度，是通過該點的所有配對之條件速度的平均：

$$u_t(y) = \mathbb{E}[\, x - z \mid x_t = y \,].$$

我們算不出這個平均，但其實也不需要。比較「理想」損失與我們算得出來的損失：

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E}_{t, x_t} \| v_\theta(x_t, t) - u_t(x_t) \|^2, \qquad \mathcal{L}_{\mathrm{CFM}} = \mathbb{E}_{t, z, x} \| v_\theta(x_t, t) - (x - z) \|^2,$$

其中 $t \sim \mathcal{U}[0, 1]$。把兩個平方展開。$\|v_\theta\|^2$ 項完全相同。交叉項也相等，因為由重複期望值定律，$\mathbb{E}[v_\theta(x_t, t)^\top (x - z)] = \mathbb{E}[v_\theta(x_t, t)^\top \, \mathbb{E}[x - z \mid x_t]] = \mathbb{E}[v_\theta^\top u_t(x_t)]$。剩下的項都不含 θ。因此

$$\nabla_\theta \mathcal{L}_{\mathrm{CFM}} = \nabla_\theta \mathcal{L}_{\mathrm{FM}},$$

也就是說，迴歸每一對的速度 $x - z$ 就能訓練出邊際速度場。這和擴散模型是同一個技巧：一個對單一樣本而言充滿雜訊的目標，其條件平均卻是對的。生成時用任何 ODE 求解器，從 $t = 0$ 到 $t = 1$ 解 $dx/dt = v_\theta(x, t)$。

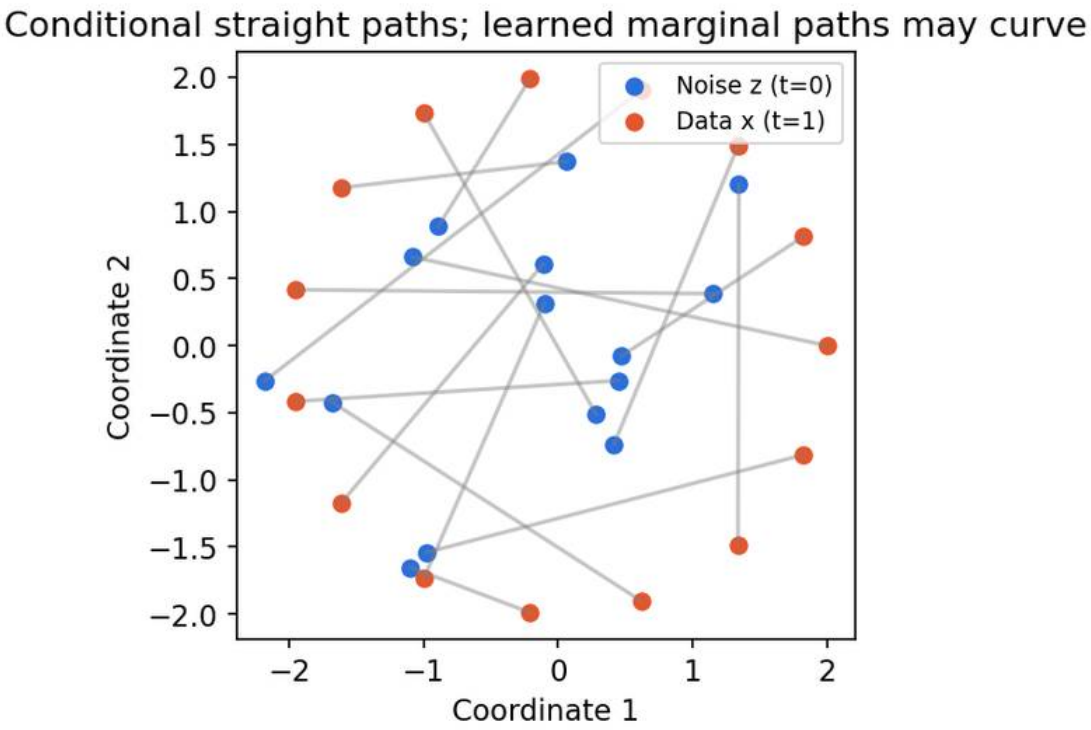


圖 7 — 條件訓練路徑是直線。不同配對的路徑會交叉，所以交叉點的邊際速度是平均值，學到的軌跡可能彎曲。

流匹配與擴散的關係

線性路徑是一條 Gaussian 路徑：給定 x，$x_t \sim \mathcal{N}(t\,x,\ (1-t)^2 I)$。記 $\hat{z} = \mathbb{E}[z \mid x_t]$（一個雜訊預測），並利用 $x = (x_t - (1-t)z)/t$，

$$u_t(x_t) = \frac{x_t - \hat{z}}{t}, \qquad \nabla \log p_t(x_t) = -\frac{\hat{z}}{1-t}, \qquad \text{因此} \qquad u_t(x_t) = \frac{x_t + (1-t)\,\nabla \log p_t(x_t)}{t}.$$

所以對這條路徑而言，流匹配速度、雜訊預測器與分數仍然可以互相換算；不同的是路徑（直線或 VP 曲線）、損失對雜訊程度的加權，以及時間慣例。擴散模型的機率流 ODE 本身就是沿著某條特定 Gaussian 路徑的流。Lipman 等人的指南 [2] 與隨機內插框架 [34] 都把兩者視為同一個建構的特例。實務上的建議與參數化時相同：這些目標是靠推導彼此關聯，而不是改個名字就能互換。

程式：兩個群集上的流匹配

這個獨立範例用線性條件路徑學習與 DDPM 程式相同的兩群集分布，再用步長 $1/N$ 的 Euler 法取樣。

```
import torch
```

```
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
net = nn.Sequential(nn.Linear(3, 64), nn.SiLU(),
                    nn.Linear(64, 64), nn.SiLU(), nn.Linear(64, 2))
opt = torch.optim.Adam(net.parameters(), lr=.002)
for step in range(3000):
    z = torch.randn(128, 2)
    side = torch.randint(2, (128, 1))*2-1
    x = torch.cat([1.5*side, torch.zeros(128, 1)], 1) + .25*torch.randn(128, 2)
    t = torch.rand(128, 1)
    xt = (1-t)*z+t*x
    loss = F.mse_loss(net(torch.cat([xt, t], 1)), x-z)
    opt.zero_grad(); loss.backward(); opt.step()
net.eval()
def euler_sample(n, steps, seed=1):
    x = torch.randn(n, 2, generator=torch.Generator().manual_seed(seed))
    with torch.no_grad():
        for k in range(steps):
            t = torch.full((n, 1), k/steps)
            x = x + net(torch.cat([x, t], 1))/steps
    return x

for steps in [1, 4, 50]:
    s = euler_sample(2000, steps)
    assert torch.isfinite(s).all()
    left = s[s[:, 0] < 0]
    print(f"{steps:2d} Euler steps: left fraction {(s[:, 0] < 0).float().mean():.3f}, "
          f"left-cluster mean x {left[:, 0].mean():.2f}, std {left[:, 0].std():.2f}")
```

對 1、4、50 步，它印出左群集平均 -0.18、-1.20、-1.45，標準差 0.12、0.42、0.29。一步 Euler 幾乎落在整體資料平均：在 $t = 0$ 時邊際速度是 $\mathbb{E}[x] - z$，所以一整步會把每一個雜訊點都送到 $\mathbb{E}[x]$。直線的條件路徑並不會讓學到的邊際流變直。圖 8 用八模態混合的精確速度呈現同樣的效應。

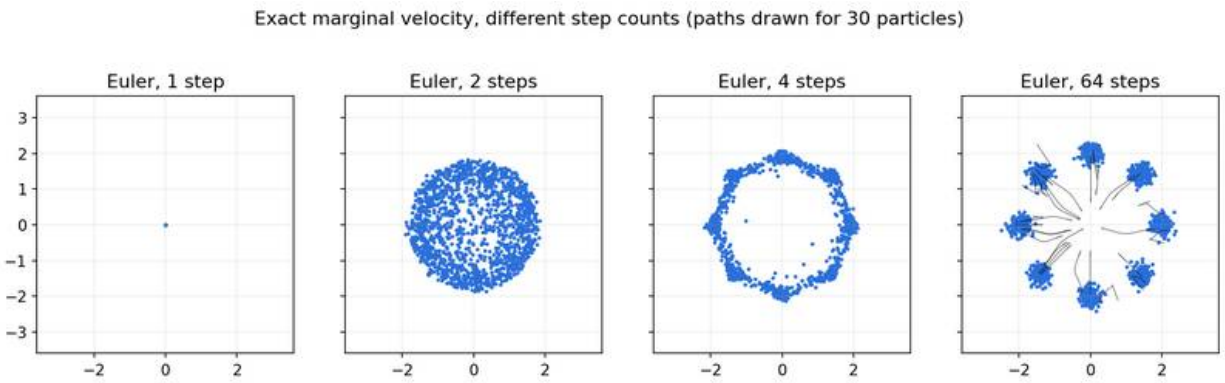


圖 *8* — 即使使用精確的邊際速度，*Euler* 取樣仍需要許多步：一步會把所有樣本塌縮到資料平均，*64* 步的軌跡（黑線）在路徑後段才轉向各模態。

整流流與 reflow

Liu、Gong 與 Liu [11] 使用同樣的直線內插（「整流流」），並加上 **reflow**：訓練後，從每個 z 解學到的 ODE 產生配對 $(z, \hat{x})$，再用這些已耦合的配對（而非獨立配對）重新訓練。因為新配對本來就由流相連，它們的直線較少交叉，學到的軌跡變得更直，需要的 Euler 步數也更少；夠直的流還可以再蒸餾成一步模型。Stable Diffusion 3 [9] 證明了整流流訓練的大型潛在 Transformer 在文字生成影像的規模上可行。

僅線上版：*Flow Matching*：先迴歸速度，再積分生成 — *https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/flow-matching 先迴歸速度再積分生成/*

Flow Matching 之後：區間、影像輸出與一步像素生成

白話版。學會流之後，有兩個不同問題：一次預測能否涵蓋整段區間？網路又該直接輸出什麼？MeanFlow 改變所建模的區間，JiT 改變輸出表示，Pixel MeanFlow 再把兩個方向接起來。依這個順序讀，不要把每個新名字都當成另一種取樣器。

先對齊時間方向

前面的 flow-matching 範例從零端雜訊走到一端資料；MeanFlow [13]、improved MeanFlow [49] 與 pixel MeanFlow [50] 採相反慣例：

$$z_t = (1-t)x + t\epsilon, \qquad t = 0:\ \text{data}, \quad t = 1:\ \text{noise}.$$

其中 x 為乾淨訓練範例，ϵ 為先驗雜訊，z_t 是插值訓練狀態。生成從一走到零。與 JiT 的時間 s 比較時，使用 $s = 1 - t$ 並反轉速度正負號。這是座標改變，不是不同生成原理。建立訓練配對的條件直線插值，也仍不代表學得的邊際 ODE 軌跡必為直線。

MeanFlow：把切線改成區間平均

瞬時速度告訴你**現在**往哪裡走；MeanFlow [13] 學習整段區間的平均，讓一次預測描述完整位移：

$$u(z_t, r, t) = \frac{1}{t-r}\int_r^t v(z_s, s)\, ds, \qquad z_r = z_t - (t-r)u(z_t, r, t).$$

其中 v 為邊際瞬時速度，z_s 位於其 ODE 軌跡，$r < t$ 為區間端點，u 為該軌跡上的平均速度。零長邊界使用極限 $u = v$。積分定義所要學的量；訓練並不對每筆資料積分求出教師軌跡。

對定義微分可得 $u = v - (t-r)du/dt$。全導數沿移動狀態計算，並固定 r，不只是對時間輸入做偏微分。原始配方使用 Jacobian-vector product，以及由當前網路建立、阻斷梯度的目標。只要學得平均夠準，完整區間 $(r, t) = (0, 1)$ 就能用於一次呼叫生成。

以下精確曲線示範兩者差異；它是代數檢查，不是訓練好的 MeanFlow 模型：

```
r, t = 0.0, 1.0
```

```
start = (t, t*t)                    # curve z(t) = (t, t^2)
instant = (1.0, 2*t)                # tangent at the starting time
average = (1.0, t+r)                # exact interval-average derivative
one_euler = tuple(z-(t-r)*v for z, v in zip(start, instant))
one_average = tuple(z-(t-r)*u for z, u in zip(start, average))
assert one_average == (r, r*r)
print("Instantaneous Euler jump:", one_euler)
print("Average-velocity jump:", one_average)
```

輸出為 `(0.0, -1.0)` 與 `(0.0, 0.0)`。正確局部切線在一次大步後仍可能錯過終點，正確區間平均則能到達；難點就在於學好那個平均。

僅線上版：*MeanFlow*：學習整段時間的平均速度 — *https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/meanflow 學習整段時間的平均速度/*

Improved MeanFlow：明確處理預測輸入與引導

iMF [49] 把訓練改寫成瞬時速度損失，網路仍輸出平均速度。關鍵改動位於導數計算內：原始配方把抽樣條件速度送入 JVP，iMF 改用網路預測的邊際速度，使乾淨訓練標籤不再成為複合預測器輸入。實作可重用零長邊界，或在訓練時使用輔助速度頭。

另一項改進把引導尺度與作用區間設為明確條件，使同一模型在推論時可調整設定，同時維持一次呼叫。這不同於事後加上有條件／無條件兩次網路計算。in-context 條件 token 也改變模型大小，因此最終系統比較不是單純的損失函數消融。

僅線上版：*Improved MeanFlow*：預測輸入與彈性引導 — *https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/improved-meanflow 預測輸入與彈性引導/*

JiT：影像輸出是另一條設計軸線

前面的 JiT 說明 已區分直接乾淨影像輸出與速度空間損失 [42]。採此處時間慣例，導出的瞬時速度為 $(z_t - \hat{x}_\theta)/t$，其中 $\hat{x}_\theta$ 為預測乾淨影像；靠近零時須小心處理。JiT 仍沿多步軌跡取樣；輸出形狀像影像，不會自動提供區間平均速度或一步樣本。

其控制比較特別有用：固定損失空間，再改直接輸出與區塊維度。它說明目標間可做代數轉換，不代表有限容量網路都同樣好訓練。不要只比較訓練配方不同的最強摘要模型。

僅線上版：*JiT*：在像素空間直接預測乾淨影像 — *https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/jit 在像素空間直接預測乾淨影像/*

Pixel MeanFlow：把影像輸出接到區間場

Pixel MeanFlow [50] 定義雙時間影像狀場 $x^\star(z_t, r, t) = z_t - t\,u(z_t, r, t)$。其中 $x^\star$ 由平均速度場誘發，不一定等於與 z_t 配對的乾淨訓練影像。網路以 x_θ 預測此場，再轉回 $u_\theta = (z_t - x_\theta)/t$。轉換使用 $t > 0$，並需端點保護。

導出的 u_θ 進入 iMF 瞬時速度損失：**影像輸出 → 平均速度 → 瞬時速度損失**。在完整區間，更新簡化成網路的影像輸出，便能一次呼叫生成像素。中間區間的影像狀流形解釋，是由視覺化及消融支持的假設，不是精確流形保證。

生成時不需要潛在 tokenizer，但較強配方使用預訓練特徵網路計算感知訓練損失。與 JiT 或潛在生成器比較時，須記錄此訓練依賴。「沒有解碼器」「一個 NFE」與「任何元件都沒有預訓練」是三種不同主張。

僅線上版：*Pixel MeanFlow*：透過去雜訊影像場一步生成像素 — *https://blog.shyandram.dev/zh-tw/learn/ml/diffusion-flow-matching/papers/pixel-meanflow* 透過去雜訊影像場一步生成像素/

區分相鄰方法

Reflow [11] 改變訓練配對的耦合，使軌跡更直。Consistency models [12] 學習同一軌跡上終點預測的一致性；有教師蒸餾與無教師訓練是不同設定。MeanFlow 參數化區間平均，JiT 則處理網路輸出空間。少步總覽 比較這些相鄰家族。

選方法前，先寫出表示空間、直接輸出、損失目標、時間方向及取樣呼叫數，再固定解析度與條件比較；區分生成器訓練及表示預訓練，適用時也算入解碼器成本。NFE 較低是呼叫次數的有用證據，不是完整延遲或品質評估。

預訓練影像與影片生成器是完整流程

白話版。預訓練生成器是一組相容元件。只載入去雜訊器權重，還不能確定表徵、文字條件、預測目標與取樣排程正確。從影像跨到影片，也會改變重建必須保留的內容。

執行檢查點前，先閱讀元件

實務流程結合文字編碼器、可選的影像／影片編碼器、時間條件預測器、取樣器，以及可選的解碼器。初次生成之後，可能還有獨立的細化模型。每個元件都預期特定張量形狀與縮放。排程器宣告 `v_prediction` 時，可能指 VP 速度；流排程器則可能預期直線內插的導數。應檢查訓練路徑，而不是從設定字串猜測。

以下已核實的例子呈現架構差異。它們是固定的研究基準，不是最新產品排行榜。參考文獻包含論文或技術報告，以及官方釋出文件。

基準	生成表徵與預測器	預訓練流程需要檢查的項目
SDXL [45]	重建潛在表徵與卷積去雜訊器	文字條件、潛在縮放、可選的基礎／細化模型組合與相容取樣設定
Stable Diffusion 3 [9]	潛在整流流與聯合文字／影像 Transformer	流的時間慣例、文字串流、時間抽樣與自編碼器相容性
FLUX.1 [46]	潛在整流流 Transformer	確切釋出版本、引導行為、文字條件、自編碼器與釋出條款
JiT [42]	直接在像素上做類別條件乾淨影像預測	影像正規化、區塊／解析度設定、推得的速度與取樣配方；不是自由文字到影像檢查點

基準	生成表徵與預測器	預訓練流程需要檢查的項目
HunyuanVideo [47]	壓縮影片潛在表徵與流匹配 Transformer	影片自編碼器、影像／影片聯合訓練、時間維度與文字條件元件
Wan2.1 [48]	影片擴散 Transformer 與時空自編碼器	文字到影片或影像到影片版本、因果影片壓縮、時間長度與解碼快取

SDXL [45] 說明較大的潛在影像生成架構與可選細化階段。Stable Diffusion 3 [9] 示範相同的大方向如何改用整流流與聯合文字／影像注意力。FLUX.1 的官方模型卡 [46] 描述釋出的整流流 Transformer，`.1-dev` 版本採用引導蒸餾。這些特性不代表檢查點可以互換。一般雙次推論的 CFG 與蒸餾後的引導輸入，可能有不同含義與成本。

影像到影像生成會編碼輸入，並加入部分雜訊，在保留原始內容與改變能力間取捨。實際雜訊程度由排程決定，不只是介面上「strength」這類標籤。比較編輯時，保留相同輸入，記錄初始時間、種子、提示、引導與重建基準。否則可能把重建偽影誤當成文字要求的編輯。

影片加入時間壓縮與一致性問題

令影片有 F 個影格、高度 H 與寬度 W。假想編碼器以 f_t 壓縮時間，以 f_s 壓縮兩個空間軸，約得到 $(F/f_t) \times (H/f_s) \times (W/f_s)$ 個潛在位置。確切形狀取決於填補與首影格的特殊處理。切分區塊後，跨所有位置的密集注意力成本隨剩餘詞元數平方成長；分別壓縮時間與空間，因此可能大幅改變成本。

HunyuanVideo 技術報告 [47] 結合 3D 影片自編碼器、流匹配 Transformer 與分階段影像／影片訓練。Wan 報告 [48] 說明因果影片 VAE，包括首影格特殊處理、空間與時間壓縮，以及分塊編碼／解碼的特徵快取。自編碼器具因果性，不代表整個影片生成器就以自回歸方式取樣；仍須另外檢查生成注意力與取樣程序。

單一影格可能很漂亮，但整段影片可能改變物體身分、突然反轉運動，或忘記被遮蔽的物體。生成前先測試保留影片的重建，再評估外觀、運動、提示對應與跨影格一致性。Melnik 等人 [31] 的綜述提供這些影片特有問題的閱讀地圖。單張影像指標無法診斷時間行為。

預訓練影片生成器也不會自動成為可靠的世界模型。預測合理影片與預測指定動作的後果不同。世界模型章節以狀態、動作、展開預測與評估進一步說明此差異。使用實際的條件化介面，並評估系統宣稱要解決的任務。

可重現的第一個實驗

先記錄流程清單：檢查點版本、編碼器／解碼器版本、文字編碼器、表徵形狀與縮放、預測目標、時間方向、求解器、評估次數、引導、資料型別與種子。若有自編碼器，接著做重建檢查，再對固定的小組提示與種子生成結果。一次比較一項變動。報告潛在方法成本時，納入解碼延遲與記憶體峰值。

本章 CPU 範例不下載權重，用來檢驗基本恆等式與失敗情況。完整解析度的預訓練生成，應依官方釋出說明 [42]、[45]–[48]，在另外配置資源的實驗中執行。實際測量、論文已發表結果與簡化計算，是三種不同證據。

少步與一步生成

白話版。如果取樣器需要 50 次網路呼叫，就訓練第二個模型，用幾次呼叫沿著軌跡跳躍——或者從頭訓練一個本來就設計成會跳的模型。

主要家族有：

- **漸進式蒸餾** [22]：學生模型學著用一步匹配老師的兩個確定性步驟；重複這個程序，每一輪把步數減半。
- **一致性模型** [12]：學習一個函數 $f_\theta(x_t, t)$，把機率流 ODE 軌跡上的任何一點映射到該軌跡的起點（乾淨點）。它必須滿足邊界條件：在最小時間 ϵ 時 $f_\theta(x, \epsilon) = x$，作法是寫成 $f_\theta(x, t) = c_{\text{skip}}(t)\, x + c_{\text{out}}(t)\, F_\theta(x, t)$，並令 $c_{\text{skip}}(\epsilon) = 1$、$c_{\text{out}}(\epsilon) = 0$。訓練要求同一條軌跡上相鄰兩點的輸出一致。在一致性蒸餾中，相鄰點來自預訓練擴散模型的一步 ODE 求解；在一致性訓練中，則來自同一筆資料在兩個雜訊程度下的加噪版本，不需要老師。取樣只需一步，或幾步「跳到乾淨、重新加噪、再跳」。改進的訓練方法 [32] 簡化了連續時間一致性模型，並在 ImageNet 512 × 512 上把它擴展到 15 億參數、以兩步取樣。
- **對抗式蒸餾**：ADD [33] 結合來自大型預訓練擴散老師的分數蒸餾與對抗損失，讓學生模型在一到四步內產生清晰影像。
- **平均速度模型**：MeanFlow [13] 透過平均速度學習區間位移；iMF [49] 改良迴歸與引導，pixel MeanFlow [50] 再接到影像輸出。Flow Matching 後的說明 展開這些差異及各自的預訓練依賴。

少步數是學出來的能力，而不是換個漂亮方程式就免費得到的結果。比較時要計算 NFE（包含引導評估）、潛在模型中解碼器的成本、解析度、品質以及多樣性——也要記得，一步網路每次呼叫可能比多步網路更大、更慢。

評估生成模型

白話版。「看起來好不好？」至少藏了三個問題：單一樣本是否逼真（擬真度）、樣本集合是否涵蓋資料的多樣性（多樣性）、樣本是否符合條件（對齊）。沒有任何單一數字能同時回答三者。

Fréchet Inception Distance

FID（Heusel 等人 [23]）用 Inception-v3 網路 [36] 嵌入真實與生成影像，對兩組特徵各擬合一個 Gaussian，再計算兩個 Gaussian 之間的 Fréchet（2-Wasserstein）距離：

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \operatorname{tr}\!\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}\right),$$

其中 μ_r, Σ_r 與 μ_g, Σ_g 是真實集合與生成集合的特徵平均與共變異數，tr 是矩陣的跡。越低越好。下面的程式把公式套用在二維「特徵」上。

```
import numpy as np
from scipy import linalg
```

```
def frechet_distance(a, b):
    """Frechet distance between Gaussians fitted to feature sets a and b."""
    mu1, mu2 = a.mean(0), b.mean(0)
    c1, c2 = np.cov(a, rowvar=False), np.cov(b, rowvar=False)
    covmean = linalg.sqrtm(c1 @ c2).real
    return float(((mu1 - mu2)**2).sum() + np.trace(c1 + c2 - 2*covmean))

rng = np.random.default_rng(0)
centers = np.array([[-1.5, 0.], [1.5, 0.]])
real = centers[rng.integers(2, size=5000)] + .25*rng.normal(size=(5000, 2))
good = centers[rng.integers(2, size=5000)] + .25*rng.normal(size=(5000, 2))
one_mode = centers[[0]] + .25*rng.normal(size=(5000, 2))      # mode collapse
blurry = .6*rng.normal(size=(5000, 2))                         # mass in the gap
print("good     :", round(frechet_distance(real, good), 3))
print("one mode :", round(frechet_distance(real, one_mode), 3))
print("blurry   :", round(frechet_distance(real, blurry), 3))
small = [frechet_distance(real[:50], good[i*50:(i+1)*50]) for i in range(20)]
print("good, 50 samples: mean", round(np.mean(small), 3), "std", round(np.std(small), 3))
```

正確的取樣器得到約 0.001，只會產生左邊模態的取樣器得到 3.9，把機率質量放在兩個模態之間的「模糊」取樣器得到 0.976。最後一行顯示一個實務陷阱：只用 50 個樣本時，即使完美的取樣器也得到 0.077 ± 0.073，因為樣本數少時 FID 有向上的偏差。只有在樣本數、參考集合、特徵擷取器與前處理都相同時，才能比較 FID。

FID 之外

- **精確率與召回率**（Kynkäänniemi 等人 [24]）把擬真度與多樣性分開：在特徵空間用 k 近鄰球估計真實與生成資料的「流形」；精確率是落在真實流形內的生成樣本比例，召回率則是落在生成流形內的真實樣本比例。我們的「單一模態」取樣器會有高精確率、低召回率。
- **特徵擷取器很重要**。Stein 等人 [25] 做了大規模的人類評估，發現沒有任何常用指標與人類判斷強烈相關；尤其擴散模型的感知真實度沒有反映在 FID 上。他們建議以 DINOv2-ViT-L/14 [39] 的特徵取代 Inception，並指出現有指標無法把記憶（memorization）與欠擬合或模態收縮區分開來。
- **CMMD**（Jayasumana 等人 [26]）把 Inception 特徵換成 CLIP [38] 嵌入，並把 Gaussian 假設換成使用 Gaussian RBF 核的最大平均差異（MMD），得到一個不偏且更省樣本的估計量。
- **概似**。機率流 ODE 可透過變數變換評估對數密度 [4]，但應回報數值容許誤差與散度估計方式。DDPM 的變分界限 [19] 是不同的量。把連續密度換成影像 bpd，還需要交代去量化與縮放慣例；這些分數本身都不等於視覺品質。
- **條件模型**需要對齊檢查（影像是否呈現提示詞所說的內容？），通常靠人工評分或視覺語言模型；也需要對訓練集做記憶檢查。
- **成本**。報告 NFE、引導評估次數、解析度，以及在註明硬體上的實際時間。

評估章節 ML 11 會再回到樣本數效應與誠實比較。

現代觀點

綜述與參考書。文獻很多，以下幾篇回顧是很好的地圖。

- Yang 等人 [1]（*ACM Computing Surveys*，2023）把擴散研究組織成三個改進方向——高效取樣、改進概似估計、特殊結構資料（流形、離散資料、不變性）——並涵蓋與 VAE、GAN、流的結合，以及跨領域應用。
- Croitoru 等人 [27]（*IEEE TPAMI*，2023）聚焦於視覺。他們指出三種通用公式化——DDPM、雜訊條件分數網路與 SDE——並將視覺應用分類，例如生成、超解析度、修補、編輯、轉換與分割。
- Cao 等人 [28]（*IEEE TKDE*，2024）從方法面切入：基礎公式化、演算法改進（取樣加速、目標設計、條件化），以及從影像到生物與醫療的應用。
- Lipman 等人 [2]（2024）是一份附程式碼的流匹配指南，涵蓋 Gaussian 與一般機率路徑、與擴散的關係，以及延伸到離散與黎曼資料。
- Lai、Song、Kim、Mitsufuji 與 Ermon [29]（2025）是一本專書長度的著作，把變分、分數與流三種觀點呈現為通往同一個時間相依速度場的三條路，接著討論引導、快速求解器與流映射（少步）模型。
- Shen 等人 [30]（*TMLR*，2025）從演算法層面（訓練、取樣、壓縮）、系統層面（硬體與服務部署）與框架層面綜述效率問題。
- Melnik 等人 [31]（*TMLR*，2024）綜述影片擴散：架構、時間一致性機制、資料集與評估。

2023–2026 的現況。綜合這些回顧與本章的原始論文，可以看到幾個趨勢。(1) 潛在空間中的 *Transformer*：DiT 類主幹 [8] 在許多大型系統中取代了 U-Net，而以整流流目標訓練的文字–影像聯合 Transformer [9] 成為文字生成影像的標準配方之一。(2) 流匹配成為預設目標：直線路徑的流匹配 [10]、[11] 現在與雜訊預測並用或取而代之，擴散則被理解為機率路徑的一種選擇 [2]、[29]。(3) 更少的評估次數：蒸餾 [22]、[33]、一致性模型 [12]、[32] 與平均速度模型 [13] 把取樣降到一到四次網路呼叫，品質逼近多步取樣器。(4) 更好的評估：社群正從以 Inception 為基礎的 FID 轉向更豐富的特徵空間、人類研究與對齊指標 [25]、[26]。(5) 影像之外：影片、3D、音訊與科學資料都重用同一套機制，其中影片的時間一致性與成本是主要的開放挑戰 [31]。

表徵與預測，2025–2026。 JiT [42] 測試像素上的直接乾淨影像預測；RAE [43] 測試凍結語意特徵空間中的生成；後續文字到影像研究 [44] 檢查更大規模的行為。這些研究區分模型操作的空間與網路直接預測的目標。前述預訓練例子 [45]–[48] 說明實際影像／影片流程為何需要元件相容性與時間重建。比較時應匹配計算量與資料，並納入表徵預訓練。

開放問題。高解析度與長影片的取樣成本；不犧牲多樣性、也不造成過飽和的有原則的引導；與人類判斷一致且能偵測記憶的評估指標；理解泛化（為什麼模型會產生新樣本而不是複製訓練資料）；有保證的可控性與編輯；以及資料來源、授權同意與濫用——這些是社會問題，而不只是技術問題。下一章 ML 09 介紹主要的另一條路線：在像素與 token 上的自迴歸生成，以及它與擴散的混合方法。

重點整理

- 已知前向擾動在抽取的雜訊程度上提供訓練目標。
- 簡化去雜訊目標改變變分下界各項的加權，本身不是精確概似。

- 預測目標可在指定路徑下互換，但加權、端點行為與模型容量仍會影響訓練。
- 取樣器需要相容排程與預測規約；應回報網路評估次數及完整延遲。
- 引導改變取樣分布；更強的條件遵循可能降低多樣性並引入偽影。
- 直線條件訓練路徑不保證學得軌跡筆直；應在相同評估協定下比較表示、求解器與少步訓練。

練習

1. **找回乾淨估計**。從 $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ 出發，由雜訊預測推導 $\hat{x}_0$。如果 ϵ_θ 有誤差 δ，在 $\bar{\alpha}_t = 0.25$ 與 $\bar{\alpha}_t = 10^{-4}$ 時，$\hat{x}_0$ 的誤差有多大？接著由乾淨預測推導直線流速度，說明 JiT 為何能輸出 x 卻使用速度損失。

   提示

   $\hat{x}_0 = (x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta)/\sqrt{\bar{\alpha}_t}$，所以誤差是 $\sqrt{1-\bar{\alpha}_t}\,\delta/\sqrt{\bar{\alpha}_t}$：在 $\bar{\alpha}_t = 0.25$ 時約 $1.7\,\delta$，在 10^{-4} 時約 $100\,\delta$。高雜訊時，$\hat{x}_0$ 是許多可能影像的平均，而不是一張影像。

2. **後驗平均的兩種形式**。把 $x_0 = (x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon)/\sqrt{\bar{\alpha}_t}$ 代入 $\tilde{\mu}_t$，證明它等於 $\frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon)$。接著把後驗程式中的 `t` 改成 50 與 900，確認公式仍然吻合。

   提示

   使用 $\bar{\alpha}_t = \alpha_t\bar{\alpha}_{t-1}$ 與 $\beta_t = 1-\alpha_t$。x_t 的係數變成 $\frac{\beta_t + \alpha_t(1-\bar{\alpha}_{t-1})}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)} = \frac{1}{\sqrt{\alpha_t}}$，因為分子等於 $1-\bar{\alpha}_t$。

3. **計算成本**。一個潛在文字生成影像模型用 30 步 DDIM 取樣，並以分開的條件與無條件前向計算做無分類器引導，最後跑一次解碼器。總共需要幾次去雜訊器評估？若換成一個已把引導「內建」的 4 步蒸餾模型，次數如何變化？對人工形狀範例，把潛在通道改成 16、像素區塊大小改成 16，重新計算純量與注意力矩陣元素數。哪些節省仍成立，哪些取決於區塊切分？

   提示

   60 次去雜訊器評估加一次解碼器；把兩個 CFG 分支批次處理可以減少實際時間，但不會減少計算量。內建引導的 4 步蒸餾模型需要 4 次評估加解碼器。要比較品質與多樣性，而不只是次數。

4. **時間慣例**。擴散各節使用「t 大＝雜訊多」；流匹配一節使用「$t=0$ 是雜訊、$t=1$ 是資料」。用 $s = 1-t$ 重新參數化這個流。dx/ds 是什麼？如果只改時間輸入的標籤，會出什麼錯？

   提示

   $dx/ds = -v(x, 1-s)$。如果把 s 餵給一個以 t 訓練的網路，卻沒有翻轉正負號與時間輸入，就會朝錯誤方向積分，把樣本推離資料。

5. **引導掃描**。在 CFG 程式中加入 $w \in \{0.5, 1.5, 5\}$，並記錄類別 1 樣本中落得比較靠近錯誤類別模態的比例。畫出到最近模態的平均距離對 w 的關係。引導從哪裡開始不再有幫助？

提示

在 $w = 1$ 時，精確的條件模型就已把每個樣本放進類別 1，所以更大的 w 在這裡無法提升類別正確率——只會讓群集變形。對一個學習出來、並不完美的條件模型而言，適度的 $w > 1$ 可以提升遵循度，這也是實務上使用它的原因。

6. **取樣器消融實驗**。固定訓練好的流匹配網路。把 1、2、5、20、100 步的 Euler 與二階 Heun 步驟（先用 Euler 預測，再平均兩端的速度）比較。使用相同的起始雜訊，測量群集平均與離散程度。在相同網路呼叫次數下，2 步 Heun 取樣器是否勝過 4 步 Euler？

提示

Heun 每步需要兩次網路呼叫。要畫出樣本，而不只看數字。對一個以瞬時速度訓練的模型做一步 Euler，並不能檢驗另外訓練的一步模型（例如一致性模型或 MeanFlow）。

參考文獻

1. L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui and M.-H. Yang, "Diffusion Models: A Comprehensive Survey of Methods and Applications," *ACM Computing Surveys*, 2023. arXiv
2. Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Q. Chen, D. Lopez-Paz, H. Ben-Hamu and I. Gat, "Flow Matching Guide and Code," arXiv:2412.06264, 2024. arXiv
3. J. Ho, A. Jain and P. Abbeel, "Denoising Diffusion Probabilistic Models," *NeurIPS*, 2020. arXiv
4. Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations," *ICLR*, 2021. arXiv
5. J. Song, C. Meng and S. Ermon, "Denoising Diffusion Implicit Models," *ICLR*, 2021. arXiv
6. J. Ho and T. Salimans, "Classifier-Free Diffusion Guidance," NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications; arXiv:2207.12598, 2022. arXiv
7. R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," *CVPR*, 2022. arXiv
8. W. Peebles and S. Xie, "Scalable Diffusion Models with Transformers," *ICCV*, 2023. arXiv
9. P. Esser, S. Kulal, A. Blattmann et al., "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis," *ICML*, PMLR 235, 2024. PMLR
10. Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel and M. Le, "Flow Matching for Generative Modeling," *ICLR*, 2023. arXiv
11. X. Liu, C. Gong and Q. Liu, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow," *ICLR*, 2023. arXiv
12. Y. Song, P. Dhariwal, M. Chen and I. Sutskever, "Consistency Models," *ICML*, 2023. arXiv
13. Z. Geng, M. Deng, X. Bai, J. Z. Kolter and K. He, "Mean Flows for One-step Generative Modeling," *NeurIPS*, 2025. arXiv
14. S. J. D. Prince, *Understanding Deep Learning*, MIT Press, 2023. book site
15. J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan and S. Ganguli, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," *ICML*, PMLR 37, 2015. PMLR

16. P. Vincent, “A Connection Between Score Matching and Denoising Autoencoders,” *Neural Computation*, vol. 23, no. 7, pp. 1661–1674, 2011. doi
17. Y. Song and S. Ermon, “Generative Modeling by Estimating Gradients of the Data Distribution,” *NeurIPS*, 2019. arXiv
18. P. Dhariwal and A. Nichol, “Diffusion Models Beat GANs on Image Synthesis,” *NeurIPS*, 2021. NeurIPS
19. A. Q. Nichol and P. Dhariwal, “Improved Denoising Diffusion Probabilistic Models,” *ICML*, PMLR 139, 2021. PMLR
20. T. Karras, M. Aittala, T. Aila and S. Laine, “Elucidating the Design Space of Diffusion-Based Generative Models,” *NeurIPS*, 2022. arXiv
21. C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li and J. Zhu, “DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps,” *NeurIPS*, 2022. arXiv
22. T. Salimans and J. Ho, “Progressive Distillation for Fast Sampling of Diffusion Models,” *ICLR*, 2022. arXiv
23. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium,” *NeurIPS (NIPS)*, 2017. arXiv
24. T. Kynkäänniemi, T. Karras, S. Laine, J. Lehtinen and T. Aila, “Improved Precision and Recall Metric for Assessing Generative Models,” *NeurIPS*, 2019. arXiv
25. G. Stein, J. C. Cresswell, R. Hosseinzadeh et al., “Exposing Flaws of Generative Model Evaluation Metrics and Their Unfair Treatment of Diffusion Models,” *NeurIPS*, 2023. arXiv
26. S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti and S. Kumar, “Rethinking FID: Towards a Better Evaluation Metric for Image Generation,” *CVPR*, 2024. CVF
27. F.-A. Croitoru, V. Hondru, R. T. Ionescu and M. Shah, “Diffusion Models in Vision: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. doi
28. H. Cao, C. Tan, Z. Gao, Y. Xu, G. Chen, P.-A. Heng and S. Z. Li, “A Survey on Generative Diffusion Models,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 36, no. 7, pp. 2814–2830, 2024. doi
29. C.-H. Lai, Y. Song, D. Kim, Y. Mitsufuji and S. Ermon, “The Principles of Diffusion Models,” monograph, arXiv:2510.21890, 2025. arXiv
30. H. Shen, J. Zhang, B. Xiong et al., “Efficient Diffusion Models: A Survey,” *Transactions on Machine Learning Research*, 2025. arXiv
31. A. Melnik, M. Ljubljanac, C. Lu, Q. Yan, W. Ren and H. Ritter, “Video Diffusion Models: A Survey,” *Transactions on Machine Learning Research*, 2024. arXiv
32. C. Lu and Y. Song, “Simplifying, Stabilizing and Scaling Continuous-Time Consistency Models,” *ICLR*, 2025. arXiv
33. A. Sauer, D. Lorenz, A. Blattmann and R. Rombach, “Adversarial Diffusion Distillation,” *ECCV*, 2024. arXiv
34. M. S. Albergo and E. Vanden-Eijnden, “Building Normalizing Flows with Stochastic Interpolants,” *ICLR*, 2023. arXiv
35. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” *CVPR*, 2009, pp. 248–255. doi
36. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *CVPR*, 2016, pp. 2818–2826. arXiv
37. O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical Image

Segmentation," *MICCAI*, LNCS 9351, pp. 234–241, 2015. arXiv
38. A. Radford, J. W. Kim, C. Hallacy et al., "Learning Transferable Visual Models From Natural Language Supervision," *ICML*, 2021. arXiv
39. M. Oquab, T. Darcet, T. Moutakanni et al., "DINOv2: Learning Robust Visual Features without Supervision," *Transactions on Machine Learning Research*, 2024. arXiv
40. B. Efron, "Tweedie's Formula and Selection Bias," *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1602–1614, 2011. doi
41. A. Dosovitskiy, L. Beyer, A. Kolesnikov et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *ICLR*, 2021. arXiv
42. T. Li and K. He, "Back to Basics: Let Denoising Generative Models Denoise," *CVPR*, 2026 (preprint 2025). paper, publication record, official JiT code.
43. B. Zheng, N. Ma, S. Tong and S. Xie, "Diffusion Transformers with Representation Autoencoders," *ICLR*, 2026 (preprint 2025). proceedings, manuscript.
44. S. Tong, B. Zheng, Z. Wang, B. Tang, N. Ma, E. Brown, J. Yang, R. Fergus, Y. LeCun and S. Xie, "Scaling Text-to-Image Diffusion Transformers with Representation Autoencoders," *CVPR Findings Track*, 2026. manuscript, author publication record.
45. D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna and R. Rombach, "SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis," *ICLR*, 2024. proceedings, official model card.
46. Black Forest Labs, "FLUX.1 [dev]," official pretrained-model card, 2024. model card.
47. W. Kong, Q. Tian, Z. Zhang et al., "HunyuanVideo: A Systematic Framework For Large Video Generative Models," technical report, arXiv:2412.03603, 2024. report, official release.
48. Team Wan, A. Wang, B. Ai, B. Wen et al., "Wan: Open and Advanced Large-Scale Video Generative Models," technical report, arXiv:2503.20314, 2025. report, official Wan2.1 release.
49. Z. Geng, Y. Lu, Z. Wu, E. Shechtman, J. Z. Kolter and K. He, "Improved Mean Flows: On the Challenges of Fastforward Generative Models," *CVPR*, pp. 30467–30476, 2026. CVF
50. Y. Lu, S. Lu, Q. Sun, H. Zhao, Z. Jiang, X. Wang, T. Li, Z. Geng and K. He, "One-step Latent-free Image Generation with Pixel Mean Flows," *ICML*, PMLR 306, pp. 81265–81277, 2026. PMLR

ML 09・視覺自迴歸：像素、Tokens 與尺度

把影像分布拆成一連串條件機率：PixelCNN、VQ-VAE 與 VQGAN tokenizer、GPT 式 token transformer、next-scale 預測、遮罩生成模型、連續 token，以及它們與擴散模型的比較。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/autoregressive-vision/

你將學到

- 以機率連鎖法則分解影像分布。
- 區分教師強制訓練與循序取樣。
- 比較像素預測與學得的視覺 token 化。
- 區分下一 token、下一尺度、遮罩與連續 token 方法。
- 評估解碼控制、快取及與擴散模型的比較。

先看全貌

自迴歸（autoregressive, AR）生成器一塊一塊地建立樣本。每一步，它看著目前為止產生的所有內容，預測下一塊的機率分布，從中抽樣，再把結果接到序列後面。這裡的「一塊」可以是一個像素值、學習出來的 codebook 中的一個整數 token、一個連續特徵向量，甚至是下一個解析度的整張圖。

> **用最簡單的話說**
>
> 想像一格一格地畫漫畫。每一格新的畫面都要和前面已經畫好的格子接得上。練習的時候，你臨摹真正的漫畫，所以永遠知道前面幾格長什麼樣子。自己創作新漫畫時，就只能接受自己先前畫的格子，即使有點歪也一樣。自迴歸影像模型就是這樣：用真實影像練習，再一塊一塊地創造新影像。

這為什麼重要？自迴歸正是大型語言模型背後的配方：用 transformer 做下一個 token 預測。把影像帶進這套配方，就能用同一種架構讀寫文字與圖片、沿用語言模型的部署工程，並用精確的似然（likelihood）衡量進展。代價是影像本來就不是序列，必須有人決定**什麼是 token**，以及**token 用什麼順序產生**。本章大部分內容都在討論這兩個決定。

Xiong 等人的綜述 [1] 依照預測單位，把視覺 AR 模型分成三大類：**以像素為基礎**、**以 token 為基礎**、**以尺度為基礎**。我們沿用這張地圖，再加上遮罩式（非由左至右）與連續 token 模型，它們模糊了 AR 與擴散之間的界線。前面關於生成模型與擴散與 flow matching的章節提供本章用到的 VAE 與擴散背景；Prince 的教科書 [9] 則以教材層級涵蓋一般的機率基礎。

鏈鎖法則不需要獨立假設

白話版：任何聯合分布都可以寫成一連串「給定前面元素，下一個元素的分布」的乘積。這是機率論的事實，而不是建模假設。

對於有順序的序列 $x = (x_1, \dots, x_N)$，

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i \mid x_{<i}), \qquad \mathcal{L}_{\text{NLL}}(\theta) = -\sum_{i=1}^{N} \log p_\theta(x_i \mid x_{<i}).$$

其中 x_i 是所選順序中的第 i 個元素，$x_{<i} = (x_1, \dots, x_{i-1})$ 是它之前的所有元素，θ 是網路參數，$\mathcal{L}_{\text{NLL}}$ 是單一訓練樣本的負對數似然（實務上會對整個資料集取平均）。

鏈鎖法則對**任何**順序都精確成立。近似只來自模型本身：有限的網路、有限的上下文視窗，或對每個條件分布形式的限制。離散值的條件分布通常是 softmax；二值像素可用 Bernoulli 分布；連續特徵則需要密度函數或另一個條件生成器（MAR 一節會看到一個例子）。

順序是計算上的選擇，不是對物理世界的主張。**光柵順序（raster order）**（由左到右、由上到下）很方便，但對二維影像來說是任意的：你下方的像素和右邊的像素一樣相關。不同順序會產生不同的條件分布，而有限的模型可能覺得某些順序比較容易學。圖 1 展示本章用到的三種預測單位。

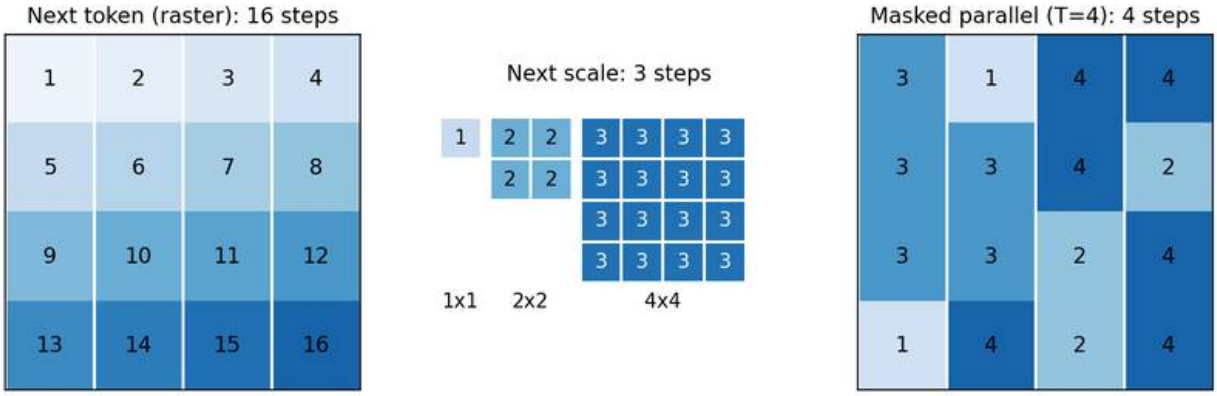


圖 *1* — 以三種生成方式產生同樣 *16* 個最終 *token*。數字代表產生該 *token* 的循序步驟：光柵自迴歸需要 *16* 步，*next-scale* 預測需要 *3* 步（每個尺度一步），$T = 4$ 的遮罩平行解碼需要 *4* 步。僅為示意圖，沒有訓練任何模型。

衡量 AR 模型：每維位元數

因為鏈鎖法則給出精確的 $\log p_\theta(x)$，AR 模型通常用**每維位元數（bits per dimension, bpd）**表示的負對數似然來比較：

$$\text{bpd}(x) = \frac{-\log_2 p_\theta(x)}{D},$$

其中 D 是影像中離散值的個數（例如 32×32 的 RGB 影像 $D = 3 \cdot 32 \cdot 32$），$\log_2$ 把 nat 換成位元。

如果模型對每個 8 位元值都給均勻機率，分數恰好是 8 bpd；任何更低的值都代表模型學到了結構。bpd 在保留（held-out）影像上量測，因此同時能偵測過擬合（overfitting）與欠擬合。

有兩點要注意。第一，bpd 和視覺品質可能不一致：模型可能把大部分似然花在肉眼看不到的雜訊上，卻在整體構圖上犯奇怪的錯。第二，對 **token** 模型而言，似然是定義在 codebook 索引上，而非像素上。有損的 tokenizer 會丟掉資訊，所以 token 層級的似然不是精確的像素空間似然，不能直接和像素模型的 bpd 比較。

Teacher forcing、因果遮罩與曝光偏差

白話版：教師強制在訓練時提供真實前綴，取樣時則使用生成前綴；輸入位移與遮罩須隱藏各預測的目標。

使用 **teacher forcing** 時，訓練損失中的每個條件分布都以真實的前文計算。由於整張訓練影像都在手上，只要禁止位置 i 讀取位置 $\geq i$，所有 N 個條件分布就能在一次前向傳遞中算完。因果 transformer 用注意力遮罩做到這一點；PixelCNN 則用遮罩過的卷積核。

常見的輸入慣例有兩種，遮罩必須與慣例相符：

- 在**位移輸入（shifted-input）**的解碼器中，輸入 `[BOS, x1, x2, ...]`，預測 `[x1, x2, x3, ...]`。這時包含對角線的下三角遮罩是正確的，因為位置 i 的輸入是前一個元素。
- 在接收**未位移**值的模型中，依賴關係必須**嚴格**在前。若包含對角線，網路就能直接複製它該預測的值。

推論時，先前抽樣的錯誤會進入後續步驟的上下文。這種訓練與推論的落差常稱為**曝光偏差（exposure bias）**。它不會讓最大似然變成錯的（分解依然精確），但對不完美的模型和長序列確實有實務影響：影像版面一開始的小錯，可能把後續步驟推進訓練中從未出現過的情境。

像素模型：PixelRNN 與 PixelCNN

白話版：把每個像素的強度當成 256 類的分類問題，輸入是它上方和左方的所有像素。

PixelRNN 與 PixelCNN 由 van den Oord、Kalchbrenner 與 Kavukcuoglu 在同一篇論文中提出 [2]。兩者都用 **256 類 softmax** 而不是高斯分布來建模每個 8 位元值，因此分布可以是多峰的（像素可以「要嘛黑、要嘛白，但不是灰」）。色彩則在每個像素內部採固定順序：先紅、再綠、再藍，讓綠色可以依賴紅色，藍色可以依賴兩者。PixelRNN 使用二維 LSTM 層（Row LSTM 與 Diagonal BiLSTM），依賴範圍可能沒有上限；PixelCNN 則以堆疊的遮罩卷積取代，感受野有限，但因為所有位置可以平行計算，訓練快得多。

遮罩卷積

遮罩卷積把卷積核乘上一個二值遮罩，將「未來」位置的權重設為零。在 [2] 中，**遮罩 A** 只用在第一層：它排除中心像素，所以第一層永遠看不到它要預測的值（彩色時，也只看得到已預測的通道）。**遮罩 B** 用在之後的所有層：它允許中心位置，因為第一層之後，中心的特徵已經只匯總了較早的像素。正式地說，對中心在 (c, c)、$c = \lfloor k/2 \rfloor$ 的 $k \times k$ 卷積核，

$$M_{uv} = \begin{cases} 1 & u < c, \\ 1 & u = c \text{ and } v < c \text{ (A) or } v \le c \text{ (B)}, \\ 0 & \text{otherwise}, \end{cases}$$

其中 u 是卷積核的列索引，v 是行索引。下面的程式建立兩種遮罩、堆疊五層，並用梯度找出 9×9 影像中，哪些輸入像素能影響中心位置的輸出。

```
import torch
from torch import nn
from torch.nn import functional as F

class MaskedConv2d(nn.Conv2d):
    """PixelCNN-style masked convolution for one channel (raster order)."""
    def __init__(self, cin, cout, k, mask_type):
        super().__init__(cin, cout, k, padding=k//2, bias=False)
        mask = torch.ones(k, k)
        c = k//2
        mask[c, c + (mask_type == "B"):] = 0   # centre (type A) and pixels to its right
        mask[c+1:] = 0                         # all rows below
        self.register_buffer("mask", mask)
    def forward(self, x):
        return F.conv2d(x, self.weight*self.mask, padding=self.padding)

torch.manual_seed(0)
layers = [MaskedConv2d(1, 8, 3, "A"), nn.Tanh()]
for _ in range(4):                             # 5 layers: reach of 5 pixels
    layers += [MaskedConv2d(8, 8, 3, "B"), nn.Tanh()]
net = nn.Sequential(*layers, MaskedConv2d(8, 1, 1, "B"))
print(net[0].mask.int())
x = torch.rand(1, 1, 9, 9, requires_grad=True)
net(x)[0, 0, 4, 4].backward()                  # logit for pixel (row 4, col 4)
dep = (x.grad[0, 0].abs() > 0).int()
print(dep)
later = torch.zeros(9, 9, dtype=torch.bool)
later[4, 4:] = True; later[5:] = True
assert dep[later].sum() == 0                   # never reads itself or later pixels
print("earlier pixels never seen (blind spot):", int(((dep == 0) & ~later).sum()))
```

assert 通過：目標位置本身及之後的像素都不會影響它的 logit，所以模型是合法的自迴歸分解。但輸出還揭露了另一件事：有六個在光柵順序中較早的像素，在目標右上方排成階梯狀，梯度為零。這就是堆疊遮罩卷積的**盲點（blind spot）**（圖 2）：每一層的感受野往左上方擴張，右上方的區域卻永遠到不了。模型依然正確，只是得到的資訊比它能得到的少。

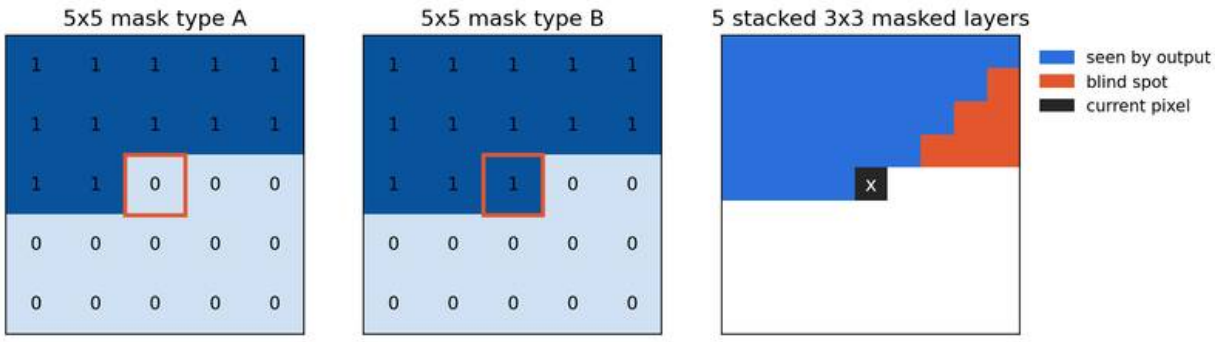


圖 2 — 左、中：遮罩 *A* 排除中心權重，遮罩 *B* 保留中心權重。右：疊了五層 3×3 遮罩卷積後，用梯度量出哪些輸入像素會影響深色格子的預測。橘色格子在光柵順序中較早，卻永遠看不到，也就是盲點。

van den Oord 等人的 Gated PixelCNN [3] 把網路拆成兩個堆疊來消除盲點：**垂直堆疊**不受限制地看到目前這列以上的所有列，**水平堆疊**看到目前這列中直到目前像素為止的部分，兩者在每層之後合併。它也把 ReLU 換成閘控單元 $y = \tanh(W_f * x) \odot \sigma(W_g * x)$，其中 $*$ 是卷積，σ 是 logistic sigmoid，$\odot$ 是逐元素乘法；並示範如何以類別標籤或嵌入向量為條件。加上這些改變後，PixelCNN 在 ImageNet [23] 上達到與 PixelRNN 相當的似然，計算成本卻低得多 [3]。

程式：最小的自迴歸模型

遮罩卷積只是強制因果性的一種方式，也不一定需要 transformer。以下完整的 CPU 實驗產生 4×4 二值影像，每一列都重複同樣四個隨機位元。一個帶遮罩的線性層，用嚴格下三角遮罩排除目前像素與之後所有像素，直接學習條件分布。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
class BinaryAR(nn.Module):
    def __init__(self, n=16):
        super().__init__()
        self.weight = nn.Parameter(torch.zeros(n, n))
        self.bias = nn.Parameter(torch.zeros(n))
        self.register_buffer("mask", torch.tril(torch.ones(n, n), diagonal=-1))
    def forward(self, x):
        return F.linear(x, self.weight*self.mask, self.bias)

model = BinaryAR()
opt = torch.optim.Adam(model.parameters(), lr=.04)
for step in range(600):
    first_row = torch.randint(2, (128, 4)).float()
    x = first_row.repeat(1, 4)
    loss = F.binary_cross_entropy_with_logits(model(x), x)
    opt.zero_grad(); loss.backward(); opt.step()
```

```
with torch.no_grad():
    a = torch.rand(2, 16)
    b = a.clone(); b[:, 8:] = torch.rand(2, 8)
    assert torch.allclose(model(a)[:, :9], model(b)[:, :9])
    samples = torch.zeros(8, 16)
    for i in range(16):
        p = model(samples)[:, i].sigmoid()
        samples[:, i] = torch.bernoulli(p)
    rows = samples.reshape(-1, 4, 4)
    print("one generated image:\n", rows[0].int())
    print("repeat agreement:", (rows[:, 1:] == rows[:, :1]).float().mean().item())
```

assert 檢查因果性：改變第 8 個位置以後的輸入，不會改變位置 0 到 8 的 logit，因為這些輸出只讀較早的輸入。訓練之後，後面各列與第一列的一致率達到 1.0，而第一列仍保持隨機（模型不應該背下某一列）。

真實分布有 16 張機率相同的影像，所以熵是每張 $\log_2 16 = 4$ 位元，也就是每像素 0.25 位元。後面幾列不含任何新的不確定性。若模型把 16 個像素都當成獨立的公平硬幣，每張影像需要 16 位元。16 與 4 位元之間的差距，正是自迴歸模型能捕捉、而獨立像素模型捕捉不到的結構。

為什麼像素模型擴展不下去

一張 256×256 的 RGB 影像有 $3 \cdot 256 \cdot 256 = 196{,}608$ 個值，所以像素層級的取樣需要大約二十萬次循序的網路計算。大部分似然也被花在細微、局部容易預測的細節上，而人們真正注意的整體構圖只占位元的一小部分。Image GPT [11] 證明，在低解析度影像上以自迴歸方式預測像素的純序列 transformer，學到的特徵在線性探測（linear probing）下遷移效果很好；但在較高解析度下，像素路線依然太昂貴。下一步就是學習更短、更有意義的序列。

離散視覺 token：VQ-VAE 與 VQGAN

白話版：另外訓練一個網路，把影像壓縮成一小格來自學習字典的「單字」，然後對單字做自迴歸，而不是對像素。

VQ-VAE

VQ-VAE [4] 有一個編碼器，在降取樣網格的每個位置輸出特徵向量 $z_e(x) \in \mathbb{R}^d$；一個由 K 個學習向量組成的 **codebook** $\{e_1, \dots, e_K\} \subset \mathbb{R}^d$；以及一個解碼器。每個特徵都被替換成最近的 code：

$$k^* = \arg\min_k \|z_e(x) - e_k\|_2^2, \qquad z_q(x) = e_{k^*}.$$

影像於是由 k^* 值組成的整數網格表示。訓練目標是

$$\mathcal{L}_{\text{VQ}} = \underbrace{\|x - D(z_q)\|^2}_{\text{reconstruction}} + \underbrace{\|\text{sg}[z_e] - e_{k^*}\|^2}_{\text{codebook}} + \beta \underbrace{\|z_e - \text{sg}[e_{k^*}]\|^2}_{\text{commitment}},$$

其中 D 是解碼器，sg[·] 是**停止梯度（stop-gradient）**運算子（前向傳遞時是恆等函數，反向傳遞時梯度為零），β 是 commitment 項的小權重（下面的程式用 0.25）。這裡把重建項寫成平方誤差；也可以換成任何重建似然。

三項各司其職。**codebook** 項把選中的 code 拉向編碼器輸出，就像 k-means 的更新。**commitment** 項把編碼器拉向它的 code，讓編碼器輸出不會離 codebook 太遠。`argmin` 沒有有用的梯度，所以重建損失的梯度要透過**直通估計器（straight-through estimator）**傳到編碼器：前向傳遞使用 z_q，反向傳遞則把 z_q 的梯度原封不動複製給 z_e，彷彿量化是恆等函數。在程式中只要一行：`z_q = z_e + (e - z_e).detach()`。

下面的玩具例子在四個群集上訓練一個二維「編碼器」、codebook 與解碼器，共有八個 code 可用：

```
import torch
from torch import nn
from torch.nn import functional as F

def quantize(z_e, codebook, beta=0.25):
    """z_e: (n, d) encoder outputs; codebook: (K, d). Returns z_q with straight-through
     ↪  gradient."""
    dist = torch.cdist(z_e, codebook)            # (n, K) Euclidean distances
    idx = dist.argmin(1)                         # nearest code index per vector
    e = codebook[idx]
    codebook_loss = F.mse_loss(e, z_e.detach())  # moves codes toward encoder outputs
    commit_loss = F.mse_loss(z_e, e.detach())    # keeps the encoder near its code
    z_q = z_e + (e - z_e).detach()               # forward: e; backward: identity
    return z_q, idx, codebook_loss + beta*commit_loss

torch.manual_seed(0)
torch.set_num_threads(1)
centres = torch.tensor([[2., 0.], [-2., 0.], [0., 2.], [0., -2.]])
encoder = nn.Linear(2, 2)                        # toy "encoder" and "decoder"
decoder = nn.Linear(2, 2)
codebook = nn.Parameter(torch.randn(8, 2)*.1)    # K = 8 codes, 4 are enough
opt = torch.optim.Adam([*encoder.parameters(), *decoder.parameters(), codebook], lr=.03)
for step in range(800):
    x = centres[torch.randint(4, (256,))] + .1*torch.randn(256, 2)
    z_q, idx, vq_loss = quantize(encoder(x), codebook)
    loss = F.mse_loss(decoder(z_q), x) + vq_loss
    opt.zero_grad(); loss.backward(); opt.step()

with torch.no_grad():
    x = centres.repeat_interleave(100, 0) + .1*torch.randn(400, 2)
    z_q, idx, _ = quantize(encoder(x), codebook)
    print("reconstruction MSE:", round(F.mse_loss(decoder(z_q), x).item(), 4))
    print("codes used:", idx.unique().numel(), "of", len(codebook))
    print("codes per cluster:", [idx[i*100:(i+1)*100].unique().tolist() for i in range(4)])
z = torch.randn(3, 2, requires_grad=True)
z_q, _, _ = quantize(z, codebook.detach())
z_q.sum().backward()
print("straight-through gradient:", z.grad[0].tolist())
```

輸出顯示重建 MSE 約 0.0096，只用了 **8 個 code 中的 4 個**（每個群集一個），直通梯度是 `[1.0`

, 1.0]。這些數字藏著兩個教訓。重建誤差幾乎正好等於群集內的雜訊變異數 $0.1^2 = 0.01$：量化保留了「屬於哪一群」，丟掉了「在群內哪個位置」。另外，一半的 codebook 是**死碼（dead code）**：從未被選到的 code 收不到 codebook 梯度，永遠不會移動（圖 3）。在真正的 tokenizer 中，codebook 使用率低會浪費容量，是需要監控的典型失敗。

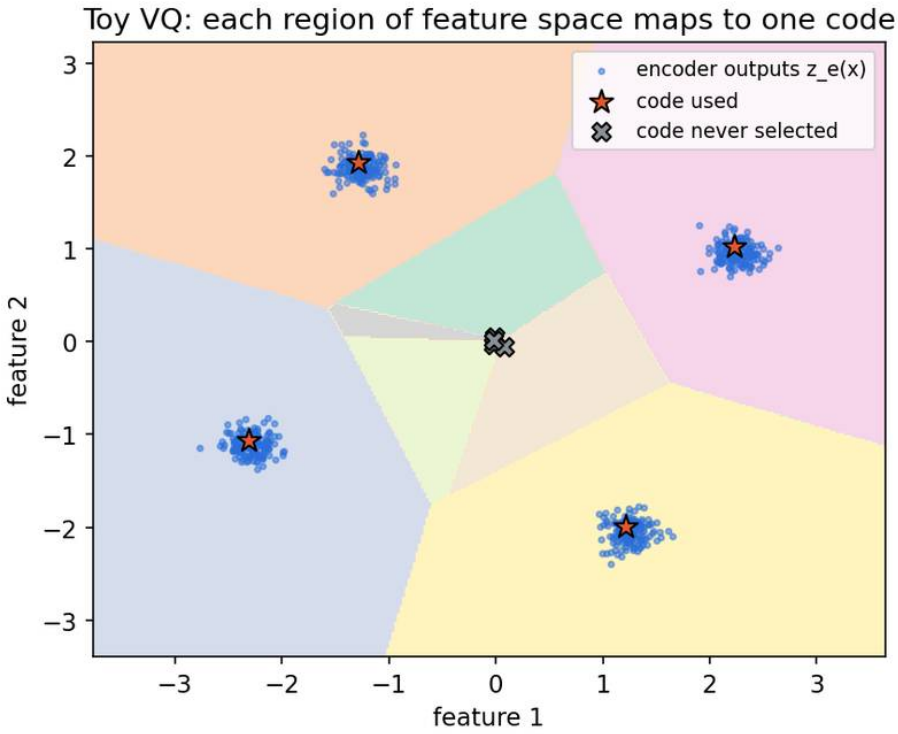


圖 3 — 玩具向量量化。每個色塊是會被映射到同一個 *code* 的特徵集合。四個 *code*（星形）被使用，每群一個；另外四個（叉號）從未被選中，停留在初始化位置附近。

兩個階段：先 tokenizer，再先驗

VQ-VAE 本身只是自編碼器，並不會告訴你哪些 token 網格比較可能出現。因此生成分成**兩個階段**（圖 4）。第一階段訓練 tokenizer。第二階段把它凍結，把每張訓練影像轉成索引網格，再對 M 個索引訓練自迴歸模型 $p_\psi(k_1, \dots, k_M)$。取樣時，先從先驗抽出索引網格，再解碼。[4] 中的先驗是建立在 code 上的 PixelCNN；VQ-VAE-2 [10] 使用兩層 code 圖（較粗的上層與較細的下層）搭配 PixelCNN 式先驗，產生了多樣且高保真度的 ImageNet 樣本。

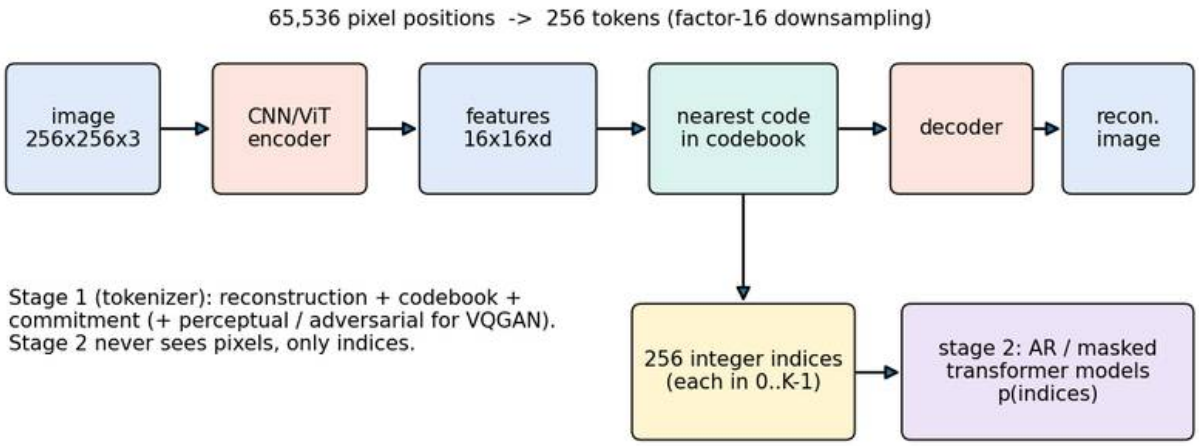


圖 *4* — 兩階段生成。*tokenizer*（上排）為重建而訓練；先驗（右下）只看得到整數索引。以 *16* 倍降取樣，一張 *256×256* 影像變成 *256* 個 *token*，而不是 *65,536* 個像素位置。

簡單的算術說明了它的吸引力。若 tokenizer 把 256 × 256 影像在每個方向降取樣 16 倍，先驗看到的是 16 × 16 = 256 個 token，而不是 65,536 個像素位置（或 196,608 個色彩值）。自注意力的成本隨序列長度平方成長，這就是可行與不可行之間的差別。

VQGAN：感知導向的 token

Esser、Rombach 與 Ommer 的 VQGAN [5] 保留兩階段設計，但訓練 tokenizer 時除了重建損失，還加上**感知損失（perceptual loss）**與**以區塊為單位的對抗損失（patch-based adversarial loss）**，並用 transformer 對 token 網格建模。對抗與感知項讓解碼器能從很少的 token 產生清晰、逼真的紋理，因為 token 不再需要精確編碼每個像素，只需要編碼足夠的資訊，讓解碼器產生一個合理的版本。這就是 VQGAN 式 tokenizer 成為 token 式生成標準前端的原因。注意分工：**tokenizer** 使用對抗訓練，**先驗**則是用最大似然訓練的普通自迴歸模型。

Codebook 的問題與替代方案

真實的 tokenizer 反覆遇到三個問題：死碼（如玩具例子）、tokenizer 能重建的內容與先驗能建模的內容之間的落差，以及文字、細線等小細節的遺失。幾種設計分別回應這些問題：

- **ViT-VQGAN** [12] 使用 Vision Transformer 的編碼器與解碼器，在低維的**分解（factorized）**空間中查找 code，並對 code 與特徵做 ℓ_2 **正規化**，改善了 codebook 使用率與重建品質。
- **RQ-VAE** [15] 使用**殘差量化（residual quantization）**：每個位置由一小疊 code 表示，每個 code 量化前面幾個 code 沒表示到的部分。這讓空間網格可以更粗（序列更短），保真度卻相近。
- **FSQ** [16]（有限純量量化，finite scalar quantization）把特徵投影到少數幾維，每一維四捨五入到少數固定的值。隱含的 codebook 大小是各維層級數的乘積；沒有會崩潰的學習 codebook，也不需要 commitment 損失。

壓縮一定會造成瓶頸。請**分別**量測 codebook 使用率、重建品質與生成器品質：生成器無法產生 tokenizer 表示不了的細節。

僅線上版：*VQ-VAE*：離散編碼與另行學習的先驗 — *https://blog.shyandram.dev/zh-tw/learn/ml/autoregressive-vision/papers/vq-vae 離散編碼與另行學習的先驗/*

建立在 token 上的自迴歸 transformer

白話版：僅解碼器 Transformer 可像預測文字 token 一樣預測下一個視覺 token；視覺 tokenizer 與解碼器決定 token 表示什麼。

從像素到 token

Image GPT [11] 把 GPT 式 transformer 直接用在低解析度像素序列上。它的主要發現是關於**表徵學習**：只訓練來預測下一個像素的模型，其特徵在上面訓練線性分類器時遷移得很好。改用 token 而不是像素，序列短到足以在更高解析度下生成。VQGAN transformer [5] 是經典例子：一個在 VQGAN 索引上以光柵順序運作的類 GPT 模型，讓 transformer 能合成若以像素序列表示會長得無法處理的高解析度影像。

把文字生成影像當成翻譯：Parti

Parti [13] 把文字生成影像視為機器翻譯。transformer **編碼器**讀取文字提示；transformer **解碼器**以它為條件，自迴歸地生成 ViT-VQGAN 影像 token [12]。報告把這個序列到序列（sequence-to-sequence）模型擴展到 200 億參數，並提出 PartiPrompts：一個涵蓋多種內容類別、超過 1,600 個英文提示的基準。

原封不動的語言模型架構：LlamaGen

LlamaGen [14] 問的是：沒有任何影像專屬歸納偏置的標準語言模型架構，擴大規模後能不能把影像生成好？它在 16 倍降取樣、$K = 16{,}384$ 個 8 維 code 的 VQ tokenizer 上，使用 Llama 式 [24] 解碼器（RMSNorm 前置正規化、SwiGLU 前饋層、二維旋轉位置編碼）。類別標籤或投影後的文字嵌入作為**前綴（prefix）**token 輸入，之後以光柵順序生成。論文釋出 1.11 億到 31 億參數的類別條件模型，並指出標準的 LLM 推論服務軟體（vLLM）能加速推論。它更廣的訊息是架構重用：除了 tokenizer，影像生成器幾乎可以和文字模型共用一切。

自迴歸模型的 classifier-free guidance

Classifier-free guidance（CFG）[18] 原本是為擴散模型提出的，但可以直接搬到 AR 的 logit 上。訓練時，條件 c 會隨機被替換成學習出來的**空（null）**嵌入，讓同一個網路同時學會條件與無條件預測。取樣的每一步計算兩組 logit 再組合：

$$\ell_{\mathrm{g}} = \ell_{\mathrm{u}} + s\,(\ell_{\mathrm{c}} - \ell_{\mathrm{u}}),$$

其中 ℓ_c 是有條件時的下一個 token logit，ℓ_u 是空條件時的 logit，s 是 guidance 強度；$s = 1$ 就回到普通的條件模型。LlamaGen 使用的正是這個形式 [14]。較大的 s 通常提升對提示或類別的貼合度與樣本銳利度，代價是多樣性。除非把兩次計算合併成一個批次，否則它也會讓每步計算量加倍，所以只看步數會低估成本。

Next-scale 預測（VAR）

白話版：不再依固定閱讀順序預測下一個 token，而是在所有較粗版本的條件下，預測整張影像下一個更清晰的版本。

光柵順序會造成奇怪的依賴：影像左上角在完全不知道右下角的情況下被生成。VAR（Visual AutoRegressive modeling）[6] 主張這種單向順序違背影像 token 之間的雙向相關與空間局部性，因此改用解析度遞增的 token 圖 $r_1, \dots, r_K$ 上的**由粗到細**順序：

$$p(r_1, \dots, r_K) = \prod_{k=1}^{K} p(r_k \mid r_1, \dots, r_{k-1}),$$

其中 r_k 是 $h_k \times w_k$ 的 token 索引網格，r_1 是 1×1，r_K 是完整的潛在解析度。在較粗的圖給定之下，r_k 的所有 token **平行**預測，所以循序步數等於尺度數，而不是 token 數。

tokenizer 是方法的一部分。VAR 的多尺度 VQ 編碼器以**殘差**方式運作：從編碼器特徵圖 f 開始，對 $k = 1, \dots, K$，把 f 降取樣到 $h_k \times w_k$，量化得到 r_k（所有尺度共用一個 codebook），把查到的 code 升取樣回完整解析度，經過一個小卷積 ϕ_k，再從 f 減掉，然後進入下一個尺度 [6]。因此每個尺度編碼的是較粗尺度還沒解釋的部分。單純把普通 token 圖縮放並不能重現這個方法。

訓練時，**區塊式因果（block-wise causal）**注意力遮罩讓 r_k 的每個 token 可以注意 $r_{\le k}$ 的所有 token，但看不到更細的尺度；推論時以 KV cache 儲存較粗的尺度。VAR 也使用 CFG。論文報告在類別條件的 ImageNet 256×256 上，VAR 把其 AR 基準的 FID 從 18.65 改善到 1.73，推論也更快，而且模型呈現類似語言模型的冪律擴展趨勢 [6]。VAR 獲得 NeurIPS 2024 最佳論文獎。和往常一樣，這些數字只適用於論文的資料、解析度與訓練設定。

尺度內的平行預測本身就是一個假設：r_k 中的 token 在給定 $r_{<k}$ 時被獨立抽樣。遮罩模型一節會在玩具問題上展示這個假設的代價。

遮罩生成模型（MaskGIT 與 Muse）

白話版：從一格全是空白的網格開始。每一輪同時猜所有空格，只保留最有把握的猜測，其餘的再塗回空白。

MaskGIT [8] 使用與 AR 模型相同的 VQ token，但改用能看到所有位置的**雙向** transformer。訓練目標是遮罩 token 預測：選一個遮罩比例，把這個比例的 token 換成特殊的 `[MASK]` token，然後在被遮住的位置上最小化交叉熵，

$$\mathcal{L}_{\text{mask}} = -\mathbb{E}\Big[\sum_{i \in M} \log p_\theta(y_i \mid y_{\bar{M}}) \Big],$$

其中 y 是 token 網格，$\mathcal{M}$ 是被遮住的位置集合，$y_{\bar{\mathcal{M}}}$ 是沒被遮住的 token。遮罩比例取為 $\gamma(r)$，其中 $r \sim \mathcal{U}(0,1)$，**遮罩排程（mask schedule）** γ 從 1 遞減到 0。

生成進行 T 次迭代。第 t 次迭代時，模型平行預測並抽樣所有被遮住的 token；每個樣本的機率就是它的**信心（confidence）**。這次迭代後仍保持遮住的 token 數為

$$n_t = \lceil \gamma(t/T)\, N \rceil,$$

N 是 token 總數；信心最低的 n_t 個新 token 會被重新遮住，其餘的則固定下來。MaskGIT 比較了數種排程，選擇**餘弦**排程 $\gamma(r) = \cos(\pi r/2)$：早期（模型所知甚少時）只固定少數 token，後期才固定較多；它以 16×16 的 token 網格和 1,024 個條目的 codebook，在 8 次迭代內生成 256×256 影像 [8]。Muse [17] 把同樣的想法擴展到文字生成影像，以預訓練大型語言模型的文字嵌入作為遮罩 token transformer 的條件。

MaskGIT **不是**一般由左到右的鏈鎖法則自迴歸：順序是依信心即時決定的，而且同一步會抽樣好幾個 token。這些同時產生的 token 在目前上下文給定下是**獨立**抽樣的。下面的玩具例子顯示這何時會造成影響。它在前面那個重複列的 4×4 影像上訓練一個小型雙向網路，並用餘弦排程解碼。

```
import math
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
N, MASK = 16, 2                                  # 16 binary tokens; state 2 = [MASK]
gamma = lambda r: math.cos(math.pi*r/2)          # cosine schedule: fraction still masked
net = nn.Sequential(nn.Flatten(), nn.Linear(3*N, 128), nn.ReLU(), nn.Linear(128, N))

def data(n):                                     # 4x4 images whose rows repeat
    return torch.randint(2, (n, 4)).repeat(1, 4)

def logits(tokens):                              # sees every position: bidirectional
    return net(F.one_hot(tokens, 3).float())

opt = torch.optim.Adam(net.parameters(), lr=3e-3)
for step in range(1500):
    x = data(256)
    ratio = torch.tensor([gamma(r) for r in torch.rand(256).tolist()])
    n_mask = (ratio*N).ceil().clamp(min=1)
    masked = torch.rand(256, N).argsort(1) < n_mask[:, None]   # random positions
    inp = torch.where(masked, MASK, x)
    loss = F.binary_cross_entropy_with_logits(logits(inp)[masked], x[masked].float())
    opt.zero_grad(); loss.backward(); opt.step()

@torch.no_grad()
def generate(n, T=4):
    tokens = torch.full((n, N), MASK)
    for t in range(1, T+1):
        p = logits(tokens).sigmoid()
        sample = torch.bernoulli(p).long()
        conf = torch.where(sample == 1, p, 1-p)              # prob. of the drawn value
```

```
        still = tokens == MASK
        conf = torch.where(still, conf, torch.inf)            # never re-mask fixed tokens
        proposal = torch.where(still, sample, tokens)
        n_keep_masked = math.floor(gamma(t/T)*N)              # 0 after the last step
        order = conf.argsort(1)                               # least confident first
        remask = torch.zeros_like(still)
        remask.scatter_(1, order[:, :n_keep_masked], True)
        tokens = torch.where(remask, MASK, proposal)
    return tokens

for T in (1, 2, 4, 8, 16):
    rows = generate(1000, T).reshape(-1, 4, 4)
    agree = (rows[:, 1:] == rows[:, :1]).float().mean().item()
    valid = (rows == rows[:, :1]).all(2).all(1).float().mean().item()
    print(f"T={T:2d}  repeat agreement={agree:.3f}  valid images={valid:.3f}")
```

（程式用 `floor` 而不是論文中的上取整，好讓這個小例子在 $t = T$ 時剩餘數量剛好是零。）輸出為：

```
T= 1  repeat agreement=0.511  valid images=0.000
T= 2  repeat agreement=0.812  valid images=0.100
T= 4  repeat agreement=0.908  valid images=0.360
T= 8  repeat agreement=0.998  valid images=0.982
T=16  repeat agreement=1.000  valid images=0.999
```

$T = 1$ 時，每個 token 都從全遮罩的上下文獨立抽樣，所以各列幾乎從不一致。迭代次數增加後，前幾個 token 早早被固定，後面的 token 可以以它們為條件；到 $T = 8$ 時幾乎每個樣本都合法。在自然影像上，同一輪解碼的 token 之間依賴較弱，所以少數幾次迭代就夠用；但**較少步數**與**較多獨立假設**之間的取捨是真實存在的（圖 5）。

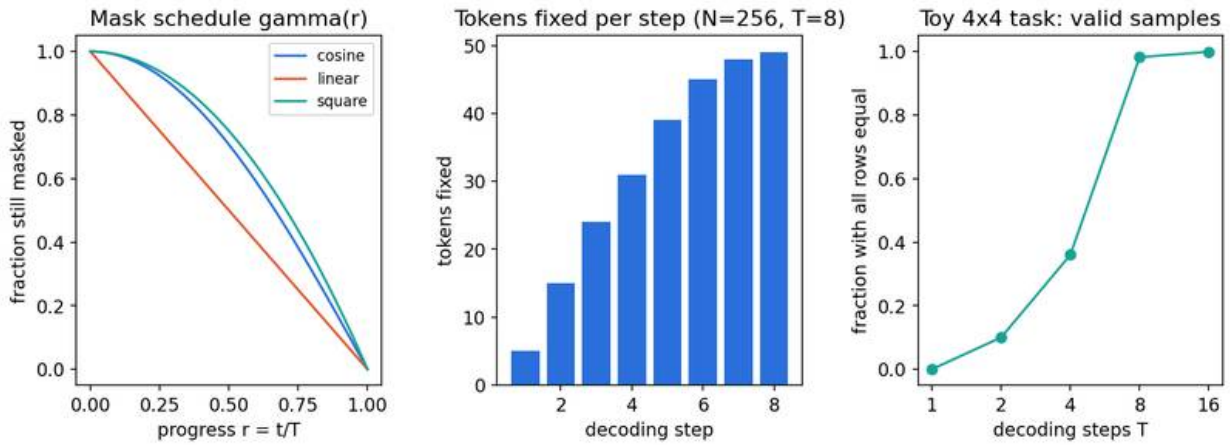


圖 5 — 左：候選的遮罩排程 $\gamma(r)$。中：在餘弦排程、256 個 *token*、$T = 8$ 時，早期只固定少數 *token*，後期固定較多。右：在重複列的玩具任務中，合法樣本比例隨解碼迭代次數增加，因為同一步中被獨立抽樣的 *token* 變少了。

連續 token 與擴散條件分布（MAR）

白話版：自迴歸定義條件順序，小型擴散模型則可表示各連續 token 的條件分布。

向量量化有損，而且不好訓練。Li 等人 [7] 指出，自迴歸需要的是一種建模 $p(x_i \mid x_{<i})$ 的方法，不一定是類別分布。他們讓 AR（或遮罩 AR）transformer 為每個連續值 token x_i 輸出條件向量 z_i，並以**擴散損失（diffusion loss）**訓練一個小型 MLP 去噪器 ϵ_θ：

$$\mathcal{L}(z_i, x_i) = \mathbb{E}_{\epsilon,t}\|\epsilon - \epsilon_\theta(x_{i,t} \mid t, z_i)\|^2, \qquad x_{i,t} = \sqrt{\bar{\alpha}_t}\, x_i + \sqrt{1-\bar{\alpha}_t}\, \epsilon,$$

其中 $\epsilon \sim \mathcal{N}(0, I)$，$t$ 是擴散時間步，$\bar{\alpha}_t$ 是 ML 08 中的雜訊排程。梯度也會經由 z_i 流進 transformer。要抽樣一個 token，就以 z_i 為條件執行小型去噪器的反向擴散過程；temperature 用來縮放注入的雜訊。論文的主要變體 **MAR**（masked autoregressive）採隨機順序、每步預測一組 token，把 MaskGIT 的外層結構和內層的連續取樣器結合在一起 [7]。

因此，「AR 對擴散」並不是二選一。一個系統可以用自迴歸決定 token 的順序與分組，再用擴散抽樣每一組。同樣地，語言解碼器可以自迴歸地生成描述影像的文字，而完全不生成像素；統一這些方法的是分解的概念，而不是某種特定的輸出類型。

取樣、temperature 與快取

白話版：訓練好的模型在每一步給出一個分布；取樣器決定抽樣時要多大膽，快取則避免重算過去。

對 K 個可能值的 logit ℓ_k，**temperature** T 會重新縮放分布：

$$p_T(k) = \frac{\exp(\ell_k / T)}{\sum_{j=1}^{K} \exp(\ell_j / T)}.$$

較低的 T 讓機率集中在最高的幾個值；較高的 T 則讓機率分散（圖 6）。**Top-k** 取樣只保留最高的 k 個 logit；**top-p**（nucleus）取樣保留機率總和達到 p 的最小集合。兩者都會把剩下的部分重新正規化。

```
import torch

def filtered_probs(logits, temperature=1.0, top_k=None, top_p=None):
    """Return the (renormalized) distribution actually sampled from."""
    logits = logits / temperature
    if top_k is not None:                                # keep the k largest logits
        kth = logits.topk(top_k).values[..., -1, None]
        logits = logits.masked_fill(logits < kth, -torch.inf)
    if top_p is not None:                                # smallest set with mass >= p
        sorted_logits, order = logits.sort(descending=True)
        cum = sorted_logits.softmax(-1).cumsum(-1)
        drop = cum - sorted_logits.softmax(-1) >= top_p  # mass before this token already >=
         ↪ p
        sorted_logits = sorted_logits.masked_fill(drop, -torch.inf)
        logits = torch.empty_like(logits).scatter(-1, order, sorted_logits)
    return logits.softmax(-1)
```

```
logits = torch.tensor([2.0, 1.0, 0.0, -1.0])
for name, kw in [("T=1", {}), ("T=0.5", {"temperature": .5}), ("T=2", {"temperature": 2.}),
                 ("top-k=2", {"top_k": 2}), ("top-p=0.8", {"top_p": .8})]:
    p = filtered_probs(logits, **kw)
    entropy = -(p[p > 0]*p[p > 0].log2()).sum().item()
    print(f"{name:10s}", [round(v, 3) for v in p.tolist()], f"entropy={entropy:.2f} bits")
```

模型本身的分布（$T = 1$）熵為 1.37 位元；$T = 0.5$ 把它降到 0.66 位元，$T = 2$ 提高到 1.80 位元。在這個例子中，top-$k = 2$ 與 top-$p = 0.8$ 都保留最可能的兩個值（第一個值的機率 0.644 小於 0.8，所以還需要第二個）。這些設定全都改變了實際抽樣的分布，因此不要把取樣器的視覺品質和模型的似然混為一談。

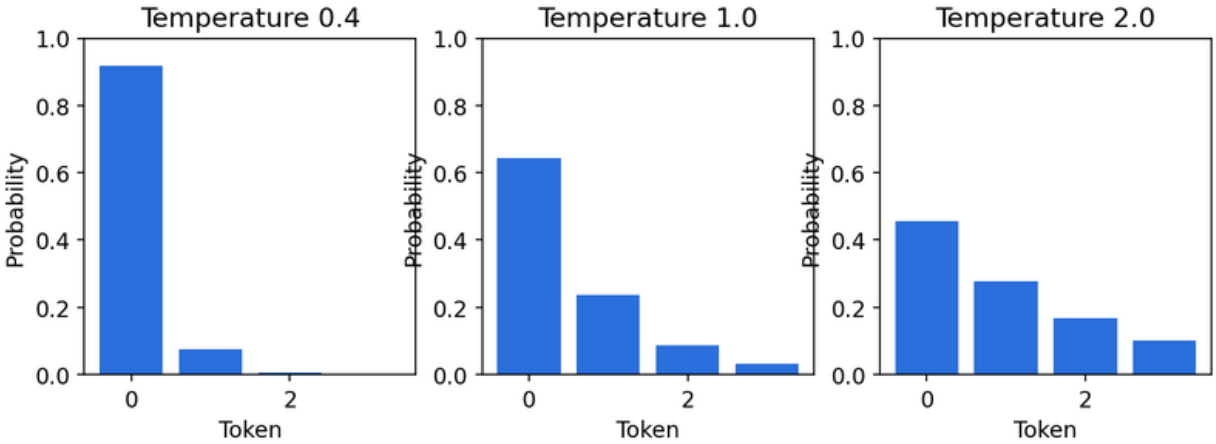


圖 6 — *temperature* 改變每個條件分布的多樣性，但無法找回 *tokenizer* 已經丟掉的資訊。

KV cache

在因果 transformer 中，新增一個 token 時，較早 token 的 key 與 value 不會改變。**KV cache** 把它們存起來，因此每一步只需計算最新 token 的 query、key 與 value，再對快取做注意力：

```
import torch
from torch.nn import functional as F

torch.manual_seed(0)
d, n = 8, 6
Wq, Wk, Wv = (torch.randn(d, d) for _ in range(3))
x = torch.randn(n, d)                                  # embeddings of n tokens

# Full causal attention over the whole sequence (what training computes).
q, k, v = x@Wq, x@Wk, x@Wv
mask = torch.ones(n, n).tril().bool()
scores = (q@k.T/d**.5).masked_fill(~mask, -torch.inf)
full = scores.softmax(-1)@v
```

```
# Incremental decoding: each step computes q, k, v for the newest token only.
K_cache, V_cache, outputs = [], [], []
for i in range(n):
    K_cache.append(x[i]@Wk); V_cache.append(x[i]@Wv)
    K, V = torch.stack(K_cache), torch.stack(V_cache)
    att = F.softmax((x[i]@Wq)@K.T/d**.5, -1)        # attends to tokens 0..i
    outputs.append(att@V)
assert torch.allclose(full, torch.stack(outputs), atol=1e-5)
print("cached decoding matches full causal attention")
print("dot products per step:", [i+1 for i in range(n)], "total:", n*(n+1)//2)
```

assert 確認快取解碼的結果與訓練時的因果輸出完全相同（這裡只有一層；真實模型會快取每一層、每個 head）。快取省去重算，卻沒有消除循序依賴，而且會隨序列長度線性成長。第 i 步仍要對 i 個快取的 key 做注意力，所以長度 n 的序列總注意力工作量是 $n(n+1)/2$，對 n 呈平方成長。這就是更短的序列（更好的 tokenizer）與更少的循序步驟（next-scale 或遮罩解碼）如此重要的原因。

對影片而言，token 可以依空間與時間排序，或分組成區塊。這時快取、壓縮與時間順序主導了成本，而物件一致性與長時間漂移需要另外評估。

自迴歸與擴散的比較

白話版：兩個家族都把一個困難的生成問題拆成許多比較簡單的問題。AR 把影像切成小塊；擴散則把雜訊程度切成多個步驟。

面向	自迴歸（token／尺度）	遮罩生成	擴散／ flow（ML 08）
每步生成什麼	一個 token（或一個尺度）	依信心選出的一組 token	對整個樣本的一次修正
循序步數	token 數或尺度數	固定且通常很少（T）	求解器步數（NFE）
常見表示	離散 code（或搭配擴散頭的連續 token [7]）	離散 code	連續的像素或潛在表示
訓練目標	精確 NLL（交叉熵）	遮罩交叉熵	去噪／速度回歸（與 ELBO 相關）
似然	在 token 上精確	不是單一精確的鏈鎖法則似然	下界或基於 ODE 的估計
天然優勢	與 LLM 共用架構與部署；容易與文字混合	步數少；修補與編輯容易	影像品質強；guidance 彈性大
常見弱點	循序解碼長；tokenizer 瓶頸；錯誤累積	同一步內的獨立假設	除非蒸餾，否則網路計算次數多

有三點讓這張表保持公允。第一，家族之間會混合：MAR [7] 在 AR 迴圈中放了一個擴散模型，而潛在擴散也依賴一個類似 tokenizer 的學習自編碼器。第二，某個家族「勝過」另一個的說法（例如 LlamaGen 的論文標題 [14]，或 VAR 與 DiT 的比較 [6]），只在特定基準、tokenizer、模型大小與 guidance 設定下成立，並非普遍成立。第三，成本應該以網路計算次數、guidance 計算次數、序列長度與實際延遲比較，不能只看「步數」。

現代觀點

這個領域已經從原始像素走向學習出來的表示，從固定的光柵順序走向彈性的預測單位，從獨立的影像生成器走向能同時讀寫文字與影像的模型。以下幾篇綜述描繪了這片版圖：

- **Xiong 等人 [1]**（TMLR）是最直接的視覺自迴歸綜述。它依表示方式分類（以像素、token、尺度為基礎的模型），也就是本章採用的分類，並涵蓋影像、影片、3D、多模態生成，以及具身智慧與醫學等應用。
- **Bond-Taylor 等人 [19]**（TPAMI 2022）並列比較 VAE、GAN、正規化流（normalizing flow）、能量模型與自迴歸模型，包括執行時間、樣本多樣性與架構限制上的取捨。它把 AR 模型放進更大的生成模型版圖中：精確的似然與穩定的訓練，代價是緩慢的循序取樣。
- **Li 等人 [20]**（TPAMI 2026）綜述多模態 LLM 的離散 tokenization，整理向量量化的各種變體，以及它們在傳統系統、單一模態 LLM 與多模態 LLM 中的應用，並討論 codebook 崩潰與梯度估計不穩定等問題。
- **Jia 等人 [22]** 綜述離散 tokenizer 在生成、理解、推薦與檢索中的應用，先拆解 tokenizer 的子模組與運作機制，再依應用整理方法，並列出現有限制與未來方向。
- **Zhao 等人 [21]** 綜述統一的多模態理解與生成模型，把它們分成以擴散為基礎、以自迴歸為基礎，以及 AR 與擴散混合三類，這正是 2025–2026 年大部分研究活動所在。

最新進展（2023–2026）。在經過查證的文獻中，有三條主線特別突出。(i) 在更好的 *tokenizer* 上擴展單純的 *AR transformer*：LlamaGen [14] 顯示，幾乎沒有影像專屬設計的語言模型架構，只要 tokenizer 與訓練夠好，就能在類別條件 ImageNet 上具競爭力。(ii) 改變預測單位：VAR 的 next-scale 預測 [6] 與 MaskGIT、Muse 的遮罩解碼 [8], [17] 減少了循序步數；MAR [7] 用每個 token 一個擴散頭，去掉了向量量化。(iii) 統一的多模態模型：同一套下一個 token 預測機制在單一模型中生成文字與影像 token，tokenizer 也被設計成同時服務理解與生成 [20], [21]。我們的解讀是：選擇**表示方式與預測單位**，如今至少和選擇骨幹網路一樣重要。

開放問題。這些綜述反覆指出以下幾點：

- *Tokenizer* 品質。重建能力決定了生成品質的上限；小字、人臉與精細幾何通常是最先受損的部分。如何在不丟細節的前提下提升 codebook 使用率與語意結構，仍是活躍的研究主題 [20], [22]。
- 離散與連續 *token*。離散 token 適合 LLM 的機制；連續 token 重建得更好，但需要連續的條件模型，例如擴散頭 [7]。
- 速度。循序解碼受延遲限制。next-scale、遮罩與平行解碼以獨立假設換取更少步數。
- 評估。token 似然、FID 類指標、提示貼合度與人類偏好衡量的是不同的東西；有損 tokenizer 的 token 似然並不是像素似然。
- 影片與長上下文。token 數量隨時間暴增，時間一致性、物件恆存性與漂移都需要專門的評估 [1]。

重點整理

- 自迴歸是一種分布分解，不限定使用 Transformer 架構。
- 教師強制提供真實前綴；因果遮罩與位移輸入須防止目標洩漏。

- token 化縮短序列，也限制解碼器的表示能力。
- token、尺度與遮罩預測採用不同分解或更新規則，須比較其相依假設。
- 溫度與截斷改變取樣機率；快取重用運算，而不改變預期的條件分布。
- 以相同資料、表示、引導及成本條件，比較自迴歸與擴散流程。

練習

1. **洩漏答案**。把 `BinaryAR` 的遮罩改成包含對角線（`diagonal=0`）。說明為什麼訓練損失可以降到接近零，取樣卻會失敗。

   提示

   每個輸出現在都能讀到自己的目標並直接複製，所以訓練損失崩落。取樣時，計算目前位置的 logit 時它的值還是 0（尚未抽樣），學到的「複製」規則因此產生無意義的結果。程式中的因果性 assert 也會失敗。

2. **每維位元數**。對重複列的 4×4 二值資料，計算以下情況的每像素位元數：(a) 真實分布；(b) 把每個像素當成獨立公平硬幣的模型；(c) 完美學會第一列、但把後面各列當成獨立公平硬幣的模型。

   提示

   (a) 4 位元 / 16 像素 = 0.25。(b) 16 / 16 = 1.0。(c) 每張 4 + 12 = 16 位元，同樣是每像素 1.0 位元：學會第一列的邊際分布沒有幫助，因為它的位元本來就是均勻的。所有的改善都來自建模後面各列對第一列的依賴。

3. **Tokenizer 預算**。一張 512×512 影像以 8 倍或 16 倍降取樣做 tokenization。分別計算序列長度，以及完整自注意力的成本比。用 16 倍時可能失去什麼？

   提示

   $64 \times 64 = 4096$ 對 $32 \times 32 = 1024$ 個 token。注意力成本隨長度平方成長，所以比值是 16。用 16 倍時，每個 token 要描述一個 16×16 區塊，除非 codebook 或解碼器能補償，否則小字和細線更容易遺失。

4. **盲點**。把遮罩卷積的程式改成 7 層、11×11 輸入、目標在中心。有多少較早的像素永遠看不到？接著畫出垂直堆疊加水平堆疊如何消除它們。

   提示

   盲點是目標右上方、往左上擴張的感受野永遠到不了的階梯區域；計算光柵順序中較早、梯度為零的格子，答案是 10。垂直堆疊看到上方完整的列（沒有左右限制），並把結果送進只處理目前這列的水平堆疊，於是每個較早的像素都變得可達。

5. **遮罩解碼步數**。用 $n_t = \lceil \gamma(t/T)N \rceil$、餘弦排程、$N = 256$、$T = 8$，計算每一步固定多少 token。為什麼排程一開始只固定少數 token？

提示

剩餘遮罩數為 252、237、213、182、143、98、50、0，所以每步固定 4、15、24、31、39、45、48、50 個 token（圖中使用 floor，數字最多差一）。在精確算術中 $\gamma(1) = 0$；但浮點數的 $\cos(\pi/2)$ 是極小的正數，直接取上整數會留下一個被遮住的 token，這也是程式改用 floor 的另一個原因。一開始模型的上下文很少，平行猜測最不可靠、也最可能互相矛盾；只固定少數 token，能讓後面的步驟以它們為條件。

6. **為方法分類。**對 PixelCNN、VQGAN transformer、Parti、LlamaGen、VAR、MaskGIT 與 MAR，寫出預測單位、表示方式（像素、離散 token、連續 token、多尺度圖）、條件分布類型，以及 16×16 token 網格所需的循序步數。

提示

PixelCNN：光柵順序的像素，256 類 softmax，每個值一步。VQGAN transformer、Parti、LlamaGen：光柵順序的離散 token，codebook 上的 softmax，256 步。VAR：多尺度離散圖，每個 token 一個 softmax、尺度內平行，每個尺度一步。MaskGIT：離散 token，雙向遮罩預測，T 步（論文中為 8）。MAR：隨機順序分組的連續 token，每個 token 一個擴散頭，步數等於組數（每組另有內層擴散步驟）。

參考文獻

1. J. Xiong, G. Liu, L. Huang et al., "Autoregressive Models in Vision: A Survey," *Transactions on Machine Learning Research* (*TMLR*), 2025 (arXiv:2411.05902, 2024). arXiv
2. A. van den Oord, N. Kalchbrenner and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," in *Proc. ICML*, PMLR 48, 2016. PMLR
3. A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves and K. Kavukcuoglu, "Conditional Image Generation with PixelCNN Decoders," in *NeurIPS*, 2016. NeurIPS
4. A. van den Oord, O. Vinyals and K. Kavukcuoglu, "Neural Discrete Representation Learning," in *NeurIPS*, 2017. NeurIPS
5. P. Esser, R. Rombach and B. Ommer, "Taming Transformers for High-Resolution Image Synthesis," in *Proc. IEEE/CVF CVPR*, 2021. CVF
6. K. Tian, Y. Jiang, Z. Yuan, B. Peng and L. Wang, "Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction," in *NeurIPS*, 2024 (Best Paper Award). arXiv
7. T. Li, Y. Tian, H. Li, M. Deng and K. He, "Autoregressive Image Generation without Vector Quantization," in *NeurIPS*, 2024. arXiv
8. H. Chang, H. Zhang, L. Jiang, C. Liu and W. T. Freeman, "MaskGIT: Masked Generative Image Transformer," in *Proc. IEEE/CVF CVPR*, 2022. CVF
9. S. J. D. Prince, *Understanding Deep Learning*, MIT Press, 2023. Book
10. A. Razavi, A. van den Oord and O. Vinyals, "Generating Diverse High-Fidelity Images with VQ-VAE-2," in *NeurIPS*, 2019. NeurIPS
11. M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan and I. Sutskever, "Generative Pretraining From Pixels," in *Proc. ICML*, PMLR 119, 2020. PMLR
12. J. Yu, X. Li, J. Y. Koh et al., "Vector-quantized Image Modeling with Improved VQGAN," in *ICLR*, 2022. arXiv

13. J. Yu, Y. Xu, J. Y. Koh et al., “Scaling Autoregressive Models for Content-Rich Text-to-Image Generation,” *Transactions on Machine Learning Research*, 2022. arXiv
14. P. Sun, Y. Jiang, S. Chen, S. Zhang, B. Peng, P. Luo and Z. Yuan, “Autoregressive Model Beats Diffusion: Llama for Scalable Image Generation,” arXiv:2406.06525, 2024. arXiv
15. D. Lee, C. Kim, S. Kim, M. Cho and W.-S. Han, “Autoregressive Image Generation using Residual Quantization,” in *Proc. IEEE/CVF CVPR*, 2022. CVF
16. F. Mentzer, D. Minnen, E. Agustsson and M. Tschannen, “Finite Scalar Quantization: VQ-VAE Made Simple,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2024. arXiv
17. H. Chang, H. Zhang, J. Barber et al., “Muse: Text-To-Image Generation via Masked Generative Transformers,” in *Proc. ICML*, PMLR 202, 2023. PMLR
18. J. Ho and T. Salimans, “Classifier-Free Diffusion Guidance,” arXiv:2207.12598, 2022 (short version at the NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications). arXiv
19. S. Bond-Taylor, A. Leach, Y. Long and C. G. Willcocks, “Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models,” *IEEE TPAMI*, vol. 44, no. 11, 2022. doi
20. J. Li, Y. Fu, J. Liu, L. Cao, W. Ji, M. Yang, I. King and M.-H. Yang, “Discrete Tokenization for Multimodal LLMs: A Comprehensive Survey,” *IEEE TPAMI*, 2026 (arXiv:2507.22920). doi
21. S. Zhao, X. Zhang, J. Guo et al., “Unified Multimodal Understanding and Generation Models: Advances, Challenges, and Opportunities,” arXiv:2505.02567, 2025; revised 2026. arXiv
22. J. Jia, J. Gao, B. Xue et al., “From Principles to Applications: A Comprehensive Survey of Discrete Tokenizers in Generation, Comprehension, Recommendation, and Information Retrieval,” arXiv:2502.12448, 2025. arXiv
23. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” *CVPR*, 2009, pp. 248–255. doi
24. H. Touvron, T. Lavril, G. Izacard et al., “LLaMA: Open and Efficient Foundation Language Models,” arXiv:2302.13971, 2023. arXiv

ML 10・基礎模型與視覺語言學習

什麼樣的模型稱為基礎模型：縮放定律、預訓練目標、CLIP 與 SigLIP、DINOv2 特徵、提示式分割、多模態語言模型、以 linear probe 與 LoRA 適配，以及如何評估風險。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/foundation-models/

你將學到

- 從預訓練與重用定義基礎模型。
- 在實際量測範圍內解讀縮放法則。
- 比較監督來源與可重用的視覺能力。
- 追蹤影像—文字對齊、視覺提示與多模態連接器。
- 選擇調適方法，測試任務表現、穩健性與限制。

先看全貌

在電腦視覺的大部分歷史裡，每個任務各有一個模型：一個網路分類花朵、另一個偵測汽車、再一個分割腫瘤，每個都從頭訓練，或從 ImageNet [43] 分類器開始。**基礎模型**則在廣泛的資料上、以大規模訓練一次，接著**適配**到許多下游任務 [1]。這個詞描述的是模型如何被訓練與重用，而不是一種新的網路層。DINOv2 [15] 或 DINOv3 [16] 影像編碼器、CLIP 式影像文字模型 [3]、SAM [5] 這類提示式分割器、多模態語言模型，以及廣泛預訓練的影像生成器，都可以扮演這個角色。

> **用最簡單的話說**
>
> 與其每換一份工作就從零教一個新學生，不如找一位已經看過幾百萬張圖、讀過幾百萬句圖說的人。他已經知道邊緣、輪子、臉和狗長什麼樣子。遇到新工作時，你只需要給他幾個例子、一句簡短的說明，或一件小小的新工具——不必重新上一整套學校。

為什麼重要？有三個理由。第一，**標註很昂貴**，而預訓練把大部分學習移到不需要人工標籤的資料上（圖說、影像本身）。第二，**一個骨幹網路服務許多任務**，讓工程投入更集中。第三，它改變了**介面**：很多時候不必重新訓練，只要用一段文字提示、一次點擊或幾個範例就能指定任務。Bommasani 等人把這種方法的集中稱為同質化（homogenization），並提醒它同時也集中了失敗：共用骨幹的一個缺陷，會被建立在它之上的每個系統繼承 [1]。

本章延續 ML 06・表示學習（什麼樣的特徵可以重用）、ML 05・Vision Transformer（常用骨幹），以及生成模型章節 ML 08 與 ML 09。Bishop 與 Bishop 的教科書 [14] 對遷移學習與表示有系統性的說明，可作為參考。本章沿用 Awais 等人的視覺基礎模型綜述 [39] 與 Caffagni 等人的多模態綜述 [2] 所畫出的架構地圖，並引用為 2026 年 10 月版本查證過的代表性報告。這些例子用來說明模型家族，不是在替商業產品排名，也不宣稱某個 checkpoint 在所有情況都最好。

什麼樣的模型稱為「基礎」模型

白話版。基礎模型是在廣泛資料上、以通用目標預訓練的大型模型，它的內部表示或使用介面可以重用在從未專門訓練過的任務上。

Bommasani 等人 [1] 在 2021 年提出這個名稱，並強調兩個性質：

- **湧現（emergence）**。能力來自規模與訓練目標，而不是被明確設計出來。例如 CLIP 只被訓練去配對影像與圖說，卻能把影像分類到用文字命名的類別 [3]；DINO 完全沒有使用標籤，它的注意力圖卻能勾勒出物體輪廓 [24]。
- **同質化（homogenization）**。許多應用收斂到少數幾個預訓練骨幹與訓練配方。

使用「湧現」時要小心。本章中它只表示一種不是訓練目標的能力，並不代表無法預測或神秘的跳躍；某個能力看起來是否「突然」出現，常常取決於用來衡量它的指標。

形式上，令 f_θ 為在上游資料集 $\mathcal{D}_{\text{up}}$ 上以目標 $\mathcal{L}_{\text{pre}}$ 預訓練的模型：

$$\theta^\star = \arg\min_\theta \; \mathbb{E}_{x \sim \mathcal{D}_{\text{up}}}[\mathcal{L}_{\text{pre}}(f_\theta; x)].$$

下游任務 k 提供一個（通常很小的）資料集 $\mathcal{D}_k$ 與任務損失 ℓ_k。適配就是選一小組新的或被修改的參數 ϕ_k，並求解

$$\phi_k^\star = \arg\min_{\phi_k} \sum_{(x,y)\in\mathcal{D}_k} \ell_k(h_{\phi_k}(f_{\theta^\star}, x), y),$$

其中 h_{ϕ_k} 是我們放在預訓練模型上方或內部的任何東西：線性分類頭、adapter、LoRA 更新、提示，或什麼都不放（純提示，$\phi_k = \emptyset$）。整章都圍繞上面兩行：$\mathcal{L}_{\text{pre}}$ 與 $\mathcal{D}_{\text{up}}$ 應該是什麼，以及 ϕ_k 可以多小。

基礎模型**不一定**要產生文字。一個凍結的影像編碼器，若它的特徵供一百個不同分類器使用，就是基礎模型。反過來，為單一狹窄任務訓練的大模型，不論多大都不算。

縮放定律

白話版。縮放法則擬合損失如何隨模型大小、資料與計算量改變，有助規劃可比條件下的訓練，但不保證外推增益。

Kaplan 等人 [17] 訓練了多組 Transformer 語言模型，報告損失與模型大小、資料量及計算量之間呈冪律關係，部分趨勢跨越超過七個數量級；在測試範圍內，寬度與深度的影響很小。對模型大小 N 的冪律寫成

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N},$$

其中 L 是保留資料上的交叉熵損失，N 是不含嵌入層的參數數量，N_c 是與 N 同單位的擬合常數，$\alpha_N > 0$ 是擬合指數。取對數得 $\log L = \alpha_N \log N_c - \alpha_N \log N$：斜率為 $-\alpha_N$ 的直線。

真實曲線會彎曲，因為有一部分損失永遠消不掉（資料本身就有雜訊或歧義）。因此 Hoffmann 等人 [18] 擬合了帶有不可約項的形式：

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}},$$

其中 D 是訓練 token 數，E 是不可約損失，A, B, α, β 是擬合常數。對稠密 Transformer，訓練計算量常用的近似是 $C \approx 6ND$ 次浮點運算 [17]（每個參數、每個 token，前向約 2 次、反向約 4 次）。固定 C，在 N 上最小化 L（令 $D = C/6N$），就得到**計算最佳**（compute-optimal）的模型大小。Hoffmann 等人訓練了超過 400 個語言模型，結論是：在計算最佳的訓練下，模型大小與訓練 token 數應等比例擴大——模型大小每加倍，token 數也應加倍 [18]。這和 Kaplan 等人較早「用相對適量的資料訓練非常大的模型」的建議不同 [17]；這個分歧提醒我們，擬合出的指數取決於實驗流程（學習率排程、納入哪些實驗、計算量怎麼算）。

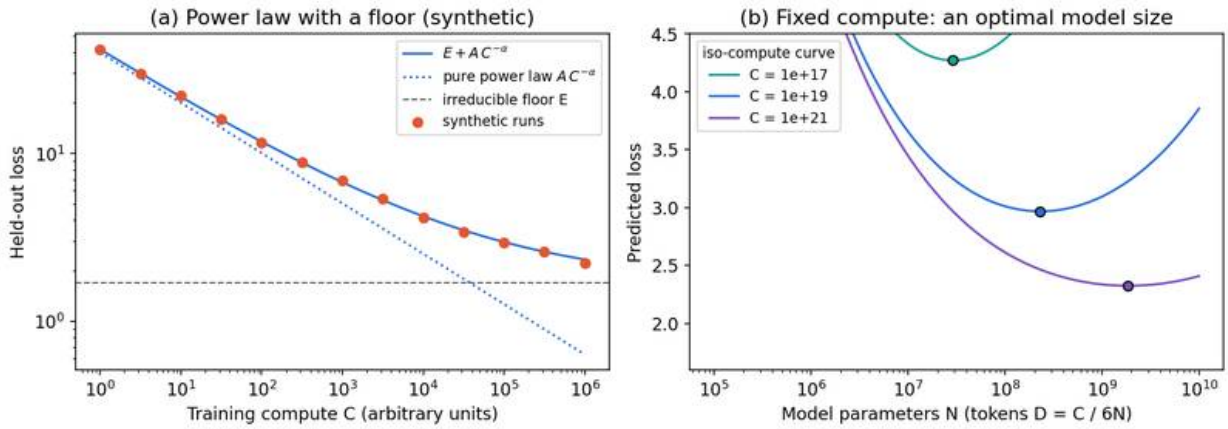


圖 *1* — (*a*) 帶有不可約下限 E 的冪律：純冪律（點線）會一直下降，在大規模時過度樂觀。(*b*) 固定計算量時，模型太小會欠擬合，太大則看到的 *token* 太少，因此每個預算都有最佳大小。所有常數皆為合成。

視覺領域也有自己的縮放研究。Zhai 等人 [19] 針對在大型標註資料上訓練的 Vision Transformer [21]，刻畫錯誤率、資料量與計算量的關係，並擴大到二十億參數的模型。Cherti 等人 [20] 在公開的 LAION 資料集 [34] 上訓練 CLIP 式模型，發現零樣本分類、檢索、linear probing 與微調都呈冪律縮放——但也發現 OpenAI 的 CLIP 與 OpenCLIP 在架構相同、配方相近的情況下，縮放行為仍不同，因為兩者的**訓練資料分布**不同。資料不是一個純量。

程式：擬合縮放定律並外插

下面的程式在 13 個合成訓練實驗中，只用較便宜的那幾個擬合 $L(C) = E + A\,C^{-\alpha}$，再預測昂貴的那幾個。所有資料都是合成的；重點在流程。

```
import numpy as np
from scipy.optimize import curve_fit

rng = np.random.default_rng(0)
# Synthetic "runs": loss = E + A * C^(-alpha), with 2% multiplicative noise.
E_true, A_true, alpha_true = 1.7, 40.0, 0.3
compute = np.logspace(0, 6, 13)            # arbitrary compute units
loss = (E_true + A_true * compute**-alpha_true) * np.exp(.02 * rng.normal(size=13))
```

```
def law(c, E, A, alpha):
    return E + A * c**-alpha

# Fit only the cheaper runs, then extrapolate to the larger ones.
small = compute <= 1e4
(E, A, alpha), _ = curve_fit(law, compute[small], loss[small], p0=(1, 10, .5))
print(f"fit: E={E:.2f}  A={A:.1f}  alpha={alpha:.3f}")
for c, l in zip(compute[~small], loss[~small]):
    print(f"C={c:9.0f}  observed {l:.3f}  predicted {law(c, E, A, alpha):.3f}")

# Naive straight line in log-log space ignores the floor E.
slope = np.polyfit(np.log(compute[small]), np.log(loss[small]), 1)[0]
print(f"pure power-law slope without floor: {slope:.3f}")
```

輸出：擬合得到 $E \approx 1.73$、$A \approx 40.0$、$\alpha \approx 0.300$；外插的損失與含雜訊的觀測值相差約 7% 以內；若在對數–對數空間畫一條不含下限的直線，斜率約為 -0.25，指數是錯的。有兩個教訓可以帶到實務：一定要納入下限；跨越多個數量級的外插應視為帶誤差範圍的預測。

縮放定律沒告訴你的事

- 它預測的是預訓練分布上的**平均損失**，不是你的任務準確率。下游指標可能平滑改善、飽和，或呈階梯式變化。
- 它只在**固定配方與資料分布**下成立 [20]。更好的資料過濾或不同的目標都會改變常數。
- 它完全不談**推論成本**。用更多資料訓練的較小模型，在相同損失下部署可能更便宜，這也是許多上線模型會訓練超過計算最佳點的原因。
- 資料是有限的。高品質資料用完後，不論再加多少計算量，B/D^{β} 這一項都不會再縮小。

預訓練目標：目標值從哪裡來？

白話版。每種預訓練方法都會發明一個預測任務。差別在於「答案」從哪裡來：人工標籤、圖說、同一張影像被遮住的部分、另一個網路，或下一個 token。

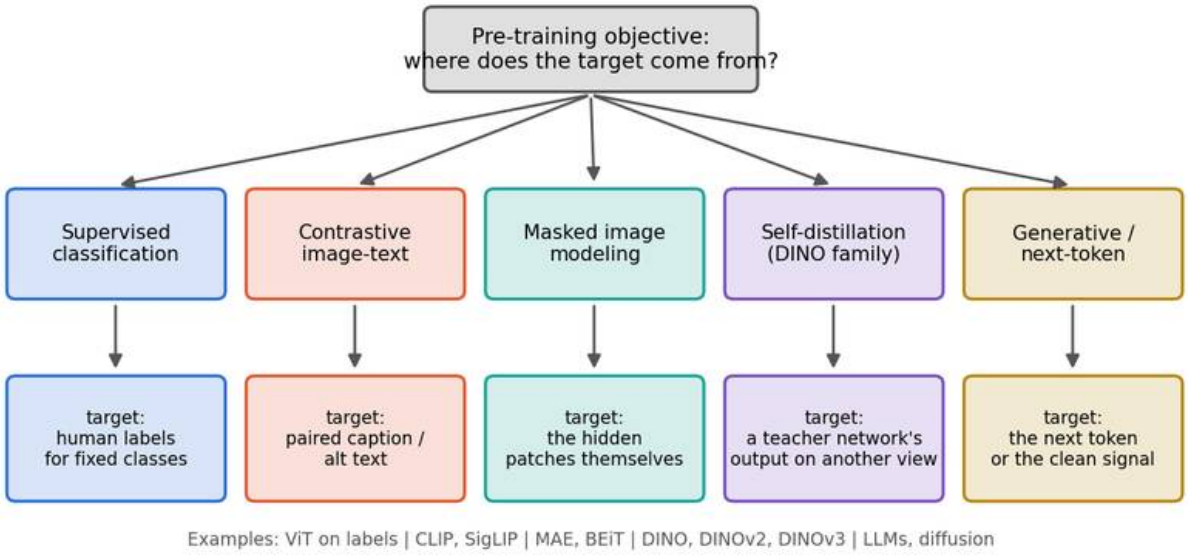


圖 2 — 依訓練目標值的來源，把預訓練目標分成五個家族。

目標	目標值來源	代表模型	典型強項
監督式分類	人工標籤	在大型標註資料上訓練的 ViT [19], [21]	在標籤空間內有強的全域語意
影像文字對比	成對的圖說／ alt text	CLIP [3]、ALIGN [25]、SigLIP [4]	以文字定義的辨識、檢索
遮罩影像建模	同一張影像被遮住的 patch	BEiT [23]、MAE [22]	微調後很強；保留局部細節
自蒸餾	教師網路在另一個視角上的輸出	DINO [24]、DINOv2 [15]、DINOv3 [16]	凍結特徵、密集對應
生成式／下一個 token	下一個 token 或乾淨訊號	LLM、擴散（ML 07）、自迴歸（ML 08）	生成；語言介面

監督式預訓練

在大型標註資料集上用交叉熵訓練分類器，然後拿掉最後一層。這是十年來的 ImageNet 配方，在大規模下仍然有效 [19], [21]。它的弱點是特徵被固定的標籤集合塑形：標籤不需要的區分可能會被丟掉。

影像文字對比學習

網路影像本來就附有圖說，因此 CLIP [3] 與 ALIGN [25] 同時訓練影像編碼器與文字編碼器，讓批次中配對正確的組合分數高於配錯的組合。ALIGN 顯示，只做最少過濾、雜訊很多的大型 alt-text 語料，仍能得到強的表示 [25]。損失函數的細節在下一節。

遮罩影像建模

把影像大部分遮住，再預測缺少的部分。BEiT [23] 遮住 patch 後預測它們的離散視覺 token，借用 BERT [44] 的遮罩語言建模概念。MAE [22] 直接預測原始像素，並觀察到遮住高比例（例如 75%）的 patch 會形成一個不平凡的任務；它的編碼器只看可見的 patch，再由輕量解碼器重建其餘部分，使訓練很有效率。令 $\mathcal{M}$ 為被遮住的 patch 索引集合，x_p 為 patch p 的（正規化）像素，$\hat{x}_p$ 為重建結果，

$$\mathcal{L}_{\text{MAE}} = \frac{1}{|\mathcal{M}|} \sum_{p \in \mathcal{M}} \|\hat{x}_p - x_p\|_2^2.$$

損失只在被遮住的 patch 上計算：預測可見的 patch 只是複製，沒有意義。遮罩建模的特徵在微調後常常非常好，但作為 linear probe 的凍結特徵時，通常比對比式或自蒸餾的特徵弱。

自蒸餾：DINO 家族

DINO [24]——作者形容為「沒有標籤的自蒸餾」——訓練一個**學生**網路 g_s，讓它在同一張影像的不同增強視角（裁切）上去配合**教師**網路 g_t。兩者都輸出 K 維分數，再用溫度 τ_s 與 τ_t 的 softmax 轉成機率：

$$P_s(x') = \text{softmax}\left(\frac{g_s(x')}{\tau_s}\right), \qquad P_t(x) = \text{softmax}\left(\frac{g_t(x) - c}{\tau_t}\right), \qquad \mathcal{L} = -\sum_{k=1}^{K} P_t(x)_k \log P_s(x')_k.$$

其中 x 與 x' 是兩個視角，c 是教師輸出的移動平均（**置中**，centering），較小的 τ_t 讓教師分布變得尖銳（**銳化**，sharpening）。置中避免某一維獨大；銳化避免輸出變成均勻分布；兩者合起來防止**崩塌**（collapse）——也就是每張影像都對應到相同輸出的平凡解。教師不接收梯度，它的權重是學生權重的指數移動平均（動量編碼器）：

$$\theta_t \leftarrow \lambda\, \theta_t + (1 - \lambda)\, \theta_s, \qquad 0 < \lambda < 1.$$

DINO 也使用多重裁切（multi-crop）訓練：學生看幾個小的局部裁切，教師看全域裁切，迫使模型學會局部與全域的對應。作者報告，所得到的 ViT 特徵含有明確的物體分割資訊，這在監督式 ViT 或卷積網路中沒有那麼明顯；而且這些特徵很適合作為 k 最近鄰分類器 [24]。

DINOv2 [15] 把這個配方擴大，關鍵是整理資料：用自動化流程建立專門、多樣且經過整理的影像資料集，而不是使用未整理的網路影像；訓練一個 10 億參數的 ViT，再蒸餾成較小的模型。作者報告這些凍結特徵在他們大多數影像層級與像素層級的基準上超越 OpenCLIP [15]。DINOv3 [16] 進一步擴大資料與模型，並提出 **Gram anchoring**，針對一個已知問題：長時間訓練時，密集（逐 patch）特徵圖的品質會退化。它也加入事後（post-hoc）策略，提升對解析度、模型大小與文字對齊的彈性。

為什麼密集特徵重要。分類器每張影像只需要一個向量；分割、深度估計、對應與追蹤則需要**每個 patch** 都有好的向量。自蒸餾編碼器之所以成為這些密集任務常用的凍結骨幹，正是因為它們的 patch 特徵不需要微調就保有語意。

生成式預訓練

預測下一個 token（語言模型、ML 09 中的自迴歸影像模型）或去雜訊（ML 08 的擴散模型）也是預訓練目標。對語言來說，它產生最有彈性的介面——用自然語言下指令——這也是多模態模型把視覺接到預訓練語言模型上，而不是反過來的原因。

影像與文字的共享空間：CLIP 與 SigLIP

白話版。對齊配對影像與文字嵌入，再依候選描述為影像評分；文字定義分類仍依賴提示及學得的涵蓋範圍。

CLIP [3] 把影像嵌入 v_i 與文字嵌入 t_j 正規化成單位長度，並以可學習的溫度 τ 形成分數 $s_{ij} = v_i^\top t_j / \tau$。對一個含 B 組配對樣本的批次，

$$\mathcal{L}_{\text{CLIP}} = \frac{1}{2}\left[\frac{1}{B}\sum_i -\log \frac{\exp(s_{ii})}{\sum_j \exp(s_{ij})} + \frac{1}{B}\sum_j -\log \frac{\exp(s_{jj})}{\sum_i \exp(s_{ij})}\right].$$

每張影像應在 B 個圖說中選出配對的那一個（第一項，逐列 softmax），每個圖說也應選出配對的影像（第二項，逐欄）。批次中其他組合都當作負樣本。訓練使用成對的文字監督，通常來自既有的圖說或 alt text；下游的「零樣本」使用並不代表模型訓練時沒有監督。

推論時，把「a photo of a dog」這類候選描述編碼，再與影像比較。文字嵌入取代了學習式分類器的類別權重：

$$\hat{y} = \arg\max_k v(x)^\top t(\text{prompt}_k).$$

候選集合與措辭都會影響結果；把多個模板（「a photo of a k」「a close-up of a k」）的嵌入平均，是降低敏感度的常見作法。零樣本的意思是在所述流程下沒有針對任務訓練參數；它不保證這些類別、甚至測試影像本身，不曾出現在預訓練資料中（見〈風險與評估〉）。

程式：配對損失與檢索機制

以下是**合成嵌入**，用來讓計算一目了然。它們不是預訓練模型的輸出，配對結果也不能證明真實世界的零樣本準確率。

```
import torch
from torch.nn import functional as F

images = F.normalize(torch.tensor([[1., .1, 0.], [.1, 1., 0.],
                                   [0., .1, 1.]]), dim=-1)
texts = F.normalize(torch.eye(3), dim=-1)
scores = images @ texts.T / .1
labels = torch.arange(3)
loss = (F.cross_entropy(scores, labels) +
        F.cross_entropy(scores.T, labels))/2
print("matching text:", scores.argmax(1).tolist())
```

```
print("paired loss:", round(loss.item(), 4))
ambiguous = F.normalize(torch.tensor([[.5, .5, .2]]), dim=-1)
print("cosines:", (ambiguous @ texts.T).tolist())
```

它會印出 `matching text: [0, 1, 2]`、約 `0.0002` 的配對損失，以及模糊影像約 `0.68, 0.68, 0.27` 的餘弦值：它與兩個描述一樣接近，任何 softmax 都無法告訴你哪一個才對。

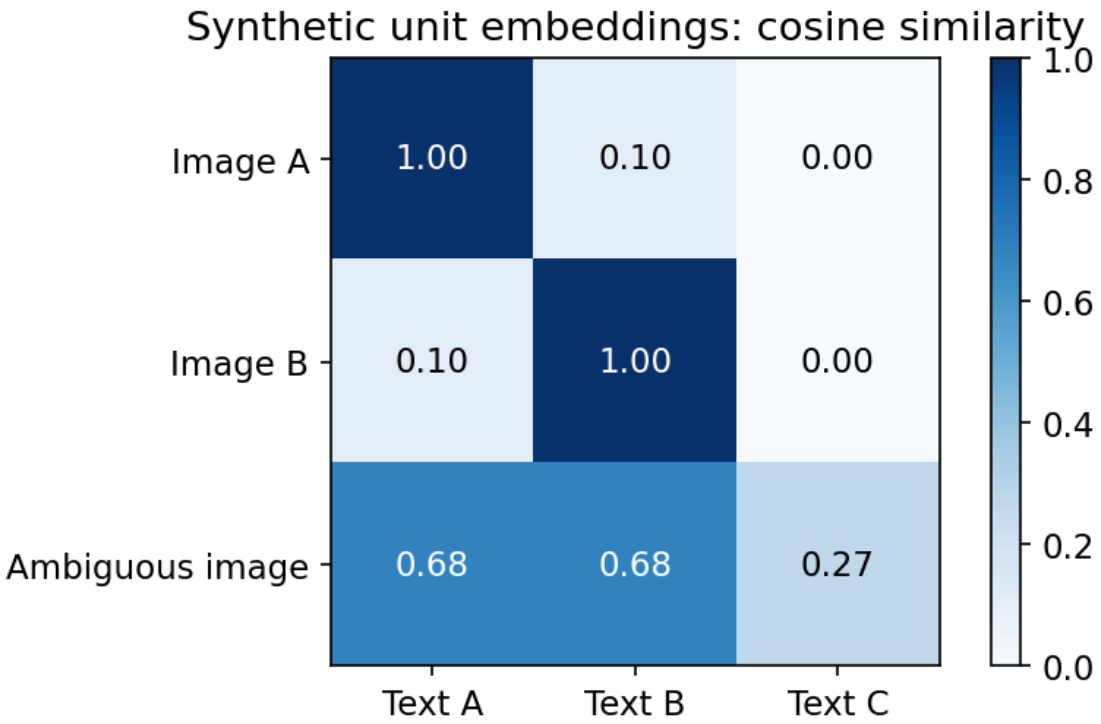


圖 3 — 餘弦相似度可以為候選排序，但它不是概念存在與否的校準機率。

SigLIP：用 sigmoid 取代 softmax

SigLIP [4] 把批次正規化的 softmax 換成對每一組影像文字配對各自做二元判斷。令配對正確時 $y_{ij} = +1$、否則 -1，並有可學習的尺度 a 與偏差 b，

$$\mathcal{L}_{\text{SigLIP}} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{j=1}^{B}\log \sigma(y_{ij}(a\, v_i^{\top} t_j + b)),$$

其中 $\sigma(u) = 1/(1+e^{-u})$，因此每一項等於 $\log(1+\exp(-y_{ij}(a\, v_i^{\top} t_j + b)))$。因為沒有任何一項需要對整個批次求和，這個損失不需要跨裝置的全域正規化，讓大批次訓練更簡單。由於正樣本只有 B 個、負樣本有 $B^2 - B$ 個，偏差 b 的初始值會讓初始預測強烈偏向負類。

共享嵌入支援檢索與開放詞彙辨識，但細粒度計數、否定（「一隻**沒有**項圈的狗」）、空間關係與罕見的領域詞彙都是已知弱點。這些需求要直接評估，不要從亮眼的整體分類分數推論。

僅線上版：*CLIP*：從影像文字配對到零樣本分類器 — *https://blog.shyandram.dev/zh-tw/learn/ml/foundation-models/papers/clip* 從影像文字配對到零樣本分類器/

提示式分割：Segment Anything 系列

白話版。提示告訴模型你想要哪個區域或概念，輸出則是遮罩。

SAM [5] 結合一個較重的影像編碼器（每張影像跑一次）、輕量的提示編碼器與輕量的遮罩解碼器，因此同一張影像上的許多提示都能很快回應。空間提示——點、框或粗略遮罩——不需要固定的類別詞彙就能選取物體。因為單次點擊有歧義（點在襯衫上可能指襯衫，也可能指整個人），SAM 會預測幾個候選遮罩並附上信心分數。它在 SA-1B 上訓練：超過 10 億個遮罩、1100 萬張具授權且尊重隱私的影像，透過「模型在迴圈中」的資料引擎蒐集 [5]。

SAM 2 [6] 把提示式分割延伸到影片，加入串流記憶：模型儲存過去影格與被提示物體的特徵，並沿時間傳遞遮罩；在之後影格上的修正點擊會更新記憶。

SAM 3 [7] 於 2025 年發布，報告在 2026 年 3 月修訂，提出**提示式概念分割**（promptable concept segmentation）：給定一個短名詞片語（例如「yellow school bus」）、影像範例或兩者，它會回傳**所有**符合實例的遮罩與身分。它的影像層級偵測器與基於記憶的影片追蹤器共用一個骨幹，並以 **presence head** 把「這個概念是否出現」的辨識與「實例在哪裡」的定位分開 [7]。

這些介面提供的資訊不同：空間點說明在哪裡，名詞片語說明是什麼，範例說明長什麼樣子。請報告建立提示的成本，以及提示來自人、真值標註或自動偵測器。用真值框當提示得到的「零樣本分割」分數，其實隱藏了大量的標註資訊。

多模態大型語言模型

白話版。把視覺特徵轉成語言模型讀得懂的「字」，再教它遵循視覺指令。

常見的設計是

$$\text{image/video} \xrightarrow{\text{vision encoder}} h \xrightarrow{\text{connector}} z \xrightarrow{\text{language decoder with text}} \text{answer}.$$

這裡 h 是視覺特徵網格（對 ViT 而言，每個 patch 一個向量），z 是位於語言模型 token 嵌入空間中的向量序列，解碼器把 z 當成一般文字的嵌入來處理。

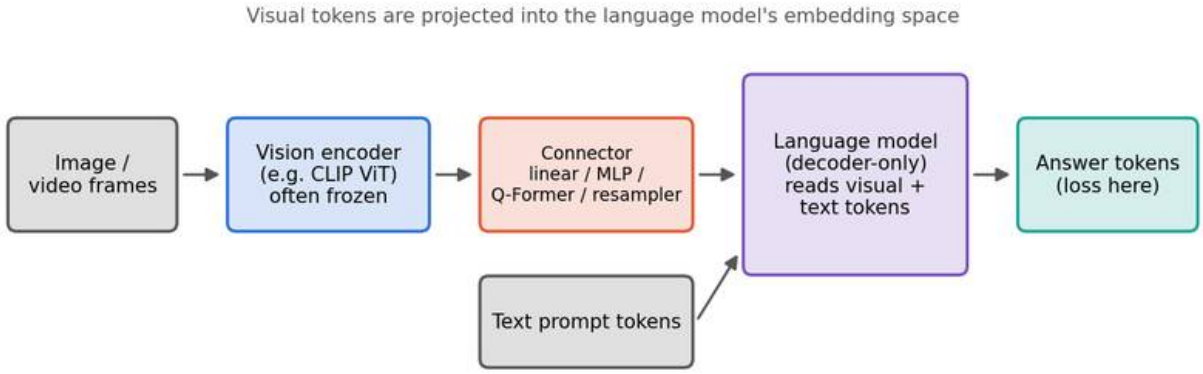


圖 *4* — 典型的多模態 *LLM* 流程。視覺特徵被轉換成語言模型嵌入空間中的 *token*。

連接器

- **線性或 MLP 投影。** LLaVA [8] 用單一投影把 CLIP 特徵映射到語言模型的嵌入空間；LLaVA-1.5 [28] 報告簡單的 MLP 連接器（搭配 336 像素解析度的 CLIP ViT-L 與學術 VQA 資料）出奇地強大且節省資料。每個 patch 變成一個視覺 token。
- **可學習查詢。** BLIP-2 [26] 在凍結的影像編碼器與凍結的語言模型之間插入輕量的 Querying Transformer（Q-Former）。一組固定的可學習查詢向量對影像特徵做交叉注意力，因此語言模型永遠只看到少量、固定數目的視覺 token。BLIP-2 分兩階段訓練 Q-Former：先對凍結的影像編碼器做表示學習，再對凍結的語言模型做生成學習。
- **重取樣與交叉注意力。** Flamingo [27] 橋接凍結的視覺與語言模型，並能處理任意交錯的影像文字序列，使模型能從直接寫在提示中的範例做少樣本學習。

取捨在於 token 數與細節：每個 patch 一個 token 保留細節，但序列很長（多張影像或影片時很昂貴）；少數可學習查詢很便宜，但可能遺失小字。

視覺指令微調

LLaVA [8] 用只處理語言的 GPT-4 [45]，根據影像的圖說與框生成多模態指令遵循資料，再分兩階段訓練：先對齊連接器（特徵對齊），再在指令資料上微調連接器與語言模型。損失是一般的下一個 token 交叉熵，只算在**答案** token 上：

$$\mathcal{L} = -\sum_{i \in \mathcal{A}} \log p_\theta(w_i \mid z,\, w_{<i}),$$

其中 w_i 是文字 token，$\mathcal{A}$ 是屬於助理回答的位置集合，z 是視覺 token。指令與提示 token 作為上下文可見，但不計入損失。

Qwen3-VL 的 2025 年報告 [9] 呈現目前的方向：一系列稠密與混合專家（mixture-of-experts）模型，原生支援最長 256K token 的影像、影片與文字交錯上下文。它是代表性報告，不代表每個任務都會因更長的上下文或更多生成的推理而受益。

示意圖沒畫出的事

閱讀文件可能需要以切塊（tiling）或動態解析度保留小字；影片問題需要時間戳與時間順序。這些細節會改變 token 數與前處理；通用的「視覺編碼器 + LLM」示意圖只是起點。流暢的回答也可能提到影像中根本不存在的物體（**物體幻覺**），或數錯數量。請區分物體定位、OCR、空間推理、事實問答與風格上的指令遵循——它們需要不同的證據與指標（見〈風險與評估〉）。整個設計空間的綜述可參考 Yin 等人 [40] 與 Caffagni 等人 [2]。

適配基礎模型

白話版。用能解決目標任務的最便宜適配方式，並在看起來像實際部署的資料上檢查。

策略	可訓練元件	適合的起點
提示／上下文範例	無	模型介面本來就符合任務
凍結特徵 + 線性分類頭（linear probe）	小型監督式讀出層	標籤少、特徵遷移良好
Prompt tuning ／ visual prompt tuning	少數可學習的輸入 token	許多任務共用同一個凍結骨幹
Adapter ／ LoRA	小型插入模組或低秩模組	記憶體有限下的領域適配
部分微調	選定的區塊加分類頭	想做一些適配，又不想負擔完整最佳化器成本
完整微調（可搭配 LP-FT）	所有模型參數	資料與計算足夠；單靠遷移不夠

Linear probe

凍結編碼器 f，先算好特徵，再擬合線性分類器 $\hat{y} = \text{softmax}(Wf(x) + b)$。它便宜、不容易過擬合，也是衡量表示品質的標準**量尺**：如果線性讀出就能成功，代表資訊在特徵中是線性可取得的。

程式：為什麼預訓練特徵在標籤少時最有幫助

這個玩具實驗模擬整個故事。觀測值是 300 維，混合了 8 個**語意**因子（決定類別）、8 個很大的**干擾**因子（想像成光線與姿態）以及雜訊。「預訓練」看到 50 個上游類別，標籤充足，並學到一個 8 維投影（線性判別分析，代替深度編碼器）。下游有四個**新**類別，每類只有少數標籤。

```
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression

rng = np.random.default_rng(0)
D, k_sem, k_nui = 300, 8, 8
sem_map = rng.normal(size=(k_sem, D))   # semantic factors -> pixels
nui_map = rng.normal(size=(k_nui, D))   # nuisance factors (light, pose...)

def sample(centers, n):
```

```
    y = rng.integers(len(centers), size=n)
    z = centers[y] + 0.7 * rng.normal(size=(n, k_sem))
    u = 4.0 * rng.normal(size=(n, k_nui))          # large nuisance variation
    x = z @ sem_map + u @ nui_map + rng.normal(size=(n, D))
    return x, y

# 1) "Pre-training": many upstream classes with plenty of labels.
up_centers = rng.normal(size=(50, k_sem))
x_up, y_up = sample(up_centers, 10_000)
encoder = LinearDiscriminantAnalysis(n_components=k_sem).fit(x_up, y_up)

# 2) Downstream: four NEW classes, few labels, linear probe only.
new_centers = rng.normal(size=(4, k_sem))
x_te, y_te = sample(new_centers, 2_000)
f_te = encoder.transform(x_te)          # frozen features, computed once
for shots in [1, 5, 20, 100]:
    raw, feat = [], []
    for repeat in range(10):
        x_tr, y_tr = sample(new_centers, 4 * shots)
        raw.append(LogisticRegression(max_iter=3000)
                   .fit(x_tr, y_tr).score(x_te, y_te))
        feat.append(LogisticRegression(max_iter=3000)
                    .fit(encoder.transform(x_tr), y_tr).score(f_te, y_te))
    print(f"{shots:>3} labels/class | raw pixels {np.mean(raw):.3f}"
          f" | frozen features {np.mean(feat):.3f}")
```

在我們的機器上，它印出的準確率大約是：每類 1 個標籤時 0.29 對 0.71、5 個時 0.62 對 0.95、20 個時 0.96 對 0.99、100 個時 0.98 對 0.99（原始輸入對凍結特徵；隨機猜測為 0.25）。預訓練投影已經丟掉干擾方向，所以少數標籤就夠了；標籤夠多時，原始輸入上的 probe 也會自己學會忽略干擾，差距隨之消失。上游類別與下游類別不同——特徵之所以能遷移，是因為兩者共享相同的語意因子。如果下游任務依賴的正是預訓練丟掉的因子（這裡是干擾因子），那麼不論加多少標籤，凍結特徵都會失敗。這就是實務上要測試的失敗模式。

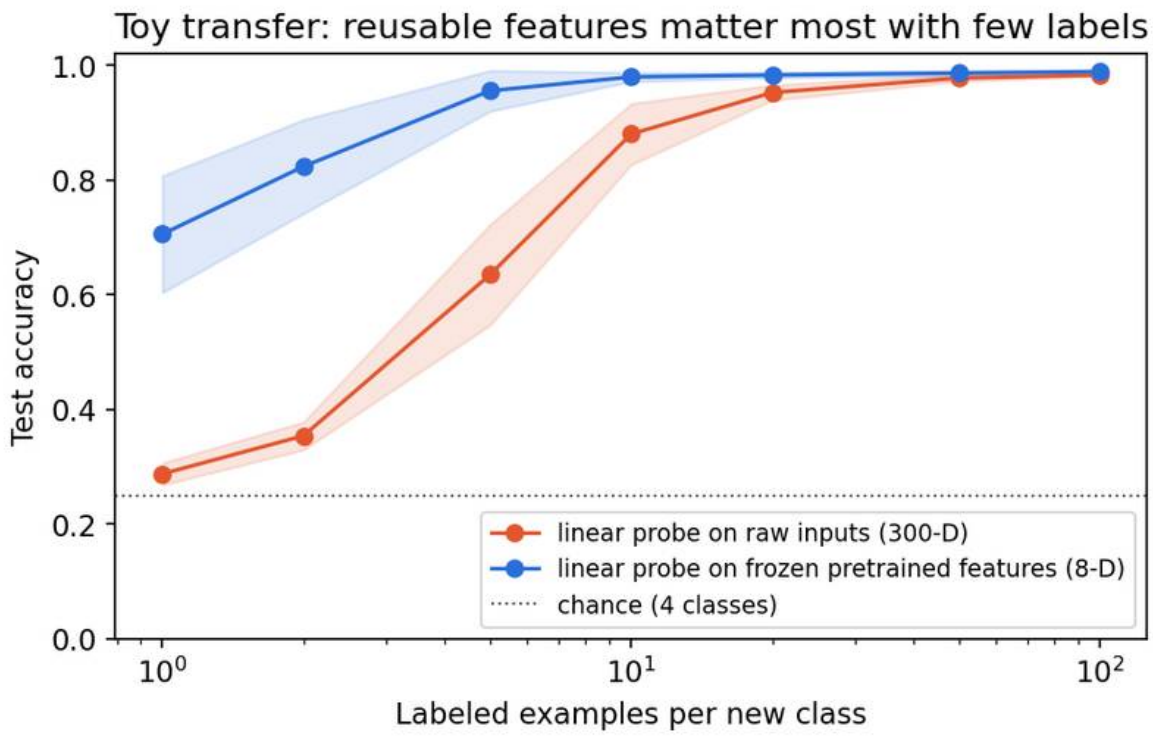


圖 5 — 在更多標籤預算下的玩具遷移實驗（*10* 次抽樣的平均 ± 一個標準差）。標籤稀少時，可重用特徵最重要。

微調、LP-FT 與穩健微調

完整微調會更新所有權重，資料足夠時通常有最好的分布內準確率。但 Kumar 等人 [32] 指出它可能扭曲好的預訓練特徵：在他們的基準上，微調平均而言分布內準確率較高，分布外準確率卻比 linear probing 低。他們的解釋是，隨機初始化的分類頭在訓練初期把又大又雜的梯度送進骨幹。**LP-FT**——先擬合 linear probe，再從這個分類頭開始微調全部參數——減少了這種扭曲，並在他們的實驗中勝過兩種單獨作法 [32]。**WiSE-FT** [33] 走另一條路：在權重空間線性內插零樣本模型與微調模型，$\theta_\gamma = (1-\gamma)\theta_{\text{zero-shot}} + \gamma\,\theta_{\text{ft}}$，對 CLIP 類模型在分布偏移下提升了穩健性，同時保持強的目標任務準確率。兩者都很便宜，在斷言「微調傷害了穩健性」之前值得先試。

參數高效微調（PEFT）

Adapter [29] 在每個 Transformer 區塊內插入小型瓶頸 MLP，只訓練它們，每個任務只增加少量參數，原網路保持不變。**Prompt tuning** 學習幾個額外的輸入 token；**Visual Prompt Tuning（VPT）** [30] 把這個作法用在 ViT 上，在凍結骨幹的 patch 序列前面加上可學習的 token。Xu 等人 [42] 綜覽 PEFT，提出加法式（additive）、部分（partial）、重參數化（reparameterized）、混合（hybrid）與統一（unified）微調的分類，並實驗比較參數與記憶體效率。

LoRA [10] 把凍結權重 $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ 的更新重參數化為

$$W = W_0 + \frac{\alpha}{r} BA, \qquad A \in \mathbb{R}^{r \times d_{\text{in}}}, \quad B \in \mathbb{R}^{d_{\text{out}} \times r},$$

其中秩 $r \ll \min(d_{\text{in}}, d_{\text{out}})$，$\alpha$ 是縮放常數。凍結 W_0，只學 A, B。可訓練權重從 $d_{\text{out}} d_{\text{in}}$ 降到 $r(d_{\text{in}} + d_{\text{out}})$。把 B 初始化為 0，使初始更新為零；A 隨機初始化，學習才能開始。訓練後 BA 可以合併回 W_0，因此推論不增加成本。QLoRA [31] 以 4 位元精度儲存凍結的基礎權重，並穿過它們反向傳播到 LoRA adapter，大幅減少記憶體，讓非常大的模型也能在單張 GPU 上微調。

程式：凍結基礎權重上的低秩更新

這個獨立範例在合成輸入上訓練一步 adapter。它展示參數化方式與初始行為，並沒有重現完整的預訓練模型微調流程。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
class LoRALinear(nn.Module):
    def __init__(self, din, dout, rank=4, alpha=4):
        super().__init__()
        self.base = nn.Linear(din, dout, bias=False)
        self.base.weight.requires_grad_(False)
        self.A = nn.Parameter(torch.randn(rank, din) * .02)
        self.B = nn.Parameter(torch.zeros(dout, rank))
        self.scale = alpha/rank
    def forward(self, x):
        return self.base(x) + self.scale * F.linear(F.linear(x, self.A), self.B)

layer = LoRALinear(32, 16)
x = torch.randn(8, 32)
assert torch.allclose(layer(x), layer.base(x))
opt = torch.optim.Adam([layer.A, layer.B], lr=.01)
loss = F.mse_loss(layer(x), torch.randn(8, 16))
opt.zero_grad(); loss.backward(); opt.step()
assert layer.base.weight.grad is None
print("trainable:", sum(p.numel() for p in layer.parameters() if p.requires_grad))
```

它印出 `trainable: 192`，也就是 $r(d_{\text{in}} + d_{\text{out}}) = 4 \times (32 + 16)$，完整矩陣則是 512。第一個 assertion 檢查初始化為零的 B 在一開始不改變模型；第二個檢查凍結的基礎權重沒有收到梯度。

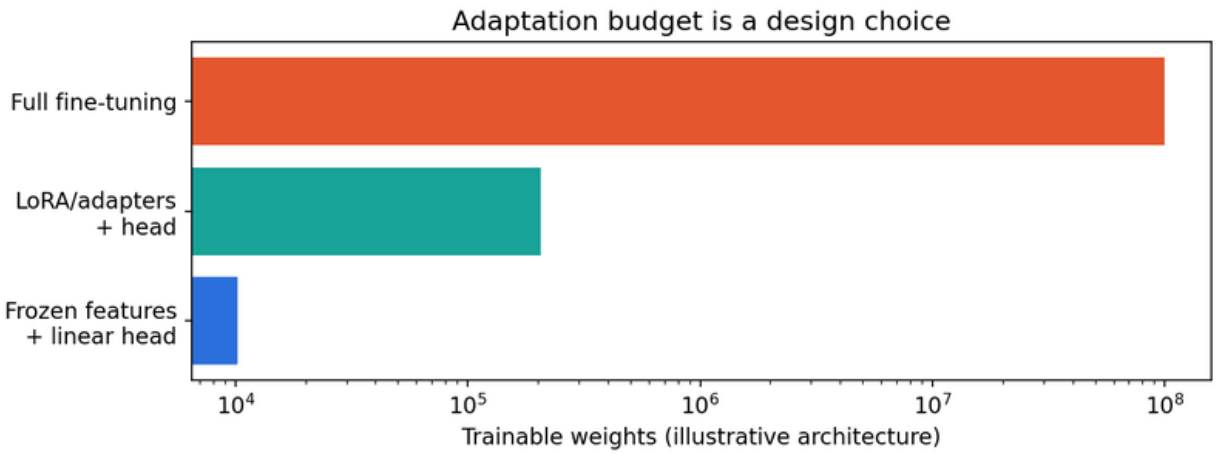


圖 *6* — 示意架構（特徵維度 *1024*、*10* 類、*12* 個矩陣上 *rank 8*、總共 *1* 億參數）的可訓練權重預算解析計算。活化記憶體、凍結權重儲存與最佳化器狀態必須另外計算。

LoRA 是一種參數化方式，不保證不會過擬合或遺忘。Adapter 仍需要領域驗證。凍結的 BatchNorm 統計量、影像正規化、輸出標籤對應，以及各 checkpoint 特有的前處理，也都需要刻意處理。

提示與上下文學習

最便宜的適配不改任何權重：為 CLIP 寫更好的文字提示、給 SAM 一個框，或像 Flamingo 那樣把幾個已解好的範例放進多模態模型的上下文 [27]。Zhang 等人的視覺語言綜述 [41] 比較了提示、特徵適配與微調在各種視覺任務上的表現。提示迭代很快，但很脆弱：措辭稍微改變就可能影響結果，因此要固定並報告提示流程。

影片、幾何與世界模型

白話版。除了辨認單一影格，模型還可以預測隨時間的變化、推論三維結構，或支援動作規劃。

V-JEPA 2 [11] 研究在特徵空間（而非像素空間）做預測的自監督影片模型，並以條件於動作的延伸版本進行規劃。VGGT [12] 以一次前饋，從一個或多個視角預測相機參數、深度、點雲圖（point maps）與軌跡。它們代表超越分類的兩條路：時間狀態建模與幾何推論。

看起來逼真的生成影片不一定遵守物理限制；場景重建即使不產生寫實像素也可能很有用。請依所宣稱的能力，量測多視角一致性、幾何、條件於動作的結果與時間行為。DLCV 13・世界模型會更詳細地討論部分可觀測性、學習動態與規劃。

風險與評估

白話版。廣泛預訓練可能把資料偏差、無根據預測及測試集重疊帶入評估；這些問題須與整體基準表現分開檢查。

資料來源、偏誤與隱私

網路規模的影像文字資料集大到無法人工檢查。LAION-5B [34] 公開了這樣的資料集（58.5 億組經 CLIP 過濾的配對），因而促成 OpenCLIP 這類開放重現 [20]；Birhane 等人 [35] 對其前身 LAION-400M 的稽核，則記錄了影像文字配對中露骨與刻板印象的內容。下游系統會繼承這些關聯。授權與同意也很重要：訓練資料、模型權重與生成輸出可能各有不同條款；個人資料（人臉、姓名、醫療影像）帶來的隱私義務，不會因為資料是公開的就消失。SAM 的 SA-1B 是一個在說明中強調具授權且尊重隱私影像的資料集例子 [5]；部署每個模型時都應記錄這類性質。

幻覺

多模態 LLM 可能提到影像中不存在的物體。POPE [37] 以輪詢方式衡量物體幻覺：針對特定物體是否存在提出是非題（包括常見物體與經常共同出現的干擾物體），使量測比評分自由生成的描述更不受模型生成風格影響。POPE 錯誤率低並不涵蓋計數、屬性或關係；這些要另外測試。

污染與「零樣本」

如果基準影像（或近似重複的影像）在預訓練資料中，「零樣本」分數有一部分其實是在測記憶。Mayilvahanan 等人 [36] 探討 CLIP 的泛化是否主要來自訓練與測試的高相似度：他們把 LAION 中與常見分布外基準太相似的影像剪除後重新訓練，部分基準的表現下降，但整體仍然很強。教訓是雙向的：相似度解釋了一部分報告的表現，但不是全部——所以要實際量測，不要預設任何一方。實務檢查包括：把測試集與可取得的訓練資料做去重複、在模型資料截止日期之後蒐集的資料上報告結果，並完整說明零樣本流程（類別名稱、模板、候選集合）。

基準

MMMU [38] 這類綜合性多模態基準——從考試、測驗與教科書蒐集 11.5K 個大學程度問題，橫跨六大學科，包含圖表、示意圖、地圖、表格等多種影像——測試知識與推理，但單一數字混合了感知、OCR、領域知識與答案格式。部署時，請建立一個小型、針對任務的評估集並納入已知的失敗案例，報告不同隨機種子或提示措辭下的變異，並與簡單基準比較（凍結特徵上的 linear probe 常常強得令人意外）。ML 11・評估與部署 會深入討論校準、資料切分與部署。

現代觀點

值得閱讀的綜述。

- **Bommasani 等人 [1]**——為基礎模型命名的報告。內容涵蓋能力（語言、視覺、機器人、推理）、應用（法律、醫療、教育）、技術（建模、訓練、適配、評估、資料）與社會影響。它的核心觀點——湧現、同質化與被繼承的失敗——至今仍是這個領域的框架。
- **Awais 等人 [39]**——電腦視覺綜述，依架構設計、訓練目標（對比式與生成式）、資料集、微調機制與提示方式整理基礎模型，並列出評估、實際應用、對抗穩健性與可解釋性等開放挑戰。

- **Zhang 等人 [41]**——TPAMI 上關於視覺任務的視覺語言模型綜述：網路架構、預訓練目標、下游任務、資料集、遷移學習與知識蒸餾，並對所回顧的方法做基準比較。
- **Yin 等人 [40] 與 Caffagni 等人 [2]**——多模態大型語言模型綜述：架構（編碼器、連接器、LLM）、訓練策略與資料（預訓練、指令微調、對齊）、評估與幻覺。
- **Han 等人 [13]**——把地圖擴展到多模態生成模型的綜述，依生成類別與技術組織。
- **Xu 等人 [42]**——參數高效微調的批判性回顧，把方法分成加法式、部分、重參數化、混合與統一五個家族，並實驗比較代表性方法。

2023–2026 年的現況。有四個趨勢特別明顯。(1) **更好的凍結特徵**：DINOv2 [15] 與 DINOv3 [16] 讓自監督編碼器強到許多密集任務直接從凍結的 patch 特徵開始。(2) **概念層級的提示**：分割從「點一個物體」（SAM [5]）走到影片記憶（SAM 2 [6]），再到「分割這個片語的每一個實例」（SAM 3 [7]）。(3) **長且交錯的多模態上下文**：Qwen3-VL [9] 等模型可同時讀取多張影像、影片與文字。(4) **超越辨識**：用於規劃的影片預測（V-JEPA 2 [11]）與前饋式幾何（VGGT [12]）把基礎模型當成世界與場景的估計器。

開放問題。

- 有依據的可靠性：減少幻覺，並讓模型在適當時候說出「看不到」[37]。
- 在污染與基準飽和下做誠實的評估 [36]。
- 資料：把整理資料當成一個縮放軸 [15], [20]，以及來源、授權與同意 [34], [35]。
- 有效率的適配與推論：高解析度影像與影片的 token 預算；能保留穩健性的 PEFT [32], [33]。
- 讓密集、三維與時間理解達到影像層級模型的語意水準。

面對任何新的「基礎」模型，請記錄它的輸入／輸出介面、訓練目標、資料來源、支援的適配方式、checkpoint 授權、前處理與任務層級的評估。這能避免把模型大小或開放式聊天介面當成具備通用能力的證據。

重點整理

- 基礎模型結合廣泛預訓練與重用，不限定單一架構，也不必生成文字。
- 縮放法則是依資料、目標與訓練配方擬合的經驗關係，外推需要驗證。
- 預訓練目標塑造能力，但不單獨決定下游表現。
- 影像—文字相似度支援檢索與文字定義類別，但相似度不是校準後的信心。
- 視覺提示與連接器控制資訊存取，應交代提示建構方式與影像解析度限制。
- 先建立低成本調適基準，再檢查較強方法的增益、成本、轉移敏感度、幻覺與資料污染。

練習

1. **候選集合的影響**。在合成 CLIP 範例中加入第四個描述。影像不變時，softmax 的「信心」會不會改變？

   提示

   會：softmax 的分母改變了。影像與某個固定描述的相似度不變，但正規化後的機率取決於候選集合。試著加入一個與模糊影像嵌入很接近的文字向量。

2. **計算最佳配置**。利用參數化定律 $L(N, D) = E + A/N^{\alpha} + B/D^{\beta}$ 與 $C = 6ND$，證明最佳模型大小的成長為 $N^{\star} \propto C^{\beta/(\alpha+\beta)}$。$\alpha = \beta$ 代表什麼？

   提示

   代入 $D = C/(6N)$，對 N 微分並令其為零：$\alpha A N^{-\alpha-1} = \beta B (6N/C)^{\beta}/N$。解得 $N^{\alpha+\beta} \propto C^{\beta}$。當 $\alpha = \beta$ 時，$N^{\star} \propto C^{1/2}$，因此 $D^{\star} \propto C^{1/2}$：參數與 token 等比例擴大，與 Hoffmann 等人的結論一致。

3. **讓遷移失效**。在 linear probe 實驗中，讓下游類別取決於干擾因子而非語意因子。隨著標籤增加，凍結特徵上的 probe 會怎樣？這對選擇基礎模型有什麼啟示？

   提示

   改為對 u 抽類別中心，而不是 z。凍結的 8 維特徵被訓練成丟掉干擾方向，所以不論加多少標籤都接近隨機猜測，而原始輸入上的 probe 會進步。骨幹只有在預訓練保留了你的任務所需資訊時才有用——投入適配之前，先用 linear probe 檢查。

4. **LoRA 預算**。一個 ViT-B 區塊有四個 768×768 的注意力投影，以及兩個大小分別為 3072×768 與 768×3072 的 MLP 矩陣。計算這六個矩陣完整微調與全部使用 rank-8 LoRA 時的可訓練權重數。

   提示

   完整：$4 \cdot 768^2 + 2 \cdot 3072 \cdot 768 = 2{,}359{,}296 + 4{,}718{,}592 = 7{,}077{,}888$。LoRA：$8 \cdot [4 \cdot (768 + 768) + 2 \cdot (3072 + 768)] = 8 \cdot (6144 + 7680) = 110{,}592$，約為完整數量的 1.6%。

5. **提示的公平性**。設計一個能把遮罩品質與提示品質分開的 SAM 評估。

   提示

   固定提示流程並計算人工互動次數。把自動提議與使用理想真值提示得到的遮罩分開評估，兩者都要報告。對 SAM 3，在同一批影像上比較名詞片語提示與範例提示。

6. **污染稽核**。有一個新的影像文字模型宣稱在某個公開基準上有很高的零樣本準確率。在相信這個數字之前，列出你會做的三項檢查。

   提示

   (a) 以嵌入相似度等方式，在可取得的訓練資料中搜尋測試影像的近似重複；(b) 在資料截止日期之後新蒐集的資料上評估；(c) 檢查對提示模板與候選類別清單的敏感度；也可以與 linear probe 及簡單基準比較。

參考文獻

1. R. Bommasani et al., “On the Opportunities and Risks of Foundation Models,” Stanford CRFM report, arXiv:2108.07258, 2021; revised 2022. arXiv
2. D. Caffagni et al., “The Revolution of Multimodal Large Language Models: A Survey,” *Findings of ACL*, 2024. arXiv
3. A. Radford et al., “Learning Transferable Visual Models From Natural Language Supervision,” *ICML*, 2021. arXiv
4. X. Zhai, B. Mustafa, A. Kolesnikov and L. Beyer, “Sigmoid Loss for Language Image Pre-Training,” *ICCV*, 2023. arXiv
5. A. Kirillov et al., “Segment Anything,” *ICCV*, 2023. arXiv
6. N. Ravi et al., “SAM 2: Segment Anything in Images and Videos,” *ICLR*, 2025. arXiv
7. N. Carion et al., “SAM 3: Segment Anything with Concepts,” arXiv:2511.16719, 2025; revised March 2026. arXiv
8. H. Liu, C. Li, Q. Wu and Y. J. Lee, “Visual Instruction Tuning,” *NeurIPS*, 2023. arXiv
9. S. Bai et al., “Qwen3-VL Technical Report,” arXiv:2511.21631, 2025. arXiv
10. E. J. Hu et al., “LoRA: Low-Rank Adaptation of Large Language Models,” *ICLR*, 2022. arXiv
11. M. Assran et al., “V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning,” arXiv:2506.09985, 2025. arXiv
12. J. Wang et al., “VGGT: Visual Geometry Grounded Transformer,” *CVPR*, 2025. arXiv
13. L. Han et al., “A Survey of Generative Categories and Techniques in Multimodal Generative Models,” arXiv:2506.10016, 2025. arXiv
14. C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024. Book
15. M. Oquab et al., “DINOv2: Learning Robust Visual Features without Supervision,” *Transactions on Machine Learning Research*, 2024. arXiv
16. O. Siméoni et al., “DINOv3,” arXiv:2508.10104, 2025. arXiv
17. J. Kaplan et al., “Scaling Laws for Neural Language Models,” arXiv:2001.08361, 2020. arXiv
18. J. Hoffmann et al., “Training Compute-Optimal Large Language Models,” arXiv:2203.15556, 2022; published as “An empirical analysis of compute-optimal large language model training,” *NeurIPS*, 2022. arXiv
19. X. Zhai, A. Kolesnikov, N. Houlsby and L. Beyer, “Scaling Vision Transformers,” *CVPR*, 2022. arXiv
20. M. Cherti et al., “Reproducible Scaling Laws for Contrastive Language-Image Learning,” *CVPR*, 2023. arXiv
21. A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *ICLR*, 2021. arXiv
22. K. He, X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, “Masked Autoencoders Are Scalable Vision Learners,” *CVPR*, 2022. arXiv
23. H. Bao, L. Dong, S. Piao and F. Wei, “BEiT: BERT Pre-Training of Image Transformers,” *ICLR*, 2022. arXiv
24. M. Caron et al., “Emerging Properties in Self-Supervised Vision Transformers,” *ICCV*, 2021. arXiv
25. C. Jia et al., “Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision,” *ICML*, 2021. arXiv

26. J. Li, D. Li, S. Savarese and S. Hoi, “BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models,” *ICML*, 2023. PMLR
27. J.-B. Alayrac et al., “Flamingo: a Visual Language Model for Few-Shot Learning,” *NeurIPS*, 2022. arXiv
28. H. Liu, C. Li, Y. Li and Y. J. Lee, “Improved Baselines with Visual Instruction Tuning,” *CVPR*, 2024. arXiv
29. N. Houlsby et al., “Parameter-Efficient Transfer Learning for NLP,” *ICML*, 2019. PMLR
30. M. Jia et al., “Visual Prompt Tuning,” *ECCV*, 2022. arXiv
31. T. Dettmers, A. Pagnoni, A. Holtzman and L. Zettlemoyer, “QLoRA: Efficient Finetuning of Quantized LLMs,” *NeurIPS*, 2023. arXiv
32. A. Kumar, A. Raghunathan, R. Jones, T. Ma and P. Liang, “Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution,” *ICLR*, 2022. arXiv
33. M. Wortsman et al., “Robust Fine-Tuning of Zero-Shot Models,” *CVPR*, 2022. arXiv
34. C. Schuhmann et al., “LAION-5B: An Open Large-Scale Dataset for Training Next Generation Image-Text Models,” *NeurIPS Datasets and Benchmarks Track*, 2022. arXiv
35. A. Birhane, V. U. Prabhu and E. Kahembwe, “Multimodal Datasets: Misogyny, Pornography, and Malignant Stereotypes,” arXiv:2110.01963, 2021. arXiv
36. P. Mayilvahanan, T. Wiedemer, E. Rusak, M. Bethge and W. Brendel, “Does CLIP’s Generalization Performance Mainly Stem from High Train-Test Similarity?,” *ICLR*, 2024. arXiv
37. Y. Li et al., “Evaluating Object Hallucination in Large Vision-Language Models,” *EMNLP*, 2023. arXiv
38. X. Yue et al., “MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI,” *CVPR*, 2024. arXiv
39. M. Awais et al., “Foundational Models Defining a New Era in Vision: A Survey and Outlook,” arXiv:2307.13721, 2023. arXiv
40. S. Yin et al., “A Survey on Multimodal Large Language Models,” *National Science Review*, 2024. arXiv
41. J. Zhang, J. Huang, S. Jin and S. Lu, “Vision-Language Models for Vision Tasks: A Survey,” *IEEE TPAMI*, vol. 46, no. 8, 2024. arXiv
42. L. Xu et al., “Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment,” arXiv:2312.12148, 2023. arXiv
43. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” *CVPR*, 2009, pp. 248–255. doi
44. J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *NAACL-HLT*, 2019, pp. 4171–4186. doi
45. OpenAI, “GPT-4 Technical Report,” arXiv:2303.08774, 2023. arXiv

ML 11・評估、效率與部署

不洩漏地切分資料、做交叉驗證、為分數加上信賴區間、檢查校準與分布偏移下的穩健性，誠實量測 FLOPs、記憶體與延遲，再把模型匯出、壓縮、監控並做到可重現。

網站版本: https://blog.shyandram.dev/zh-tw/learn/ml/evaluation-deployment/

你將學到

- 設計能支持部署主張的基準與資料切分。
- 依取樣結構選擇交叉驗證與不確定性估計。
- 區分準確率、校準與轉移下的穩健性。
- 在目標系統量測運算量、記憶體與延遲。
- 驗證匯出模型、監控部署，並記錄可重現的實驗。

先看全貌

訓練模型只是專案的中段，不是終點。在任何人依賴它之前，你必須回答四個問題：它在真正會遇到的案例上有效嗎？這個數字有多可靠？它知道自己什麼時候可能出錯嗎？它符合要執行的裝置、延遲與預算嗎？上線之後還有第五個問題：它現在還有效嗎？

> **用最簡單的話說**
>
> 一個背下去年考卷的學生，考去年的考卷當然滿分，但這什麼也說明不了。公平的考試要用新題目（乾淨的資料切分）。一次考試可能運氣好，所以你想知道換一天考分數會晃動多少（信賴區間）。好學生遇到可能答錯的題目也會說「我不太確定」（校準）。如果正式考場很吵，你也要在吵雜環境下測試（穩健性）。如果是限時考試，答對但太慢也不行（延遲）。學生開始工作後，還是要有人不時抽查他的成果（監控）。

這一章是 ML 系列的最後一章。它整理 Goodfellow 等人 [1, ch. 11] 的實務方法論與 James 等人 [2] 的統計學習觀點，再補上把訓練好的網路變成可靠服務所需的工程步驟。視覺任務專屬的指標（top-1、mAP、PSNR、FID 等）在 DLCV 02 與 DLCV 03 中說明；本章聚焦所有任務共通的部分。

先建立有意義的基準模型

白話版。在投入更多算力之前，先看看簡單方法在完全相同的切分上表現如何。基準模型告訴你複雜模型到底多換到了什麼。

影像任務常見的好基準包括：在原始像素或手工特徵上做邏輯迴歸；最近鄰或 SVM 分類器；以及在預訓練模型凍結特徵上接一個小型線性分類頭（見 ML 10）。邏輯迴歸學習線性邊界；支援向

量機（support vector machine, SVM）最大化帶正則化的間隔；最近鄰直接比較樣本；樹集成則切分特徵空間。傳統圖形識別章節 IP 13 有更完整的推導。

作為參考，二元線性 SVM 最小化

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\max(0,\,1 - y_i(w^\top x_i + b)), \qquad y_i \in \{-1, +1\},$$

其中 w 與 b 是權重與偏差，x_i 是第 i 個特徵向量，$C > 0$ 在間隔寬度與訓練錯誤之間取捨。$\max(0,\, 1 - y_i f(x_i))$ 稱為**鉸鏈損失（hinge loss）**：位於正確一側且間隔至少為 1 的點損失為零，其餘則線性增加。

程式碼：把前處理放進 pipeline 的基準模型

這個例子只使用內建資料。模型選擇用驗證集；選出的模型只在測試集上評估一次。標準化與 PCA 都在各自的訓練 pipeline 內擬合，絕不在整份資料上擬合。

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, balanced_accuracy_score

X, y = load_digits(return_X_y=True)
Xa, Xt, ya, yt = train_test_split(X, y, test_size=.2, stratify=y, random_state=0)
Xtr, Xv, ytr, yv = train_test_split(Xa, ya, test_size=.25, stratify=ya, random_state=1)
models = {      # listed from simplest to most complex
    "linear": make_pipeline(StandardScaler(),
                            LogisticRegression(max_iter=2000)),
    "PCA + RBF SVM": make_pipeline(StandardScaler(), PCA(n_components=30),
                                   SVC(C=10, gamma="scale")),
}
scores = {}
for name, model in models.items():
    model.fit(Xtr, ytr)
    scores[name] = accuracy_score(yv, model.predict(Xv))
best = max(scores, key=scores.get)   # tie -> first listed, i.e. the simpler model
pred = models[best].predict(Xt)
print("validation:", scores)
print("selected:", best)
print("test accuracy:", accuracy_score(yt, pred))
print("test balanced accuracy:", balanced_accuracy_score(yt, pred))
```

在這個切分上，兩個候選模型的驗證準確率剛好都是 0.9639（360 張中答對 347 張）。「平手時選較簡單的模型」這條規則是事先定好的，因此選出線性模型，它在測試集上的準確率為 0.9639、平衡準確率為 0.9636。這個小實驗藏著兩個教訓。第一，平手規則是評估協定的一部分，應該在

看結果之前寫下來。第二，驗證集只有 360 張時，一張影像就值 0.28 個百分點，所以小實驗中許多看似「勝出」的差距其實只是雜訊；後面的信賴區間一節會把這點說得更精確。

資料切分與資料洩漏

白話版。測試資料的新穎方式應符合部署情境；若主張適用於新群組或未來資料，共用病人、場景或時段可能高估結果。

依主張決定切分單位

同一位病人、同一段影片、同一批產品或同一台相機的影像彼此相關。把相鄰影格隨機切分，會讓幾乎相同的畫面同時出現在訓練與測試中。先問部署時「新」代表什麼，再依該單位切分：模型會遇到新病人就依病人切，會遇到新影片就依影片切，要裝到新醫院就依院區切，要預測未來就依時間切。



圖 *1* — 依泛化主張選擇切分單位。以群組分開可以避免幾乎重複的情境跨越切分邊界。

分層抽樣（stratification）讓各切分的類別比例相近，但它不能取代群組切分。切分前先去除重複，並把完全或近似重複的樣本放在同一側。若部署母體會隨時間漂移，依時間切分（用過去訓練、用最近一段時間測試）才是誠實的做法。

資料洩漏的各種面貌

資料洩漏是指任何讓「預測當下不該取得的資訊」流進模型或模型選擇過程的途徑。Kapoor 與 Narayanan 發現資料洩漏錯誤影響了 17 個使用 ML 的科學領域、共 329 篇論文，並提出洩漏類型的分類，其中包括缺少訓練／測試分離、在整份資料上做前處理、重複樣本、時間洩漏與不正當特徵 [4]。scikit-learn 文件也列出最常見的程式錯誤 [3]。實用檢查表：

洩漏	例子	修正
在全部資料上前處理	在訓練＋測試上擬合 `StandardScaler`、PCA 或詞彙表	把前處理放進只在訓練折上擬合的 `Pipeline`
相關的單位	同一段影片的影格、同一次掃描的切片、同一張照片的裁切	依群組切分
重複樣本	同一張影像被爬取兩次	切分前先用雜湊或嵌入去重
時間	用下週的資料預測本週	依時間順序切分
目標代理變數	在結果**之後**才記錄的特徵（例如「是否已接受治療」）	稽核每個特徵的記錄時間
重複使用測試集	一邊看測試分數一邊調超參數	決策用驗證集，測試只做一次

最後一列最隱微。每次看了測試分數再修改，測試集就變得更像驗證集一點。公開基準在整個社群的規模上都面臨這個問題。

交叉驗證

白話版。不只做一次訓練／驗證切分，而是輪流：把資料分成 K 份，訓練 K 次，每次留出不同的一份來評估，再取平均。每個樣本都恰好被評估一次，也能看到分數的變動幅度。

K 折交叉驗證與其變形

把索引分成 K 個互斥的折 $F_1, \dots, F_K$。對每個 k，用除了 F_k 以外的所有折訓練出 $\hat{f}^{(-k)}$，並在 F_k 上計算誤差。交叉驗證估計值為

$$\widehat{\mathrm{Err}}_{\mathrm{CV}} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{|F_k|}\sum_{i\in F_k}\ell\big(y_i, \hat{f}^{(-k)}(x_i)\big),$$

其中 ℓ 是單一樣本的損失（例如 0–1 錯誤），$|F_k|$ 是該折的大小。常見選擇為 $K=5$ 或 $K=10$ [2]。各種變形只差在如何組成各折：

- **分層 K 折**讓各折的類別比例相同；類別不平衡時很有用。
- **群組 K 折**把每個群組（病人、影片、院區）只分到一個折，因此沒有群組同時出現在訓練與留出資料中。
- **時間序列切分**永遠用較早的資料訓練、用較晚的資料驗證。
- **留一群組**是每折只放一個群組的群組 K 折；群組很少時有用。

各折分數並不獨立（訓練集互相重疊），所以各折之間的標準差會低估真正的不確定性。把它當成穩定性的粗略指標，而不是正式的信賴區間。

程式碼：隨機 K 折對群組資料做了什麼

我們模擬 40 位病人、每人 25 張影像。同一位病人的影像共享一個「指紋」（病人特有的外觀），而標籤以病人為單位固定。真正的類別訊號很弱。

```
import numpy as np
from sklearn.model_selection import KFold, GroupKFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier

rng = np.random.default_rng(0)
n_groups, per_group = 40, 25                  # e.g. 40 patients, 25 images each
groups = np.repeat(np.arange(n_groups), per_group)
y = np.repeat(rng.integers(0, 2, n_groups), per_group)   # label fixed per patient
identity = rng.normal(size=(n_groups, 20))[groups]       # patient "fingerprint"
signal = 0.3 * y[:, None] * np.ones((1, 20))             # weak real class signal
X = identity + signal + 0.3 * rng.normal(size=(len(y), 20))

model = RandomForestClassifier(n_estimators=200, random_state=0)
random_cv = cross_val_score(model, X, y, cv=KFold(5, shuffle=True, random_state=0))
group_cv = cross_val_score(model, X, y, cv=GroupKFold(5), groups=groups)
print(f"random K-fold accuracy: {random_cv.mean():.3f} +/- {random_cv.std():.3f}")
print(f"group  K-fold accuracy: {group_cv.mean():.3f} +/- {group_cv.std():.3f}")
```

輸出：

```
random K-fold accuracy: 0.999 +/- 0.002
group  K-fold accuracy: 0.657 +/- 0.158
```

隨機切分回報近乎完美的準確率，因為隨機森林認出了每位病人的指紋，直接抄該病人的標籤。群組 *K* 折符合「對新病人有效」這個主張，回報的是 0.657，而且各折差異很大。兩個數字在算術上都沒錯，但只有一個回答了真正的問題。

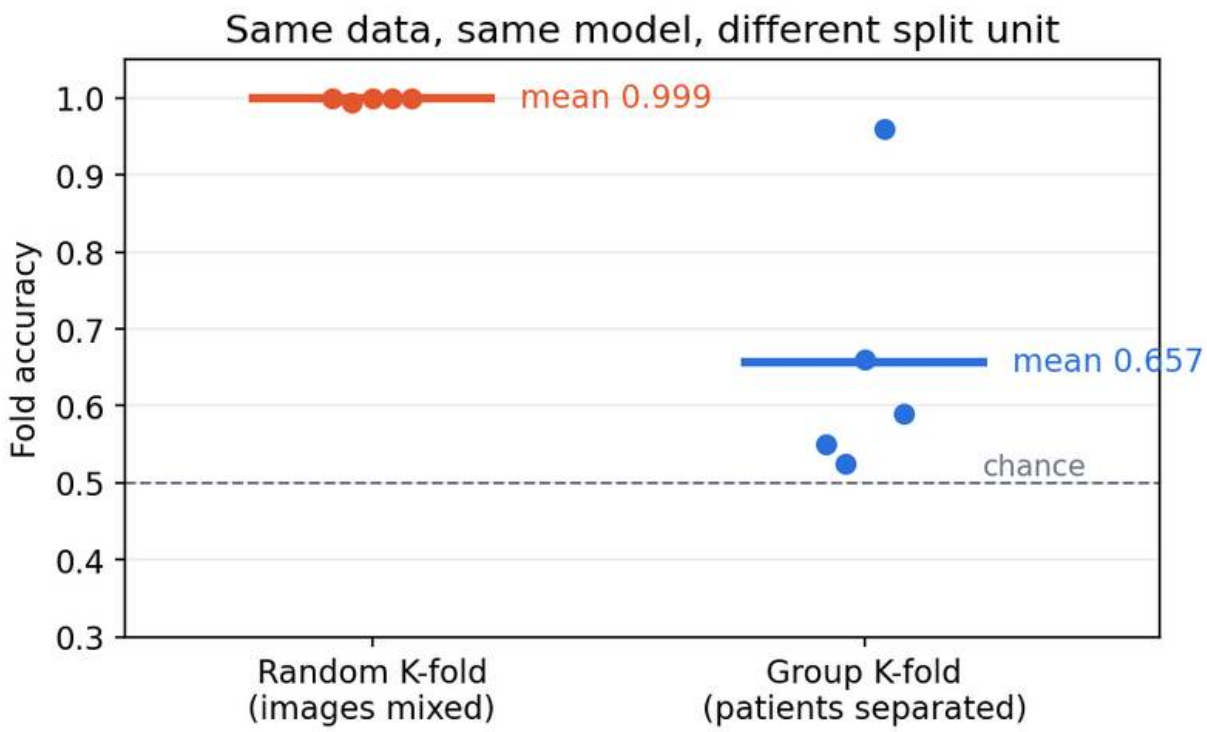


圖 2 — 上方程式碼的輸出，每個點代表一折。造成 *34* 個百分點差距的是切分單位，不是模型。

巢狀交叉驗證

如果同時用交叉驗證來**選**超參數又**回報**表現，回報的數字會偏樂觀：你挑的是剛好在那些折上看起來最好的設定。Varma 與 Simon 用兩類在統計上完全相同的「虛無」資料示範了這點：用交叉驗證調整 SVM 後，在 38% 的模擬資料集上 CV 誤差估計低於 30%，而調好的分類器在獨立測試資料上並不比亂猜好 [5]；Cawley 與 Talbot 則顯示，對模型選擇準則的過擬合可能和不同學習演算法之間的差距一樣大 [6]。

巢狀交叉驗證用兩層迴圈解決這個問題。外層留出一折做評估；在每個外層訓練集內，由內層交叉驗證選擇超參數。外層折只用來為內層選出的設定打分數。在 scikit-learn 中，這只是把 `GridSearchCV` 放進 `cross_val_score`。得到的結果估計的是整個**程序**（包含調參）的表現，也就是你重新訓練時實際會執行的東西。

對深度網路而言，完整的巢狀交叉驗證通常太昂貴。常見的折衷是固定一組訓練／驗證／測試切分，為每個被比較的方法訂明調參預算，並只打開測試集一次。

指標回顧

白話版。先決定一個錯誤的代價，再選數字。單一純量很少能說完整個故事。

任務	有用的量測	常見陷阱
分類	各類別精確率／召回率、macro-F1、平衡準確率、AUROC/AUPRC	準確率隱藏罕見失敗；PR 曲線受盛行率影響
偵測／分割	AP、IoU/Dice、邊界誤差	比較不同 IoU 門檻或評估協定的結果
檢索	Recall@k、在明確相關集合下的 mAP	圖庫中的重複樣本灌高匹配率
影像復原	PSNR/SSIM、感知指標、與觀測的一致性	看似合理的紋理不一定是真實細節
生成	FID/KID、涵蓋率、條件對齊、人工評估	特徵層面的真實感不代表忠於提示
影片／幾何	時間一致性、任務準確率、幾何誤差	單張影格好看可能掩蓋結構不一致

分類指標（混淆矩陣、top-k、ROC/PR 曲線）的定義與陷阱在 DLCV 02 中詳細說明，偵測指標（IoU、AP、COCO mAP）則在 DLCV 03。對生成模型而言，FID 比較特徵分布的高斯擬合（平均與共變異數）；KID 則使用無偏的核最大平均差異估計，因此在樣本數少時表現較穩定 [7]。兩者都不是人類偏好的通用量度。務必寫明特徵擷取器、參考集合、樣本數、解析度與取樣設定。

正例只有 1% 時，永遠預測負例的分類器有 99% 準確率而召回率為零，因此只看準確率會掩蓋失敗。精確率也受盛行率影響。若只有類別比例改變，而各類別內的輸入分布與決策門檻保持固定，召回率與偽陽性率才會不變，精確率則會改變 [17]。這是標籤偏移假設；換醫院、相機或母體時，各類別內的外觀也可能改變，因此三種比率都應重新測量，不能直接假設穩定。

統計顯著性與信賴區間

白話版。測試分數是從有限樣本算出來的，換一批樣本就會有些不同。要回報一個範圍而不只是一個點，並在**相同**樣本上比較兩個模型。

準確率的標準誤

若 n 個測試樣本彼此獨立，模型真正的準確率為 p，則答對的個數服從二項分布，觀測準確率 $\hat{p}$ 的標準誤為

$$\mathrm{SE}(\hat{p}) = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}, \qquad 95\%\ \text{interval} \approx \hat{p} \pm 1.96\,\mathrm{SE}.$$

這裡 $\hat{p}$ 是答對比例，1.96 是標準常態分布的第 97.5 百分位數。當 $\hat{p} = 0.972$、$n = 540$ 時，$\mathrm{SE} \approx 0.0071$，區間大約是 ± 1.4 個百分點。當 $\hat{p}$ 非常接近 0 或 1、或 n 很小時，這個常態近似很差，應改用 Wilson 或精確二項區間。若測試樣本有群組結構（每位病人多張影像），有效樣本數比較接近群組數，此時必須以群組為單位重抽樣來計算區間。

Bootstrap

Bootstrap 以「從觀測資料中放回抽樣」來估計任意統計量的抽樣分布 [9]。抽出 B 組大小為 n 的重抽樣，在每組上重新計算統計量，並取第 2.5 與第 97.5 百分位數作為近似的 95% 區間（**百分位數區間**）。它適用於沒有簡單公式的指標，例如 macro-F1、AUROC 或 mAP。

比較兩個模型時，要用**成對** bootstrap：每次只抽一次樣本索引，並在同一組重抽樣上評估兩個模型。因為兩個模型通常在許多相同的困難樣本上失敗，配對能消除這部分共同變異，使差值的區間比兩個獨立區間緊得多。

McNemar 檢定

兩個分類器在同一個測試集上評估時，只有它們意見不同的樣本才帶有「誰比較好」的資訊。令 n_{01} 為模型 A 答對而 B 答錯的樣本數，n_{10} 為相反情況。在「兩個模型錯誤率相同」的虛無假設下，每個不一致樣本落在兩邊的機率相等，所以 $n_{10} \sim \text{Binomial}(n_{01}+n_{10}, \frac{1}{2})$。精確 McNemar 檢定就由此計算雙尾二項 p 值。Dietterich 比較了五種比較學習演算法的檢定，發現對只能執行一次的演算法（昂貴模型的典型情況），McNemar 檢定是唯一第一型錯誤可以接受的方法；若能負擔執行十次，他建議使用 5×2 交叉驗證 t 檢定 [8]。

程式碼：成對 bootstrap 與 McNemar

```
import numpy as np
from scipy.stats import binomtest
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
a = make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000)).fit(Xtr, ytr)
b = make_pipeline(StandardScaler(), SVC(C=10, gamma="scale")).fit(Xtr, ytr)
ca, cb = a.predict(Xte) == yte, b.predict(Xte) == yte    # per-example correctness

# 1) Percentile bootstrap: resample test examples with replacement.
rng = np.random.default_rng(0)
idx = rng.integers(0, len(yte), size=(10_000, len(yte)))
diff = cb[idx].mean(axis=1) - ca[idx].mean(axis=1)       # paired: same resample
lo, hi = np.percentile(diff, [2.5, 97.5])
print(f"accuracy A={ca.mean():.4f}  B={cb.mean():.4f}  n={len(yte)}")
print(f"B - A = {cb.mean() - ca.mean():+.4f}, 95% bootstrap CI [{lo:+.4f}, {hi:+.4f}]")

# 2) Exact McNemar test: only the discordant examples carry information.
n01 = int(np.sum(ca & ~cb))      # A right, B wrong
n10 = int(np.sum(~ca & cb))      # A wrong, B right
p = binomtest(n10, n01 + n10, 0.5).pvalue
print(f"discordant pairs: A-only={n01}, B-only={n10}, exact McNemar p={p:.4f}")
```

輸出：

```
accuracy A=0.9722  B=0.9852  n=540
B - A = +0.0130, 95% bootstrap CI [-0.0019, +0.0278]
```

```
discordant pairs: A-only=5, B-only=12, exact McNemar p=0.1435
```

SVM 高出 1.3 個百分點，但區間包含零，McNemar 的 p 值是 0.14。540 張中只有 17 張能區分兩個模型，而 12 比 5 在公平硬幣下並不罕見。以這個測試集而言，誠實的結論是：「B 大概至少一樣好；說它更好的證據很薄弱。」

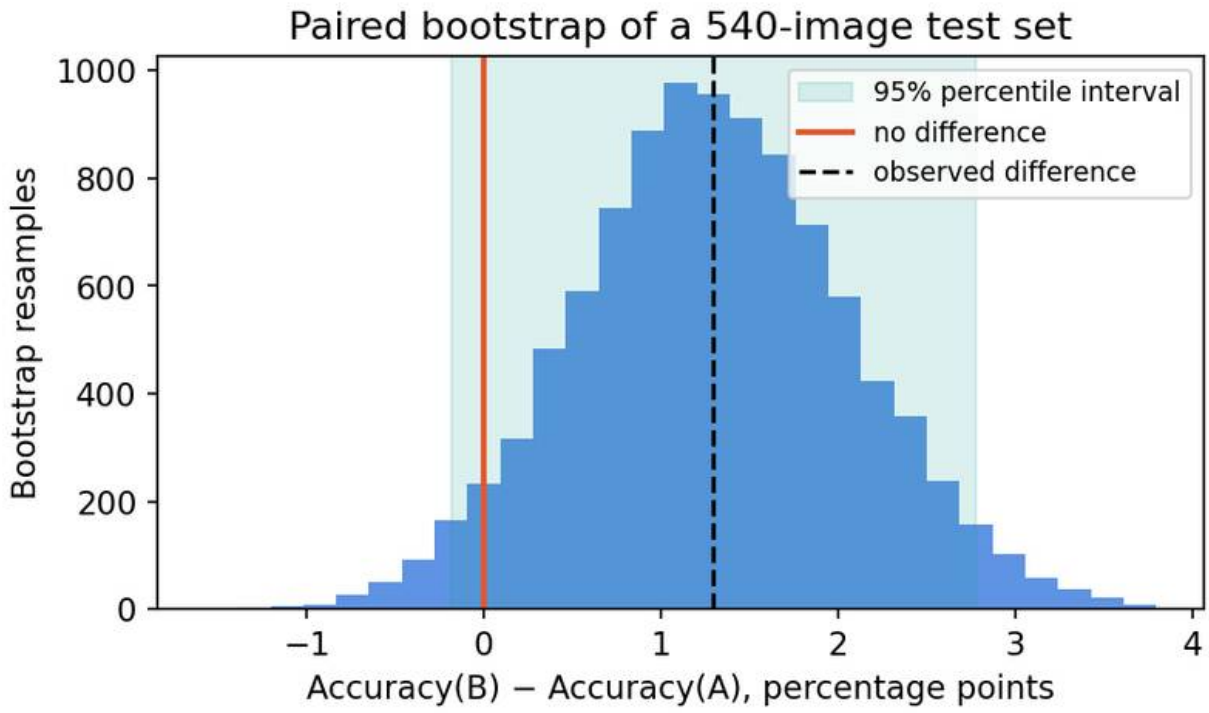


圖 3 — 上方程式碼中 $Accuracy(B) - Accuracy(A)$ 的成對 *bootstrap* 分布。95% 區間跨過零。

其他變異來源

測試樣本不是唯一的隨機來源。對神經網路而言，隨機種子會改變初始化、資料順序與資料擴增；資料切分本身也是隨機的；某些 GPU 核心是非確定性的。Bouthillier 等人為整個基準評測過程建模，發現資料取樣、參數初始化與超參數選擇造成的變異會明顯影響結論，並建議在重複實驗中盡可能隨機化這些來源 [10]。實務原則：

- 每個設定用多個種子執行（常見 3–5 個），回報平均與分散程度。
- 用成對檢定，在相同樣本與切分上比較方法。
- 若比較很多模型或很多資料集，總會有一些純屬偶然地「勝出」；要做多重比較校正，或在新資料上確認。
- 回報效果大小與區間。統計顯著的差異在實務上可能無關緊要；實務上很大的差異，在小資料集上也可能尚未被統計證實。

校準：知道自己可能出錯

白話版。模型的信心應該和它答對的頻率相符。在所有以 80% 信心做出的預測中，大約 80% 應該是對的。準確率與校準是兩種不同的性質。

量測機率品質

令 p_{ik} 為樣本 i 屬於類別 k 的預測機率，$y_{ik} \in \{0,1\}$ 為 one-hot 標籤。兩個**合適評分規則**（proper scoring rule，在期望上由真實機率達到最小）為：

$$\text{NLL} = -\frac{1}{n}\sum_{i=1}^{n} \log p_{i,y_i}, \qquad \text{Brier} = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} (p_{ik} - y_{ik})^2,$$

其中 y_i 是樣本 i 的真實類別，K 是類別數。NLL 是負對數概似（negative log-likelihood）。有些函式庫會把 Brier 分數再除以 K；請說明你使用哪種慣例。

期望校準誤差（expected calibration error, ECE）依最高類別信心把預測分到 M 個區間 $B_1, \dots, B_M$，並在每個區間比較平均信心與準確率：

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\text{acc}(B_m) - \text{conf}(B_m)\right|,$$

其中 $\text{acc}(B_m)$ 是第 m 個區間的答對比例，$\text{conf}(B_m)$ 是該區間的平均信心。**可靠度圖**（reliability diagram）逐區間畫出 acc 對 conf；完美校準的模型落在對角線上。ECE 不是合適評分規則，且取決於區間數與分箱方式，因此應與 NLL 或 Brier 分數及可靠度圖一起回報。

溫度縮放

Guo 等人觀察到，比起數十年前的網路大得多的現代深度網路常常過度自信，而一個純量修正就出奇地有效 [11]。**溫度縮放**（temperature scaling）在 softmax 之前把 logits ℓ 除以 $T > 0$：

$$\hat{p}_k = \frac{\exp(\ell_k / T)}{\sum_j \exp(\ell_j / T)}, \qquad T^\star = \arg\min_{T>0} \ \text{NLL}_{\text{cal}}(T).$$

$T > 1$ 會軟化過度自信的預測；$T < 1$ 則讓信心不足的預測更尖銳。因為把所有 logits 除以同一個正數不會改變誰最大，預測類別（也就是準確率）不變。T 必須在留出的校準集上擬合，絕不能在測試集上擬合。後來 Minderer 等人發現，最新的影像分類器，尤其是 Vision Transformer 這類不使用卷積的模型，屬於校準最好的一群，而且「校準隨分布偏移而衰退」等舊趨勢在它們身上較不明顯 [12]。

程式碼：溫度縮放，以及偏移對它的影響

```
import numpy as np
from scipy.optimize import minimize_scalar
from scipy.special import log_softmax, softmax
```

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

X, y = load_digits(return_X_y=True)
X = X / 16.0
Xa, Xte, ya, yte = train_test_split(X, y, test_size=0.4, stratify=y, random_state=0)
Xtr, Xcal, ytr, ycal = train_test_split(Xa, ya, test_size=0.3, stratify=ya, random_state=0)
net = MLPClassifier(hidden_layer_sizes=(128,), alpha=1e-6, max_iter=300,
                    random_state=0).fit(Xtr, ytr)

def logits(X):                        # log-probabilities = logits up to a constant
    return np.log(np.clip(net.predict_proba(X), 1e-12, 1.0))

def ece(prob, y, bins=10):
    conf, pred = prob.max(1), prob.argmax(1)
    edges = np.linspace(0, 1, bins + 1)
    total = 0.0
    for lo, hi in zip(edges[:-1], edges[1:]):
        m = (conf > lo) & (conf <= hi)
        if m.any():
            total += m.mean() * abs((pred[m] == y[m]).mean() - conf[m].mean())
    return total

def nll(T, z, y):
    return -log_softmax(z / T, axis=1)[np.arange(len(y)), y].mean()

T = minimize_scalar(nll, bounds=(0.05, 20), method="bounded",
                    args=(logits(Xcal), ycal)).x
print(f"fitted temperature T = {T:.2f}")
noise = np.random.default_rng(0).normal(size=Xte.shape)
for sigma in [0.0, 0.3, 0.6]:         # Gaussian pixel noise = a simple covariate shift
    Xs = np.clip(Xte + sigma * noise, 0, 1)
    z = logits(Xs)
    p_raw, p_ts = softmax(z, axis=1), softmax(z / T, axis=1)
    print(f"noise {sigma:.1f}: acc={(p_raw.argmax(1) == yte).mean():.3f}  "
          f"ECE raw={ece(p_raw, yte):.3f}  ECE scaled={ece(p_ts, yte):.3f}  "
          f"NLL raw={nll(1, z, yte):.3f}  NLL scaled={nll(T, z, yte):.3f}")
```

輸出（scikit-learn 可能印出收斂警告，在這裡無害）：

```
fitted temperature T = 1.19
noise 0.0: acc=0.972  ECE raw=0.007  ECE scaled=0.015  NLL raw=0.108  NLL scaled=0.110
noise 0.3: acc=0.794  ECE raw=0.088  ECE scaled=0.063  NLL raw=0.693  NLL scaled=0.639
noise 0.6: acc=0.435  ECE raw=0.380  ECE scaled=0.347  NLL raw=2.998  NLL scaled=2.598
```

有三點觀察。在乾淨資料上，這個小網路本來就校準得很好，而用 324 張校準影像擬合 T 反而讓 ECE 略微**變差**：在小資料上擬合的修正可能引入雜訊。加入像素雜訊後，模型嚴重過度自信：在 $\sigma = 0.6$ 時它只有 43.5% 答對，信心卻高得多。在乾淨資料上擬合的溫度縮放，在偏移下只幫上一點忙。這與 Ovadia 等人的大型研究一致：他們發現許多不確定性方法的校準會隨偏移增加而惡化 [16]。

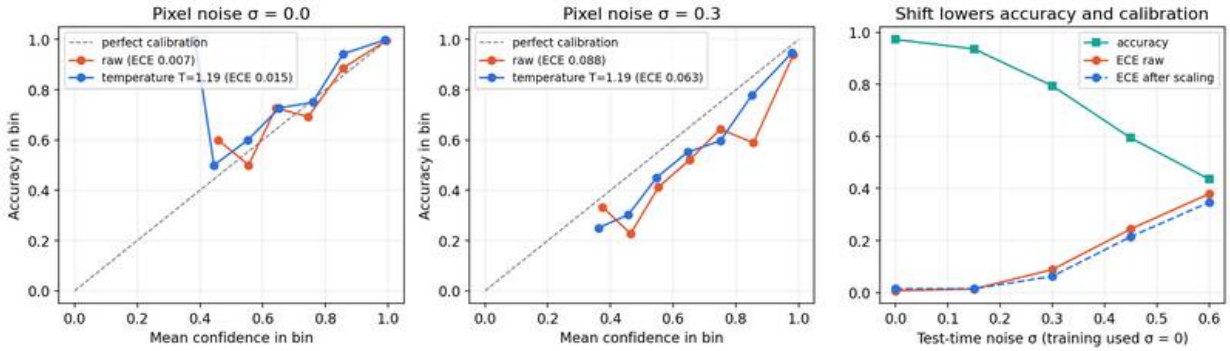


圖 4 — 上述網路的可靠度圖與偏移掃描（使用同一組雜訊樣式，依 σ 縮放）。點落在對角線下方代表過度自信。

僅線上版：*Temperature Scaling*：不改類別的信心校準 — *https://blog.shyandram.dev/zh-tw/learn/ml/evaluation-deployment/papers/temperature-scaling* 不改類別的信心校準/

穩健性與分布偏移

白話版。新相機、光線或族群可能改變表現與校準；應測試相關轉移，不假設部署與訓練相同。

三種偏移

把聯合分布寫成 $p(x, y) = p(y \mid x)\,p(x) = p(x \mid y)\,p(y)$，其中 x 是輸入、y 是標籤。偏移依訓練時（p_{tr}）與部署時（p_{te}）哪一部分改變來命名：

- **共變量偏移（covariate shift）**：$p_{\text{te}}(x) \neq p_{\text{tr}}(x)$，但 $p(y \mid x)$ 不變。新相機、新光線、新掃描儀。
- **標籤（先驗）偏移（label shift）**：$p_{\text{te}}(y) \neq p_{\text{tr}}(y)$，但 $p(x \mid y)$ 不變。疫情爆發時某種疾病變得更常見。
- **概念偏移（concept shift）**：$p(y \mid x)$ 本身改變。「垃圾郵件」的定義隨時間演變。

在純共變量偏移下，期望測試損失可改寫成加權的訓練損失：$\mathbb{E}_{\text{te}}[\ell] = \mathbb{E}_{\text{tr}}[w(x)\,\ell]$，重要性權重 $w(x) = p_{\text{te}}(x)/p_{\text{tr}}(x)$。對影像而言這些權重很難估計，但這個等式說明了：測試輸入落在訓練資料稀少的區域時，偏移的影響最大。在標籤偏移下，Lipton 等人證明，只要有固定黑箱分類器在留出訓練資料上的混淆矩陣，加上它在未標註測試資料上的預測頻率，就足以估計新的類別先驗並重新加權模型 [17]。

探測偏移的基準

- **ImageNet-C** 對 ImageNet 驗證影像施加 15 種損壞（雜訊、模糊、天氣、數位），並分多個嚴重程度；**ImageNet-P** 則加入擾動序列。兩者量測的是對常見損壞的穩健性，而不是對抗攻擊 [13]。我們的像素雜訊掃描就是一個迷你版本。
- **ImageNetV2** 盡可能依照原始收集流程重新建立 ImageNet 測試集。許多模型的準確率下降了 11–14%，但在原始測試集上的進步轉換成在新測試集上更大的進步，所以這個下降反映的是影

像稍微更難，而不是對重複使用的測試集過擬合 [14]。即使是小心的重新收集，本身也是一種偏移。
- **WILDS** 收集了十個具有自然偏移的資料集，包括來自不同醫院的腫瘤組織、新地點的自動相機以及來自新國家的衛星影像，並報告分布內與分布外表現之間有很大落差，而現有的緩解方法都沒能補上 [15]。

實務教訓：一定要加入至少一個依照你預期的部署變化（新院區、新裝置、較晚的時間）建立的**偏移**評估集，並把分布內與偏移後的結果並列回報。

效率：FLOPs、記憶體與延遲

白話版。「快」與「小」各有好幾種意思。運算量較少的模型不一定比較快，參數較少的模型也不一定比較省記憶體。要在目標裝置上量測。

計算運算量

一次**乘加運算**（multiply–accumulate, MAC）是一次乘法加一次加法，所以 1 MAC ≈ 2 個浮點運算（FLOPs）。對有 d_{in} 個輸入、d_{out} 個輸出的全連接層，以及有 C_{in} 個輸入通道、C_{out} 個輸出通道、$k \times k$ 卷積核與 $H' \times W'$ 輸出的卷積層：

$$\text{FLOPs}_{\text{linear}} \approx 2\, d_{\text{in}} d_{\text{out}}, \qquad \text{FLOPs}_{\text{conv}} \approx 2\, C_{\text{in}} k^2 C_{\text{out}} H' W'$$

以上是每個樣本的運算量（偏差與激活函數的成本很小）。各函式庫回報的是 MAC 還是 FLOPs 並不一致，請寫明。

為什麼 FLOPs 不等於速度

Dehghani 等人指出，常見的成本指標（參數量、FLOPs、吞吐量、延遲）可能彼此矛盾，只回報其中一個會讓效率看起來失真 [18]。主要原因：

- **記憶體流量**。若某層必須從記憶體讀取權重，卻只做很少的運算，它受限於記憶體頻寬而非運算。對單一樣本套用線性層時，要讀取 $4d_{\text{in}}d_{\text{out}}$ 位元組的 fp32 權重，只做 $2d_{\text{in}}d_{\text{out}}$ 個 FLOPs：每位元組 0.5 個 FLOPs，遠不足以讓現代加速器忙起來。批次化可以提高這個比率（稱為**運算強度**，arithmetic intensity）。
- **平行度**。深度可分離卷積與許多小層的 FLOPs 很少，卻無法充分利用硬體。
- **核心支援**。在目標執行環境上沒有最佳化核心的運算，無論 FLOPs 多少都可能很慢。

記憶體帳

推論時，記憶體要放權重（參數量 × 每個值的位元組數）、目前仍在使用的各層激活值、核心的工作空間，對自迴歸 Transformer 還有會隨上下文長度與批次大小成長的 **KV 快取**。訓練時還要加上梯度與最佳化器狀態：Adam 每個參數多存兩個值，所以使用 Adam 的 fp32 模型在計入激活值之前，就大約需要權重記憶體的 4×。

延遲量測協定

實際延遲取決於硬體、執行環境、批次大小、輸入解析度、數值精度與執行緒設定，而且前幾次呼叫因為一次性的初始化會比較慢。可靠的協定：

1. 固定並回報裝置、執行環境版本、精度、批次大小、輸入尺寸與執行緒數。
2. 先執行暖機迭代並丟棄結果。
3. 計時多次並回報百分位數（p50 與 p90 或 p99），而不只是平均：使用者感受到的是尾端。
4. 在 GPU 上，停止計時前要先同步裝置，因為核心是非同步啟動的。
5. 除了網路本身，也要量測端到端時間（前處理、資料傳輸、解碼）。

柏拉圖前緣（Pareto frontier）包含那些「沒有其他候選在每個面向（準確率、延遲、記憶體）都至少一樣好、且至少在一個面向嚴格更好」的候選模型。呈現前緣能讓取捨一目了然，而不必捏造一個唯一的「最佳模型」。

部署：匯出、壓縮與執行環境

白話版。匯出、壓縮與執行環境選擇可能改變預測及效能；應重新評估實際在目標裝置執行的產物。

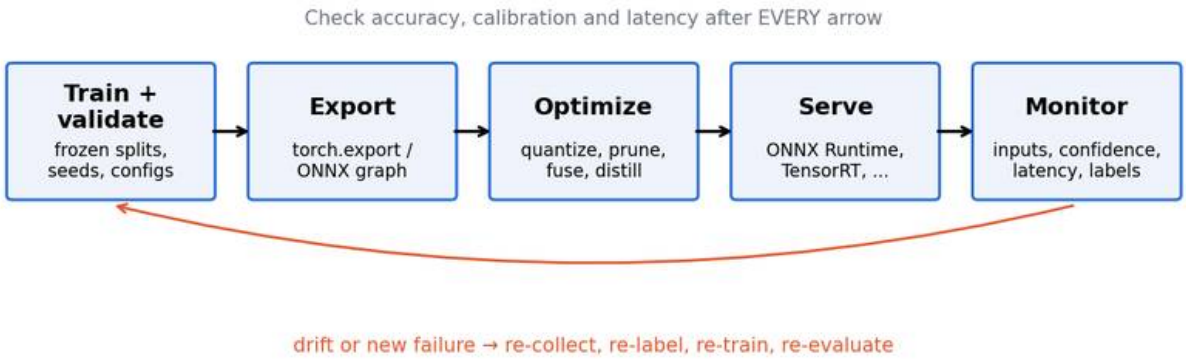


圖 *5* — 部署迴圈。每一步之後都應重新檢查準確率、校準與延遲，因為匯出、壓縮與執行環境都會改變數值。

匯出與推論執行環境

訓練程式碼寫在彈性高、由 Python 驅動的框架中。部署則偏好靜態的計算描述：

- `torch.export` 以預先（ahead-of-time）的方式把 PyTorch 模組追蹤成 `ExportedProgram`，也就是一張張量運算圖，可以序列化，並在沒有原始 Python 類別的情況下用不同輸入執行 [19]。
- **ONNX**（Open Neural Network Exchange）是機器學習模型的開放標準檔案格式，讓在一個框架中訓練的模型能被其他工具執行 [20]。

- **ONNX Runtime** 是 ONNX 模型的跨平台推論引擎。它把運算圖切成子圖，交給各個硬體專屬的**執行提供者**（execution provider，例如 CUDA、TensorRT、OpenVINO 或 CoreML）[21]。
- **NVIDIA TensorRT** 是一套 SDK，把模型（來自各框架或 ONNX）編譯成 NVIDIA GPU 上的最佳化引擎，使用層融合與 FP16、BF16、FP8、INT8 等較低精度 [22]。

這些執行環境靠融合運算、為確切的張量形狀挑選快速核心以及降低精度來加速。因此同一個模型在不同執行環境上的輸出可能略有差異。請在一批驗證資料上比較匯出後與原始模型的輸出，並在轉換後重新量測準確率。

量化

量化（quantization）用較少的位元來儲存與計算。在常見的**仿射（非對稱）均勻**量化中，實數值 x 會被映射成整數 q，再映射回來：

$$q = \mathrm{clip}\Big(\mathrm{round}\big(\tfrac{x}{s}\big) + z,\ 0,\ 2^{b} - 1\Big), \qquad \hat{x} = s\,(q - z),$$

其中 b 是位元寬度，$s > 0$ 是**尺度**（可表示值之間的間距），z 是整數**零點**，選擇它使實數 0 能被精確表示，$\hat{x}$ 是反量化後的近似值。對包含 0 的範圍 $[x_{\min}, x_{\max}]$，$s = (x_{\max} - x_{\min})/(2^{b} - 1)$、$z = \mathrm{round}(-x_{\min}/s)$。**對稱**變體固定 $z = 0$ 並使用有號範圍。Jacob 等人的純整數運算方案說明了如何用 8 位元整數矩陣乘法執行整個網路，以及如何為此進行訓練 [23]。

誤差有兩個來源：範圍內的**捨入**（每個值最多 $s/2$）與範圍外的**截斷**。兩者都取決於尺度的**粒度**：每個張量一個尺度、每個輸出通道一個，或每一小組權重一個。Nagel 等人提供實用指南，涵蓋這些選擇、**訓練後量化**（post-training quantization, PTQ：訓練後用一小份具代表性的資料校準範圍）與**量化感知訓練**（quantization-aware training, QAT：訓練時模擬捨入，讓權重去適應）[24]。Gholami 等人則綜述了更廣的設計空間 [25]。

```
import numpy as np

def quantize(x, bits=8, axis=None):
    """Asymmetric (affine) uniform quantization, per tensor or per channel."""
    qmin, qmax = 0, 2**bits - 1
    lo = x.min(axis=axis, keepdims=True) if axis is not None else x.min()
    hi = x.max(axis=axis, keepdims=True) if axis is not None else x.max()
    lo, hi = np.minimum(lo, 0), np.maximum(hi, 0)      # keep 0 exactly representable
    s = (hi - lo) / (qmax - qmin)                       # scale
    z = np.round(qmin - lo / s)                         # zero point (integer)
    q = np.clip(np.round(x / s) + z, qmin, qmax)        # stored integers
    return s * (q - z)                                  # dequantized values

rng = np.random.default_rng(0)
# 64 output channels whose weight magnitudes differ by up to 100x
W = rng.normal(size=(64, 256)) * np.logspace(-2, 0, 64)[:, None]
for bits in [8, 4]:
    e_t = np.abs(quantize(W, bits) - W).mean()
    e_c = np.abs(quantize(W, bits, axis=1) - W).mean()
    print(f"{bits}-bit mean |error|: per-tensor {e_t:.5f}  per-channel {e_c:.5f}")
print("zero maps back to:", quantize(np.array([-1.0, 0.0, 3.0]))[1])
```

輸出：

```
8-bit mean |error|: per-tensor 0.00632  per-channel 0.00124
4-bit mean |error|: per-tensor 0.06718  per-channel 0.02112
zero maps back to: 0.0
```

當各通道的範圍差異很大時，共用一個尺度會把大部分格點浪費在最大的通道上，讓小通道被粗糙地捨入。在這裡，逐通道尺度讓 8 位元時的誤差降到約五分之一。在大型語言模型中，少數離群的激活通道會在更大規模上造成同樣的問題，這正是 GPTQ（逐層、補償誤差的權重捨入）與 AWQ（依激活找出並保護重要權重通道）等方法在 3–4 位元純權重量化中處理的問題 [42], [43]。

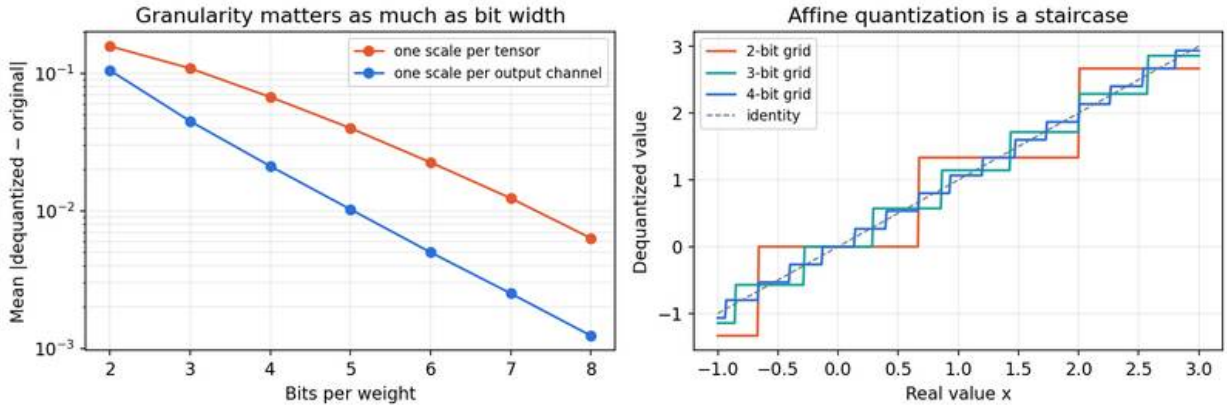


圖 *6* — 每多一個位元，量化誤差大約減半；尺度粒度的選擇則讓整條曲線上下平移。

剪枝

剪枝（pruning）移除貢獻很小的權重、通道、注意力頭或區塊。**非結構化**剪枝把個別權重設為零；它可以移除很大比例的參數，例如 Deep Compression 結合剪枝、以量化實現的權重共享與 Huffman 編碼，大幅縮小網路 [26]。但零星分布的零只有在執行環境具備該稀疏樣式的核心時才會省時間。**結構化**剪枝移除整個通道或注意力頭，產生較小的稠密張量，任何執行環境都能利用。Blalock 等人彙整了 81 篇剪枝論文，發現基準與指標不一致，使人難以判斷哪些方法真的有效，並提出標準化的評測框架 [27]。Hoefler 等人深入綜述了訓練與推論中的稀疏性 [28]。

知識蒸餾

蒸餾（distillation）訓練一個小的**學生**模型去模仿大的**教師**模型。Hinton 等人提出讓學生匹配教師軟化後的輸出分布 [29]。設溫度為 T，教師與學生的 logits 分別為 ℓ^t 與 ℓ^s，軟化機率為 $p^{(T)} = \mathrm{softmax}(\ell/T)$，權重 $\lambda \in [0,1]$：

$$\mathcal{L} = (1-\lambda)\,\mathcal{L}_{\mathrm{CE}}(y,\, p^{s,(1)}) + \lambda\, T^2\, D_{\mathrm{KL}}(p^{t,(T)} \,\|\, p^{s,(T)}).$$

軟化的教師分布帶有「暗知識」：哪些錯誤類別彼此相似。軟項的梯度按 $1/T^2$ 縮放，所以乘上 T^2 能在改變 T 時保持兩項的平衡 [29]。Gou 等人依**傳遞什麼**（輸出、中間特徵、樣本間關係）與**如**

何傳遞（離線、線上、自蒸餾）整理了後續研究 [30]。學生會繼承教師的能力，也會繼承它的錯誤與偏差。

程式碼：純權重 int8、延遲與匯出

這個範例把一個小 MLP 的權重以 int8 儲存（每個輸出通道一個尺度），並在 CPU 上量測檔案大小與延遲。只用到 PyTorch 核心功能。

```
import io, time
import numpy as np
import torch
from torch import nn

torch.manual_seed(0)
model = nn.Sequential(nn.Linear(512, 2048), nn.ReLU(),
                      nn.Linear(2048, 2048), nn.ReLU(),
                      nn.Linear(2048, 10)).eval()
x = torch.randn(32, 512)

class Int8WeightLinear(nn.Module):
    """Weight-only int8 storage, symmetric per output channel."""
    def __init__(self, lin):
        super().__init__()
        w = lin.weight.detach()
        scale = w.abs().amax(dim=1, keepdim=True) / 127
        self.register_buffer("q", torch.round(w / scale).to(torch.int8))
        self.register_buffer("scale", scale)
        self.register_buffer("bias", lin.bias.detach().clone())
    def forward(self, x):                       # dequantize, then fp32 matmul
        return nn.functional.linear(x, self.q.float() * self.scale, self.bias)

int8 = nn.Sequential(*[Int8WeightLinear(m) if isinstance(m, nn.Linear) else m
                       for m in model]).eval()

def size_mb(m):
    buf = io.BytesIO()
    torch.save(m.state_dict(), buf)
    return buf.getbuffer().nbytes / 1e6

def latency_ms(m, x, warmup=10, runs=100):
    with torch.inference_mode():
        for _ in range(warmup):                 # first calls pay one-off costs
            m(x)
        times = []
        for _ in range(runs):
            t0 = time.perf_counter()
            m(x)
            times.append((time.perf_counter() - t0) * 1e3)
    return np.percentile(times, [50, 90]).round(2)

with torch.inference_mode():
    gap = (model(x) - int8(x)).abs().max().item()
    agree = (model(x).argmax(1) == int8(x).argmax(1)).float().mean().item()
```

```
print(f"fp32 file: {size_mb(model):.1f} MB  p50/p90 latency {latency_ms(model, x)} ms")
print(f"int8 file: {size_mb(int8):.1f} MB  p50/p90 latency {latency_ms(int8, x)} ms")
print(f"max |logit difference| = {gap:.4f}, top-1 agreement = {agree:.3f}")

# Ahead-of-time graph capture: a portable program, independent of this Python class.
exported = torch.export.export(model, (x,))
ops = [n.target for n in exported.graph.nodes if n.op == "call_function"]
print(type(exported).__name__, "->", [str(o).split(".")[1] for o in ops])
```

在 CPU 上（四個執行緒）執行一次的輸出：

```
fp32 file: 21.1 MB  p50/p90 latency [2.74 4.49] ms
int8 file: 5.3 MB  p50/p90 latency [ 9.23 11.06] ms
max |logit difference| = 0.0018, top-1 agreement = 1.000
ExportedProgram -> ['linear', 'relu', 'linear', 'relu', 'linear']
```

你的延遲數字會不同；重點在於趨勢。檔案小了四倍，輸出幾乎沒變。但 int8 版本反而**更慢**，因為它每次呼叫都把權重轉回 fp32，再執行同樣的 fp32 矩陣乘法。只有在執行環境具備該運算的整數（或低精度）核心時，或工作負載受記憶體頻寬限制、讀取較少位元組成為主因時，較低精度才會變快。這正是「壓縮了」與「加速了」之間的落差，也是 TensorRT、ONNX Runtime 以及 PyTorch 自家的 `torchao` 函式庫 [46] 要彌補的地方。這個 MLP 的運算量為每個樣本 $2 \times (512 \cdot 2048 + 2048 \cdot 2048 + 2048 \cdot 10) \approx 10.5$ MFLOPs，兩個版本完全相同。

部署後的監控

白話版。即使程式碼完全沒變，模型也可能變差，因為資料變了。要觀察輸入與輸出，盡可能收集新的標籤，並事先決定什麼情況會觸發調查。

Sculley 等人描述 ML 系統如何累積隱藏的「技術債」：糾纏的特徵、未申報的預測使用者、模型輸出影響未來訓練資料的回饋迴圈，以及依賴會在無預警下改變的外部資料 [33]。Paleyes 等人收集已發表的部署案例研究，發現部署流程的每個階段都會出問題 [39]。有用的訊號，由便宜到最有資訊量：

- **系統健康**：延遲百分位數、錯誤率、記憶體、吞吐量。
- **輸入統計**：影像亮度、解析度、裝置類型、缺失欄位；與訓練時的參考時段比較。
- **輸出統計**：預測類別與信心的分布。這些不需要標籤。
- **延遲到來的標籤**：定期標註一小部分線上資料，是直接估計準確率的唯一方式。

要比較線上時段與參考時段，雙樣本檢定是簡單的起點。對經驗累積分布函數分別為 F_{ref} 與 F_{live} 的一維樣本，**Kolmogorov–Smirnov 統計量**為

$$D = \sup_t |F_{\text{ref}}(t) - F_{\text{live}}(t)|,$$

也就是兩條曲線之間最大的垂直差距。Rabanser 等人比較了多種偏移偵測方法，發現先用預訓練分類器降維（例如取其 softmax 輸出）再做雙樣本檢定，在他們研究的偏移中表現最好 [32]。

```
import numpy as np
from scipy.stats import ks_2samp
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X, y = load_digits(return_X_y=True)
X = X / 16.0
Xtr, Xlive, ytr, ylive = train_test_split(X, y, test_size=0.5, random_state=0)
Xtr, Xref, ytr = Xtr[:600], Xtr[600:], ytr[:600]   # reference window from training time
clf = LogisticRegression(max_iter=2000).fit(Xtr, ytr)
ref_conf = clf.predict_proba(Xref).max(1)     # label-free signal: top-class confidence

rng = np.random.default_rng(0)
for name, noise in [("week 1 (clean)", 0.0), ("week 2 (new camera)", 0.25)]:
    batch = rng.choice(len(Xlive), 200, replace=False)
    Xb = np.clip(Xlive[batch] + noise * rng.normal(size=Xlive[batch].shape), 0, 1)
    conf = clf.predict_proba(Xb).max(1)
    stat, p = ks_2samp(ref_conf, conf)
    acc = (clf.predict(Xb) == ylive[batch]).mean()   # usually NOT available live
    print(f"{name}: mean confidence {conf.mean():.3f}, KS={stat:.3f}, p={p:.2g}, "
          f"(hidden) accuracy {acc:.3f}")
```

輸出：

```
week 1 (clean): mean confidence 0.846, KS=0.089, p=0.28, (hidden) accuracy 0.950
week 2 (new camera): mean confidence 0.757, KS=0.277, p=1.3e-08, (hidden) accuracy 0.885
```

信心分布在沒有任何標籤的情況下就標記出第 2 週，而隱藏的準確率證實確實發生了變化。偵測器告訴你資料**有沒有**改變，而不是準確率**是否**下降：無害的偏移也會觸發它，而在很大的時段下，即使微小的變化也會變得「顯著」。用它來觸發調查與標註，而不是自動重新訓練。

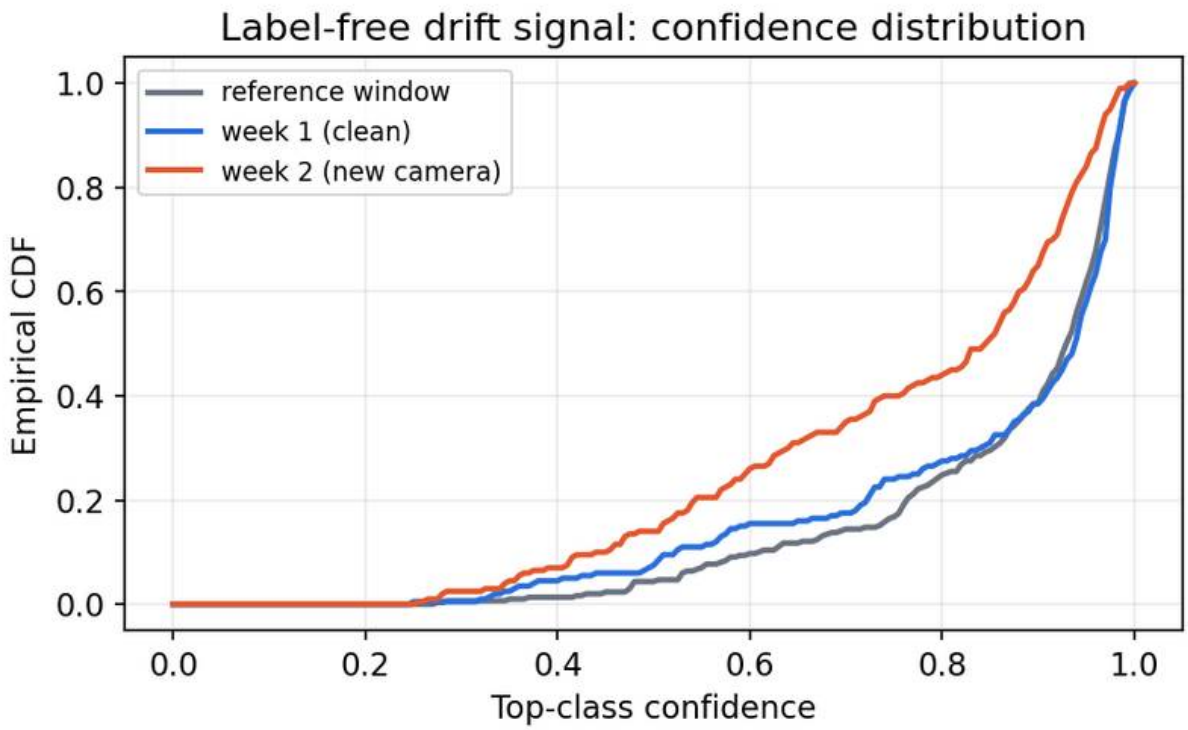


圖 7 — *Kolmogorov–Smirnov* 統計量就是參考曲線與線上時段曲線之間最大的垂直差距。

可重現性與文件

白話版。別人（包括六個月後的你）應該能重跑你的實驗並得到相同的數字，讀者也應該能看出這個模型是做什麼用的、哪些場合不該使用。

讓實驗可重複

為每個函式庫固定並記錄隨機種子，但要知道它的極限：PyTorch 文件明言，不同版本、不同 commit 或不同平台之間無法保證完全可重現的結果，並建議在需要逐位元一致時設定種子、呼叫 `torch.use_deterministic_algorithms(True)` 並關閉 cuDNN benchmark [34]。除了種子，還要記錄：

- 資料集版本、切分 ID（實際的樣本 ID 清單）與去重規則；
- 前處理與資料擴增程式碼、預訓練檢查點及其來源；
- 完整設定（最佳化器、學習率排程、停止規則）、軟體環境與硬體；
- 每一個驗證階段的決策，以及唯一一次的最終測試評估。

Pineau 等人報告的 NeurIPS 2019 可重現性計畫，推出了程式碼提交政策、社群可重現性挑戰，以及正好涵蓋上述項目的檢查表 [37]。

為模型與資料寫文件

模型卡（model cards）是隨訓練好的模型附上的簡短文件，說明預期用途、評估資料、依相關群體與條件細分的表現，以及已知限制 [35]。**資料集說明書**（datasheets for datasets）要求資料集建立者記錄動機、組成、收集過程、前處理、建議用途與維護方式 [36]。兩者都讓本章的評估決策，對依賴它們的人變得透明。

結業實驗

選一個具備適當規模、可合法使用資料集的任務。定義部署母體與切分單位。比較三個候選：一個傳統基準、凍結預訓練特徵加小分類頭，以及一個微調過的網路。對每一個，回報附 bootstrap 區間的測試指標、與最佳基準的成對比較、可靠度圖、一個偏移集上的結果，以及目標裝置上的延遲百分位數。寫一頁模型卡。最終測試集只打開一次；若它揭露了失敗而你修改了模型，就把舊結果視為開發證據，並為修改後的主張取得新的評估。

現代觀點

評估與部署的工程本身已經成為一個研究領域。以下幾篇綜述與回顧勾勒了它的樣貌。

資料洩漏與可重現性。 Kapoor 與 Narayanan 對以 ML 為基礎之科學的回顧，記錄了 17 個領域中 329 篇論文的洩漏問題，並提出「模型資訊表」（model info sheets），要求作者論證其訓練／測試分離為何成立 [4]。結合 Bouthillier 等人的變異研究 [10] 與 NeurIPS 可重現性計畫 [37]，共識是：多數過度樂觀的結果來自評估協定的錯誤，而不是什麼特殊的統計問題。

不確定性與校準。 Gawlikowski 等人綜述深度網路的不確定性估計：他們區分可降低的模型（認知）不確定性與不可降低的資料（偶然）不確定性，把方法分為確定性網路、貝氏網路、集成與測試時資料擴增，並回顧校準方法及其實務限制 [38]。Ovadia 等人則補上實證警告：事後校準在偏移下不夠用，而對多個模型做邊際化的方法（例如集成）表現較好 [16]。

實務部署。 Paleyes 等人綜述已發表的案例研究，把挑戰對應到部署流程的各個階段，並顯示實務工作者在每個階段都會遇到問題 [39]，延伸了 Sculley 等人的技術債觀點 [33]。

模型壓縮。 Cheng 等人提供了 CNN 剪枝、低秩分解、緊湊卷積與蒸餾的經典綜覽 [31]；Gholami 等人 [25] 與 Nagel 等人 [24] 聚焦量化；Hoefler 等人聚焦稀疏性 [28]；Gou 等人聚焦蒸餾 [30]。對大型語言模型，Zhu 等人綜述了針對數十億參數模型的量化、剪枝、蒸餾與低秩方法 [41]。2023–2026 年一個明顯的趨勢是：4 位元純權重量化（GPTQ [42]、AWQ [43]）與 FP8 等低精度格式，已經從論文走進生產工具：TensorRT 列出 FP8、INT8、FP4 與 INT4 等支援精度 [22]，而 PyTorch 的量化工具正整合到 `torchao` 函式庫中 [46]。

全面評估。對基礎模型，HELM 同時在許多情境與多個指標上評估許多模型（準確率、校準、穩健性、公平性、偏差、毒性與效率），並主張單一排行榜數字會掩蓋這些取捨 [40]。同樣的多指標習慣也適用於視覺模型。

規模化。規模定律（scaling laws）擬合模型大小、資料、算力與損失之間的經驗關係。Hoffmann 等人發現在固定算力預算下，語言模型的參數量與訓練 token 數應大致等比例擴增 [44]；Zhai 等人研究 ViT 準確率如何隨模型大小、資料與算力擴展 [45]。一個領域擬合出的常數不應直接搬到另一個領域；可以遷移的教訓是：每個結果都要報告背後的算力與資料。

開放問題。評估資料混入網路規模的預訓練資料（資料污染）很難排除；未知偏移下的校準仍未解決；延遲數字很少能在不同硬體世代之間轉移；生成式與互動式系統的評估（見 DLCV 13）仍缺乏公認的協定。

重點整理

- 依部署時的新資料單位切分，並在各訓練折內擬合前處理。
- 相依資料使用分組或時間切分，並把模型選擇與回報評估分開。
- 在配對範例上比較模型，並依實際獨立取樣單位估計不確定性。
- 校準不同於準確率；溫度縮放可能改善相符資料的校準，但不保證轉移後仍成立。
- 運算量、參數量、記憶體與延遲衡量不同成本，應測試完整目標流程。
- 匯出或壓縮後重新評估，監控資料變化，並記錄版本、協定及預期用途。

練習

1. **洩漏稽核**。同事先在全部 1,797 張數字影像上擬合 PCA，再切分訓練與測試並回報測試準確率。說明洩漏在哪裡，以及為什麼 PCA 從未看過標籤卻仍有洩漏。你會如何修正程式碼？

   提示

   主成分方向是用測試影像算出來的，所以表示法被調整得貼近測試分布。把 `PCA` 放進 `Pipeline`，使它只在每個訓練折上擬合。對 PCA 而言這個偏差常常不大，但對使用標籤的監督式前處理（例如依標籤做特徵選擇），偏差可能很大。

2. **罕見的正例**。某資料集只有 1% 是正例影像。永遠預測負例的分類器，準確率、正例召回率與平衡準確率各是多少？你會向醫師回報哪個指標？為什麼？

   提示

   準確率 99%、召回率 0、平衡準確率 0.5（正例召回率 0 與負例召回率 1 的平均）。應回報在臨床選定操作點下的召回率（敏感度）與特異度或精確率，並附上區間，因為真正重要的是漏掉一個正例的代價。

3. **測試集要多大？**你預期準確率約 90%，希望 95% 信賴區間約為 ± 1 個百分點。用常態近似，大約需要多少個獨立測試樣本？如果影像來自 50 位病人，會有什麼不同？

   提示

   解 $1.96\sqrt{0.9 \cdot 0.1/n} = 0.01$，得 $n \approx 3{,}460$。若只有 50 位病人，樣本並不獨立；有效樣本數比較接近 50 而非影像張數，所以區間會寬得多。請用以病人為單位重抽樣的 bootstrap。

4. **手算溫度**。一個三類別模型輸出 logits $(4, 1, 0)$。計算 $T = 1$ 與 $T = 2$ 時最高類別的機率。預測類別會改變嗎？什麼時候擬合出的 T 會小於 1？

提示

$T = 1$：$e^4/(e^4 + e^1 + e^0) \approx 0.936$。$T = 2$：$e^2/(e^2 + e^{0.5} + 1) \approx 0.736$。argmax 不變。當模型信心不足時（例如強正則化或標籤平滑之後），擬合出的 $T < 1$。

5. **壓縮的宣稱**。一個剪枝後的模型參數只剩原本的一半，但在你的 GPU 上延遲完全相同。提出三種可能解釋，以及一個能區分它們的實驗。

提示

非結構化的零仍被稠密核心拿去相乘；剩下的成本主要來自激活值、記憶體流量或未改變的層；或批次太小，使核心啟動開銷成為主因。逐層剖析時間，並與一個張量真正變小的結構化剪枝模型比較。

6. **設計一個監控方案**。你的瑕疵偵測器在工廠產線上執行；人工檢驗的標籤要兩週後才會回來。設計監控方案：記錄什麼、哪些統計量要與哪個參考比較、什麼門檻會觸發警報，以及警報後要做什麼。

提示

每天記錄延遲、影像統計（亮度、模糊程度）、預測瑕疵率與信心直方圖；以 KS 或族群穩定度指標與驗證時的參考時段比較；在持續性變化時才發出警報，而不是單日波動。警報後立即標註一批樣本、計算附區間的準確率，然後才決定要重新校準、微調還是回滾。

參考文獻

1. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016, ch. 11 “Practical Methodology”. book
2. G. James, D. Witten, T. Hastie, R. Tibshirani and J. Taylor, *An Introduction to Statistical Learning: with Applications in Python*, Springer, 2023. book
3. scikit-learn developers, “Common pitfalls and recommended practices,” scikit-learn documentation. docs
4. S. Kapoor and A. Narayanan, “Leakage and the reproducibility crisis in machine-learning-based science,” *Patterns*, vol. 4, no. 9, 2023. doi
5. S. Varma and R. Simon, “Bias in error estimation when using cross-validation for model selection,” *BMC Bioinformatics*, vol. 7, art. 91, 2006. doi
6. G. C. Cawley and N. L. C. Talbot, “On over-fitting in model selection and subsequent selection bias in performance evaluation,” *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010. paper
7. M. Bińkowski, D. J. Sutherland, M. Arbel and A. Gretton, “Demystifying MMD GANs,” ICLR, 2018. arXiv
8. T. G. Dietterich, “Approximate statistical tests for comparing supervised classification learning algorithms,” *Neural Computation*, vol. 10, no. 7, 1998. doi
9. B. Efron, “Bootstrap methods: another look at the jackknife,” *The Annals of Statistics*, vol. 7, no. 1, 1979. doi

10. X. Bouthillier et al., “Accounting for variance in machine learning benchmarks,” MLSys, 2021. arXiv
11. C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, “On calibration of modern neural networks,” ICML, 2017. arXiv
12. M. Minderer et al., “Revisiting the calibration of modern neural networks,” NeurIPS, 2021. arXiv
13. D. Hendrycks and T. Dietterich, “Benchmarking neural network robustness to common corruptions and perturbations,” ICLR, 2019. arXiv
14. B. Recht, R. Roelofs, L. Schmidt and V. Shankar, “Do ImageNet classifiers generalize to ImageNet?,” ICML, 2019. paper
15. P. W. Koh et al., “WILDS: A benchmark of in-the-wild distribution shifts,” ICML, 2021. paper
16. Y. Ovadia et al., “Can you trust your model’s uncertainty? Evaluating predictive uncertainty under dataset shift,” NeurIPS, 2019. arXiv
17. Z. C. Lipton, Y.-X. Wang and A. Smola, “Detecting and correcting for label shift with black box predictors,” ICML, 2018. arXiv
18. M. Dehghani, A. Arnab, L. Beyer, A. Vaswani and Y. Tay, “The efficiency misnomer,” ICLR, 2022. arXiv
19. PyTorch, “torch.export,” PyTorch documentation. docs
20. ONNX, “Open Neural Network Exchange: the open standard for machine learning interoperability.” site
21. ONNX Runtime, “Welcome to ONNX Runtime,” documentation. docs
22. NVIDIA, “NVIDIA TensorRT documentation.” docs
23. B. Jacob et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” CVPR, 2018. doi
24. M. Nagel et al., “A white paper on neural network quantization,” arXiv:2106.08295, 2021. arXiv
25. A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” book chapter in *Low-Power Computer Vision*; arXiv:2103.13630, 2021. arXiv
26. S. Han, H. Mao and W. J. Dally, “Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding,” ICLR, 2016. arXiv
27. D. Blalock, J. J. Gonzalez Ortiz, J. Frankle and J. Guttag, “What is the state of neural network pruning?,” MLSys, 2020. arXiv
28. T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden and A. Peste, “Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks,” *Journal of Machine Learning Research*, vol. 22, 2021. arXiv
29. G. Hinton, O. Vinyals and J. Dean, “Distilling the knowledge in a neural network,” NIPS Deep Learning Workshop, 2014; arXiv:1503.02531. arXiv
30. J. Gou, B. Yu, S. J. Maybank and D. Tao, “Knowledge distillation: a survey,” *International Journal of Computer Vision*, vol. 129, 2021. doi
31. Y. Cheng, D. Wang, P. Zhou and T. Zhang, “Model compression and acceleration for deep neural networks: the principles, progress, and challenges,” *IEEE Signal Processing Magazine*, vol. 35, no. 1, 2018. doi arXiv
32. S. Rabanser, S. Günnemann and Z. C. Lipton, “Failing loudly: an empirical study of methods for detecting dataset shift,” NeurIPS, 2019. arXiv

33. D. Sculley et al., “Hidden technical debt in machine learning systems,” NeurIPS, 2015. paper
34. PyTorch, “Reproducibility,” PyTorch documentation. docs
35. M. Mitchell et al., “Model cards for model reporting,” ACM FAT*, 2019. arXiv
36. T. Gebru et al., “Datasheets for datasets,” *Communications of the ACM*, vol. 64, no. 12, 2021. doi
37. J. Pineau et al., “Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program),” *Journal of Machine Learning Research*, vol. 22, 2021. paper
38. J. Gawlikowski et al., “A survey of uncertainty in deep neural networks,” *Artificial Intelligence Review*, vol. 56, 2023. doi
39. A. Paleyes, R.-G. Urma and N. D. Lawrence, “Challenges in deploying machine learning: a survey of case studies,” *ACM Computing Surveys*, vol. 55, no. 6, 2022. doi
40. P. Liang et al., “Holistic evaluation of language models,” *Transactions on Machine Learning Research*, 2023. arXiv
41. X. Zhu, J. Li, Y. Liu, C. Ma and W. Wang, “A survey on model compression for large language models,” *Transactions of the Association for Computational Linguistics*, vol. 12, 2024. doi
42. E. Frantar, S. Ashkboos, T. Hoefler and D. Alistarh, “GPTQ: accurate post-training quantization for generative pre-trained transformers,” ICLR, 2023. arXiv
43. J. Lin et al., “AWQ: activation-aware weight quantization for LLM compression and acceleration,” MLSys, 2024. arXiv
44. J. Hoffmann et al., “Training compute-optimal large language models,” arXiv:2203.15556, 2022; NeurIPS 2022 version titled “An empirical analysis of compute-optimal large language model training.” arXiv
45. X. Zhai, A. Kolesnikov, N. Houlsby and L. Beyer, “Scaling vision transformers,” CVPR, 2022. doi
46. PyTorch, “torchao documentation.” docs

篇 II

電腦視覺深度學習

DLCV 01・深度學習的影像處理與電腦視覺

深度學習如何重新整理影像處理、影像分析與電腦視覺：從傳統圖形識別流程，到 CNN、Vision Transformer、自監督骨幹網路與可提示的基礎模型。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/deep-learning-for-image-processing/

你將學到

- 區分影像到影像，以及影像到語意的任務。
- 說明端到端學習與預訓練骨幹—輸出頭流程。
- 連結卷積、步幅、填補、池化與解析度及感受野。
- 比較 CNN、編碼器—解碼器與 Transformer 的設計選擇。
- 為辨識、復原、生成或視覺語言任務選擇起點。

先看全貌

這篇是深度學習系列的入口。它不會深入任何單一任務，而是先給你一張地圖：影像問題有哪些種類、哪些網路設計能解決它們、這些設計又為什麼長成現在這樣。後續的教學會再分別深入影像分類、物件偵測、影像復原、低光源增強與物件追蹤。

> **用最簡單的話說**
>
> 想像一間照相館裡有兩種員工。**修圖師傅**拿到一張刮傷、模糊或太暗的照片，交回一張比較乾淨的照片：照片進、照片出。**偵探**看著一張照片，告訴你裡面**有什麼**：「一隻貓，坐在沙發上，在左邊。」以前的電腦視覺，是由專家幫這兩位員工準備好一套固定的工具。深度學習則讓他們看過幾百萬個例子之後，自己打造工具。現在甚至出現了一位「超級員工」，看過數十億張圖，你只要用幾個字告訴他要做什麼就好。

為什麼重要：幾乎所有現代相機的處理流程、醫學影像產品、駕駛輔助系統與修圖 App，都是由本篇介紹的元件組成。看懂這張地圖，你就能為新問題挑一個合理的起點、帶著批判的眼光讀論文，也能看出數位影像處理系列裡的哪些傳統想法仍活在網路裡面（其實很多）。

> **關於本篇**
>
> 本篇改寫並擴充自作者先前的文章〈深度學習的數位影像處理介紹〉[1]，保留原文的架構與觀點，並補上細節、數學與程式碼。

相關篇章。 DLCV 01A 把本篇的觀念變成實用的訓練流程：資料管線、學習率排程、正則化、除錯與公平的實驗比較。DLCV 01B 整理網路的積木（殘差與注意力區塊、正規化、上取樣、跳接），並說明分類網路與影像復原網路如何以不同方式組合它們。

影像任務的三個層次

白話版。影像工作可以改善像素、擷取結構或性質，或推論場景意義。依輸入與輸出區分任務。

精確版。依照 Gonzalez 與 Woods [2] 的分法，可以用「輸入什麼、輸出什麼」把影像工作分成三個層次。

層次	輸入 → 輸出	典型操作	傳統例子
影像處理（狹義；低階）	影像 → 影像	逐像素或鄰域的基本步驟	直方圖等化、濾波、傅立葉轉換（IP 03、IP 04）
影像分析（中階）	影像 → 屬性或被標示出的區域	分割、特徵擷取、描述，運用先驗知識	人臉影像中的膚色區域、X 光影像中的血管區域（IP 10、IP 12）
電腦視覺（高階；理解）	影像 → 意義	辨識、詮釋、判斷	分類、「畫面中有沒有行人？」（IP 13）

這三者的界線並不嚴格，實際系統往往三者都會用到。例如一個血管分割工具，會先去雜訊（低階）、再擷取血管圖（中階），最後可能標記出狹窄處（高階）。這三個層次描述的是**計算的種類**，而不是三個分開的部門。

深度學習如何重畫這張地圖

在深度學習的論文裡，你幾乎看不到「中階」這個詞。這個領域依照**輸出的形式**，只把任務分成兩大類 [1]：

- **低階視覺**：輸出是一張和輸入（大致）同樣大小的影像，目標是逐像素準確的訊號。例如去雜訊、去模糊、超解析度、除霧、低光源增強、影像到影像轉換。
- **高階視覺**：輸出是**語意**描述，可能是一個標籤、一組帶標籤的方框，或是每個像素一個標籤。

原本的「影像分析」被併入兩者之一。分割的輸出是逐像素的圖，**看起來**像影像到影像的任務；但它的數值是需要辨識才能得到的類別標籤，所以被歸為高階視覺。至於輸出一張更乾淨的亮度影像的邊緣或血管增強，則歸為低階視覺。

這樣分的理由很實際。兩類任務在損失函數（像素迴歸 vs. 分類）、架構（全解析度的編碼器–解碼器 vs. 逐步降採樣的骨幹網路）、訓練資料（劣化/乾淨影像對 vs. 人工標註）與評估指標（保真度 vs. 正確率）上都不同。圖 1 畫出了這個重新整理。

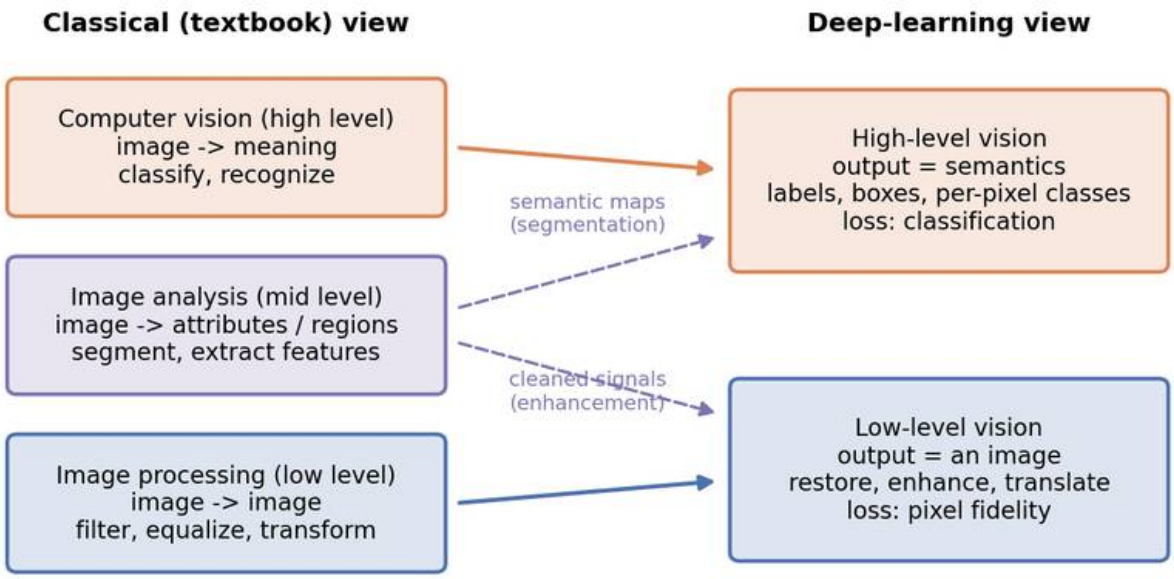


圖 *1* — 傳統的三個層次（左）與深度學習使用的兩大類（右）。中階的影像分析依輸出是「清理過的訊號」還是「語意圖」而被分到兩邊。

從圖形識別流程到端對端學習

白話版。傳統辨識把人工特徵與分類分開；端到端學習則共同訓練兩者。

精確版。傳統流程 [3] 有四個階段：

1. **感測器**（sensor）。擷取圖形（相機、掃描器、顯微鏡）。
2. **特徵生成與選擇**（feature generation & selection）。以人工設計的映射 ϕ（動差、紋理統計、邊緣直方圖、關鍵點描述子），從影像 I 計算出向量 $\mathbf{x} = \phi(I) \in \mathbb{R}^d$，再保留最有資訊量的分量。
3. **分類器設計**（classifier）。學習決策規則 $\hat{y} = g(\mathbf{x}; \theta)$，$\theta$ 為參數，例如貝氏分類器、最近鄰規則或支援向量機。
4. **系統驗證**（evaluation）。在保留的資料上量測錯誤率，太高就回到第 2 或第 3 步。

被學習的只有 θ。特徵映射 ϕ 由工程師固定，所以整個系統的上限就是 ϕ 的品質。特徵擷取（IP 12）那一章可以看到，這些映射背後花了多少巧思。

端對端學習讓特徵映射也變成可學習的。深度網路計算

$$\hat{y} = g(\phi(I; \mathbf{w}); \theta),$$

其中 $\phi(\cdot\,; \mathbf{w})$ 是權重為 $\mathbf{w}$ 的多層網路，$g(\cdot\,; \theta)$ 是一個小的 head，$\mathbf{w}$ 與 θ 在標註資料 (I, y) 上以梯度下降與反向傳播共同最小化損失 $\mathcal{L}(\hat{y}, y)$。輸入就是原始像素；除了架構與損失函數之外，沒有任何東西是人工設計的。

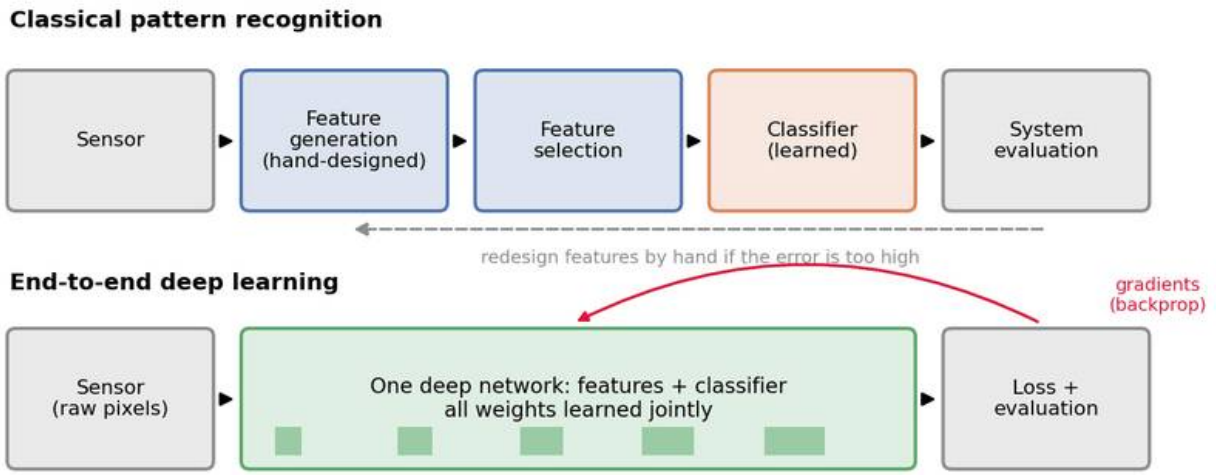


圖 2 — 傳統圖形識別（上）與端對端學習（下）。深度網路裡「特徵」與「分類器」的功能區分仍然存在，但兩者之間沒有清楚的分界。

網路內部的功能區分其實還在：前面的層像特徵擷取器，最後幾層像分類器。但實務上，你通常無法指著某一層說「特徵到這裡結束」[1]。這帶來兩個結果：

- **資料取代了設計**。端對端系統需要大量標註資料；人工設計的 ϕ 原本承載的先驗知識，現在得由網路從資料中學到。
- **分工以更大的規模回歸**。大約從 2021 年起，標準作法是用一個**預訓練骨幹**（在巨大的資料集上訓練一次，常常不需要標註）加上一個在你的小資料集上訓練的**輕量任務 head**。這又是「特徵擷取器＋分類器」，只是特徵擷取器是學來的，而不是設計出來的。我們會在自監督骨幹那一節回到這一點。

卷積神經網路

白話版。卷積層拿一個小視窗在影像上滑動，在每個位置計算加權和，跟 IP 03 的空間濾波器一模一樣。差別在於：CNN 會**學習**濾波器的權重、一次使用很多個濾波器，並把數十層這樣的層與非線性函數交錯疊起來。

卷積層是一組學習而來的濾波器

精確版。設輸入特徵圖為 $X \in \mathbb{R}^{C_{\text{in}} \times H \times W}$（通道、高、寬）。一個有 C_{out} 個 $k \times k$ 濾波器的卷積層計算

$$Y_o(x,y) = b_o + \sum_{c=1}^{C_{\text{in}}} \sum_{i=-r}^{r} \sum_{j=-r}^{r} W_{o,c}(i,j)\, X_c(x+i, y+j), \qquad o = 1, \dots, C_{\text{out}},$$

其中奇數 k 時 $r = (k-1)/2$，$W_{o,c}(i,j)$ 是第 o 個濾波器在輸入通道 c、位移 (i,j) 的權重，b_o 是偏差。嚴格來說這是**相關**（correlation，不翻轉卷積核），如卷積教學所述；由於權重是學出來的，

翻不翻轉沒有差別。之後接一個逐元素的非線性函數，例如 $\mathrm{ReLU}(z) = \max(0, z)$。如果沒有它，一疊卷積層會塌縮成一個大的線性濾波器 [1]。

有兩個性質讓這種層特別適合影像：

- **局部連接**（local connectivity）。每個輸出只依賴一個 $k \times k$ 鄰域，因為相鄰像素彼此高度相關，距離遠的像素則關係弱得多。
- **權重共享**（weight sharing）。同一個濾波器用在每一個位置。這讓卷積層具有**平移等變性**（translation equivariance：輸入平移，輸出也跟著同樣平移），也大幅減少參數量。

省下的量非常可觀。從 $224 \times 224 \times 3$ 的影像連到 64 個神經元的全連接層需要 $224 \cdot 224 \cdot 3 \cdot 64 + 64 \approx 960$ 萬個參數。從 3 個通道到 64 個通道的 3×3 卷積只需要 $3 \cdot 3 \cdot 3 \cdot 64 + 64 = 1{,}792$ 個，而且輸出的是 64 張**全解析度**的特徵圖，而不是 64 個數字。

下面的程式把 PyTorch 卷積層的兩個濾波器手動設成 Sobel 卷積核，讓這一層變成邊緣偵測器。訓練則會改由梯度下降來決定這些數字。

```
import torch, torch.nn as nn, torch.nn.functional as F
from skimage import data, img_as_float

x = torch.tensor(img_as_float(data.camera()), dtype=torch.float32)[None, None]  # (1, 1, 512,
 ↪  512)
conv = nn.Conv2d(1, 2, kernel_size=3, padding=1, bias=False)
sobel_x = torch.tensor([[-1., 0., 1.], [-2., 0., 2.], [-1., 0., 1.]])
with torch.no_grad():
    conv.weight[0, 0] = sobel_x        # channel 0: responds to vertical edges
    conv.weight[1, 0] = sobel_x.T      # channel 1: responds to horizontal edges
    y = F.relu(conv(x))                # (1, 2, 512, 512): two feature maps
```

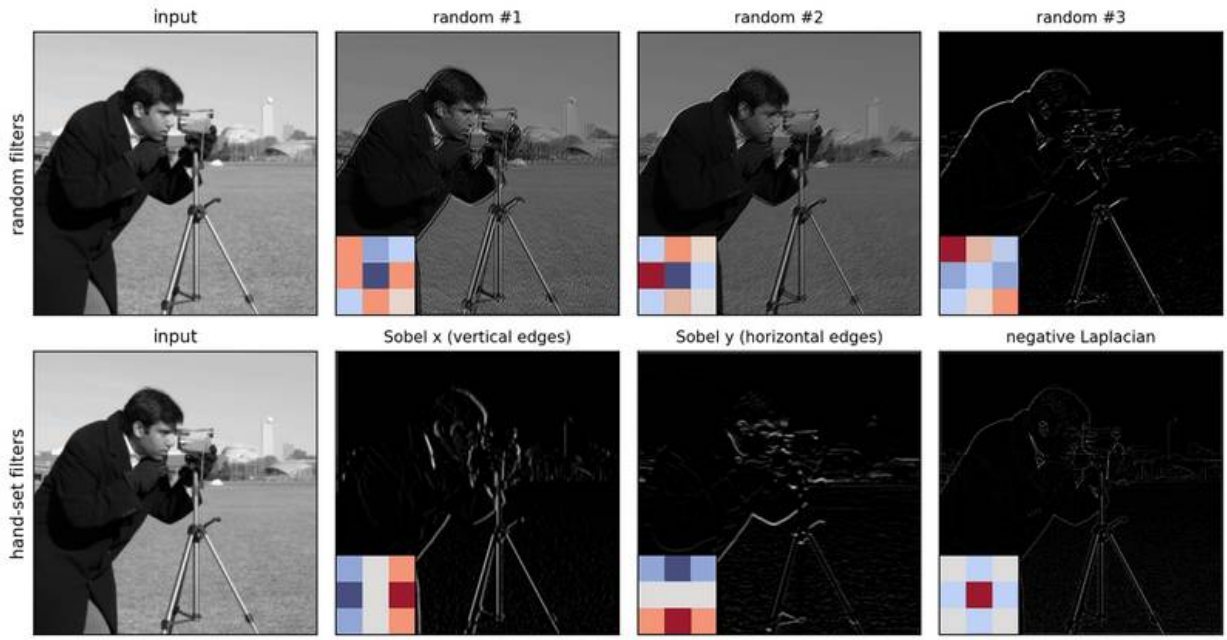


圖 *3* — 攝影師影像經 *3×3* 卷積（並經過 *ReLU*）的特徵圖。上：隨機初始權重，得到平滑與邊緣混雜的結果。下：手動設定的 *Sobel* 與 *Laplacian* 權重。訓練會把隨機濾波器推向有用的濾波器。

訓練好的第一層濾波器長什麼樣子？我們可以用 scikit-learn 內建的 8×8 手寫數字 [36]，在 CPU 上幾秒鐘就訓練完一個極小的 CNN。

```
import torch, torch.nn as nn
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

torch.manual_seed(0)
X, y = load_digits(return_X_y=True)                    # 1,797 images of 8x8 pixels
X = torch.tensor(X / 16.0, dtype=torch.float32).reshape(-1, 1, 8, 8)
Xtr, Xte, ytr, yte = train_test_split(X, torch.tensor(y), test_size=0.25, random_state=0)

net = nn.Sequential(
    nn.Conv2d(1, 8, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),     # 8x8 -> 4x4
    nn.Conv2d(8, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),    # 4x4 -> 2x2
    nn.Flatten(), nn.Linear(16 * 2 * 2, 10),
)
opt = torch.optim.Adam(net.parameters(), lr=1e-2)
for epoch in range(30):
    for i in range(0, len(Xtr), 64):
        loss = nn.functional.cross_entropy(net(Xtr[i:i + 64]), ytr[i:i + 64])
        opt.zero_grad(); loss.backward(); opt.step()
print((net(Xte).argmax(1) == yte).float().mean())       # about 0.97
filters = net[0].weight.detach()[:, 0]                   # (8, 3, 3) learned kernels
```

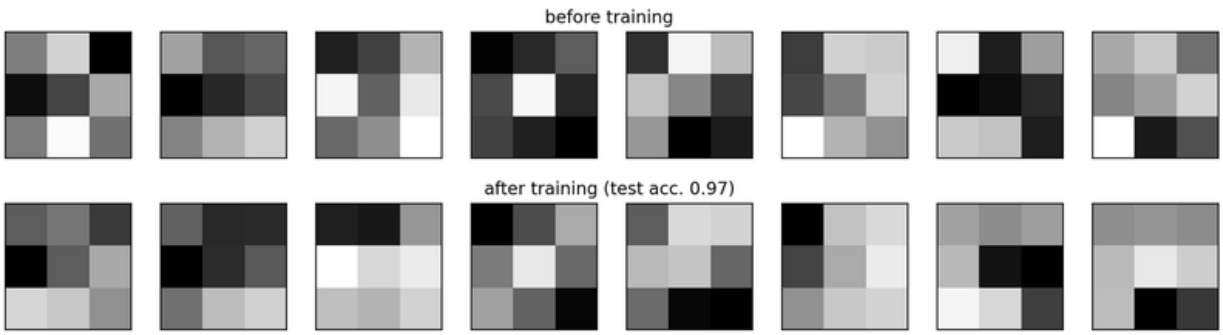


圖 *4* — 小型數字 *CNN* 的八個第一層 *3×3* 濾波器，訓練前（上）與訓練後（下）。其中幾個朝有方向性的明暗轉換移動，也就是人工設計者會挑選的類微分圖樣；沒有人告訴網路要這麼做。（*8×8* 的小輸入與 *3×3* 卷積核讓效果較不明顯；在自然影像上訓練的大型網路會更清楚地呈現這個現象。）

步幅、填補與輸出大小

白話版。填補（padding）在影像外圍加一圈邊框，讓視窗可以停在邊緣像素上。步幅（stride）讓視窗一次跳不只一個像素，因此輸出會變小。

精確版。對寬度為 W 的輸入、卷積核大小 k、每邊填補 p 個像素、步幅 s，輸出寬度為

$$W_{\text{out}} = \left\lfloor \frac{W + 2p - k}{s} \right\rfloor + 1.$$

$k = 3$、$p = 1$、$s = 1$ 時大小不變（“same” 填補）；$s = 2$ 時減半。預設是補零；但在低階網路中，常偏好鏡射或複製邊界的填補（與 IP 03 討論的邊界處理相同），因為補零會在影像邊緣產生暗色瑕疵。

池化

池化（pooling）用一個數字概括每個小視窗，通常取最大值（max pooling）或平均值（average pooling），一般使用 2×2 視窗、步幅 2。它把解析度減半、讓表示對小幅平移較不敏感，並以很低的成本擴大感受野。許多現代網路改用步幅卷積取代池化，讓網路自己學會怎麼降採樣。在分類網路的最後，**全域平均池化**（global average pooling）把 $C \times H \times W$ 的特徵圖變成給分類 head 用的 C 維向量。

感受野

白話版。一個神經元的感受野，是輸入影像中能影響它的那一塊區域。一層 3×3 看得到 3×3 個像素；疊兩層，每個輸出看得到 5×5。降採樣會讓它長得快得多。

精確版。將各層編號為 $\ell = 1, \dots, L$，卷積核大小 k_ℓ、步幅 s_ℓ。令 j_ℓ 為**跳距**（jump），也就是第 ℓ 層兩個相鄰單元在輸入影像上相距多少像素；r_ℓ 為感受野大小；輸入層 $r_0 = 1$、$j_0 = 1$。則

$$r_\ell = r_{\ell-1} + (k_\ell - 1)\, j_{\ell-1}, \qquad j_\ell = j_{\ell-1}\, s_\ell.$$

每一層增加 $k_\ell - 1$ 個輸入單元，而每個輸入單元相當於 $j_{\ell-1}$ 個像素。若處處步幅為 1，L 層 3×3 得到 $r_L = 2L + 1$：線性成長。每一次步幅 2 都讓跳距加倍，後面的層讓感受野長得兩倍快，因此隨深度大致呈指數成長。

```
def receptive_field(layers):
    r, j = 1, 1                       # receptive field and jump, measured in input pixels
    for k, s in layers:               # (kernel size, stride) of each layer
        r, j = r + (k - 1) * j, j * s
    return r

print(receptive_field([(3, 1)] * 3))                                                  # 7
print(receptive_field([(3, 1), (3, 1), (2, 2), (3, 1), (3, 1), (2, 2), (3, 1)])) # 24
```

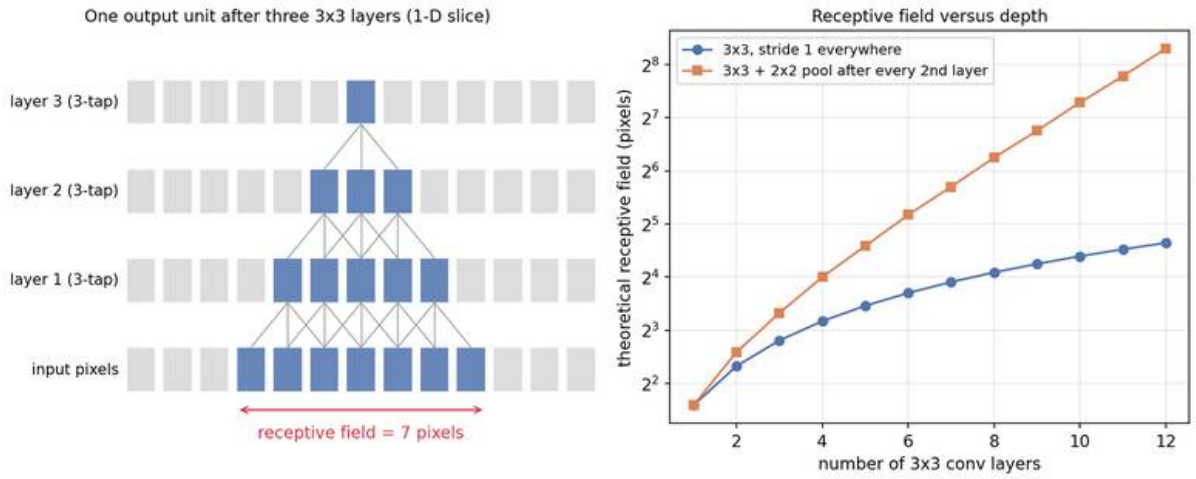


圖 *5* — 左：疊三層 *3* 抽頭的層，讓一個輸出擁有 *7* 個像素的感受野。右：理論感受野對深度的關係，比較處處步幅 *1* 與每兩層之後做一次 *2* 倍降採樣。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

實務上還有兩點要注意。第一，上面的公式給的是**理論**感受野。Luo 等人 [12] 發現，以「每個輸入像素對輸出梯度實際影響多少」來量測的**有效**感受野（effective receptive field）大致呈高斯分佈，只佔理論感受野的一小部分，中心附近的像素佔主導地位。第二，感受野決定網路**能**用到什麼：感受野比雜訊相關長度還小的去雜訊器，無法去除那種雜訊；最後一層只看得到物體一部分的分類器，也無法從上下文辨識它。

特徵階層

由於感受野隨深度變大、解析度變小，CNN 會建立一個**特徵階層**：前面的層對邊緣與色塊有反應（圖 4），中間的層對紋理與部件有反應，後面的層則對大範圍的物體層級圖樣有反應。典型的骨幹網路分成好幾個**階段**（stage），每個階段結束時做 2 倍降採樣並把通道數加倍，所以特徵從「高解析度、通道少、局部」一路變成「低解析度、通道多、語意」。高階任務讀取後面的階段；密集預測與低階任務還需要前面高解析度的階段，這正是編碼器–解碼器存在的原因（見下文）。

經典 CNN 架構

白話版。 2012 到 2016 年間，一系列一個比一個深的網路贏得了 ImageNet 大規模視覺辨識挑戰賽（ILSVRC）[6]。每一個都帶來一個至今仍在使用的設計想法。

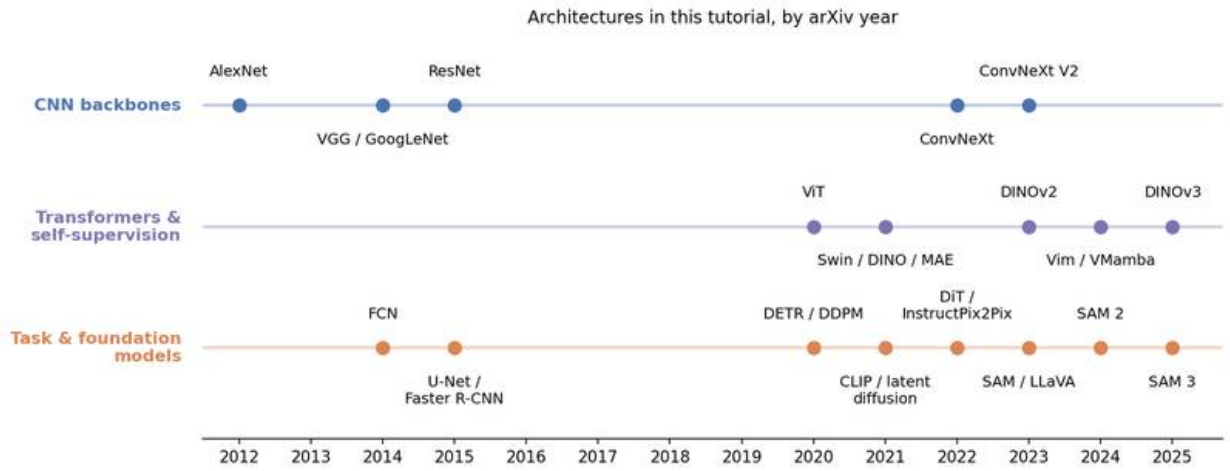


圖 *6* — 本篇提到的架構時間軸（依 *arXiv* 年份）。*LeNet-5*（*1998*）與 *1980* 年代末更早的卷積網路位於這條軸的左側之外。

LeNet-5，以及 CNN 真正的起點

原文的標題寫著「CNN（1998~）」。1998 年是 LeCun、Bottou、Bengio 與 Haffner 發表 LeNet-5 [5] 的年份，並不是 CNN 的起點：以反向傳播訓練的卷積網路，早在 1989 年就已經用來辨識手寫郵遞區號 [4]。

LeNet-5 是一個以網路形式呈現的標準圖形識別系統 [1]。C1 到 C5 是**特徵擷取器**：卷積層（C1、C3、C5）與子取樣層（S2、S4）交錯，把 32×32 的數字影像變成 120 維的特徵向量；F6 與輸出層則是**分類器**。包含特徵在內的所有東西，都以分類損失透過梯度下降訓練。端對端的想法其實早已到位，只是還需要再等十五年的資料量與運算力，才成為主流。

AlexNet（2012）

AlexNet [7] 有五個卷積層與三個全連接層（共八個可學習的層，約 6,000 萬個參數）。它的元件個別來看並不新，新的是組合：訓練速度遠快於飽和型 sigmoid 的 ReLU 激活函數、高效率的 GPU 實作、大量的資料擴增，以及全連接層中的 dropout。它以很大的差距贏得 ILSVRC 2012，這也是為什麼 2012 年通常被視為視覺深度學習時代的起點。

VGG（2014）

Simonyan 與 Zisserman [8] 只問了一個問題：如果只用 3×3 卷積，並把網路加深，會怎樣？他們最好的配置有 16 與 19 個權重層。3×3 之所以足夠，答案就在上面的感受野公式：兩層 3×3 看得到 5×5，三層看得到 7×7。輸入、輸出都是 C 個通道時，三層 3×3 需要 $3 \cdot 9C^2 = 27C^2$ 個權重，而一層 7×7 需要 $49C^2$ 個，而且前者還多插入了兩個非線性函數。VGG 整齊一致的設計，讓它多年來都是預設的預訓練特徵擷取器。

GoogLeNet / Inception（2014）

22 層深的 GoogLeNet [9] 贏得 ILSVRC 2014。它的 **Inception 模組**平行執行 1×1、3×3、5×5 卷積與一個池化分支，再把輸出串接起來，讓每個模組同時看到好幾種尺度。為了負擔得起，便宜的 1×1 卷積會先減少通道數。1×1 卷積在每個像素上混合通道而不看鄰居，相當於逐像素的全連接層，至今仍是到處可見的基本元件。

ResNet（2015），以及殘差連接為什麼有效

照理說越深越好，但實際上並非如此：He 等人 [10] 觀察到，一個普通的 56 層網路，**訓練誤差**反而比 20 層的高。這不是過擬合（overfitting），而是最佳化失敗：較深的網路理論上可以複製淺網路、再把多出來的層設成恆等映射，但梯度下降找不到那個解。

解法是**殘差區塊**（residual block，圖 7）。與其讓一疊層直接學習映射 $H(\mathbf{x})$，不如讓它學殘差 $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$，再把輸入加回來：

$$\mathbf{y} = \sigma(\mathbf{x} + F(\mathbf{x}; W)),$$

其中 $\mathbf{x}$ 是區塊輸入，F 通常是 conv–BN–ReLU–conv–BN（BN 為批次正規化，batch normalization），權重為 W，σ 為 ReLU。當通道數或解析度改變時，捷徑改用 1×1 卷積取代恆等映射。

僅線上版：批次正規化：在網路內部做標準化 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/deep-learning-for-image-processing/notes/批次正規化在網路內部做標準化/*

為什麼有效：

- **恆等映射變得容易**。如果多出來的深度沒有用處，區塊只需要把 F 推向零，這比讓一疊非線性層模仿恆等映射容易得多。把 F 最後一個 BN 的縮放參數初始化為零，每個區塊一開始就是恆等映射。
- **梯度有一條高速公路**。忽略最後的非線性，$\partial\mathbf{y}/\partial\mathbf{x} = I + \partial F/\partial\mathbf{x}$。恆等項把梯度從損失直接帶到前面的層，不必穿過每一個權重矩陣，所以不會隨深度消失。
- **集成的觀點**。把 L 個區塊展開，會得到許多長短不一的路徑之和，因此一個很深的 ResNet 在某種程度上就像許多較淺網路的集合。

```
class ResidualBlock(nn.Module):
    """Basic block: y = ReLU(x + F(x)), with F = conv-BN-ReLU-conv-BN."""
    def __init__(self, c):
        super().__init__()
        self.f = nn.Sequential(
            nn.Conv2d(c, c, 3, padding=1, bias=False), nn.BatchNorm2d(c), nn.ReLU(),
            nn.Conv2d(c, c, 3, padding=1, bias=False), nn.BatchNorm2d(c),
        )
    def forward(self, x):
        return torch.relu(x + self.f(x))

blk = ResidualBlock(16)
nn.init.zeros_(blk.f[4].weight)                 # zero the last BN scale, so F(x) = 0
x = torch.randn(2, 16, 32, 32)
```

```
print(torch.allclose(blk(x), torch.relu(x)))     # True: the block starts as the identity
 ↪  (then ReLU)
```

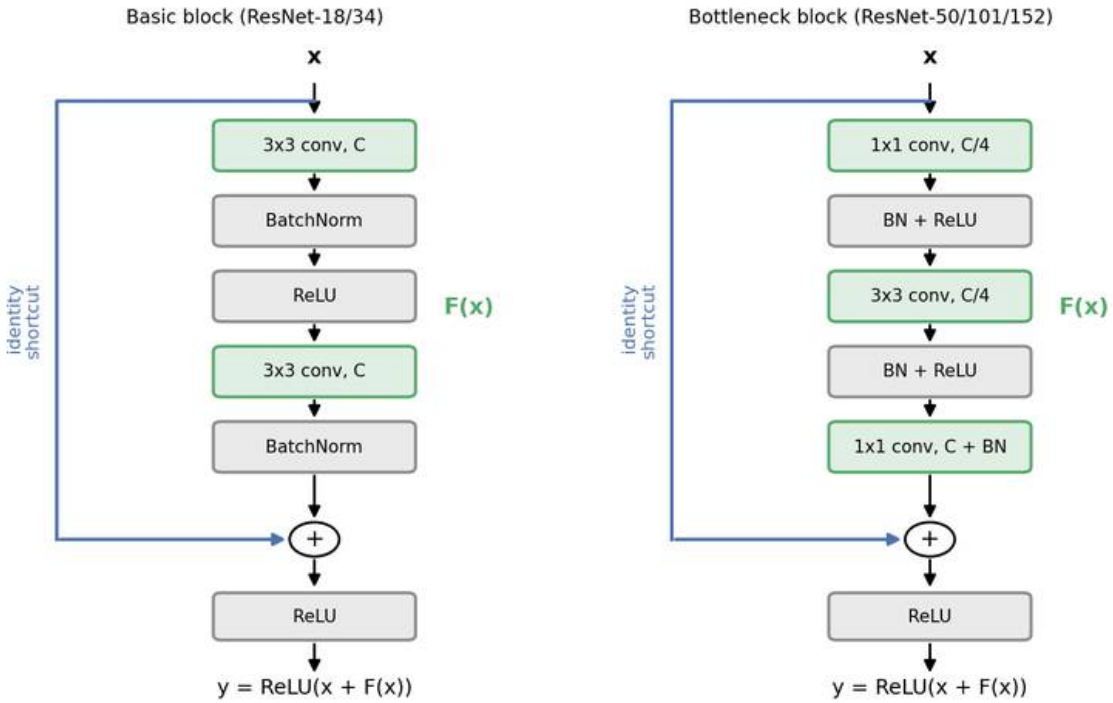


圖 *7* — 左：基本殘差區塊。右：*ResNet-50/101/152* 使用的瓶頸區塊，以 *1×1* 卷積在便宜的 *3×3* 卷積前後縮減並恢復通道數。

論文在 ImageNet 上訓練了 **18、34、50、101 與 152 層**的 ResNet，其中較深的三種使用瓶頸區塊。有名的 **1202 層**網路只出現在論文的 CIFAR-10 實驗中 [10]、[11]（32×32 的小影像）；它可以順利訓練，但過擬合，測試結果反而比 110 層的版本差。ResNet 家族贏得 ILSVRC 2015，而殘差連接如今幾乎存在於所有深度網路中，Transformer 也不例外。

用於密集預測的編碼器–解碼器

白話版。分類骨幹為了換取語意而丟掉解析度。分割或復原需要在每個像素上都給出答案，所以要再加上後半段，把解析度找回來。

精確版。 Long 等人提出的**全卷積網路**（fully convolutional network, FCN）[13] 把分類器中的全連接層換成卷積，讓網路對任意大小的輸入輸出一張粗略的類別分數圖，再以學習而來的轉置卷積上採樣，並與前面較精細層的分數融合。**U-Net** [14] 把這個想法做成對稱的形式：收縮的**編碼器**（卷積區塊與降採樣）與擴張的**解碼器**（上採樣與卷積區塊），中間以**跳接**（skip connection）把每個編碼器階段的特徵圖串接到同一解析度的解碼器階段。編碼器提供「是什麼」，跳接提供「在哪裡」。U-Net 原本是為標註很少、需要大量資料擴增的生醫影像設計的；同樣的形狀如今撐起了醫學影像分割、DLCV 09 介紹的幾乎所有復原網路，以及許多擴散模型內部的去雜訊器。

Vision Transformer

白話版。 Transformer [15] 原本是為句子設計的：它把輸入當成一串 token，讓每個 token 都能看到其他所有 token。Vision Transformer（ViT）[16] 把影像切成小方塊（patch），把每個 patch 當成一個「字」，再把這串序列送進一個沒有修改過的 Transformer 編碼器。

切塊、嵌入、注意力

精確版。對影像 $I \in \mathbb{R}^{C\times H\times W}$ 與 patch 大小 P：

1. **切塊**。把 I 切成 $N = HW/P^2$ 個不重疊的 patch，並把每一塊攤平成向量 $\mathbf{p}_i \in \mathbb{R}^{CP^2}$。對 224×224 的 RGB 影像與 $P = 16$，$N = 196$，每個向量有 768 個元素。
2. **嵌入**。把每個 patch 線性映射成 D 維 token，$\mathbf{z}_i = E\,\mathbf{p}_i$，$E \in \mathbb{R}^{D\times CP^2}$。在最前面加上一個可學習的類別 token $\mathbf{z}_{\text{cls}}$，並加上可學習的**位置嵌入**（position embedding）$\mathbf{e}_i$，因為注意力本身不知道 token 來自哪裡：$Z^{(0)} = [\mathbf{z}_{\text{cls}}; \mathbf{z}_1; \dots; \mathbf{z}_N] + [\mathbf{e}_0; \dots; \mathbf{e}_N]$。
3. **編碼**。套用 L 個 Transformer 編碼器區塊，每個區塊包含多頭自注意力（MSA）子層與 MLP 子層，兩者都搭配層正規化（LN）與殘差連接：

$$
\begin{aligned}
Z' &= Z^{(\ell-1)} + \mathrm{MSA}(\mathrm{LN}(Z^{(\ell-1)})),\\
Z^{(\ell)} &= Z' + \mathrm{MLP}(\mathrm{LN}(Z')).
\end{aligned}
$$

4. **Head**。把最後的類別 token（或所有 token 的平均）送進 MLP head，它扮演的就是圖形識別中分類器的角色 [1]。

對 token 矩陣 $Z \in \mathbb{R}^{(N+1)\times D}$，單一注意力頭計算

$$
\mathrm{Attn}(Z) = \mathrm{softmax}\!\left(\frac{QK^{\top}}{\sqrt{d}}\right)V, \qquad Q = ZW_Q,\; K = ZW_K,\; V = ZW_V,
$$

其中 $W_Q, W_K, W_V \in \mathbb{R}^{D\times d}$ 是學習而來的投影，d 是每個頭的維度。softmax 矩陣的第 i 列表示 token i 從其他每個 token 汲取多少資訊。和卷積相比，這裡的「濾波器」是依內容計算出來的，而且從第一層起就涵蓋整張影像。代價是計算量隨 token 數呈平方成長，$O(N^2 d)$，這也是為什麼使用 patch 而不是像素。

```
img = torch.randn(1, 3, 224, 224)
P, D = 16, 768
patches = img.unfold(2, P, P).unfold(3, P, P)                    # (1, 3, 14, 14, 16, 16)
patches = patches.permute(0, 2, 3, 1, 4, 5).reshape(1, -1, 3 * P * P)
print(patches.shape)                                              # (1, 196, 768)
embed = nn.Conv2d(3, D, kernel_size=P, stride=P)                  # patchify + linear embed
 ↪  in one op
tokens = embed(img).flatten(2).transpose(1, 2)                    # (1, 196, 768)
```

最後兩行說明了一個實用的等式：patch 嵌入**就是**一個卷積核大小與步幅都等於 P 的卷積。

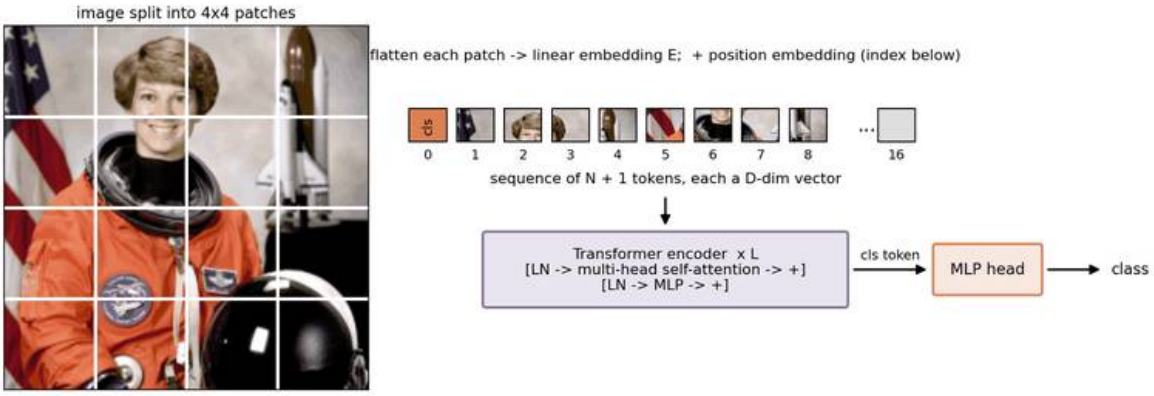


圖 *8* — 一張圖看懂 *ViT*：切塊、線性嵌入、加上位置嵌入與類別 *token*、經過 *Transformer* 編碼器，再由類別 *token* 做分類。（為了清楚起見只畫 *4×4*；*ViT-B/16* 在 *224×224* 輸入上使用 *14×14* 個 *patch*。）

歸納偏置與規模的需求

原文提到 ViT「更難收斂」。更精確的說法 [1]、[16] 與**歸納偏置**（inductive bias）有關，也就是架構在看到資料之前就內建的假設。CNN 假設局部性與權重共享，所以它已經「知道」相鄰像素要一起看，也知道同一個圖樣出現在影像中任何位置都代表同一件事。ViT 幾乎沒有任何假設：除了切 patch 之外，所有空間結構，甚至位置嵌入的二維排列，都必須自己學。

這是一種取捨，而不是缺陷。資料少的時候，CNN 的假設是寶貴的先發優勢，只在 ImageNet-1k 上從頭訓練的 ViT 表現不如同等的 ResNet。資料夠多時，這些假設反而成為限制；在非常大的資料集上預訓練的 ViT，微調後可以追平甚至超越最好的 CNN [16]。實務上的結論是：ViT 都是**預訓練後**才使用，預訓練資料可能是巨大的標註資料集，也越來越常是自監督學習。

ViT 之後的骨幹網路

白話版。 ViT 之後，Swin 加入影像結構、ConvNeXt 重新設計 CNN，DINO ／ MAE 系列則學習可重用的自監督特徵。

Swin Transformer：局部性回歸

Swin [17] 只在不重疊的局部視窗（例如 7×7 個 token）內計算注意力，讓計算量對影像大小呈線性。在交替的區塊中，視窗格線會**位移**半個視窗，讓資訊跨越視窗邊界。在階段之間，相鄰的 token 會被合併，使解析度減半。結果是一個階層式、多尺度的特徵金字塔，形狀就像 CNN 骨幹，正好是偵測與分割 head 所需要的。Swin 把 CNN 的歸納偏置，也就是局部性與階層性，放回了 Transformer 裡。

ConvNeXt：CNN 的反擊

Liu 等人 [18] 從 ResNet-50 出發，一步一步採用 Transformer 的訓練方法與設計選擇來將它現代化：「切塊式」的 stem（步幅 4 的 4×4 卷積）、大的 7×7 逐通道卷積（depthwise convolution）、倒置瓶頸的 MLP、較少的激活與正規化層，以及層正規化。得到的純 CNN，**ConvNeXt**，在分類、偵測與分割上的準確度與擴展性都與 Swin 打平。結論是：ViT 的優勢有很大一部分來自訓練方法與宏觀設計，而不是注意力本身。**ConvNeXt V2** [18] 接著加入了為卷積改寫的遮罩自編碼器預訓練（全卷積遮罩自編碼器，FCMAE）以及全域響應正規化（global response normalization）層。如今大型基礎模型多以 ViT 為骨幹，但在低階任務與邊緣裝置上，卷積仍然很常見 [1]。

自監督預訓練：DINO 與 MAE

自監督學習（self-supervised learning）從影像本身產生訓練訊號，因此不需要標註。主流的作法有兩種，而且常常被混為一談。

DINO：自蒸餾（self-distillation）[19]。學生網路 g_s 與教師網路 g_t 架構相同，各自輸出一個 K 維的機率向量。對一張影像做好幾個隨機裁切：兩個大的「全域」視圖與數個小的「局部」視圖。教師只看全域視圖，學生看所有視圖，並被訓練去符合教師的輸出：

$$\min_{\theta_s} \sum_{x \in \{x_1^g, x_2^g\}} \sum_{\substack{x' \in V \\ x' \neq x}} H(P_t(x),\, P_s(x')), \qquad H(a, b) = -a^\top \log b,$$

其中 V 是所有視圖的集合，P_t、P_s 是教師與學生的 softmax 輸出，θ_s 是學生的權重。教師不經由梯度訓練，它的權重是學生權重的指數移動平均，$\theta_t \leftarrow \lambda\theta_t + (1-\lambda)\theta_s$，$\lambda$ 接近 1；教師的輸出還會經過置中（centering）與銳化（sharpening），避免訓練塌縮成常數輸出。讓「小裁切」去符合「整張圖」，會迫使網路從局部辨識出物體。一個令人驚訝的湧現性質是：DINO ViT 的注意力圖在從未看過任何遮罩的情況下，就能分割出主要物體。

僅線上版：知識蒸餾：向老師模型的軟答案學習 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/deep-learning-for-image-processing/notes/知識蒸餾向老師模型的軟答案學習/*

MAE：遮罩預測（masked prediction）[20]。遮罩自編碼器（masked autoencoder）隨機遮住大部分的 patch（75 % 效果最好），只用 ViT 編碼看得見的 patch，再讓一個輕量解碼器重建被遮住的像素：

$$\mathcal{L}_{\text{MAE}} = \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} \|\hat{\mathbf{p}}_i - \mathbf{p}_i\|_2^2,$$

其中 $\mathcal{M}$ 是被遮住的 patch 索引集合，$\mathbf{p}_i$ 是 patch i（正規化後）的像素，$\hat{\mathbf{p}}_i$ 是重建結果。這種「遮住再預測」的遊戲，相當於語言模型中遮字預測的影像版本；相對地，DINO 是教師–學生蒸餾法 [1]。MAE 的編碼器只處理四分之一的 token，所以很省計算，微調效果也非常好；DINO 類的特徵則更適合**凍結**使用，直接接一個線性 head。

DINOv2 與 DINOv3：凍結即用的通用特徵

DINOv2 [21] 把 DINO 式的自蒸餾擴展到 10 億參數的 ViT，在一個自動**篩選**（curated）過的大型影像集合上訓練，再蒸餾成較小的模型。它凍結後的特徵，只需要一個輕量 head，就能同時勝

任影像層級任務（分類、檢索）與像素層級任務（分割、深度估計）。**DINOv3** [21] 進一步擴大資料與模型規模，並提出 **Gram anchoring**，這是一種正則化方法，能防止密集的 patch 特徵在非常長的訓練過程中退化，讓同一個凍結骨幹提供高品質的密集特徵。

這讓我們回到了圖形識別流程。「特徵擷取器＋分類器」以**預訓練凍結骨幹＋輕量 head** 的形式回來了，只是特徵擷取器現在是在極大規模、沒有標註的情況下學一次就好。

狀態空間骨幹

Vision Mamba（Vim）與 **VMamba** [22] 以選擇性狀態空間模型（selective state-space model）取代注意力，用與序列長度呈線性的成本掃描 token 序列。影像不是序列，所以這些模型會沿不只一條路徑掃描 patch 網格（Vim 是雙向的；VMamba 的二維選擇性掃描會走好幾條路線）。它們對高解析度輸入很有吸引力，但目前仍偏向研究路線，還不是預設的骨幹。

高階視覺任務

白話版。高階任務的差別在於答案要多精確地指出**位置**：整張圖一個標籤、每個物體一個方框，或是每個像素一個標籤。

任務	輸出	典型損失	典型指標	代表模型
分類	一個標籤（或機率向量）	交叉熵	top-1 / top-5 正確率	ResNet、ViT、ConvNeXt [10]、[16]、[18]
物件偵測	一組（方框、類別、分數）	分類＋方框迴歸，或集合匹配	多個 IoU 門檻下的平均精確度（AP）	Faster R-CNN、DETR [23]
語意分割	每個像素一個類別	逐像素交叉熵、Dice（重疊損失，見下文）	mean IoU	FCN、U-Net [13]、[14]
實例分割	每個物體一個遮罩與類別	偵測＋遮罩損失	mask AP	可提示式：SAM 系列 [24]

分類

為整張影像指定一個標籤。網路是骨幹＋全域池化＋線性層，以交叉熵 $\mathcal{L} = -\log \hat{p}_y$ 訓練，其中 $\hat{p}_y$ 是真實類別 y 的預測機率。在自監督學習出現之前，分類也是產生大多數骨幹網路的**預訓練任務**。DLCV 02 會深入介紹。

物件偵測

找出每個物體、它的類別，以及它的位置；位置以軸對齊的方框表示（感興趣區域，ROI，由角點座標或中心與長寬決定）[1]。有兩條設計路線可以說明這個領域 [23]。**Faster R-CNN** 是兩階段式：區域提議網路（region proposal network）從共享的 CNN 特徵中提出候選方框，第二個 head 再對每個候選分類並微調；重複的框由非極大值抑制（non-maximum suppression）移除。**DETR** 把偵測視為**集合預測**：Transformer 解碼器輸出固定數量的預測，再以預測與真實物體之間的二

分匹配定義損失，省去了人工設計的錨框與非極大值抑制。方框品質以交併比（intersection over union）衡量：

$$\mathrm{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|},$$

其中 A、B 分別是預測與真實區域。與它密切相關的 Dice 係數 $2|A \cap B|/(|A| + |B|)$，以可微分的軟性形式作為分割損失。平均精確度則在一個或多個 IoU 門檻下概括精確率–召回率曲線。細節請見 DLCV 03。

語意分割與實例分割

這裡的感興趣區域是物體確切的像素集合，而不是方框 [1]。

- **語意分割**（semantic segmentation）為每個像素標上類別。兩個重疊的人會變成同一塊「人」的區域。
- **實例分割**（instance segmentation）還要把同一類別的不同物體分開。兩個人會變成「人 1」與「人 2」。

語意分割是上面編碼器–解碼器 [13]、[14] 的本行。實例分割則再加上一個類似偵測的元件，為每個物體提出一個遮罩。

可提示的分割：SAM、SAM 2、SAM 3

分割領域如今有了**可提示的基礎模型** [24]。**SAM**（Segment Anything）接收一張影像與一個提示（點、方框或粗略遮罩），回傳被指定物體的遮罩。笨重的 ViT 影像編碼器每張影像只跑一次，輕量的提示編碼器與遮罩解碼器則每個提示跑一次，所以互動很快。它在 SA-1B 上訓練；SA-1B 包含 1,100 萬張影像上超過 10 億個遮罩，是以「模型在迴圈中」的資料引擎建立的。**SAM 2** 把這個能力擴展到影片，以串流記憶（streaming memory）把物體一幀一幀帶下去，並在新的影片資料集（SA-V）上訓練。**SAM 3** 加入了**可提示的概念分割**（promptable concept segmentation）：給它一個短名詞片語，例如「黃色校車」，或是影像範例，或兩者並用，它會找出、分割並（在影片中）追蹤**所有**符合的實例，而不只是點擊指向的那一個物體。到了 SAM 3，偵測、實例分割與追蹤之間的界線已經變得很薄；追蹤請見 DLCV 07。

低階視覺任務

白話版。低階任務輸入一張影像，回傳同一個場景經過改善或轉換的影像。輸出以逐像素的方式評判，或以看起來有多自然來評判。

一般的形式是一個反問題。劣化的觀測 $\mathbf{y}$ 被建模為

$$\mathbf{y} = \mathcal{D}(\mathbf{x}) + \mathbf{n},$$

其中 $\mathbf{x}$ 是乾淨影像，$\mathcal{D}$ 是劣化運算子（模糊、降採樣、霧、曝光不足），$\mathbf{n}$ 是雜訊，與 IP 05 的模型相同。網路 f_θ 在成對資料上訓練，最小化例如 $\|f_\theta(\mathbf{y}) - \mathbf{x}\|_1$。由於真實的成對資料很少，通常會

對乾淨影像套用一個建模好的劣化來**合成**成對資料，而合成劣化與真實劣化之間的落差，正是這個領域的核心難題。

- **影像復原**（image restoration）找回乾淨影像，又可稱為「清晰化」：去雜訊、去模糊、超解析度（小圖變大圖）、除霧（霧濛濛的影像變清楚）、去雨。架構是全解析度的編碼器–解碼器（類 U-Net），或不降採樣的殘差網路，現在也常加入視窗注意力。請見 DLCV 09。
- **影像增強**（image enhancement）讓內容更容易看清楚，但沒有唯一的「正確答案」：低光源增強（讓暗影像中的細節變清楚，某種程度上是聰明的調亮；如果是把黑夜變白天，就屬於轉譯了）與水下影像增強。請見 DLCV 09C。
- **影像對齊**（image alignment，或稱影像配準）估計把一張影像對應到另一張影像的幾何轉換或稠密位移場，是連拍融合、HDR 與醫學影像比對的前置步驟。
- **影像到影像轉換**（image-to-image translation）把影像從一個領域或風格轉到另一個：白天變黑夜、夏天變冬天、草圖變照片。換臉的「深偽」（deepfake）就是同一個想法的（被濫用的）延伸應用。自從擴散模型出現後，轉換與編輯常改由一個以輸入影像為條件的生成模型完成，見下一節。

生成與視覺–語言模型

白話版。有兩類較新的模型不太能放進舊地圖：一類從文字**創造**影像，另一類能**談論**影像。兩者都依賴大規模預訓練，以及把影像和語言連結起來。

擴散模型：從雜訊到影像

去雜訊擴散機率模型（denoising diffusion probabilistic model, DDPM）[27] 定義了一個前向過程，在 T 步內逐漸對影像 $\mathbf{x}_0$ 加入高斯雜訊。寫成封閉形式為

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I),$$

其中 $t \in \{1, \dots, T\}$ 是步數，$\bar{\alpha}_t \in (0,1)$ 是固定的排程，從接近 1 遞減到接近 0。網路 $\boldsymbol{\epsilon}_\theta$ 被訓練來預測加入的雜訊：

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}_0, t, \boldsymbol{\epsilon}} \| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \|_2^2,$$

生成時則反向執行這個過程：從純雜訊開始，一步一步去雜訊。換句話說，生成器就是**反覆套用許多次的學習式去雜訊器**，這讓生成直接和上面的復原問題連在一起。

- **潛在擴散**（latent diffusion）[28] 不在像素上、而是在預訓練自編碼器的壓縮潛在空間中進行擴散，大幅降低成本，使高解析度的文字生成影像變得可行；文字條件透過交叉注意力（cross-attention）加入。它是 Stable Diffusion 的基礎。
- **DiT**（Diffusion Transformer）[28] 把 U-Net 去雜訊器換成作用在潛在 patch 上的 Transformer，並顯示品質會隨模型計算量穩定提升。
- **InstructPix2Pix** [29] 依照文字指令（「把它變成冬天」）編輯給定的影像，讓傳統的影像到影像轉換成為條件生成的一個特例。

視覺–語言模型

視覺問答（visual question answering, VQA），也就是針對一張影像回答自由形式的問題，以前需要串起影像特徵、影像描述（captioning）與文字問答的流程 [1]。兩個進展讓它變成單一模型。

CLIP [25] 以**對比式**（contrastive）目標，在 4 億組影像–文字對上訓練影像編碼器 f 與文字編碼器 g。在一個包含 B 組配對的批次中，令正規化後的嵌入為 $\mathbf{u}_i = f(I_i)$、$\mathbf{v}_j = g(T_j)$，溫度為 τ，影像到文字的損失為

$$\mathcal{L}_{I\to T} = -\frac{1}{B}\sum_{i=1}^{B} \log \frac{\exp(\mathbf{u}_i^\top \mathbf{v}_i/\tau)}{\sum_{j=1}^{B}\exp(\mathbf{u}_i^\top \mathbf{v}_j/\tau)},$$

總損失則與對稱的文字到影像項平均。配對的組合被拉近，批次中其他所有組合被推開。由於標籤空間變成了「任何一句話」，CLIP 可以做零樣本（zero-shot）分類：把「a photo of a cat」、「a photo of a dog」……等提示嵌入，再挑出與影像最接近的那一個。

LLaVA [26] 透過一個投影層，把 CLIP 視覺編碼器接上大型語言模型（LLM）；投影層把視覺 token 映射到 LLM 的詞嵌入空間，再以關於影像的指令跟隨對話進行微調。目前大多數多模態 LLM 都沿用這種**視覺編碼器＋投影層＋ LLM** 的架構，所以 VQA、影像描述與許多辨識任務，現在都可以透過對同一個模型下提示來完成。

現代觀點

貫穿本篇的趨勢，是從「一個任務一個模型」走向「一個基礎模型＋提示或輕量 head」。以下的綜述論文，是理解這個領域如何走到今天的好地圖。

綜述。六篇綜述涵蓋了本篇談到的範圍。Li 等人 [30] 把 CNN 的設計空間逐個元件攤開。Khan 等人 [31] 依任務整理 Vision Transformer 的研究，並強調 Transformer 需要大規模預訓練或架構上的先驗，才能有好的資料效率。Jing 與 Tian [32] 以及 Balestriero 等人 [33] 涵蓋自監督學習，從早期的前置任務到今天的對比式、自蒸餾與遮罩建模家族。Awais 等人 [34] 與 Zhang 等人 [35] 綜述視覺基礎模型與視覺–語言模型，包括提示類型、遷移方法，以及評估、偏差與強健性等開放挑戰。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/deep-learning-for-image-processing/papers/現代觀點綜述導讀/*

2024–2026 年的最新進展。不引用任何基準數字，大致的局面如下：

- **骨幹網路**。以自監督或影像–文字資料預訓練的大型 ViT 是高階任務的預設選擇，凍結的 DINOv2/DINOv3 特徵 [21] 是密集任務的強基準。現代 CNN（ConvNeXt V2 [18]）仍具競爭力，在延遲、記憶體或高解析度是主要考量時更受青睞。狀態空間骨幹 [22] 是活躍的替代路線。
- **分割與追蹤**已收斂到可提示模型：影片物體用 SAM 2，開放詞彙概念用 SAM 3 [24]。
- **生成**已轉向以 Transformer 為去雜訊器的潛在擴散 [28]，編輯則轉向以指令為條件的擴散 [29]。
- **理解**越來越多交給 LLaVA 架構的多模態 LLM [26]。

- **低階視覺**仍大量依賴在合成劣化上訓練的任務專用編碼器–解碼器，並越來越常結合生成式（擴散）先驗；見 DLCV 04 與 DLCV 05。

開放問題。 (1) 資料與評估：基準測試趨於飽和，也可能與網路規模的預訓練資料重疊，難以區分泛化與記憶 [34]。(2) 真實世界的劣化：在合成雜訊或模糊上訓練的模型，遇到真實相機流程仍會失敗。(3) 保真度與幻覺：生成式先驗產生銳利、但有時是捏造的細節，這在醫學或鑑識用途中無法接受。(4) 效率：注意力對 token 數呈平方成長，基礎模型也很難在裝置端執行。(5) 大規模下的密集特徵：DINOv3 之所以需要 Gram anchoring [21]，正是因為密集特徵品質可能隨訓練規模擴大而退化。(6) 以未篩選網路資料訓練的模型，其偏差、強健性與可解釋性 [34]。

重點整理

- 先定義輸出：影像、標籤、方框、遮罩或生成內容。
- 端到端學習依任務訓練特徵；預訓練骨幹讓新輸出頭重用特徵。
- 卷積共用局部濾波器；步幅與池化改變解析度，堆疊層擴大感受野。
- 殘差路徑有助最佳化；編碼器—解碼器跳接有助保留空間細節。
- ViT 高度依賴訓練規模；Swin 加入局部性與階層結構，ConvNeXt 則更新 CNN 設計。
- DINO 透過自蒸餾學習，MAE 透過遮罩預測學習；遷移特徵仍須在目標任務評估。
- 提示與語言介面擴大模型用途，各任務仍需要自己的評估。

練習

1. **手算感受野**。一個網路依序為：7×7 卷積、步幅 2；3×3 最大池化、步幅 2；接著四個 3×3、步幅 1 的卷積。計算每一層之後的理論感受野與跳距，並用 `receptive_field` 函式驗證。

   提示

   從 $r = 1$、$j = 1$ 開始。第一個卷積之後：$r = 7$、$j = 2$。池化之後：$r = 7 + 2 \cdot 2 = 11$、$j = 4$。之後每一層 3×3 增加 $2 \cdot 4 = 8$，所以 $r = 19, 27, 35, 43$，$j = 4$。

2. **參數預算**。輸入與輸出通道都是 $C = 256$ 時，比較（忽略偏差）一層 5×5 卷積、兩層疊起來的 3×3 卷積，以及一個瓶頸結構（1×1 降到 64 通道、64 通道的 3×3、1×1 回到 256）的權重數。哪些的感受野相同？

   提示

   $25C^2 \approx 164$ 萬；$2 \cdot 9C^2 \approx 118$ 萬；瓶頸 $256 \cdot 64 + 9 \cdot 64^2 + 64 \cdot 256 \approx 7$ 萬。前兩者都看得到 5×5；瓶頸只看得到 3×3，但便宜得多，這就是 ResNet-50 以上使用它的原因。

3. **沒有捷徑時，恆等映射為什麼很難**。在網路內部，區塊的輸入通常來自前一個 ReLU，因此是非負的。證明對這樣的輸入，$F \equiv 0$ 的殘差區塊輸出恰好是 $\mathbf{y} = \mathbf{x}$。接著說明普通區塊（conv–BN–ReLU–conv–BN–ReLU）要輸出 $\mathbf{y} = \mathbf{x}$ 必須學到什麼。這對非常深的普通網路代表什麼？

提示

對 $\mathbf{x} \geq 0$，$\mathrm{ReLU}(\mathbf{x} + 0) = \mathbf{x}$。普通區塊則必須學到恰好抵消兩次卷積與兩次正規化的權重，這是一個精確又脆弱的解，梯度下降很少找得到。殘差區塊只需要 $F = 0$，而權重衰減與零初始化本來就把它往那裡推。因此很深的普通網路很難讓多出來的層「什麼都不做」，這與 ResNet 作者觀察到的退化現象一致。

4. **ViT 的 token 數與計算量**。對 512×512 的輸入，計算 patch 大小為 32、16、8 時的 token 數，以及一層全域注意力的相對成本（$\propto N^2$）。Swin 使用 7×7 視窗的視窗注意力，會如何改變這個成長趨勢？

提示

$N = (512/P)^2$：256、1024、4096 個 token；成本比為 $1 : 16 : 256$。視窗注意力每層成本為 $N \cdot 49$（每個 token 只關注 49 個 token），所以隨 N 線性成長。

5. **學來的與手動設定的濾波器**。修改數字 CNN，讓第一層凍結為四個手動設定的卷積核（兩個 Sobel、一個 Laplacian、一個平均模糊）加上四個隨機卷積核，只訓練其餘部分。與完全學習的網路比較測試正確率；再解凍隨機卷積核，觀察它們會變成什麼樣子。

提示

設定好權重之後執行 `conv.weight.requires_grad_(False)`，並只把其餘參數交給最佳化器。在這個簡單的資料集上兩者正確率相近；重點在於手動設定的微分濾波器本來就是合理的第一層，而解凍後的隨機卷積核往往會朝有方向性、類微分的圖樣移動（可對照圖 4）。

6. **選擇配方**。你有 2,000 張標註過的顯微鏡影像，需要 (a) 每張影像的品質標籤、(b) 逐像素的細胞遮罩、(c) 去雜訊後的影像。根據本篇的地圖，為每一項提出骨幹、head 與訓練策略，並說明哪一項最能受益於凍結的自監督骨幹。

提示

(a) 凍結的 DINOv2/v3，或微調的 ConvNeXt ＋線性 head。(b) U-Net（可搭配預訓練編碼器），或以點提示 SAM 再微調。(c) 全解析度的復原網路，在符合該顯微鏡雜訊特性的合成雜訊/乾淨影像對上訓練。凍結的語意特徵對 (a) 與 (b) 幫助最大；(c) 更依賴逐像素準確的統計特性，而不是語意。

參考文獻

1. Shyandram, “深度學習的數位影像處理介紹,” Medium post, 2024; updated 2026. link
2. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018. publisher page
3. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2008. publisher page
4. Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. doi:10.1162/neco.1989.1.4.541

5. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi:10.1109/5.726791
6. O. Russakovsky, J. Deng, H. Su, et al., “ImageNet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. doi:10.1007/s11263-015-0816-y ·arXiv:1409.0575
7. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *NeurIPS*, 2012. proceedings
8. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *ICLR*, 2015. arXiv:1409.1556
9. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *CVPR*, 2015. doi:10.1109/CVPR.2015.7298594 ·arXiv:1409.4842
10. K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CVPR*, 2016. doi:10.1109/CVPR.2016.90 ·arXiv:1512.03385
11. A. Krizhevsky, “Learning multiple layers of features from tiny images,” technical report, University of Toronto, 2009 (the CIFAR-10/100 datasets). dataset page
12. W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” *NeurIPS*, 2016. arXiv:1701.04128
13. J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” *CVPR*, 2015. arXiv:1411.4038
14. O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” *MICCAI*, 2015. arXiv:1505.04597
15. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *NeurIPS*, 2017. ML anthology
16. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” *ICLR*, 2021. arXiv:2010.11929
17. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin Transformer: Hierarchical vision Transformer using shifted windows,” *ICCV*, 2021. doi:10.1109/ICCV48922.2021.00986 ·arXiv:2103.14030
18. Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” *CVPR*, 2022, arXiv:2201.03545; and S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie, “ConvNeXt V2: Co-designing and scaling ConvNets with masked autoencoders,” *CVPR*, 2023, arXiv:2301.00808.
19. M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, “Emerging properties in self-supervised vision Transformers,” *ICCV*, 2021. doi:10.1109/ICCV48922.2021.00951 ·arXiv:2104.14294
20. K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, “Masked autoencoders are scalable vision learners,” *CVPR*, 2022. doi:10.1109/CVPR52688.2022.01553 ·arXiv:2111.06377
21. M. Oquab, T. Darcet, T. Moutakanni, et al., “DINOv2: Learning robust visual features without supervision,” *Transactions on Machine Learning Research*, 2024, arXiv:2304.07193; and O. Siméoni, H. V. Vo, M. Seitzer, F. Baldassarre, M. Oquab, et al., “DINOv3,” arXiv preprint, 2025, arXiv:2508.10104.
22. L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang, “Vision Mamba: Efficient visual

representation learning with bidirectional state space model," *ICML*, 2024, arXiv:2401.09417; and Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, J. Jiao, and Y. Liu, "VMamba: Visual state space model," *NeurIPS*, 2024, arXiv:2401.10166.

23. S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *NeurIPS*, 2015, arXiv:1506.01497; and N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with Transformers," *ECCV*, 2020, arXiv:2005.12872.
24. A. Kirillov, E. Mintun, N. Ravi, et al., "Segment anything," *ICCV*, 2023, arXiv:2304.02643; N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, et al., "SAM 2: Segment anything in images and videos," *ICLR*, 2025, arXiv:2408.00714; and N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, et al., "SAM 3: Segment anything with concepts," *ICLR*, 2026, arXiv:2511.16719.
25. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, et al., "Learning transferable visual models from natural language supervision," *ICML*, 2021. arXiv:2103.00020
26. H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *NeurIPS*, 2023. arXiv:2304.08485
27. J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *NeurIPS*, 2020. arXiv:2006.11239
28. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *CVPR*, 2022, arXiv:2112.10752; and W. Peebles and S. Xie, "Scalable diffusion models with Transformers," *ICCV*, 2023, arXiv:2212.09748.
29. T. Brooks, A. Holynski, and A. A. Efros, "InstructPix2Pix: Learning to follow image editing instructions," *CVPR*, 2023. arXiv:2211.09800
30. Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, 2022. doi:10.1109/TNNLS.2021.3084827 ·arXiv:2004.02806
31. S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys*, vol. 54, no. 10s, 2022. doi:10.1145/3505244 ·arXiv:2101.01169
32. L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, 2021. doi:10.1109/TPAMI.2020.2992393 ·arXiv:1902.06162
33. R. Balestriero, M. Ibrahim, V. Sobal, et al., "A cookbook of self-supervised learning," arXiv preprint, 2023. arXiv:2304.12210
34. M. Awais, M. Naseer, S. Khan, R. M. Anwer, H. Cholakkal, M. Shah, M.-H. Yang, and F. S. Khan, "Foundation models defining a new era in vision: A survey and outlook," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, 2025. doi:10.1109/TPAMI.2024.3506283 ·arXiv:2307.13721
35. J. Zhang, J. Huang, S. Jin, and S. Lu, "Vision-language models for vision tasks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, 2024. doi:10.1109/TPAMI.2024.3369699 ·arXiv:2304.00685
36. scikit-learn developers, "sklearn.datasets.load_digits" (a copy of the test set of the UCI optical recognition of handwritten digits dataset). docs
37. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ICML*, PMLR 37, pp. 448–456, 2015. PMLR ·arXiv:1502.03167
38. S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *NeurIPS*, vol. 31, 2018. proceedings ·arXiv:1805.11604

39. G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” NIPS 2014 Deep Learning Workshop, arXiv preprint, 2015. arXiv:1503.02531

DLCV 01A・深度視覺模型的訓練實務

訓練視覺網路的實務流程：資料管線與資料擴增、初始化、學習率搜尋與排程、批次大小、正則化、混合精度、微調、除錯，以及公平的實驗。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/

你將學到

- 建立先排除低成本錯誤、再進行大型訓練的流程。
- 防止資料洩漏，讓影像增強與標籤保持一致。
- 選擇學習率、排程、批次大小與梯度累積設定。
- 使用正則化、混合精度與預訓練模型微調。
- 除錯損失與梯度，再以固定預算及多個隨機種子比較模型。

先看全貌

你已經知道影像用的卷積網路與 Transformer 網路怎麼建構（DLCV 01）。本篇談的是工作的另一半：讓網路訓練得好、訓練得快，並且知道結果是不是真的。

> **用最簡單的話說**
>
> 訓練網路就像在大廚房裡做一道新菜。先嚐一小匙，確認食譜到底行不行（先過擬合一個批次）。接著確認烤箱最高能開到多熱才不會燒焦（學習率範圍測試）。不要一開始就大火猛燒，而是慢慢加熱（熱身），到最後再把火調小，免得燒焦（衰減）。而在你告訴別人新食譜比較好之前，先多做幾次，因為一頓好吃的晚餐可能只是運氣（多個隨機種子）。

為什麼要用一整篇談這個？因為訓練配方的影響常常和架構一樣大。He 等人把學習率熱身、餘弦衰減等訓練改良，加上網路的一些小改動，把 ResNet-50 的 ImageNet top-1 準確率從 75.3% 提高到 79.29% [1]；DLCV 02 介紹了類似的結果。而且訓練的失敗常常是**無聲的**：資料管線有錯的網路通常還是能訓練，只是比較差，所以不會有任何崩潰來提醒你 [2]。本篇的習慣，就是實務工作者及早抓出這些錯誤的方法。

多數工具背後的理論，本站別處已經講過，本篇直接連過去而不重複：

主題	詳細內容在
浮點數、梯度檢查、NaN 值的除錯順序	ML 01C
SGD、動量、Adam、AdamW、初始化、正規化層、梯度裁剪、基本排程	ML 02
基準、資料切分與洩漏、顯著性檢定、可重現性、延遲與部署	ML 11
批次正規化如何運作；知識蒸餾	DLCV 01 補充資料
分類配方：資料擴增、損失與最佳化器的選擇、遷移學習、校準	DLCV 02

本篇補上的是**工作流程**：做事的順序、能抓出錯誤的快速測試，以及對多數視覺模型都管用的預設值。

訓練檢查清單

白話版。大型訓練前，先確認小模型能擬合一個批次，一次只改一項因素。

下面這份清單就是整篇的濃縮。每一項都連到解釋它的小節。

1. **先定好評估方式**。在訓練任何東西之前，先定義切分單位、指標與一個簡單的基準（資料管線；ML 11）。
2. **看資料**。在完整的輸入管線（解碼、縮放、資料擴增、正規化）之後畫出一個批次，並把標註疊在上面。一分鐘內就能抓出大部分管線錯誤。
3. **從已知可靠的預設值開始**。標準架構、He 初始化、有的話就用預訓練骨幹、AdamW 或帶動量的 SGD（初始化）。
4. **過擬合一個批次**。在單一固定批次上，訓練損失必須降到幾乎為零（除錯）。
5. **找出學習率**：先做一次簡短的範圍測試，再選一個有熱身與衰減的排程（學習率）。
6. **設定批次大小**：依記憶體決定，學習率隨之調整（批次大小）。
7. **訓練完整模型，什麼都記錄**（損失、指標、學習率、梯度範數），並依驗證分數保留最佳檢查點。
8. **看到過擬合才加正則化**，一次一種工具（正則化）。
9. **等實驗正確後再加速**：用混合精度（混合精度）。
10. **公平比較：**同樣的預算、同樣的資料、多個隨機種子，每個消融實驗只改一件事；測試集最後只開一次（公平的實驗）。

執行範例

為了用真實數字示範每一步，這些程式碼訓練一個用 NumPy 寫的小型多層感知器（MLP）：64 個輸入、一個含 32 個 ReLU 單元的隱藏層、10 個輸出，以 SGD 加動量訓練，資料是 scikit-learn 的 8×8 手寫數字 [3]。它不是深度視覺模型，但本篇的每個訓練決策都適用於它，而且每次執行在 CPU 上都不到一秒。下文所有稱為「我們的小玩具實驗」的數字都來自這份程式碼；圖表腳本 `scripts/figures/dl_01a_training_practice.py` 可以重現它們。

下面的程式碼把它定義一次：分層切分的資料載入、用訓練切分的統計量做正規化、帶 He 初始化的兩層網路、反向傳播，以及能接受任何學習率排程的訓練迴圈。迴圈同時保存權重的指數移動平均，EMA 一節會用到。後面的程式碼會重複使用這些函式，所以請先執行這一段。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

X, y = load_digits(return_X_y=True)                    # 1,797 images of 8x8 pixels, 10
 ↪  classes
X_tmp, X_te, y_tmp, y_te = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
X_tr, X_va, y_tr, y_va = train_test_split(X_tmp, y_tmp, test_size=0.25, stratify=y_tmp,
 ↪  random_state=0)
mu, sd = X_tr.mean(0), X_tr.std(0) + 1e-8              # statistics of the TRAINING split
 ↪  only
X_tr, X_va, X_te = [(a - mu) / sd for a in (X_tr, X_va, X_te)]

def init(seed, sizes=(64, 32, 10)):                    # He initialization for ReLU layers
```

```
    rng = np.random.default_rng(seed)
    return [[rng.normal(0, np.sqrt(2 / m), (m, n)), np.zeros(n)]
            for m, n in zip(sizes[:-1], sizes[1:])]

def forward(P, X):
    h = [X]
    for i, (W, b) in enumerate(P):
        z = h[-1] @ W + b
        h.append(np.maximum(z, 0) if i < len(P) - 1 else z)   # ReLU; the last layer gives
        ↪  logits
    return h

def loss_and_grads(P, X, y):
    h = forward(P, X)
    z = h[-1] - h[-1].max(1, keepdims=True)             # stable log-softmax
    logp = z - np.log(np.exp(z).sum(1, keepdims=True))
    loss = -logp[np.arange(len(y)), y].mean()           # cross-entropy
    d = np.exp(logp)                                    # softmax probabilities
    d[np.arange(len(y)), y] -= 1
    d /= len(y)                                         # dLoss/dlogits
    grads = []
    for i in reversed(range(len(P))):
        grads.insert(0, [h[i].T @ d, d.sum(0)])
        if i:
            d = (d @ P[i][0].T) * (h[i] > 0)            # back through the ReLU
    return loss, grads

def evaluate(P, X, y):
    z = forward(P, X)[-1]
    z = z - z.max(1, keepdims=True)
    logp = z - np.log(np.exp(z).sum(1, keepdims=True))
    return -logp[np.arange(len(y)), y].mean(), (z.argmax(1) == y).mean()

def train(schedule, peak, epochs=15, bs=16, momentum=0.9, seed=0, ema_decay=0.99):
    P = init(seed)
    V = [[np.zeros_like(W), np.zeros_like(b)] for W, b in P]   # momentum buffers
    E = [[W.copy(), b.copy()] for W, b in P]                    # EMA copy of the weights
    rng = np.random.default_rng(seed + 1000)            # data order depends on the seed too
    T, t, log = epochs * int(np.ceil(len(y_tr) / bs)), 0, []
    for _ in range(epochs):
        order = rng.permutation(len(y_tr))
        for i in range(0, len(y_tr), bs):
            idx = order[i:i + bs]
            _, G = loss_and_grads(P, X_tr[idx], y_tr[idx])
            lr = schedule(t, T, peak)
            t += 1
            for (W, b), (gW, gb), v, e in zip(P, G, V, E):   # SGD with momentum, in place
                v[0] = momentum * v[0] + gW
                v[1] = momentum * v[1] + gb
                W -= lr * v[0]
                b -= lr * v[1]
                e[0] += (1 - ema_decay) * (W - e[0])    # e <- decay * e + (1 - decay) * w
                e[1] += (1 - ema_decay) * (b - e[1])
        log.append(evaluate(P, X_tr, y_tr) + evaluate(P, X_va, y_va))
    return P, np.array(log), E   # log columns: train loss, train acc, val loss, val acc
```

切分結果是 942 張訓練影像、315 張驗證影像與 540 張測試影像；本篇不會動到測試影像。

資料管線

白話版。檢查完整資料管線產生的張量；其數值與標籤決定學習內容及評估是否有效。

資料切分、資料洩漏與測試集

測試集必須「新」得和部署時的資料一樣。要依「將會是新的」那個單位來切分（病人、影片、相機、地點或時間區段），切分前先去除重複，所有前處理步驟都只用訓練切分來擬合，所有決策都用驗證集做，這樣測試集只會被打開一次。ML 11 講了洩漏的各種形式與如何衡量不確定性；本篇假設你已遵守這些規則。

正規化統計量

當每個輸入通道的平均值接近零、分散程度接近一時，網路訓練得最好。標準步驟是

$$x'_c = \frac{x_c - \mu_c}{\sigma_c},$$

其中 x_c 是通道 c（例如紅、綠或藍）的像素值，μ_c 與 σ_c 是通道 c 的平均值與標準差，x'_c 是網路看到的值。由此得到兩條實務規則。

- **從頭訓練：**只用訓練切分計算 μ_c 與 σ_c，把它們和模型一起存起來，在驗證、測試與部署時套用同樣的存下來的值。執行範例正是這樣做的，只是以每個像素而不是每個通道來算。
- **微調預訓練模型：**使用模型預訓練時的統計量，而不是你新資料集的。第一層的濾波器是為那個尺度的輸入學出來的。

這裡最常見的錯誤是不一致：訓練時影像縮放到 $[0, 1]$，部署時卻是原始的 0–255 值，或是一處用 RGB、另一處用 BGR。兩者都還是會產生預測，只是很差。

類別不平衡

白話版。類別不平衡可能讓整體高準確率掩蓋稀有類別失敗；可調整取樣、損失權重或分類器初始化。

Buda 等人在難度逐步提高的基準上比較了卷積網路的各種補救方法 [4]。在他們的實驗中，過取樣（oversampling）稀有類別在幾乎所有情境下都是最主要的方法，完全消除不平衡時效果最好，而且與某些傳統模型不同，它並未使網路過擬合；他們也建議，當整體準確率是關注重點時，用門檻調整來補償類別先驗 [4]。這三種手段如下：

```
import numpy as np

counts = np.array([900, 80, 20])                        # training images per class
labels = np.repeat([0, 1, 2], counts)
```

```
w = counts.sum() / (len(counts) * counts)               # option 1: weight the loss
print("loss weight per class:", np.round(w, 2))

p = 1.0 / counts[labels]                                # option 2: oversample rare classes
p /= p.sum()
rng = np.random.default_rng(0)
drawn = labels[rng.choice(len(labels), size=3000, p=p)]
print("class share in 3,000 sampled examples:", np.round(np.bincount(drawn) / 3000, 3))

prior = counts / counts.sum()                           # option 3: start the output bias at
 ↪  the prior
b0 = np.log(prior)
print("initial softmax output:", np.round(np.exp(b0) / np.exp(b0).sum(), 3))
```

```
loss weight per class: [ 0.37  4.17 16.67]
class share in 3,000 sampled examples: [0.33  0.338 0.332]
initial softmax output: [0.9  0.08 0.02]
```

選項 1 把每個範例的損失乘上 $w_k = N/(K\,n_k)$，其中 N 是訓練範例數，K 是類別數，n_k 是類別 k 的數量。選項 2 以正比於 $1/n_k$ 的機率抽取每個範例，讓每個類別出現的次數相同。選項 3 把輸出偏置設為 $b_k = \log(n_k/N)$，讓尚未訓練的網路就已經預測出類別頻率；Lin 等人在密集偵測中用了同樣的想法，讓每個位置一開始的前景機率約為 0.01 [5]（見 DLCV 03 的焦點損失）。不論選哪一種，都要回報每個類別的指標（DLCV 02）。

資料擴增作為正則化

白話版。資料擴增讓網路每次看到每張影像稍微不同的版本，所以它無法死記像素，必須學會哪些東西不會變。黃金法則：只有在新影像的正確標籤已知時，才允許做這個轉換。

這條規則決定了每個任務該用什麼擴增：

任務	通常安全	需要小心或不對
照片分類	隨機裁切與縮放、水平翻轉、色彩抖動、輕微模糊	直立場景的垂直翻轉；文字、數字與對左右敏感的類別做翻轉
偵測	翻轉、裁切、縮放、色彩抖動，對影像**與**邊界框一起做	切到物體的裁切：把框裁剪，剩下太少就丟掉
分割	對影像**與**遮罩做相同的幾何轉換	對遮罩做內插：標籤要用最近鄰縮放，絕不用雙線性
關鍵點與姿態	對影像**與**點做幾何轉換	水平翻轉也必須**對調**左右關鍵點的標籤（DLCV 05）
影像復原與低階視覺	翻轉、90° 旋轉、對輸入與目標做相同的裁切	會改變目標本身的色彩或模糊擴增
醫學與遙測	方向沒有意義時的旋轉	會改變診斷數值的強度變化

對於密集任務，關鍵字是一**致性**：套用在像素上的每個幾何變動，都必須以相同的座標套用在標籤上。下面的程式碼把一張硬幣照片連同分割遮罩與其中一枚硬幣的邊界框一起翻轉和裁切，然後檢查轉換後的框是否等於轉換後遮罩的框：

```
import numpy as np
from scipy import ndimage
from skimage import data, filters, measure, segmentation

img = data.coins()                                          # 303 x 384 grayscale photo
fg = segmentation.clear_border(img > filters.threshold_otsu(img))
lab = measure.label(ndimage.binary_fill_holes(fg))
coin = max(measure.regionprops(lab), key=lambda r: r.area)
mask = lab == coin.label                                    # segmentation target for one coin
box = tuple(int(v) for v in coin.bbox)                      # detection target (y0, x0, y1, x1)
H, W = img.shape

def hflip(img, mask, box):
    y0, x0, y1, x1 = box
    return img[:, ::-1], mask[:, ::-1], (y0, W - x1, y1, W - x0)

def crop(img, mask, box, top, left, h, w):
    y0, x0, y1, x1 = box                                    # shift, then clip to the crop
    new = (max(y0 - top, 0), max(x0 - left, 0), min(y1 - top, h), min(x1 - left, w))
    return img[top:top + h, left:left + w], mask[top:top + h, left:left + w], new

i2, m2, b2 = crop(*hflip(img, mask, box), top=90, left=0, h=180, w=240)
r, c = np.nonzero(m2)
print("original box:               ", box)
print("box after flip + crop:      ", b2)
print("box of the transformed mask:", (int(r.min()), int(c.min()), int(r.max()) + 1,
 ↪  int(c.max()) + 1))
```

```
original box:                (156, 315, 218, 380)
box after flip + crop:       (66, 4, 128, 69)
box of the transformed mask: (66, 4, 128, 69)
```

對寬度為 W 的影像做水平翻轉時，欄位從 x_0 到 x_1 的框會移到欄位 $W - x_1$ 到 $W - x_0$：左右邊界互換角色。只翻轉影像而不翻轉標籤，會給網路互相矛盾的目標（圖 1，右）。Karpathy 把這類錯誤，也就是翻轉影像卻沒翻轉標籤，列為會讓網路無聲地訓練得更差、而不是直接失敗的錯誤之一 [2]。實務上，請使用能讓影像連同遮罩與框一起轉換的函式庫，例如 Albumentations [6]，並且仍然用上面這樣的檢查驗證一次。

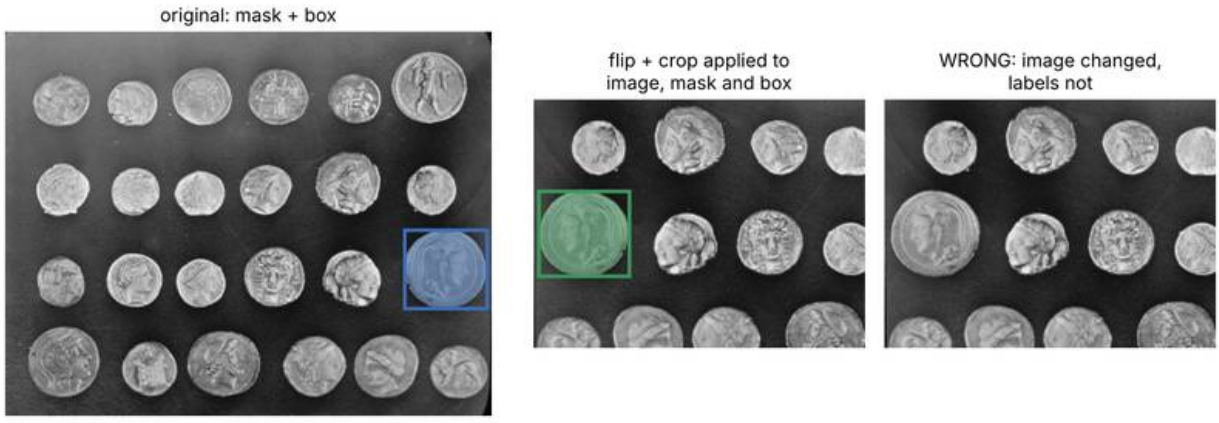


圖 *1* — 密集任務的幾何資料擴增。翻轉與裁切必須讓遮罩和框跟著像素一起移動（中）。只轉換影像（右）會留下不再對應任何物體的標註；這裡舊的框完全落在裁切範圍之外。

要多強？擴增強度是一個超參數：用驗證集調整它，並預期強擴增的模型其訓練損失會高於驗證損失。Shorten 與 Khoshgoftaar 整理了影像擴增的各個家族 [7]；DLCV 02 展示了現代分類配方中的混合式擴增。

實務上的初始化與正規化

白話版。好的開始會讓訊號在每一層都保持合理的大小，使訓練的最初幾步用來學習，而不是從錯誤的尺度中恢復。對標準架構來說，下面的預設值都經過充分驗證；除非有理由，否則就用它們。

- **權重。**ReLU 類的激活函數之前的層用 He 初始化：以變異數 $2/n_{\text{in}}$ 抽取權重，其中 n_{in} 是一個單元的輸入數，這能讓各層的激活變異數大致維持不變 [8]。偏置從零開始。ML 02 推導了這一點。
- **殘差區塊。**把每個殘差分支中最後一個批次正規化層的尺度 γ 初始化為零。這樣每個區塊一開始就是恆等映射，訊號沿著捷徑傳遞，早期訓練更容易；Goyal 等人發現這對他們所有的模型都有幫助，對大批次訓練尤其如此 [9]，He 等人也把它列入他們的技巧集 [1]。
- **輸出層。**類別不平衡時，把分類器偏置初始化為類別先驗的對數（上面的選項 3）。
- **預訓練權重。**如果有適合你輸入類型的預訓練骨幹，就從它開始。只有新的分類頭是從頭初始化的。

實務上的正規化層。批次正規化（batch normalization, BN）[10] 用當前小批次的平均值與變異數把每個通道標準化，再保存移動平均供推論使用（DLCV 01 補充資料）。兩個習慣可避免多數 BN 問題：

1. **切換模式。**訓練模式使用批次統計量並更新移動平均；評估模式使用存下來的平均。測試時忘記切換到評估模式，是個經典錯誤。
2. **注意每個裝置上的批次大小。**BN 的統計量是在單一裝置上的範例中計算的。如果每個裝置只有一兩張影像（偵測與分割中很常見），統計量就會變得雜訊很大。群組正規化（Group Normalization）在每個範例內對成組的通道做正規化，不依賴批次大小，在 BN 退化的地方仍

能維持準確；Wu 與 He 也描述了偵測中常見的做法：凍結預訓練骨幹的 BN，把它存下的統計量當作固定的線性層 [11]。

Transformer 使用在每個範例內部計算的正規化，不依賴批次；ML 02 比較了各種正規化層。

選擇學習率

白話版。學習率是每一步往下坡走的大小。太小，訓練就像在爬；太大，損失就會亂跳或爆掉。最好的學習率在訓練過程中也會變：早期大步有幫助，最後小步有幫助。

Bengio 寫道，學習率通常是單一最重要的超參數 [12]。這裡的做法是：先量一次可用的範圍，再讓排程在這個範圍中移動。

學習率範圍測試

白話版。執行一次訓練，每一步都把學習率稍微調高一點，同時觀察損失。損失不再下降、開始上升的地方，就是學習率變得太大了。

Smith 提出這個**學習率範圍測試**，同時提出週期性學習率：短時間訓練，同時讓學習率在低值與高值之間增加，並讀出網路仍然能學習的範圍 [13]。在他後來的指引中，損失最低（或準確率最高）處的學習率是可用的最大值，適合作為先升後降的排程的最大值，而固定的學習率則必須更小，否則網路根本不會開始收斂；對於單一週期的排程，他建議最小值比最大值低 10 到 20 倍 [14]。

Smith 從驗證損失或準確率讀曲線；我們的版本改用每個批次的訓練損失，不需任何額外成本。它在最多 200 步內讓學習率從 10^{-4} 指數增加到 10，使每個數量級獲得相同的時間，並用指數移動平均平滑有雜訊的批次損失：

$$\bar{\ell}_t = \beta\,\bar{\ell}_{t-1} + (1-\beta)\,\ell_t, \qquad \hat{\ell}_t = \frac{\bar{\ell}_t}{1-\beta^{t+1}},$$

其中 ℓ_t 是第 t 步那個批次的損失，$\bar{\ell}_t$ 是移動平均（從 $\bar{\ell}_{-1}=0$ 開始），$\beta=0.9$ 是平滑係數，$\hat{\ell}_t$ 是畫在圖上的偏差校正平均；這個校正去除了朝零起始值被拉低的影響。當平滑後的損失超過其最小值的四倍時，測試就停止。下面的程式碼重複使用執行範例中的 `init` 與 `loss_and_grads`。

```
def lr_range_test(lo=1e-4, hi=10.0, steps=200, bs=16, momentum=0.9, seed=0):
    P = init(seed)
    V = [[np.zeros_like(W), np.zeros_like(b)] for W, b in P]
    rng = np.random.default_rng(seed)
    lrs, smooth, avg = [], [], 0.0
    for t in range(steps):
        lr = lo * (hi / lo) ** (t / (steps - 1))        # exponential sweep
        idx = rng.choice(len(y_tr), bs, replace=False)
        loss, G = loss_and_grads(P, X_tr[idx], y_tr[idx])
        avg = 0.9 * avg + 0.1 * loss                    # smooth the noisy batch loss
        lrs.append(lr)
        smooth.append(avg / (1 - 0.9 ** (t + 1)))       # bias-corrected average
        if not np.isfinite(loss) or smooth[-1] > 4 * min(smooth):
```

```
            break                                       # the loss has blown up: stop
        for (W, b), (gW, gb), v in zip(P, G, V):
            v[0] = momentum * v[0] + gW
            v[1] = momentum * v[1] + gb
            W -= lr * v[0]
            b -= lr * v[1]
    return np.array(lrs), np.array(smooth)

lrs, smooth = lr_range_test()
best = lrs[smooth.argmin()]
print(f"lowest smoothed loss {smooth.min():.3f} at lr {best:.3f}; stopped at lr
 ↪ {lrs[-1]:.3f}")
print(f"suggested peak for a one-cycle or warmup schedule: about {best:.2f}")
print(f"suggested constant learning rate: about {best / 10:.3f}")
```

```
lowest smoothed loss 0.270 at lr 0.138; stopped at lr 0.392
suggested peak for a one-cycle or warmup schedule: about 0.14
suggested constant learning rate: about 0.014
```

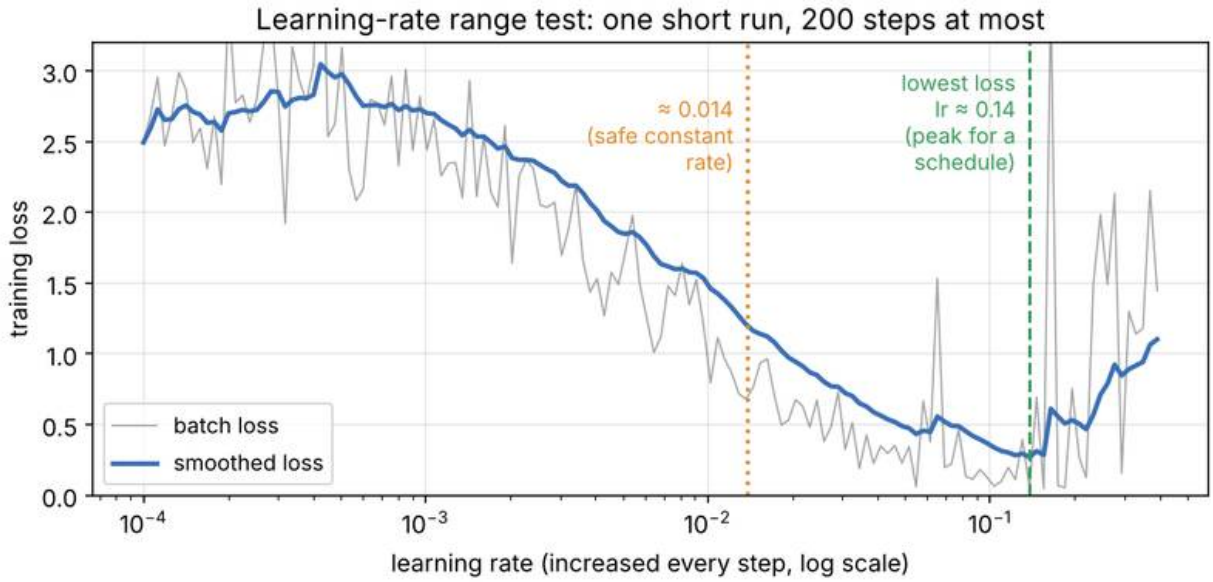


圖 *2* — 在小玩具 *MLP* 上的學習率範圍測試（我們的實驗）。學習率大到足以產生進展後，損失開始下降，在 *0.14* 附近觸底，然後上升；測試在 *0.39* 時停止。

由左到右讀圖 2：先是小到什麼都改變不了，然後是損失陡降的有用範圍，最後是太大的步伐。程式碼取最小值的十分之一作為固定學習率；這個係數是我們的經驗法則，不是 [14] 中的數值。一次範圍測試只需要幾百步，所以每當你更改模型、批次大小、最佳化器或權重衰減時，都應該重跑。把結果當成一個範圍，而不是精確的值：曲線有雜訊，也會隨隨機種子而變。

熱身（warmup）

白話版。一開始用很小的學習率，在最初的幾百或幾千步內把它升到目標值。一開始權重是隨機的，最佳化器的移動統計量也是空的，所以全尺寸的一步可能把模型丟進一個再也出不來的糟糕區域。

使用 W 步的線性熱身時，第 $t < W$ 步的學習率是

$$\eta_t = \eta_{\text{peak}} \, \frac{t+1}{W},$$

其中 η_{peak} 是熱身結束時達到的目標學習率。Goyal 等人說明了為什麼形狀很重要：在大批次的 ImageNet 訓練中，前幾個 epoch 用固定的低學習率、然後跳到目標值，會讓訓練誤差出現尖峰，而逐漸升高則避免了這個尖峰 [9]。熱身對大批次、大學習率、自適應最佳化器與 Transformer 最重要，在微調中也是便宜的保險。Goyal 等人報告，即使是固定低學習率的熱身，在原型設計偵測與分割模型（把預訓練層與新層一起微調）時也特別有幫助 [9]。我們的小玩具在前 5% 的步數內熱身。

排程：階梯式、餘弦與 one-cycle

白話版。熱身之後，學習率應該下降。怎麼降其實不如「有降」重要：大步用來探索，最後的小步讓模型穩定在一個好的區域。

ML 02 介紹了階梯式衰減、餘弦退火與熱身；本節補上 one-cycle，並在同一個問題上比較全部。以 t 表示當前步數，T 表示總步數，η_{peak} 表示最大學習率。圖 3 中的四種排程是：

- **固定：** $\eta_t = \eta_{\text{peak}}$。簡單，但最後幾步和最初幾步一樣大。
- **階梯式衰減：** 在固定的里程碑處（例如訓練進度的 50% 與 75%）把學習率乘上係數 γ（通常是 0.1）。這是用 SGD 訓練 CNN 的經典選擇。
- **熱身 + 餘弦：** 經過 W 步熱身後，$\eta_t = \tfrac{1}{2}\eta_{\text{peak}}(1 + \cos(\pi q))$，其中進度 $q = (t - W)/(T - W)$ 從 0 走到 1，所以學習率從 η_{peak} 平滑滑到零。餘弦形狀來自 SGDR，它還會週期性地重啟 [15]；He 等人在 ImageNet 分類中採用單純的餘弦衰減，準確率比他們的階梯式衰減基準更好 [1]。
- **One-cycle：** 一次長的上升，從低學習率升到 η_{peak}，然後降到遠低於起始學習率的值。Smith 與 Topin 用這種「1cycle」策略搭配很大的最大學習率，把某些網路訓練得比標準排程快得多，他們稱之為超收斂（super-convergence）[16]。我們的版本遵循 PyTorch `OneCycleLR` 的預設值：前 30% 的步數從 $\eta_{\text{peak}}/25$ 開始做餘弦形狀的上升，然後做餘弦形狀的下降，降到比起點低 10^4 倍 [17]。該實作還會在學習率上升時降低動量，這是 Smith 的建議 [14]；我們的小玩具則讓動量保持不變。

```
import numpy as np

def constant(t, T, peak):
    return peak

def step_decay(t, T, peak, milestones=(0.5, 0.75), factor=0.1):
    return peak * factor ** sum(t >= m * T for m in milestones)
```

```
def warmup_cosine(t, T, peak, warmup=0.05):
    W = max(1, int(warmup * T))
    if t < W:
        return peak * (t + 1) / W                       # linear warmup
    q = (t - W) / max(1, T - W)                         # progress after warmup, 0 -> 1
    return 0.5 * peak * (1 + np.cos(np.pi * q))         # cosine decay to 0

def one_cycle(t, T, peak, pct=0.3, div=25, final_div=1e4):
    up, lo = max(1, int(pct * T)), peak / div
    if t < up:                                          # rise from peak/div to peak
        return lo + (peak - lo) * 0.5 * (1 - np.cos(np.pi * t / up))
    q = (t - up) / max(1, T - up)                       # fall far below the start
    end = lo / final_div
    return end + (peak - end) * 0.5 * (1 + np.cos(np.pi * q))

T = 900
for f in (constant, step_decay, warmup_cosine, one_cycle):
    print(f"{f.__name__:13s}", [f"{f(t, T, 0.1):.2g}" for t in (0, 20, 270, 450, 700, 899)])
```

```
constant      ['0.1', '0.1', '0.1', '0.1', '0.1', '0.1']
step_decay    ['0.1', '0.1', '0.1', '0.01', '0.001', '0.001']
warmup_cosine ['0.0022', '0.047', '0.084', '0.054', '0.013', '3.4e-07']
one_cycle     ['0.004', '0.0053', '0.1', '0.081', '0.023', '1e-06']
```

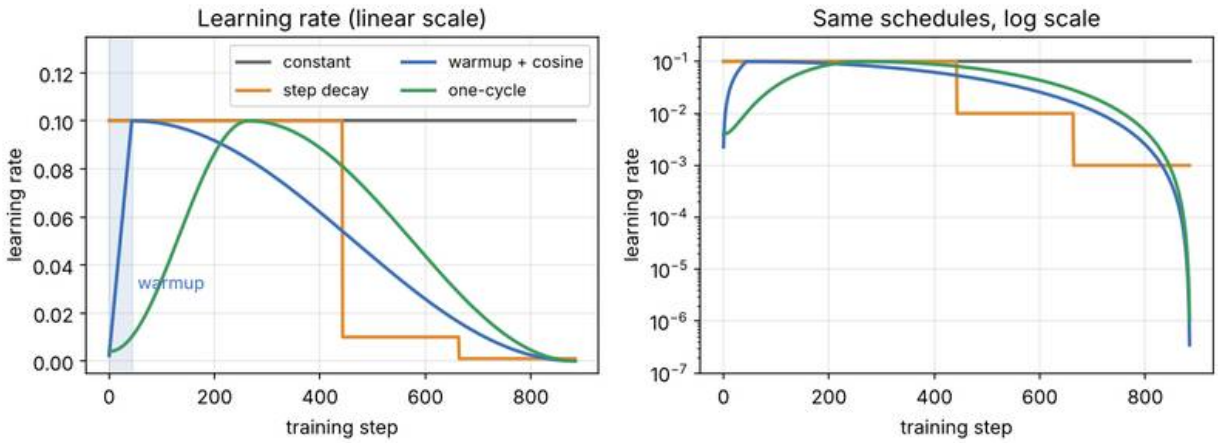


圖 3 — 本篇使用的四種排程，峰值為 *0.1*，共 *885* 步（小玩具的 *15* 個 *epoch*）。對數尺度的視圖（右）顯示餘弦與 *one-cycle* 在最後降得有多低。

僅線上版：帶重啟的餘弦退火 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/notes/帶重啟的餘弦退火/*

排程在小玩具上的表現

下面的迴圈用每一種排程（峰值 0.1）各訓練執行範例一次，再用固定學習率 0.01 訓練一次。它使用執行範例中的 `train` 與上面剛定義的排程函式。

```
for f in (constant, step_decay, warmup_cosine, one_cycle):
    _, log, _ = train(f, peak=0.1)
    print(f"{f.__name__:13s} peak 0.1   final train loss {log[-1, 0]:8.3f}   val acc {log[-1,
     ↪  3]:.3f}")
_, log, _ = train(constant, peak=0.01)
print(f"{'constant':13s} peak 0.01  final train loss {log[-1, 0]:8.3f}   val acc {log[-1,
 ↪  3]:.3f}")
```

```
constant      peak 0.1   final train loss  995.468   val acc 0.917
step_decay    peak 0.1   final train loss    0.000   val acc 0.940
warmup_cosine peak 0.1   final train loss    0.000   val acc 0.971
one_cycle     peak 0.1   final train loss    0.000   val acc 0.956
constant      peak 0.01  final train loss    0.021   val acc 0.959
```

圖 4 逐 epoch 顯示各次實驗。這個小玩具得出三點結論，與上面的建議相符：

1. **接近邊緣的峰值需要排程**。在固定的 0.1（接近範圍測試的最小值）下，損失爆炸，驗證準確率在 81% 到 92% 之間擺盪。同一個峰值放進先升後降的排程中就沒問題（這次實驗中，熱身 + 餘弦達到 97.1%，one-cycle 達到 95.6%）。
2. **衰減能救回來，但太晚**。階梯式衰減把一半的訓練花在不穩定的學習率上；學習率在第 8 個 epoch 降下來之後，損失才恢復，驗證準確率達到 94.0%。
3. **固定學習率夠小就能用**。在 0.01（範圍測試建議的峰值的十分之一）下，固定排程訓練得很平穩（95.9%），只是在同樣的預算內走得不夠遠：它的訓練損失還有 0.021。

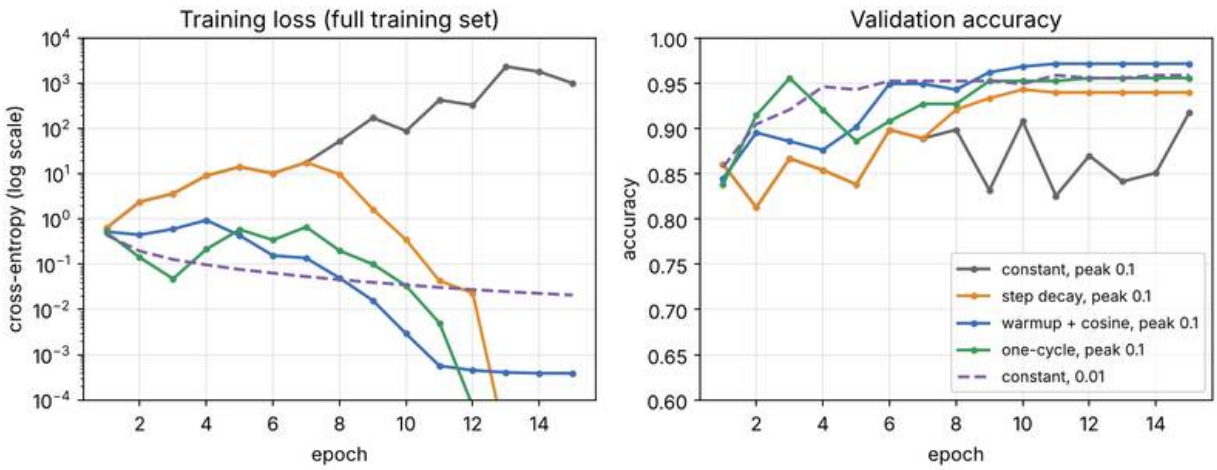


圖 *4* — 圖 *3* 的各種排程在小玩具 *MLP* 上的表現，隨機種子 *0*（我們的實驗）。峰值為 *0.1* 時，固定學習率發散，階梯式衰減在第一次下降後恢復，熱身 + 餘弦與 *one-cycle* 則訓練穩定。虛線是固定學習率 *0.01*。

不要從這張圖解讀餘弦與 one-cycle 之間的名次：這只是一個隨機種子，而下面的種子研究顯示它們的順序會隨種子而翻轉。可靠的結論是第一點。

該選哪個排程。熱身後接餘弦衰減是安全的預設；重現舊論文時，階梯式衰減沒問題；one-cycle 適合短預算；事先不知道訓練長度時，固定學習率加一小段最後的衰減會有幫助（現代觀點）。排程是以步定義的，所以當 epoch 數、批次大小或資料集大小改變時，要重新計算 T。

批次大小與梯度累積

白話版。較大的批次給出雜訊較少的梯度，所以每一步可以走得更大。如果批次加倍，通常可以把學習率也加倍，但有個限度。（ML 02 定義了批次、iteration 與 epoch。）

Goyal 等人在 ImageNet [19] 上訓練 ResNet-50 [18]，使用最多 8,192 張影像的小批次而不損失準確率，靠的是兩條規則 [9]：

- **線性縮放規則：**當小批次大小乘以 k 時，把學習率也乘以 k，其他超參數（例如權重衰減）保持不變。He 等人以 $\eta = 0.1 \times B/256$ 的形式陳述同一條規則，適用於批次大小為 B、用 SGD 訓練的 ResNet-50 [1]。
- **漸進式熱身：**在最初 5 個 epoch 內，把學習率從小批次的值升到縮放後的值。

這條規則是一個近似。它假設一次大步所取代的 k 個小步之間，梯度變化不大；這在訓練早期（所以需要熱身）以及超過某個批次大小之後都不成立；在他們的 ImageNet 實驗中，這個極限約在 8,000 張影像 [9]。對於自適應最佳化器，這個關係沒有那麼簡單，所以批次大小大幅變動後，請重新執行範圍測試。

僅線上版：*Accurate, Large Minibatch SGD*（*Goyal* 等人）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/papers/accurate-large-minibatch-sgdgoyal-等人/*

梯度累積。當你想要的批次放不進記憶體時，就跑幾個較小的微批次（micro-batch），把它們的梯度加起來，只做一次最佳化器更新。當損失是對範例取平均，且 k 個微批次大小相等時，

$$\nabla_\theta L_{\text{batch}} = \frac{1}{k} \sum_{j=1}^{k} \nabla_\theta L_j,$$

其中 L_{batch} 是全部 $k \cdot m$ 個範例的平均損失，L_j 是含 m 個範例的微批次 j 的平均損失，θ 是參數。下面的程式碼在小玩具上檢查這一點（它重複使用執行範例中的 `init` 與 `loss_and_grads`）：

```
P = init(0)
xb, yb = X_tr[:32], y_tr[:32]
_, full = loss_and_grads(P, xb, yb)                    # one batch of 32

acc = [[np.zeros_like(W), np.zeros_like(b)] for W, b in P]
k = 4                                                  # 4 micro-batches of 8
for j in range(k):
    _, G = loss_and_grads(P, xb[8 * j:8 * j + 8], yb[8 * j:8 * j + 8])
    for a, (gW, gb) in zip(acc, G):
        a[0] += gW / k                                 # divide by k: mean of means
        a[1] += gb / k
diff = max(np.abs(a[0] - g[0]).max() for a, g in zip(acc, full))
print(f"largest difference, accumulated vs full batch: {diff:.1e}")
```

```
largest difference, accumulated vs full batch: 1.1e-16
```

差異是浮點捨入造成的：更新其實是相同的。在真實模型中，有兩件事會破壞這個等價性。批次正規化是對每個微批次計算統計量，所以累積並不會讓 BN 得到更大的批次。另外，如果損失是對每張影像數量不定的項目取平均（分割中的有效像素、偵測中的框、圖說中的 token），累積後的總和要除以總數，而不是除以 k。

僅線上版：損失不是單純平均時的梯度累積 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/notes/損失不是單純平均時的梯度累積/*

實務上的正則化

白話版。正則化工具讓死記更困難，使網路學到能泛化的規律。它們都朝同一個方向拉，所以要一次加一種，而且只有在驗證曲線顯示過擬合時才加。

Smith 與 Topin 強調了一個容易忽略的重點：大的學習率本身也有正則化的效果，所以較高的峰值需要較少其他種類的正則化，而且所有形式（大學習率、小批次、權重衰減、dropout）必須一起平衡 [16]。當你更改排程時，要重新調整權重衰減與資料擴增，反之亦然。

權重衰減。權重衰減在每一步把每個權重稍微縮小一點。對單純的 SGD 來說，這等於在損失中加入 L2 懲罰；對 Adam [20] 這類自適應最佳化器就不是了，因為懲罰項的梯度會被逐參數重新縮放。Loshchilov 與 Hutter 的 AdamW 把衰減從梯度步驟中解耦 [21]；ML 02 推導了其中的差異。兩條實務規則：

- 在 PyTorch 中，`torch.optim.AdamW` 套用的是解耦衰減，而 `torch.optim.Adam` 的 `weight_decay` 參數則是把 L2 項加進梯度，除非設定 `decoupled_weight_decay=True` [22]。對 Transformer 與現代 CNN 配方，請使用解耦衰減。
- 不要對偏置，以及正規化層的尺度與位移做衰減。He 等人把這種「不對偏置衰減」列為他們的改良之一 [1]；DLCV 02 以程式碼展示了這兩個參數群組。

Dropout 與隨機深度。Dropout 在訓練時把隨機的激活值歸零 [23]；隨機深度（stochastic depth）則隨機跳過整個殘差區塊，跳過機率隨深度增加 [24]。Dropout 常見於分類頭與 Transformer 區塊，隨機深度則常見於深層殘差網路與視覺 Transformer。兩者在訓練模式與評估模式下的行為不同。

提前停止與檢查點選擇。定期在驗證集上評估，保留最佳檢查點，並在分數一段時間沒有改進時停止（這段時間稱為「耐心」）。在 Prechelt 對多層感知器的研究中，較慢的停止準則使泛化略有改善，但訓練成本高出許多 [25]。Tuning Playbook 建議定期儲存檢查點，並事後回溯挑選最佳的一個 [26]。

權重的 EMA

白話版。在權重本身旁邊，再保留一份緩慢移動的權重平均，並評估這個平均。最後幾步的雜訊會互相抵消，就像把同一個場景的好幾張照片平均起來能去除雜訊一樣。

DLCV 02 把 EMA 列為現代分類配方的一部分；本節談的是如何選擇它的視窗。每次最佳化器更新後，更新

$$\bar{\theta} \leftarrow \mu\,\bar{\theta} + (1-\mu)\,\theta,$$

其中 θ 是目前的權重，$\bar{\theta}$ 是其指數移動平均（EMA），$\mu \in [0,1)$ 是衰減率。這個平均實際上涵蓋大約最近 $1/(1-\mu)$ 步：$\mu = 0.99$ 時約 100 步，$\mu = 0.999$ 時約 1,000 步。對隨機梯度下降的迭代值取平均，是最佳化領域的老想法 [27]，而隨機權重平均（stochastic weight averaging, SWA）把等權重的版本帶進了深度學習 [28]。Morales-Brotons 等人系統性地研究了權重的 EMA：EMA 模型的泛化比最後一次迭代好，所需的學習率衰減也較少，而除非重新計算批次正規化的統計量，否則 BN 會限制平均視窗能有多長 [29]。

視窗必須比整個訓練短。我們的小玩具把這一點具體化（它重複使用上面的 `train`、`constant` 與 `evaluate`）：

```
for decay in (0.99, 0.999):
    raw, avg = [], []
    for s in range(5):
        P, log, E = train(constant, peak=0.05, seed=s, ema_decay=decay)
        raw.append(evaluate(P, X_va, y_va)[1])
        avg.append(evaluate(E, X_va, y_va)[1])
    print(f"decay {decay}: window ~{1 / (1 - decay):.0f} steps   "
          f"val acc raw {np.mean(raw):.3f}   EMA {np.mean(avg):.3f}")
print("steps in the whole run:", 15 * int(np.ceil(len(y_tr) / 16)))
```

```
decay 0.99: window ~100 steps   val acc raw 0.963   EMA 0.963
decay 0.999: window ~1000 steps   val acc raw 0.963   EMA 0.945
steps in the whole run: 885
```

在這個小而簡單的問題上，$\mu = 0.99$ 時 EMA 沒有帶來好處，而 $\mu = 0.999$ 時它反而更差：約 1,000 步的視窗比整個 885 步的訓練還長，所以平均仍帶有糟糕的早期迭代的權重。EMA 在更新帶有雜訊的長時間訓練中才有回報，實務上也是這樣用的。

僅線上版：權重的 *EMA*：如何選擇衰減率 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/notes/權重的-ema 如何選擇衰減率/*

混合精度與損失縮放

白話版。用 16 位元數字做大部分運算，可以把記憶體流量減半並使用快速的硬體單元，但 16 位元放不下所有東西。要把少數敏感的量保留在 32 位元，並縮放損失，讓很小的梯度不會被捨入成零。

Micikevicius 等人用兩項技術讓 float16 訓練可行：以 float32 保存權重的主副本來累積更新，以及損失縮放，也就是在反向傳播前把損失乘上係數 S，更新前再把梯度除以 S [30]。這兩個問題都很容易重現：

```
import numpy as np
```

```
g = np.float32(2e-8)                                    # a small, real gradient value
print("float16 gradient:          ", np.float16(g))     # underflows to zero
S = np.float32(1024)                                    # loss scale
g16 = np.float16(g * S)                                 # scaled: representable in float16
print("scaled, then unscaled:     ", np.float32(g16) / S)
print("float16 weight 1 + 1e-4:   ", np.float16(1) + np.float16(1e-4))   # update lost
print("float32 weight 1 + 1e-4:   ", np.float32(1) + np.float32(1e-4))
```

```
float16 gradient:           0.0
scaled, then unscaled:      2.0023435e-08
float16 weight 1 + 1e-4:    1.0
float32 weight 1 + 1e-4:    1.0001
```

第一行是**下溢**（underflow）：2×10^{-8} 小於 float16 能表示的最小正值，所以變成零，而乘上 1,024 就把它移回可表示的範圍。第三行是**更新遺失**：在 1.0 附近，float16 的數字間隔太大，無法反映 10^{-4} 的變化，這就是權重本身要留在 float32 的原因。

在 PyTorch 中，自動混合精度結合了 `torch.autocast`（讓每個運算以合適的精度執行）與 `torch.amp.GradScaler`（執行動態損失縮放）；其在 CPU 上的 bfloat16 範例只用 autocast [31]。bfloat16 保留了 float32 的 8 個指數位元，也就保留了它的範圍，所以很少需要損失縮放（ML 01C）。先在 float32 下把實驗做對，再開啟混合精度並檢查損失曲線幾乎不變；如果 NaN 只出現在混合精度下，就去找在 16 位元下溢位的運算。ML 01C 詳細解釋了這些格式。

遷移學習與微調策略

白話版。預訓練網路已經認得邊緣、紋理與物體的部位。微調要教它你的任務，卻不破壞那些知識：新的部分快速改變，舊的部分輕輕調整。

DLCV 02 定義了線性探測與完整微調；本節談的是如何在兩者之間選擇，以及如何讓微調更溫和。Yosinski 等人測量了每一層特徵的通用程度：早期層學到的特徵可以跨資料集遷移，後期層則更偏向特定任務，來源與目標任務差距越大，可遷移性越低，不過即使任務相距很遠，遷移來的特徵仍然優於隨機特徵 [32]。這就是下列策略背後的原因，由最便宜到最有彈性排列：

策略	訓練什麼	適用時機
線性探測	只訓練凍結特徵上的新線性分類頭	資料很少；特徵對這個任務已經夠好；也用來衡量特徵品質
部分微調	分類頭與最後幾個階段	資料量中等；目標接近預訓練的領域
完整微調	全部，骨幹用較小的學習率	資料夠多，或領域差距大
先線性探測再微調（LP-FT）	先只訓練分類頭，再訓練全部	在意對分布偏移的穩健性時
參數高效微調	小型附加模組，例如低秩更新（LoRA [33]）	非常大的模型；許多任務共用一個骨幹

有三個工具能讓完整微調更溫和：

- **逐層學習率**。給每個層群組自己的學習率，越早的層越小。Howard 與 Ruder 把這稱為判別式微調（discriminative fine-tuning），並與逐步解凍（gradual unfreezing：先訓練最後一層，之

後每個 epoch 再多解凍一個層群組）一起提出 [34]。在視覺配方中，骨幹的學習率常比分類頭小 10 到 100 倍（DLCV 02）。

- **LP-FT**。Kumar 等人指出，用隨機初始化的分類頭做完整微調，可能會扭曲良好的預訓練特徵：在擬合隨機分類頭的過程中，骨幹對類似訓練資料的輸入改變很大，對看起來不同的輸入則改變小得多。先訓練線性分類頭，再微調全部（LP-FT），在他們的實驗中同時保留了微調在分布內的增益，以及線性探測對分布外的穩健性 [35]。
- **熱身與 BN**。短暫的熱身能保護預訓練權重，不被新分類頭一開始的大梯度破壞；而在每個裝置的批次很小時，骨幹的 BN 層常被凍結（上文）。

僅線上版：逐層學習率與逐步解凍 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/notes/逐層學習率與逐步解凍/*

僅線上版：*Fine-Tuning can Distort Pretrained Features*（*Kumar* 等人）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/training-practice/papers/fine-tuning-can-distort-pretrained-featureskumar-等人/*

除錯一次訓練

白話版。多數訓練錯誤不會崩潰；它們只是讓模型稍微變差。除錯就是執行正確的管線一定能通過的便宜測試，並依能隔離原因的順序進行（圖 5）。

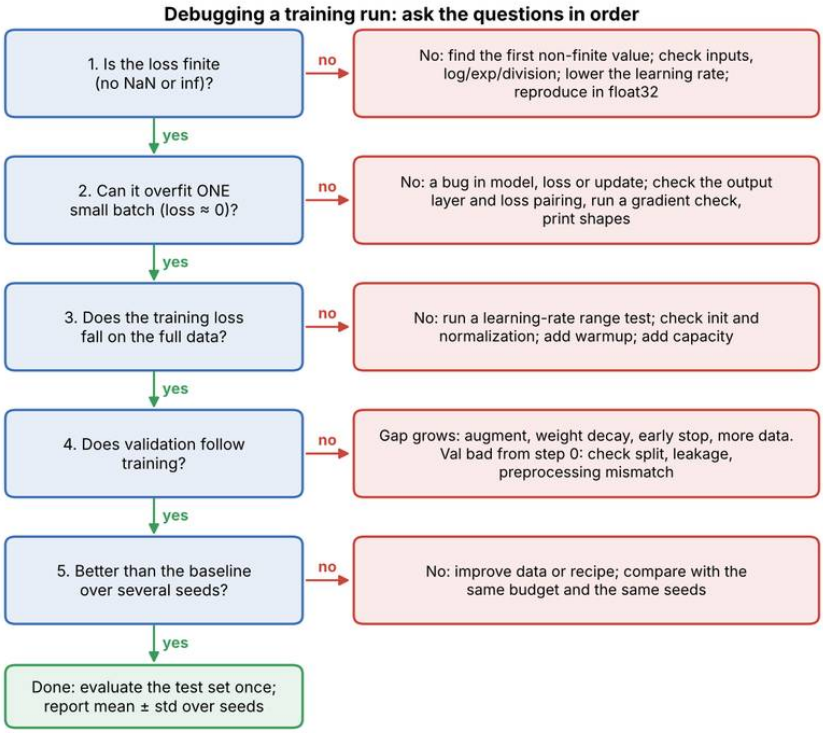


圖 5 — 除錯順序。每個問題都假設上面的問題答案是「是」，所以失敗會指向一小組可能的原因。

先過擬合一個批次

白話版。取一個小批次，反覆在它上面訓練。正確的模型與訓練迴圈能記住幾張影像，所以損失一定會降到接近零。如果做不到，就是哪裡壞了，而你在幾秒鐘內就發現，而不是幾天之後。

Karpathy 的網路訓練配方把這個測試放在很前面：過擬合單一小批次，而在那之前，先檢查初始化時的損失是不是它該有的值 [2]。對於 K 個平衡的類別和初始化良好的最後一層，最初的預測接近均勻，所以交叉熵一開始約為 $\ln K$（我們的數字資料是 $\ln 10 \approx 2.30$）。下面的程式碼先檢查這一點，再對 16 張數字影像執行兩次單批次測試：一次是正確的，一次帶有一個很容易犯的錯，也就是在模型中以及損失內部都套用 softmax。它重複使用執行範例中的 `init` 與 `forward`。

```
def overfit_one_batch(double_softmax=False, steps=300, lr=0.1, seed=0):
    P = init(seed)
    xb, yb = X_tr[:16], y_tr[:16]                        # ONE fixed batch of 16 images
    for step in range(steps):
        h = forward(P, xb)
        z = h[-1]
        if double_softmax:                               # BUG: the "logits" are already
         ↪  probabilities
            z = np.exp(z - z.max(1, keepdims=True))
            z = z / z.sum(1, keepdims=True)
        s = z - z.max(1, keepdims=True)
        p = np.exp(s) / np.exp(s).sum(1, keepdims=True)
        loss = -np.log(p[np.arange(16), yb]).mean()
        d = p.copy()
        d[np.arange(16), yb] -= 1
        d /= 16
        if double_softmax:                               # chain rule through the extra
         ↪  softmax
            d = z * (d - (d * z).sum(1, keepdims=True))
        for i in reversed(range(len(P))):
            gW, gb = h[i].T @ d, d.sum(0)
            if i:
                d = (d @ P[i][0].T) * (h[i] > 0)
            P[i][0] -= lr * gW
            P[i][1] -= lr * gb
    return loss

print(f"loss at init:          {evaluate(init(0), X_tr, y_tr)[0]:.4f}   (ln 10 =
 ↪  {np.log(10):.4f})")
print(f"correct loss:          {overfit_one_batch():.4f}")
print(f"double-softmax bug:    {overfit_one_batch(double_softmax=True):.4f}")
print(f"floor of the bug:      {-np.log(np.e / (np.e + 9)):.4f}")
```

```
loss at init:          2.8204   (ln 10 = 2.3026)
correct loss:          0.0036
double-softmax bug:    1.4661
floor of the bug:      1.4612
```

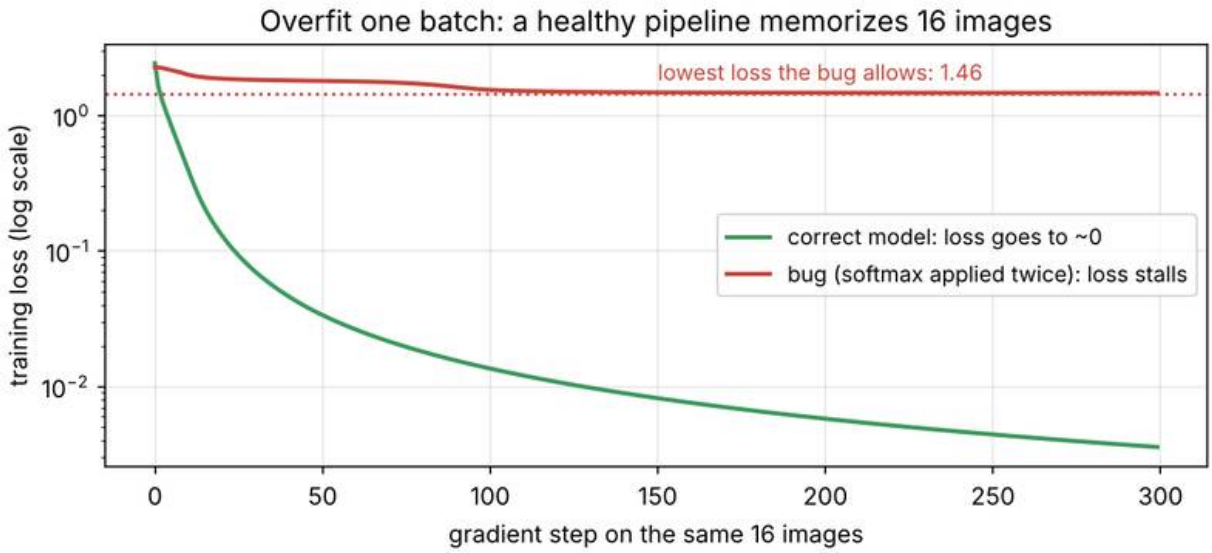


圖 *6* — 過擬合一個批次（我們的實驗）。正確的管線能記住 *16* 張影像；雙重 *softmax* 的錯誤不論訓練多久，都無法低於 *1.46*。

初始損失略高於 $\ln 10$，是因為經過 He 初始化的輸出層會產生中等大小的 logit，所以最初的預測並非完全均勻；這裡無傷大雅。如果數值遠高於 $\ln K$，就表示網路一開始就自信地答錯，把最後一層的初始權重縮小就能解決。這個錯誤有一個硬性的下限。經過第一次 softmax 之後，傳給損失的「logit」介於 0 到 1 之間，所以第二次 softmax 最好也只能讓真實類別得到 e^1，其餘九個類別各得 e^0：機率為 $e/(e+9) \approx 0.23$，損失為 1.46。在完整資料集上，這個錯誤不會崩潰，仍然能產生可用的分類器，所以可能不被發現；單批次測試在幾秒內就能揭露它。

通過測試所能證明的，比看起來少。Zhang 等人證明了標準的影像分類器可以完美擬合隨機標籤，甚至隨機雜訊影像 [36]，所以記住一個批次只能說明模型、損失與更新能夠學習，不能說明標籤是正確的。標錯或對不齊的資料，必須靠實際去看它才能發現（檢查清單第 2 項）。

讀懂損失曲線

白話版。損失曲線是一次訓練的主要紀錄：它的形狀告訴你該懷疑管線的哪個部分。把訓練與驗證曲線畫在一起，損失用對數尺度。

ML 02 解釋了四種基本形狀（平坦、過擬合、收斂、健康）。以下幾種模式在視覺訓練中很典型：

- **正好卡在 $\ln K$**。網路輸出均勻的機率：梯度沒有傳到它那裡（張量被分離、參數被凍結、學習率為零），或是標籤是常數。
- **會恢復的損失尖峰**。學習率接近穩定性的邊緣（如圖 4），或是有幾個批次含有損毀的影像或標籤。記錄損失最大的批次的索引，然後去看它們。
- **驗證比訓練更好**。在強擴增或 dropout 下是正常的，因為訓練損失是在更困難的、經過擴增的輸入上量測的。
- **驗證準確率持平但驗證損失上升**。網路對它答錯的範例變得過度自信；排序沒問題，但機率不對（DLCV 02 的校準）。

- **每個里程碑都會下降一截**。階梯式衰減下這是預期的。如果餘弦排程結束時曲線仍在陡降，就表示預算太短。
- **驗證很好，部署很差**。懷疑是前處理不一致，或是捷徑學習（shortcut learning）：網路依賴一個驗證集與訓練集共有、但部署資料沒有的線索 [37]（DLCV 02）。

梯度範數、死單元與 NaN

白話版。每隔幾百步記錄幾個健康指標：每一層梯度的大小、從不激活的單元所占的比例，以及有沒有東西變成 NaN。訓練出問題時，這些數字通常最先出問題。

下面的程式碼在三種固定學習率下訓練三個 epoch 之後，計算其中兩項。它重複使用上面的 `train`、`forward`、`loss_and_grads` 與 `constant`。

```
def health(P, X, y):
    _, G = loss_and_grads(P, X, y)
    h = forward(P, X)[1]                                  # hidden ReLU activations
    dead = (h.max(0) == 0).mean()                         # units that are zero for every input
    norms = [np.linalg.norm(gW) for gW, _ in G]
    return norms, dead

for peak in (0.01, 0.1, 0.3):
    P, log, _ = train(constant, peak=peak, epochs=3)
    norms, dead = health(P, X_tr, y_tr)
    print(f"lr {peak:<5} train loss {log[-1, 0]:9.3g}  grad norms "
          f"{norms[0]:9.3g} {norms[1]:9.3g}  dead units {dead:.0%}")
```

```
lr 0.01  train loss     0.127  grad norms    0.0664    0.0496  dead units 0%
lr 0.1   train loss      3.61  grad norms      1.44      2.11  dead units 0%
lr 0.3   train loss   2.1e+08  grad norms  1.02e+04  1.31e+04  dead units 0%
```

在健康的實驗與發散的實驗之間，梯度範數增加了五個以上的數量級，而且遠早於損失變成 NaN，所以記錄它們是便宜的保險。梯度裁剪（ML 02）是合理的防護，但它只是掩蓋太大的學習率，而不是解決它。這裡沒有單元死掉，因為這個小玩具是以爆炸的方式失敗。**死亡 ReLU**（dead ReLU）對每個輸入都輸出零，所以得不到梯度，也就再也無法恢復；許多死單元（通常發生在一次巨大的更新之後，或偏置強烈為負時）意味著容量的損失。

當損失變成 NaN 或無限大時，要找出第一個產生非有限值的運算，而不是把所有東西都裁剪：依序檢查輸入、前向傳遞、損失、梯度與更新的大小，並且在怪罪混合精度之前，先在 float32 中重現。ML 01C 詳細給出了這個順序，而它關於檢查導數的一節則展示如何測試手寫的反向傳播。

可重現性與公平的實驗

白話版。一個結果必須能重複，而且背後的比較必須公平，才有用。固定隨機種子、記錄你做了什麼，並且把每個設定跑好幾次，因為單憑隨機機會，就可能讓分數的變動大過你想展示的改進。

隨機種子、確定性與記錄

要為每個亂數產生器設定種子：Python 的、NumPy 的、框架的，以及資料載入器工作行程的。即使如此，各次執行仍可能不同：PyTorch 文件指出，不保證跨版本、跨提交或跨平台，或在 CPU 與 GPU 之間得到完全可重現的結果。在同一個環境內，它建議使用 `torch.use_deterministic_algorithms(True)`、關閉 cuDNN 的 benchmark，並透過 `worker_init_fn` 與 `generator` 為資料載入器工作行程設定種子，同時警告確定性的運算常常比較慢 [38]。要記錄足夠的資訊，讓實驗能重跑：程式碼版本、設定、資料版本與切分識別碼、隨機種子、環境與硬體，以及損失、指標、學習率與梯度範數的曲線（ML 11）。

隨機種子有多重要？

隨機種子改變初始化與資料順序，因此也改變結果。下面的程式碼在小玩具上訓練兩種設定，A（熱身 + 餘弦）與 B（one-cycle），每種各用五個隨機種子，並逐對比較（它重複使用 `train` 與排程函式）：

```
seeds = range(5)
A = [train(warmup_cosine, peak=0.1, seed=s)[1][-1, 3] for s in seeds]
B = [train(one_cycle, peak=0.1, seed=s)[1][-1, 3] for s in seeds]
print("A (warmup + cosine):", np.round(A, 3))
print("B (one-cycle):      ", np.round(B, 3))
d = np.array(B) - np.array(A)                              # paired: same seed, same data order
print(f"A mean {np.mean(A):.3f} +- {np.std(A, ddof=1):.3f}   B mean {np.mean(B):.3f} +-
 ↪ {np.std(B, ddof=1):.3f}")
print(f"paired difference B - A: mean {d.mean():+.3f}, range {d.min():+.3f} to
 ↪ {d.max():+.3f}")
```

```
A (warmup + cosine): [0.971 0.962 0.956 0.956 0.959]
B (one-cycle):       [0.956 0.952 0.952 0.978 0.956]
A mean 0.961 +- 0.007   B mean 0.959 +- 0.011
paired difference B - A: mean -0.002, range -0.016 to +0.022
```

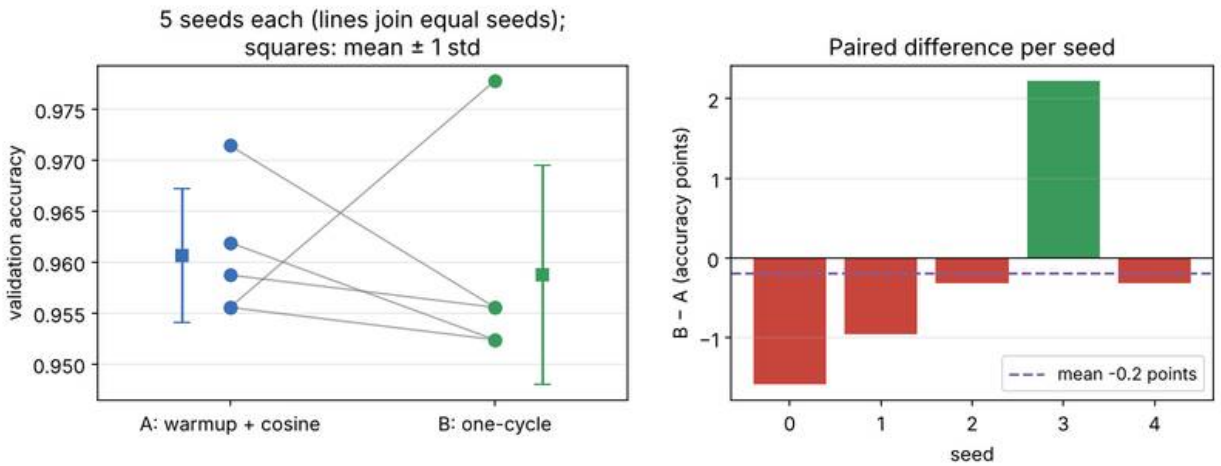


圖 7 — 小玩具上的種子變異（我們的實驗，驗證集 *315* 張影像）。只用單一種子，可能支持任一種結論：種子 *3* 說 *B* 贏 *2.2* 個百分點，種子 *0* 說 *A* 贏 *1.6* 個百分點。五個種子加起來，兩者無法區分。

只看種子 3，你會報告 one-cycle 贏 2.2 個百分點；只看種子 0，則會說它輸 1.6 個百分點。五個種子下，平均差異是 0.2 個百分點，而分散程度遠大於此：這個小玩具上沒有可偵測的差異（而且這裡一張驗證影像就值約 0.3 個百分點）。要回報跨種子的平均值與標準差，在相同的種子與切分上做**配對**比較，並且在宣稱有改進之前，使用 ML 11 的信賴區間與配對檢定。發表的大規模結果也這樣做：Goyal 等人把每個 ImageNet 誤差都報告為五次試驗的平均值與標準差 [9]。

基準、消融實驗與預算

公平的實驗只改變一件事，其他一切保持固定：

- **基準**。一定要包含一個簡單的基準與先前最好的配方，並在同一條管線中由你自己訓練。把新方法與舊論文裡的基準相比，會把方法的效果和配方的效果混在一起（DLCV 02）。
- **相同的調參預算**。讓每個方法的超參數都有相同的試驗次數。一個調了一週的方法，贏過只跑一次的基準，是贏在調參，而不是贏在設計。
- **相同的訓練預算**。在相同的步數或相同的運算量下比較；單是較長的排程就可能解釋一個增益。
- **消融實驗**。從完整方法中一次移除一個元件，每個都用多個隨機種子，並報告下降多少。一次改變兩件事的消融實驗，無法告訴你哪一個才重要。
- **只評估一次測試集**。所有決策都用驗證資料做；最後選定的結果只在測試集上評估一次。

Deep Learning Tuning Playbook 把這些整理成漸進的策略：決定每一輪實驗的目標，把正在研究的超參數、必須針對每個設定重新調整的超參數（「干擾」超參數），以及保持固定的超參數區分開來，並且只在證據充分時才採納一個改變 [26]。

現代觀點

白話版。本篇的核心習慣是穩定的。近期的工作改變的是：如何比較訓練方法、排程有多少必須事先固定，以及大型預訓練模型如何適應新任務。

綜述與指南怎麼說

實用的訓練知識，更多存在於綜述與指南，而不是單篇論文。Bengio 在 *Neural Networks: Tricks of the Trade* 中的章節，收集了基於梯度的訓練建議，從學習率與小批次大小到提前停止與超參數搜尋 [12]。Deep Learning Tuning Playbook 涵蓋了架構、最佳化器、批次大小與初始設定的選擇、漸進而科學的調參過程、要訓練多久，以及檢查點選擇與實驗追蹤等管線細節 [26]。Shorten 與 Khoshgoftaar 把影像擴增整理成幾何與色彩轉換、核濾波器、混合影像、隨機擦除、特徵空間與對抗式擴增、生成式與風格轉換擴增，以及元學習的策略，也討論了測試時擴增 [7]。Xin 等人把視覺模型的參數高效微調分成基於附加的、基於部分的、統一式的與多任務的調整，並提出統一的基準 [39]。

公平地比較訓練方法本身就很困難。Dahl 等人主張，社群無法可靠地辨識訓練演算法的改進，因為不清楚何時才算訓練完成、結果對工作負載的細節很敏感，而且需要調參的方法很難公平比較；他們提出了 AlgoPerf 基準，衡量在固定硬體上跨多個工作負載達到目標結果所需的時間 [40]。

2023–2026 年的最先進技術

- **有基準的訓練演算法**。第一屆 AlgoPerf 競賽收到來自 10 個團隊的 18 份提交。在由提交者提供搜尋空間的規則下，優勝者使用非對角預條件（non-diagonal preconditioning），甚至在掛鐘時間上也勝過 Adam 等常用方法；在不允許調參的規則下，優勝者基於 Schedule-Free AdamW [41]。
- **需要調的排程更少**。Defazio 等人的 Schedule-Free 方法統一了排程與迭代值平均，不需要事先知道停止步數，除了動量最佳化器本身的超參數之外也沒有新的超參數 [42]。一個更簡單的替代方法是固定學習率，後面接一小段「冷卻」衰減，MiniCPM 稱之為熱身-穩定-衰減（warmup-stable-decay, WSD）[43]。Hägele 等人發現，它的擴展性與餘弦排程一樣可預測，只要在訓練的一小部分做冷卻就夠，而且權重平均能沿著訓練軌跡改善結果，不需額外的訓練成本；他們的實驗是在語言模型上，所以要自己驗證能否遷移到視覺的工作負載 [44]。
- **處處都用權重平均**。除了上面的 EMA 研究 [29]，平均也用在微調中：WiSE-FT 在預訓練模型的零樣本權重與微調後權重之間做內插，改善分布偏移下的穩健性 [45]，而 model soups 則把數個微調後的模型取平均，在不增加推論成本的情況下提高準確率 [46]。
- **微調基礎模型**。面對非常大的預訓練骨幹，完整微調可以被線性探測、LP-FT [35]，或 LoRA [33]、[39] 這類參數高效方法取代。實務上的問題從「哪種架構」轉為「哪個預訓練模型，以及要調整它的多少部分」。

尚待解決的問題

1. **能遷移的超參數**。最佳學習率、熱身長度與權重衰減仍會隨模型大小、批次大小與資料集而變；範圍測試與線性縮放規則是近似，不是定律。
2. **公平而負擔得起的比較**。各論文的調參預算不同，隨機種子往往少到無法區分方法。AlgoPerf 這類基準針對訓練演算法處理了這個問題 [40]，但對多數視覺任務還沒有可比較的標準。
3. **大規模下的確定性**。位元層級完全一致的可重現性，跨硬體與軟體版本都不受保證 [38]，這讓大型訓練中的小幅退步很難診斷。
4. **密集與特殊任務的資料擴增**。在幾何與強度都帶有標籤資訊的偵測、分割、醫學與遙測影像上，如何自動選擇資料擴增，仍是未解的問題。
5. **微調而不遺忘**。把基礎模型調整到狹窄的任務，同時保留它在分布外的穩健性，仍是活躍的研究問題 [35]、[45]。

重點整理

- 擴大訓練前，先檢查處理後的批次、初始損失，並確認能過擬合一個批次。
- 依部署時的新資料單位切分，只用訓練資料擬合前處理，空間標籤須隨影像一起變換。
- 用學習率範圍測試、暖身與衰減；改變批次大小時重新檢查排程。

- 梯度累積是否等同大批次，取決於損失加權與批次相依運算；應依實際樣本、像素或方框數正規化。
- 一起調整互相影響的正則化方式，並用驗證資料選擇檢查點。
- 微調預訓練特徵時採取較小幅度更新，並檢查小批次下的正規化行為。
- 報告多個隨機種子、相同預算與單項消融，測試集保留到最後評估。

練習

1. **先範圍測試，再排程**。用批次大小 8、16 與 64（修改 `bs`）重跑範圍測試。最小值處的學習率如何移動？接著在每個批次大小下，用它自己的範圍測試得到的峰值訓練 `warmup_cosine`，並與線性縮放規則比較。

   提示

   在我們的實驗（隨機種子 0 與 1）中，最小值在批次 8 時約為 0.15–0.16，批次 16 時為 0.13–0.14，批次 64 時為 0.35：它隨批次大小上升，但不成精確的比例，而且簡短的測試有雜訊。記得在固定 epoch 數時，以步數計的 T 會隨批次變大而變小，所以較大的批次也得到較少的更新次數。

2. **因錯誤原因通過的單批次測試**。把 `y_tr[:16]` 洗牌（讓標籤不再對應影像），然後執行 `overfit_one_batch`。損失會怎樣？這說明了該測試能偵測什麼、不能偵測什麼？檢查清單中的哪一項檢查本來可以抓出被洗牌的標籤？

   提示

   損失仍然降到接近零（在我們用 `np.random.default_rng(0)` 排列標籤的實驗中是 0.0056），因為網路能記住任意 16 個標籤，就像 Zhang 等人在大規模隨機標籤上所示 [36]。這個測試檢查的是模型、損失與更新，不是資料。看一個畫上標籤的批次（第 2 項）可以抓出對不齊的標籤；驗證準確率接近隨機水準也可以。

3. **翻轉關鍵點**。擴充 `hflip` 函式，讓它也能轉換以 `(x, y, name)` 表示的關鍵點清單，名稱如 `left_eye` 與 `right_eye`。寫一個檢查，在翻轉後的 `left_eye` 仍叫 `left_eye` 時失敗。

   提示

   座標對像素中心遵循 $x \to W - 1 - x$（對連續座標則是 $W - x$，與這裡使用的框慣例相符），y 不變。然後用查找表交換每一對左右名稱。連續套用兩次翻轉必須精確回到原本的關鍵點，這是個好測試。

4. **有被忽略像素的累積**。分割損失在標籤不是「忽略」的像素上，對逐像素交叉熵取平均。微批次 1 有 1,000 個有效像素，平均損失 0.5；微批次 2 有 3,000 個有效像素，平均損失 0.9。合併後批次的平均損失是多少？而把兩個微批次的平均值再取平均，會得到什麼？

   提示

   合併後的平均是 $(1000 \cdot 0.5 + 3000 \cdot 0.9)/4000 = 0.8$。把兩個平均值再取平均得 $(0.5 + 0.9)/2 = 0.7$，這低估了有效像素較多的微批次。梯度也適用同樣的加權：累積總和，再除以總數。

5. **要幾個隨機種子？**用 `train`，對設定 A 與 B 各跑 10 個隨機種子，而不是 5 個。平均差異的正負號會改變嗎？估計配對平均差異的標準誤 $s_d/\sqrt{n}$，其中 s_d 是各種子差異的標準差，n 是種子數，並判斷用 10 個種子能不能偵測到 0.5 個百分點的改進。

   提示

   在我們用種子 0–9 的實驗中，正負號確實改變：B − A 的平均差異變成約 +0.6 個百分點，而五個種子時是 −0.2 個百分點。在 $s_d \approx 1.3$ 個百分點下，標準誤是 $1.3/\sqrt{10} \approx 0.4$ 個百分點，所以 0.5 個百分點的改進離零只有約一個標準誤：不具說服力。更大的驗證集與更多隨機種子都有幫助；ML 10 展示了配對 bootstrap 區間。

6. **微調計畫。**你有 2,000 張有標註的 X 光影像、一個在自然照片上預訓練的骨幹，以及一張一次只能放 8 張影像的 GPU。寫一份訓練計畫：正規化統計量、資料擴增（哪些不安全？）、如何處理批次正規化、微調策略、學習率的做法，以及你要如何判斷某個改變有沒有幫助。

   提示

   使用預訓練的正規化統計量；避免會改變診斷數值的強度擴增，並且在左右側別有意義時小心翻轉；因為每批 8 張影像的統計量有雜訊，所以凍結骨幹的 BN 統計量（或在新層使用群組正規化）；先做線性探測，再做 LP-FT，早期層用較小的學習率並加一小段熱身；對分類頭做一次範圍測試；依病人切分；用多個隨機種子和配對區間比較各變體。

參考文獻

1. T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie and M. Li, “Bag of tricks for image classification with convolutional neural networks,” in *Proc. IEEE/CVF CVPR*, 2019. doi
2. A. Karpathy, “A recipe for training neural networks,” blog post, 2019. link
3. scikit-learn developers, “load_digits,” *scikit-learn documentation*, version 1.9.1. link
4. M. Buda, A. Maki and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Networks*, vol. 106, 2018. doi
5. T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, “Focal loss for dense object detection,” in *Proc. IEEE ICCV*, 2017. doi
6. A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin and A. A. Kalinin, “Albumentations: Fast and flexible image augmentations,” *Information*, vol. 11, no. 2, 2020. doi
7. C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *Journal of Big Data*, vol. 6, 2019. doi
8. K. He, X. Zhang, S. Ren and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in *Proc. IEEE ICCV*, 2015. doi
9. P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia and K. He, “Accurate, large minibatch SGD: Training ImageNet in 1 hour,” arXiv:1706.02677 (technical report), 2017. arXiv
10. S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proc. ICML*, PMLR 37, 2015. link
11. Y. Wu and K. He, “Group normalization,” in *Proc. ECCV*, 2018. doi

12. Y. Bengio, “Practical recommendations for gradient-based training of deep architectures,” in *Neural Networks: Tricks of the Trade*, 2nd ed., LNCS 7700, Springer, 2012. doi
13. L. N. Smith, “Cyclical learning rates for training neural networks,” in *Proc. IEEE WACV*, 2017. doi
14. L. N. Smith, “A disciplined approach to neural network hyper-parameters: Part 1 — learning rate, batch size, momentum, and weight decay,” US Naval Research Laboratory Technical Report 5510-026, arXiv:1803.09820, 2018. arXiv
15. I. Loshchilov and F. Hutter, “SGDR: Stochastic gradient descent with warm restarts,” in *Proc. ICLR*, 2017. arXiv
16. L. N. Smith and N. Topin, “Super-convergence: Very fast training of neural networks using large learning rates,” in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, Proc. SPIE 11006, 2019. doi
17. PyTorch contributors, “torch.optim.lr_scheduler.OneCycleLR,” *PyTorch 2.15 documentation*, 2026. link
18. K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE CVPR*, 2016. doi
19. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE CVPR*, 2009. doi
20. D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. ICLR*, 2015. arXiv
21. I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *Proc. ICLR*, 2019. arXiv
22. PyTorch contributors, “torch.optim.Adam,” *PyTorch 2.15 documentation*, 2026. link
23. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, 2014. link
24. G. Huang, Y. Sun, Z. Liu, D. Sedra and K. Q. Weinberger, “Deep networks with stochastic depth,” in *Proc. ECCV*, 2016. doi
25. L. Prechelt, “Early stopping — but when?,” in *Neural Networks: Tricks of the Trade*, 2nd ed., LNCS 7700, Springer, 2012. doi
26. V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue and Z. Nado, “Deep learning tuning playbook,” Google Research, GitHub repository, 2023. link
27. B. T. Polyak and A. B. Juditsky, “Acceleration of stochastic approximation by averaging,” *SIAM Journal on Control and Optimization*, vol. 30, no. 4, 1992. doi
28. P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov and A. G. Wilson, “Averaging weights leads to wider optima and better generalization,” in *Proc. UAI*, 2018. arXiv
29. D. Morales-Brotons, T. Vogels and H. Hendrikx, “Exponential moving average of weights in deep learning: Dynamics and benefits,” *Transactions on Machine Learning Research*, 2024. arXiv
30. P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh and H. Wu, “Mixed precision training,” in *Proc. ICLR*, 2018. arXiv
31. PyTorch contributors, “Automatic mixed precision package — torch.amp,” *PyTorch 2.15 documentation*, 2026. link
32. J. Yosinski, J. Clune, Y. Bengio and H. Lipson, “How transferable are features in deep neural networks?,” in *Advances in Neural Information Processing Systems 27*, 2014. link

33. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang and W. Chen, “LoRA: Low-rank adaptation of large language models,” in *Proc. ICLR*, 2022. arXiv
34. J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” in *Proc. ACL*, 2018. doi
35. A. Kumar, A. Raghunathan, R. Jones, T. Ma and P. Liang, “Fine-tuning can distort pretrained features and underperform out-of-distribution,” in *Proc. ICLR*, 2022. arXiv
36. C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, “Understanding deep learning requires rethinking generalization,” in *Proc. ICLR*, 2017. arXiv
37. R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge and F. A. Wichmann, “Shortcut learning in deep neural networks,” *Nature Machine Intelligence*, vol. 2, 2020. doi
38. PyTorch contributors, “Reproducibility,” *PyTorch 2.15 documentation*, 2026. link
39. Y. Xin, J. Yang, S. Luo, Y. Du, Q. Qin, K. Cen, Y. He, Z. Zhang, B. Fu, X. Yang, G. Zhai, M.-H. Yang and X. Liu, “Parameter-efficient fine-tuning for pre-trained vision models: A survey and benchmark,” arXiv:2402.02242, 2024. arXiv
40. G. E. Dahl, F. Schneider, Z. Nado, N. Agarwal, C. S. Sastry, P. Hennig, S. Medapati, R. Eschenhagen, P. Kasimbeg, D. Suo et al., “Benchmarking neural network training algorithms,” arXiv: 2306.07179, 2023. arXiv
41. P. Kasimbeg, F. Schneider, R. Eschenhagen, J. Bae, C. S. Sastry, M. Saroufim, B. Feng, L. Wright, E. Z. Yang, Z. Nado, S. Medapati, P. Hennig, M. Rabbat and G. E. Dahl, “Accelerating neural network training: An analysis of the AlgoPerf competition,” in *Proc. ICLR*, 2025. arXiv
42. A. Defazio, X. A. Yang, H. Mehta, K. Mishchenko, A. Khaled and A. Cutkosky, “The road less scheduled,” in *Advances in Neural Information Processing Systems 37*, 2024. link
43. S. Hu, Y. Tu, X. Han, C. He, G. Cui, X. Long, Z. Zheng, Y. Fang, Y. Huang, W. Zhao et al., “MiniCPM: Unveiling the potential of small language models with scalable training strategies,” in *Proc. Conference on Language Modeling (COLM)*, 2024. arXiv
44. A. Hägele, E. Bakouch, A. Kosson, L. Ben Allal, L. Von Werra and M. Jaggi, “Scaling laws and compute-optimal training beyond fixed training durations,” in *Advances in Neural Information Processing Systems 37*, 2024. link
45. M. Wortsman, G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs, R. Gontijo Lopes, H. Hajishirzi, A. Farhadi, H. Namkoong and L. Schmidt, “Robust fine-tuning of zero-shot models,” in *Proc. IEEE/CVF CVPR*, 2022. doi
46. M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith and L. Schmidt, “Model soups: Averaging weights of multiple fine-tuned models improves accuracy without increasing inference time,” in *Proc. ICML*, PMLR 162, 2022. link
47. PyTorch contributors, “Automatic mixed precision examples,” *PyTorch 2.15 documentation*, 2026. link
48. PyTorch contributors, “DistributedDataParallel,” PyTorch 2.15.0 source documentation, 2026. Official source.

DLCV 01B・網路的積木與組合：分類與影像復原網路

深度視覺網路由哪些積木搭成、如何把它們組合成分類網路與影像復原網路、對照比較配對的模型，以及如何探索設計空間而不自欺。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/

你將學到

- 辨認網路的 stem、階段、區塊、neck 與 head。
- 追蹤區塊如何混合空間與通道資訊，並估計成本。
- 解釋分類與復原為何以不同方式使用解析度、正規化與跳接。
- 比較對應的網路設計，並以小型實驗診斷組合錯誤。
- 在控制訓練與計算預算的條件下解讀消融。

先看全貌

在論文的圖裡，視覺網路看起來很複雜，但它其實只由少數幾種積木重複、連接而成。兩個用了相同積木的網路，行為可能差很多，因為它們的**組合**方式不同：解析度在哪裡改變、跳接接到哪裡、正規化放在哪裡、輸出怎麼形成。本章談的就是這些組合上的決定。

> **用最簡單的話說**
>
> 想像一個工具箱：鋸子、刷子、膠水。家具師傅用它們把一根大原木變成一隻小小的椅腳：削掉大部分的木頭，只留下重要的部分。修復畫作的人用同一批工具，做法卻完全不同：畫必須維持原來的大小，每一筆都留在原位，只是變得更乾淨。分類器就是家具師傅：它不斷丟掉像素，直到只剩下答案（「貓」）。影像復原網路則是修復師：它必須把每一個像素都還給你。工具相同，計畫相反。

為什麼重要。實務上很少需要發明新的積木，更常見的是挑選骨幹網路、把它改到新任務上，或判斷一篇論文的十項改動裡哪幾項真的有用。把分類骨幹原封不動拿去做去雜訊，會丟掉細節；而某個在別篇論文有效的積木，放進你的設計空間未必有效。

基礎在哪裡。這些積木本身在站內別處已經教過，本章只連結過去、不重複：卷積、感受野與殘差區塊見 ML 03 與 DLCV 01；Transformer 區塊、Swin 與高效率注意力見 ML 05；分類骨幹網路見 DLCV 02；編碼器–解碼器與 ASPP 見 DLCV 04；復原的骨幹網路與上取樣見 DLCV 09 與 DLCV 09A。本章所有程式碼只用 NumPy 與 SciPy，在 CPU 上幾秒就能跑完，資料是合成資料或 `skimage.data`。

網路的解剖

白話版。視覺網路通常包含擷取特徵的 stem、重複區塊組成的主體，以及任務 head。閱讀時追蹤三者中的解析度變化。

精確版。把特徵圖寫成形狀為 $C \times H \times W$ 的張量，其中 C 是通道數（**寬度**），$H \times W$ 是空間大小。常見的組成如下：

- **Stem**：最前面的一兩層。ResNet 用步幅 2 的 7×7 卷積，加上步幅 2 的 3×3 最大池化，所以主體是從輸入解析度的 1/4 開始 [1]。ConvNeXt 改成「patchify」：步幅 4 的 4×4 卷積 [2]；ViT 則把影像切成 16×16 的 patch [3]。復原網路只用一個全解析度的 3×3 卷積，SwinIR 稱它為**淺層特徵擷取（shallow feature extraction）** [4]。
- **Stage**：一串共用同一解析度與同一寬度的區塊。**區塊（block）**是重複的單位（殘差區塊、Transformer 區塊）。
- **下取樣層（downsampling layer，也叫轉換層）**：把 H 與 W 減半，通常同時把 C 加倍。它可以是 stage 第一個區塊裡的步幅卷積（ResNet）、獨立的步幅 2 的 2×2 卷積（ConvNeXt、NAFNet）、patch merging（Swin），或 pixel unshuffle（Restormer）。
- **Neck**：選用的模組，用來混合多個 stage 的特徵，例如偵測器使用的特徵金字塔網路（Feature Pyramid Network，FPN）[5]。
- **Head**：分類用的是全域平均池化加一個線性層；復原用的是最後一個卷積，超解析度還要再加上取樣器。

一個 stage 的**輸出步幅（output stride）**，是輸入大小與該 stage 特徵圖大小的比值。ResNet 的四個 stage，輸出步幅依序是 4、8、16、32 [1]；Swin 也一樣 [6]。整個網路都維持同一解析度與同一寬度的（如 ViT）稱為**等向（isotropic）**；stage 解析度逐步下降的稱為**階層式（hierarchical）**。

ResNet 也提出了被抄得最多的整體設計規則：特徵圖縮小一半時，濾波器數量加倍，「好讓每一層的時間複雜度維持不變」[1]。一個輸入與輸出都是 C 個通道的 3×3 卷積，作用在 $H \times W$ 的特徵圖上，要花 $9HWC^2$ 次乘加（multiply-accumulate，MAC）；特徵圖縮小一半、寬度加倍後，成本是 $9 \cdot \frac{H}{2} \cdot \frac{W}{2} \cdot (2C)^2 = 9HWC^2$，完全相同。圖 1 畫出本章要比較的三種整體架構。

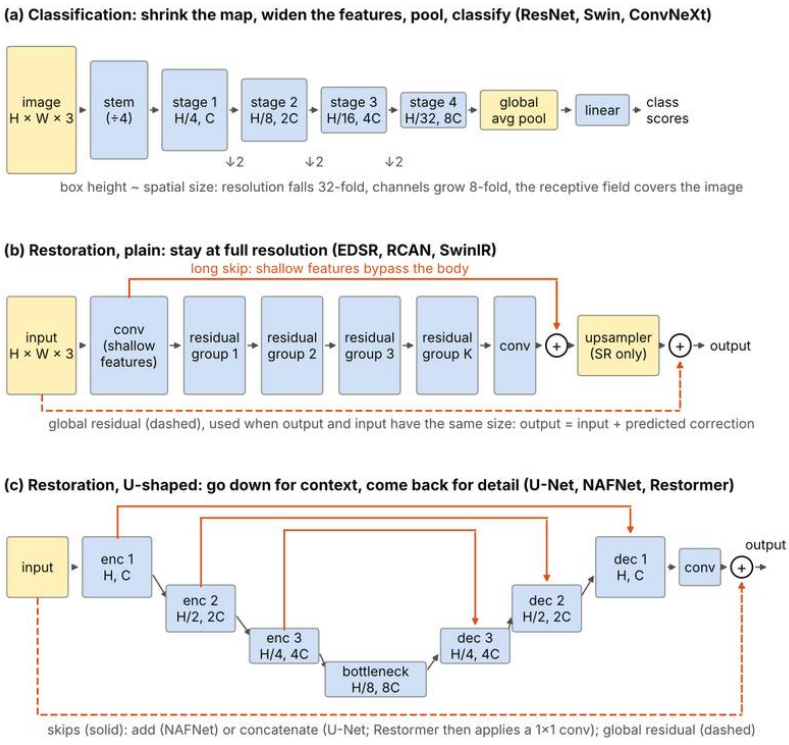


圖 *1* — 三種整體架構。(*a*) 分類器像漏斗：解析度下降、寬度增加，全域池化去掉空間格點。(*b*)

一般復原網路維持全解析度，並加上長跳接與全域跳接。(*c*) *U* 形復原網路往下取得脈絡、再往上找回細節。

積木目錄

白話版。區塊混合空間與通道資訊；跳接、正規化與非線性函數控制混合對訊號與成本的影響。

MetaFormer 論文把這個分工講得很明確：Transformer 區塊是一個 **token 混合器（token mixer）**（注意力）加一個**通道混合器（channel mixer）**（MLP），各自放在帶有正規化的殘差分支裡；把注意力換成簡單的空間池化運算，在 ImageNet 上仍有具競爭力的準確率 [7]，這暗示整體結構本身就貢獻了大部分的效能 [8]。圖 2 用這個共同格式畫出本章的各個區塊。

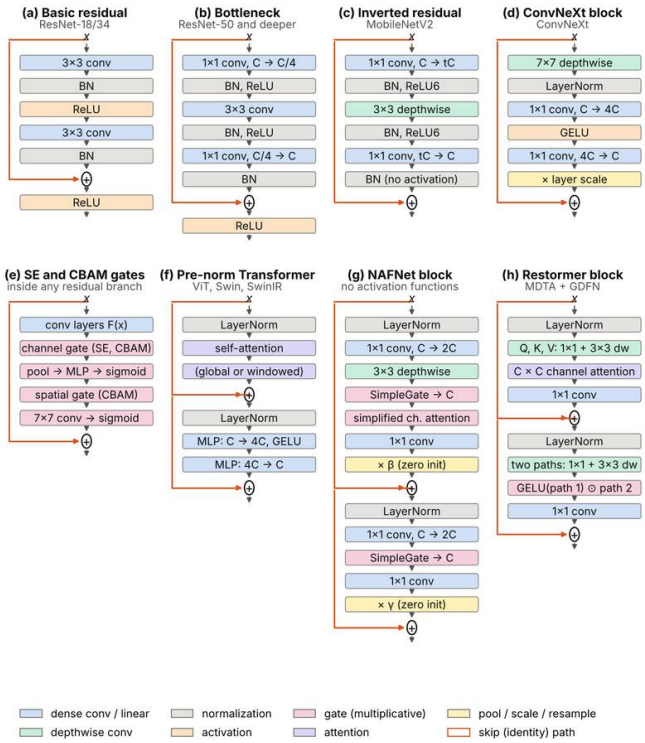


圖 2 — 以同一格式畫出的積木目錄。顏色代表運算的種類；橘色箭頭是恆等跳接。(*a*)–(*d*) 是分類用的區塊，(*e*) 是注意力附加模組，(*f*) 是 *Transformer* 區塊，(*g*)–(*h*) 是復原用的區塊。

下表為每個區塊配一句白話說明，並標出它的基礎在哪裡教。

區塊	一句白話	用在	基礎
殘差（基本）	學一個修正量 $F(x)$，加回輸入	ResNet-18/34、EDSR	DLCV 01
瓶頸	用 1×1 壓縮通道，濾波，再擴張回來	ResNet-50 與更深的網路	DLCV 02
深度可分離	每個通道各自濾波，再用 1×1 混合通道	MobileNet [9]	DLCV 02

區塊	一句白話	用在	基礎
反向殘差	先擴張、做深度濾波、再線性投影回來；跳接接在細的兩端	MobileNetV2 [10]	ML 03
ConvNeXt 區塊	大的深度卷積核，加上 Transformer 風格的 MLP	ConvNeXt	DLCV 02
SE / CBAM	依據池化統計算出的閘，重新調整通道（以及位置）的權重	SENet、RCAN、NAFNet	DLCV 02 補充資料
自注意力	每個 token 都看所有其他 token	ViT	ML 05
視窗注意力	在局部視窗內做注意力，視窗在層與層之間位移	Swin、SwinIR	ML 05
通道（轉置）注意力	在通道之間做注意力，成本隨像素數線性增加	Restormer	本章
閘控區塊	把兩半特徵相乘，取代活化函數	NAFNet、Restormer 的 GDFN	本章

殘差連接與殘差縮放

白話版。殘差區塊把學得的修正量加回輸入；需檢查正規化、活化函數的位置及修正量的初始尺度。

精確版。一般形式是

$$y = x + \alpha F(x),$$

其中 x 是區塊輸入，F 是殘差分支（卷積、注意力、正規化），y 是輸出，α 是分支的縮放係數。非線性函數常見的放法有三種：

- **後活化（post-activation）**（原始 ResNet）：$y = \mathrm{ReLU}(x + F(x))$，BN 在 F 裡面 [1]。
- **前活化（pre-activation）**：BN 與 ReLU 移進分支裡、放在每個卷積之前，使捷徑成為純粹的恆等映射；He 等人發現這樣在非常深時比較容易訓練 [11]。Pre-norm 的 Transformer 區塊 $y = x + \mathrm{MHA}(\mathrm{LN}(x))$ 是同一個想法（ML 05）。
- **加法之後沒有活化函數**：EDSR、MobileNetV2、ConvNeXt、NAFNet 與 Restormer 的區塊都以加法收尾。

有幾種設計會明確控制 α。EDSR 發現特徵圖數量變大後訓練在數值上不穩定，於是在其 32 個區塊、256 個通道的最終模型中，固定使用 0.1 的殘差縮放 [12]。ConvNeXt 使用 Layer Scale [13]，也就是每個通道各自可學習的 α，初始值為 10^{-6} [2]。NAFNet 的程式碼把每個分支乘上初始值為零的可學習逐通道參數，使每個區塊一開始都是恆等映射 [14]；把 ResNet 分支最後一個 BN 的縮放係數初始化為零，效果相同（DLCV 01A）。共同的理由是：一疊恆等區塊很容易最佳化，之後每個區塊再慢慢長出自己的修正量。

跳接也有兩種尺度。**短跳接**包住一個區塊；**長跳接**包住一組區塊或整個主體。RCAN 的「residual in residual」結構兩種都用：殘差群組帶有長跳接，每個群組裡的區塊再帶有短跳接 [15]。

訓練與推論也可以使用不同的模組計算圖：RepVGG [54] 在訓練時加入額外的仿射分支，部署前再合併成一個卷積。

僅線上版：*RepVGG*：訓練用分支，部署用直線網路 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/papers/repvgg* 訓練用分支部署用直線網路/

瓶頸、反向殘差與深度可分離卷積

白話版。空間濾波與通道混合不必擠在同一個昂貴的層裡完成。瓶頸讓昂貴的 3×3 在較少的通道上執行；深度卷積讓每個通道各自濾波，把混合通道的工作留給便宜的 1×1 層。這些區塊的成本會差上很多倍，所以值得算清楚。

精確版。一個從 a 個通道到 b 個通道的稠密 $k\times k$ 卷積，有 k^2ab 個權重；在步幅為 1 時，每個輸出像素要花 k^2ab 次 MAC。作用在 a 個通道上的深度 $k\times k$ 卷積，兩者都是 k^2a。ResNet 的瓶頸把 C 降到 $C/4$、濾波，再擴張回來 [1]。MobileNetV2 的**反向殘差（inverted residual）**則相反：先用 1×1 卷積把 C 擴張到 tC（論文主要實驗的擴張倍率是 $t=6$），用 3×3 深度卷積濾波，再用**線性**的 1×1 卷積投影回來，而且不接活化函數，因為作者主張在窄的層裡接 ReLU 會破壞資訊 [10]。論文給出的成本是：對 $h\times w$ 的輸入、d' 個輸入通道與 d'' 個輸出通道，共 $h\cdot w\cdot d'\cdot t\,(d'+k^2+d'')$ 次乘加 [10]。下面的程式碼在 $C=64$、特徵圖 128×128 的條件下，數出圖 2 各區塊每個輸出像素的權重與 MAC，並用這個公式檢驗反向殘差。

```
# Weights (no biases or norm parameters) and MACs per output pixel of common blocks,
# all with C input and C output channels at stride 1. A dense k x k conv from a to b
# channels has k*k*a*b weights and costs k*k*a*b MACs per pixel; a depthwise one k*k*a.
def blocks(C, n_tokens):
    g = int(2.66 * C)                                   # Restormer GDFN hidden width
    return {                                            # name: (weights, MACs per pixel)
        'basic residual (2 x 3x3)':     (18 * C * C, 18 * C * C),
        'bottleneck (1x1, 3x3, 1x1)':   (2 * C * C // 4 + 9 * (C // 4) ** 2,) * 2,
        'depthwise separable 3x3':      (9 * C + C * C,) * 2,
        'inverted residual, t = 6':     (12 * C * C + 54 * C,) * 2,
        'ConvNeXt (7x7 dw, 4x MLP)':    (49 * C + 8 * C * C,) * 2,
        'NAFNet block':                 (7 * C * C + 18 * C, 6 * C * C + 18 * C),  # SCA acts
        ↪  on a pooled vector
        'SE add-on, r = 16':            (2 * C * C // 16, C),                       # one
        ↪  multiply per pixel
        'ViT block, global attention':  (12 * C * C, 12 * C * C + 2 * n_tokens * C),
        'Swin block, 8 x 8 windows':    (12 * C * C, 12 * C * C + 2 * 64 * C),
        'Restormer block (MDTA+GDFN)':  (4 * C * C + 27 * C + 3 * g * C + 18 * g,
                                         4 * C * C + 27 * C + 2 * C * C // 4 + 3 * g * C + 18
                                          ↪  * g),  # 4 heads
    }

C, side = 64, 128
for name, (p, m) in blocks(C, side * side).items():
    print(f"{name:30s} {p:7d} weights  {m:8d} MACs/pixel")

# Check the inverted residual against MobileNetV2's formula h*w*d'*t*(d' + k^2 + d'') per
↪  pixel.
t, k = 6, 3
assert blocks(C, 1)['inverted residual, t = 6'][1] == C * t * (C + k * k + C)
```

```
basic residual (2 x 3x3)         73728 weights     73728 MACs/pixel
```

```
bottleneck (1x1, 3x3, 1x1)        4352 weights      4352 MACs/pixel
depthwise separable 3x3           4672 weights      4672 MACs/pixel
inverted residual, t = 6         52608 weights     52608 MACs/pixel
ConvNeXt (7x7 dw, 4x MLP)        35904 weights     35904 MACs/pixel
NAFNet block                     29824 weights     25728 MACs/pixel
SE add-on, r = 16                  512 weights        64 MACs/pixel
ViT block, global attention      49152 weights   2146304 MACs/pixel
Swin block, 8 x 8 windows        49152 weights     57344 MACs/pixel
Restormer block (MDTA+GDFN)      53812 weights     55860 MACs/pixel
```

這些數字帶出四個教訓。

1. **寬度相同時，區塊的成本差了 10 倍以上**。瓶頸與可分離區塊比基本區塊便宜約 16 到 17 倍。所以真實的網路從來不在相同寬度下比較區塊：MobileNetV2 把反向殘差用在細的張量上，ResNet-50 把瓶頸用在寬的張量上。比較區塊要在相同成本下比，而不是在相同的 C 下比。
2. **在像素上做全域注意力，對復原來說負擔不起**。把 128×128 特徵圖的每個像素都當成一個 token，ViT 區塊每個像素的成本約是 8×8 視窗 Swin 區塊的 37 倍，而且影像越大差距越大。這就是復原用的 Transformer 要用視窗（SwinIR）或跨通道的注意力（Restormer）的原因；一般的成本論證見 ML 05。
3. **注意力附加模組在 MAC 上很便宜**。SE 閘只增加 $C^2/8$ 個權重，以及每個通道每個像素一次乘法；它的代價在記憶體搬運，不在算術。
4. **MAC 不等於延遲**。深度層每搬動一個位元組只做很少的運算，所以實際可能比 MAC 數字顯示的慢；ML 03 與 DLCV 02 的效率一節都談過。請在目標硬體上實測。

便宜的區塊是不是也比較好，要看設計空間。RegNet 的族群研究發現，它最好的模型既不用瓶頸，也不用反向瓶頸，而且在它的設定下深度卷積比反向瓶頸更差 [16]；ConvNeXt 卻發現在它自己那串改動裡，反向瓶頸略好一些 [2]。兩者可以同時成立：比較的結果取決於固定不動的其他選擇。

CNN 內的注意力：SE 與 CBAM

白話版。閘會看整張特徵圖的摘要，決定這張特定影像要把哪些通道（或位置）調大或調小。這是讓卷積網路擁有一點「依內容而定、看全影像」行為的便宜方法。

精確版。Squeeze-and-excitation（SE）根據各通道平均值組成的向量 $\bar{u}$，算出通道閘 $s = \sigma(W_2\, \delta(W_1\, \bar{u}))$，其中 W_1 與 W_2 是兩層 MLP 的權重（先以比例 r 縮減維度、再還原），δ 是 ReLU，σ 是 sigmoid；然後把特徵圖的每個通道 u_c 乘上 s_c。公式與程式碼見 DLCV 02 補充資料 [17]。CBAM 從兩方面擴充它 [18]：它的通道閘同時用平均與最大值池化每個通道，並讓兩個向量通過同一個共用的 MLP；另外還有第二個**空間**閘，沿通道方向池化後套用 7×7 卷積，讓區塊也能把整個區域調大或調小。CBAM 的消融實驗發現，先通道後空間的順序，比兩個閘平行執行更好 [18]。

僅線上版：*CBAM 的通道閘與空間閘 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/notes/cbam-的通道閘與空間閘/*

放在哪裡。SE 和 CBAM 都放在殘差分支上、加法之前：SE 把閘作用在非恆等分支，「在與恆等分支相加之前」[17]；CBAM 則把模組套在每個 ResNet 區塊的卷積輸出上 [18]。RCAN 把通道注意力帶進超解析度 [15]；NAFNet 後來把它簡化成 $X \cdot W\,\mathrm{pool}(X)$，也就是對池化向量做一次線

性映射，沒有 ReLU 也沒有 sigmoid，而它的消融實驗中沒有損失任何效果 [19]。「現代觀點」中的注意力綜述，把 SE 歸入「通道注意力」一族，把 CBAM 歸入「通道加空間」[20]。

Token 混合器：自注意力、視窗注意力與通道注意力

白話版。注意力是依內容算出權重，在 token 之間混合資訊；卷積用的則是固定權重。設計上的問題是**哪些 token 彼此交談**：全部的 token（全域）、同一個局部視窗內的 token，或是 Restormer 的新招：讓通道之間交談，而不是像素之間。

精確版。對 N 個寬度為 C 的 token 做全域多頭自注意力，分數與加權和約需 $2N^2C$ 次 MAC，另外投影還要 $4NC^2$；視窗注意力把範圍限制在 $M \times M$ 視窗內，成本是 $2M^2NC$（ML 05）[6]。Restormer 的**多頭深度卷積轉置注意力（multi-Dconv head transposed attention，MDTA）**則是在通道之間計算注意力 [21]：

$$\hat{X} = W_p\,\mathrm{Attention}(\hat{Q},\hat{K},\hat{V}) + X, \qquad \mathrm{Attention}(\hat{Q},\hat{K},\hat{V}) = \hat{V}\cdot \mathrm{Softmax}(\hat{K}\cdot\hat{Q}/\alpha),$$

其中 X 是輸入特徵圖，$\hat{X}$ 是輸出，$Q = W_d^Q W_p^Q Y$（K、V 同理）由經過 layer normalization 的輸入 Y 算出：先做 1×1 卷積 W_p^Q，再做 3×3 深度卷積 W_d^Q；帽子符號表示重新整形的矩陣 $\hat{Q}\in\mathbb{R}^{HW\times C}$、$\hat{K}\in\mathbb{R}^{C\times HW}$、$\hat{V}\in\mathbb{R}^{HW\times C}$；$\alpha$ 是可學習的溫度參數；沒有上標的 W_p 是輸出的 1×1 卷積。注意力圖 $\hat{K}\hat{Q}$ 的大小是 $C\times C$，而不是 $HW\times HW$，所以成本隨像素數線性增加，而每個輸出像素仍透過通道統計量依賴整張影像。深度卷積把局部脈絡補了回來。和一般多頭注意力一樣，通道會分成幾個頭，各有各的注意力圖。

Restormer 的前饋部分（GDFN）和下面的閘一起介紹。這些混合器之間的取捨，會在復原的各節再次出現：視窗注意力在空間上混合得強、但只在局部；通道注意力以低成本達到全域；而一項比較研究主張，在超解析度上 Restormer 的空間映射能力比 SwinIR 弱 [22]。

正規化：BN、LN、GN，以及何時不用

白話版。正規化層把特徵調整到標準大小，再讓網路學出它偏好的尺度。各層的差別在於**哪些數字共用統計量**。在分類中，這個選擇多半只影響訓練穩定度；在復原中，選錯可能改變輸出的亮度與對比。

精確版。所有常見的層都在某個數值集合上計算 $y = \gamma(x-\mu)/\sqrt{\sigma^2+\epsilon}+\beta$（範本見 ML 02）。對形狀為 $N\times C\times H\times W$ 的 mini-batch：

層	統計量計算的範圍	是否依賴批次？
BatchNorm（BN）[23]	每個通道在 N,H,W 上	是；測試時用移動平均
LayerNorm（LN）[24]	單一樣本的特徵（用在影像時，實作不同：整個 C,H,W，或每個位置的 C）	否
InstanceNorm（IN）[25]	每個樣本、每個通道在 H,W 上	否
GroupNorm（GN）[26]	每個樣本在 H,W 與一組通道上	否；$G=1$ 等於 LN，$G=C$ 等於 IN

BN 是分類 CNN 的預設選擇，在大批次下表現很好。它已知的失敗情況是小批次：GN 的作者報

告，在批次大小為 2 時，ImageNet 上的 ResNet-50 搭配 GN，錯誤率比搭配 BN 的版本低 10.6% [26]。BN 計算了什麼、為什麼訓練與推論的行為不同，見 DLCV 01 補充資料。

為什麼復原網路拿掉 BN。 EDSR 把 BN 從殘差區塊中移除，理由是 BN 把特徵正規化之後，會讓網路失去「範圍彈性（range flexibility）」；這項改動提升了準確率，並且比 SRResNet（SRGAN 論文的殘差 SR 網路 [27]）少用了約 40% 的訓練記憶體 [12]。ESRGAN 補充說，當訓練與測試的統計量不同時，BN 容易產生偽影，在以 GAN 損失訓練的較深網路中尤其明顯，所以也把它拿掉 [28]。這並不是定律：DnCNN 把殘差學習和 BN 結合起來做高斯去雜訊，並報告兩者互相幫助 [29]。下面的程式碼用一個固定、未訓練的殘差分支 $r(x)$（卷積、正規化、ReLU、1×1 卷積）與全域殘差 $y = x + r(x)$ 來示範這個機制。同一張中灰度影像 A，被放在兩個不同的批次裡復原。

```
import numpy as np
from scipy.ndimage import gaussian_filter, correlate

rng = np.random.default_rng(0)

def image(level, size=32):
    """A smooth synthetic image with mean brightness `level` (values roughly in [0, 1])."""
    t = gaussian_filter(rng.standard_normal((size, size)), 2)
    return level + 0.15 * t / t.std()

C = 4
W1 = rng.normal(0.1, 0.2, (C, 3, 3))          # 3x3 conv, 1 -> C channels (fixed random
 ↪  weights)
w2 = rng.normal(0.0, 0.05, C)                 # 1x1 conv, C -> 1 channel

def features(batch):                          # (N, H, W) -> (N, C, H, W)
    return np.stack([[correlate(x, W1[c], mode='nearest') for c in range(C)] for x in batch])

def branch(batch, norm, running=None):
    """Residual branch: conv -> normalize -> ReLU -> 1x1 conv."""
    f = features(batch)
    if norm == 'bn-train':                    # per channel, over the whole batch
        mu, var = f.mean((0, 2, 3), keepdims=True), f.var((0, 2, 3), keepdims=True)
    elif norm == 'bn-eval':                   # per channel, stored training statistics
        mu, var = running
    else:                                     # 'instance': per sample and channel
        mu, var = f.mean((2, 3), keepdims=True), f.var((2, 3), keepdims=True)
    h = np.maximum((f - mu) / np.sqrt(var + 1e-5), 0)
    return np.tensordot(w2, h, axes=(0, 1))   # (N, H, W)

train = np.stack([image(rng.uniform(0.3, 0.7)) for _ in range(64)])
f = features(train)
running = (f.mean((0, 2, 3), keepdims=True), f.var((0, 2, 3), keepdims=True))

A = image(0.5)
dark = np.stack([image(0.15) for _ in range(7)])
bright = np.stack([image(0.85) for _ in range(7)])
for norm in ['bn-train', 'bn-eval', 'instance']:
    for name, others in [('dark', dark), ('bright', bright)]:
        batch = np.concatenate([A[None], others])
        out = A + branch(batch, norm, running)[0]          # global residual: y = x + r(x)
        print(f"{norm:8s}  A with 7 {name:6s} images: output mean {out.mean():.3f}")
```

```
# Per-sample normalization forgets each image's level and contrast:
B = 0.5 * A + 0.3
r_A, r_B = branch(A[None], 'instance')[0], branch(B[None], 'instance')[0]
print("instance-norm branch, A vs 0.5*A + 0.3: max |r_A - r_B| =", np.abs(r_A -
 ↪  r_B).max().round(4))
print("outputs with the global residual: means", (A + r_A).mean().round(3), (B +
 ↪  r_B).mean().round(3))
```

```
bn-train  A with 7 dark   images: output mean 0.629
bn-train  A with 7 bright images: output mean 0.482
bn-eval   A with 7 dark   images: output mean 0.509
bn-eval   A with 7 bright images: output mean 0.509
instance  A with 7 dark   images: output mean 0.521
instance  A with 7 bright images: output mean 0.521
instance-norm branch, A vs 0.5*A + 0.3: max |r_A - r_B| = 0.0005
outputs with the global residual: means 0.521 0.575
```

在訓練模式下，BN 讓同一張影像（輸入平均值 0.492）的輸出亮度變成 0.629 或 0.482，只取決於同批次的其他影像（圖 3）。在評估模式下，批次不再有影響，但儲存的統計量來自其他影像，所以網路在測試時的行為和訓練時不同：這就是 EDSR 與 ESRGAN 描述的不一致。逐樣本的正規化不依賴批次，卻在另一個方向上「看不見」：分支對 A 和對 $0.5A + 0.3$ 給出相同的輸出（差距只在 ϵ 的量級），所以只有恆等路徑承載了影像本身的亮度與對比。這就是為什麼有做正規化的復原網路（NAFNet、Restormer 與 SwinIR 都用 LayerNorm）要把它放在殘差分支**裡面**，並搭配全域殘差 [19]、[21]、[4]。這裡的權重是隨機且未訓練的；訓練可以部分補償，但對批次的依賴是結構性的。

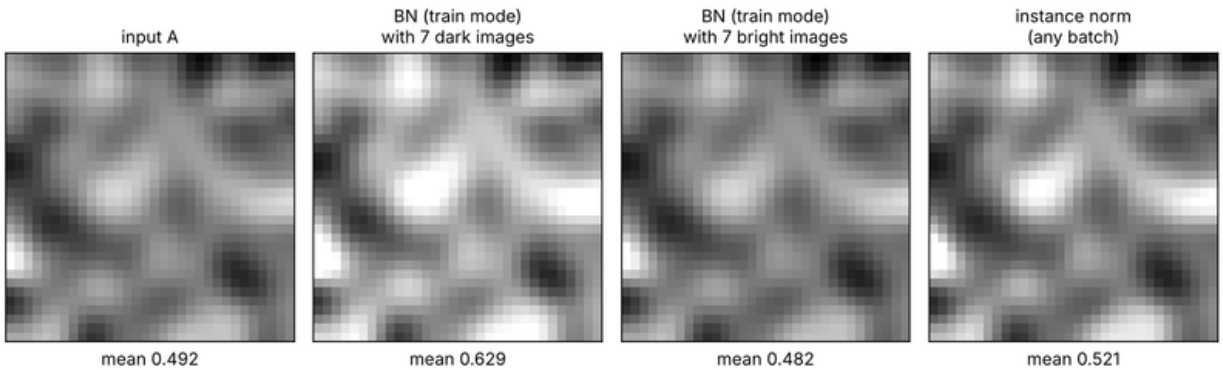


圖 3 — 同一張影像通過同一個未訓練的殘差分支。使用訓練模式的批次正規化時，輸出亮度取決於同批次的其他影像；使用逐樣本正規化則不會。

NAFNet 的消融實驗顯示了另一面：在它的基本區塊加上 LayerNorm，使它能以大 10 倍的學習率訓練，並在 SIDD 去雜訊基準 [31] 上讓 PSNR [30] 提高 0.44 dB，在 GoPro 去模糊基準 [32] 上提高 3.39 dB [19]。正規化對復原網路也有幫助，只要它沒有拿走輸出所需要的資訊。

活化函數與閘

白話版。非線性函數讓堆疊的層能做的事不只是一個線性映射。傳統的選擇是對每個數字套用固定的函數（ReLU、GELU）。閘則是把兩個學出來的特徵**相乘**，讓其中一個決定另一個能通過多少。現代的復原區塊使用閘，而 NAFNet 更是什麼都不用，只用閘。

精確版。 ReLU、GELU 與 SwiGLU 前饋區塊 [33] 的定義見 ML 02。Dauphin 等人的閘控線性單元（gated linear unit，GLU）計算 $f(X) \odot \sigma(g(X))$，其中 f、g 是兩個線性映射，σ 是 sigmoid [34]。NAFNet 的 **SimpleGate** 拿掉了 sigmoid 與第二個映射 [19]：

$$\mathrm{SimpleGate}(X) = X_1 \odot X_2, \qquad X = [X_1; X_2],$$

其中 X 是有 $2C$ 個通道的特徵圖，X_1 與 X_2 分別是它的前 C 個與後 C 個通道，$\odot$ 是逐元素相乘。輸入的兩個線性函數相乘後是二次的，所以區塊不需要任何活化函數也是非線性的；作者指出，GELU 本身 $x\,\Phi(x)$（Φ 是標準常態分布的累積分布函數 [35]）就是閘的一個特例。Restormer 的 GDFN 使用 $\phi(W_d^1 W_p^1 \,\mathrm{LN}(X)) \odot W_d^2 W_p^2 \,\mathrm{LN}(X)$，其中 ϕ 是 GELU，兩條路徑各由 1×1 與 3×3 深度卷積組成，通道擴張係數為 $\gamma = 2.66$ [21]。在 NAFNet 的消融實驗中，把 GELU 換成 SimpleGate，在 SIDD 上多了 +0.08 dB，在 GoPro 上多了 +0.41 dB [19]。

上取樣：轉置卷積、pixel shuffle 與 resize-convolution

白話版。上取樣讓特徵圖變大。常見的做法有三種：讓每個輸入像素把一塊學出來的圖樣「畫」到輸出上（轉置卷積）；每個輸入像素算出幾個值再重新排列（pixel shuffle）；或是先放大、再做卷積（resize-convolution）。前兩種很容易畫出規則的棋盤格圖樣，第三種不會。

精確版。這三種方法與 pixel shuffle 的確切索引規則，見 DLCV 09 與 DLCV 09A [36]。對組合來說，重要的是一個計數論證。步幅為 s 時，輸出像素分成 s^2 種**相位（phase）**（它們的列與行位置除以 s 的餘數）。在轉置卷積中，某個相位的輸出只會接收到卷積核的一部分權重。平坦輸入在這個相位上的值，等於這些權重的總和，也就是**相位和（phase sum）**。若各相位和不相等，平坦的輸入就會變成週期性的圖樣。其中一個成因，Odena 等人稱為**不均勻重疊（uneven overlap）**：卷積核大小不能被步幅整除時，各相位接收到的權重數量不同 [37]。即使大小可以整除，隨機權重也會讓各相位和不相等，所以剛初始化的層仍然會產生圖樣；Aitken 等人把這兩項，加上損失函數中的步幅運算，列為棋盤格偽影的三個來源 [38]。子像素卷積有同樣的自由度，因為每個相位都有自己的濾波器。Resize-convolution（先用最近鄰上取樣，再做普通卷積）在每個輸出像素都套用同一個完整的卷積核，所以所有相位看到的總和相同。下面的程式碼把一個平坦的 16×16 輸入上取樣 2 倍。

```
import numpy as np
from scipy.ndimage import correlate

rng = np.random.default_rng(0)

def transposed_conv(x, w, s):
    """Each input pixel 'paints' a scaled copy of the k x k kernel w; copies are s pixels
     ↪  apart."""
    k, (H, W) = w.shape[0], x.shape
    out = np.zeros((s * (H - 1) + k, s * (W - 1) + k))
    for i in range(H):
```

```
        for j in range(W):
            out[s * i:s * i + k, s * j:s * j + k] += x[i, j] * w
    return out

def resize_conv(x, w, s):
    """Nearest-neighbour upsampling by s, then an ordinary 'same' convolution with w."""
    return correlate(np.kron(x, np.ones((s, s))), w, mode='nearest')

def subpixel_conv(x, ws, s):
    """s*s ordinary convolutions whose outputs are interleaved (pixel shuffle)."""
    out = np.zeros((s * x.shape[0], s * x.shape[1]))
    for n, w in enumerate(ws):
        out[n // s::s, n % s::s] = correlate(x, w, mode='nearest')
    return out

x = np.ones((16, 16))                          # flat input: a good upsampler returns a flat
 ↪  image
cases = {
    'transposed k=3, s=2, uniform w': transposed_conv(x, np.ones((3, 3)), 2)[4:-4, 4:-4],
    'transposed k=4, s=2, uniform w': transposed_conv(x, np.ones((4, 4)), 2)[4:-4, 4:-4],
    'transposed k=4, s=2, random w ': transposed_conv(x, rng.uniform(0.5, 1.5, (4, 4)),
     ↪  2)[4:-4, 4:-4],
    'sub-pixel, 4 random 3x3 kernels': subpixel_conv(x, rng.uniform(0.5, 1.5, (4, 3, 3)),
     ↪  2)[4:-4, 4:-4],
    'resize + conv, random 3x3 w   ': resize_conv(x, rng.uniform(0.5, 1.5, (3, 3)), 2)[4:-4,
     ↪  4:-4],
}
for name, y in cases.items():
    phases = [y[a::2, b::2].mean() for a in (0, 1) for b in (0, 1)]     # the four 2x2
     ↪  sub-pixel phases
    print(f"{name}: phase means {np.round(phases, 2)}  relative spread {np.ptp(y) /
     ↪  y.mean():.2f}")
```

```
transposed k=3, s=2, uniform w: phase means [4. 2. 2. 1.]  relative spread 1.30
transposed k=4, s=2, uniform w: phase means [4. 4. 4. 4.]  relative spread 0.00
transposed k=4, s=2, random w : phase means [4.04 3.22 5.01 3.85]  relative spread 0.44
sub-pixel, 4 random 3x3 kernels: phase means [ 8.71 10.13  9.49  8.91]  relative spread 0.15
resize + conv, random 3x3 w   : phase means [7.86 7.86 7.86 7.86]  relative spread 0.00
```

$k = 3$、$s = 2$ 時，就算權重一致，也是一個相位接收 4 個權重、兩個接收 2 個、一個接收 1 個：4 : 2 : 2 : 1 的圖樣是由形狀本身決定的。$k = 4$ 時每個相位都接收 4 個權重，所以一致的權重會得到平坦的輸出，隨機權重則否。只有 resize-convolution 在任何權重下都是平坦的。圖 4 把同樣四種上取樣器用在平坦輸入與一張照片上。訓練可以把各相位和拉近，Odena 等人報告他們最好的結果來自最近鄰 resize-convolution [37]；對 pixel shuffle 來說，讓所有相位都從同一個卷積核初始化，可以在一開始就消除圖樣 [38]，如 DLCV 09A 所述。

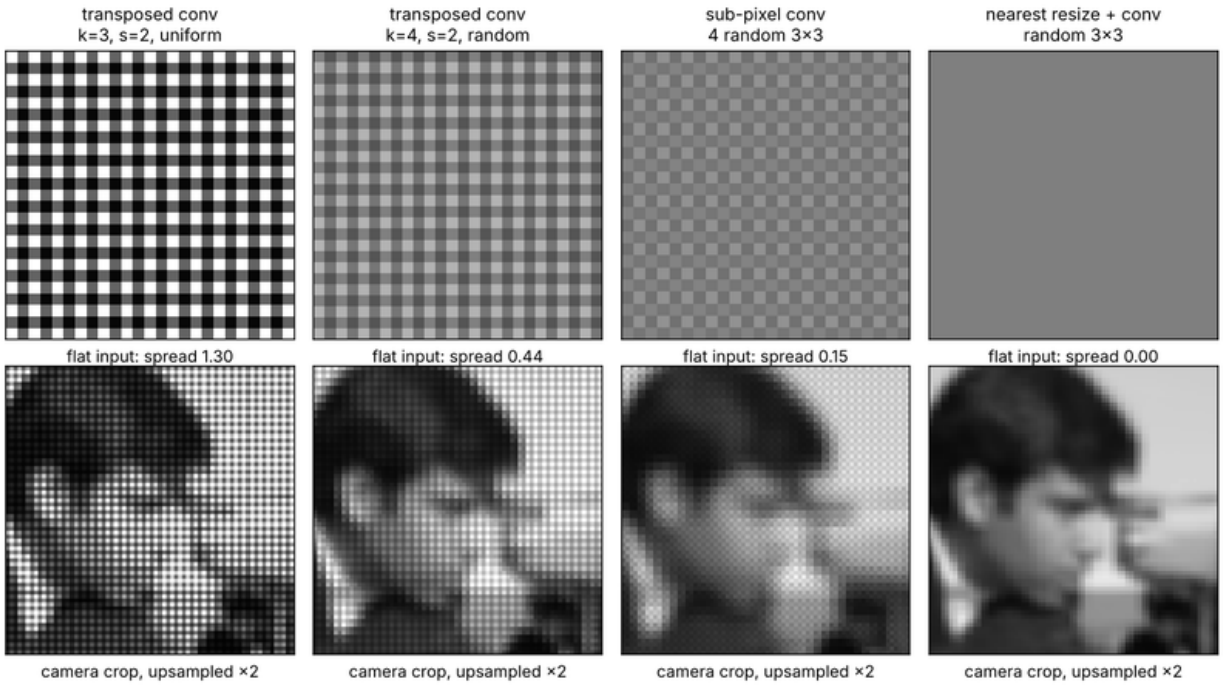


圖 *4* — 未訓練的上取樣器產生的棋盤格偽影。上：平坦輸入（相對差距印在下方）。下：用同樣的卷積核把 *camera* 影像的裁切區塊上取樣 2 倍，並重新縮放成單位平均增益。只有 *resize-convolution* 由構造本身就沒有這種圖樣。

僅線上版：一維轉置卷積的相位和 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/notes/一維轉置卷積的相位和/*

下取樣也有對應的選擇：步幅卷積（ResNet）、池化、獨立的步幅 2 的 2×2 卷積（ConvNeXt、NAFNet [14]）、patch merging（Swin）與 pixel unshuffle（Restormer [21]）。ConvNeXt 報告說，它獨立的 2×2 下取樣層一開始會讓訓練發散，直到它在每個空間解析度改變的地方加上正規化層為止 [2]。

跳接：相加或串接

白話版。跳接把較早的特徵圖帶到較後面的層。它可以**相加**（兩張圖形狀必須相同，結果維持原來的寬度），也可以**串接**（把兩張圖沿通道疊起來，由下一層決定怎麼組合）。串接比較有彈性，也比較貴。

精確版。串接後再接一個 1×1 卷積，已經把相加包含為特例：設編碼器特徵圖為 e、解碼器特徵圖為 d，各有 C 個通道，那麼權重矩陣為 $[I \;\; I]$（兩個 $C \times C$ 的單位矩陣並排）的 1×1 卷積，會把 $[e; d]$ 映射成 $e + d$。所以串接能學到相加能學到的一切，還能學更多，代價是更寬的層，以及必須儲存兩張圖。下面的程式碼檢驗這個等式，並在 256×256 輸入、寬度由 32 到 256 的 4 層 U-Net 解碼器中，數出每種合併方式之後第一個卷積的成本。

```
import numpy as np
```

```
rng = np.random.default_rng(0)

# 1) Concatenation followed by a 1x1 convolution contains addition as a special case.
C, H, W = 8, 5, 5
enc, dec = rng.standard_normal((C, H, W)), rng.standard_normal((C, H, W))
cat = np.concatenate([enc, dec])                        # (2C, H, W)
W_fuse = np.hstack([np.eye(C), np.eye(C)])              # a 1x1 conv is a (C_out x C_in)
 ↪  matrix
print("concat + 1x1 conv [I I] == add:", np.allclose(np.tensordot(W_fuse, cat, 1), enc +
 ↪  dec))

# 2) What each choice costs in the decoder of a 4-level U-Net on a 256 x 256 input.
widths, size = [32, 64, 128, 256], 256
totals = {'add': [0, 0], 'concat': [0, 0], 'concat + 1x1': [0, 0]}
for level, c in enumerate(widths):
    hw = (size // 2 ** level) ** 2                      # pixels at this level
    first = {'add': 9 * c * c,                          # first 3x3 conv after the merge
             'concat': 9 * 2 * c * c,
             'concat + 1x1': 2 * c * c + 9 * c * c}     # 1x1 halves channels, then 3x3
    for k, p in first.items():
        totals[k][0] += p
        totals[k][1] += p * hw
for k, (p, m) in totals.items():
    print(f"{k:13s} merge-layer weights {p:7d}   MACs {m / 1e9:.2f} G")
```

```
concat + 1x1 conv [I I] == add: True
add           merge-layer weights  783360   MACs 2.42 G
concat        merge-layer weights 1566720   MACs 4.83 G
concat + 1x1  merge-layer weights  957440   MACs 2.95 G
```

串接會讓合併層的權重與 MAC 加倍；像 Restormer 那樣先做一次 1×1 融合，可以拿回大部分省下的成本。著名設計之間的選擇並不一致：

- **U-Net** 在把通道減半的 2×2 上卷積之後，將（裁切過的）編碼器特徵圖與解碼器特徵圖串接 [39]。
- **FPN** 用相加：較粗的特徵圖以最近鄰內插上取樣 2 倍，加上經 1×1 投影的骨幹特徵圖，再對每個合併後的特徵圖做 3×3 卷積，以減少上取樣造成的混疊 [5]（DLCV 03）。
- **Restormer** 串接之後，在除了最上層以外的所有層級，用 1×1 卷積把通道減半 [21]。
- **NAFNet** 的程式碼把編碼器特徵圖加到上取樣後的解碼器特徵圖上 [14]。
- **DenseNet** 把一個區塊內所有先前各層的輸出串接起來 [40]（DLCV 02 補充資料）。

多尺度模組：FPN、ASPP 與 U-Net

白話版。物體和退化都有大有小。多尺度模組讓一個網路同時看好幾種尺度：建立特徵圖金字塔（FPN）、用好幾種大小的濾波器探測同一張特徵圖（ASPP），或先往下再往上走（U-Net）。

這些與其說是積木，不如說是組合模式，每一個都在用到它的地方教過：FPN 在 DLCV 03 [5]；空洞空間金字塔池化（atrous spatial pyramid pooling，ASPP），也就是以多種擴張率平行進行的

空洞卷積，在 DLCV 04 [41]；U-Net 在 DLCV 01 與 IP 10 論文導讀 [39]。有一條規則把它們串在一起：**脈絡來自往粗的方向走，細節來自一直保持細緻的路徑**。FPN 的橫向連接、U-Net 的跳接，以及 DLCV 03B 中分割解碼器的低階跳接，都是那第二條路徑。

為分類而組合

白話版。分類器要回答的是關於整張影像的一個問題。所以它很快地縮小影像，把運算花在小而寬的特徵圖上（那裡每個位置已經看得到很大的區域），最後把空間格點整個丟掉。

原則，以及各自的理由：

1. **用 stem 提早下取樣**。最前面的層要處理最多的像素，所以 stem 在主體開始之前就把解析度降到 1/4（ResNet 的步幅 2 的 7×7 卷積加最大池化 [1]；ConvNeXt 的步幅 4 的 4×4 patchify 層，在它的路線圖中沒有損失準確率 [2]）。
2. **每個 stage 解析度減半、寬度加倍**。這讓每層的成本大致維持不變（見解剖一節），並讓感受野以等比方式成長。
3. **把大多數區塊放在第三個 stage**。 ResNet-50 每個 stage 的區塊數是 (3, 4, 6, 3)；ConvNeXt 依循 Swin 的 1:1:3:1 比例，改成 (3, 3, 9, 3)，準確率因此提高 [2]。第三個 stage 的輸出步幅是 16，感受野夠大，位置的數量也還夠。
4. **全域池化，再做線性分類**。全域平均池化讓輸出不受物體位置影響，也讓網路能接受任意大小的輸入。ViT 論文發現 class token 與全域平均池化的效果相近，但需要不同的學習率 [3]。
5. **把昂貴的全域混合放在 token 少的地方**。全域注意力在後面的 stage 才負擔得起；混合型設計把卷積放前面、注意力放後面（ML 05）。像 SE 這種便宜的閘可以放進每個區塊。
6. **批次大的時候，用批次正規化沒問題**；批次小時改用 GN 或 LN。

感受野長得多快？ ML 03 的遞迴式涵蓋了步幅堆疊的情況；下面的程式碼也處理上取樣：從一個輸出單元往回走，追蹤能影響它的輸入區間。它在 256×256 的輸入上比較 ResNet-50 與兩種復原網路：EDSR-baseline 形狀的一般全解析度主體（16 個殘差區塊，每個有兩個 3×3 卷積、64 個通道，再加頭尾），以及寬度從 32 到 512 的 4 層 U-Net。

```
def analyze(layers, H=256):
    """Exact 1-D receptive field of one output unit, plus the multiply-accumulates (MACs)
    of a square H x H input. A layer is ('conv', k, stride, c_in, c_out) or ('up', f),
    where 'up' is nearest-neighbour upsampling by f (the convolution after it pays the
     ↪  cost)."""
    macs, size = 0, H
    for layer in layers:                                # forward: track the map size and cost
        if layer[0] == 'conv':
            _, k, s, cin, cout = layer
            size //= s
            macs += size * size * k * k * cin * cout
        else:
            size *= layer[1]
    rf = 0
    for p0 in range(16):                                # a few output positions (alignment
     ↪  matters)
        lo = hi = p0
        for layer in reversed(layers):                  # backward: which inputs reach unit p0?
```

```
            if layer[0] == 'conv':
                _, k, s, _, _ = layer
                pad = (k - 1) // 2
                lo, hi = s * lo - pad, s * hi - pad + k - 1
            else:
                lo, hi = lo // layer[1], hi // layer[1]
        rf = max(rf, hi - lo + 1)
    return rf, macs, size

# Classification: ResNet-50 (stride on the 3x3 conv of each first block, max-pool as a k=3,
 ↪  s=2 layer).
resnet = [('conv', 7, 2, 3, 64), ('conv', 3, 2, 64, 0)]        # stem conv, then max-pool (no
 ↪  MACs)
cin = 64
for width, blocks, stride in [(64, 3, 1), (128, 4, 2), (256, 6, 2), (512, 3, 2)]:
    for b in range(blocks):
        s = stride if b == 0 else 1
        resnet += [('conv', 1, 1, cin, width), ('conv', 3, s, width, width),
                   ('conv', 1, 1, width, 4 * width)]
        if b == 0:
            resnet += [('conv', 1, 1, cin, 4 * width)]          # projection shortcut (cost
             ↪  only)
        cin = 4 * width

# Restoration A: plain full-resolution body, 64 channels, 32 convs of 3x3
 ↪  (EDSR-baseline-like).
plain = [('conv', 3, 1, 3, 64)] + [('conv', 3, 1, 64, 64)] * 32 + [('conv', 3, 1, 64, 3)]

# Restoration B: 4-level U-Net, 2 convs per level, widths double as resolution halves.
w = [32, 64, 128, 256, 512]
unet = [('conv', 3, 1, 3, w[0])]
for i in range(4):
    unet += [('conv', 3, 1, w[i], w[i])] * 2 + [('conv', 2, 2, w[i], w[i + 1])]
unet += [('conv', 3, 1, w[4], w[4])] * 2
for i in reversed(range(4)):
    unet += [('up', 2), ('conv', 1, 1, w[i + 1], w[i]),
             ('conv', 3, 1, 2 * w[i], w[i]), ('conv', 3, 1, w[i], w[i])]   # concat skip
              ↪  doubles c_in
unet += [('conv', 3, 1, w[0], 3)]

for name, net in [('ResNet-50', resnet), ('plain 64ch x 34', plain), ('U-Net 4 levels',
 ↪  unet)]:
    rf, macs, size = analyze(net)
    print(f"{name:16s} RF {rf:4d} px   output {size:3d}x{size:<3d}   {macs / 1e9:6.2f}
     ↪  GMACs")
```

```
ResNet-50        RF  427 px   output   8x8       5.34 GMACs
plain 64ch x 34  RF   69 px   output 256x256    77.54 GMACs
U-Net 4 levels   RF  206 px   output 256x256    14.47 GMACs
```

U-Net 的跳接在計算中省略了，因為最深的那條路徑決定了感受野（對所有路徑暴力檢查，得到的同樣是 206）。圖 5 畫出成長的過程。ResNet-50 用 5.3 GMAC 達到 427 像素，但回傳的是 8×8

的特徵圖；一般的復原網路每層只多 2 個像素，用 77.5 GMAC 才有 69 像素；U-Net 在全輸出解析度下達到 206 像素，運算量不到前者的五分之一。這些是理論上的感受野；用梯度量得的有效感受野較小，而且集中在中央 [42]（DLCV 01）。

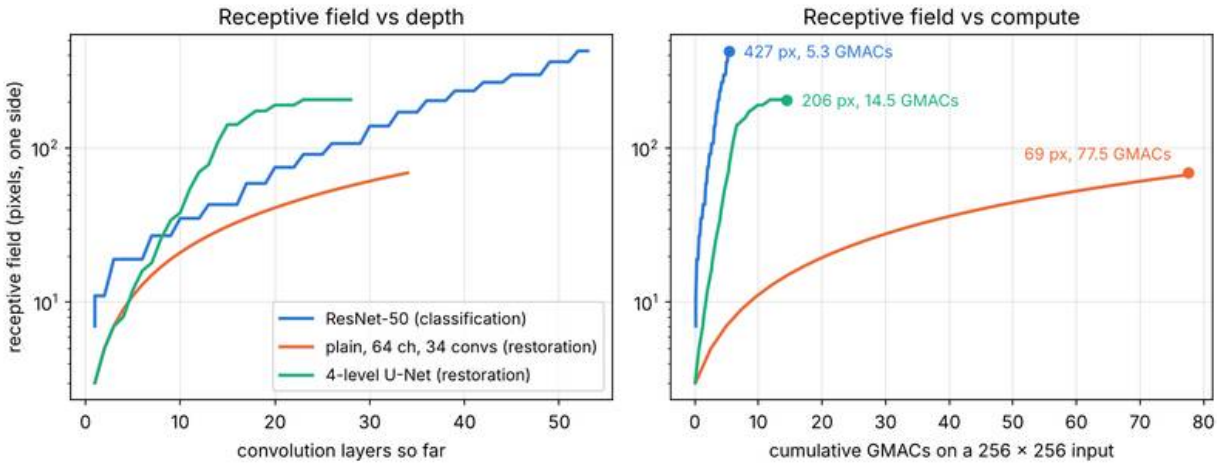


圖 5 — 程式碼中三個網路的感受野（單邊、理論值）隨深度與運算量變化的情形。下取樣便宜地買到感受野；全解析度的主體則要為每個像素付費。

僅線上版：經過上取樣與跳接的感受野 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/notes/經過上取樣與跳接的感受野/*

為復原而組合

白話版。復原網路要回傳一張大小相同（超解析度則是更大）、而且幾乎就是輸入的影像。所以它保持解析度，或藉由跳接回到原解析度；它只預測修正量；它避免會改變強度範圍的層；它取得脈絡，卻不必在每一層都為全解析度付費。

原則：

1. **保持解析度，或回到原解析度**。 X-Restormer 背後的比較研究把復原網路分成三種架構：U 形編碼器–解碼器（如 Restormer 與 NAFNet）、保持原始大小的一般 residual-in-residual 網路（RCAN、SwinIR），以及多階段漸進式網路（用好幾個子網路依序處理影像）[22]。它形容一般設計有利於重建高頻細節但很昂貴，U 形設計則因為下取樣而讓感受野快速成長，能處理大的輸入 [22]。圖 1(b) 與 (c) 畫出兩種主要的形狀。
2. **學殘差**。輸出是輸入加上預測的修正量，$\hat{x} = y + R(y)$，其中 y 是退化的輸入，R 是網路。VDSR 預測的是高解析度影像與其內插後輸入之間的殘差 [43]；DnCNN 預測雜訊 [29]；SwinIR 對不需要上取樣的任務使用 $I_{\text{HQ}} = H(I_{\text{LQ}}) + I_{\text{LQ}}$ [4]；Restormer 輸出殘差影像 R 並回傳 $\hat{I} = I + R$ [21]；NAFNet 的程式碼把輸入加到最後一個卷積的輸出上 [14]。修正量很小、以零為中心，所以不必把容量浪費在複製影像上。
3. **為低頻加一條長跳接**。 SwinIR 把淺層特徵直接送到重建模組，$H_{\text{REC}}(F_0 + F_{\text{DF}})$，因為淺層特徵主要含有低頻，深層特徵就能專心處理遺失的高頻 [4]。EDSR 與 RCAN 在主體或群組外圍使用長跳接 [12]、[15]。

4. **保住強度範圍**。主體不用 BN（EDSR、ESRGAN），正規化只放在殘差分支內（NAFNet、Restormer、SwinIR），如正規化一節所示 [12]、[28]。
5. **讓感受野配合退化，並且花得聰明**。X-Restormer 的研究主張，超解析度主要依賴局部資訊與強的空間映射，U 形的下取樣與上取樣對去雜訊有幫助，而動態去模糊與去霧需要大範圍或全域的資訊 [22]。感受野的程式碼顯示了代價：全解析度的主體每層只多幾個像素的感受野，卻要付全額成本；下取樣讓感受野快速且便宜地成長；通道注意力或狀態空間掃描則以線性成本涵蓋整張影像 [21]、[44]。
6. **依目標設計輸出層**。最後一層是普通卷積（SwinIR 在去雜訊時的重建、Restormer 的最後一個卷積、NAFNet 的結尾層 [4]、[21]、[14]）；在那裡接 ReLU 或 sigmoid 會截掉修正量。超解析度則把上取樣器放在最後，讓主體在低解析度下執行（DLCV 09A）。
7. **為完整大小的測試影像做準備**。復原網路用裁切區塊訓練，卻在整張影像上測試。U-Net 需要大小能被 2^{levels} 整除，NAFNet 的程式碼用補邊處理這件事 [14]。全域統計量（例如通道注意力的池化）在訓練時涵蓋的是一個裁切區塊，測試時涵蓋的卻是整張影像；NAFNet 指出這會降低效能，並採用先前研究的補救辦法 [19]，Restormer 則以逐步加大的 patch 訓練 [21]。

僅線上版：*Chen 等人：A Comparative Study of Image Restoration Networks for General Backbone Network Design（X-Restormer，ECCV 2024）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/papers/chen-等人 a-comparative-study-of-image-restoration-networks-for-general-backbone-network-designx-restormereccv-2024/*

並排比較：改了什麼、為什麼

白話版。看出組合如何運作最快的方法，是把一個分類模型和由它衍生的復原模型並排放，列出差異。每項差異都有上面兩份清單中的理由。

ResNet 對 EDSR 與 NAFNet

元件	ResNet [1]	EDSR [12]	NAFNet [19]、[14]	為什麼改
Stem	步幅 2 的 7 × 7 + 最大池化	一個 3 × 3 卷積	一個 3 × 3 卷積	保留每個像素
解析度	4 個 stage，輸出步幅 32	固定（低解析度輸入的大小）	U 形，下取樣用步幅 2 的 2×2 卷積，上取樣用 1 × 1 卷積 + pixel shuffle	細節與脈絡的取捨
區塊	conv-BN-ReLU-conv-BN，相加，ReLU	conv-ReLU-conv，×0.1（大模型），相加	LN、1 × 1、3 × 3 dw、SimpleGate、SCA、1 × 1、×β、相加；然後是閘控 MLP、×γ、相加	保住範圍；便宜的閘控
正規化	BN	無	分支內使用 LayerNorm	見正規化一節
跳接	只有短跳接	短跳接 + 繞過主體的長跳接	短跳接 + 層級之間的相加跳接 + 全域殘差	低頻繞過主體
Head	全域平均池化 + FC	取樣器（卷積 + pixel shuffle）+ 卷積	3 × 3 卷積 + 輸入	輸出是影像

元件	ResNet [1]	EDSR [12]	NAFNet [19]、[14]	為什麼改
損失	交叉熵	L1	消融實驗中用 PSNR 損失	迴歸目標

EDSR 自己的表列出了從 SRResNet 出發的步驟：兩個基準模型都是 16 個區塊、每個 64 個濾波器，移除 BN，用 L1 取代 L2，完整模型則是 32 個區塊、每個 256 個濾波器，加上 0.1 的殘差縮放，參數量從 1.5M 增加到 43M [12]。

Swin 對 SwinIR

元件	Swin [6]	SwinIR [4]	為什麼改
輸入	4×4 patch 切分與線性嵌入	全解析度的 3×3 卷積（淺層特徵）	每個像素一個 token
階層	4 個 stage 之間做 patch merging	沒有：6 個殘差 Swin Transformer 區塊（RSTB），解析度固定	復原需要每個像素
視窗	$M = 7$	$M = 8$（JPEG 偽影移除用 7）	JPEG [45] 以 8×8 為單位編碼；作者觀察到用 8 時表現下降
區塊群組	Swin 層	每個 RSTB 含 6 個 Swin 層 + 一個 3×3 卷積 + 一條殘差連接	卷積帶來平移等變性；殘差有助於聚合
Head	全域平均池化 + 線性層	作用在 $F_0 + F_{\text{DF}}$ 上的卷積（SR 另加 pixel shuffle）	低頻繞過主體
寬度	96 個通道以上	180 個通道，6 個頭	在準確率與大小之間取得平衡

SwinIR 的消融實驗報告，每個 RSTB 內的殘差連接使 PSNR [30] 提高 0.16 dB，而在那裡改用 1×1 卷積幾乎沒有進步，可能是因為它不像 3×3 卷積那樣能擷取局部相鄰的資訊 [4]。JPEG 視窗大小是一個很好的例子，說明組合上的選擇會與資料互相影響：作者推測 $M = 8$ 時的下降，可能來自 JPEG 的 8×8 區塊切分。

ViT 對 Restormer

元件	ViT [3]	Restormer [21]	為什麼改
Token	16×16 的 patch	每個像素（經卷積嵌入）	不丟掉任何資訊
Token 混合器	對 patch 的全域自注意力，$N \times N$ 的圖	MDTA：通道注意力，$C \times C$ 的圖，搭配深度卷積	成本隨像素線性增加，仍有全域涵蓋
通道混合器	MLP，4 倍擴張，GELU	GDFN：閘控、深度卷積、擴張 2.66	閘控與局部脈絡
形狀	等向，單一解析度	4 層 U 形，pixel-unshuffle 下取樣，pixel-shuffle 上取樣，串接 + 1×1 的跳接，加上精修 stage	以可負擔的成本做多尺度

元件	ViT [3]	Restormer [21]	為什麼改
Head	class token + MLP	卷積成殘差影像，加到輸入上	輸出是影像
訓練	固定解析度	逐步加大的 patch，從 128×128 開始	從更大的區域學習

Restormer 的設定從第 1 層到第 4 層使用 `[4, 6, 6, 8]` 個區塊、`[1, 2, 4, 8]` 個頭與 `[48, 96, 192, 384]` 個通道，另外精修 stage 有 4 個區塊 [21]。兩欄都在 pre-norm 的殘差分支裡使用 LayerNorm：Transformer 區塊本身保留了下來；改變的是 token 混合器，以及它周圍的整體架構。

探索設計空間

白話版。設計上的問題（「區塊 X 有幫助嗎？」）沒有一般性的答案；它只在一組特定的其他選擇、資料與預算下才有答案。所以一次只改一件事，固定預算，看不只一個任務或隨機種子，而且可以的話，比較整個模型族群，而不是只看單一優勝者。

讀一份消融實驗

NAFNet 的論文是可讀性很高的消融實驗範本。它從 U-Net 中由卷積、ReLU 與捷徑組成的基本區塊出發，在 16 GMAC 的固定預算下，一次加入一項改動，並報告在 SIDD [31]（真實相機去雜訊）與 GoPro [32]（動態去模糊）上的 PSNR [19]：

步驟	SIDD (dB)	GoPro (dB)
PlainNet	39.29	28.51
+ LayerNorm（以及大 10 倍的學習率）	39.73	31.90
ReLU → GELU	39.71	32.11
+ 通道注意力	39.85	32.35
GELU → SimpleGate	39.93	32.76
CA → 簡化 CA	39.96	32.85

讀這類表格時要檢查：

- **預算固定嗎？**這裡是固定的（16 GMAC），所以進步不只是「多用了運算」。
- **步驟是累積的嗎？**是的：每一列都包含前面所有的改動，所以這張表衡量的是每項改動在**前面改動已經存在的前提下**的效果，而不是單獨的效果。換個順序，增量可能不同。
- **各任務一致嗎？**不一定：GELU 在 SIDD 上沒有差別，在 GoPro 上卻有幫助。對某種退化重要的改動，對另一種不一定重要。
- **訓練配方與區塊混在一起了嗎？** LayerNorm 那一列同時改了學習率，因為 LayerNorm 讓較大的學習率變得穩定。作者明確說出了這一點；許多論文沒有。
- **雜訊有多大？**每列只跑一次，就沒有誤差線。差幾百分之一 dB 的差異，應該讀成「沒有損失」，論文對最後一列就是這樣解讀的。

僅線上版：*Chen* 等人：*Simple Baselines for Image Restoration*（*NAFNet*，*ECCV 2022*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/papers/chen-等人 simple-baselines-for-image-restorationnafneteccv-2022/*

ConvNeXt 的路線圖對分類也是同樣的讀法：ResNet-50 光靠現代訓練配方，ImageNet top-1 就從 76.1% 提高到 78.8%，再經過一連串架構改動提高到 82.0%，其中最大的單項改動所增加的，還不到配方的貢獻 [2]。這就是為什麼用不同配方訓練的架構不能直接比較（DLCV 01A）。

僅線上版：*ConvNeXt* 路線圖，一步一步 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/notes/convnext-路線圖一步一步/*

比較族群：設計空間

RegNet 把「哪個單一模型最好？」換成「哪個**設計空間**更常產生好模型？」[16]。設計空間是由幾個參數（每個 stage 的區塊數、寬度、瓶頸比例、群組寬度）定義的一組模型。使用的工具是**誤差經驗分布函數（error empirical distribution function）**：

$$F(e) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}[e_i < e],$$

其中 e_i 是從該空間取樣並訓練的 n 個模型中第 i 個的誤差，$\mathbf{1}[\cdot]$ 在其引數為真時等於 1、否則為 0，$F(e)$ 是誤差低於 e 的模型所占的比例。曲線位置較高的空間，整體而言比較好，這比比較兩個手工調整的模型穩健得多。RegNet 取樣了 $n = 500$ 個模型，運算量為 3.6 億到 4 億 FLOPs，每個訓練 10 個 epoch，並保留不會損害曲線（共用的瓶頸比例與群組寬度）或能改善曲線（寬度與深度隨 stage 遞增）的限制 [16]。最好的模型在各種運算量級下都有約 20 個區塊，不用瓶頸，而且受益於 SE [16]。

僅線上版：*Radosavovic* 等人：*Designing Network Design Spaces*（*RegNet*，*CVPR 2020*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/papers/radosavovic-等人 designing-network-design-spacesregnetcvpr-2020/*

僅線上版：*RegNet* 的量化線性寬度 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/notes/regnet-的量化線性寬度/*

實用流程

1. 在比較區塊之前，先固定預算（MAC、參數量、記憶體或實測延遲）與訓練配方。
2. 一次只改一件事，並記錄每一次執行，包括沒有幫助的。
3. 在不只一個任務、資料集或隨機種子上評估；報告變異範圍。
4. 在目標模型大小上確認；對很小的模型有幫助的區塊，對大模型未必有幫助。
5. 做架構搜尋時，用 NAS 綜述的三個維度來思考：搜尋空間、搜尋策略，以及如何估計效能 [46]。便宜的估計（如短時間訓練，RegNet 的 10 個 epoch）適合用來為族群排序，但應該用完整訓練加以確認。

常見的組合錯誤

白話版。多數架構上的錯誤不是積木選錯，而是對的積木放在錯的地方。下面的清單列出症狀、原因與解法。

- **在復原的主體使用 BN**。症狀：亮度或顏色隨批次飄移，或測試時出現偽影。解法：拿掉 BN，或在殘差分支內做逐樣本的正規化，並搭配全域殘差。
- **每個裝置的批次很小，卻用 BN**（偵測、分割、高解析度）。解法：GroupNorm，或凍結預訓練的 BN 統計量（DLCV 01A）。
- **轉置卷積的卷積核大小不能被步幅整除**。症狀：棋盤格紋理。解法：resize-convolution，或初始化一致的 pixel shuffle。
- **復原網路沒有全域殘差**。網路必須先學會複製影像，才能修正它。解法：預測修正量。
- **輸出端接 ReLU 或 sigmoid**。修正量不能是負的，或亮部飽和。解法：最後一層用線性層；只在顯示與評估時才截斷。
- **感受野比退化還小**。症狀：大範圍的模糊或霧仍然殘留。解法：加下取樣層級、空洞卷積或全域混合；先計算感受野。
- **復原網路過早下取樣**，或直接沿用分類骨幹：細微的細節遺失。解法：保留一條全解析度路徑（跳接、精修 stage）。
- **在全解析度上做全域注意力**，記憶體會爆掉。改用視窗、通道注意力或下取樣。
- **U-Net 的輸入大小不能被 2^{levels} 整除**，或搭配串接跳接時使用奇數大小。症狀：形狀錯誤或跳接錯位。解法：補邊再裁切，如 NAFNet 的程式碼所做 [14]。
- **把參數量當成速度**。深度卷積與大量串接的設計，即使參數或 MAC 很少，也可能很慢。請實測延遲與記憶體。
- **比較用不同配方或預算訓練的設計**：配方本身的貢獻，可能超過任何單一區塊 [2]。
- **忽略與資料的交互作用**，例如 SwinIR 的視窗大小與 JPEG 的區塊 [4]。

現代觀點

綜述怎麼說

Khan 等人 [47] 回顧 CNN 架構，把其中的創新分成七類：空間利用、深度、多路徑、寬度、特徵圖利用、通道增強與注意力。這套分類對本章很有用，因為大多數類別是組合上的選擇（深度、寬度、多路徑連接），而不是新的層類型。

Guo 等人 [20] 綜述電腦視覺中的注意力，依「選擇什麼」來分類：通道注意力（注意什麼，例如 SE）、空間注意力（注意哪裡，包含自注意力）、時間注意力（注意何時）、分支注意力（注意哪個分支），以及兩種混合：通道加空間（例如 CBAM）與空間加時間。他們把注意力視為依輸入而定的「動態權重調整」，與本章的閘觀點相符。

Khan 等人 [48] 回顧視覺中的 Transformer，從自注意力的基礎與大規模預訓練，談到辨識、偵測、分割、生成、影片與低階視覺的應用，比較常見設計的優缺點，最後提出尚待研究的方向。

Elsken、Metzen 與 Hutter [46] 沿著三個維度整理神經架構搜尋：搜尋空間、搜尋策略與效能估計策略。這份綜述是本章手動探索設計空間之後，很自然的下一步。

Chen 等人 [22] 是比較研究而不是綜述：它在五項任務上對五種復原骨幹網路（包括 SwinIR、Restormer 與 NAFNet）做基準測試，依架構（U 形、一般 residual-in-residual、多階段）與核心

運算（卷積、空間自注意力、轉置自注意力）分類，並發現在某項任務上很強的網路，常常在另一項任務上表現不足。

2023–2026 年的最新進展

- **卷積與注意力收斂到同一種區塊形狀。** ConvNeXt V2 在 ConvNeXt 上加了 Global Response Normalization 層與遮罩自編碼器預訓練 [49]；MetaFormer 的觀點把 token 混合器視為一個可互換的零件 [8]。
- **預訓練和區塊一樣重要。** Battle of the Backbones 在超過 1,500 次訓練中比較了許多預訓練骨幹網路在各項任務上的表現，發現在它考慮的多數任務上，以大型資料集監督式預訓練的卷積網路仍然表現最好，而自監督骨幹網路在對照的比較中具有競爭力 [50]。
- **復原網路混用運算。** HAT 把通道注意力與視窗自注意力結合，並加入視窗之間重疊的交叉注意力 [51]；X-Restormer 把 Restormer 一半的通道注意力區塊換成空間注意力區塊，以強化它的空間映射 [22]；MambaIR 與 MambaIRv2 用能以線性成本涵蓋整張影像的狀態空間掃描取代注意力，加入局部增強與通道注意力，v2 更加入非因果的「注意力式」建模 [44]、[52]。

尚未解決的問題

1. **通用的復原骨幹網路。**沒有任何單一組合在所有退化上都最好；比較研究把「任務通用性」視為尚待達成的目標 [22]。
2. **公平的比較。**架構、訓練配方與預訓練糾纏在一起，而像 RegNet 這種族群層級的研究需要數百次訓練 [16]，對高解析度的復原模型來說太昂貴。
3. **實際的感受野。**理論感受野很容易算；訓練好的復原網路實際用了多少脈絡則難以測量，而歸因分析顯示它可能比設計所允許的少 [51]、[42]。
4. **成本模型。** MAC、記憶體搬運與延遲彼此不一致，尤其是深度卷積、閘控與注意力；設計規則仍然取決於硬體。
5. **架構內部的訓練–測試偏移。**全域統計量（池化、正規化、注意力）在裁切區塊與完整影像上的行為不同，而目前採用的補救辦法改的是訓練或測試流程，而不是架構 [19]、[21]。

僅線上版：*Chen* 等人：*Simple Baselines for Image Restoration*（*NAFNet*，*ECCV 2022*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/building-blocks/papers/chen-等 人 simple-baselines-for-image-restorationnafneteccv-2022/*

重點整理

- 比較個別區塊前，先追蹤解析度變化與跳接路徑。
- 在相同成本下比較空間／通道混合；延遲須與參數量及 MACs 分開量測。
- 分類器常下採樣再池化；復原器則須保留或重建空間細節與強度。
- 殘差學習與長跳接保留輸入的直接路徑，並須檢查輸出層允許的數值範圍。
- 使用批次統計的正規化會引入批次相依性；逐樣本正規化仍可能移除有用的強度資訊。
- 把偽影歸咎於訓練資料前，先檢查上採樣相位平衡、跳接對齊與填補。
- 區分區塊與訓練配方的影響，並比較設計家族，而不只比較單一最高分模型。

練習

1. **減半與加倍**。一個 stage 在 56×56 的特徵圖上，執行 128 個通道的 3×3 卷積。(a) 這樣的一層要花多少 MAC？(b) 下一個 stage 把特徵圖減半、寬度加倍；現在一層要花多少？(c) 如果改成把寬度變成三倍，成本會怎麼變？

   提示

   (a) $9 \cdot 56^2 \cdot 128^2 = 462{,}422{,}016$ 次 MAC。(b) $9 \cdot 28^2 \cdot 256^2$ 是同一個數字。(c) 通道數變成 $3 \cdot 128 = 384$ 時，成本是 $9 \cdot 28^2 \cdot 384^2$，也就是 (a) 的 $9/4$：寬度以平方進入，面積則是線性。

2. **區塊預算**。使用 `blocks` 函數，在 $C = 64$ 時，找出擴張倍率 t 為多少時，輸入與輸出都是 C 個通道的反向殘差，成本會和基本殘差區塊相同。接著找出寬度 C' 為多少時，$t = 6$ 的反向殘差成本會和 $C = 64$ 的基本區塊相同。這對「在相同寬度下比較區塊」說明了什麼？

   提示

   令 $t(2C^2 + 9C) = 18C^2$；$C = 64$ 時得到 $t = 18 \cdot 64/(2 \cdot 64 + 9) \approx 8.4$。第二個問題要解 $12C'^2 + 54C' = 73{,}728$，得到 $C' \approx 76$。反向殘差在相同寬度下並不「便宜」；它之所以便宜，是因為 MobileNet 把它用在細的張量上。

3. **批次大小為一的 BN**。在正規化的程式碼中，用 `'bn-train'` 單獨復原 A（批次大小為 1）。把輸出平均值與 `'instance'` 的結果比較，並解釋批次大小為 1 的 BN 與實例正規化之間的關係。接著在批次中加入 32 張隨機的中灰度影像，看看訓練模式與評估模式能靠得多近。

   提示

   $N = 1$ 時，BN 在 (N, H, W) 上的統計量，正好就是逐樣本、逐通道的統計量，所以結果等於實例正規化（0.521）。當有許多來自訓練分布的影像時，批次統計量會趨近移動平均，輸出也趨近評估模式的值。

4. **讓轉置卷積變平坦**。對 $k = 4$、$s = 2$，寫出 16 個卷積核權重要滿足什麼條件，平坦輸入才會得到平坦輸出（邊界以外）。用一個隨機的 2×2 卷積核構造出這樣的卷積核，用上取樣程式碼檢查，並把它和 resize-convolution 連起來看。

   提示

   對 $a, b \in \{0, 1\}$，四個相位和 $\sum_{m,n} w[a + 2m, b + 2n]$ 必須相等。一種構造方式是對任意 2×2 卷積核 `v`，令 `w = np.kron(v, np.ones((2, 2)))`：這樣每個相位剛好接收 `v` 的每個元素一次，所以所有相位和都等於 `v` 的總和。Resize-convolution 對任何卷積核都有同樣的性質，這就是它平坦的原因。

5. **設計的感受野**。把感受野程式碼中的 U-Net 改成三層而不是四層，以及四層但每層三個卷積。哪一種改動每 GMAC 換到更多的感受野？接著把一般網路中間的 32 層換成 16 層擴張率為 2 的層（在 `analyze` 中加入擴張率欄位）。新的感受野是多少？這種設計有 DLCV 04 中的哪個問題的風險？

提示

卷積核為 k、擴張率為 d 的層，涵蓋 $d(k-1)+1$ 個輸入，所以在往回推的步驟中，把 `k - 1` 換成 `d * (k - 1)`。拿掉一個層級，等於拿掉 U-Net 中最粗、最便宜、感受野最大的部分；每層多加卷積，則會在每個解析度都增加成本。堆疊擴張率相同的層，會讓感受野出現空洞（網格效應，gridding）。

參考文獻

1. K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2016. CVF ·arXiv
2. Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell and S. Xie, "A ConvNet for the 2020s," in *Proc. IEEE/CVF CVPR*, 2022. CVF ·arXiv
3. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations* (*ICLR*), 2021. arXiv
4. J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool and R. Timofte, "SwinIR: Image Restoration Using Swin Transformer," in *Proc. IEEE/CVF International Conference on Computer Vision Workshops* (*ICCVW*), 2021. CVF ·arXiv
5. T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proc. IEEE CVPR*, 2017. CVF ·arXiv
6. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows," in *Proc. IEEE/CVF International Conference on Computer Vision* (*ICCV*), 2021. CVF ·arXiv
7. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy et al., "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015. doi
8. W. Yu, M. Luo, P. Zhou, C. Si, Y. Zhou, X. Wang, J. Feng and S. Yan, "MetaFormer Is Actually What You Need for Vision," in *Proc. IEEE/CVF CVPR*, 2022. CVF ·arXiv
9. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017. arXiv
10. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *Proc. IEEE/CVF CVPR*, 2018. CVF ·arXiv
11. K. He, X. Zhang, S. Ren and J. Sun, "Identity Mappings in Deep Residual Networks," in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, Springer, 2016. doi ·arXiv
12. B. Lim, S. Son, H. Kim, S. Nah and K. M. Lee, "Enhanced Deep Residual Networks for Single Image Super-Resolution," in *Proc. IEEE CVPR Workshops* (*CVPRW*), 2017. CVF ·arXiv
13. H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve and H. Jégou, "Going Deeper With Image Transformers," in *Proc. IEEE/CVF ICCV*, 2021. CVF
14. L. Chen, X. Chu, X. Zhang and J. Sun, NAFNet official implementation, `basicsr/models/archs/NAFNet_arch.py`, GitHub (megvii-research/NAFNet), accessed 2026. code

15. Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong and Y. Fu, “Image Super-Resolution Using Very Deep Residual Channel Attention Networks,” in *Proc. ECCV*, 2018. CVF ·arXiv
16. I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He and P. Dollár, “Designing Network Design Spaces,” in *Proc. IEEE/CVF CVPR*, 2020. CVF ·arXiv
17. J. Hu, L. Shen and G. Sun, “Squeeze-and-Excitation Networks,” in *Proc. IEEE/CVF CVPR*, 2018. CVF ·arXiv
18. S. Woo, J. Park, J.-Y. Lee and I. S. Kweon, “CBAM: Convolutional Block Attention Module,” in *Proc. ECCV*, 2018. CVF ·arXiv
19. L. Chen, X. Chu, X. Zhang and J. Sun, “Simple Baselines for Image Restoration,” in *Proc. ECCV*, LNCS, Springer, 2022. doi ·arXiv
20. M.-H. Guo, T.-X. Xu, J.-J. Liu, Z.-N. Liu, P.-T. Jiang, T.-J. Mu, S.-H. Zhang, R. R. Martin, M.-M. Cheng and S.-M. Hu, “Attention Mechanisms in Computer Vision: A Survey,” *Computational Visual Media*, vol. 8, no. 3, pp. 331–368, 2022. doi ·arXiv
21. S. W. Zamir, A. Arora, S. Khan, M. Hayat, F. S. Khan and M.-H. Yang, “Restormer: Efficient Transformer for High-Resolution Image Restoration,” in *Proc. IEEE/CVF CVPR*, 2022. CVF · arXiv
22. X. Chen, Z. Li, Y. Pu, Y. Liu, J. Zhou, Y. Qiao and C. Dong, “A Comparative Study of Image Restoration Networks for General Backbone Network Design,” in *Proc. ECCV 2024*, LNCS, Springer, 2025. doi ·arXiv
23. S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in *Proc. International Conference on Machine Learning (ICML)*, PMLR 37, 2015. PMLR
24. J. L. Ba, J. R. Kiros and G. E. Hinton, “Layer Normalization,” arXiv:1607.06450, 2016. arXiv
25. D. Ulyanov, A. Vedaldi and V. Lempitsky, “Instance Normalization: The Missing Ingredient for Fast Stylization,” arXiv:1607.08022, 2016. arXiv
26. Y. Wu and K. He, “Group Normalization,” in *Proc. ECCV*, 2018. CVF ·arXiv
27. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” in *Proc. IEEE CVPR*, 2017. CVF
28. X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao and C. C. Loy, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” in *Proc. ECCV Workshops*, 2018. CVF · arXiv
29. K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang, “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising,” *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017. doi ·arXiv
30. Q. Huynh-Thu and M. Ghanbari, “Scope of Validity of PSNR in Image/Video Quality Assessment,” *Electronics Letters*, vol. 44, no. 13, pp. 800–801, 2008. doi
31. A. Abdelhamed, S. Lin and M. S. Brown, “A High-Quality Denoising Dataset for Smartphone Cameras,” in *Proc. IEEE/CVF CVPR*, 2018. CVF
32. S. Nah, T. H. Kim and K. M. Lee, “Deep Multi-Scale Convolutional Neural Network for Dy-

namic Scene Deblurring," in *Proc. IEEE CVPR*, 2017. CVF

33. N. Shazeer, "GLU Variants Improve Transformer," arXiv:2002.05202, 2020. arXiv
34. Y. N. Dauphin, A. Fan, M. Auli and D. Grangier, "Language Modeling with Gated Convolutional Networks," in *Proc. ICML*, PMLR 70, 2017. PMLR
35. D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," arXiv:1606.08415, 2016. arXiv
36. W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert and Z. Wang, "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," in *Proc. IEEE CVPR*, 2016. CVF ·arXiv
37. A. Odena, V. Dumoulin and C. Olah, "Deconvolution and Checkerboard Artifacts," *Distill*, vol. 1, no. 10, 2016. Distill ·doi
38. A. Aitken, C. Ledig, L. Theis, J. Caballero, Z. Wang and W. Shi, "Checkerboard Artifact Free Sub-Pixel Convolution: A Note on Sub-Pixel Convolution, Resize Convolution and Convolution Resize," arXiv:1707.02937, 2017. arXiv
39. O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proc. Medical Image Computing and Computer-Assisted Intervention* (*MICCAI*), LNCS, Springer, pp. 234–241, 2015. doi ·arXiv
40. G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *Proc. IEEE CVPR*, 2017. CVF
41. L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018. doi ·arXiv
42. W. Luo, Y. Li, R. Urtasun and R. Zemel, "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2016. NeurIPS ·arXiv
43. J. Kim, J. K. Lee and K. M. Lee, "Accurate Image Super-Resolution Using Very Deep Convolutional Networks," in *Proc. IEEE CVPR*, 2016. CVF ·arXiv
44. H. Guo, J. Li, T. Dai, Z. Ouyang, X. Ren and S.-T. Xia, "MambaIR: A Simple Baseline for Image Restoration with State-Space Model," in *Proc. ECCV 2024*, LNCS, Springer, 2025. doi ·arXiv
45. ITU-T, "Recommendation T.81: Information technology – Digital compression and coding of continuous-tone still images – Requirements and guidelines" (the JPEG standard), 1992. ITU
46. T. Elsken, J. H. Metzen and F. Hutter, "Neural Architecture Search: A Survey," *Journal of Machine Learning Research*, vol. 20, 2019. JMLR ·arXiv
47. A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455–5516, 2020. doi ·arXiv
48. S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan and M. Shah, "Transformers in Vision: A Survey," *ACM Computing Surveys*, vol. 54, no. 10s, 2022. doi ·arXiv
49. S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon and S. Xie, "ConvNeXt V2: Co-

Designing and Scaling ConvNets With Masked Autoencoders," in *Proc. IEEE/CVF CVPR*, 2023. CVF ·arXiv

50. M. Goldblum, H. Souri, R. Ni, M. Shu, V. Prabhu, G. Somepalli, P. Chattopadhyay, M. Ibrahim et al., "Battle of the Backbones: A Large-Scale Comparison of Pretrained Models across Computer Vision Tasks," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2023. arXiv
51. X. Chen, X. Wang, J. Zhou, Y. Qiao and C. Dong, "Activating More Pixels in Image Super-Resolution Transformer," in *Proc. IEEE/CVF CVPR*, 2023. CVF ·arXiv
52. H. Guo, Y. Guo, Y. Zha, Y. Zhang, W. Li, T. Dai, S.-T. Xia and Y. Li, "MambaIRv2: Attentive State Space Restoration," in *Proc. IEEE/CVF CVPR*, 2025. CVF ·arXiv
53. A. Araujo, W. Norris and J. Sim, "Computing Receptive Fields of Convolutional Neural Networks," *Distill*, vol. 4, no. 11, 2019. Distill ·doi
54. X. Ding, X. Zhang, N. Ma, J. Han, G. Ding and J. Sun, "RepVGG: Making VGG-style ConvNets Great Again," in *Proc. IEEE/CVF CVPR*, pp. 13733–13742, 2021. CVF paper
55. S. Zagoruyko and N. Komodakis, "DiracNets: Training Very Deep Neural Networks Without Skip-Connections," arXiv:1706.00388, 2017. arXiv
56. X. Ding, Y. Guo, G. Ding and J. Han, "ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks," in *Proc. IEEE/CVF ICCV*, 2019. CVF
57. H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid Scene Parsing Network," in *Proc. IEEE CVPR*, 2017. CVF
58. M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proc. IEEE CVPR*, 2016. CVF
59. E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan and Q. V. Le, "AutoAugment: Learning Augmentation Strategies From Data," in *Proc. IEEE/CVF CVPR*, 2019. CVF
60. H. Zhang, M. Cisse, Y. N. Dauphin and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," in *Proc. ICLR*, 2018. arXiv

DLCV 01C・模型壓縮與高效推論

如何在不自欺的前提下讓訓練好的視覺模型更小更快：成本是什麼、該怎麼量測，以及剪枝、量化、知識蒸餾、低秩分解、小型架構的設計與搜尋，並說明為什麼論文裡的加速不等於你裝置上的加速。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/

你將學到

- 區分模型大小、運算量、記憶體流量、延遲與吞吐量。
- 比較非結構化、結構化與硬體支援的稀疏剪枝。
- 選擇量化尺度、校準方式，以及 PTQ 或 QAT。
- 說明蒸餾、低秩分解與高效率架構設計。
- 檢查任務敏感度，並在目標裝置量測匯出模型。

先看全貌

訓練好的網路，對於某個特定裝置上的某個特定工作，通常比需要的更大、更慢。**模型壓縮**（model compression）是一組在訓練之後（或訓練期間）移除這些多餘部分的技術，而**高效推論**（efficient inference）則是讓較小的模型在真實硬體上真的跑得更快的工程。兩者並不相同，這個領域裡大多數的失望都來自把它們混為一談。

> **用最簡單的話說**
>
> 想像你要從大房子搬進小公寓。你可以丟掉從來不用的東西（剪枝）。你可以把衣服折得更緊，讓同樣的衣服占更少空間（量化）。你可以請一位已經住在小公寓的朋友告訴你哪幾樣東西真正重要，然後照著他的選擇做（蒸餾）。或者一開始就買可折疊的家具（設計小型網路）。但搬家公司是照箱子的重量與趟數收費，不是看你清單寫得多整齊。同樣地，電腦在意的是它要搬多少位元組，以及有沒有合適的工具打開這些箱子，而不是你的模型在紙面上看起來有多小。

為什麼重要。大部分部署的視覺模型都在預算之下運作（手機相機、每秒 60 張影格的工廠產線、靠電池的機器人），壓縮決定了模型放不放得進這個預算。壓縮也會改變模型：量化或剪枝後的網路是另一個函數，可能在原本不會出錯的地方出錯。因此整章只有一條規則：**在目標裝置上量測成本，並在部署資料上量測壓縮後的準確率**。

基礎知識在哪裡。本章建立在站內其他地方教過的內容上，並以連結取代重複：計算 FLOPs、記憶體帳、延遲協定、仿射量化公式與匯出執行環境，見 ML 11；帶溫度的 logit 蒸餾損失與經過測試的梯度檢查，見 DLCV 01 補充資料；深度可分離卷積與區塊成本，見 DLCV 02 與 DLCV 01B；偵測器的速度量測，見 DLCV 03A。這裡所有程式碼都只用 NumPy 與 scikit-learn，在 CPU 上幾秒就跑完，並使用 scikit-learn 內建的 digits 資料集或合成資料。**我們程式碼產生的每一個準確率數字都是玩具結果**，來自 8×8 像素的數字辨識任務；它說明的是一個機制，而不是真實視覺模型上效果的大小。

下表是本章的地圖。

技術	縮小什麼	需要什麼	速度可能從哪裡來
非結構化剪枝	非零權重的數量	微調；要稀疏核才跑得快	稀疏核，或受限於記憶體時的壓縮儲存
結構化剪枝	通道、注意力頭、區塊	微調	較小的稠密張量：任何執行環境都適用
量化	每個權重與激活值的位元數	校準資料，或量化感知訓練	整數或低精度核；搬動的位元組更少
蒸餾	架構（較小的學生）	一個教師與一次訓練	學生在設計上就比較小
低秩分解	把一個大層拆成兩個細層	通常需要微調	乘加運算更少，前提是細層能有效率地執行
高效設計與搜尋	架構本身	搜尋或設計預算	依硬體挑選的區塊

「成本」是什麼意思

白話版。壓縮前先確定受限資源：儲存空間、執行記憶體、延遲或吞吐量。參數或運算較少，不保證推論較快。

六個彼此不同的數字

描述執行一個網路要花多少成本的有六個數字：

- **參數量** P：學習到的值的數量。每個值用 b 位元時，權重在磁碟與記憶體中占 $Pb/8$ 位元組。剪枝與量化都會降低這個數字。
- **乘加運算**（multiply–accumulate, MAC）或 **FLOPs**：每張影像的運算量。一次 MAC 是一次乘法加一次加法，通常算作 2 FLOPs；各函式庫的計算方式不一致，所以要寫明你回報的是哪一種。一個從 C_{in} 個通道到 C_{out} 個通道、輸出為 $H' \times W'$ 的 $k \times k$ 卷積，成本是 $k^2 C_{\text{in}} C_{\text{out}} H' W'$ 次 MAC（ML 11）。
- **記憶體流量**（memory traffic）：在主記憶體與運算單元之間搬動的位元組數。權重、輸入激活值與輸出激活值，至少都要被讀取或寫入一次。
- **尖峰記憶體**（peak memory）：一次前向傳遞中，同一時間存在的最大記憶體量（權重加上仍需要的激活值）。對高解析度的影像復原網路而言，它可能由激活值而不是權重主導。
- **延遲**（latency）：在目標裝置上，一個輸入從頭到尾的實際經過時間。相機管線或使用者感受到的就是這個。
- **吞吐量**（throughput）：許多影像一起處理時，每秒處理的影像數。大批次會提高吞吐量，但不會降低單張影像的延遲。

Dehghani 等人指出，這些成本指標可能互相矛盾，只回報其中一個，可能會對哪個模型比較有效率得出錯誤的結論 [1]。壓縮會讓這種不一致更嚴重，因為每種技術攻擊的是不同的數字：剪枝針對參數量與 MAC，量化針對位元組，而兩者都不會自動針對延遲。

運算強度與屋頂線

白話版。層的速度可能受運算或記憶體存取限制；若瓶頸是資料搬移，減少計算的幫助有限。

精確版。 Williams、Waterman 與 Patterson 的**屋頂線模型**（roofline model）[2] 用下式限制一個計算可達到的速度：

$$\text{attainable speed} = \min(\pi,\ \beta \cdot I), \qquad I = \frac{\text{operations}}{\text{bytes moved}},$$

其中 π 是裝置的尖峰運算速率（這裡以每秒 MAC 數計）、β 是它的記憶體頻寬（每秒位元組數），I 是該計算的**運算強度**（arithmetic intensity，每位元組的 MAC 數）。兩個上限交會處的強度 $I^* = \pi/\beta$ 稱為**脊點**（ridge point）：低於它時，該層**受限於記憶體**（memory-bound），高於它時則**受限於運算**（compute-bound）。單一層的簡單時間估計是

$$t \approx \max\left(\frac{\text{MACs}}{\pi},\ \frac{\text{bytes}}{\beta}\right).$$

下面的程式碼把這個式子套用到一個**虛構的**加速器（4 TMAC/s、100 GB/s；真實裝置各不相同）上的幾個層，並假設每個權重、輸入與輸出值恰好各穿過記憶體一次。真實的核會重複利用快取並讓資料傳輸與運算重疊，所以這些數字只是理想化的示意。

```
# A made-up accelerator: 4 TMAC/s peak compute, 100 GB/s memory bandwidth.
PEAK_MACS, BANDWIDTH = 4e12, 100e9

def layer(kind, C=64, H=56, W=56, batch=1):
    """MACs and bytes moved, if weights, inputs and outputs each cross memory once."""
    if kind == "conv 3x3":
        weights, macs = 9 * C * C, 9 * C * C * H * W
    elif kind == "depthwise 3x3":
        weights, macs = 9 * C, 9 * C * H * W
    elif kind == "pointwise 1x1":
        weights, macs = C * C, C * C * H * W
    elif kind == "linear 2048x2048":
        weights, macs, C, H, W = 2048 * 2048, 2048 * 2048, 2048, 1, 1
    acts = 2 * C * H * W                                    # one input and one output tensor
    return batch * macs, weights + batch * acts

for kind, batch, nbytes in [("conv 3x3", 1, 4), ("depthwise 3x3", 1, 4), ("pointwise 1x1", 1,
↪  4),
                            ("depthwise 3x3", 1, 1), ("linear 2048x2048", 1, 4), ("linear
                             ↪  2048x2048", 64, 4)]:
    macs, values = layer(kind, batch=batch)
    traffic = values * nbytes
    t_compute, t_memory = macs / PEAK_MACS, traffic / BANDWIDTH
    bound = "compute" if t_compute > t_memory else "memory"
    print(f"{kind:17s} batch {batch:2d} {8 * nbytes:2d}-bit: {macs / 1e6:7.1f} M MACs, "
          f"{macs / traffic:5.1f} MACs/byte, ~{max(t_compute, t_memory) * 1e6:6.1f} us
           ↪  ({bound}-bound)")
print(f"ridge point: {PEAK_MACS / BANDWIDTH:.0f} MACs/byte")
```

輸出：

```
conv 3x3          batch  1 32-bit:   115.6 M MACs,  65.9 MACs/byte, ~  28.9 us
↪  (compute-bound)
```

```
depthwise 3x3     batch  1 32-bit:     1.8 M MACs,   1.1 MACs/byte, ~  16.1 us (memory-bound)
pointwise 1x1     batch  1 32-bit:    12.8 M MACs,   7.9 MACs/byte, ~  16.2 us (memory-bound)
depthwise 3x3     batch  1  8-bit:     1.8 M MACs,   4.5 MACs/byte, ~   4.0 us (memory-bound)
linear 2048x2048  batch  1 32-bit:     4.2 M MACs,   0.2 MACs/byte, ~ 167.9 us (memory-bound)
linear 2048x2048  batch 64 32-bit:   268.4 M MACs,  15.1 MACs/byte, ~ 178.3 us (memory-bound)
ridge point: 40 MACs/byte
```

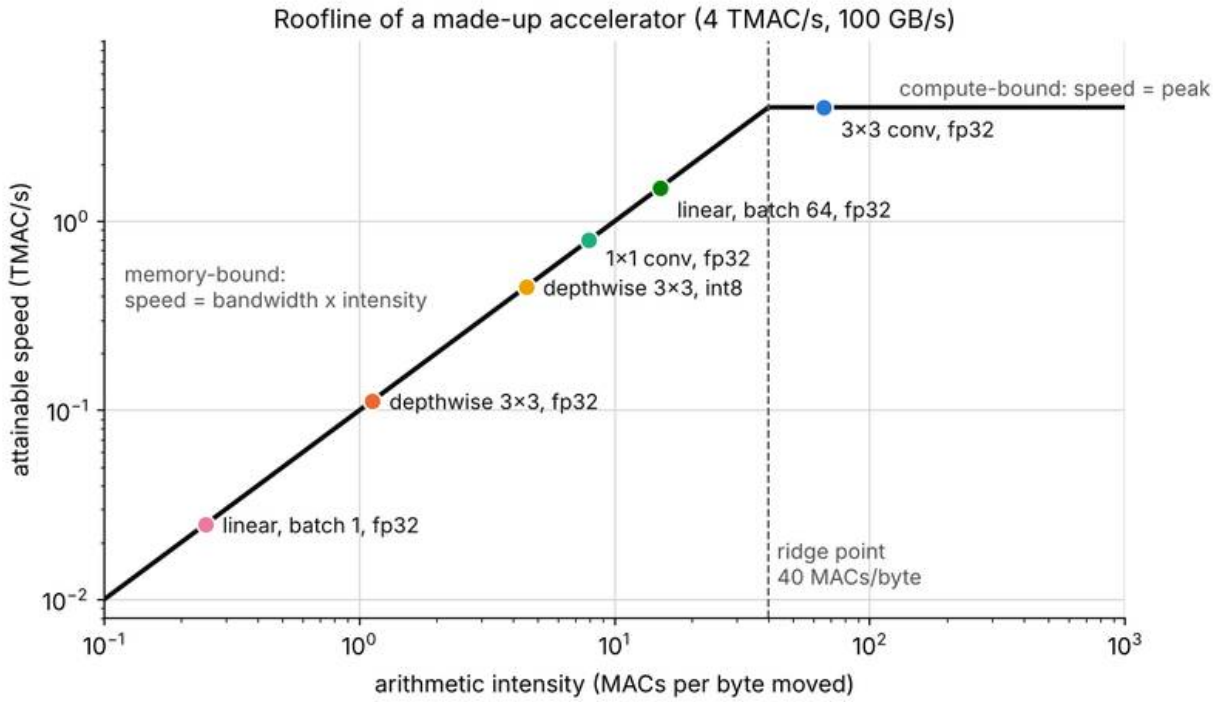


圖 *1* — 一個虛構加速器的屋頂線，以及程式碼中的各層。脊點左邊的所有層都受限於記憶體流量，所以減少它們的運算幾乎沒有幫助；減少它們的位元組（更少的位元、更大的批次）才會讓它沿著斜線往上移。

後面會反覆出現三個教訓。第一，深度卷積層的 MAC 比稠密的 3 × 3 卷積少 64 倍，估計卻只快了約 1.8 倍，因為它受限於記憶體（這就是 DLCV 01B 中「MAC 不等於延遲」這項警告背後的算術）。第二，把那一層改用 8 位元而不是 32 位元儲存，估計時間縮短為 1/4，而一個 MAC 都沒有變：對受限於記憶體的層，量化的好處來自搬動更少的位元組。第三，批次 64 的線性層做的工作是批次 1 的 64 倍，花的時間卻差不多，因為它的權重只讀一次就重複使用，所以大批次的吞吐量幾乎說明不了批次 1 的延遲。

誠實地量測

請遵循 ML 11 的延遲協定（固定並回報裝置、執行環境、精度、批次大小、解析度與執行緒數；暖機；百分位數；在 GPU 上同步），以及 DLCV 03A 的端到端檢查清單。壓縮另外多了三條自己的規則：

1. **與公平的稠密基準比較**：一個大小相同、用相同預算訓練的稠密模型。Liu 等人發現，在他們檢驗的結構化剪枝方法中，從頭訓練剪枝後的架構，表現持平或優於微調剪枝後的權重 [3]；Blalock 等人回顧了 81 篇剪枝論文，發現基準與指標不一致，使大多數比較無從進行 [4]。
2. **在匯出之後、在你實際出貨的執行環境上量測準確率**。匯出、量化、融合後的計算圖，在數值上是和你訓練的不同的程式。
3. **回報取捨曲線，而不是單一個點**。在幾種壓縮程度下，準確率對延遲（與記憶體）的曲線，才能顯示一個方法是移動了整條曲線，還是只是在曲線上挑了另一個點。

剪枝

白話版。剪枝移除選定權重或通道。零散零值需稀疏核心才能加速；移除通道會縮小稠密運算，但延遲仍須實測。

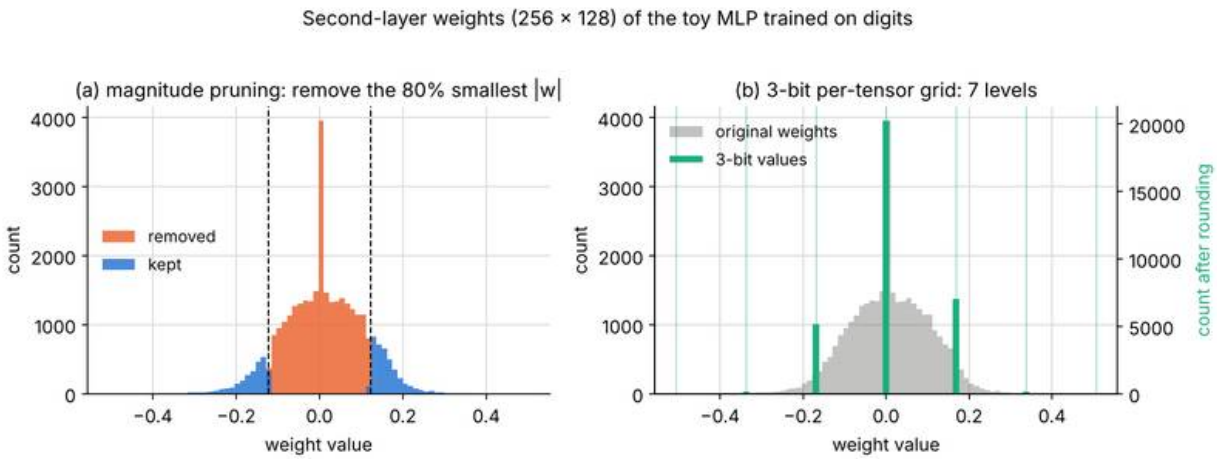


圖 2 — 兩種壓縮對本章全程使用的玩具 *MLP* 的一個權重矩陣做了什麼。(*a*) 量值剪枝移除分布的中間部分。(*b*) 整個張量共用一個尺度的 *3* 位元格點：約 *62%* 的權重捨入成精確的零，*99%* 落在中間三個格點，所以格點大部分都花在少數大權重上。

移除什麼

把網路的參數寫成向量 $\boldsymbol{\theta} \in \mathbb{R}^N$。剪枝選擇一個二元**遮罩**（mask）$\mathbf{m} \in \{0,1\}^N$，並用 $\mathbf{m} \odot \boldsymbol{\theta}$ 取代 $\boldsymbol{\theta}$，其中 $\odot$ 是逐元素相乘。**稀疏度**（sparsity）是零所占的比例，

$$s = 1 - \frac{\|\mathbf{m}\|_0}{N},$$

其中 $\|\mathbf{m}\|_0$ 計算遮罩中非零項的個數。各種方法的差別在於遮罩的**粒度**：

- **非結構化**（細粒度）：任何單一權重都可以被移除。這在選擇移除什麼上有最大的自由度，但結果是一個形狀仍為稠密、卻充滿零星零的張量。Li 等人指出，除非有稀疏卷積函式庫，這種不規則的稀疏並不會大幅降低卷積層的成本 [5]。

- **結構化**（粗粒度）：整個濾波器、通道、注意力頭或區塊被移除。移除一個卷積的輸出通道 j，也會移除下一層的輸入通道 j，所以兩個張量都縮小，而且仍是稠密的。Li 等人正是因為這個原因，把整個濾波器連同與它相連的特徵圖一起剪掉：結果可以用一般的稠密矩陣函式庫執行 [5]。
- **半結構化**：硬體可以利用的固定小樣式。NVIDIA 的稀疏 Tensor Core 支援 **2:4 稀疏**，也就是每連續四個權重中至少有兩個零；該論文報告這種樣式的數學吞吐量是稠密單元的兩倍 [6]。

僅線上版：*N:M* 半結構化稀疏 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/notes/nm-半結構化稀疏/*

該移除哪些權重：量值與其他準則

最簡單、至今仍最常用的準則是**量值**（magnitude）：移除 $|\theta_i|$ 最小的權重。它可以逐層套用（每層移除相同比例）或全域套用（整個網路用同一個閾值；小權重多的層因此損失較多）。對結構化剪枝，則改用一組權重的量值：Li 等人用每個濾波器的絕對權重和（ℓ_1 範數）為濾波器排名 [5]。**Network slimming** 在訓練時對批次正規化層的尺度因子 γ 加上 ℓ_1 懲罰，把不重要通道的尺度推向零，然後移除 $|\gamma|$ 最小的通道 [7]。還有許多其他準則（以梯度為基礎、二階的、以資料為基礎的）；Hoefler 等人對它們做了綜述 [8]。

結構化剪枝有一個非結構化剪枝沒有的陷阱：**耦合的通道**，例如殘差相加兩側的通道，它們必須一起被移除。Fang 等人的 DepGraph 可以從計算圖自動找出這類群組 [9]。

僅線上版：剪枝耦合的通道 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/notes/剪枝耦合的通道/*

剪枝、微調、再重複

移除權重會改變函數，所以剪枝之後要接著訓練。Han 等人描述了標準的三步驟配方：訓練網路以學出哪些連接重要、剪掉不重要的連接，並重新訓練剩下的權重；他們在 ImageNet 上報告，把 AlexNet 的參數從 6,100 萬降到 670 萬、把 VGG-16 的從 1.38 億降到 1,030 萬，而準確率沒有下降 [10]。他們的 Deep Compression 流程又在剪枝之上加入量化與 Huffman 編碼 [11]。

剪枝常被分成幾個較小的步驟，每一步之後都做一些訓練，而不是一次大幅砍掉。Zhu 與 Gupta 的**漸進式**排程，在訓練期間把稀疏度從零升到目標值，一開始升得快，接近尾聲時升得慢 [12]。他們也比較了記憶體占用相同的大而稀疏模型與小而稠密模型，發現在他們測試的各種架構上，大而稀疏的模型一致地更準確 [12]。

僅線上版：漸進式剪枝排程 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/notes/漸進式剪枝排程/*

程式碼：小型 MLP 的非結構化與結構化剪枝

這段程式碼先在 scikit-learn 的 digits 上訓練一個 64–256–128–10 的 MLP，再用兩種方式剪枝：移除每一層最小的權重，以及移除整個隱藏單元（傳入權重的 ℓ_1 範數最小者，兩個隱藏層都做）。

每個剪枝後的模型都微調 10 個 epoch，並在每一步最佳化器更新之後重新套用遮罩。最後，把剪得最多的結構化模型切成一個真正更小的稠密網路。

```
import copy, warnings
import numpy as np
from sklearn.datasets import load_digits
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
warnings.filterwarnings("ignore", category=ConvergenceWarning)

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X / 16.0, y, test_size=0.3, random_state=0, stratify=y)
base = MLPClassifier(hidden_layer_sizes=(256, 128), max_iter=300, random_state=0).fit(Xtr,
 ↪  ytr)
print(f"dense: {sum(w.size for w in base.coefs_)} weights, test accuracy {base.score(Xte,
 ↪  yte):.3f}")

def fine_tune(net, masks, epochs=10):
    order_rng = np.random.default_rng(0)
    for _ in range(epochs):
        order = order_rng.permutation(len(Xtr))
        for i in range(0, len(Xtr), 128):
            net.partial_fit(Xtr[order[i:i + 128]], ytr[order[i:i + 128]])   # one Adam step
            for w, m in zip(net.coefs_, masks):
                w *= m                                                      # keep pruned
                 ↪  weights at zero

def unstructured_masks(net, sparsity):          # smallest |w| in each layer -> 0
    return [np.abs(w) > np.quantile(np.abs(w), sparsity) for w in net.coefs_]

def structured_masks(net, keep):                # drop hidden units with the smallest L1 norm
 ↪  of incoming weights
    masks = [np.ones_like(w) for w in net.coefs_]
    for l in range(len(net.coefs_) - 1):
        norms = np.abs(net.coefs_[l]).sum(axis=0)
        dead = np.argsort(norms)[: int(round((1 - keep) * norms.size))]
        masks[l][:, dead] = 0                   # incoming weights of the unit ...
        masks[l + 1][dead, :] = 0               # ... and its outgoing weights
    return masks

for name, make, levels in [("remove weights", unstructured_masks, [0.8, 0.9, 0.95]),
                           ("keep units", structured_masks, [0.5, 0.25, 0.125])]:
    for level in levels:
        net = copy.deepcopy(base)
        masks = make(net, level)
        for w, m in zip(net.coefs_, masks):
            w *= m
        before = net.score(Xte, yte)
        fine_tune(net, masks)
        kept = int(sum(m.sum() for m in masks))
        print(f"{name} {level:6.1%}: {kept:5d} weights left, "
              f"accuracy {before:.3f} -> {net.score(Xte, yte):.3f} after fine-tuning")

# The last structured model is a genuinely smaller dense network: slice out the live units.
```

```
alive = [masks[l].any(axis=0) for l in range(2)]
small_W = [net.coefs_[0][:, alive[0]], net.coefs_[1][alive[0]][:, alive[1]],
 ↪  net.coefs_[2][alive[1]]]
small_b = [net.intercepts_[0][alive[0]], net.intercepts_[1][alive[1]], net.intercepts_[2]]
h = Xte
for l, (w, b) in enumerate(zip(small_W, small_b)):
    h = h @ w + b if l == 2 else np.maximum(h @ w + b, 0)
print("sliced shapes:", [w.shape for w in small_W],
      "| same predictions:", bool((h.argmax(1) == net.predict(Xte)).all()))
```

輸出（玩具結果）：

```
dense: 50432 weights, test accuracy 0.980
remove weights  80.0%: 10087 weights left, accuracy 0.909 -> 0.974 after fine-tuning
remove weights  90.0%:  5044 weights left, accuracy 0.563 -> 0.954 after fine-tuning
remove weights  95.0%:  2523 weights left, accuracy 0.170 -> 0.885 after fine-tuning
keep units  50.0%: 17024 weights left, accuracy 0.887 -> 0.981 after fine-tuning
keep units  25.0%:  6464 weights left, accuracy 0.550 -> 0.980 after fine-tuning
keep units  12.5%:  2720 weights left, accuracy 0.402 -> 0.957 after fine-tuning
sliced shapes: [(64, 32), (32, 16), (16, 10)] | same predictions: True
```

從中能讀到什麼。**微調做了大部分的工作**：移除 90% 的權重會讓準確率掉到 0.563，十個 epoch 就把它帶回 0.954。**結構化剪枝產生了一個真正的網路**，形狀為 64–32–16–10，任何函式庫都能把它當成一般稠密矩陣來執行。而在剩下約 2,500 個權重時，小的稠密網路（0.957）勝過大的稀疏網路（0.885），這與 Zhu 與 Gupta 在大型模型上的發現相反 [12]：我們的網路對 8 × 8 的數字任務來說參數過多，讓它變窄幾乎沒有代價。不要從這個玩具往任一方向推論。

樂透彩券假說

白話版。如果剪枝後的網路可以這麼小，為什麼不直接訓練小網路？Frankle 與 Carbin 發現，在他們的實驗中，隨機重新初始化的剪枝網路訓練效果較差，但同一個稀疏網路若**重設回它原本的初始權重**，訓練效果就和完整網路一樣好。他們提出，大型網路從一開始就含有這種幸運的稀疏子網路（「中獎彩券」，winning tickets）。

這些彩券是用**反覆量值剪枝**找到的：訓練、剪掉量值最小的權重、把留下的權重重設為初始值，然後重複；在小型 MNIST 與 CIFAR-10 網路上，這個做法一致地找到了只有原本大小 10–20% 或更小的彩券 [13]。在 ImageNet 的規模上，權重必須重設為訓練早期的值，而不是初始化時的值 [14]；Liu 等人則報告，在學習率調得好的情況下，他們的實驗中使用彩券初始化並沒有比隨機初始化更好 [3]。實務上的教訓並不誇張：這個假說談的是哪些稀疏網路是可訓練的，而不是更便宜的壓縮方法（找一張彩券要花好幾次完整訓練）。

僅線上版：*The Lottery Ticket Hypothesis*（*Frankle 與 Carbin，ICLR 2019*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/papers/the-lottery-ticket-hypothesisfrankle-與-carbiniclr-2019/*

量化

白話版。用 32 位元浮點數訓練的網路，並不需要那麼高的精度才能做出好的預測。量化（quantization）把權重（通常還有激活值）存成小整數（8 位元，有時是 4 位元），並搭配少數幾個尺度因子。這能讓記憶體減少 4 到 8 倍，在有整數運算的硬體上還能加速計算。困難在於，一個小格點必須同時涵蓋大量的小數字與少數非常大的數字。

均勻仿射量化

這是最常見的方案，因為它允許有效率的定點運算 [15]，它把實數值 x 映射成整數 q，再映射回來 [16]：

$$q = \mathrm{clip}\Big(\mathrm{round}\Big(\frac{x}{s}\Big) + z,\ q_{\min},\ q_{\max}\Big), \qquad \hat{x} = s\,(q - z),$$

其中 $s > 0$ 是**尺度**（scale，相鄰可表示值之間的間距），z 是整數**零點**（zero point，精確表示實數 0 的那個整數），$[q_{\min}, q_{\max}]$ 是 b 位元格式的整數範圍（例如無號 8 位元是 $[0, 255]$，對稱有號格式是 $[-127, 127]$），round 是四捨五入到最近的整數，clip 是夾進範圍內，$\hat{x}$ 是 x 經過反量化後的近似值。對於包含 0 的實數範圍 $[x_{\min}, x_{\max}]$，常見的選擇是

$$s = \frac{x_{\max} - x_{\min}}{q_{\max} - q_{\min}}, \qquad z = q_{\min} - \mathrm{round}\Big(\frac{x_{\min}}{s}\Big).$$

對稱量化固定 $z = 0$ 並使用有號範圍。精確表示 0 很重要，因為零填補與 ReLU 的輸出會產生許多精確的零 [15]。會出現兩種誤差：範圍內的值有**捨入誤差**，最多 $s/2$；範圍外的值有**截斷誤差**。範圍越大，截斷越少，但 s 變大，捨入誤差也隨之變大 [15]。ML 11 有經過測試的實作與誤差對位元數的曲線。

為什麼整數核可以把尺度提到外面

白話版。只有在一層最昂貴的部分，也就是那一長串乘積之和，能完全用整數執行時，整數才有用。如果每個權重列有一個尺度、每個激活張量有一個尺度，這些尺度就可以從和裡面提出來，只在最後套用一次。

對線性層的一個輸出 $y_i = \sum_j W_{ij} x_j$（卷積是對一個區塊做同樣的和），把權重以每個輸出通道一個尺度 $s_{w,i}$ 做對稱量化，$W_{ij} \approx s_{w,i}\, q^{w}_{ij}$，並把輸入以尺度 s_x 與零點 z_x 做仿射量化，$x_j \approx s_x(q^{x}_{j} - z_x)$。於是

$$y_i \approx s_{w,i}\, s_x \sum_j q^{w}_{ij}\,(q^{x}_{j} - z_x),$$

其中的和只涉及整數，並在 32 位元整數暫存器中累加；單一個乘積 $s_{w,i} s_x$ 對每個輸出只套用一次。這就是為什麼權重的逐通道尺度很便宜（每個輸出通道一個因子），而**激活值**的逐通道尺度卻不便宜：它們會落在和的裡面，每個輸入通道都需要重新縮放累加器 [15]。這也是為什麼常見的選擇是對稱的權重加上非對稱的激活值：非對稱的權重零點會為每個輸出加上一個與資料相關的項 [15]。偏置通常保留為 32 位元整數 [16]。

```
import numpy as np

rng = np.random.default_rng(0)
W = rng.normal(0, 0.2, size=(16, 256))              # a layer: 16 outputs, 256 inputs
x = rng.normal(0.3, 1.0, size=256)                  # input activations (not centred at 0)

# Weights: symmetric int8, one scale per output channel (zero point 0).
s_w = np.abs(W).max(axis=1) / 127
W_q = np.round(W / s_w[:, None]).astype(np.int8)

# Activations: affine uint8 with a scale and an integer zero point.
lo, hi = min(x.min(), 0.0), x.max()
s_x = (hi - lo) / 255
z_x = int(np.round(-lo / s_x))
x_q = np.clip(np.round(x / s_x) + z_x, 0, 255).astype(np.uint8)
print("activation scale:", round(s_x, 5), "| zero point:", z_x)

# Integer-only inner products, accumulated in int32, then one rescale per output.
acc = W_q.astype(np.int32) @ (x_q.astype(np.int32) - z_x)
y_int = s_w * s_x * acc
y_ref = W @ x
print("max |integer result - float result|:", np.abs(y_int - y_ref).max().round(4))
print("relative error:", (np.linalg.norm(y_int - y_ref) / np.linalg.norm(y_ref)).round(4))
print("largest |accumulator|:", np.abs(acc).max(), "| int16 limit:", 2**15 - 1)
```

輸出：

```
activation scale: 0.01967 | zero point: 117
max |integer result - float result|: 0.0495
relative error: 0.0083
largest |accumulator|: 83167 | int16 limit: 32767
```

整數路徑重現浮點結果，誤差在約 1% 以內。累加器達到 83,167，遠超過 16 位元能容納的範圍，這就是為什麼即使輸入是 8 位元，加速器仍使用 32 位元累加器 [15]。在進入下一層之前，32 位元的結果會用下一層的尺度轉回 8 位元（「重新量化」，requantization）。

僅線上版：用定點乘數重新量化 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/notes/用定點乘數重新量化/*

粒度與範圍設定

粒度（granularity）是有多少個值共用一個尺度：每個張量一個、每個輸出通道一個，或每一小組連續權重一個。粒度越細，格點越能貼合每一組自己的範圍。**範圍設定**（range setting，也稱校準）是 $[x_{\min}, x_{\max}]$ 的選擇方式 [15]：

- **Min-max**：觀察到的最小值與最大值。沒有截斷，但一個離群值就會把所有人的格點拉寬。
- **百分位數**：忽略一小部分最極端的值。

- **MSE**：在具代表性的資料上，搜尋使 $\|x - \hat{x}\|^2$ 最小的範圍，在截斷與捨入之間取捨。

權重事先已知，不需要資料。激活值取決於輸入，所以它們的範圍是靠讓一小份未標註的**校準集**（calibration set）影像跑過網路來估計；Nagel 等人指出，500 到 1,000 張未標註影像通常就夠了 [15]。

僅線上版：選擇截斷範圍：*min-max*、百分位數與 *MSE* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/notes/選擇截斷範圍 min-max 百分位數與-mse/*

什麼會壞掉：離群值、折疊後的批次正規化與激活值

白話版。當一個共用的尺度必須涵蓋大小差異極大的值時，量化就會失敗。大的值迫使格點變粗，而大量的小值全都被捨入到少數幾個整數。

典型的視覺例子是 MobileNetV2。Nagel 等人指出，它的一個深度可分離層中，不同輸出通道的權重範圍差異極大，以致一組量化參數無法服務整個張量；在折疊批次正規化並使用 8 位元逐張量量化時，他們的模型在 ImageNet 上的 top-1 準確率從 71.72% 掉到 0.12% [17]。他們的解法不需要資料或訓練：因為 $c > 0$ 時 $\mathrm{ReLU}(c\,t) = c\,\mathrm{ReLU}(t)$，所以可以把一層的某個輸出通道縮小，同時把下一層對應的輸入通道放大，函數不會改變（**跨層均衡化**，cross-layer equalization）。在把 ReLU6 換成具有這個性質的 ReLU 之後，均衡化加上偏置吸收，讓模型在 8 位元下達到 70.92% [17]。

我們的玩具重現了這個機制。我們取訓練好的 MLP，把一個隱藏單元放大 100 倍（並把下一層的權重縮小為 1/100）。浮點網路計算的仍是完全相同的函數，但它的激活值與權重範圍現在含有一個巨大的通道。

```
import warnings
import numpy as np
from sklearn.datasets import load_digits
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
warnings.filterwarnings("ignore", category=ConvergenceWarning)

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X / 16.0, y, test_size=0.3, random_state=0, stratify=y)
net = MLPClassifier(hidden_layer_sizes=(256, 128), max_iter=300, random_state=0).fit(Xtr,
 ↪  ytr)

def q_weight(w, bits, per_channel):     # symmetric: one scale per output unit, or one per
 ↪  tensor
    qmax = 2 ** (bits - 1) - 1
    m = np.abs(w).max(axis=0, keepdims=True) if per_channel else np.abs(w).max()
    return np.clip(np.round(w * qmax / m), -qmax, qmax) * m / qmax

def q_act(a, hi, bits):                 # post-ReLU activations: grid on [0, hi], zero point
 ↪  0
    s = hi / (2 ** bits - 1)
    return np.clip(np.round(a / s), 0, 2 ** bits - 1) * s

def run(Ws, bs, X, wbits=None, abits=None, his=None, per_channel=True, record=None):
```

```
    h = X
    for l, (w, b) in enumerate(zip(Ws, bs)):
        h = h @ (q_weight(w, wbits, per_channel) if wbits else w) + b
        if l < len(Ws) - 1:
            h = np.maximum(h, 0)
            if record is not None:
                record.append(h)
            if abits:
                h = q_act(h, his[l], abits)
    return h

acc = lambda logits: (logits.argmax(1) == yte).mean()
for c in [1, 100]:
    Ws = [w.copy() for w in net.coefs_]
    bs = [b.copy() for b in net.intercepts_]
    Ws[0][:, 0] *= c; bs[0][0] *= c; Ws[1][0, :] /= c   # same function: ReLU(c*t) =
     ↪  c*ReLU(t)
    calib = []
    run(Ws, bs, Xtr[:100], record=calib)                 # 100 training images calibrate the
     ↪  ranges
    minmax = [a.max() for a in calib]
    pct99 = [np.percentile(a, 99) for a in calib]
    print(f"unit 0 scaled by {c}: float {acc(run(Ws, bs, Xte)):.3f} | "
          f"W8A8 {acc(run(Ws, bs, Xte, 8, 8, minmax)):.3f} | "
          f"W8A4 {acc(run(Ws, bs, Xte, 8, 4, minmax)):.3f} | "
          f"W8A4 clipped {acc(run(Ws, bs, Xte, 8, 4, pct99)):.3f} | "
          f"W4 per-tensor {acc(run(Ws, bs, Xte, 4, per_channel=False)):.3f} | "
          f"W4 per-channel {acc(run(Ws, bs, Xte, 4)):.3f}")
```

輸出（玩具結果；「W8A4」表示 8 位元權重與 4 位元激活值）：

```
unit 0 scaled by 1: float 0.980 | W8A8 0.980 | W8A4 0.978 | W8A4 clipped 0.978 | W4
 ↪  per-tensor 0.983 | W4 per-channel 0.980
unit 0 scaled by 100: float 0.980 | W8A8 0.980 | W8A4 0.163 | W8A4 clipped 0.980 | W4
 ↪  per-tensor 0.102 | W4 per-channel 0.978
```

兩個網路是同一個函數，但有了一個離群通道，4 位元 min-max 的激活值就崩潰到 0.163，4 位元逐張量的權重崩潰到 0.102。把激活值範圍在第 99 百分位數截斷（0.980），以及使用逐單元的權重尺度（0.978），都能解決。這裡截斷剛好不花任何代價；當離群通道攜帶真正的資訊時就會有代價，這就是為什麼會有移動或隔離離群值的方法。

同樣的模式在更大的規模上，推動了 Transformer 的量化研究：Dettmers 等人在大型語言模型中發現系統性的離群特徵維度，並把它們保留在 16 位元 [18]，而 SmoothQuant 用等價的逐通道重新縮放，把難度從激活值轉移到權重，與跨層均衡化屬於同一類想法 [19]。

訓練後量化與量化感知訓練

白話版。你可以只用一點校準資料就量化一個已完成的模型（快、不需要標籤），或者在訓練模型的同時讓它「感受」到捨入，讓它學到能經得起捨入的權重（慢一些，但在低位元寬度下更準確）。

訓練後量化（post-training quantization, PTQ）取一個訓練好的浮點模型，只用一小份校準集來轉換。Nagel 等人建議的流程是：跨層均衡化；對稱的權重量化器與非對稱的激活量化器（若硬體支援，權重逐通道）；以 MSE 為基礎的權重範圍；然後在有校準資料時使用 **AdaRound**（學習每個權重該向上還是向下捨入 [20]），否則使用解析式的偏置校正；最後設定激活值範圍 [15]。

量化感知訓練（quantization-aware training, QAT）在前向傳遞中插入「偽量化」（先量化、再立刻反量化），並照常訓練。困難在於 round 幾乎處處梯度為零。**直通估計器**（straight-through estimator, STE）[21] 在反向傳遞中把捨入視為恆等映射。對於對稱量化器 $\hat{x} = s\,\mathrm{clip}(\mathrm{round}(x/s), q_{\min}, q_{\max})$，這給出

$$\frac{\partial \hat{x}}{\partial x} \approx \begin{cases} 1, & q_{\min} \le x/s \le q_{\max}, \\ 0, & \text{otherwise}, \end{cases}$$

所以梯度會原封不動地通過範圍內的值，並在被截斷的值處停止 [15]。被更新的是浮點的「影子」權重，而前向傳遞永遠使用它們量化後的版本。

何時用哪一種：在白皮書的實驗中，PTQ 足以處理 8 位元的權重與激活值（所有測試的模型都在浮點結果的 0.7% 以內），而更低的位元寬度，特別是 4 位元激活值，就需要 QAT；使用深度可分離層的網路，即使做了 QAT 仍然是最難的 [15]。

僅線上版：超越四捨五入：*AdaRound*、*BRECQ* 與 *LSQ* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/notes/超越四捨五入 adaroundbrecq-與-lsq/*

混合精度與其他數字格式

並非所有層都同樣敏感，所以**混合精度量化**（mixed-precision quantization）讓不同的層使用不同的位元寬度：HAWQ 根據二階（Hessian）資訊選擇 [22]，HAQ 則根據硬體模擬器回饋的延遲與能耗，並發現邊緣與雲端加速器的最佳策略並不相同 [23]。整數不是唯一的選擇：8 位元浮點編碼（E4M3 與 E5M2）[24] 把位元花在指數上，能容忍更寬的數量級範圍，代價是步階更少。

不要把這和**混合精度訓練**（DLCV 01A）混淆，後者保留一份 32 位元的權重主副本，並用 16 位元計算以加快訓練。混合精度量化談的是部署後的模型。

知識蒸餾

白話版。訓練一個小的**學生**（student）去模仿大的**教師**（teacher），而不只是模仿正確的標籤。教師的完整輸出（它認為哪些錯誤類別也有可能）以及它的中間特徵，每個範例帶有的資訊比一個標籤更多，所以學生從同樣的影像可以學到更多。

Hinton 等人帶溫度 T 與 T^2 因子的回應式損失，已在 DLCV 01 補充資料中推導並測試，並在 ML 11 中摘要 [25]；這裡不重複。用一行來說：學生最小化 $(1-\alpha)\,\mathrm{CE}(y, \sigma(\mathbf{z})) + \alpha\, T^2\, \mathrm{CE}(\sigma(\mathbf{v}/T), \sigma(\mathbf{z}/T))$，其中 $\mathbf{z}$ 與 $\mathbf{v}$ 分別是學生與教師的 logits，y 是標籤，σ 是 softmax，CE 是交叉熵，$\alpha \in [0,1]$ 是軟項的權重。Hinton 等人指出，像 Ba 與 Caruana 那樣用 L2 損失迴歸 logits 來匹配它們 [26]，是這個損失在高溫下的極限 [25]。

程式碼：把玩具 MLP 蒸餾成 8 單元的學生

教師是剪枝程式碼中那個有 50,432 個權重的 MLP。學生只有一個含 8 個單元的隱藏層（592 個權重），用全批次梯度下降訓練：一次只用標籤，兩次使用蒸餾損失。

```
import warnings
import numpy as np
from sklearn.datasets import load_digits
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
warnings.filterwarnings("ignore", category=ConvergenceWarning)

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X / 16.0, y, test_size=0.3, random_state=0, stratify=y)
teacher = MLPClassifier(hidden_layer_sizes=(256, 128), max_iter=300, random_state=0).fit(Xtr,
 ↪  ytr)
h = Xtr
for l, (w, b) in enumerate(zip(teacher.coefs_, teacher.intercepts_)):   # teacher logits
    h = h @ w + b if l == 2 else np.maximum(h @ w + b, 0)
z_teacher = h

def softmax(z, T=1.0):
    e = np.exp((z - z.max(axis=1, keepdims=True)) / T)
    return e / e.sum(axis=1, keepdims=True)

def train_student(T, alpha, hidden=8, steps=500, lr=0.2, seed=0):
    """64 -> 8 -> 10 ReLU network, full-batch gradient descent on the distillation loss."""
    rng = np.random.default_rng(seed)
    W1, b1 = rng.normal(0, np.sqrt(2 / 64), (64, hidden)), np.zeros(hidden)
    W2, b2 = rng.normal(0, np.sqrt(1 / hidden), (hidden, 10)), np.zeros(10)
    Y, p_T = np.eye(10)[ytr], softmax(z_teacher, T)
    for _ in range(steps):
        a = np.maximum(Xtr @ W1 + b1, 0)
        z = a @ W2 + b2
        # gradient of (1-alpha)*CE(y, softmax(z)) + alpha*T^2*CE(p_T, softmax(z/T)) w.r.t. z
        g = ((1 - alpha) * (softmax(z) - Y) + alpha * T * (softmax(z, T) - p_T)) / len(Xtr)
        ga = (g @ W2.T) * (a > 0)
        W2 -= lr * a.T @ g;  b2 -= lr * g.sum(0)
        W1 -= lr * Xtr.T @ ga; b1 -= lr * ga.sum(0)
    return (np.maximum(Xte @ W1 + b1, 0) @ W2 + b2).argmax(1)

print(f"teacher: {sum(w.size for w in teacher.coefs_)} weights, accuracy {teacher.score(Xte,
 ↪  yte):.3f}")
for T in (1, 4):
    print(f"mean top probability of the teacher on training images at T = {T}: "
          f"{softmax(z_teacher, T).max(axis=1).mean():.3f}")
for name, T, alpha in [("labels only", 1, 0.0), ("distill, T = 1", 1, 0.5), ("distill, T =
 ↪  4", 4, 0.5)]:
    accs = [(train_student(T, alpha, seed=s) == yte).mean() for s in range(5)]
    print(f"student (592 weights), {name:14s}: accuracy {np.mean(accs):.3f} +/-
     ↪  {np.std(accs):.3f} over 5 seeds")
```

輸出（玩具結果）：

```
teacher: 50432 weights, accuracy 0.980
mean top probability of the teacher on training images at T = 1: 0.998
mean top probability of the teacher on training images at T = 4: 0.787
student (592 weights), labels only   : accuracy 0.936 +/- 0.025 over 5 seeds
student (592 weights), distill, T = 1: accuracy 0.936 +/- 0.025 over 5 seeds
student (592 weights), distill, T = 4: accuracy 0.954 +/- 0.006 over 5 seeds
```

$T = 4$ 的蒸餾把學生的平均準確率從 0.936 提高到 0.954，並讓它對隨機種子的依賴小得多。$T = 1$ 時則毫無改變，第二行說明了原因：在自己的訓練影像上，教師幾乎是確定的（平均最高機率 0.998），所以它的軟目標實際上就是標籤。溫度才是讓教師對錯誤類別的看法顯現出來的東西。這個效果很小，而且只適用於這個玩具；能遷移的是機制，不是數字。

特徵與注意力蒸餾

教師的中間表示也可以被模仿。**FitNets** 訓練一個細而深的學生，讓它的某個中間層（「被引導」層）經過一個小的學習式**迴歸器**（regressor）後，去預測教師的某個中間層（「提示」層）；需要迴歸器，是因為兩個層的寬度不同，作者使用卷積迴歸器來讓它的大小保持在可控範圍 [27]。**注意力轉移**（attention transfer）模仿的是教師的特徵在哪裡活躍，而不是特徵本身：對特徵圖 $A \in \mathbb{R}^{C\times H\times W}$，它形成一張空間圖

$$F(A) = \sum_{c=1}^{C} |A_c|^p,$$

其中 A_c 是通道 c，$p \geq 1$ 是指數（該論文也研究了 $p = 1$ 以及對通道取最大值），並訓練學生在多個深度上匹配教師經 ℓ_2 正規化的圖 [28]。因為 $F(A)$ 沒有通道維度，教師與學生的寬度不需要一致。Gou 等人依傳遞的內容（回應、特徵、範例之間或層之間的關係）以及傳遞的方式（從固定教師離線、線上，或自蒸餾）來整理這個領域 [29]。

蒸餾有沒有用，取決於什麼

- **教師與學生必須看到相同的輸入。** Beyer 等人發現，當教師與學生拿到每張影像完全相同的擴增視角（「一致」的教學）、擴增很強，並且訓練時間很長（「有耐心」）時，蒸餾效果最好；不一致的設定在他們的實驗中停在較低的水準 [30]。
- **更好的教師不一定是更好的老師。** Cho 與 Hariharan 發現，更大、更準確的教師往往沒有產生更好的學生，他們把原因歸於容量不匹配，發現經由中等大小模型的多步蒸餾無效，並報告提早停止教師的訓練可以緩解這個問題 [31]。
- **蒸餾是一種訓練方法，不是對既有模型的壓縮**：它需要一次訓練、資料，以及事先選定的學生架構。例如 DeiT 訓練視覺 Transformer 時，加了一個向卷積教師學習的蒸餾 token（DLCV 02）[32]。

低秩分解

白話版。一個大的權重矩陣所含的資訊，往往比它的大小所暗示的少得多。把它寫成兩個細矩陣的乘積，就能把一個大層換成兩個小層，權重與運算都少很多。

任何 $m \times n$ 矩陣 W 都有奇異值分解 $W = U\Sigma V^\top$，其中 U（$m \times m$）與 V（$n \times n$）的各行互相正交且長度為 1，Σ 是對角矩陣，對角線上是奇異值 $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$。保留最大的 r 個，就得到 Frobenius 範數下最佳的秩-r 近似（ML 01A）：

$$W \approx AB, \qquad A = U_{:,1:r}\,\Sigma_{1:r,1:r} \in \mathbb{R}^{m \times r}, \qquad B = V^\top_{:,1:r} \in \mathbb{R}^{r \times n},$$

其中 $U_{:,1:r}$ 是 U 的前 r 行，r 是選定的**秩**（rank）。層 $x \mapsto Wx$ 變成兩層 $x \mapsto A(Bx)$，權重數為 $r(m+n)$ 而不是 mn，只有當 $r < mn/(m+n)$ 時才會節省權重與 MAC。對於核為 $W \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k \times k}$ 的卷積，把它重塑成 $C_{\text{out}} \times (C_{\text{in}} k^2)$ 的矩陣再分解：B 變成有 r 個輸出通道的 $k \times k$ 卷積，A 是回到 C_{out} 個通道的 1×1 卷積。Denton 等人利用卷積濾波器中這類線性結構，在 CPU 與 GPU 上把卷積層加速約兩倍，準確率差距在 1% 以內 [33]；Jaderberg 等人建立了在空間域中秩為 1 的濾波器基底（一個 $k \times 1$ 後接一個 $1 \times k$ 的濾波器）[34]；Zhang 等人把低秩加速擴展到用於分類與偵測的非常深的網路，其最小化的是每層非線性響應的重建誤差，而不是權重的重建誤差 [35]。

```
import warnings
import numpy as np
from sklearn.datasets import load_digits
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
warnings.filterwarnings("ignore", category=ConvergenceWarning)

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(X / 16.0, y, test_size=0.3, random_state=0, stratify=y)
net = MLPClassifier(hidden_layer_sizes=(256, 128), max_iter=300, random_state=0).fit(Xtr,
 ↪  ytr)
(W1, W2, W3), (b1, b2, b3) = net.coefs_, net.intercepts_

U, S, Vt = np.linalg.svd(W2, full_matrices=False)       # W2 is 256 x 128
energy = np.cumsum(S**2) / np.sum(S**2)
for r in [4, 8, 16, 32, 128]:
    A, B = U[:, :r] * S[:r], Vt[:r]                     # W2 ~ A @ B: a 256->r and an r->128
     ↪  layer
    h = np.maximum(np.maximum(Xte @ W1 + b1, 0) @ A @ B + b2, 0)
    acc = ((h @ W3 + b3).argmax(1) == yte).mean()
    print(f"rank {r:3d}: {A.size + B.size:5d} weights instead of {W2.size} "
          f"({energy[r - 1]:.0%} of the squared singular values), accuracy {acc:.3f}")
```

輸出（玩具結果）：

```
rank   4:  1536 weights instead of 32768 (29% of the squared singular values), accuracy 0.620
rank   8:  3072 weights instead of 32768 (47% of the squared singular values), accuracy 0.978
rank  16:  6144 weights instead of 32768 (58% of the squared singular values), accuracy 0.981
```

```
rank  32: 12288 weights instead of 32768 (72% of the squared singular values), accuracy 0.981
rank 128: 49152 weights instead of 32768 (100% of the squared singular values), accuracy
 ↪  0.980
```

秩 8 保留了準確率，所用的權重不到該層的十分之一，儘管它只保留了 47% 的奇異值平方。權重的 Frobenius 誤差並不是任務誤差的好指標：重要的是這一層對它實際看到的輸入做了什麼，這就是 Zhang 等人改為最小化響應誤差的原因 [35]。最後一行顯示了損益兩平規則：滿秩時分解後的形式權重比原本更多，因為 $128 > 256 \cdot 128/384 \approx 85$。把一層拆成兩層，還會為中間結果多出一次記憶體往返，所以在受限於記憶體的裝置上，延遲的節省會比 MAC 的節省小。

三個玩具並排看

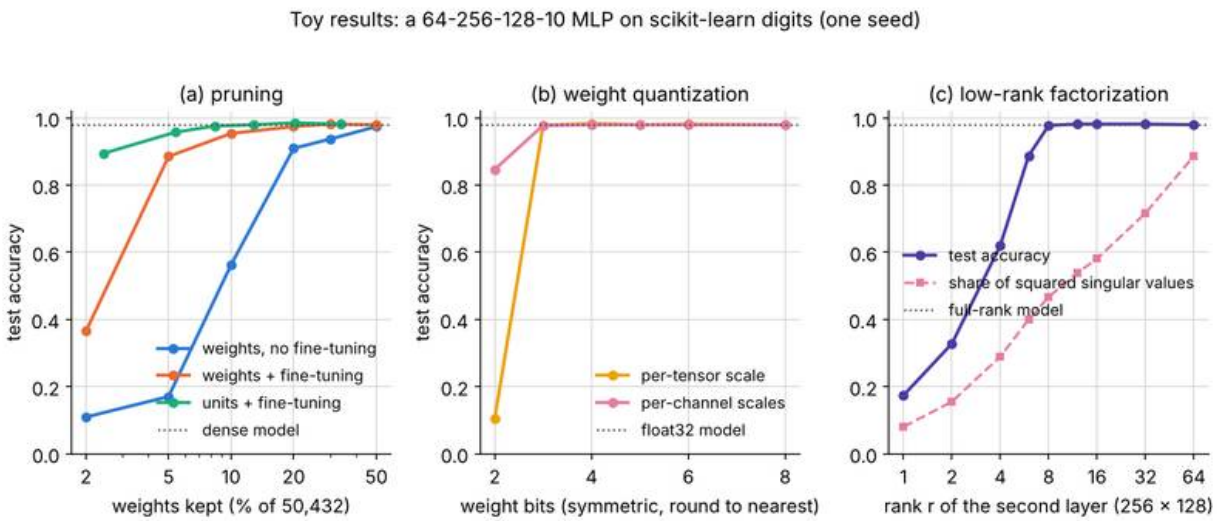


圖 *3* — 同一個網路上的三種玩具壓縮（單一種子；玩具結果）。(*a*) 微調救回了剪枝，而且在這個參數過多的玩具上，移除整個單元撐得最好。(*b*) 權重可以撐到 *3* 位元；在 *2* 位元時，只有逐通道尺度能讓模型保持可用。(*c*) 準確率在秩 *8* 就飽和了，遠早於權重近似變好的時候。

一開始就設計得小

白話版。與其在訓練後縮小一個大網路，你可以從一開始就選擇小而且對硬體友善的架構，不論是手工設計（高效率區塊）還是靠搜尋。實務上兩種做法會結合：設計良好的小網路再做量化，有時也做剪枝或蒸餾。

主要的手工設計家族在別處已經涵蓋，這裡只列出：**深度可分離卷積**與反向殘差（MobileNet [36]、MobileNetV2 [37]；推導與成本見 DLCV 02 與 ML 03）、深度／寬度／解析度的**複合縮放**（EfficientNet [38]；DLCV 02），以及逐區塊的成本算術（DLCV 01B）。本章開頭的屋頂線觀點，是看待這所有家族的正確鏡頭：一個區塊要在某個裝置上有效率，就要對那個裝置而言每位元組做足夠的運算，並且在那裡有快速的核。

重參數化：訓練時豐富，部署時簡單

結構重參數化（structural re-parameterization）訓練一個有數條平行線性分支的區塊，並在部署時把它們合併成一個卷積。RepVGG 訓練的區塊含有一個 3×3 卷積、一個 1×1 卷積與一條恆等分支，每個後面都接批次正規化，部署時則是由 3×3 卷積與 ReLU 組成的簡單堆疊 [39]。折疊的推導與數值檢查，見 DLCV 03A 補充資料；針對增強網路的版本，見 DLCV 09C 補充資料。

重參數化會與量化互相影響。YOLOv6 報告指出，由於大量使用重參數化區塊，先前的 PTQ 技術無法產生好的結果；在它的 1.0 版中，YOLOv6-S 在 8 位元 PTQ 下的 COCO val AP 從 42.4 掉到 35.0，使用 RepOptimizer 訓練後則回到 40.9，因為它縮窄了激活值的分布 [40]。合併後的核加進了範圍差異很大的分支，這正是逐張量量化處理得最差的情況。

神經架構搜尋

白話版。神經架構搜尋（neural architecture search, NAS）讓演算法選擇層的類型、寬度與核大小，並以準確率來評分候選者，在考慮硬體的形式中，還以目標裝置上量測到的延遲來評分。

Elsken 等人用三項選擇描述每一種 NAS 方法：**搜尋空間**（允許哪些架構）、**搜尋策略**（如何提出候選者）與**效能估計策略**（如何在不完整訓練的情況下為候選者評分）[41]。三個想法對壓縮很重要：

- **權重共享讓搜尋負擔得起**。最早的強化學習方法是把每個被提出的架構分別訓練 [42]；ENAS 讓所有候選者共享同一張大圖的權重 [43]，DARTS 把離散的選擇鬆弛成連續的選擇並用梯度下降最佳化 [44]，而 Once-for-All 訓練一個網路，從中取出不同深度、寬度、核大小與解析度的子網路給不同的裝置使用，不需要重新訓練 [45]。
- **針對量測到的延遲搜尋，而不是 FLOPs**。 MnasNet 把在真實手機上量測到的延遲放進目標函數 [46]；ProxylessNAS 直接在目標任務與硬體上搜尋 [47]；FBNet 的可微分搜尋出於同樣的觀察：FLOPs 不一定反映延遲 [48]。
- **保留一個隨機基準**。 Li 與 Talwalkar 發現，在他們使用的兩個基準上，帶提早停止的隨機搜尋至少和 ENAS 一樣好 [49]。

部署：從小模型到快模型

白話版。壓縮後的模型，要等到編譯器或執行環境把它變成少數幾個大的融合運算，並有支援它的數字格式與稀疏樣式的核，才會變快。論文裡的速度宣稱，常假設了你的裝置沒有的支援。

運算子融合與批次正規化折疊

每個獨立的運算都會從記憶體讀入它的輸入，再把輸出寫回去。**運算子融合**（operator fusion）把連續的運算（卷積、偏置、批次正規化、激活函數）合併成一個核，讓中間結果留在快速的晶片內記憶體；從屋頂線的觀點來看，它藉由去掉記憶體往返來提高運算強度。最常見的融合是**批次正規化折疊**（batch-norm folding）：推論時，帶有學習到的尺度 γ_c、平移 β_c、累積平均 μ_c 與變異數 σ_c^2 的批次正規化 [50]，對每個輸出通道 c 是一個仿射映射，所以可以被吸收進前一個卷積，

$$W'_c = \frac{\gamma_c}{\sqrt{\sigma_c^2 + \epsilon}} W_c, \qquad b'_c = \frac{\gamma_c}{\sqrt{\sigma_c^2 + \epsilon}} (b_c - \mu_c) + \beta_c,$$

其中 W_c 與 b_c 是輸出通道 c 的核與偏置，ϵ 是一個很小的常數。經過測試的 NumPy 版本在 DLCV 03A 補充資料。對壓縮而言，請注意因子 $\gamma_c / \sqrt{\sigma_c^2 + \epsilon}$：它在不同通道之間不一樣，所以折疊可能會把各通道的權重範圍拉開，這是 MobileNetV2 難以逐張量量化的原因之一 [17]。在裝置上，批次正規化是在量化之前折疊的，所以整數核看到的是折疊後的權重，而 QAT 必須正確地模擬折疊 [15]。

整數核與執行環境支援

只有當執行環境對計算圖中的每個運算都有整數（或低精度）核時，量化才會加速推論。ML 11 展示了一個純權重 int8 的模型，它小了四倍卻更慢，因為每次呼叫都要把權重轉回浮點。在真實部署中，要檢查：目標是否支援逐通道的權重尺度（不是所有硬體都支援 [15]）；哪些運算子退回浮點（每次退回都會多一次反量化與量化）；以及執行環境是否讓計算圖在各層之間維持整數。匯出與執行環境工具本身（ONNX、ONNX Runtime、TensorRT、`torch.export`）在 ML 11 有說明。

為什麼論文的加速在裝置上會縮水

- **FLOPs 不是瓶頸**。一個移除了受限於記憶體的層 90% MAC 的方法，幾乎不會改變它們的時間（圖 1）。
- **稀疏需要核**。非結構化稀疏，只有在那個裝置上有針對該樣式的稀疏核時才會加速 [5]、[8]；2:4 稀疏在部分 GPU 上有專用硬體 [6]；結構化剪枝不需要任何特別的東西。
- **批次大小**。大批次的吞吐量不是批次 1 的延遲（上面的線性層例子）。
- **管線的其餘部分**。前處理、NMS 這類後處理（DLCV 03A）、縮放與資料傳輸，不會因為網路變小而變小。
- **不支援的運算子**。單一個沒有低精度核的運算子，就可能迫使加入的轉換比省下的還貴。

壓縮視覺模型：哪些地方敏感

白話版。視覺模型的不同部分，對壓縮的反應差異很大。每個通道權重很少的層、必須在數值上精確的輸出（框座標、像素值），以及範圍隨影像而變的激活值，是常見的弱點。

分類骨幹

由深度可分離層建成的網路，是最難量化的常見骨幹。在 Nagel 等人的實驗中，使用 4 位元權重與 8 位元激活值時，逐張量 PTQ 讓 MobileNetV2 與 EfficientNet-lite 的 ImageNet top-1 分別損失約 2.5 與 4.2 個點，他們推測原因是深度可分離卷積；逐通道尺度為 EfficientNet-lite 找回了 2.8 個點 [15]。各通道權重範圍差異很大的層，是最先該考慮逐通道尺度或較高精度的候選，而逐層敏感度測試（一次只量化一層並測量下降，如練習 5）可以很快找出其他的層。

偵測器

偵測器預測連續的框座標與分數，兩者都會受粗糙格點之害。在白皮書中，EfficientDet-D1 在浮點下達到 40.08 COCO mAP，8 位元 PTQ 下是 38.29，但 4 位元逐張量權重只有 0.31（逐通道是 35.08）；QAT 讓 4 位元權重的結果提高到 35.34 與 36.75 [15]。YOLOv6 這類重參數化的偵測器需要對量化友善的訓練（見上文）[40]。因為嚴格 IoU 下的準確率是粗糙的框座標最先傷到的，所以量化之後要檢查高 IoU 閾值的 AP，而不只是 IoU 0.5 的 AP（DLCV 03A）。

蒸餾對偵測器也需要小心。Wang 等人發現，把一般的蒸餾套用到偵測器只有很小的增益，因此改讓學生在物件附近的位置模仿教師的特徵（位置由錨框估計）；這在他們的實驗中改善了 KITTI、Pascal VOC 與 COCO 上的學生 [51]。

密集預測與影像復原

對超解析度、去雜訊與增強而言，輸出是一張影像，每個量化誤差都可能變得看得見。有兩個性質讓這些網路比分類器更難。第一，許多影像復原網路拿掉了批次正規化；Li 等人發現這會讓激活值的動態範圍大得多，破壞了假設激活值已正規化的量化方案，並提出了為每層學習最大值尺度的做法（PAMS）[52]。第二，Hong 等人觀察到，超解析度網路中每個通道的激活值分布都不同，而且還會隨輸入影像改變，並提出逐通道、感知分布的量化（DAQ）[53]。所以要用與部署內容相似的影像做校準（低光源模型就用暗景），用 DLCV 09 的指標在完整解析度下評估，並檢查平滑的漸層，色帶最先出現在那裡。NTIRE 高效率超解析度挑戰賽在固定的 PSNR 閾值下，把執行時間、FLOPs 與參數量分成不同賽道排名 [54]（DLCV 09A）；DLCV 09C 涵蓋高效率的增強網路。

視覺 Transformer 與擴散模型

在視覺 Transformer 中，softmax 與 GELU 之後的激活值遠非高斯分布，PTQ4ViT 對每個這樣的張量使用兩個範圍來處理 [55]；Liu 等人加入一個排序損失，讓量化後的注意力保留注意力分數的順序 [56]。擴散模型多了時間維度：同一個去雜訊網路要在許多雜訊等級下執行，它的激活值分布會隨時間步改變。Q-Diffusion 把這點，連同捷徑層中的雙峰激活值，視為關鍵困難，並使用感知時間步的校準 [57]；PTQ4DM 研究的是同一個多時間步校準的問題 [58]。大型分割基礎模型則靠蒸餾來壓縮：MobileSAM 蒸餾 Segment Anything Model [59] 的大型影像編碼器成一個小的（「解耦蒸餾」）[60]，EfficientSAM 則預訓練小型編碼器去重建 SAM 的特徵 [61]。

現代觀點

綜述怎麼說

- **一般壓縮**。 Cheng 等人把這個領域分成四類：參數剪枝與量化、低秩分解、轉移式／緊湊卷積濾波器，以及知識蒸餾，並討論每一類的優缺點 [62]。Menghani 把高效深度學習整理成五個領域：壓縮技術、學習技術（如蒸餾）、自動化（超參數最佳化與架構搜尋）、高效架構，以及基礎設施（軟體與硬體），並附上實務者指南 [63]。

- **量化**。 Gholami 等人從基礎（均勻與非均勻量化器、範圍校準、粒度、微調）到進階主題（純整數推論、混合精度、考慮硬體與蒸餾輔助的量化、極低位元量化）綜述量化 [64]；Nagel 等人的白皮書是實用的配套，有經過測試的 PTQ 與 QAT 流程以及除錯工作流程 [15]。
- **剪枝與稀疏**。 Hoefler 等人整理了 300 多篇關於推論與訓練稀疏的論文，包括如何在硬體上得到真正的加速 [8]。Cheng、Zhang 與 Shi 依通用或特定加速、何時剪枝、如何剪枝，以及與其他壓縮技術的融合來整理剪枝，並涵蓋大型語言與多模態模型的剪枝 [65]。Blalock 等人仍是剪枝結果該如何回報這件事的警示性參考 [4]。
- **蒸餾**。 Gou 等人依知識類型、訓練方案、教師—學生架構與演算法對蒸餾分類 [29]；Wang 與 Yoon 則聚焦於視覺，以及蒸餾為什麼有效 [66]。
- **架構搜尋**。 Elsken 等人提供了上面使用的三維度分類法 [41]；White 等人在超過 1,000 篇 NAS 論文之後寫作，給出搜尋空間、演算法與加速技巧的分類法，並附上基準與最佳實務 [67]。

僅線上版：*A White Paper on Neural Network Quantization*（*Nagel* 等人，*2021*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/model-compression/papers/a-white-paper-on-neural-network-quantizationnagel-等人 2021/*

2023–2026 年的最新進展

- **超大型模型的低位元權重**。以補償誤差的捨入（GPTQ [68]）或感知激活值並保護重要通道（AWQ [69]）的方式，把權重量化到 3–4 位元，已成為大型 Transformer 的標準做法；ML 11 介紹了這兩者。SmoothQuant 把離群值從激活值遷移到權重，使語言模型的 8 位元權重與激活值變得可行 [19]。
- **4 位元的生成式視覺模型**。繼 Q-Diffusion [57] 之後，SVDQuant 把大型擴散模型的權重與激活值都量化到 4 位元：它把激活值的離群值移進權重，再用一個小的高精度低秩分支吸收權重的離群值，並把該分支融合進低位元核，以避免額外的記憶體流量；它結合了本章的兩項技術：低秩分解與量化 [70]。
- **8 位元浮點格式**（E4M3、E5M2），在訓練與推論上作為 int8 的替代方案 [24]。
- **蒸餾後的基礎模型**。大型視覺基礎模型的小型學生，例如分割用的 MobileSAM 與 EfficientSAM [60]、[61]，以及 DLCV 02 討論的蒸餾版 DINOv2 骨幹。
- **適用於任何計算圖的結構化剪枝**。 DepGraph 式的相依分析，讓通道剪枝能用同一個方法套用到 CNN、Transformer 與其他架構 [9]。
- **由挑戰賽推動的高效影像復原**，其中執行時間、FLOPs 與參數量是分開計分的 [54]。

尚未解決的問題

- **預測裝置速度**。 FLOPs 與參數量仍然主導論文，但只有目標裝置上量測到的延遲才重要，而它取決於核、記憶體與批次大小 [1]。
- **公平的比較**。大小與訓練預算相同的基準、多個種子，以及完整的取捨曲線，仍然常常缺少 [4]、[3]。
- **低位元激活值**。 4 位元權重已越來越常規；4 位元激活值，尤其是在含有深度層、注意力與隨時間變化的分布的網路中，仍然要付出準確率的代價 [15]、[57]。
- **壓縮在準確率之外改變了什麼**。平均準確率可能掩蓋稀有類別、偏移輸入或校準上的變化；壓縮之後去量測這些，仍不是標準做法。

- **影像復原與生成模型的壓縮**，其誤差在輸出中看得見，而沒有單一指標能完整反映。

重點整理

- 定義實際部署預算，並在目標裝置量測端到端成本。
- 減少 MACs 不一定改善記憶體受限層；應檢查搬移位元組數與可用核心。
- 非結構化剪枝需稀疏運算支援才能加速；結構化剪枝產生較小稠密層，仍須實測。
- 先試 PTQ，檢查校準與離群值；所需精度造成過大準確率損失時再用 QAT。
- 蒸餾取決於教師資訊、對應輸入視圖與訓練方式；教師較大不保證較好。
- 匯出後驗證壓縮模型及敏感輸出，並與同樣充分訓練的小型稠密模型比較。

練習

1. **屋頂線算術**。在第一段程式碼的虛構加速器（4 TMAC/s、100 GB/s）上，估計一個在 28×28 特徵圖上、128 個輸入與 128 個輸出通道的 3×3 卷積，在 16 位元精度下的時間。它是受限於運算還是記憶體？接著找出一個 1024×1024 的 fp32 線性層要變成受限於運算的最小批次大小。

   提示

   這個卷積做 $9 \cdot 128 \cdot 128 \cdot 784 \approx 115.6$ M 次 MAC，搬動 $147{,}456 + 2 \cdot 128 \cdot 784 = 348{,}160$ 個值，也就是每個值 2 位元組時約 0.70 MB，所以它的強度約為每位元組 166 次 MAC：受限於運算，約 28.9 µs。對批次 B 的線性層，強度是 $B \cdot 1024^2 / (4\,(1024^2 + 2048B))$；令它等於 40 得到 $B \geq 233$。可以改寫程式碼中的 `layer` 函式來檢查。

2. **大而稀疏，還是小而稠密？**在剪枝程式碼中，窄的稠密網路在約 2,500 個權重時勝過稀疏網路，與 Zhu 與 Gupta 在大型模型上的結果不同。設計一個能讓這個比較更公平的玩具實驗（想想每個模型被允許訓練多久、如何初始化），並預測如果原始網路本來就很窄，例如 64–32–16–10，結果會如何改變。

   提示

   讓兩個模型有相同的最佳化器總步數，分別調整它們的學習率，並加入第三個模型：從頭訓練的窄架構（Liu 等人主張的那種基準）。一個本來就很窄的網路沒有多少冗餘可以移除，所以結構化剪枝應該會更快失去準確率，而能把重要權重保留在任何位置的非結構化模型，可能反而勝出。

3. **手動量化**。用仿射方案與 min-max 範圍，把數值 $-0.8, 0, 0.3, 2.4$ 量化成 4 位元無號整數。計算 s、z、每個 q 與每個 $\hat{x}$。為什麼 -0.8 的誤差不是零，儘管它就是範圍的一端？誤差有沒有可能大於 $s/2$？

   提示

   $s = 3.2/15 \approx 0.2133$，$z = 0 - \text{round}(-0.8/0.2133) = -\text{round}(-3.75) = 4$，所以 $q = 0, 4, 5, 15$，$\hat{x} \approx -0.853, 0, 0.213, 2.347$。把 z 捨入成整數，會讓整個格點平移，使 0 恰好落在格點上；格點因此涵蓋 $[-0.853, 2.347]$ 而不是 $[-0.8, 2.4]$，所以兩個端點值都不在格點上（兩個誤差都約為 0.053，小於 $s/2 \approx 0.107$）。高於 $2.347 + s/2$ 或低於 $-0.853 - s/2$ 的值會被截斷，它的誤差就可能超過 $s/2$。

4. **蒸餾中的溫度與權重**。用 $T \in \{2, 8\}$ 與 $\alpha \in \{0.5, 0.9\}$ 重新執行蒸餾程式碼。哪些設定對 8 單元的學生有幫助？為什麼很高的溫度可能比適中的溫度幫助更小？請在回答中使用軟項的梯度 $\alpha T\,(\sigma(\mathbf{z}/T) - \sigma(\mathbf{v}/T))$。

   提示

   有一次執行中，$T = 2$（$\alpha = 0.5$ 與 0.9）得到 0.948 與 0.953，但 $T = 8$ 得到 0.937 與 0.916，而只用標籤是 0.936。在很高的溫度下，軟目標趨近均勻分布，對正確類別說得更少，而 $\alpha = 0.9$ 時標籤項剩下的權重很小；因子 αT 也會改變有效的步長，所以學習率會與 T 互相影響。下結論之前，先拿差異和各種子之間的變異比較。

5. **逐層敏感度**。用離群值程式碼中的函式 `q_weight` 與 `run`，**一次只把一層**的權重量化到 2 位元（逐通道），其他層保持浮點，並依準確率下降幅度為三層排名。然後只把最穩健的兩層量化到 2 位元，把最敏感的那層保持在 8 位元。這個混合精度模型需要的權重記憶體，與全部 8 位元相比如何？

   提示

   傳給 `run` 一串量化與未量化的權重矩陣，而不是使用它的 `wbits` 參數。三層的權重數分別是 $64 \cdot 256$、$256 \cdot 128$ 與 $128 \cdot 10$；記憶體是 $\sum_\ell P_\ell\, b_\ell / 8$ 位元組，再加上每個輸出單元一個尺度。這就是 HAWQ 這類以敏感度為基礎的混合精度的想法，只是用量測到的準確率取代了 Hessian 資訊。

參考文獻

1. M. Dehghani, A. Arnab, L. Beyer, A. Vaswani and Y. Tay, “The efficiency misnomer,” ICLR, 2022. arXiv
2. S. Williams, A. Waterman and D. Patterson, “Roofline: an insightful visual performance model for multicore architectures,” *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009. doi
3. Z. Liu, M. Sun, T. Zhou, G. Huang and T. Darrell, “Rethinking the value of network pruning,” ICLR, 2019. arXiv
4. D. Blalock, J. J. Gonzalez Ortiz, J. Frankle and J. Guttag, “What is the state of neural network pruning?,” MLSys, 2020. arXiv
5. H. Li, A. Kadav, I. Durdanovic, H. Samet and H. P. Graf, “Pruning filters for efficient ConvNets,” ICLR, 2017. arXiv
6. A. Mishra, J. Albericio Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu and P. Micikevicius, “Accelerating sparse deep neural networks,” arXiv:2104.08378, 2021. arXiv
7. Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan and C. Zhang, “Learning efficient convolutional networks through network slimming,” ICCV, 2017. arXiv
8. T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden and A. Peste, “Sparsity in deep learning: pruning and growth for efficient inference and training in neural networks,” *Journal of Machine Learning Research*, vol. 22, no. 241, pp. 1–124, 2021. paper
9. G. Fang, X. Ma, M. Song, M. B. Mi and X. Wang, “DepGraph: towards any structural pruning,” CVPR, 2023. doi arXiv
10. S. Han, J. Pool, J. Tran and W. J. Dally, “Learning both weights and connections for efficient neural networks,” NeurIPS (NIPS), 2015. arXiv

11. S. Han, H. Mao and W. J. Dally, “Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding,” ICLR, 2016. arXiv
12. M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” arXiv:1710.01878, 2017. arXiv
13. J. Frankle and M. Carbin, “The lottery ticket hypothesis: finding sparse, trainable neural networks,” ICLR, 2019. arXiv
14. J. Frankle, G. K. Dziugaite, D. M. Roy and M. Carbin, “Linear mode connectivity and the lottery ticket hypothesis,” ICML, 2020. arXiv
15. M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen and T. Blankevoort, “A white paper on neural network quantization,” arXiv:2106.08295, 2021. arXiv
16. B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” CVPR, 2018. doi arXiv
17. M. Nagel, M. van Baalen, T. Blankevoort and M. Welling, “Data-free quantization through weight equalization and bias correction,” ICCV, 2019. arXiv
18. T. Dettmers, M. Lewis, Y. Belkada and L. Zettlemoyer, “LLM.int8(): 8-bit matrix multiplication for transformers at scale,” NeurIPS, 2022. arXiv
19. G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth and S. Han, “SmoothQuant: accurate and efficient post-training quantization for large language models,” ICML, 2023. arXiv
20. M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos and T. Blankevoort, “Up or down? Adaptive rounding for post-training quantization,” ICML, 2020. arXiv
21. Y. Bengio, N. Léonard and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” arXiv:1308.3432, 2013. arXiv
22. Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney and K. Keutzer, “HAWQ: Hessian aware quantization of neural networks with mixed-precision,” ICCV, 2019. arXiv
23. K. Wang, Z. Liu, Y. Lin, J. Lin and S. Han, “HAQ: hardware-aware automated quantization with mixed precision,” CVPR, 2019. arXiv
24. P. Micikevicius et al., “FP8 formats for deep learning,” arXiv:2209.05433, 2022. arXiv
25. G. Hinton, O. Vinyals and J. Dean, “Distilling the knowledge in a neural network,” NIPS Deep Learning Workshop, 2014; arXiv:1503.02531. arXiv
26. L. J. Ba and R. Caruana, “Do deep nets really need to be deep?,” NeurIPS (NIPS), 2014. arXiv
27. A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta and Y. Bengio, “FitNets: hints for thin deep nets,” ICLR, 2015. arXiv
28. S. Zagoruyko and N. Komodakis, “Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer,” ICLR, 2017. arXiv
29. J. Gou, B. Yu, S. J. Maybank and D. Tao, “Knowledge distillation: a survey,” *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021. doi
30. L. Beyer, X. Zhai, A. Royer, L. Markeeva, R. Anil and A. Kolesnikov, “Knowledge distillation: a good teacher is patient and consistent,” CVPR, 2022. arXiv
31. J. H. Cho and B. Hariharan, “On the efficacy of knowledge distillation,” ICCV, 2019. arXiv
32. H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles and H. Jégou, “Training data-efficient image transformers & distillation through attention,” ICML, 2021. arXiv
33. R. Denton, W. Zaremba, J. Bruna, Y. LeCun and R. Fergus, “Exploiting linear structure within convolutional networks for efficient evaluation,” NeurIPS (NIPS), 2014. arXiv

34. M. Jaderberg, A. Vedaldi and A. Zisserman, “Speeding up convolutional neural networks with low rank expansions,” BMVC, 2014. arXiv
35. X. Zhang, J. Zou, K. He and J. Sun, “Accelerating very deep convolutional networks for classification and detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, 2016. doi arXiv
36. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, “MobileNets: efficient convolutional neural networks for mobile vision applications,” arXiv:1704.04861, 2017. arXiv
37. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, “MobileNetV2: inverted residuals and linear bottlenecks,” CVPR, 2018. arXiv
38. M. Tan and Q. V. Le, “EfficientNet: rethinking model scaling for convolutional neural networks,” ICML, 2019. arXiv
39. X. Ding, X. Zhang, N. Ma, J. Han, G. Ding and J. Sun, “RepVGG: making VGG-style ConvNets great again,” CVPR, 2021. arXiv
40. C. Li et al., “YOLOv6: a single-stage object detection framework for industrial applications,” arXiv:2209.02976, 2022. arXiv
41. T. Elsken, J. H. Metzen and F. Hutter, “Neural architecture search: a survey,” *Journal of Machine Learning Research*, vol. 20, pp. 1–21, 2019. arXiv
42. B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” ICLR, 2017. arXiv
43. H. Pham, M. Y. Guan, B. Zoph, Q. V. Le and J. Dean, “Efficient neural architecture search via parameter sharing,” ICML, 2018. arXiv
44. H. Liu, K. Simonyan and Y. Yang, “DARTS: differentiable architecture search,” ICLR, 2019. arXiv
45. H. Cai, C. Gan, T. Wang, Z. Zhang and S. Han, “Once-for-all: train one network and specialize it for efficient deployment,” ICLR, 2020. arXiv
46. M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard and Q. V. Le, “MnasNet: platform-aware neural architecture search for mobile,” CVPR, 2019. arXiv
47. H. Cai, L. Zhu and S. Han, “ProxylessNAS: direct neural architecture search on target task and hardware,” ICLR, 2019. arXiv
48. B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia and K. Keutzer, “FBNet: hardware-aware efficient ConvNet design via differentiable neural architecture search,” CVPR, 2019. doi arXiv
49. L. Li and A. Talwalkar, “Random search and reproducibility for neural architecture search,” UAI, 2019. arXiv
50. S. Ioffe and C. Szegedy, “Batch normalization: accelerating deep network training by reducing internal covariate shift,” ICML, 2015. arXiv
51. T. Wang, L. Yuan, X. Zhang and J. Feng, “Distilling object detectors with fine-grained feature imitation,” CVPR, 2019. arXiv
52. H. Li, C. Yan, S. Lin, X. Zheng, Y. Li, B. Zhang, F. Yang and R. Ji, “PAMS: quantized super-resolution via parameterized max scale,” ECCV, 2020. arXiv
53. C. Hong, H. Kim, S. Baik, J. Oh and K. M. Lee, “DAQ: channel-wise distribution-aware quantization for deep image super-resolution networks,” WACV, 2022. arXiv
54. B. Ren, Y. Li, N. Mehta, R. Timofte et al., “The ninth NTIRE 2024 efficient super-resolution

challenge report," CVPR Workshops, 2024. doi arXiv
55. Z. Yuan, C. Xue, Y. Chen, Q. Wu and G. Sun, "PTQ4ViT: post-training quantization for vision transformers with twin uniform quantization," ECCV, 2022. doi arXiv
56. Z. Liu, Y. Wang, K. Han, S. Ma and W. Gao, "Post-training quantization for vision transformer," NeurIPS, 2021. arXiv
57. X. Li, Y. Liu, L. Lian, H. Yang, Z. Dong, D. Kang, S. Zhang and K. Keutzer, "Q-Diffusion: quantizing diffusion models," ICCV, 2023. doi arXiv
58. Y. Shang, Z. Yuan, B. Xie, B. Wu and Y. Yan, "Post-training quantization on diffusion models," CVPR, 2023. arXiv
59. A. Kirillov et al., "Segment anything," ICCV, 2023. doi arXiv
60. C. Zhang, D. Han, Y. Qiao, J. U. Kim, S.-H. Bae, S. Lee and C. S. Hong, "Faster segment anything: towards lightweight SAM for mobile applications," arXiv:2306.14289, 2023. arXiv
61. Y. Xiong et al., "EfficientSAM: leveraged masked image pretraining for efficient segment anything," CVPR, 2024. doi arXiv
62. Y. Cheng, D. Wang, P. Zhou and T. Zhang, "Model compression and acceleration for deep neural networks: the principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 126–136, 2018. doi arXiv
63. G. Menghani, "Efficient deep learning: a survey on making deep learning models smaller, faster, and better," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023. doi arXiv
64. A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney and K. Keutzer, "A survey of quantization methods for efficient neural network inference," book chapter in *Low-Power Computer Vision*; arXiv:2103.13630, 2021. arXiv
65. H. Cheng, M. Zhang and J. Q. Shi, "A survey on deep neural network pruning: taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10558–10578, 2024. doi arXiv
66. L. Wang and K.-J. Yoon, "Knowledge distillation and student-teacher learning for visual intelligence: a review and new outlooks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 6, pp. 3048–3068, 2022. doi arXiv
67. C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey and F. Hutter, "Neural architecture search: insights from 1000 papers," arXiv:2301.08727, 2023. arXiv
68. E. Frantar, S. Ashkboos, T. Hoefler and D. Alistarh, "GPTQ: accurate post-training quantization for generative pre-trained transformers," ICLR, 2023. arXiv
69. J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan and S. Han, "AWQ: activation-aware weight quantization for LLM compression and acceleration," MLSys, 2024. arXiv
70. M. Li, Y. Lin, Z. Zhang, T. Cai, X. Li, J. Guo, E. Xie, C. Meng, J.-Y. Zhu and S. Han, "SVDQuant: absorbing outliers by low-rank components for 4-bit diffusion models," ICLR, 2025. arXiv
71. Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang and S. Gu, "BRECQ: pushing the limit of post-training quantization by block reconstruction," ICLR, 2021. arXiv
72. S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy and D. S. Modha, "Learned step size quantization," ICLR, 2020. arXiv

DLCV 02・深度影像分類

深度網路如何判斷一張影像裡是什麼：損失函數與評估指標、資料集、CNN 與 Transformer 架構背後的設計理由、現代訓練配方、預訓練、穩健性、校準與可解釋性。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/

你將學到

- 為單標籤與多標籤任務選擇輸出、損失及指標。
- 依設計與成本比較 CNN 和 Transformer 骨幹。
- 區分架構改進與訓練配方的效果。
- 選擇線性探測、微調或零樣本遷移。
- 評估類別不平衡、校準、分布轉移與部署效率。

先看全貌

影像分類對整張影像只問一個問題：這些標籤裡，哪一個最符合？它是最早的深度學習基準任務，到今天仍是大多數骨幹網路誕生的地方。物體偵測、分割或影像復原網路，通常都從一個先以影像分類訓練、或至少以影像分類評估過的骨幹網路出發。

> **用最簡單的話說**
>
> 想像一個小朋友看著圖畫書學認動物。一開始他只看到一團一團的顏色。翻過很多頁之後，他學會先注意耳朵，再注意臉，最後說出「這是貓」。深度分類器也是一層一層這樣做：前面的層注意邊緣和顏色，中間的層注意眼睛、輪子這類零件，最後一層投票選出標籤。訓練就是爸媽在旁邊說了幾千次「不對，那是狗」。這篇教學其餘的部分在講：怎麼做一本更好的圖畫書（資料）、一顆更好的腦袋（架構）、一套更好的教法（訓練配方），以及更好的考試（評估）。因為一個只看過「貓坐在沙發上」的小朋友，可能會以為「沙發」就代表「貓」。

本篇假設你已經知道卷積、CNN 的層與反向傳播。分類的古典面向（貝氏分類器、感知器、用 NumPy 寫的小型 CNN、手推 softmax 與交叉熵）在IP 13。深度學習用於影像的整體歷史在 DLCV 01。這裡我們專注、深入地談分類本身。若想快速複習激活函數與最佳化器，可以參考 ML 02・類神經網路。CNN 這一側的發展，Li 等人 [1] 的綜述是很好的單篇入門。

相關篇章。 DLCV 02A 把分類變成比對：介紹開放集合問題中的臉部辨識、嵌入與以角度邊界為基礎的損失。DLCV 01B 則比較分類網路與影像復原網路如何組合同一批積木。

分類問題的定義

白話版。分類器產生各類別分數；損失比較其機率與標籤，再更新網路以改善預測。

單標籤、多標籤與細粒度

令 $\mathbf{x} \in \mathbb{R}^{H\times W\times 3}$ 為一張 RGB 影像，$\mathcal{Y} = \{1, \dots, K\}$ 為 K 個類別的集合。參數為 θ 的網路 f_θ 把影像映射到一個 **logits** 向量 $\mathbf{z} = f_\theta(\mathbf{x}) \in \mathbb{R}^K$，每個類別一個未正規化的分數。

- **單標籤（多類別）**分類假設恰好有一個正確類別 $y \in \mathcal{Y}$。以 softmax 把 logits 轉成機率向量：

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad k = 1, \dots, K,$$

其中 p_k 是模型對 $P(y = k \mid \mathbf{x})$ 的估計。訓練時最小化**交叉熵**，也就是正確類別機率的負對數，在一個小批次上取平均：

$$\mathcal{L}_{\text{CE}} = -\frac{1}{B}\sum_{i=1}^{B} \log p_{y_i}(\mathbf{x}_i),$$

其中 B 是批次大小，y_i 是影像 $\mathbf{x}_i$ 的標籤。它對 logits 的梯度（每張影像）是 $\partial \mathcal{L}/\partial \mathbf{z} = \mathbf{p} - \mathbf{e}_y$，$\mathbf{e}_y$ 是正確類別的 one-hot 向量；推導見 DIP 第 13 章。

- **多標籤**分類允許任意個標籤同時成立（一張街景可能同時有「汽車」、「行人」、「腳踏車」）。每個類別各自接一個 sigmoid $\sigma(z_k) = 1/(1+e^{-z_k})$ 與二元交叉熵 $-\sum_k [t_k \log \sigma(z_k) + (1-t_k)\log(1-\sigma(z_k))]$，其中 $t_k \in \{0,1\}$ 表示類別 k 是否出現。評估指標也要換：以各類別的平均精確度（mean average precision）取代準確率。
- **細粒度**分類要分開外觀非常相近的子類別，例如 200 種鳥或 1,000 種飛機型號。損失函數與單標籤相同，但訊號藏在小地方（鳥喙形狀、翅膀上的條紋），類別通常呈長尾分佈，而且非專家標註者造成的標籤雜訊很常見。iNaturalist [6] 是標準的大型細粒度基準。

評估指標：top-1、top-5 與其他

Top-1 準確率是最高分類別恰好正確的影像比例。**Top-k 準確率**則只要正確類別落在前 k 高分之中就算對。ImageNet 挑戰賽回報 top-5 錯誤率，因為很多影像裡有好幾個物體，而有些類別幾乎無法區分 [2]。但 top-5 會掩蓋真正的錯誤，所以現代論文大多回報 top-1。資料不平衡時，請回報**平衡準確率**（各類別召回率的平均）或逐類別結果，因為分類器只要忽略稀有類別，也能拿到很高的準確率。

```
import torch
import torch.nn.functional as F

logits = torch.tensor([[2.0, 1.0, 0.1, -1.0],    # 2 images, 4 classes
                       [0.5, 2.5, 2.4, 0.0]])
y = torch.tensor([0, 2])

p = logits.softmax(dim=1)
ce = F.cross_entropy(logits, y)                       # mean of -log p[i, y_i]
ce_ls = F.cross_entropy(logits, y, label_smoothing=0.1)
top1 = (logits.argmax(1) == y).float().mean()
```

```
top2 = (logits.topk(2, dim=1).indices == y[:, None]).any(1).float().mean()
print(p.round(decimals=3))
print(f"CE {ce:.3f}  CE+LS {ce_ls:.3f}  top-1 {top1:.2f}  top-2 {top2:.2f}")

# multi-label: one independent sigmoid per class, binary cross-entropy
targets = torch.tensor([[1., 0., 1., 0.], [0., 1., 1., 0.]])
print(f"BCE {F.binary_cross_entropy_with_logits(logits, targets):.3f}")
```

第二張影像在類別 1 和 2 之間幾乎平手。Top-1 算它錯，top-2 算它對。支持與反對 top-5 的理由，都在這一個例子裡。

混淆矩陣與由它衍生的指標

白話版。準確率概括正確程度，混淆矩陣指出哪些類別混淆；再用適合的指標揭露重要錯誤。

二元情況：TP、FP、FN、TN

把某一類當作**陽性**（positive，例如「生病」），其餘當作**陰性**（negative）。每一個預測都會落在四個格子之一：

	預測為陽性	預測為陰性
實際為陽性	真陽性（true positive, TP）	偽陰性（false negative, FN）
實際為陰性	偽陽性（false positive, FP）	真陰性（true negative, TN）

用最簡單的話說

煙霧警報器就是一個分類器。真的失火、它響了，是真陽性。你只是把吐司烤焦、它也響了，是偽陽性：很吵，但沒有人受傷。真的失火、它卻沒響，是偽陰性：這才是危險的那一種。什麼事都沒有、它也安靜，是真陰性。好的警報器不只是「大部分時間都對」（它幾乎每天都安靜，要做到這點太容易了），而是要在失火時響、在烤吐司時安靜。不同的指標，對這兩個目標的看重程度不一樣。

設測試樣本共 $N = \mathrm{TP} + \mathrm{FP} + \mathrm{FN} + \mathrm{TN}$ 個，標準的比值為

$$\text{accuracy} = \frac{\mathrm{TP}+\mathrm{TN}}{N}, \quad \text{precision} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \quad \text{recall (TPR)} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}},$$

$$\text{specificity (TNR)} = \frac{\mathrm{TN}}{\mathrm{TN}+\mathrm{FP}}, \qquad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}} = 1 - \mathrm{TNR}.$$

- **精確率（precision）**回答「模型說是陽性的時候，有多常是對的？」當誤報代價很高時它最重要：會把正常信件藏起來的垃圾郵件過濾器、自動觸發的產品召回。
- **召回率（recall）**，又稱**敏感度**（sensitivity）或**真陽性率**（true positive rate, TPR），回答「所有真正的陽性中，我們抓到了多少？」當漏抓代價很高時它最重要：癌症篩檢、產線上的瑕疵偵測。
- **特異度**（specificity），即**真陰性率**（true negative rate, TNR），是陰性類別的召回率。**偽陽性率**（false positive rate, FPR）是它的補數，也就是被錯誤標記的陰性樣本比例。

精確率與召回率互相拉扯：降低決策門檻會標記更多樣本，召回率上升，精確率通常下降。把兩者合成一個數字的指標是：

$$F_1 = \frac{2\,\mathrm{P}\,\mathrm{R}}{\mathrm{P}+\mathrm{R}} = \frac{2\,\mathrm{TP}}{2\,\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}, \qquad F_\beta = \frac{(1+\beta^2)\,\mathrm{P}\,\mathrm{R}}{\beta^2\,\mathrm{P}+\mathrm{R}},$$

其中 P 是精確率，R 是召回率，$\beta > 0$ 表示召回率比精確率重要幾倍（F_2 偏重召回率，$F_{0.5}$ 偏重精確率）。F_1 是調和平均，所以只有兩者都高時它才會高。也請注意 F_1 忽略了什麼：真陰性完全沒有出現在式子裡。

另外兩個總結指標用到了全部四格：

$$\text{balanced accuracy} = \frac{\mathrm{TPR}+\mathrm{TNR}}{2}, \qquad \mathrm{MCC} = \frac{\mathrm{TP}\cdot\mathrm{TN} - \mathrm{FP}\cdot\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}.$$

平衡準確率（balanced accuracy）是各類別召回率的平均，所以一個忽略某一類的分類器，不論那一類多稀有，都只能拿到 0.5。**Matthews 相關係數**（Matthews correlation coefficient, MCC）是真實與預測二元標籤之間的 Pearson 相關係數：+1 表示完美，0 表示不比隨機好，−1 表示完全相反。Chicco 與 Jurman [57] 主張在二元評估中應優先使用 MCC，而不是準確率與 F_1，因為只有分類器在四個格子上都表現好時，它才會高。Sokolova 與 Lapalme [58] 系統性地比較了這些指標，以及它們各自如何隨混淆矩陣的變化而反應。

為什麼 99:1 時準確率會誤導。假設 1,000 張掃描影像中有 10 張是生病的。一個永遠回答健康的「模型」可以拿到 99.0% 的準確率。模型 B 標記了 30 張，抓到 10 張中的 8 張；它的準確率較低（97.6%），卻是唯一有用的那一個：

```
import numpy as np
from sklearn.metrics import (accuracy_score, balanced_accuracy_score, confusion_matrix,
                             f1_score, matthews_corrcoef, precision_score, recall_score)

# 1,000 scans, 10 sick (label 1). Model A says "healthy" to everyone.
# Model B flags 30 scans and catches 8 of the 10 sick ones.
y = np.r_[np.ones(10), np.zeros(990)].astype(int)
pred_a = np.zeros(1000, dtype=int)
pred_b = np.r_[np.ones(8), np.zeros(2), np.ones(22), np.zeros(968)].astype(int)

for name, p in [("all-negative", pred_a), ("model B", pred_b)]:
    tn, fp, fn, tp = confusion_matrix(y, p).ravel()
    print(f"{name:12s} TP={tp} FP={fp} FN={fn} TN={tn}  "
          f"acc={accuracy_score(y, p):.3f} bal-acc={balanced_accuracy_score(y, p):.3f} "
          f"P={precision_score(y, p, zero_division=0):.3f} R={recall_score(y, p):.3f} "
          f"F1={f1_score(y, p, zero_division=0):.3f} MCC={matthews_corrcoef(y, p):.3f}")
# all-negative TP=0 FP=0 FN=10 TN=990  acc=0.990 bal-acc=0.500 P=0.000 R=0.000 F1=0.000
 ↪  MCC=0.000
# model B      TP=8 FP=22 FN=2 TN=968  acc=0.976 bal-acc=0.889 P=0.267 R=0.800 F1=0.400
 ↪  MCC=0.454
```

平衡準確率、F_1 與 MCC 都把模型 B 排在第一；準確率卻把它排在最後。經驗法則：只有在類別大致平衡、所有錯誤代價相同時才用準確率；當某一種錯誤比較嚴重時，回報精確率與召回率（或依實際代價選定 β 的 F_β）；需要一個能反映不平衡的單一總結數字時，用平衡準確率或 MCC。

多類別混淆矩陣

有 K 個類別時，混淆矩陣 $C \in \mathbb{N}^{K\times K}$ 的元素 C_{ij} 是**真實**類別為 i（列）、**預測**類別為 j（行）的測試影像數；這是 scikit-learn [62] 的慣例。對角線是預測正確的數量。把每一列除以該列總和，得到**依列正規化**的矩陣：它的對角線就是各類別的召回率，非對角元素則顯示哪些類別被搞混。圖 1 是一個只用 scikit-learn 8×8 手寫數字中 200 張訓練的邏輯迴歸所得到的結果：大部分錯誤是 3 → 8、4 → 1 與 7 → 9，都是筆畫相似的組合。

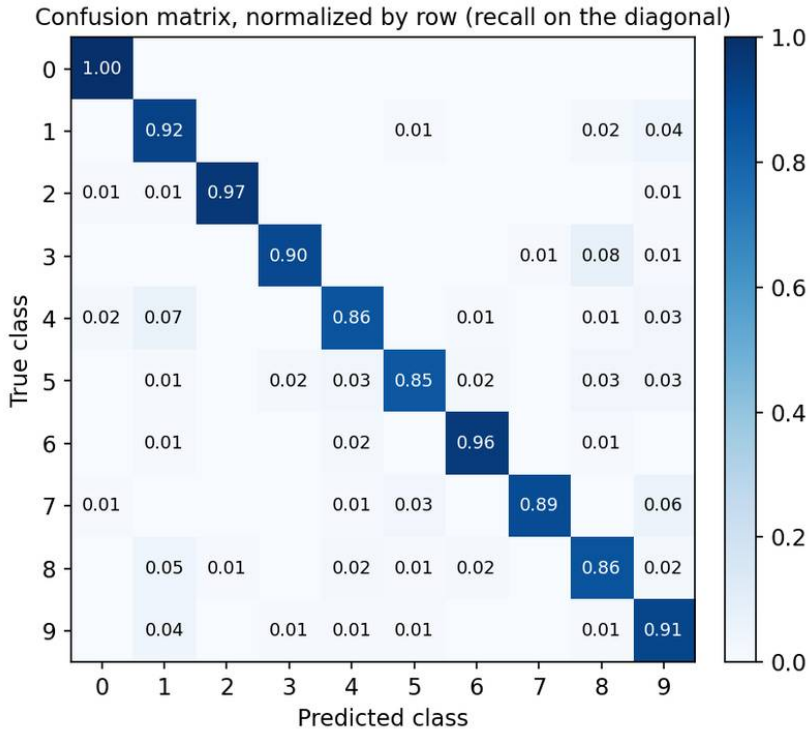


圖 *1* — 以 *200* 張數字訓練、在 *899* 張上測試的邏輯迴歸之依列正規化混淆矩陣（準確率 *91.1%*）。每一列總和為 *1*；對角線是各類別的召回率，較亮的非對角格指出值得修正的混淆。

輪流把每個類別 k 當作「陽性」、其他類別當作「陰性」，就得到逐類別的計數：$\mathrm{TP}_k = C_{kk}$，$\mathrm{FP}_k = \sum_{i\neq k} C_{ik}$（第 k 行的其餘部分），$\mathrm{FN}_k = \sum_{j\neq k} C_{kj}$（第 k 列的其餘部分）。因此逐類別精確率 $\mathrm{P}_k = C_{kk}/\sum_i C_{ik}$，召回率 $\mathrm{R}_k = C_{kk}/\sum_j C_{kj}$。把 K 個逐類別數字合成一個，常見的有三種做法：

- **巨觀平均**（macro average）：對類別直接取平均，$\mathrm{F}_1^{\text{macro}} = \frac{1}{K}\sum_k \mathrm{F}_{1,k}$。每個類別權重相同，所以稀有類別和常見類別一樣重要。
- **微觀平均**（micro average）：先把各類別的 TP_k、FP_k、FN_k 加總，再算一次指標。在單標籤多類別問題中，每一個錯誤都同時是一個 FP（對預測的類別）和一個 FN（對真實的類別），所以微觀精確率 = 微觀召回率 = 微觀 F_1 = 準確率。
- **加權平均**（weighted average）：以各類別的**支持數**（support）n_k（真實樣本數）為權重，對逐類別數值取平均。它很接近微觀平均，幾乎一樣會掩蓋稀有類別的失敗。

在不平衡的測試集上，幾種平均可能差很多。如果模型在常見類別上很好、在稀有類別上很差，

微觀平均（被常見類別主導）會給出高分，巨觀平均則低得多；這個差距本身就是訊號。當每個類別都重要時（罕見疾病、稀有物種）回報巨觀平均；當每一張影像同等重要時，回報微觀平均／準確率。

僅線上版：巨觀與微觀平均的計算實例 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/巨觀與微觀平均的計算實例/*

Cohen's kappa [59] 是扣除巧合後的準確率：它把預測與標籤之間觀察到的一致程度，和一個只依觀察到的比例隨機猜各類別的分類器所能達到的一致程度相比。完全一致時為 1，與隨機相當時為 0；在比較模型與人類標註者時很常見。

僅線上版：*Cohen's kappa*：扣除巧合的一致性 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/cohens-kappa 扣除巧合的一致性/*

```
from sklearn.datasets import load_digits
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (average_precision_score, classification_report,
                             confusion_matrix, precision_recall_fscore_support,
                              ↪  roc_auc_score)
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import label_binarize

X, y = load_digits(return_X_y=True)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.5, stratify=y, random_state=0)
clf = LogisticRegression(max_iter=2000).fit(X_tr[:200], y_tr[:200])   # only 200 labels
pred, prob = clf.predict(X_te), clf.predict_proba(X_te)

cm = confusion_matrix(y_te, pred)                     # rows = true, columns = predicted
cm_norm = confusion_matrix(y_te, pred, normalize="true")   # each row sums to 1 (Figure 1)
print(classification_report(y_te, pred, digits=3))    # per-class P, R, F1, support +
 ↪  averages
for avg in ["macro", "micro", "weighted"]:
    p, r, f, _ = precision_recall_fscore_support(y_te, pred, average=avg)
    print(f"{avg:8s} P={p:.3f} R={r:.3f} F1={f:.3f}")
print(f"ROC-AUC (one-vs-rest, macro) {roc_auc_score(y_te, prob, multi_class='ovr'):.4f}")
Y = label_binarize(y_te, classes=range(10))           # one binary column per class
print(f"mean AP over classes {average_precision_score(Y, prob, average='macro'):.4f}")
# macro    P=0.914 R=0.911 F1=0.912
# micro    P=0.911 R=0.911 F1=0.911
# weighted P=0.915 R=0.911 F1=0.912
# ROC-AUC (one-vs-rest, macro) 0.9958
# mean AP over classes 0.9742
```

這裡巨觀與微觀平均幾乎相同，因為手寫數字的各類別是平衡的；換成不平衡的測試集就不會如此。

不依賴門檻的評估：ROC 曲線與精確率–召回率曲線

上面每一個指標都需要一條決策規則（argmax，或「分數超過 t 就是陽性」）。會輸出分數的分類器，可以一次在所有門檻上評估：把 t 從高掃到低，每個門檻記下一個點。

- **ROC 曲線**畫的是 TPR 對 FPR。曲線下面積 **ROC-AUC** 等於「隨機取一個陽性樣本，它的分數高於隨機取的一個陰性樣本」的機率；0.5 是隨機，1.0 是完美分開。Fawcett [54] 是標準的入門文獻。
- **精確率–召回率（PR）曲線**畫的是精確率對召回率。它的總結指標**平均精確度**（average precision, AP），是在每次多找回一個陽性樣本的召回率位置上，對精確率取平均：

$$\mathrm{AP} = \sum_{n} (\mathrm{R}_n - \mathrm{R}_{n-1})\, \mathrm{P}_n,$$

其中 P_n 與 R_n 是依分數由高到低排序後、第 n 個門檻的精確率與召回率（這是 scikit-learn [62] 的定義；PASCAL VOC [61] 等基準用過內插的變體，所以要確認論文回報的是哪一種）。

為什麼嚴重不平衡時要看 PR。 FPR 的分母是陰性樣本數。當陰性樣本是陽性的 99 倍時，即使有上千個誤報，FPR 也幾乎不動，所以 ROC 曲線看起來很漂亮，精確率卻很差。精確率的分母是被標記的樣本數，誤報會直接打擊它，而 AP 的隨機基準線等於陽性比例，而不是 0.5。Davis 與 Goadrich [55] 證明：一條曲線在 ROC 空間中占優勢，若且唯若它在 PR 空間中也占優勢；並指出最佳化 ROC-AUC 並不等於最佳化 AP。Saito 與 Rehmsmeier [56] 以模擬與文獻回顧顯示，在不平衡資料上，PR 圖比 ROC 圖更有資訊量。圖 2 把這件事具體化：在陽性只占 1% 的任務上，兩個 ROC-AUC 相同（0.95）的分類器，AP 卻明顯不同（0.45 對 0.36）。

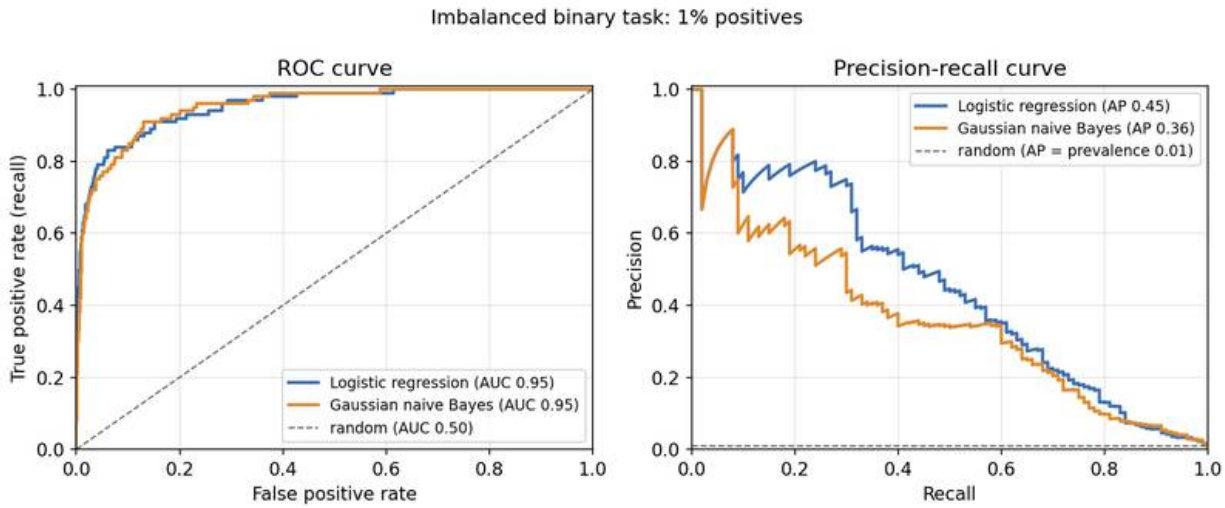


圖 2 — 在陽性只占 *1%* 的合成任務上（*10,000* 個測試點中有 *100* 個陽性），兩個分類器的 *ROC* 曲線（左）與精確率–召回率曲線（右）。兩者 *ROC-AUC* 都是 *0.95*，但邏輯迴歸在每個召回率下的精確率都高得多（*AP 0.45* 對 *0.36*）。在這個任務上，一個全部判為陰性的分類器可以拿到 *99%* 的準確率。

當類別數 $K > 2$ 時，兩種曲線都以一**對其餘**（one-vs-rest）的方式計算：類別 k 為陽性、其他全部為陰性，使用分數 p_k。再把 K 個面積做巨觀平均（`roc_auc_score(..., multi_class="ovr")`），或把所有（類別, 影像）配對合併後做微觀平均。

多標籤指標

在多標籤分類中，每個標籤都是自己的一個「有／沒有」問題，所以標準的總結指標是**逐類別的平均精確度**，再對類別平均成 **mAP**，就像 PASCAL VOC 分類任務 [61] 的做法。需要門檻的指標，要嘛對每個標籤算二元指標再對標籤平均（巨觀或微觀，如上），要嘛對每張影像的預測標籤集合評分，再對影像取平均。

僅線上版：多標籤指標細說 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/多標籤指標細說/*

校準：機率說的是實話嗎？

準確率只檢查 argmax。**校準**（calibration）檢查信心值是否名副其實：在所有信心為 0.8 的預測中，應該大約有 80% 是對的。正式地說，若對每個信心值 c 都有 $P(\hat{y} = y \mid \hat{p} = c) = c$，分類器就是校準良好的，其中 $\hat{y}$ 是預測類別，$\hat{p} = \max_k p_k$ 是它的機率。常用的總結指標是**期望校準誤差**（expected calibration error, ECE）：把預測依信心分到 M 個區間 B_m，計算

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left| \mathrm{acc}(B_m) - \mathrm{conf}(B_m) \right|,$$

其中 n 是測試影像數，$\mathrm{acc}(B_m)$ 是第 m 個區間內的準確率，$\mathrm{conf}(B_m)$ 是區間內的平均信心。Guo 等人 [47] 發現，用交叉熵長時間訓練的現代深度網路傾向**過度自信**，而只用一個純量就能修正得相當好。我們會在穩健性一節回來談。

資料集與它們的角色

白話版。基準讓方法在固定條件下比較。訓練與評估資料須分開，並檢查提升能否遷移到該測試集之外。

訓練用資料集

資料集	影像數	類別數	解析度	角色
MNIST [3]	訓練 6 萬／測試 1 萬	10 個數字	28×28 灰階	健全性檢查、教學
CIFAR-10 / CIFAR-100 [4]	訓練 5 萬／測試 1 萬	10 ／ 100（20 個超類別）	32×32 RGB	小規模的架構與配方研究
ImageNet-1k（ILSVRC 2012）[2]	約 128 萬訓練／ 5 萬驗證	1,000	不固定，通常以 224×224 訓練	骨幹網路的標準基準
ImageNet-21k [5]	約 1,300–1,400 萬（版本不同）	約 2.1 萬	不固定	更大規模的監督式預訓練
iNaturalist [6]	數十萬，依年份而異	數千個物種	不固定	細粒度、長尾

MNIST 已經被解決，現在只用來抓程式錯誤。CIFAR 小到可以跑大量受控實驗，所以正則化論文特別愛用。ImageNet-1k 在 2012 年挑戰賽之後成為參考標準 [2], [10]：架構通常以「ImageNet-1k 驗證集、224×224 的 top-1」來回報，模型庫裡幾乎每個預訓練骨幹都附有一個 ImageNet 分

數。ImageNet-21k 是 ImageNet-1k 所取自的更大階層；Ridnik 等人 [5] 以清理過、考慮語意階層的配方，讓在它上面預訓練變得實際可行。

評估用基準資料集

白話版。訓練資料負責教，評估資料負責考。只考過一份考卷的模型，可能只是學會了那份考卷。所以現代論文會用一整組測試集來評估骨幹網路，每一個都設計來探測一件事：原始分佈、從同一分佈重新抽的樣本、損壞或風格化的影像、不尋常的視角、細粒度類別、很少的標註預算，或稀有類別。

最常見的評估集可以分成五組：

- **分佈內**。 ImageNet-1k 驗證集 [2] 是實質上的測試集；**ImageNet-ReaL** [8] 為同一批影像重新標註，允許一張影像有多個標籤。
- **重新抽樣**。 **ImageNetV2** [7] 依原始收集流程重做一次，得到新的測試影像。
- **分佈偏移**。 **ImageNet-C** [42] 加上合成損壞，**ImageNet-R** [43] 收錄藝術、卡通等各種表現形式，**ImageNet-A** [44] 收集會騙倒 ResNet-50 的自然照片，**ImageNet-Sketch** [63] 是素描，**ObjectNet** [64] 則呈現不尋常旋轉、背景與視角下的物體。
- **長尾與細粒度**。 **ImageNet-LT** [73] 在長尾子集上訓練、在平衡資料上測試；iNaturalist [6]、CUB-200-2011 [65] 與 Stanford Cars [66] 測試細粒度辨識。
- **遷移**。 Kornblith 等人 [33] 的 12 個資料集與 **VTAB** [72] 這類測試組，衡量凍結或微調後的特徵能多好地適應小型、專門的任務。

兩點實務提醒。好幾個細粒度基準回報的是**各類別平均準確率**，而不是一般的準確率，不要在同一個比較裡混用兩者。另外，CLIP [40] 這類影像–文字模型常以**零樣本**方式評估，完全不在目標資料上訓練：這時提示的寫法就成了超參數，不能在測試集上調整；而網路規模的預訓練資料也可能早已包含測試影像。

僅線上版：評估用基準資料集細說 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/評估用基準資料集細說/*

這些數字有多可靠？

有三條研究路線改變了我們解讀 ImageNet 準確率的方式。照原始流程重建測試集（ImageNetV2 [7]）讓所有模型都掉分，但排名大致保持。重新標註的研究 [8], [9] 顯示，單一標籤與標籤錯誤都可能改變哪個模型看起來最好。而每個資料集在視角與背景上都有自己的偏差，所以只要可能，就用獨立收集的資料來測試。

僅線上版：*ImageNet* 的數字有多可靠？— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/imagenet-的數字有多可靠/*

把基準分數當成關於「這個基準」的陳述

「81.0% top-1」的意思是：在這 50,000 張特定的驗證影像上、用這套前處理、這組標籤所得到的結果。請回報解析度、裁切數量、是否用了額外資料或蒸餾，並在差異小於一個百分點時回報不同隨機種子間的變異。

評估流程

基準分數有多誠實，取決於背後的評估流程。

- **三種切分，三種任務**。在**訓練集**上擬合參數。在**驗證集**上（或在訓練資料內做交叉驗證）選擇超參數、提前停止點、溫度、門檻與提示。**測試集**只在最後碰一次。若基準只公開驗證集（如 ImageNet-1k），就從訓練集中切出一部分用來調整。
- **絕不在測試集上調整**。每一個看過測試結果之後才做的決定（多加一種擴增、多訓練一個 epoch、換一個檢查點），都會讓模型稍微擬合測試集。整個領域的論文累積起來，就是**測試集重複使用**（test-set reuse），也就是對基準的適應性過擬合。ImageNetV2 [7] 發現在新樣本上排名大致保持，這令人安心；但同一份研究也顯示絕對數字無法直接轉移，而新的測試集是唯一直接的檢驗方法。
- **回報變異**。用多個隨機種子訓練（至少三個，小資料集要更多），回報平均 ± 標準差，或信賴區間。在 10,000 張測試影像上，ImageNetV2 [7] 給出的 95% 信賴區間約為 ±1%；在這樣的測試集上進步 0.3 個百分點，算不上結果。
- **回報完整配方**。解析度、裁切數、額外資料、蒸餾、指標的哪一種變體（top-1 或各類別平均準確率、哪一種 AP 定義），以及標籤集的哪一個版本（原始或 ReaL）。

CNN 架構與其背後的理由

白話版。每一個著名的 CNN 都修正了前一代的某個具體問題：訓練太慢、參數太多、太深就難以最佳化、在手機上太昂貴。知道每個架構解決了什麼問題，比背它們的層表更有用。

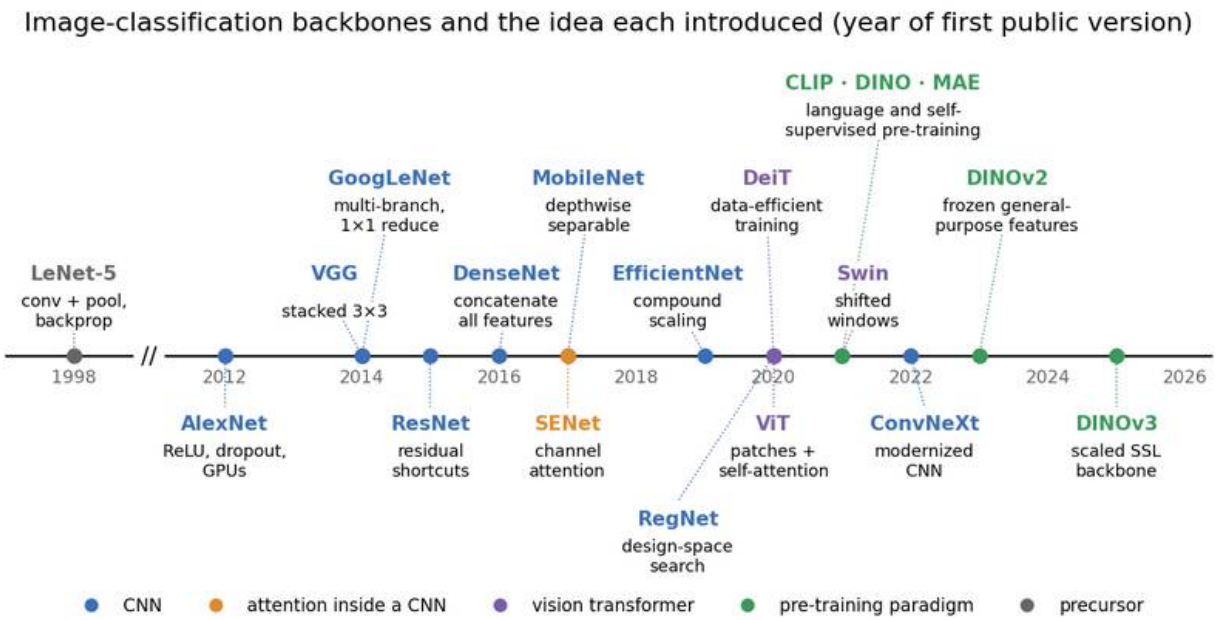


圖 3 — 影像分類骨幹網路與各自引入的主要想法。年份為首次公開版本（通常是 *arXiv*）的年份。顏色區分純 *CNN*、*CNN* 內的注意力機制、視覺 *Transformer*，以及預訓練典範。

AlexNet：深度變得可訓練

2012 年以前，最好的 ImageNet 系統使用手工特徵（SIFT、Fisher 向量）加線性分類器。AlexNet [10] 是一個五層卷積、三層全連接的網路，在 ImageNet 上端到端訓練。讓它能在這個規模上成功的有三個要素。**ReLU** 激活函數 $\max(0, z)$ 對正輸入不會飽和，梯度不會像 tanh 那樣消失，論文回報訓練速度快了好幾倍。**Dropout** 用在大型全連接層上對抗過擬合，那些層占了大部分參數。**GPU** 讓這一切可行：模型被拆在兩張 GPU 上。大量的資料擴增（隨機裁切與翻轉、顏色擾動）是第四個、常被忘記的要素。設計上的教訓是：只要資料和算力足夠，學出來的特徵勝過設計出來的特徵。

VGG：小卷積核，疊起來

VGG [11] 問了一個更乾淨的問題：如果只用 3×3 卷積，然後一直加深（16–19 個有權重的層），會怎樣？兩層 3×3 疊起來看到的是 5×5 區域，三層是 7×7。對 C 個輸入與輸出通道而言，

$$\underbrace{2 \cdot 3^2 C^2}_{\text{two } 3\times 3} = 18C^2 \;<\; \underbrace{5^2 C^2}_{\text{one } 5\times 5} = 25C^2,$$

（忽略偏差項）。所以同樣的感受野，疊層的參數更少，而且兩層之間多了一個非線性。VGG 的一致設計讓它多年來都是熱門的特徵擷取器。它的弱點是成本：全連接頭與寬大的前幾層讓它很笨重。

GoogLeNet ／ Inception：寬度、分支與 1×1 瓶頸

GoogLeNet [12] 則改攻成本。每個 **Inception 模組**平行執行 1×1、3×3、5×5 卷積與池化，再把輸出串接起來，讓下一層自己挑選適合的尺度。在很多通道上跑大卷積核很昂貴，所以先用 1×1 **卷積**減少通道數。1×1 卷積就是在每個像素上做一次跨通道的線性映射：它混合通道而不看鄰居，是改變寬度最便宜的方法。GoogLeNet 也用**全域平均池化**取代龐大的全連接頭。後續的 Inception-v3 論文 [13] 把大卷積核分解成小卷積的堆疊，或 $1 \times n$ 與 $n \times 1$ 的組合，並提出了**標籤平滑**（label smoothing），我們會在訓練一節再見到它。

ResNet：學習殘差

問題。一直疊加普通的層，最後網路反而變差，連訓練集上都一樣。這種「退化」是最佳化問題，不是過擬合：更深的網路原則上可以複製較淺的網路，再把多出來的層設成恆等映射，但 SGD 就是找不到這個解。

想法。 He 等人 [14] 讓一個區塊學習殘差 $\mathcal{F}$，再把輸入加回去：

$$\mathbf{y} = \mathcal{F}(\mathbf{x}; \{W_i\}) + \mathbf{x},$$

其中 $\mathbf{x}$ 是區塊輸入，$\mathcal{F}$ 是權重為 W_i 的兩到三層卷積，$\mathbf{y}$ 是輸出。如果最佳映射接近恆等，區塊只需要把 $\mathcal{F}$ 推向零，這很容易。當形狀改變（通道變多、步幅為 2）時，捷徑改成 1×1 投影。

為什麼好訓練。因為每個區塊都把輸入加回去，深層堆疊的輸出等於任何一個較早區塊的輸入，再加上一串殘差的總和。損失的梯度因此能沿著捷徑直接抵達每一個區塊，不會因為許多小因子

連乘而消失。後續一篇論文 [15] 也發現，讓捷徑保持純粹的恆等（把 BN 與 ReLU 移進殘差分支，即「預激活」）能讓更深的網路更容易訓練。

僅線上版：殘差網路為什麼好訓練：展開後的梯度 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/殘差網路為什麼好訓練展開後的梯度/*

選讀：批次正規化（*DLCV 01* 的補充資料）。本頁處處用到的 *BN* 層在計算什麼，以及訓練與推論為何使用不同的統計量。

瓶頸結構。對更深的 ResNet（50 層以上），每個區塊變成 1×1（降維）$\to 3\times3 \to 1\times1$（升維），如圖 4(c)。在 256 通道時，兩層 3×3 的基本區塊有 $2\cdot9\cdot256^2 \approx 118$ 萬個權重；降到 64 通道的瓶頸區塊則有 $256\cdot64 + 9\cdot64^2 + 64\cdot256 \approx 7$ 萬個。昂貴的 3×3 跑在很薄的張量上，這正是 50–152 層負擔得起的原因。

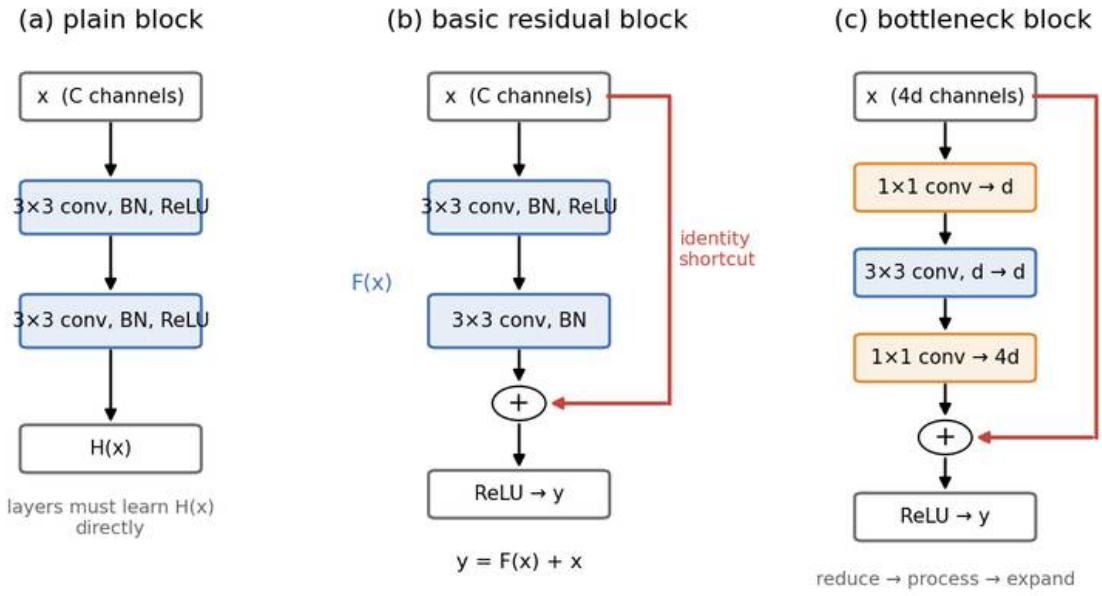


圖 4 — (*a*) 普通區塊必須學習完整的映射 $H(x)$。(*b*) 殘差區塊學習 $F(x) = H(x) - x$，再透過恆等捷徑把輸入加回去。(*c*) 瓶頸區塊在縮減後的寬度 d 上做 *3×3* 卷積，再升回 *4d*。

本篇實驗所用的基本區塊，以 PyTorch 寫成：

```
import torch.nn as nn
import torch.nn.functional as F

class BasicBlock(nn.Module):
    def __init__(self, cin, cout, stride=1):
        super().__init__()
        self.conv1 = nn.Conv2d(cin, cout, 3, stride, 1, bias=False)
        self.bn1 = nn.BatchNorm2d(cout)
        self.conv2 = nn.Conv2d(cout, cout, 3, 1, 1, bias=False)
        self.bn2 = nn.BatchNorm2d(cout)
        self.short = nn.Identity()
        if stride != 1 or cin != cout:   # projection shortcut when shapes change
```

```
            self.short = nn.Sequential(nn.Conv2d(cin, cout, 1, stride, bias=False),
            ↪  nn.BatchNorm2d(cout))

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        return F.relu(out + self.short(x))   # y = F(x) + x
```

之後有兩種 CNN 設計進一步發展了這些想法。**DenseNet** [16] 讓區塊內每一層接收所有前面各層輸出的串接，以很少的參數再利用特徵，代價是記憶體。**RegNet** [19] 研究的是一整群類 ResNet 網路，發現好的網路在「各階段寬度如何遞增」上遵循簡單的規則。

僅線上版：另外兩種 *CNN* 設計：*DenseNet* 與 *RegNet* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/另外兩種-cnn-設計 densenet-與-regnet/*

MobileNet：把卷積拆開

MobileNet [17] 的目標是手機。在 $H \times W$ 的特徵圖上，從 M 個通道到 N 個通道的標準 $K \times K$ 卷積需要 HWK^2MN 次乘加。**深度可分離**（depthwise separable）卷積把它拆成兩步：**逐通道**（depthwise）$K \times K$ 卷積，每個通道各自濾波（HWK^2M）；以及**逐點**（pointwise）1×1 卷積，負責混合通道（$HWMN$）。兩者的成本比為

$$\frac{HWK^2M + HWMN}{HWK^2MN} = \frac{1}{N} + \frac{1}{K^2},$$

所以當 $K = 3$、輸出通道很多時，可分離版本大約便宜 8–9 倍。它押注的是：空間濾波與通道混合不需要一起學。這個賭注押得很成功，深度卷積後來在 EfficientNet 與 ConvNeXt 中再次出現。

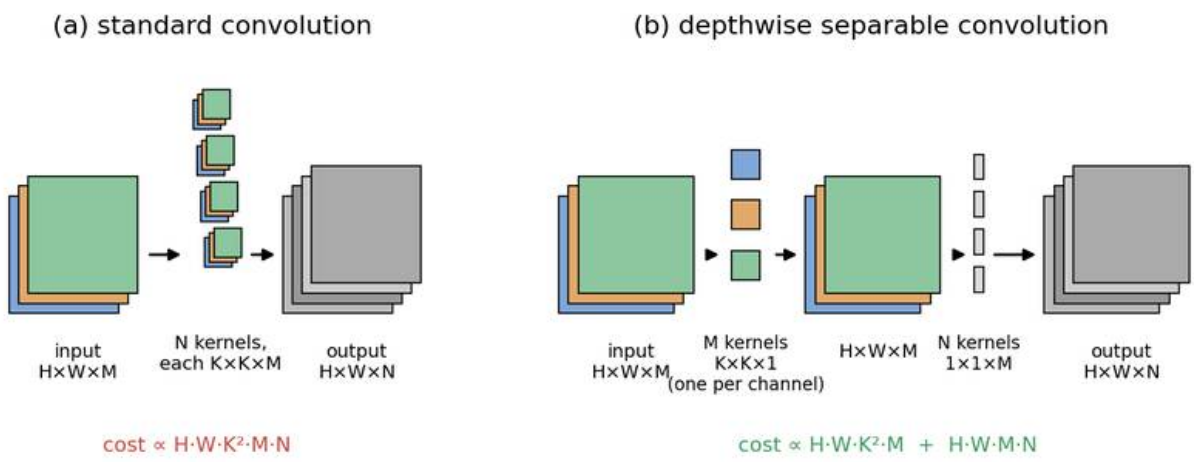


圖 5 — 標準卷積在同一步裡完成濾波與通道混合。深度可分離卷積先對每個通道各自濾波（*depthwise*），再用 *1×1* 卷積核混合通道（*pointwise*）。

```
import torch.nn as nn
```

```
def n_params(m):
    return sum(p.numel() for p in m.parameters())

M, N, K = 128, 128, 3
standard = nn.Conv2d(M, N, K, padding=1, bias=False)
separable = nn.Sequential(nn.Conv2d(M, M, K, padding=1, groups=M, bias=False),   # depthwise
                          nn.Conv2d(M, N, 1, bias=False))                         #
                           ↪  pointwise
print("standard:", n_params(standard), " separable:", n_params(separable),
      f" ratio {n_params(separable) / n_params(standard):.3f} vs 1/N + 1/K^2 = {1/N +
       ↪  1/K**2:.3f}")
# standard: 147456  separable: 17536  ratio 0.119 vs 1/N + 1/K^2 = 0.119
```

EfficientNet：深度、寬度、解析度一起放大

手上有一個好的小網路，要怎麼把它變大？加深、加寬，還是提高輸入解析度？Tan 與 Le [18] 主張這三者互相影響：解析度更高的影像需要更深（更大的感受野）與更寬（更細的圖樣）。**複合縮放**（compound scaling）用一個係數 ϕ 同時放大三者：

$$d = \alpha^\phi, \quad w = \beta^\phi, \quad r = \gamma^\phi, \qquad \text{s.t. } \alpha\beta^2\gamma^2 \approx 2, \;\; \alpha, \beta, \gamma \geq 1,$$

其中 d、w、r 分別是深度、寬度與解析度的倍率。FLOPs 大約與 $d\,w^2 r^2$ 成正比，所以 ϕ 每加 1，成本大約加倍。常數是在基礎網路上做小型網格搜尋得到的（基礎網路本身由架構搜尋找出，使用帶深度卷積與 squeeze-and-excitation 的反向瓶頸區塊）。「平衡地放大」這個想法，比那些特定常數活得更久。

ConvNeXt：一步一步現代化的 CNN

在視覺 Transformer 拿下排行榜之後，Liu 等人 [20] 問：它們的優勢有多少來自注意力機制，又有多少來自其他一切？他們從一個用 Transformer 風格配方訓練的 ResNet-50 出發，一次只改一件事：仿照 Swin 的階段比例、「patchify」主幹（4×4、步幅 4 的卷積）、用更大的 7×7 深度卷積、反向瓶頸、更少的激活與正規化層、GELU 與 LayerNorm。結果 ConvNeXt 在 ImageNet 與下游任務上，追平了相近成本的 Swin 類 Transformer。教訓是：架構家族的差異，沒有伴隨而來的許多小設計與訓練選擇那麼重要。

注意力機制與 Transformer

白話版。卷積只用固定的權重看一個小鄰域。注意力機制讓每個位置根據內容，自己決定要聽哪些其他位置的話。視覺 Transformer 把這個機制套用到影像小塊上。

注意力最早以輕量的形式進入 CNN。**Squeeze-and-excitation（SE）區塊** [21] 用全域平均把特徵圖的每個通道濃縮成一個數字，再由一個小網路把這些摘要轉成介於 0 與 1 之間的權重，重新縮放各通道。網路因此能在整張影像看起來像動物時，加強「毛皮」相關的通道。SE 區塊是 EfficientNet 與許多行動裝置網路的一部分。

僅線上版：*Squeeze-and-excitation*：*CNN* 裡的注意力 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/squeeze-and-excitationcnn-裡的注意力/*

ViT：把影像當成一串小塊

視覺 Transformer（Vision Transformer, ViT）[22] 幾乎完全拿掉卷積。影像 $\mathbf{x} \in \mathbb{R}^{H\times W\times C}$ 被切成 $N = HW/P^2$ 個不重疊的 $P \times P$ 小塊（patch）。每個小塊攤平後線性投影成 D 維的 **token**。前面再加上一個可學習的**類別 token**，並加上可學習的**位置嵌入**：

$$\mathbf{z}_0 = [\,\mathbf{x}_{\text{class}};\ \mathbf{x}_p^1\mathbf{E};\ \ldots;\ \mathbf{x}_p^N\mathbf{E}\,] + \mathbf{E}_{\text{pos}},$$

其中 $\mathbf{x}_p^i \in \mathbb{R}^{P^2C}$ 是第 i 個攤平的小塊，$\mathbf{E} \in \mathbb{R}^{P^2C\times D}$ 是小塊嵌入矩陣，$\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1)\times D}$ 是位置嵌入。接著由 L 層標準 Transformer 編碼器（多頭自注意力與 MLP，各自帶 LayerNorm 與殘差連接）處理這個序列，最後由一個小型分類頭讀取類別 token 的最終狀態，輸出 logits。單一個注意力頭的自注意力為

$$\text{Attn}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\Big(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\Big)\mathbf{V},$$

其中 $\mathbf{Q} = \mathbf{Z}\mathbf{W}_Q$、$\mathbf{K} = \mathbf{Z}\mathbf{W}_K$、$\mathbf{V} = \mathbf{Z}\mathbf{W}_V$ 是 token 矩陣 $\mathbf{Z}$ 的線性投影，d 是 key 的維度。注意力矩陣大小為 $(N+1)\times(N+1)$：從第一層開始，每個小塊就能注意到所有其他小塊，所以感受野一開始就是全域的。

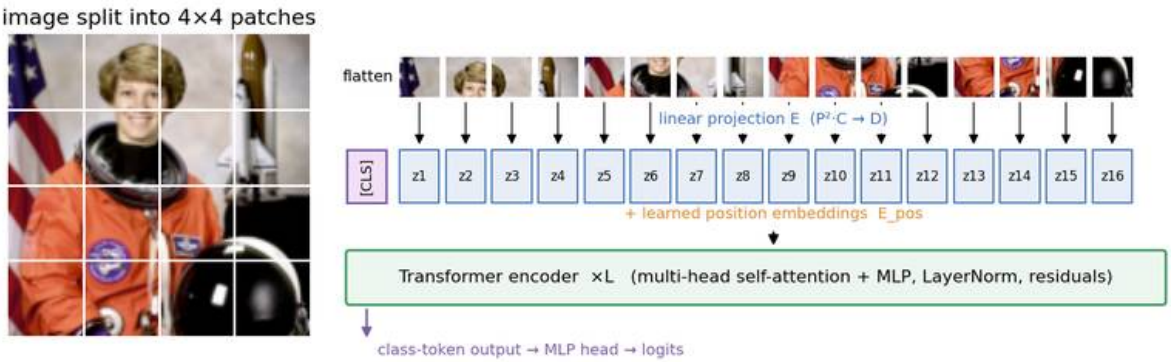


圖 6 — *ViT* 的輸入流程。所有小塊共用同一個線性投影變成 *token*（等價於核大小 $P\times P$、步幅 P 的卷積）。加上類別 *token* 與可學習的位置嵌入後，用類別 *token* 的最終狀態來分類。

```
import torch
import torch.nn as nn

B, C, H, W, P, D = 2, 3, 32, 32, 8, 64
img = torch.randn(B, C, H, W)

# 1) patchify: (B, C, H, W) -> (B, N, P*P*C) with N = (H/P)*(W/P)
patches = img.unfold(2, P, P).unfold(3, P, P)                 # (B, C, H/P, W/P, P, P)
patches = patches.permute(0, 2, 3, 1, 4, 5).reshape(B, -1, C * P * P)
N = patches.shape[1]
```

```
# 2) linear patch embedding + class token + learned position embedding
embed = nn.Linear(C * P * P, D)          # same as nn.Conv2d(C, D, kernel_size=P, stride=P)
cls = nn.Parameter(torch.zeros(1, 1, D))
pos = nn.Parameter(torch.randn(1, N + 1, D) * 0.02)
z = torch.cat([cls.expand(B, -1, -1), embed(patches)], dim=1) + pos   # (B, N+1, D)

# 3) one head of scaled dot-product self-attention
Wq, Wk, Wv = (nn.Linear(D, D, bias=False) for _ in range(3))
q, k, v = Wq(z), Wk(z), Wv(z)
A = (q @ k.transpose(1, 2) / D ** 0.5).softmax(dim=-1)        # (B, N+1, N+1), rows sum to 1
out = A @ v
print("tokens:", N + 1, " attention:", tuple(A.shape), " class-token output:", tuple(out[:,
 ↪  0].shape))
```

為什麼 ViT 需要大量資料。 CNN 內建了一些假設，稱為**歸納偏差**（inductive bias）：局部性（一個像素主要和鄰居有關）與平移等變性（同一個濾波器用在每個位置）。ViT 幾乎沒有這些；除了切小塊與位置嵌入之外，空間結構都得從資料中學。ViT 論文 [22] 回報：只在 ImageNet-1k 上訓練時，ViT 落後於規模相近的 ResNet；但在大得多的資料集上預訓練（ImageNet-21k，或 3 億張影像的 JFT）之後，就能追平甚至超越。弱先驗在小規模時要付出資料的代價，到了大規模就不再是限制。同一篇論文也嘗試了**混合**（hybrid）模型：把 ResNet 主幹的特徵圖、而不是原始小塊，送進 Transformer；混合模型在計算預算小時較有幫助，模型變大後差距就消失了。

DeiT：只用 ImageNet 訓練 ViT

DeiT [23] 證明「ViT 需要巨量資料」這個結論，有一部分其實是關於訓練配方。用強力的資料擴增（RandAugment、Mixup、CutMix、random erasing）、隨機深度、重複擴增與 AdamW，只在 ImageNet-1k 上訓練的 ViT-B 就達到 83.1% top-1。DeiT 還加入了**蒸餾 token**：第二個可學習的 token，和類別 token 一樣，但它的輸出被訓練去匹配教師網路的預測（他們發現卷積網路當教師效果最好），而類別 token 則以真實標籤訓練。這是知識蒸餾 [51] 的一種 Transformer 專屬形式。測試時把兩個頭的輸出結合起來。

選讀：知識蒸餾（*DLCV 01* 的補充資料）。蒸餾損失與溫度的作用。

Swin：視窗、位移與階層

全域注意力的成本隨 token 數量呈平方成長，對高解析度輸入與密集預測任務是個問題。Swin [24] 只在不重疊的 $M \times M$ 視窗內計算注意力，並在相鄰層之間把視窗網格位移半個視窗，讓資訊能跨越視窗邊界。各階段之間合併 2×2 相鄰小塊來減少 token 數，形成步幅為 4、8、16、32 的類 CNN 金字塔。對 $h \times w$ 個 token、C 個通道，論文給出

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C, \qquad \Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC,$$

所以在視窗大小 M 固定時，視窗注意力對 token 數 hw 是線性的。Swin 帶回了兩個 CNN 先驗：局部性與階層性，成為偵測與分割也通用的骨幹。視覺 Transformer 的整體版圖可參考 Khan 等人的綜述 [25]。

訓練配方

白話版。同一個網路，訓練方法不同，準確率可以差好幾個百分點。現代訓練配方是一組小技巧，每一個都讓「死背」更難，或讓最佳化更平順。

資料擴增

- **隨機縮放裁切**（random resized crop）隨機取一塊占原圖面積一定比例、長寬比隨機的區域，再縮放到訓練解析度。它教會網路尺度與位置不變性，是 ImageNet 訓練的預設做法。**水平翻轉**只在不改變標籤時使用（大多數照片可以，文字或數字不行）。
- **Mixup** [26] 混合兩張影像與它們的標籤：$\tilde{\mathbf{x}} = \lambda \mathbf{x}_i + (1-\lambda)\mathbf{x}_j$、$\tilde{\mathbf{y}} = \lambda \mathbf{y}_i + (1-\lambda)\mathbf{y}_j$，其中 $\lambda \sim \mathrm{Beta}(\alpha, \alpha)$，$\mathbf{y}$ 為 one-hot。它鼓勵模型在訓練樣本之間呈線性行為。
- **CutMix** [27] 把影像 j 的一個隨機矩形貼到影像 i 上，並依貼上的面積比例混合標籤。和 Mixup 不同，每個像素都仍是自然的，網路必須從局部視野辨認物體。
- **RandAugment** [28] 用兩個數字取代學出來的擴增策略：隨機套用 N 個操作（旋轉、剪切、色階分離、對比……），共用同一個強度 M。論文回報這個極小的搜尋空間就能追平或超越先前的自動化策略，而合適的強度取決於模型與資料集大小。

```
import numpy as np
import torch
import torch.nn.functional as F

def mixup(x, y, num_classes, alpha=0.2):
    lam = float(np.random.beta(alpha, alpha))
    idx = torch.randperm(x.size(0))
    y1 = F.one_hot(y, num_classes).float()
    return lam * x + (1 - lam) * x[idx], lam * y1 + (1 - lam) * y1[idx]

def cutmix(x, y, num_classes, alpha=1.0):
    lam = float(np.random.beta(alpha, alpha))
    idx = torch.randperm(x.size(0))
    H, W = x.shape[-2:]
    h, w = int(H * np.sqrt(1 - lam)), int(W * np.sqrt(1 - lam))
    cy, cx = np.random.randint(H), np.random.randint(W)
    y0, y1_, x0, x1_ = max(cy - h // 2, 0), min(cy + h // 2, H), max(cx - w // 2, 0), min(cx
     ↪  + w // 2, W)
    x = x.clone()
    x[..., y0:y1_, x0:x1_] = x[idx][..., y0:y1_, x0:x1_]
    lam = 1 - (y1_ - y0) * (x1_ - x0) / (H * W)            # actual area kept
    y1 = F.one_hot(y, num_classes).float()
    return x, lam * y1 + (1 - lam) * y1[idx]

def soft_ce(logits, soft_targets):
    return -(soft_targets * logits.log_softmax(1)).sum(1).mean()
```

標籤平滑

以 one-hot 為目標的交叉熵會永遠把正確 logit 往上推；只有在間距無限大時，損失才會到零。**標籤平滑** [13] 把目標換成

$$\mathbf{q} = (1-\varepsilon)\,\mathbf{e}_y + \frac{\varepsilon}{K}\mathbf{1},$$

其中 ε（常用 0.1）是平滑強度，$\mathbf{e}_y$ 是 one-hot 向量，$\mathbf{1}$ 是全 1 向量。此時最佳解是有限的 logit 間距，對網路有正則化效果，通常也會減少過度自信。在 PyTorch 裡只是一個參數：`F.cross_entropy(logits, y, label_smoothing=0.1)`。

最佳化器、排程與正則化

- **AdamW** [29] 把權重衰減與自適應梯度步驟解耦：不是把 $\lambda\theta$ 加到梯度上（Adam 會再對每個參數重新縮放），而是直接縮小權重，$\theta \leftarrow \theta - \eta(\hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{v}}}+\epsilon) + \lambda\theta)$，其中 η 是學習率，$\hat{\mathbf{m}}, \hat{\mathbf{v}}$ 是 Adam 的動差估計，λ 是衰減係數。它是 Transformer 與現代 CNN 配方的預設選擇。偏差與正規化層的參數通常不做衰減。
- **暖身＋餘弦排程**。訓練最初幾步，正規化統計量與 Adam 的動差估計都還不準，大學習率會不穩定，所以先讓學習率在幾個 epoch 內線性上升，接著沿半個餘弦曲線降到接近零，$\eta_t = \frac{1}{2}\eta_{\max}(1+\cos(\pi t/T))$，$t$ 是暖身後的步數，T 是剩餘步數；這個想法由 SGDR [30] 推廣開來。
- **隨機深度**（stochastic depth）[31] 在訓練時隨機跳過整個殘差分支（區塊變成恆等映射），跳過機率隨深度增加。它縮短了訓練時的有效深度，效果類似一群較淺網路的集成。
- **權重的 EMA** 維護一個指數移動平均 $\bar{\theta} \leftarrow \mu\bar{\theta} + (1-\mu)\theta$（$\mu$ 接近 1，例如 0.999），評估時使用 $\bar{\theta}$。它把最後幾步 SGD 的雜訊平滑掉。

```
import math
import torch

def warmup_cosine(step, total, warmup):
    if step < warmup:
        return (step + 1) / warmup                      # linear warmup
    p = (step - warmup) / max(1, total - warmup)
    return 0.5 * (1 + math.cos(math.pi * p))            # cosine decay to 0

model = torch.nn.Linear(10, 3)
decay = [p for n, p in model.named_parameters() if p.ndim > 1]      # weights
no_decay = [p for n, p in model.named_parameters() if p.ndim <= 1]  # biases, norm scales
opt = torch.optim.AdamW([{"params": decay, "weight_decay": 0.05},
                         {"params": no_decay, "weight_decay": 0.0}], lr=1e-3)
total = 1000
sched = torch.optim.lr_scheduler.LambdaLR(opt, lambda s: warmup_cosine(s, total, warmup=50))
ema = torch.optim.swa_utils.AveragedModel(model,
 ↪  multi_avg_fn=torch.optim.swa_utils.get_ema_multi_avg_fn(0.999))
for step in range(total):
    loss = model(torch.randn(16, 10)).pow(2).mean()
    opt.zero_grad(); loss.backward(); opt.step(); sched.step(); ema.update_parameters(model)
```

一個小實驗

為了看這些元件一起運作，我們在 scikit-learn 的 8 × 8 手寫數字上，訓練了一個 7.8 萬參數的 ResNet（「TinyResNet」：一個主幹、三個通道數分別為 16、32、64 的基本區塊、全域平均池化）。影像先放大到 32 × 32，讓有步幅的階段有空間運作。為了讓過擬合看得見，我們只用 **100 張訓練影像**，另外 500 張用於校準、1,197 張用於測試；使用 AdamW（權重衰減 0.05）、批次大小 32、40 個 epoch，先暖身兩個 epoch 再做餘弦衰減。一組完全不做擴增；另一組使用隨機旋轉（±15°）、縮放與平移（類似隨機縮放裁切的仿射變換；不翻轉，因為翻轉後的數字是另一個符號），並搭配標籤平滑 0.1。所有程式碼都在 `scripts/figures/dl_02_image_classification.py`，在 CPU 上大約一分鐘跑完。

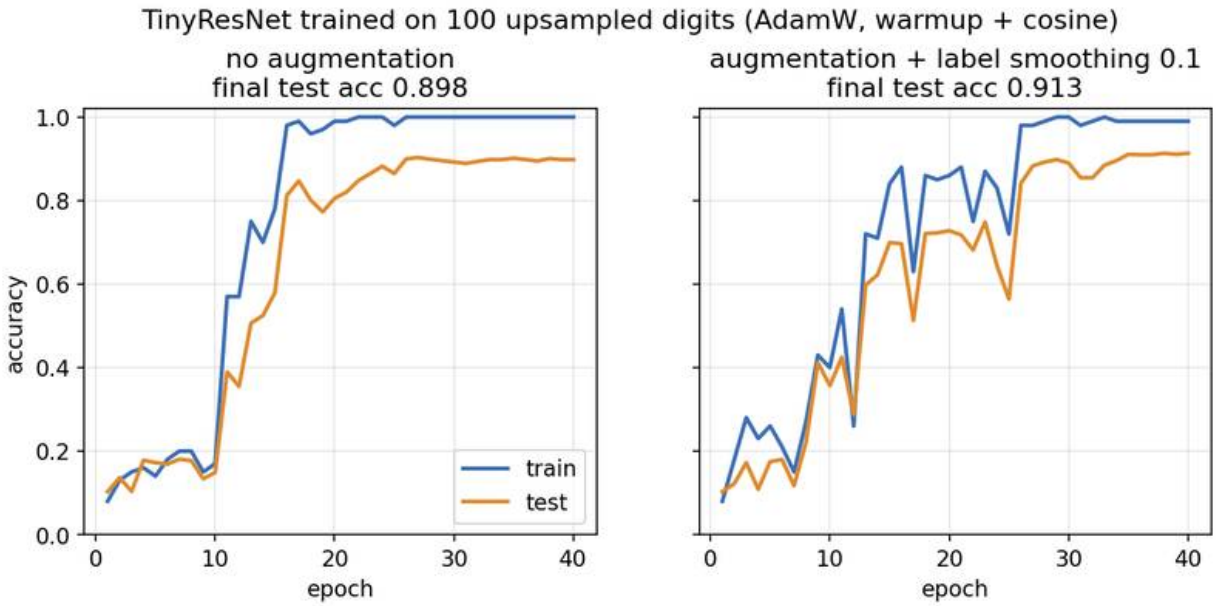


圖 7 — 以 *100* 張數字影像訓練的學習曲線。沒有擴增（左）時，網路背下了整個訓練集，測試準確率 *89.8%*。加上擴增與標籤平滑（右）後達到 *91.3%*。在這麼簡單的任務上差距不大；在自然影像上，配方的影響大得多。

沒有擴增的模型大約二十個 epoch 後就達到 100% 訓練準確率：只有 100 張影像，背起來很容易。有擴增的模型每個 epoch 看到的都是稍微不同的數字，學得比較慢，泛化稍好（91.3% 對 89.8%）。單一隨機種子上 1.5 個百分點不必過度解讀；重點是機制。

「ResNet strikes back」

配方重要性最驚人的證據，來自 Wightman、Touvron 與 Jégou [32]。他們保持 2015 年的 ResNet-50 架構不變，用現代配方重新訓練：隨機縮放裁切與翻轉、RandAugment、Mixup 與 CutMix、LAMB 最佳化器配餘弦排程、二元交叉熵損失，以及更長的訓練。原汁原味的 ResNet-50 在 224×224 的 ImageNet 上，不用額外資料或蒸餾就達到 **80.4% top-1**，而他們比較的標準 torchvision 權重是 76.1%。這帶來兩個結論。第一，許多「新架構打敗 ResNet-50」的宣稱，其實是拿用新配方

訓練的新模型，去比用舊配方訓練的舊模型。第二，比較架構時，要用相同且調好的配方訓練它們，就像 ConvNeXt [20] 與 RegNet [19] 的做法。

遷移學習與微調

白話版。從頭訓練一個骨幹網路需要大量資料。比較好的做法是：從一個已經在大型資料集上訓練好的網路出發，再把它調整到你的任務上。你可以凍結它的特徵、只訓練新的最後一層，也可以溫和地重新訓練所有參數。

令 g_ϕ 為一個預訓練骨幹，把影像映射成特徵向量 $\mathbf{h} = g_\phi(\mathbf{x}) \in \mathbb{R}^D$，$\mathbf{W} \in \mathbb{R}^{K \times D}$ 為新的線性分類頭。

- **線性探測**（linear probe）。凍結 ϕ，只訓練 $\mathbf{W}$（等於在固定特徵上做邏輯迴歸）。它便宜、需要的標籤少，也是衡量表徵好壞的標準方法。
- **完整微調**（full fine-tuning）。同時更新 ϕ 與 $\mathbf{W}$，通常骨幹的學習率比分類頭小 10–100 倍，搭配短暫的暖身，有時還會用逐層學習率衰減（越前面的層學習率越小）。它讓特徵適應新的領域，標註資料夠多時通常勝出。
- 介於兩者之間的，有部分微調（只調最後幾個階段），以及在凍結骨幹上加入小型可訓練模組的參數高效方法。

Kornblith 等人 [33] 量測了 ImageNet 準確率能多好地預測遷移表現。對微調而言，ImageNet 上越好的模型通常遷移得越好，相關性很強。對固定特徵而言，相關性較弱，而且取決於正則化選擇：有些提升 ImageNet 準確率的設定（例如標籤平滑與 dropout），反而產生了在其他任務上更差的凍結特徵。好的分類器與好的特徵擷取器相關，但不是同一件事。

```
import torch
import torch.nn as nn

backbone = nn.Sequential(nn.Conv2d(3, 32, 3, 2, 1), nn.ReLU(), nn.Conv2d(32, 64, 3, 2, 1),
 ↪  nn.ReLU(),
                          nn.AdaptiveAvgPool2d(1), nn.Flatten())          # stand-in for a
                           ↪  pre-trained model
head = nn.Linear(64, 5)

# linear probe: freeze the backbone, train only the head
for p in backbone.parameters():
    p.requires_grad = False
probe_opt = torch.optim.AdamW(head.parameters(), lr=1e-3)

# full fine-tuning: unfreeze, use a smaller learning rate for the backbone than for the head
for p in backbone.parameters():
    p.requires_grad = True
ft_opt = torch.optim.AdamW([{"params": backbone.parameters(), "lr": 1e-5},
                            {"params": head.parameters(), "lr": 1e-3}], weight_decay=0.05)
```

自監督學習與基礎模型分類器

白話版。標籤很貴，影像很便宜。自監督學習自己發明一個答案就藏在影像裡的任務（這兩塊裁切是不是來自同一張照片？被遮住的地方原本是什麼？），透過解這個任務學到特徵，最後才用少量標籤。

對比式學習：SimCLR 與 MoCo

對比式方法把同一張影像的兩個擴增**視圖**的嵌入拉近，把不同影像的視圖推遠。在 SimCLR [34] 中，每個視圖都要在批次裡所有其他視圖中找出自己的搭檔，這是一個「錯誤答案」就是其他影像的分類任務；強力擴增（特別是裁切加上顏色扭曲）與大批次被證明不可或缺。MoCo [35] 則維護一個由緩慢更新的編碼器副本所產生的嵌入佇列，不需超大批次也能有大量負樣本。

僅線上版：對比式學習細說：*SimCLR* 與 *MoCo* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/notes/對比式學習細說 simclr-與-moco/*

DINO 與 DINOv2：自我蒸餾

DINO [36] 不再使用明確的負樣本。學生網路看幾個裁切（包括小的局部裁切），被訓練去匹配看全域裁切的教師網路的輸出分佈。教師是學生的 EMA；對教師輸出做中心化與銳化，可以防止崩塌成常數。用 ViT 時，DINO 特徵展現出驚人的性質：類別 token 的注意力圖不需任何分割標籤就能把物體分割出來，而在凍結特徵上用簡單的 k-NN 分類器效果就很好。DINOv2 [38] 以自動篩選的資料集與一個大型 ViT（再蒸餾成較小的模型）擴大規模，產生通用的**凍結特徵**，不需微調就能用於分類、分割與深度估計。DINOv3 [39] 繼續擴大規模，並提出「Gram anchoring」，避免密集的小塊特徵在長時間訓練中劣化。

MAE：遮罩影像建模

MAE [37] 借用了遮罩語言模型的想法。它遮掉大比例的小塊（論文中為 75%），只把看得見的小塊送進 ViT 編碼器，再讓一個輕量解碼器重建被遮掉的像素。因為編碼器跳過被遮掉的 token，預訓練很便宜。MAE 特徵的線性可分性不如對比式特徵（線性探測較弱），但**微調**效果非常好。這是一再出現的模式：不同的預訓練目標，適合不同的適應方式。

CLIP：用語言做零樣本分類

CLIP [40] 用數億組影像與說明文字，以對稱的對比損失同時訓練影像編碼器 g 與文字編碼器 h，讓配對的影像與文字有高的餘弦相似度。之後**不需要任何訓練影像**就能建立分類器：為每個類別寫一句提示，例如 `a photo of a dog.`，將它嵌入，然後預測

$$\hat{y} = \arg\max_{k} \frac{g(\mathbf{x})^{\top} h(t_k)}{\|g(\mathbf{x})\| \, \|h(t_k)\|},$$

其中 t_k 是類別 k 的提示。文字嵌入扮演線性分類器權重的角色，只是它們是從文字算出來的。論文回報零樣本 CLIP 的準確率追平了一個用 ImageNet 128 萬張標註影像訓練的 ResNet-50，而平

均多個提示模板（「a photo of a …」、「a blurry photo of a …」）會有幫助。零樣本準確率高度依賴提示的用字、類別名稱在視覺上是否有意義，以及網路資料對該領域的涵蓋程度。

穩健性與評估陷阱

白話版。一個在測試集上拿 90% 的分類器，碰到不同相機拍的照片、有人刻意加上肉眼看不見的雜訊，或它學到的「簡單線索」（背景、浮水印）不見時，可能會錯得一塌糊塗。而且就算答對了，它的信心也可能不對。

分佈偏移：ImageNet-C、-R 與 -A

Hendrycks 等人為 ImageNet 分類器建立了三個廣泛使用的偏移測試集。**ImageNet-C** [42] 套用 15 種演算法產生的損壞（雜訊、模糊、天氣、數位瑕疵），各有五個嚴重程度，並以相對於基準模型的錯誤率總結（mean corruption error）。**ImageNet-R** [43] 收集了 200 個 ImageNet 類別的各種表現形式（藝術、卡通、雕塑、素描、玩具），測試模型理解的是形狀與概念，還是照片的紋理。**ImageNet-A** [44] 收錄未經修改的自然照片，這些照片經過篩選、專門騙倒 ResNet-50；標準模型在上面表現非常差。這三個測試集揭露不同的弱點，改善其中一個不保證改善其他。大規模預訓練（如 CLIP 與 DINOv2）對這類自然偏移的穩健性提升，往往遠大於在相同資料下更換架構。

對抗樣本：FGSM

Goodfellow 等人 [45] 用網路在高維空間中的**線性**來解釋對抗樣本：對每一個輸入維度做一點點、與權重方向一致的改變，累加起來就是輸出的巨大變化。**快速梯度符號法**（fast gradient sign method, FGSM）一步就完成擾動：

$$\mathbf{x}_{\text{adv}} = \text{clip}\left(\mathbf{x} + \varepsilon \, \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(f_\theta(\mathbf{x}), y))\right),$$

其中 ε 限制每個像素的改變量（ℓ_∞ 範數），$\mathcal{L}$ 是損失函數，clip 把像素值限制在有效範圍內。

```
import torch.nn.functional as F

def fgsm(model, x, y, eps):
    x = x.clone().requires_grad_(True)
    F.cross_entropy(model(x), y).backward()
    return (x + eps * x.grad.sign()).clamp(-1, 1).detach()
```

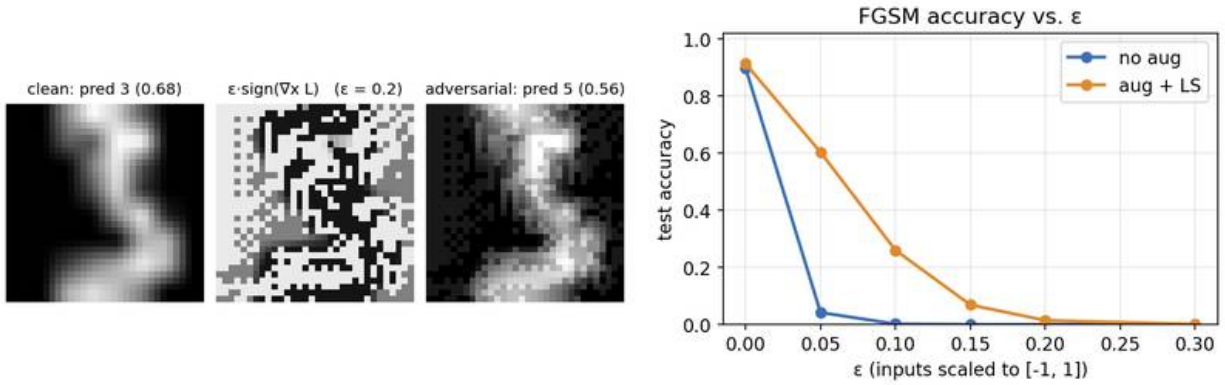


圖 *8* — 在 *TinyResNet* 模型上的 *FGSM*（輸入範圍為 $[-1, 1]$）。$\varepsilon = 0.05$ 時，未擴增模型的測試準確率從 *89.8%* 掉到 *4.2%*，擴增＋標籤平滑模型從 *91.3%* 掉到 *60.3%*；$\varepsilon = 0.2$ 時兩者都接近零。

關於圖 8 有兩點要注意。第一，擴增模型在小 ε 時看起來比較穩健，但已知標籤平滑會讓訓練點附近的損失曲面變平，可能削弱單步梯度攻擊，卻沒有帶來真正的穩健性；正確的檢驗是更強的迭代攻擊。第二，在 $[-1,1]$ 範圍內 $\varepsilon = 0.05$ 對 8×8 的數字來說是看得見的；在自然影像上，肉眼無法察覺的擾動就足以騙過標準的 ImageNet 模型。

捷徑學習

Geirhos 等人 [46] 把許多失敗案例歸在同一個名字下：**捷徑學習**（shortcut learning）。網路會用最容易的線索來最小化訓練損失，而那常常不是我們想要的線索：看背景而不是動物、看醫院的標記而不是病灶、看紋理而不是形狀。捷徑在獨立同分佈的測試集上看起來沒問題，因為測試集裡也有同樣的捷徑；一遇到分佈偏移就會失敗。對策是：刻意在分佈外測試、設計能打破可疑捷徑的測試，並用 Grad-CAM（見下文）這類可解釋性工具檢查模型在看哪裡。

校準與溫度縮放

溫度縮放（temperature scaling）[47] 在 softmax 之前把所有 logits 除以同一個純量 $T > 0$：$\mathbf{p} = \mathrm{softmax}(\mathbf{z}/T)$。$T$ 是在保留的**校準**集（絕不是測試集）上最小化負對數概似而得。$T > 1$ 讓過度自信的預測變柔和；$T < 1$ 讓信心不足的預測變銳利。因為除以 T 不改變 argmax，準確率不變。

```
import torch
import torch.nn.functional as F

def fit_temperature(logits, y, iters=300):
    """Learn one scalar T > 0 on a held-out calibration split."""
    log_t = torch.zeros(1, requires_grad=True)
    opt = torch.optim.Adam([log_t], lr=0.05)
    for _ in range(iters):
        loss = F.cross_entropy(logits / log_t.exp(), y)
        opt.zero_grad(); loss.backward(); opt.step()
    return log_t.exp().item()
```

```
def ece(probs, y, n_bins=10):
    conf, pred = probs.max(1)
    correct = (pred == y).float()
    edges = torch.linspace(0, 1, n_bins + 1)
    total = 0.0
    for lo, hi in zip(edges[:-1], edges[1:]):
        m = (conf > lo) & (conf <= hi)
        if m.any():
            total += m.float().mean() * (correct[m].mean() - conf[m].mean()).abs()
    return float(total)

# toy check: an over-confident "model" = true logits multiplied by 3
torch.manual_seed(0)
true_logits = torch.randn(4000, 10)
y = torch.distributions.Categorical(logits=true_logits).sample()
over = 3 * true_logits
T = fit_temperature(over[:2000], y[:2000])
print(f"T = {T:.2f}  ECE {ece(over[2000:].softmax(1), y[2000:]):.3f} -> "
      f"{ece((over[2000:] / T).softmax(1), y[2000:]):.3f}")
# T = 2.90  ECE 0.345 -> 0.018
```

在這個玩具例子裡，logits 被放大了三倍，擬合出的溫度（2.90）找回了這個倍數。而在我們真正的 TinyResNet（未擴增的那個模型）上，校準誤差的方向相反。

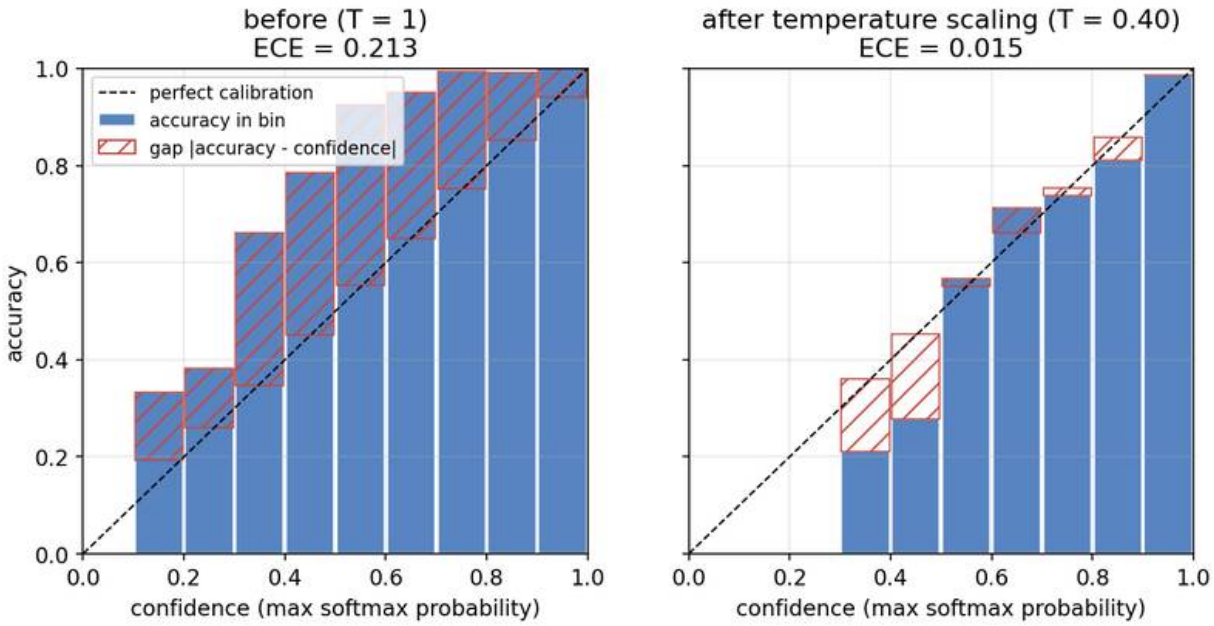


圖 *9* — 未擴增 *TinyResNet* 在 *1,197* 張測試數字上的可靠度圖。縮放前（左）多數區間的準確率高於信心：模型信心不足（*ECE 0.213*）。在另外 *500* 張校準影像上擬合出的溫度 $T = 0.40$，讓 *ECE* 降到 *0.015*，測試 *NLL* 從 *0.515* 降到 *0.298*，而且沒有改變任何一個預測。

為什麼信心不足？我們的小模型訓練時間短、權重衰減又強，使 logits 保持很小。Guo 等人 [47]

觀察到的是大型、長時間訓練的現代網路過度自信，並指出深度、寬度、批次正規化與較小的權重衰減都是成因。校準誤差沒有固定的方向；要量測它。另外也要注意，校準本身在分佈偏移下很脆弱：在分佈內資料上擬合的溫度，不保證在 ImageNet-C 式的偏移上仍然校準良好。

效率

白話版。模型必須放得進裝置、跑得進時間預算。參數量、計算量與實際速度是三種不同的量測，而使用者感受到的只有最後一種。

- **參數量**決定模型在磁碟與記憶體中的大小。一個 float32 參數占 4 個位元組。
- **FLOPs**（或乘加運算 MACs；一次 MAC 通常算作兩個 FLOPs）衡量每張影像的運算量。對核大小 $K \times K$、輸入 C_{in} 通道、輸出 C_{out} 通道、輸出特徵圖為 $H' \times W'$ 的卷積，MACs $= H'W'K^2 C_{\text{in}} C_{\text{out}}$。計算量隨解析度的平方成長。
- **延遲與吞吐量**要在目標硬體上量測。深度卷積的 FLOPs 很少，但運算密度低，所以在 GPU 上常比 FLOPs 所暗示的還慢；DenseNet 的串接則耗費記憶體頻寬。一定要實測。
- **量化**（quantization）把權重與激活值存成 8 位元整數，而不是 32 位元浮點數。Jacob 等人 [50] 描述了只用整數運算的推論，搭配量化感知訓練：在訓練時模擬捨入，讓網路學會容忍它。用少量校準資料做訓練後量化，則是較便宜的替代方案。
- **蒸餾** [51] 訓練一個小的學生網路去匹配大型教師網路的軟預測，就像 DeiT 用蒸餾 token 所做的那樣。

可解釋性：CAM 與 Grad-CAM

白話版。疊在影像上的熱圖，顯示哪些區域把某個類別的分數推高了。它是健全性檢查，不是推理過程的證明。

類別激活映射（class activation mapping, CAM）[48] 適用於以全域平均池化加一層線性層結尾的網路。若 A^k 是最後一個卷積層的第 k 個特徵圖，w_k^c 是它連到類別 c 的權重，則類別分數為 $\sum_k w_k^c \cdot \text{mean}(A^k)$，而 $\sum_k w_k^c A^k$ 這張圖顯示分數來自哪裡。**Grad-CAM** [49] 用空間平均的梯度取代這些權重，把方法推廣到任何可微分的架構：

$$\alpha_k^c = \frac{1}{Z} \sum_{i,j} \frac{\partial y^c}{\partial A_{ij}^k}, \qquad L_{\text{Grad-CAM}}^c = \text{ReLU}\Big(\sum_k \alpha_k^c A^k \Big),$$

其中 y^c 是類別 c 的 logit，A_{ij}^k 是通道 k 在位置 (i, j) 的激活值，Z 是位置總數。ReLU 只保留對該類別有正面影響的區域。粗糙的熱圖（我們的網路中是 8×8）再上採樣到影像大小。

```
import torch.nn.functional as F

def grad_cam(model, x, target=None):
    model.eval()
    feats = model.features(x)                          # (1, K, h, w) last conv stage
    feats.retain_grad()
    logits = model.fc(feats.mean(dim=(2, 3)))
    c = logits.argmax(1).item() if target is None else target
```

```
    logits[0, c].backward()
    alpha = feats.grad.mean(dim=(2, 3), keepdim=True)  # channel importance weights
    cam = F.relu((alpha * feats).sum(1, keepdim=True))
    cam = F.interpolate(cam, size=x.shape[-2:], mode="bilinear", align_corners=False)
    return (cam / (cam.max() + 1e-8))[0, 0].detach(), c
```

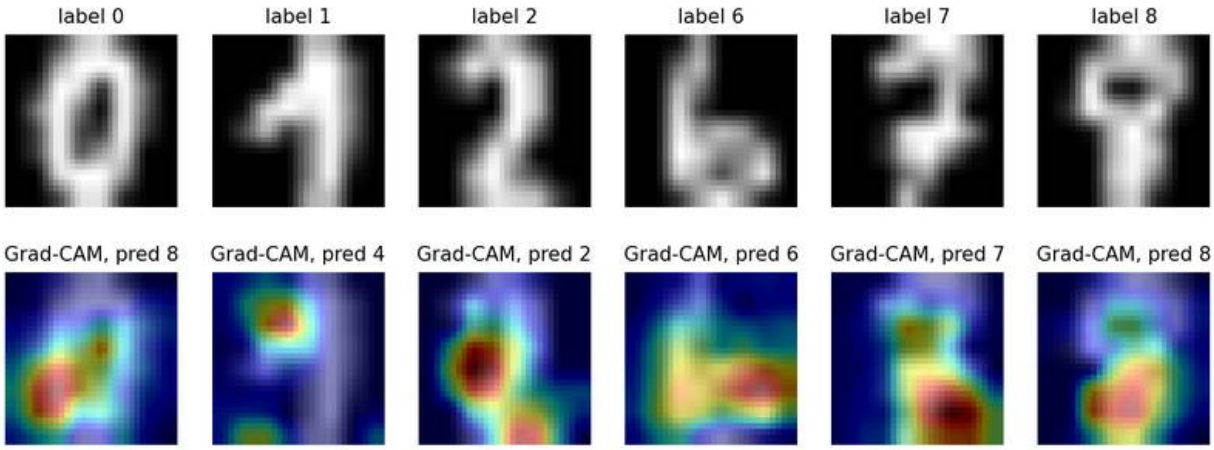


圖 *10* — 以擴增版 *TinyResNet*、對六個數字各自的第一個測試樣本，畫出預測類別的 *Grad-CAM*。其中兩個分錯了：*0* 被判成 *8*，熱區落在圓圈的左下部分；*1* 被判成 *4*，熱區落在它左上方的勾上。熱圖指出了誤導性證據被找到的位置。

解釋方法在模型出錯時最有用，就像圖 10：它們顯示了哪些筆畫看起來像錯誤的類別。但對顯著圖要謹慎。它們很粗糙（受限於最後一層特徵圖的解析度），而且只解釋單一輸入的單一類別分數，不是模型的一般推理方式。

現代觀點

值得一讀的綜述。 Li 等人 [1] 從基本構成元件到進階模型回顧 CNN，並以實驗比較激活函數、損失函數與最佳化器。Khan 等人 [25] 綜述視覺 Transformer，強調它們對大規模預訓練的依賴，以及平方複雜度注意力的成本。Gui 等人 [41] 把自監督學習分成以脈絡為基礎、對比式、生成式與混合式方法。在基準可靠性方面，ImageNetV2 [7]、重新標註 [8] 與標籤錯誤 [9] 三項研究共同主張：使用多個測試集、多標籤評估，並回報變異。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-classification/papers/現代觀點綜述導讀/*

2023–2026 年的樣貌。五個趨勢定義了當前的實務。

1. **預訓練一次，便宜地適應。**大多數新的分類器都不是從頭訓練。它們從一個以自監督（DINOv2 [38]、DINOv3 [39]）、影像–文字對比（CLIP [40] 及其開源重現）或遮罩建模（MAE [37]）訓練的大型骨幹出發，再以線性探測、*k*-NN 規則或短暫微調來適應。
2. **零樣本與開放詞彙分類。**影像–文字模型把標籤集合變成文字。類別清單可以在測試時更換、不需重新訓練，模糊了分類與檢索之間的界線。

3. **視覺的縮放定律**。 Zhai 等人 [52] 把 ViT 擴大到數十億參數，刻畫了錯誤率如何隨計算量、資料量與模型大小下降，並觀察到較大的模型在少樣本遷移中更有樣本效率。Cherti 等人 [53] 在公開資料上為開源 CLIP 訓練擬合出可重現的冪律，發現縮放行為取決於訓練資料分佈與下游任務。
4. **架構趨於收斂**。 ConvNeXt [20] 與 Swin [24] 顯示，在相同配方與規模下，卷積與注意力會落在相近的位置。現在的選擇更多取決於硬體、解析度與下游任務（密集預測偏好階層式設計），而不是 top-1 準確率。
5. **評估不再只看 ImageNet-1k top-1**。穩健性測試組（ImageNet-C/R/A [42]–[44] 與 ImageNetV2 [7]）、校準，以及遷移到大量下游資料集，現在都是骨幹論文的標準內容。

開放問題。對自然分佈偏移的穩健性仍落後於乾淨資料上的準確率，而穩健訓練方法帶來的進步往往只針對它所瞄準的那種偏移。沒有反事實測試，捷徑學習 [46] 很難被發現。分佈偏移下的校準仍未解決。長尾與細粒度辨識依然困難，因為稀有類別的樣本最少。基準飽和與標籤雜訊，讓 ImageNet-1k 上的微小準確率差異失去意義。而主流的基礎模型是用網路規模的資料訓練的，其組成、授權與偏差都很難稽核，這直接影響「零樣本」分類器能認出什麼、認不出什麼。

重點整理

- 互斥標籤使用 softmax，多標籤任務使用獨立 sigmoid 輸出。
- 在相同資料、訓練與計算條件下比較骨幹，架構名稱本身不足以解釋分數差異。
- 預訓練提供可重用特徵；線性探測評估特徵，微調則改變特徵。
- 檢查混淆矩陣與逐類別結果，並註明平均方式為 macro 或 micro。
- 用保留資料擬合校準，並同時評估信心與準確率。
- 分開測試分布轉移、捷徑與效率；報告多個隨機種子，只用驗證資料調整選擇。

練習

1. **感受野與參數量**。證明三層疊加的 3×3 卷積（步幅 1）感受野為 7×7。在輸入與輸出都是 C 通道、無偏差項的情況下，與單一 7×7 卷積比較參數量。

   提示

   每層 3×3 讓感受野增加 2 個像素：$3 \to 5 \to 7$。參數量：$3 \cdot 9C^2 = 27C^2$ 對 $49C^2$，少了約 45%，還多了兩個非線性。

2. **瓶頸區塊的算術**。一個 ResNet 瓶頸區塊輸入有 $4d$ 個通道，降到 d。以 d 表示 1×1–3×3–1×1 分支的權重數（無偏差），並與寬度 $4d$、兩層 3×3 的基本區塊比較。比值是多少？

   提示

   瓶頸：$4d \cdot d + 9d^2 + d \cdot 4d = 17d^2$。寬度 $4d$ 的基本區塊：$2 \cdot 9 \cdot 16d^2 = 288d^2$。比值為 $17/288 \approx 0.059$，權重少了約 17 倍。

3. **標籤平滑的最佳解**。對一個有 K 個類別、平滑強度 ε 的樣本，假設所有錯誤類別的 logit 相等。證明平滑目標下的交叉熵在「正確類別與每個錯誤類別的 logit 差」為 $\log\left((1-\varepsilon+\varepsilon/K)\,/\,(\varepsilon/K)\right)$ 時最小。代入 $K=1000$、$\varepsilon=0.1$ 計算。

   提示

   交叉熵對 $\mathbf{p}$ 的最小值發生在 $\mathbf{p}=\mathbf{q}$。所以 $p_y = 1-\varepsilon+\varepsilon/K$，其他每個 $p_k=\varepsilon/K$，logit 差就是兩者比值的對數：$\log(0.9001/0.0001)\approx 9.1$。沒有平滑時，最佳間距是無限大。

4. **ViT 的 token 數**。一個使用 16×16 小塊的 ViT 收到 224×224 的影像。有多少 token 進入編碼器？若以 384×384 微調，token 數是多少？注意力矩陣大小成長幾倍？位置嵌入又該怎麼處理？

   提示

   $(224/16)^2 = 196$ 個小塊加一個類別 token：197。在 384 時：$24^2+1=577$。注意力矩陣成長 $(577/197)^2\approx 8.6$ 倍。學到的位置嵌入是 14×14 的網格，要以二維內插放大到 24×24；ViT 論文在較高解析度微調時正是這麼做的。

5. **溫度不改變準確率**。證明對任何 $T>0$，$\arg\max_k \mathrm{softmax}(\mathbf{z}/T)_k = \arg\max_k z_k$。接著用 `scripts/figures/dl_02_image_classification.py`，改在測試集上而不是校準集上擬合溫度，比較回報的 ECE。為什麼那個數字過於樂觀？

   提示

   除以正的常數與指數函數都是嚴格遞增的，而 softmax 的分母是共用的。在測試集上擬合 T 等於用測試標籤調整參數，因此回報的 ECE 不再是對新資料表現的誠實估計。

6. **製造一條捷徑**。修改數字實驗，讓訓練集中每個「3」的左上角都有一個亮點，但測試集沒有。訓練 TinyResNet，回報測試集上 3 的準確率，並看幾個訓練用 3 的 Grad-CAM 熱圖。你預期會看到什麼？如果你事先不知道有這條捷徑，要怎麼發現這個問題？

   提示

   網路光看角落的亮點就能分類訓練集中的 3，所以測試集上 3 的準確率會下降、訓練準確率卻仍完美，而 Grad-CAM 往往會亮在角落。不知道捷徑時，可以找熱區落在背景上的熱圖、在獨立收集的資料上評估，並做反事實測試（抹除或移動可疑區域，看預測是否改變）。

7. **巨觀與微觀 F1**。一個三類別模型在 100 張影像上測試。列為真實類別、行為預測：類別 A（90 張）→ 88 A、1 B、1 C；類別 B（5 張）→ 3 A、2 B、0 C；類別 C（5 張）→ 4 A、0 B、1 C。計算逐類別的精確率、召回率與 F_1，再算巨觀 F_1 與微觀 F_1。如果報告的重點是能否偵測稀有的 B 與 C，你會放哪一個數字？為什麼？

   提示

   各行總和為 95、3、2。精確率：$88/95=0.926$、$2/3=0.667$、$1/2=0.500$。召回率：$88/90=0.978$、$2/5=0.400$、$1/5=0.200$。用 $F_1=2\mathrm{TP}/(2\mathrm{TP}+\mathrm{FP}+\mathrm{FN})$：$176/185=0.951$、$4/8=0.500$、$2/7=0.286$。巨觀 $F_1=(0.951+0.500+0.286)/3=0.579$。微觀 F_1 等於準確率 $91/100=0.910$。微觀被類別 A 主導；巨觀則揭露 B 與 C 大多被漏掉，所以對稀有類別而言，巨觀（加上逐類別召回率）才是誠實的數字。可以用 `precision_recall_fscore_support(..., average="macro")` 驗算。

參考文獻

1. Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022. doi:10.1109/TNNLS.2021.3084827
2. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, et al., “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision*, vol. 115, pp. 211–252, 2015. doi:10.1007/s11263-015-0816-y
3. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi:10.1109/5.726791
4. A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” technical report, University of Toronto, 2009. dataset page
5. T. Ridnik, E. Ben-Baruch, A. Noy, and L. Zelnik-Manor, “ImageNet-21K Pretraining for the Masses,” *NeurIPS*, 2021. arXiv
6. G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, et al., “The iNaturalist Species Classification and Detection Dataset,” *CVPR*, 2018. arXiv
7. B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do ImageNet Classifiers Generalize to ImageNet?,” *ICML*, PMLR vol. 97, 2019. paper
8. L. Beyer, O. J. Hénaff, A. Kolesnikov, X. Zhai, and A. van den Oord, “Are We Done with ImageNet?,” arXiv:2006.07159, 2020. arXiv
9. C. G. Northcutt, A. Athalye, and J. Mueller, “Pervasive Label Errors in Test Sets Destabilize Machine Learning Benchmarks,” *NeurIPS*, 2021. arXiv
10. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Advances in Neural Information Processing Systems 25* (*NIPS*), 2012. paper
11. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *ICLR*, 2015. arXiv
12. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” *CVPR*, 2015. doi:10.1109/CVPR.2015.7298594
13. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *CVPR*, 2016. doi:10.1109/CVPR.2016.308
14. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *CVPR*, 2016. doi:10.1109/CVPR.2016.90
15. K. He, X. Zhang, S. Ren, and J. Sun, “Identity Mappings in Deep Residual Networks,” *ECCV*, 2016. arXiv
16. G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” *CVPR*, 2017. arXiv
17. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” arXiv:1704.04861, 2017. arXiv
18. M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *ICML*, 2019. arXiv
19. I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, “Designing Network Design Spaces,” *CVPR*, 2020. arXiv

20. Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” *CVPR*, 2022. arXiv
21. J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-Excitation Networks,” arXiv:1709.01507, 2017 (journal version in *IEEE TPAMI*). arXiv
22. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *ICLR*, 2021. arXiv
23. H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, “Training Data-Efficient Image Transformers & Distillation through Attention,” *ICML*, PMLR vol. 139, 2021. paper
24. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” *ICCV*, 2021. doi:10.1109/ICCV48922.2021.00986
25. S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, “Transformers in Vision: A Survey,” *ACM Computing Surveys*, 2022. arXiv
26. H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “mixup: Beyond Empirical Risk Minimization,” *ICLR*, 2018. arXiv
27. S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features,” *ICCV*, 2019. arXiv
28. E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, “RandAugment: Practical Automated Data Augmentation with a Reduced Search Space,” *CVPR Workshops*, 2020. doi:10.1109/CVPRW50498.2020.00359
29. I. Loshchilov and F. Hutter, “Decoupled Weight Decay Regularization,” *ICLR*, 2019. arXiv
30. I. Loshchilov and F. Hutter, “SGDR: Stochastic Gradient Descent with Warm Restarts,” *ICLR*, 2017. arXiv
31. G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, “Deep Networks with Stochastic Depth,” *ECCV*, 2016. doi:10.1007/978-3-319-46493-0_39
32. R. Wightman, H. Touvron, and H. Jégou, “ResNet Strikes Back: An Improved Training Procedure in timm,” arXiv:2110.00476, 2021. arXiv
33. S. Kornblith, J. Shlens, and Q. V. Le, “Do Better ImageNet Models Transfer Better?,” *CVPR*, 2019. arXiv
34. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A Simple Framework for Contrastive Learning of Visual Representations,” *ICML*, 2020. arXiv
35. K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum Contrast for Unsupervised Visual Representation Learning,” *CVPR*, 2020. arXiv
36. M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, “Emerging Properties in Self-Supervised Vision Transformers,” *ICCV*, 2021. doi:10.1109/ICCV48922.2021.00951
37. K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, “Masked Autoencoders Are Scalable Vision Learners,” *CVPR*, 2022. doi:10.1109/CVPR52688.2022.01553
38. M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, et al., “DINOv2: Learning Robust Visual Features without Supervision,” *Transactions on Machine Learning Research*, 2024. arXiv
39. O. Siméoni, H. V. Vo, M. Seitzer, F. Baldassarre, M. Oquab, C. Jose, et al., “DINOv3,” arXiv:2508.10104, 2025. arXiv

40. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, et al., "Learning Transferable Visual Models From Natural Language Supervision," *ICML*, PMLR vol. 139, 2021. paper
41. J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo, and D. Tao, "A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (arXiv:2301.05712, 2023). arXiv
42. D. Hendrycks and T. Dietterich, "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations," *ICLR*, 2019. arXiv
43. D. Hendrycks, S. Basart, N. Mu, S. Kadavath, F. Wang, E. Dorundo, et al., "The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization," *ICCV*, 2021. arXiv
44. D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song, "Natural Adversarial Examples," *CVPR*, 2021. arXiv
45. I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and Harnessing Adversarial Examples," in *Proc. International Conference on Learning Representations* (*ICLR*), 2015. arXiv
46. R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, "Shortcut Learning in Deep Neural Networks," *Nature Machine Intelligence*, 2020. arXiv
47. C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On Calibration of Modern Neural Networks," *ICML*, 2017. arXiv
48. B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," *CVPR*, 2016. doi:10.1109/CVPR.2016.319
49. R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," *ICCV*, 2017. doi:10.1109/ICCV.2017.74
50. B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *CVPR*, 2018. doi:10.1109/CVPR.2018.00286
51. G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," arXiv: 1503.02531, 2015. arXiv
52. X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling Vision Transformers," *CVPR*, 2022. arXiv
53. M. Cherti, R. Beaumont, R. Wightman, M. Wortsman, G. Ilharco, C. Gordon, et al., "Reproducible Scaling Laws for Contrastive Language-Image Learning," *CVPR*, 2023. arXiv
54. T. Fawcett, "An Introduction to ROC Analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. doi:10.1016/j.patrec.2005.10.010
55. J. Davis and M. Goadrich, "The Relationship Between Precision-Recall and ROC Curves," *Proc. 23rd International Conference on Machine Learning* (*ICML*), pp. 233–240, 2006. doi:10.1145/1143844.1143874
56. T. Saito and M. Rehmsmeier, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets," *PLOS ONE*, vol. 10, no. 3, e0118432, 2015. doi:10.1371/journal.pone.0118432
57. D. Chicco and G. Jurman, "The Advantages of the Matthews Correlation Coefficient (MCC) over F1 Score and Accuracy in Binary Classification Evaluation," *BMC Genomics*, vol. 21, article 6, 2020. doi:10.1186/s12864-019-6413-7
58. M. Sokolova and G. Lapalme, "A Systematic Analysis of Performance Measures for Classification Tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009. doi:10.1016/

j.ipm.2009.03.002
59. J. Cohen, “A Coefficient of Agreement for Nominal Scales,” *Educational and Psychological Measurement*, vol. 20, no. 1, pp. 37–46, 1960. doi:10.1177/001316446002000104
60. M.-L. Zhang and Z.-H. Zhou, “A Review on Multi-Label Learning Algorithms,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 8, pp. 1819–1837, 2014. doi:10.1109/TKDE.2013.39
61. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010. doi:10.1007/s11263-009-0275-4
62. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. paper
63. H. Wang, S. Ge, E. P. Xing, and Z. C. Lipton, “Learning Robust Global Representations by Penalizing Local Predictive Power,” *NeurIPS*, 2019 (introduces ImageNet-Sketch). arXiv
64. A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, and B. Katz, “ObjectNet: A Large-Scale Bias-Controlled Dataset for Pushing the Limits of Object Recognition Models,” *NeurIPS*, 2019. dataset page
65. C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, “The Caltech-UCSD Birds-200-2011 Dataset,” Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. dataset page
66. J. Krause, M. Stark, J. Deng, and L. Fei-Fei, “3D Object Representations for Fine-Grained Categorization,” *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 554–561, 2013 (introduces Stanford Cars). doi:10.1109/ICCVW.2013.77
67. S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi, “Fine-Grained Visual Classification of Aircraft,” arXiv:1306.5151, 2013. arXiv
68. O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar, “Cats and Dogs,” *CVPR*, 2012 (introduces Oxford-IIIT Pets). dataset page
69. M.-E. Nilsback and A. Zisserman, “Automated Flower Classification over a Large Number of Classes,” *Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)*, 2008 (introduces Oxford Flowers-102). dataset page
70. L. Bossard, M. Guillaumin, and L. Van Gool, “Food-101 – Mining Discriminative Components with Random Forests,” *ECCV*, 2014. dataset page
71. L. Fei-Fei, R. Fergus, and P. Perona, “One-Shot Learning of Object Categories,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006 (the reference given for Caltech-101). dataset page
72. X. Zhai, J. Puigcerver, A. Kolesnikov, et al., “A Large-scale Study of Representation Learning with the Visual Task Adaptation Benchmark,” arXiv:1910.04867, 2019. arXiv
73. Z. Liu, Z. Miao, X. Zhan, J. Wang, B. Gong, and S. X. Yu, “Large-Scale Long-Tailed Recognition in an Open World,” *CVPR*, 2019 (introduces ImageNet-LT). arXiv

DLCV 02A・臉部辨識與度量學習

深度網路如何認出訓練時從未見過的人：封閉與開放集合中的驗證與識別、偵測與特徵點對齊、正規化嵌入、對比損失與三元組損失、以邊界為基礎的 softmax 損失（SphereFace、CosFace、ArcFace）、固定 FAR 下的 TAR、ROC 與 DET 曲線、CMC 與 FPIR、資料集及其撤下、族群差異、隱私與呈現攻擊。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/

你將學到

- 區分驗證、封閉集識別與開放集識別。
- 追蹤偵測、特徵點對齊、正規化嵌入與相似度評分。
- 比較成對損失與角度間隔分類損失。
- 為驗證或登錄庫搜尋選擇門檻與指標。
- 評估資料集適用性、人口群體差異、隱私與呈現攻擊。

先看全貌

分類器是從一份固定的答案清單裡挑一個。臉部辨識系統不能這樣運作：它明年會遇到的人，並不在訓練集裡。它學的是一種**衡量兩張臉有多像**的方法，再用一個門檻做決定。

> **用最簡單的話說**
>
> 想像學校門口有一位警衛，大部分訪客他都沒見過。他不可能把全世界的人都背下來。於是每位訪客出示一張附照片的卡片，警衛只問一個問題：「眼前這張臉，跟卡片上的臉夠不夠像？」多年下來，警衛學會哪些差異不重要（換了髮型、戴了眼鏡、臉上有陰影），哪些差異很重要（眼睛與鼻子的形狀）。臉部辨識模型學的是同一種本領：把每張臉變成一串很短的數字，讓同一個人的兩張照片得到相近的數字，不同人的照片得到不同的數字。

這個想法叫**度量學習**（metric learning），應用範圍遠不止人臉。同一套做法（學一個嵌入、用距離比較、用門檻決定）也驅動著行人重識別（DLCV 07）、影像檢索、語者驗證，以及 ML 06 的對比式自監督方法。人臉是把這套做法推得最遠的地方，因為規模極大（數百萬個身分、百萬分之一的錯誤率），賭注也高（手機解鎖、邊境管制、警方搜尋）。

臉部辨識同時也是帶有真實社會成本的技術：錯誤不會平均落在不同群體身上，訓練資料常在未經同意下蒐集，有些用途還受到法律限制。本篇把這些主題當成工程的一部分來談，陳述事實，並附上查證過的來源。

兩份綜述構成這個領域的骨架：Wang 與 Deng 對深度臉部辨識的回顧 [1]，以及 NIST 對商用演算法族群效應的報告 [2]。我們會在「現代觀點」回頭談這兩份文獻。

相關章節。 softmax、交叉熵、ROC 與校準的基礎在 DLCV 02。重識別、針對人的三元組損失與 CMC 指標在 DLCV 07。對比式學習與 InfoNCE 在 ML 06，影像–文字嵌入（CLIP）在 ML 10。最近鄰比對與開放集合拒絕的傳統觀點在 PR 04 與 PR 10。

辨識就是比較：開放集合問題

白話版。驗證判斷兩張臉是否同一人，識別則搜尋已儲存身分；兩者都能比較訓練時未見的人。

驗證、識別、封閉集合與開放集合

令 x 為一張臉部影像，$f_\theta(x) \in \mathbb{R}^d$ 為它的嵌入，由參數為 θ 的網路產生。比較兩張影像 x_a, x_b 會得到一個**相似度分數**

$$s(x_a, x_b) = \cos \angle(f_\theta(x_a), f_\theta(x_b)) = \frac{f_\theta(x_a)^\top f_\theta(x_b)}{\|f_\theta(x_a)\| \, \|f_\theta(x_b)\|} \in [-1, 1],$$

其中 $\|\cdot\|$ 是歐氏範數。已經儲存參考影像的人稱為**已註冊**（enrolled）；他們儲存的嵌入（每人一個或多個）構成**圖庫**（gallery），而要被辨識的新影像稱為**探測影像**（probe）。一個固定的**門檻**（threshold）t 把分數轉成決定。

任務	問題	輸出	典型用途
驗證（1:1）	探測影像 x 是宣稱的那個人 c 嗎？	若 $s(x, g_c) \geq t$ 就接受，否則拒絕	手機解鎖、護照閘門
封閉集合識別（1:N）	x 是 N 位已註冊者中的哪一位？	$\arg\max_n s(x, g_n)$	基準測試；已知、封閉的群體
開放集合識別（1:N）	x 有註冊嗎？如果有，是誰？	最佳匹配的分數 $\geq t$ 就回傳它，否則回答「未註冊」	監控名單、照片搜尋
分類（DLCV 02）	x 屬於哪個訓練類別？	訓練時見過的類別之一	測試時不用在人臉上

這裡 g_n 是第 n 位的圖庫嵌入，N 是圖庫大小。圖 1 畫出三種辨識任務。重點在最後一列：臉部分類器的類別是訓練時的身分，而測試時幾乎每張探測影像都屬於別人。NIST 對實際部署系統運作方式的描述也是同一個意思：臉部辨識演算法內建的並不是某個特定的人；它從通常沒見過的人身上抽出特徵向量，比較兩個向量來得到分數 [2]。

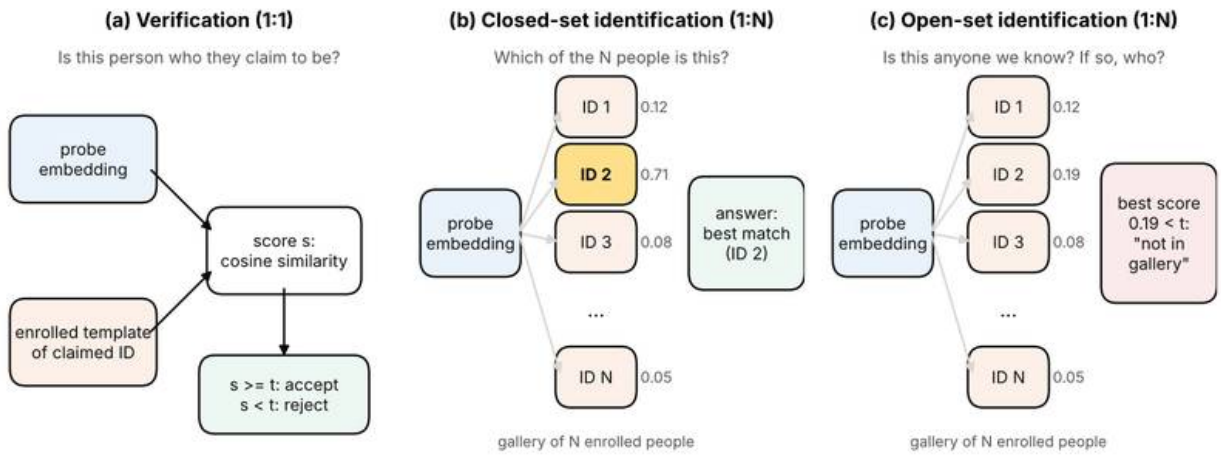


圖 *1* — (*a*) 驗證把探測影像與一個宣稱的身分比較。(*b*) 封閉集合識別一定會回傳最佳匹配。(*c*) 開放集合識別還需要門檻，因為這個人可能根本沒有註冊。

封閉集合與開放集合是評估方式的性質，不是網路的性質。封閉集合識別的測試保證每張探測影像在圖庫中都有對應的人，所以「挑最佳匹配」原則上一定正確。開放集合測試包含未註冊者的探測影像，這時門檻就不可避免。多數真實的搜尋都是開放集合。一般分類器也有同樣的區別，稱為開放集合辨識，見 PR 10；對臉部而言，這是常態，不是邊緣情況。

為什麼不直接訓練一個分類器？

白話版。分類器只為訓練身分配置輸出；辨識則重用學得的嵌入來比較新身分。

用 softmax 與交叉熵在 C 個身分上訓練標準分類器，其倒數第二層本來就已經很有用。DeepFace [6] 是最早的深度系統之一，它在一個擁有身分標籤、超過 4,000 人的資料集上訓練網路，並證明所學的表示可以推廣到不受限制的照片中的人臉；輸出層本身不會用在新的人身上。FaceNet [7] 則改為直接用三元組損失訓練嵌入。這兩條路線（分類式訓練與成對訓練）後來在 2017–2019 年匯流成以邊界為基礎的 softmax 損失，也就是我們下面要學的內容。

什麼樣的嵌入適合開放集合使用？SphereFace 把理想準則說成兩個性質：**類別內的最大距離**應該小於**類別間的最小距離** [12]。普通分類器只要求每個訓練樣本落在決策邊界的正確一側，並不在乎類別是否緊密。邊界（margin）就是用來要求緊密度的方法。

臉部辨識流程

白話版。先偵測臉部、定位特徵點，以旋轉與縮放對齊裁切影像，再計算嵌入。

1. **偵測**（detection）找出每張臉周圍的框。
2. **特徵點定位**（landmark localization）找出幾個關鍵點，通常是五個：兩個眼睛中心、鼻尖與兩個嘴角。
3. **對齊**（alignment）扭轉影像，使特徵點對上固定的模板，並裁切成固定大小的小影像（例如 112×112 像素）。
4. **嵌入**（embedding）把裁切影像映射成向量，通常做 L_2 正規化，使長度為 1。
5. **比較**（comparison）計算與一個模板（驗證）或圖庫（識別）的餘弦相似度，再套用門檻。

偵測與特徵點

臉部偵測是專門針對單一類別的物件偵測，DLCV 03 的一般想法都適用。傳統的里程碑是 Viola–Jones 串接分類器，在 DLCV 03 的傳統偵測器補充說明中有整理。臉部辨識文獻中常見的前端有兩種深度偵測器，兩者都在預測框的同時預測五個特徵點：

- **MTCNN** [4] 是三個卷積網路組成的串接結構，由粗到細預測臉部框與特徵點位置，訓練時使用線上困難樣本挖掘策略。
- **RetinaFace** [5] 是具有多工輸出頭的單階段偵測器；它的作者在 WIDER FACE 資料集上手動標註五個特徵點，並證明這項額外的監督有助於偵測困難的臉。

這些不是歷史細節：CosFace 用 MTCNN 的特徵點對齊臉部 [13]，ArcFace 的期刊版本則用 RetinaFace 的特徵點 [16]。偵測器與特徵點的誤差會傳遞到後面每一步，所以辨識結果永遠是整條流程的結果。

以相似變換對齊

白話版。我們希望每張臉都正對鏡頭、大小與位置相同。相似變換（旋轉、等比例縮放、平移）的彈性剛好足以做到這點，又不會扭曲臉的形狀。

相似變換（similarity transform）把點 $\mathbf{p} \in \mathbb{R}^2$ 映射為

$$T(\mathbf{p}) = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \mathbf{p} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}, \qquad a = \sigma\cos\phi,\ b = \sigma\sin\phi,$$

其中 $\sigma > 0$ 是縮放比例，ϕ 是旋轉角，(t_x, t_y) 是平移量。它有四個參數 (a, b, t_x, t_y)，而且方程式對這些參數是線性的。給定偵測到的特徵點 $\mathbf{p}_1, \dots, \mathbf{p}_5$ 與模板位置 $\mathbf{q}_1, \dots, \mathbf{q}_5$，我們選擇使 $\sum_i \|T(\mathbf{p}_i) - \mathbf{q}_i\|^2$ 最小的參數：十條方程式、四個未知數，一次最小平方解。CosFace 與 SphereFace 都描述它們的裁切影像是由偵測到的特徵點「經相似變換」得到的 [12], [13]。

```
import numpy as np

# A toy 5-point template (left eye, right eye, nose tip, left and right mouth
# corners) in a 112 x 112 crop. The coordinates are our own illustration.
template = np.array([[38., 46], [74, 46], [56, 66], [42, 86], [70, 86]])

def fit_similarity(src, dst):
    """Least-squares a, b, tx, ty with dst ~ [[a, -b], [b, a]] @ src + t."""
    n = len(src)
    A = np.zeros((2 * n, 4))
    A[0::2] = np.c_[src[:, 0], -src[:, 1], np.ones(n), np.zeros(n)]
    A[1::2] = np.c_[src[:, 1],  src[:, 0], np.zeros(n), np.ones(n)]
    a, b, tx, ty = np.linalg.lstsq(A, dst.reshape(-1), rcond=None)[0]
    return np.array([[a, -b, tx], [b, a, ty]])

# "Detected" landmarks: the template rotated by 20 degrees, scaled by 1.8,
# shifted into a large photo, plus 1-pixel detector noise.
rng = np.random.default_rng(0)
ang, sc = np.radians(20), 1.8
R = sc * np.array([[np.cos(ang), -np.sin(ang)], [np.sin(ang), np.cos(ang)]])
detected = template @ R.T + [310, 205] + rng.normal(scale=1.0, size=(5, 2))

M = fit_similarity(detected, template)          # photo -> crop
warped = detected @ M[:, :2].T + M[:, 2]
scale = np.hypot(M[0, 0], M[1, 0])
print(f"scale {scale:.3f} (expect {1 / sc:.3f}), "
      f"rotation {np.degrees(np.arctan2(M[1, 0], M[0, 0])):.1f} deg (expect -20.0)")
print(f"RMS landmark error after alignment: {np.sqrt(((warped - template) **
 ↪  2).sum(1).mean()):.2f} px")
```

```
scale 0.559 (expect 0.556), rotation -19.8 deg (expect -20.0)
RMS landmark error after alignment: 0.56 px
```

這個擬合幾乎完全還原了縮放與旋轉，剩下的誤差（0.56 像素）就是 1 像素的特徵點雜訊經五個點平均後留下的部分。矩陣 `M` 接著用來把整張影像扭轉成裁切影像。

僅線上版：對齊臉部：最小平方相似變換 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/notes/對齊臉部最小平方相似變換/*

嵌入與模板

嵌入網路是 DLCV 02 中的普通影像骨幹（ResNet 或視覺 Transformer），只是最後一層輸出的是 d 維向量，而不是類別分數。FaceNet 用 $d = 128$ [7]；ArcFace 的實驗用 $d = 512$ 與 112×112 的裁切影像 [16]。當一個人用多張影像或一段影片註冊時，這些嵌入會合併成一個**模板**（template）；ArcFace 直接取模板中所有影像的特徵中心（平均）[16]。既然網路的輸出在比較前會先正規化，把平均後的向量再正規化一次也就很自然。

超球面上的嵌入

白話版。如果我們把每個嵌入縮放成長度 1，它們就全都落在一個球面上。於是「兩張臉有多像」變成「兩支箭頭之間的夾角有多小」。長度常常反映影像品質而不是身分，現在它不會再干擾比較。

為什麼要正規化，為什麼用餘弦

對單位向量 $\mathbf{u}, \mathbf{v}$（即 $\|\mathbf{u}\| = \|\mathbf{v}\| = 1$），

$$\|\mathbf{u} - \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 - 2\,\mathbf{u}^\top \mathbf{v} = 2 - 2\cos\theta,$$

其中 θ 是兩者的夾角。所以在球面上，歐氏距離平方與餘弦相似度給出相同的排序，對其中一個設門檻就等於對另一個設門檻。正規化拿掉了一個網路原本可能濫用的自由度：在沒有正規化的 softmax 中，網路只要把容易樣本的特徵拉長，就能降低它們的損失，而不必讓類別變得更緊密。NormFace 分析了這點，並證明直接在正規化後的特徵上訓練，需要額外的縮放因子才能收斂 [11]。

高維度還帶來一個有用的事實：**隨機方向幾乎彼此正交**。如果 $\mathbf{u}$ 與 $\mathbf{v}$ 在 $\mathbb{R}^d$ 的單位球面上獨立且均勻分佈，則 $\mathbb{E}[\cos\theta] = 0$，且 $\cos\theta$ 的標準差是 $1/\sqrt{d}$。因此異人分數會緊密聚集在 0 附近，門檻只需要超過這個狹窄分佈的尾端。

```
import numpy as np

rng = np.random.default_rng(0)
e = rng.normal(size=(2000, 512))
e /= np.linalg.norm(e, axis=1, keepdims=True)       # 2,000 random unit vectors
a, b = e[0], e[1]
cos = a @ b
print(f"cos = {cos:.4f}   |a-b|^2 = {np.sum((a - b) ** 2):.4f}   2-2cos = {2 - 2 * cos:.4f}")

S = e @ e.T
off = S[np.triu_indices(len(e), 1)]                 # all distinct pairs
print(f"random pairs: mean cos {off.mean():.4f}, std {off.std():.4f}, "
      f"1/sqrt(512) = {1 / np.sqrt(512):.4f}, max {off.max():.3f}")
```

```
cos = 0.0472   |a-b|^2 = 1.9057   2-2cos = 1.9057
random pairs: mean cos -0.0000, std 0.0442, 1/sqrt(512) = 0.0442, max 0.214
```

在 512 維的將近兩百萬個隨機配對中，最大的餘弦只有 0.214。訓練好的網路產生的不是隨機嵌入，但這個畫面仍然成立：不同的人落在接近 90° 的夾角上，而訓練的工作就是把每個人的影像拉進圍繞同一方向的一個小球冠。

成對損失：對比損失與三元組損失

白話版。訓練嵌入最直接的方法，是給網路看成對或三張一組的影像，並告訴它「這兩張應該靠近，那兩張應該遠離」。困難在於這類例子幾乎都已經很容易，除非我們去找困難的例子，否則網路從中學不到什麼。

對比損失

對比損失（contrastive loss）由 Chopra、Hadsell 與 LeCun [8] 為了用孿生網路做臉部驗證而提出，後來由 Hadsell、Chopra 與 LeCun [9] 發展完整，它一次看一對影像：

$$\mathcal{L}_{\text{con}}(x_a, x_b) = y\, D^2 + (1-y)\,[\,m - D\,]_+^2, \qquad D = \|f_\theta(x_a) - f_\theta(x_b)\|,$$

其中若這對影像是同一個人則 $y=1$，否則 $y=0$；$m>0$ 是邊界，$[z]_+ = \max(z,0)$。同人的配對會被拉近；異人的配對只會被推開，直到相距 m 為止。除了常數倍數與標籤約定外，這個平方 hinge 形式就是 [9] 中的形式；[8] 則提出了背後的想法：訓練孿生網路，讓學到的距離把同人與異人的配對分開。

三元組損失與困難負樣本挖掘

FaceNet [7] 的三**元組損失**（triplet loss）不是對距離設門檻，而是比較距離。它的定義，以及在重識別中的用法，見 DLCV 07 與 ML 06；FaceNet 的形式是在單位長度的嵌入上用距離平方：

$$\mathcal{L}_{\text{tri}}(a,p,n) = [\,\|f(a)-f(p)\|^2 - \|f(a)-f(n)\|^2 + \alpha\,]_+,$$

其中 a 是錨點影像（anchor），p 是正樣本（同一個人），n 是負樣本（另一個人），$f = f_\theta$ 是嵌入，$\alpha > 0$ 是邊界。本章要補充的是**挖掘**（mining）問題。對一組距離平方為 d_{ap} 的錨點–正樣本配對，一個距離平方為 d_{an} 的負樣本屬於：

- **容易**（easy），若 $d_{an} \geq d_{ap} + \alpha$（損失為零、梯度為零），
- **半困難**（semi-hard），若 $d_{ap} \leq d_{an} < d_{ap} + \alpha$（比正樣本遠，但落在邊界之內；這是 FaceNet 的定義 [7]），
- **困難**（hard），若 $d_{an} < d_{ap}$（比正樣本更近）。

FaceNet 在大的小批次內線上挑選半困難負樣本，因為訓練初期最困難的負樣本會導致不好的局部極小值 [7]。後來廣泛使用的另一種做法是**批次內最困難**（batch-hard）挖掘：在由 P 個身分 × K 張影像組成的批次中，替每個錨點取最遠的正樣本與最近的負樣本 [10]。

```
import numpy as np

rng = np.random.default_rng(0)
P, Kimg, alpha = 8, 4, 0.2                  # 8 people x 4 images, margin
centres = rng.normal(size=(P, 64))
y = np.repeat(np.arange(P), Kimg)
e = centres[y] + 1.5 * rng.normal(size=(len(y), 64))
e /= np.linalg.norm(e, axis=1, keepdims=True)
d2 = ((e[:, None] - e[None]) ** 2).sum(-1)  # squared distances

easy = semi = hard = 0
for a in range(len(y)):
    for p in np.flatnonzero((y == y[a]) & (np.arange(len(y)) != a)):
        dn = d2[a, y != y[a]]               # distances to all negatives
        hard += (dn < d2[a, p]).sum()
        semi += ((dn >= d2[a, p]) & (dn < d2[a, p] + alpha)).sum()
        easy += (dn >= d2[a, p] + alpha).sum()
tot = easy + semi + hard
print(f"{tot} triplets: easy {easy / tot:.1%}, semi-hard {semi / tot:.1%}, hard {hard /
 ↪  tot:.1%}")

# batch-hard: for each anchor, farthest positive and closest negative
same = y[:, None] == y[None]
hp = np.where(same, d2, -np.inf).max(1)
hn = np.where(~same, d2, np.inf).min(1)
loss = np.maximum(hp - hn + alpha, 0)
print(f"batch-hard loss {loss.mean():.3f}; anchors with non-zero loss: {(loss > 0).sum()} of
 ↪  {len(y)}")
```

```
2688 triplets: easy 92.8%, semi-hard 5.4%, hard 1.9%
batch-hard loss 0.213; anchors with non-zero loss: 28 of 32
```

在這個合成的批次中，全部 2,688 組有效三元組裡有 92.8% 屬於容易，沒有任何貢獻。對所有三元組取平均會稀釋少數有用的三元組；批次內最困難挖掘則為 32 個錨點中的 28 個找到非零損失。這是人臉規模下成對訓練的核心弱點：可能的三元組數量隨資料集大小的三次方增長，幾乎全都是容易的，結果還取決於取樣策略與批次的組成。

以邊界為基礎的 softmax 損失

白話版。與其讓影像彼此比較，不如給每個訓練身分一個學出來的「錨定方向」放在球面上，並訓練每張影像以明確的邊界指向自己身分的方向：不只比指向其他方向更近，而是近一個固定的角度。這樣每張影像只有一項損失，就同時與所有身分比較。

從 softmax 到正規化 softmax

從 DLCV 02 的 softmax 交叉熵出發。對影像 i 的特徵 $\mathbf{x}_i \in \mathbb{R}^d$，其身分標籤為 $y_i \in \{1, \dots, C\}$，最後一層的權重行向量為 $W_j \in \mathbb{R}^d$、偏差為 b_j，損失為

$$\mathcal{L}_{\text{softmax}} = -\log \frac{e^{W_{y_i}^\top \mathbf{x}_i + b_{y_i}}}{\sum_{j=1}^{C} e^{W_j^\top \mathbf{x}_i + b_j}}.$$

令 $b_j = 0$，並把 logit 寫成 $W_j^\top \mathbf{x}_i = \|W_j\| \, \|\mathbf{x}_i\| \cos\theta_j$，其中 θ_j 是 $\mathbf{x}_i$ 與 W_j 之間的夾角。現在把兩者都正規化：$\|W_j\| = 1$，並把 $\|\mathbf{x}_i\|$ 固定為常數 s。這時 logit 是 $s\cos\theta_j$，預測只取決於角度 [11], [13], [16]：

$$\mathcal{L}_{\text{norm}} = -\log \frac{e^{s\cos\theta_{y_i}}}{\sum_{j=1}^{C} e^{s\cos\theta_j}}.$$

尺度 $s > 0$ 是必要的，因為餘弦落在 $[-1, 1]$：沒有它，即使角度排列得完美，softmax 仍然離有信心很遠，損失就降不下來。NormFace 分析了這種無法收斂的現象 [11]，CosFace 則推導出 s 的一個下界，隨類別數增加而增大 [13]；CosFace 與 ArcFace 都使用 $s = 64$ [13], [16]。

僅線上版：尺度 s 要多大？— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/notes/尺度-s-要多大/*

現在每一行 W_j 都是單位向量：球面上身分 j 的**類別中心**（class center），也叫**代理點**（proxy）。這是與成對損失相比最關鍵的結構改變。每張影像在每一步都與全部 C 個類別中心比較，所以沒有什麼需要挖掘，而且比較是全域的，不受限於一個批次。類別中心只用於訓練；測試時會丟掉最後一層，只保留嵌入網路 [16]。

加上邊界的三種方式

正規化 softmax 在對所有 $j \neq y_i$ 都有 $\theta_{y_i} < \theta_j$ 時就算滿足：特徵與自己的中心之間的夾角，比與任何其他中心的都小。這會把決策邊界剛好放在兩個中心的正中間，所以靠近邊界的樣本雖然正確，卻只是勉強正確。**邊界**（margin）把目標 logit 從 $s\cos\theta_{y_i}$ 改成 $s\,\psi(\theta_{y_i})$，其中 $\psi(\theta) < \cos\theta$，使目標 logit 更難達成，其他 logit 仍是 $s\cos\theta_j$：

$$\mathcal{L}_{\text{margin}} = -\log \frac{e^{s\,\psi(\theta_{y_i})}}{e^{s\,\psi(\theta_{y_i})} + \sum_{j \neq y_i} e^{s\cos\theta_j}}.$$

三個經典損失只在 ψ 上不同：

損失	目標 logit $\psi(\theta)$	邊界類型	論文中的數值
SphereFace（A-Softmax）[12]	$\cos(m\theta)$，延伸成保持單調（見下）	乘法式角度邊界	$m = 4$
CosFace（LMCL）[13]；也有 AM-Softmax [14]	$\cos\theta - m$	加法式餘弦邊界	$m = 0.35$、$s = 64$
ArcFace [15], [16]	$\cos(\theta + m)$	加法式角度邊界	$m = 0.5$、$s = 64$

損失	目標 logit $\psi(\theta)$	邊界類型	論文中的數值

這裡 m 是邊界（SphereFace 為整數，CosFace 為餘弦的偏移量，ArcFace 為以弧度表示的角度），$\theta = \theta_{y_i}$ 是樣本與自己的類別中心之間的夾角；最後一欄列的是論文中報告的數值。由於 $\cos(m\theta)$ 在 $[0, \pi]$ 上不是單調的，SphereFace 對 $\theta \in [k\pi/m, (k+1)\pi/m]$、$k \in \{0, \dots, m-1\}$ 使用 $\psi(\theta) = (-1)^k \cos(m\theta) - 2k$ [12]。最初的 SphereFace 只正規化權重，訓練初期還需要普通 softmax 的幫助 [16]。AM-Softmax [14] 與 CosFace 大約同時發表，用的是同樣的加法式餘弦邊界。

```
import numpy as np

def sphereface_psi(theta, m=4):
    """SphereFace's monotone extension of cos(m*theta) to [0, pi]."""
    k = np.floor(theta * m / np.pi)
    return (-1) ** k * np.cos(m * theta) - 2 * k

print(" angle  softmax  SphereFace  CosFace  ArcFace")
for deg in [10, 30, 45, 60, 90]:
    t = np.radians(deg)
    print(f"{deg:5d}   {np.cos(t):6.3f}   {sphereface_psi(t):8.3f}   "
          f"{np.cos(t) - 0.35:6.3f}   {np.cos(t + 0.5):6.3f}")
```

```
 angle  softmax  SphereFace  CosFace  ArcFace
   10    0.985      0.766    0.635    0.781
   30    0.866     -0.500    0.516    0.520
   45    0.707     -1.000    0.357    0.282
   60    0.500     -1.500    0.150    0.024
   90    0.000     -3.000   -0.350   -0.479
```

這張表列出五個角度下的目標 logit（除以 s）。每種邊界都會降低正確類別的 logit，所以損失要等到角度很小才會變小。懲罰的增長方式各不相同：CosFace 在每個角度都減去同樣的 0.35；ArcFace 的懲罰在 0° 附近很小（在 10° 為 0.204），角度越寬越大；$m = 4$ 的 SphereFace 在寬角度時非常嚴格。

圓上的幾何

白話版。想像圓上有兩個類別中心。沒有邊界時，圓恰好被它們平分。有了邊界，每個類別只佔不到一半，中間有一段不屬於任何人的區域：落在那裡的樣本仍然有損失。訓練會把樣本推出這段區域，推向自己的中心。

兩個類別的中心 W_1, W_2 相距角度 φ，一個特徵落在它們之間的弧上，當 $\psi(\theta_1) > \cos\theta_2$（其中 $\theta_2 = \varphi - \theta_1$）時，類別 1 的樣本就「滿足」了。解出來就是類別 1 的邊界：

- 正規化 softmax：$\theta_1 < \varphi/2$（沒有間隙）；

- SphereFace：$m\,\theta_1 < \theta_2$，所以 $\theta_1 < \varphi/(m+1)$；兩個類別邊界之間的間隙是 $\varphi(m-1)/(m+1)$，它**隨 φ 增加**；
- CosFace：$\cos\theta_1 - m > \cos(\varphi - \theta_1)$；間隙在餘弦上是固定量，所以它在角度上的大小取決於 φ；
- ArcFace：$\theta_1 + m < \theta_2$，所以 $\theta_1 < (\varphi - m)/2$；間隙對任何 φ 都恰好是 m 弧度。

ArcFace 的作者稱這個邊界等於超球面上的**測地距離**（geodesic distance），這就是名稱的由來 [15], [16]。圖 2 畫出 $\varphi = 80°$ 時的四種情況；SphereFace 用 $m = 2$，好讓它的區域能放進同一張圖。

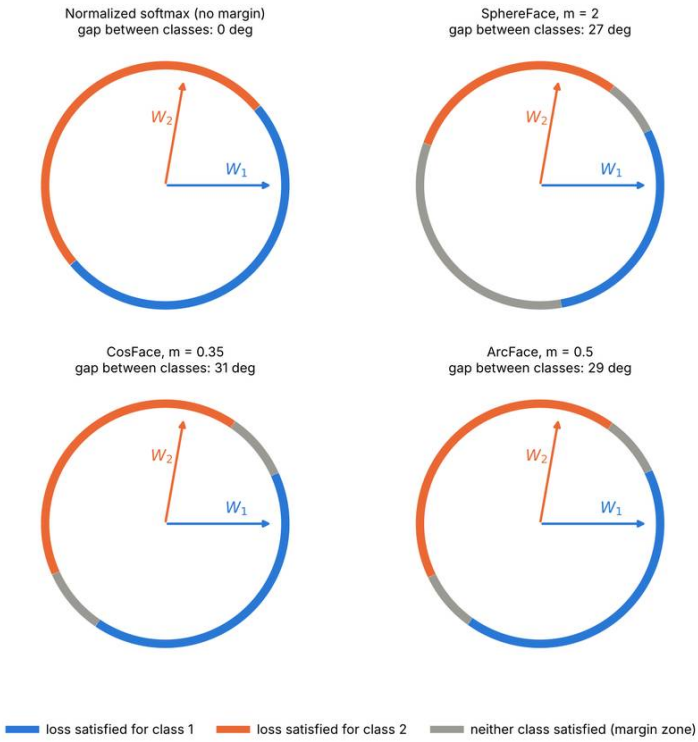


圖 2 — 兩個中心相距 *80°* 的類別，各損失在哪些地方被滿足。灰色弧是邊界區，落在其中的任一類別的樣本仍有損失。兩個中心之間的間隙：沒有邊界時為 *0°*，*SphereFace*（$m = 2$）為 *27°*，*CosFace*（$m = 0.35$）為 *31°*，*ArcFace*（$m = 0.5$）為 *29°*（$= 0.5$ 弧度）。

從公式還可以得到一個實務細節：$\cos(\theta + m)$ 只有在 $\theta + m \le \pi$ 時才遞減。對於與自己中心的夾角大於 $\pi - m$ 的樣本（嚴重分類錯誤或標籤錯誤的樣本），ArcFace 的目標 logit 在樣本越離越遠時反而會開始上升。實作必須明確處理這個範圍，例如在那裡改用單調的懲罰。在下面的玩具程式中，所有角度都遠低於這個範圍。

僅線上版：並排比較邊界損失：一個公式與它的特殊情況 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/notes/並排比較邊界損失一個公式與它的特殊情況/*

玩具實驗：softmax 對 ArcFace

白話版。我們用同一個小網路在合成的「人臉」上訓練兩次：一次用普通 softmax，一次用 ArcFace。然後看訓練集的叢集有多緊密，更重要的是，網路能多好地區分它沒見過的人。

資料模仿人臉問題。每個合成身分是 16 維輸入裡 4 維「身分」子空間中的一個隨機點；每張影像還會在其餘 12 個維度加上很大的隨機「干擾」成分（代表姿勢與光線），並在所有維度加上小雜訊。網路是一個 64 個 ReLU 單元的隱藏層，後面接一個線性映射到 $d = 4$ 的嵌入。前向傳遞與梯度都用手寫實作，包括通過正規化與通過 $\cos(\theta + m)$ 的連鎖律，並使用普通的梯度下降。我們在 100 個身分上訓練，並在 50 個訓練時從未出現的身分上測試驗證。固定 FAR = 1% 的 TAR（下一節定義）是：當門檻讓 1% 的異人配對通過時，被接受的同人測試配對的比例。

```
import numpy as np

rng = np.random.default_rng(0)
D, K = 16, 4                                        # input dim, identity dims
U = np.linalg.qr(rng.normal(size=(D, D)))[0]      # random orthonormal basis

def sample(ids, n_per):
    """Identity signal in 4 directions, large 'pose/light' nuisance in 12."""
    y = np.repeat(np.arange(len(ids)), n_per)
    nuis = 1.5 * rng.normal(size=(len(y), D - K))
    x = ids[y] @ U[:, :K].T + nuis @ U[:, K:].T + 0.3 * rng.normal(size=(len(y), D))
    return x, y

unit = lambda v, ax: v / np.linalg.norm(v, axis=ax, keepdims=True)

def train(x, y, C, d=4, margin=None, s=16.0, steps=800, lr=0.2, H=64):
    """One-hidden-layer embedding net + class centres W, plain gradient descent.
    margin=None: ordinary softmax on raw features; otherwise ArcFace logits."""
    r = np.random.default_rng(1)
    W1 = r.normal(size=(D, H)) / 4; b1 = np.zeros(H)
    W2 = r.normal(size=(H, d)) / 8; W = r.normal(size=(d, C))
    Y = np.eye(C)[y]
    for _ in range(steps):
        h = np.maximum(x @ W1 + b1, 0); z = h @ W2          # embeddings
        if margin is None:
            L = z @ W
        else:
            f, w = unit(z, 1), unit(W, 0)
            th = np.arccos(np.clip(f @ w, -1 + 1e-7, 1 - 1e-7))   # angles to centres
            L = s * np.where(Y > 0, np.cos(th + margin), np.cos(th))
        p = np.exp(L - L.max(1, keepdims=True)); p /= p.sum(1, keepdims=True)
        g = (p - Y) / len(y)                                  # dLoss/dlogits
        if margin is None:
            gz, gW = g @ W.T, z.T @ g
        else:                                                 # chain rule through
         ↪  cos(theta+m)
            gc = g * s * np.where(Y > 0, np.sin(th + margin) / np.sin(th), 1.0)
            gf, gw = gc @ w.T, f.T @ gc
            gz = (gf - (gf * f).sum(1, keepdims=True) * f) / np.linalg.norm(z, axis=1,
             ↪  keepdims=True)
            gW = (gw - (gw * w).sum(0, keepdims=True) * w) / np.linalg.norm(W, axis=0,
             ↪  keepdims=True)
```

```
        gh = (gz @ W2.T) * (h > 0)
        W1 -= lr * x.T @ gh; b1 -= lr * gh.sum(0); W2 -= lr * h.T @ gz; W -= lr * gW
    return lambda xx: unit(np.maximum(xx @ W1 + b1, 0) @ W2, 1)

def intra_angle(e, y):            # mean angle (degrees) to own class mean direction
    cen = unit(np.stack([e[y == j].mean(0) for j in np.unique(y)]), 1)
    return np.degrees(np.arccos(np.clip((e * cen[y]).sum(1), -1, 1))).mean()

def tar_at_far(e, y, far=0.01):   # verification on all pairs of unseen people
    i, j = np.triu_indices(len(y), 1)
    sc, same = (e[i] * e[j]).sum(1), y[i] == y[j]
    thr = np.quantile(sc[~same], 1 - far)
    return (sc[same] > thr).mean()

ids = rng.normal(size=(150, K))
xtr, ytr = sample(ids[:100], 10)          # 100 training identities
xte, yte = sample(ids[100:], 10)          # 50 identities never seen in training
for name, m in [("softmax", None), ("norm-softmax", 0.0), ("ArcFace m=0.5", 0.5)]:
    emb = train(xtr, ytr, 100, margin=m)
    print(f"{name:14s} train intra {intra_angle(emb(xtr), ytr):5.1f} deg | "
          f"unseen intra {intra_angle(emb(xte), yte):5.1f} deg | "
          f"unseen TAR@FAR=1% {tar_at_far(emb(xte), yte):.3f}")
```

```
softmax        train intra  27.1 deg | unseen intra  43.9 deg | unseen TAR@FAR=1% 0.082
norm-softmax   train intra  16.1 deg | unseen intra  30.1 deg | unseen TAR@FAR=1% 0.098
ArcFace m=0.5  train intra  11.6 deg | unseen intra  24.4 deg | unseen TAR@FAR=1% 0.248
```

這個腳本在 CPU 上大約跑十秒。在這次玩具實驗中，訓練影像到其類別中心的平均夾角，從 27.1°（softmax）降到 16.1°（只做正規化），再降到 11.6°（ArcFace），而沒見過的人的叢集也縮緊了（43.9° 到 24.4°）。FAR = 1% 時，對未見過的人的驗證率大約變為三倍，從 0.082 到 0.248。有兩點提醒。第一，絕對數字偏低，因為對有雜訊的資料用 4 維嵌入本來就是困難的問題；只有比較才有意義。第二，這些都是單次執行；換另外兩個隨機種子，在我們的執行中 softmax < 正規化 softmax < ArcFace 的順序維持不變，但確切的數值會改變。

圖 3 在有六個身分的二維嵌入中呈現同樣的效果，可以直接看出來：softmax 讓每個類別沿著一條射線散開（它可以靠拉長特徵來降低損失），而 ArcFace 把每個類別擠進一小段弧。

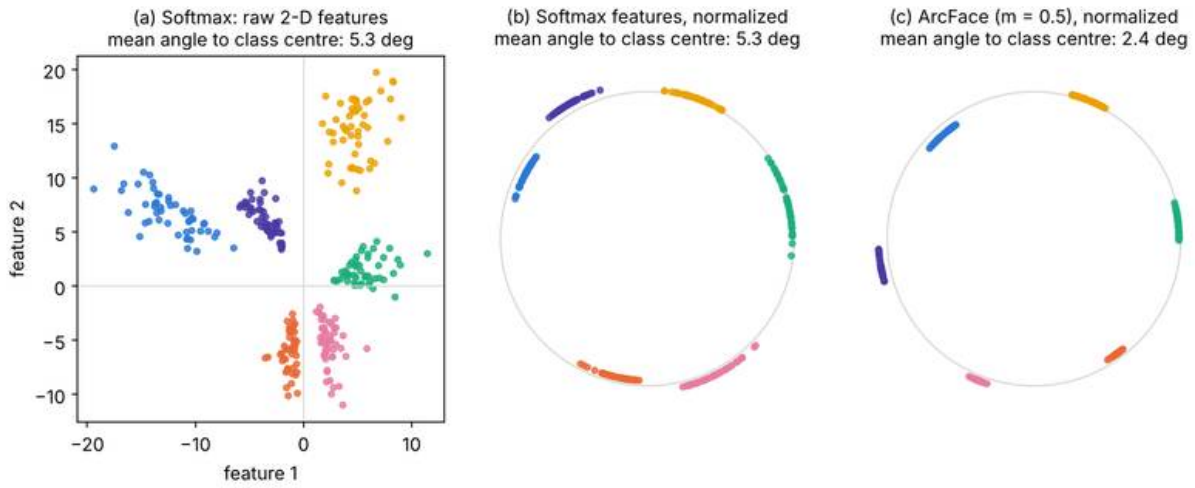


圖 3 — 玩具網路把六個合成身分嵌入二維。(*a*) *Softmax* 特徵沿徑向散開。(*b*) 正規化後，*softmax* 的類別覆蓋很寬的弧（與類別中心的平均夾角 *5.3°*）。(*c*) $m = 0.5$ 的 *ArcFace* 給出更緊的弧（*2.4°*）。數字來自這次玩具實驗。

邊界會直接讓訓練身分的叢集變緊。這是否能推廣到新的人，取決於訓練身分有多少、多樣到什麼程度；訓練身分很少時，網路可以把它們完美分開，卻什麼一般性的東西都沒學到。這就是臉部辨識訓練集要包含數十萬到數百萬個身分的原因之一（見資料集一節）。

類別中心、子中心與自適應邊界

白話版。一旦每個身分有了中心，幾種改良就很自然地出現。它們都保留邊界 softmax，只改變中心或邊界的某一件事。

中心損失（center loss）[17] 比角度損失更早出現，它把每個特徵拉向學出來的類別中心。Sub-center ArcFace [18], [16] 給每個身分多個中心，讓有雜訊或標籤錯誤的影像可以聚集在遠離乾淨影像的地方。MagFace [19] 與 AdaFace [20] 把邊界連到影像品質（利用特徵的大小），讓品質很差的影像不會主導訓練。Partial FC [21] 每一步只更新被取樣的一部分中心，來降低數百萬個中心的成本。要記住的只有一點：類別中心是訓練用的工具，這些方法調整的是影像被拉向它們的嚴格程度與成本。

僅線上版：一個類別不只一個中心：子中心、品質感知邊界與 *Partial FC* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/notes/一個類別不只一個中心子中心品質感知邊界與-partial-fc/*

評估驗證

白話版。取很多組同一個人的照片配對（**同人**配對，genuine pairs），以及很多組不同人的配對（**異人**配對，impostor pairs）。好的系統會給同人配對高分，給異人配對低分。任何一個門檻都要在兩種錯誤之間取捨：放冒名者進來，以及把真正的使用者擋在門外。

同人與異人分數，FAR 與 TAR

對分數 s 的門檻 t，

$$\mathrm{FAR}(t) = P(s \ge t \mid \text{impostor}), \qquad \mathrm{FRR}(t) = P(s < t \mid \text{genuine}), \qquad \mathrm{TAR}(t) = 1 - \mathrm{FRR}(t).$$

FAR 是**誤接受率**（false accept rate），FRR 是**誤拒絕率**（false reject rate），TAR 是**真接受率**（true accept rate）。NIST 的評估在一對一比較中使用等價的名稱：**誤匹配率**（false match rate, FMR）與**誤不匹配率**（false non-match rate, FNMR）[2], [3]。這正是 DLCV 02 中的假陽性率與假陰性率；這裡的「陽性」類別是「同一個人」。

兩種分佈的大小差距很大。若有 P 個人、每人 k 張影像，則同人配對有 $P\binom{k}{2}$ 組，異人配對約有 $\binom{Pk}{2} - P\binom{k}{2}$ 組，隨 P 的平方增長。因為異人配對這麼多，工作點會寫成**固定 FAR 下的 TAR**：「TAR@FAR = 10^{-4}」是把門檻設在每一萬組異人配對只接受一組時，被接受的同人配對的比例。IJB-B 與 IJB-C 等基準回報 TAR@FAR = 10^{-4} [16]，NIST 持續進行的一對一評估則在它的某個資料集上，依 FMR = 10^{-6} 時的 FNMR 為演算法排名 [3]。

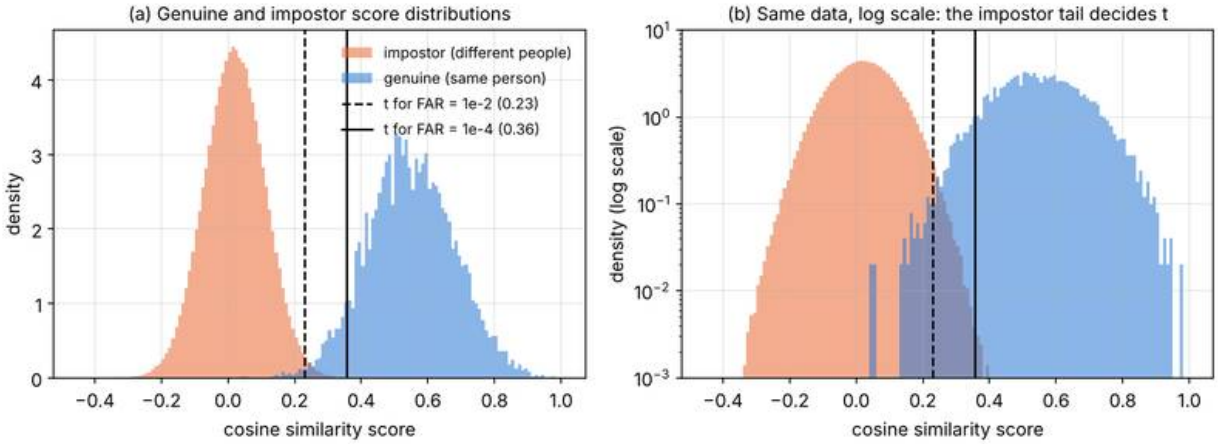


圖 4 — 合成的同人（藍）與異人（橙）分數分佈。(*a*) 線性尺度。(*b*) 對數尺度：小 *FAR* 的門檻由異人分佈的遠端尾巴決定，在線性軸上完全看不到。

```
import numpy as np
from sklearn.metrics import roc_curve

rng = np.random.default_rng(0)
genuine = np.clip(rng.normal(0.55, 0.13, 5_000), -1, 1)       # same-person cosines
impostor = np.clip(rng.normal(0.02, 0.09, 500_000), -1, 1)    # different-person cosines
y = np.r_[np.ones(genuine.size), np.zeros(impostor.size)]
fpr, tpr, thr = roc_curve(y, np.r_[genuine, impostor])

for far in [1e-2, 1e-3, 1e-4]:
    i = np.searchsorted(fpr, far, side="right") - 1          # last point with FPR <= far
    print(f"FAR <= {far:.0e}: threshold {thr[i]:.3f}, TAR {tpr[i]:.3f}, "
          f"false accepts {int(round(fpr[i] * impostor.size))}")
i = np.argmin(np.abs(fpr - (1 - tpr)))
print(f"EER ~ {(fpr[i] + 1 - tpr[i]) / 2:.4f} at threshold {thr[i]:.3f}")
```

```
FAR <= 1e-02: threshold 0.231, TAR 0.994, false accepts 4825
FAR <= 1e-03: threshold 0.300, TAR 0.975, false accepts 495
FAR <= 1e-04: threshold 0.356, TAR 0.936, false accepts 49
EER ~ 0.0072 at threshold 0.241
```

等錯誤率（equal error rate, EER）是 FAR 等於 FRR 的門檻處的錯誤率；它是方便的單一數字，但描述的工作點是安全系統很少會採用的。請注意門檻如何移動：把 FAR 從 10^{-2} 降到 10^{-4}，門檻從 0.231 升到 0.356，並讓 TAR 少了 5.8 個百分點。也請注意最後一行依據的錯誤有多少：只有 49 次誤接受。用 500,000 組異人配對完全無法量測 10^{-6} 的 FAR；你需要數百萬組，而且這些配對必須相當獨立。

僅線上版：量測極小的錯誤率 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/notes/量測極小的錯誤率/*

ROC 與 DET 曲線

DLCV 02 的 ROC 曲線畫的是 TAR 對 FAR。臉部驗證常適合使用**對數 FAR 軸**：例如 FAR 介於 10^{-6} 與 10^{-2} 的操作區域，在線性軸上會被擠到左邊緣。座標尺度是呈現方式，不是 ROC 定義的一部分；零 FAR 也不能畫在對數軸上。**偵測錯誤權衡曲線**（detection error trade-off, DET）[22] 畫出兩種錯誤率，也就是 FRR 對 FAR，兩個軸都用標準常態累積分佈函數的反函數 Φ^{-1} 轉換（「常態離差」尺度）。如果同人與異人分數都是高斯分佈，DET 曲線就是一條直線，越靠左下越好。Martin 等人為語者辨識提出它，並解釋了它為何傾向接近線性 [22]；現在它是生物辨識系統的標準工具。

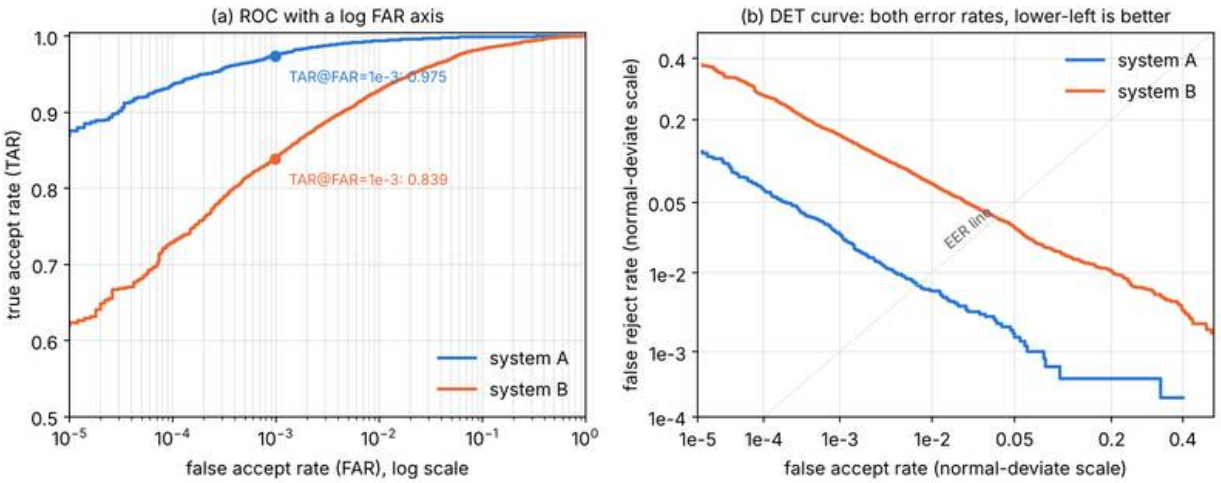


圖 5 — (*a*) 對數 *FAR* 軸的 *ROC*；圓點標示 $FAR = 10^{-3}$ 時的 *TAR*。(*b*) 同樣兩個合成系統的 *DET* 曲線：兩種錯誤率都用常態離差軸。*EER* 在曲線與虛線對角線的交點。

選擇門檻

門檻是系統的一部分，不是網路的一部分，而且必須在能代表部署情境的資料上選擇。從上面的公式可以得到三條規則。(1) 根據應用場景決定你能容忍的 FAR（手機解鎖與鑑識搜尋的代價差很多），再把 t 設成**留出**資料集上異人分數的對應分位數；在測試集上調整 t 會高估結果。(2) 在該門檻下，同時回報 TAR 與 FAR，絕不只報其中一個。(3) 確認你部署環境的異人分數，看起來與校準集相似：新的相機、新的族群或新的影像品質，都會改變異人分佈，因此改變固定 t 下的真實 FAR。最後一點我們會在公平性一節回頭談。

評估識別

白話版。在搜尋中，探測影像要與每一位已註冊者比較。在封閉集合測試中，我們問：正確的人有沒有排第一（或在前 k 名內）？在開放集合測試中，我們還要問：當這個人根本沒有註冊時，系統有沒有保持沉默？

封閉集合：rank-1 與 CMC 曲線

對封閉集合識別，把圖庫依與探測影像的相似度排序。**Rank-1 準確率**是對應者排第一的探測影像比例，而**累積匹配特徵**（cumulative match characteristic, CMC）CMC(k) 是對應者在前 k 名內的比例。公式與程式碼在 DLCV 07；臉部圖庫通常每人只有一個模板，所以很少用 mAP。

關鍵的影響因素是**圖庫大小**。每多一位已註冊者，就多一次冒名者得分超過真正對應者的機會。MegaFace 就是為了衡量這件事而建立：它在圖庫中加入多達一百萬張干擾照片，並回報干擾者數量從 10 增加到 1M 時的識別與驗證結果 [24]。

開放集合：FNIR 與 FPIR

在開放集合搜尋中，只有當候選者的分數達到門檻 t 時，系統才回傳它。兩個錯誤率描述這種情況 [2]：

$$\mathrm{FNIR}(t) = \frac{\#\{\text{有對應者的搜尋，其對應者未以分數} \ge t \text{ 被回傳}\}}{\#\{\text{有對應者的搜尋}\}}, \qquad \mathrm{FPIR}(t) = \frac{\#\{\text{無對應者的搜尋，回傳了任何分數} \ge t \text{ 的候選者}\}}{\#\{\text{無對應者的搜尋}\}}.$$

有對應者（mated）的搜尋是指探測影像的那個人已經註冊；**無對應者**（non-mated）的搜尋是指探測影像的那個人沒有註冊。FNIR（假陰性識別率，false negative identification rate）與 FPIR（假陽性識別率，false positive identification rate）在一對多搜尋中扮演的角色，分別相當於 FRR 與 FAR；在這個簡化版本中，只有第 1 名的候選者才會被回傳。如果一個無對應者探測影像的 N 次圖庫比較，每一次都是誤匹配率為 $\mathrm{FMR}(t)$ 的獨立試驗，則

$$\mathrm{FPIR}(t) \approx 1 - (1 - \mathrm{FMR}(t))^{N} \approx N \cdot \mathrm{FMR}(t) \quad \text{for small } N \cdot \mathrm{FMR}(t),$$

所以在固定門檻下，FPIR 大致與圖庫大小成正比。這就是為什麼一對多搜尋需要比驗證低得多的一對一誤匹配率。

```
import numpy as np
```

```
rng = np.random.default_rng(0)
dim, noise = 128, 1.1
unit = lambda v: v / np.linalg.norm(v, axis=-1, keepdims=True)

def identify(n_gallery, n_probe=300, thr=0.36):
    people = rng.normal(size=(n_gallery + n_probe, dim))
    gallery = unit(people[:n_gallery] + noise * rng.normal(size=(n_gallery, dim)))
    mated = rng.choice(n_gallery, n_probe, replace=False)            # enrolled people
    p_in = unit(people[mated] + noise * rng.normal(size=(n_probe, dim)))
    p_out = unit(people[n_gallery:] + noise * rng.normal(size=(n_probe, dim)))  # not
     ↪  enrolled
    S_in, S_out = p_in @ gallery.T, p_out @ gallery.T
    rank1 = (S_in.argmax(1) == mated).mean()
    # open set: a search "returns" the top candidate only if its score >= thr
    fnir = 1 - ((S_in.argmax(1) == mated) & (S_in.max(1) >= thr)).mean()
    fpir = (S_out.max(1) >= thr).mean()
    return rank1, fnir, fpir

for n in [300, 3_000, 30_000]:
    r1, fnir, fpir = identify(n)
    print(f"gallery {n:6d}: rank-1 {r1:.3f} | threshold 0.36: FNIR {fnir:.3f}, FPIR
     ↪  {fpir:.3f}")
```

```
gallery    300: rank-1 0.990 | threshold 0.36: FNIR 0.090, FPIR 0.010
gallery   3000: rank-1 0.953 | threshold 0.36: FNIR 0.113, FPIR 0.043
gallery  30000: rank-1 0.903 | threshold 0.36: FNIR 0.117, FPIR 0.330
```

在同樣的合成嵌入品質下，圖庫從 300 人增加到 30,000 人，rank-1 從 0.990 降到 0.903；在固定門檻下，未註冊的探測影像觸發誤報的比例，從 1% 升到 33%。圖 6 是類似實驗的 CMC 曲線與 FPIR 對門檻曲線。沒有圖庫大小的識別結果毫無意義，而且圖庫變大時，開放集合系統必須重新設定門檻。

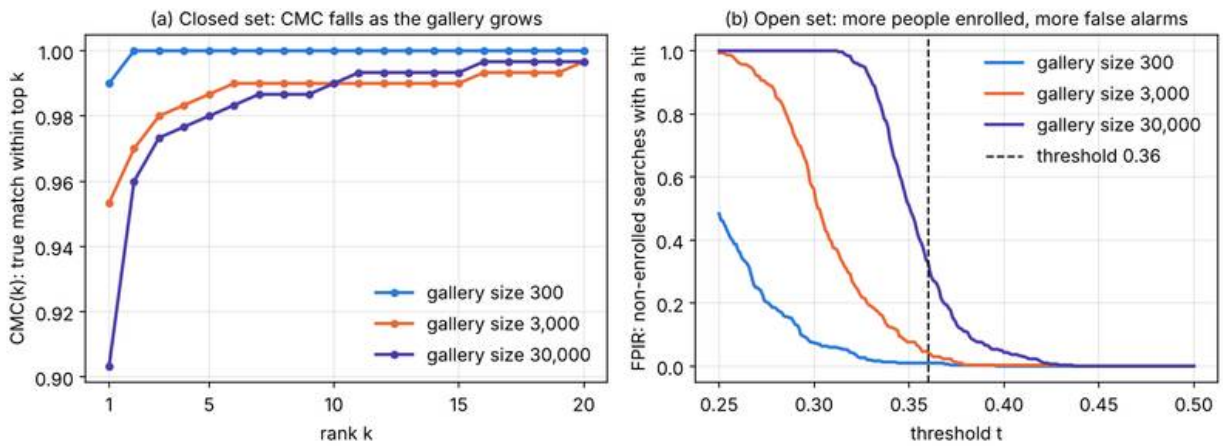


圖 *6* — 一個合成的搜尋實驗。(*a*) 封閉集合的 *CMC* 隨圖庫變大而下降。(*b*) 各門檻下的開放集合 *FPIR*：在固定門檻下，已註冊者越多，未註冊探測影像的誤報就越多。

資料集與基準

白話版。臉部的訓練集是依人分組的大量網路照片清單；測試集則以固定的協定定義配對或搜尋。有幾個最知名的資料集是在未經照片中人物同意的情況下從網路蒐集的，其中有些已經被撤下。

資料集	角色	內容（依其來源的描述）	狀態
LFW [23]	驗證測試	不受限制的新聞照片；約 13K 張照片、5K 個人 [24]	已飽和；2019 年加上免責聲明，警告不可由它推斷商用適用性 [30]
IJB-B [27]	驗證、識別、分群	1,845 位受試者；21,798 張靜態影像，以及來自 7,011 段影片的 55,026 個影格，附人工標註的框與眼睛／鼻子位置	標準測試集；媒體採用創用 CC 授權 [27]
IJB-C [28]	同 IJB-B，規模更大	3,531 位受試者；31.3K 張靜態影像，以及來自 11,779 段影片的 117.5K 個影格 [16]	標準測試集 [16]
MegaFace [24], [25]	大規模識別	超過 690K 個人的 1M 張干擾照片；後來的訓練集有 672,057 個身分的 4.7M 張照片	**已停止營運**；資料不再發布 [25]
MS-Celeb-1M [26]	訓練	網路影像，用於辨識與知識庫連結的一百萬名名人的基準	**2019 年 4 月下架** [30]
WebFace260M / WebFace42M [29]	訓練	4M 個身分／ 260M 張臉（含雜訊），以及清理後的 2M 個身分／ 42M 張臉	作者描述為最大的公開訓練集 [29]
DigiFace-1M [31]	訓練	用電腦圖學管線算繪的合成臉	合成資料，不是真人照片

LFW（Labeled Faces in the Wild）[23] 定義了不受限制的臉部驗證與其標準協定。它的準確率現在已經飽和：最先進的模型在上面超過 99.8% [31]，所以它不再能區分好系統與很好的系統。**IJB-B** 與 **IJB-C**（IARPA Janus Benchmarks）[27], [28] 取代它成為標準的學術測試：它們包含影片影格、大幅度的姿勢變化，以及在 TAR@FAR = 10^{-4} 等與實際營運相關的工作點上的協定，而且評估的是模板（影像集合），不是單張影像。**MegaFace** [24] 讓圖庫大小成為核心問題。**NIST 的評估** [2], [3] 使用大型、非公開的營運資料集，並直接測試送交的演算法，所以它們是部署規模錯誤率的參考。

被撤下的資料集。 MS-Celeb-1M 在 2019 年 4 月被 Microsoft 撤下；Peng、Mathur 與 Narayanan 指出，撤下發生在批評報導之後，許多「名人」其實是相當普通的人，影像是未經同意彙整的，而 Microsoft 給的理由是研究挑戰賽已經結束 [30]。MegaFace 自己的網站表示挑戰賽已經結束，資料不再發布 [25]。同一份研究發現，被撤回的資料常常透過副本、清理過的衍生版本與預訓練模型繼續流傳，而授權條款對限制這類使用幾乎沒有作用 [30]；DLCV 07 中的 DukeMTMC 案例也很類似。實務上：不要使用被撤下的資料集或其衍生物，記錄每個訓練集的來源與授權，並預期審查者會問。

合成訓練資料是一種回應。DigiFace-1M 用電腦圖學管線算繪人臉，動機是網路爬取資料中的偏差、標籤雜訊與缺乏同意 [31]；DCFace 用擴散模型生成身分與風格 [32]。Boutros 等人調查了

這個方向及其尚待解決的挑戰 [33]。

公平性與族群差異

白話版。一個系統可以和另一個系統有相同的整體準確率，卻對某些群體犯下多得多的錯誤。因為部署的系統通常對所有人用同一個門檻，不同群體之間分數分佈的小差異，可能變成錯誤率的大差異。

測量結果顯示什麼

最廣泛的公開測量是 NIST 2019 年關於族群效應的報告 [2]。它把來自四個美國政府資料集的 849 萬人、1,827 萬張影像，交給 99 個開發者所提供、大多為商用的 189 種演算法處理。以下是它的主要發現，依報告自己的說法：

- **假陽性**的差異遠大於假陰性：在不同族群之間，假陽性率常常相差 10 倍到超過 100 倍，而假陰性則更取決於個別演算法，差異通常小於 3 倍。
- 這個模式**取決於演算法與資料**。例如，對高品質的申請照片，許多演算法在西非、東非與東亞人身上的假陽性率最高，但有一些中國開發的演算法，東亞的效應是相反的。假陽性在女性身上也比男性高，在老年人與兒童身上也偏高。
- **影像品質**會影響假陰性：控制良好的拘留照片與品質較低的過境照片，結果不同。
- 有些高度準確的一對多演算法，假陽性差異**測不出來**，所以差異並非不可避免。
- 報告建議，每一項關於族群的主張都要說明指的是哪一種錯誤（假陽性或假陰性）、在固定門檻下、針對各個群體，因為許多先前的報告沒有這樣做 [2]。

學術研究補充了互補的證據。Wang 等人建立了 Racial Faces in-the-Wild（RFW）測試集，並測量商用 API 與研究演算法中的種族偏差 [34]。廣為引用的 Gender Shades 研究 [35] 測得，膚色較深的女性錯誤率高達 34.7%，而膚色較淺的男性最多 0.8%；請注意它評估的是商用的**性別分類**，不是臉部辨識，NIST 明確要求報告者要區分這兩者 [2]。Drozdowski 等人調查了生物辨識領域中偏差的估計與緩解方法 [36]。

玩具範例：一個門檻，兩個群體

這個機制簡單到可以模擬。假設有兩個群體 A 與 B，同人分數分佈相同，但 B 的異人分數稍高：B 內不同的人在網路看來稍微更像。在這個玩具範例中我們直接假設有這個偏移；在真實系統中，必須調查它的成因。在合併的異人分數上，設定一個全域門檻使 FMR = 10^{-3}。

```
import numpy as np

rng = np.random.default_rng(0)
n_imp, n_gen = 400_000, 4_000
# Same genuine distribution; group B's impostor scores sit slightly higher.
imp = {"A": rng.normal(0.00, 0.09, n_imp), "B": rng.normal(0.06, 0.09, n_imp)}
gen = {"A": rng.normal(0.55, 0.13, n_gen), "B": rng.normal(0.55, 0.13, n_gen)}

pooled = np.r_[imp["A"], imp["B"]]
```

```
t_global = np.quantile(pooled, 1 - 1e-3)          # FMR = 0.1% on the pooled data
print(f"global threshold {t_global:.3f}")
for g in "AB":
    print(f"  group {g}: FMR {(imp[g] >= t_global).mean():.5f}, "
          f"FNMR {(gen[g] < t_global).mean():.4f}")
for g in "AB":
    t_g = np.quantile(imp[g], 1 - 1e-3)             # per-group threshold, FMR 0.1% each
    print(f"group {g} own threshold {t_g:.3f}: FNMR {(gen[g] < t_g).mean():.4f}")
```

```
global threshold 0.322
  group A: FMR 0.00018, FNMR 0.0403
  group B: FMR 0.00182, FNMR 0.0400
group A own threshold 0.279: FNMR 0.0210
group B own threshold 0.337: FNMR 0.0517
```

異人分數平均值偏移 0.06（三分之二個標準差），在同一個門檻下就使群體 B 的誤匹配率是群體 A 的 10 倍（0.182% 對 0.018%），雖然兩個群體的誤不匹配率相等。為每個群體選擇各自的門檻可以使 FMR 相等，但這時群體 B 的 FNMR 升到 5.2%，A 的則降到 2.1%：不是這種錯誤率不同，就是那種不同。哪一種取捨可以接受是政策與法律問題，不是純技術問題，而且依群體設定門檻本身也會引發疑慮。技術上的教訓則比較明確：在工作點的門檻下，針對每個群體測量兩種錯誤率，使用跟部署資料相似的資料；並記得整體 FAR 可能掩蓋十倍的差異（圖 7）。

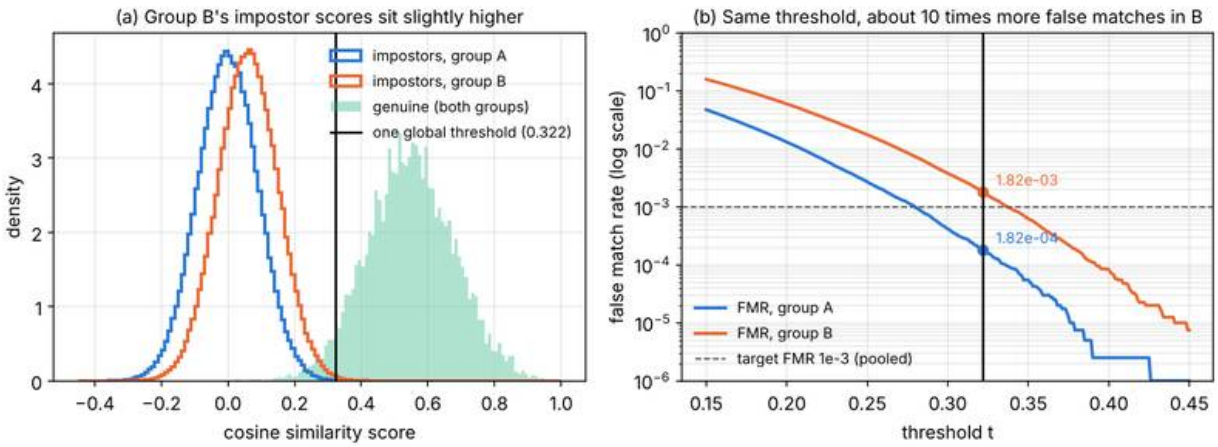


圖 *7* — 一個合成的兩群體範例。(*a*) 群體 *B* 的異人分佈偏移了 *0.06*。(*b*) 在同一個全域門檻下，群體 *B* 的誤匹配率約是群體 *A* 的 *10* 倍，雖然合併後的比率符合 10^{-3} 的目標。

呈現攻擊

白話版。辨識網路回答的是「這是誰？」，不是「這是真實、活著的臉嗎？」。印出來的照片、播放影片的手機螢幕或面具，都可以拿給鏡頭看。偵測這類攻擊是另一個任務。

呈現攻擊（presentation attack）是試圖用人造物欺騙感測器；**呈現攻擊偵測**（presentation attack detection, PAD），也叫臉部反欺騙（face anti-spoofing），判斷輸入是真實的（bona fide，活人）

還是攻擊。Yu 等人調查了它的深度學習方法 [37]：除了二元的「活體對攻擊」標籤之外，許多方法會使用逐像素的輔助監督，例如偽深度圖；這個領域現在強調**領域泛化**（訓練時沒見過的攻擊、相機與光線）與新型態的**開放集合**攻擊；多模態感測器（深度、紅外線）或特殊感測器（光場、閃光）也有幫助。PAD 與辨識並行運作，兩種錯誤分別評估；一個很強的辨識器若沒有 PAD，只要有已註冊者的一張好照片就很容易被騙。像臉部變形融合（morphing）的護照照片與深偽（deepfake）等數位攻擊與此相關，但屬於不同的威脅。

隱私、同意與法律

白話版。臉部嵌入是生物辨識的識別碼：它可以把一個人在網路與各種攝影機中的照片串連起來，而且人不能像換密碼那樣換臉。許多法律體系因此把它視為敏感資料。本節描述背景，不構成法律建議。

兩點歐洲法律的內容，說明了監管者怎麼看臉部辨識（各司法管轄區的規則不同，也會改變）：

- 在歐盟的《一般資料保護規則》（General Data Protection Regulation）下，「為了唯一識別自然人而處理的生物辨識資料」是**敏感**個人資料的類別之一，受到特定的處理條件約束（GDPR 第 9 條）[38]。
- 歐盟《人工智慧法》（AI Act，Regulation (EU) 2024/1689）在其**禁止的做法**中，列出了為建立或擴充臉部辨識資料庫而不分對象地從網際網路或監視錄影蒐集影像，以及在公共場所為執法目的進行即時遠端生物辨識；這些禁令自 2025 年 2 月 2 日起適用 [39]。確切的範圍與條件由法規本身定義。

對工程師來說，實際的影響很具體。要知道每一張訓練影像從哪裡來、依據什麼條款；上面提到被撤下的資料集顯示，「公開可見」不等於「可作任何用途」。把儲存的模板當作敏感資料處理：限制誰能查詢圖庫，記錄搜尋紀錄，並刪除不再需要的模板。應用場景允許時，優先採用驗證用的模板留在使用者裝置上的設計。如 NIST 所建議，記錄預期用途與各群體實測的錯誤率 [2]。合成訓練資料 [31], [33] 能減少但無法消除這些疑慮，因為部署的系統仍然處理真實的臉。

失效模式與實務建議

白話版。大多數失敗來自訓練與測試條件不一致：臉轉向一側、被遮住一半、老了許多、模糊，或來自系統很少見到的族群。在一種情境調好的門檻，換到另一種情境，意義會悄悄改變。

- **姿勢**。大的偏航角（側臉）會破壞五點對齊，並讓一半的臉消失。IJB-B 使用人工標註的臉部框建立，正是因為由一般臉部偵測器篩選的資料集，會低估這類姿勢與遮擋 [27]；ArcFace 的作者也指出，姿勢與年齡差距是普通 softmax 特徵表現下降的條件 [16]。
- **遮擋**。口罩、太陽眼鏡、手與頭髮會遮住臉的一部分；偵測器也許仍會觸發，但特徵點會漂移，嵌入則往異人區域移動。
- **年齡**。多年前註冊的模板比較容易對不上；MegaFace 評估了與年齡有關的表現 [24]，NIST 則發現老年人與兒童的假陽性偏高 [2]。
- **影像品質**。低解析度、動態模糊、壓縮與光線不佳會提高誤不匹配；NIST 把幾項假陰性差異歸因於影像品質，而不是族群 [2]。品質感知的方法（MagFace、AdaFace [19], [20]）以及拍攝時的品質檢查都有幫助。

- **門檻漂移**。在一個資料集上選的門檻，換到另一種相機、族群或圖庫大小，會得到不同的 FAR（圖 6 與圖 7）。要用類似部署環境的異人配對重新校準，並在部署後監控分數分佈，如 ML 11 所述。
- **標籤雜訊與身分重疊**。從網路蒐集的訓練集含有標籤錯誤的影像，以及同一個人用兩個名字出現；sub-center ArcFace [18] 與資料集清理 [29] 針對這個問題。訓練身分與測試身分重疊會高估開放集合的結果，所以在回報前要檢查。
- **評估捷徑**。只回報 LFW 上的準確率、只回報小型封閉圖庫的 rank-1，或只回報一種錯誤率，恰好會隱藏上面這些失敗。

現代觀點

綜述怎麼說

Wang 與 Deng 的綜述 [1] 是深度臉部辨識的標準地圖。它把損失分成**歐氏距離類**（對比、三元組、中心損失）、**角度／餘弦邊界類**（SphereFace、CosFace、ArcFace）與 **softmax 變體**（特徵與權重正規化），並追溯歐氏距離類損失如何主導早期幾年，以及角度／餘弦邊界損失與特徵、權重正規化如何自 2017 年起接手。Drozdowski 等人 [36] 調查了生物辨識中的族群偏差：偏差如何估計、可能如何緩解，以及仍待解決的社會與技術問題。Yu 等人 [37] 調查了深度臉部反欺騙，包括針對領域泛化與開放集合攻擊所設計的方法。Boutros 等人 [33] 調查了臉部辨識用的合成資料，整理其使用情境，並討論尚待解決的挑戰。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/face-metric-learning/papers/現代觀點綜述導讀/*

2023–2026 年的最新進展

- **在超大資料集上使用以邊界為基礎的 softmax 損失，仍是標準做法**。自 ArcFace 以來的主要變化，是連到影像品質的自適應邊界（MagFace [19]、AdaFace [20]）、對標籤雜訊的穩健性（sub-center ArcFace [18]），以及在數百萬個身分上高效率訓練（Partial FC [21]，CNN 與 ViT 骨幹皆適用）。
- **基準轉向低 FAR、以模板為單位與實際營運的評估**。 LFW 已經飽和 [31]；IJB-B/C 在 TAR@FAR = 10^{-4} 成為學術界的慣例 [16]，NIST 的 FRTE（2023 年以前稱為 FRVT 的評估 [3]）則在大型營運資料上測試送交的演算法，誤匹配率低到 10^{-6}。
- **合成與保護隱私的訓練資料**是活躍的方向，有電腦圖學算繪（DigiFace-1M [31]）與擴散生成（DCFace [32]）的資料集，動機是網路資料的同意、偏差與標籤雜訊問題 [33]。
- **公平性要被測量與回報**，而不是假設：NIST 的族群分析 [2] 與 RFW [34] 等專用測試集，讓各群體的錯誤率成為評估的一部分。

尚未解決的問題

1. **對未見過條件的泛化**：極端姿勢、遮擋、非常低的解析度與長時間的年齡差距。
2. **族群差異**：在類似部署環境的資料上，針對每個群體測量兩種錯誤，並在減少差異的同時，不只是把錯誤從一個群體搬到另一個群體。

3. **沒有同意問題的訓練資料**，並且仍能達到網路規模真實資料的準確率 [33]。
4. **訓練時沒見過類型的呈現攻擊與數位攻擊** [37]。
5. **經過校準、具品質感知的分數**，讓門檻在不同相機與族群之間保持意義。
6. **治理**：資料集發布後的管理 [30]，以及各司法管轄區不同的法律限制 [38], [39]。

重點整理

- 嵌入可比較訓練時未見的身分，單靠訓練分類標籤無法做到。
- 選擇指標或門檻前，先說明辨識任務與登錄庫大小。
- 偵測與對齊錯誤會傳遞到嵌入比較。
- 成對損失需要有資訊的樣本；間隔損失則塑造類別中心周圍的分隔。
- 報告驗證 TAR 時註明 FAR 與冒名配對數；搜尋則使用對應的識別指標。
- 依群體與操作條件檢查錯誤，確認資料集可用性，並保護生物特徵樣板。

練習

1. **開放還是封閉？**對下列每個系統，說明它做的是驗證、封閉集合識別還是開放集合識別，以及哪一種錯誤最重要：(a) 手機解鎖，(b) 把一個家庭的相簿分給五位家庭成員，(c) 在體育場入口搜尋 10,000 人的監控名單。

   提示

   (a) 驗證；誤接受是安全失敗，誤拒絕只是不方便（使用者可以再試一次）。(b) 只有在每張照片都是這五人之一時才是封閉集合識別；真實的相簿裡有客人，所以實際上是開放集合。(c) 開放集合識別；經過的人大多不在名單上，所以選定門檻下的 FPIR 乘上經過的人數，就是誤報的數量。

2. **ArcFace 的邊界**。兩個類別中心相距 $\varphi = 60°$。對落在它們之間的弧上的特徵，分別求出正規化 softmax、$m = 0.5$ 弧度的 ArcFace，以及 $m = 4$ 的 SphereFace，類別 1 的損失不再被滿足的、與 W_1 的夾角。這裡哪一種邊界最大？當 $\varphi = 120°$ 時，SphereFace 的邊界會怎樣？

   提示

   Softmax：$30°$。ArcFace：$(\varphi - m)/2 = (60° - 28.6°)/2 \approx 15.7°$，所以間隙是 $28.6°$。SphereFace：$\varphi/(m+1) = 12°$，間隙 $\varphi(m-1)/(m+1) = 36°$。當 $\varphi = 120°$ 時，SphereFace 的間隙加倍成 $72°$，而 ArcFace 的仍是 $28.6°$。

3. **玩具實驗中的 CosFace**。修改玩具實驗，讓目標 logit 為 $s(\cos\theta - m)$，$m = 0.35$（CosFace）。現在 $\partial(\text{目標 logit})/\partial\cos\theta$ 是多少？執行它，並把印出的三個數字與 ArcFace 的比較。

   提示

   在 logits 中把 `np.cos(th + margin)` 換成 `np.cos(th) - margin`，並把梯度因子 `np.sin(th + margin) / np.sin(th)` 換成 1（$\cos\theta - m$ 對 $\cos\theta$ 的導數是 1，所以整個因子就只是 s）。預期結果會接近 ArcFace 的；確切的數字取決於這次執行。

4. **需要多少異人配對？**你想宣稱 FAR = 10^{-5}。(a) 有 300,000 組異人配對且沒有觀察到任何誤接受時，真實 FAR 的 95% 上界大約是多少？(b) 需要多少組異人配對，才能讓零錯誤時的上界降到 10^{-5}？(c) 為什麼你的配對所代表的「獨立試驗」，可能比它們的數量少？

提示

(a) 三法則：約 $3/300{,}000 = 10^{-5}$。(b) 約 300,000，但前提是沒有發生錯誤；若有幾個錯誤，就需要更多。(c) 重複使用同樣影像或同樣人的配對彼此相關：一張不尋常的影像可以同時產生許多誤接受。

5. **圖庫成長**。在識別的程式碼中，圖庫為 300 人時，門檻 0.36 下無對應者探測影像觸發誤報的機率是 0.010。用 FPIR $\approx 1-(1-\text{FMR})^N$ 估計每次比較的 FMR，然後預測圖庫為 3,000 與 30,000 時的 FPIR。與印出的數值比較，並解釋任何差異。

提示

$1-(1-\text{FMR})^{300} = 0.010$ 得到 FMR $\approx 3.4\times10^{-5}$，預測 3,000 時約為 0.096，30,000 時約為 0.64。印出的 0.043 與 0.330 比較低。只有 300 個無對應者探測影像時，FPIR 0.010 只是 3 次誤報，所以 FMR 的估計非常不確定；應該改用大量異人比較直接估計 FMR。

6. **公平的門檻**。在兩群體的程式碼中，找出使兩個群體 FNMR 相等的單一門檻，以及使 FMR 相等的門檻。一個門檻能同時做到兩者嗎？你會向決策者報告什麼？

提示

因為同人分佈相同，任何單一門檻都會給出相等的 FNMR（不計取樣雜訊：印出的執行中是 0.0403 對 0.0400）；沒有任何單一門檻能使 FMR 相等，因為異人分佈不同。要報告在提議的門檻下兩個群體的兩種錯誤率，以及各種替代方案（依群體設門檻、更好的模型），並把取捨的選擇留給對此負責的人。

參考文獻

1. M. Wang and W. Deng, "Deep face recognition: A survey," *Neurocomputing*, vol. 429, pp. 215–244, 2021. doi
2. P. Grother, M. Ngan and K. Hanaoka, "Face Recognition Vendor Test (FRVT) Part 3: Demographic Effects," NIST Interagency Report 8280, National Institute of Standards and Technology, 2019. doi
3. National Institute of Standards and Technology, "Face Recognition Technology Evaluation (FRTE) 1:1 Verification," evaluation web page, accessed 2026. NIST
4. K. Zhang, Z. Zhang, Z. Li and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016. doi
5. J. Deng, J. Guo, E. Ververas, I. Kotsia and S. Zafeiriou, "RetinaFace: Single-shot multi-level face localisation in the wild," in *Proc. IEEE/CVF CVPR*, 2020. CVF
6. Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proc. IEEE CVPR*, 2014. CVF

7. F. Schroff, D. Kalenichenko and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” in *Proc. IEEE CVPR*, 2015. CVF
8. S. Chopra, R. Hadsell and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *Proc. IEEE CVPR*, vol. 1, pp. 539–546, 2005. doi
9. R. Hadsell, S. Chopra and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” in *Proc. IEEE CVPR*, vol. 2, pp. 1735–1742, 2006. doi
10. A. Hermans, L. Beyer and B. Leibe, “In defense of the triplet loss for person re-identification,” arXiv:1703.07737, 2017. arXiv
11. F. Wang, X. Xiang, J. Cheng and A. L. Yuille, “NormFace: L2 hypersphere embedding for face verification,” in *Proc. ACM International Conference on Multimedia*, pp. 1041–1049, 2017. doi
12. W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj and L. Song, “SphereFace: Deep hypersphere embedding for face recognition,” in *Proc. IEEE CVPR*, 2017. CVF
13. H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li and W. Liu, “CosFace: Large margin cosine loss for deep face recognition,” in *Proc. IEEE/CVF CVPR*, 2018. CVF
14. F. Wang, J. Cheng, W. Liu and H. Liu, “Additive margin softmax for face verification,” *IEEE Signal Processing Letters*, vol. 25, no. 7, pp. 926–930, 2018. doi
15. J. Deng, J. Guo, N. Xue and S. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” in *Proc. IEEE/CVF CVPR*, 2019. CVF
16. J. Deng, J. Guo, J. Yang, N. Xue, I. Kotsia and S. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 5962–5979, 2022. doi
17. Y. Wen, K. Zhang, Z. Li and Y. Qiao, “A discriminative feature learning approach for deep face recognition,” in *Proc. ECCV*, pp. 499–515, 2016. doi
18. J. Deng, J. Guo, T. Liu, M. Gong and S. Zafeiriou, “Sub-center ArcFace: Boosting face recognition by large-scale noisy web faces,” in *Proc. ECCV*, pp. 741–757, 2020. doi
19. Q. Meng, S. Zhao, Z. Huang and F. Zhou, “MagFace: A universal representation for face recognition and quality assessment,” in *Proc. IEEE/CVF CVPR*, 2021. CVF
20. M. Kim, A. K. Jain and X. Liu, “AdaFace: Quality adaptive margin for face recognition,” in *Proc. IEEE/CVF CVPR*, 2022. CVF
21. X. An, J. Deng, J. Guo, Z. Feng, X. Zhu, J. Yang and T. Liu, “Killing two birds with one stone: Efficient and robust training of face recognition CNNs by Partial FC,” in *Proc. IEEE/CVF CVPR*, 2022. CVF
22. A. Martin, G. Doddington, T. Kamm, M. Ordowski and M. Przybocki, “The DET curve in assessment of detection task performance,” in *Proc. Eurospeech*, pp. 1895–1898, 1997. ISCA
23. G. B. Huang, M. Mattar, T. Berg and E. Learned-Miller, “Labeled Faces in the Wild: A database for studying face recognition in unconstrained environments,” in *Workshop on Faces in ‘Real-Life’ Images: Detection, Alignment, and Recognition*, Marseille, 2008. HAL
24. I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller and E. Brossard, “The MegaFace benchmark: 1 million faces for recognition at scale,” in *Proc. IEEE CVPR*, 2016. CVF
25. University of Washington, “MegaFace and MF2: Million-scale face recognition,” project web page, accessed 2026. project page
26. Y. Guo, L. Zhang, Y. Hu, X. He and J. Gao, “MS-Celeb-1M: A dataset and benchmark for large-scale face recognition,” in *Proc. ECCV*, pp. 87–102, 2016. doi
27. C. Whitelam, E. Taborsky, A. Blanton, B. Maze, J. Adams, T. Miller, N. Kalka, A. K. Jain, J. A.

Duncan, K. Allen, J. Cheney and P. Grother, “IARPA Janus Benchmark-B face dataset,” in *Proc. IEEE CVPR Workshops*, 2017. CVF

28. B. Maze, J. Adams, J. A. Duncan, N. Kalka, T. Miller, C. Otto, A. K. Jain, W. T. Niggel, J. Anderson, J. Cheney and P. Grother, “IARPA Janus Benchmark – C: Face dataset and protocol,” in *Proc. International Conference on Biometrics (ICB)*, pp. 158–165, 2018. doi
29. Z. Zhu, G. Huang, J. Deng, Y. Ye, J. Huang, X. Chen, J. Zhu, T. Yang, J. Lu, D. Du and J. Zhou, “WebFace260M: A benchmark unveiling the power of million-scale deep face recognition,” in *Proc. IEEE/CVF CVPR*, 2021. CVF
30. K. Peng, A. Mathur and A. Narayanan, “Mitigating dataset harms requires stewardship: Lessons from 1000 papers,” in *Proc. NeurIPS Datasets and Benchmarks Track*, 2021. proceedings
31. G. Bae, M. de La Gorce, T. Baltrušaitis, C. Hewitt, D. Chen, J. Valentin, R. Cipolla and J. Shen, “DigiFace-1M: 1 million digital face images for face recognition,” in *Proc. IEEE/CVF WACV*, 2023. CVF
32. M. Kim, F. Liu, A. Jain and X. Liu, “DCFace: Synthetic face generation with dual condition diffusion model,” in *Proc. IEEE/CVF CVPR*, 2023. CVF
33. F. Boutros, V. Struc, J. Fierrez and N. Damer, “Synthetic data for face recognition: Current state and future prospects,” *Image and Vision Computing*, vol. 135, art. 104688, 2023. doi
34. M. Wang, W. Deng, J. Hu, X. Tao and Y. Huang, “Racial Faces in the Wild: Reducing racial bias by information maximization adaptation network,” in *Proc. IEEE/CVF ICCV*, pp. 692–702, 2019. doi
35. J. Buolamwini and T. Gebru, “Gender Shades: Intersectional accuracy disparities in commercial gender classification,” in *Proc. Conference on Fairness, Accountability and Transparency*, PMLR 81, pp. 77–91, 2018. PMLR
36. P. Drozdowski, C. Rathgeb, A. Dantcheva, N. Damer and C. Busch, “Demographic bias in biometrics: A survey on an emerging challenge,” *IEEE Transactions on Technology and Society*, vol. 1, no. 2, pp. 89–103, 2020. doi
37. Z. Yu, Y. Qin, X. Li, C. Zhao, Z. Lei and G. Zhao, “Deep learning for face anti-spoofing: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, pp. 5609–5631, 2023. doi
38. European Commission, “What personal data is considered sensitive?,” data protection guidance referring to Article 9 of the GDPR, accessed 2026. European Commission
39. European Commission, “AI Act” (Regulation (EU) 2024/1689), policy page, accessed 2026. European Commission
40. M. Turk and A. Pentland, “Eigenfaces for recognition,” *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991. doi
41. P. N. Belhumeur, J. P. Hespanha and D. J. Kriegman, “Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 711–720, 1997. doi

DLCV 03・物件偵測：從 R-CNN 到 DETR

深度網路如何找出並標記影像中的每一個物件：邊界框迴歸、IoU、錨框、NMS 與特徵金字塔；二階段、一階段、無錨框與 Transformer 偵測器；開放詞彙模型；以及偵測該如何評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-detection/

你將學到

- 定義偵測輸出，並區分方框與像素遮罩。
- 說明方框迴歸、IoU、指派、抑制與特徵金字塔。
- 比較候選區域、密集、無錨框與集合預測設計。
- 理解文字提示如何改變類別詞彙。
- 解讀 AP 協定，並量測完整部署流程。

先看全貌

分類器只回答一個關於整張影像的問題：**這是什麼？**偵測器要回答更難的問題：**什麼東西在哪裡？**它必須找出每一個感興趣的物件，在每個物件外面畫一個緊貼的矩形（**邊界框**，bounding box），寫上名稱，並說明自己有多確定。物件的數量事先不知道：可能是零個、一個，也可能是三百個。

> **用最簡單的話說**
>
> 想像一頁「威利在哪裡？」。**分類器**只會告訴你：「這一頁裡有威利。」**偵測器**則會在威利身上貼一張貼紙，再在每一隻狗、每一顆氣球上各貼一張，並在每張貼紙上寫「這是什麼、有多確定」（「狗，蠻確定的」）。難的地方跟你自己找的時候一樣：遠處的小東西、躲在別的東西後面的東西，還有不要在同一隻狗身上貼兩張貼紙。

偵測是多數「真實世界」視覺系統的入口。自駕車偵測行人與車輛；手機在對焦之前先偵測人臉；工廠偵測瑕疵；而追蹤器（見文末連結的追蹤章節）大多是在**把偵測結果隨時間串起來**。這個領域也產生了許多如今遍布深度學習的想法：多任務損失函數、特徵金字塔、焦點損失，以及以二元匹配做集合預測。

本篇假設你已經熟悉 CNN 與影像分類（DLCV 02），以及IP 12的傳統特徵概念。有兩篇綜述可以框住整個故事：Zou 等人的《Object Detection in 20 Years》[1]，以及 Liu 等人發表於 IJCV 的深度通用物件偵測綜述 [2]。我們會在「現代觀點」回到這兩篇。

相關篇章。 DLCV 03A 深入介紹 YOLO 系列與即時 DETR；DLCV 04 則從方框走到像素遮罩（語意、實例與全景分割）。

偵測問題的定義

白話地說：輸入是一張影像，輸出是一小串（邊界框、類別、分數）三元組。訓練資料則是人工畫好、標上類別名稱的邊界框。

精確地說，給定影像 $I \in \mathbb{R}^{H\times W\times 3}$，偵測器輸出一個集合

$$\hat{Y} = \{(\hat{b}_i, \hat{c}_i, \hat{s}_i)\}_{i=1}^{N},$$

其中 $\hat{b}_i \in \mathbb{R}^4$ 是一個邊界框，$\hat{c}_i \in \{1,\dots,K\}$ 是 K 個類別之一，$\hat{s}_i \in [0,1]$ 是信心分數，而 N 隨影像不同而改變。真值（ground truth）是 M 個標註物件構成的集合 $Y = \{(b_j, c_j)\}_{j=1}^{M}$。常見的邊界框格式有兩種：角點 (x_1, y_1, x_2, y_2) 與中心–尺寸 (c_x, c_y, w, h)。邊界框是**與座標軸對齊**的；旋轉框是另一種變體，用於空拍影像與文字偵測。

有三個性質讓偵測比分類困難：

1. **輸出大小不固定**。網路必須決定要輸出**幾個**東西。本篇的每一種架構，本質上都是對這個問題的不同回答。
2. **同時定位與辨識**。模型對物件外觀的變化要具有不變性，對物件的**位置**卻要很敏感；這股張力形塑了每一種偵測頭（head）的設計。
3. **極度不平衡**。一張典型影像只有少數幾個物件，卻有上萬個候選位置與尺寸，幾乎全是背景。

偵測與相鄰任務的比較

圖 1 在同一個合成場景上比較四種經典辨識任務。**分類**每張影像給一個標籤。**偵測**每個物件給一個邊界框與一個標籤。**語意分割**為每個像素標上類別，因此兩顆相碰的球會融成一團。**實例分割**為每個物件給出各自的像素遮罩，也就是「偵測＋每個框一張遮罩」；稍後會看到，Mask R-CNN [16] 是取得它的標準方法。

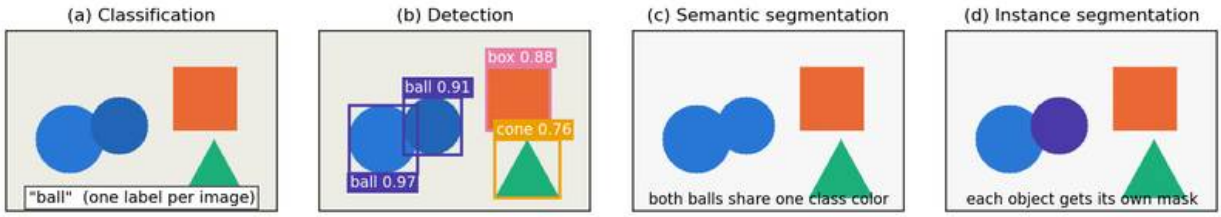


圖 *1* — 同一個場景在四種任務下的樣子：(*a*) 分類、(*b*) 偵測、(*c*) 語意分割、(*d*) 實例分割。

傳統偵測：滑動視窗與手工特徵

白話地說：滑動視窗偵測在各位置與尺度套用分類器，再保留預測含有物件的視窗。

滑動視窗。在縮放比例為 λ 的影像金字塔上，以步幅 s 滑動固定大小 $w \times h$ 的視窗，視窗數量大約是

$$N_{\text{win}} \approx \sum_{k \ge 0} \frac{(H\lambda^{-k} - h)(W\lambda^{-k} - w)}{s^2},$$

其中 k 是金字塔的層索引。對一張 640×480 的影像、64×128 的視窗、$s = 8$、$\lambda = 1.2$，單一長寬比就有約 10^4 個視窗，每一個都得分類。這就是IP 13的樣板比對（template matching）放大版，也有同樣的弱點：只有一種樣板形狀、要搜尋很多尺度，而且大量運算浪費在背景上。

三個傳統偵測器說明了手工特徵能走多遠。**Viola–Jones（2001）** [5] 讓人臉偵測達到即時：簡單的矩形特徵能在常數時間內算出，一連串越來越嚴格的分類器只用幾次便宜的測試，就丟掉大多數背景視窗。**HOG ＋線性 SVM（2005）** [6] 以梯度方向的直方圖描述每個視窗，讓行人偵測變得實用。**可變形部件模型**（deformable part models, DPM）[7] 在粗略的整體物件濾波器周圍加上幾個可移動的部件濾波器，一直稱霸 PASCAL VOC 基準 [3]，直到 2012 年。

僅線上版：傳統偵測器細說：*Viola–Jones、HOG 與 DPM* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-detection/notes/傳統偵測器細說 violajoneshog-與-dpm/*

這個時代為何結束。傳統偵測器**慢**（要在位置、尺度、長寬比上窮舉掃描，每個視窗都經過手工設計的流程評分），而且**脆弱**（只有一種手工特徵、少數幾個固定樣板，換到新類別、新視角或雜亂場景就表現不佳），也無法在類別之間共享特徵。深度偵測器保留了有用的想法，例如在特徵金字塔中掃描、串接、部件與困難負樣本挖掘（hard-negative mining），並把手工特徵換成學到的特徵。

核心基本元件

接下來的每一種偵測器都由同一組零件組成。我們仔細定義每一個，並附上可執行的程式碼。

邊界框參數化與迴歸目標

白話地說：以參考為基礎的方框迴歸，預測候選區域、錨框或參考點的修正量。

給定中心–尺寸形式的參考框 $P = (P_x, P_y, P_w, P_h)$ 與其匹配的真值框 $G = (G_x, G_y, G_w, G_h)$，R-CNN [8] 提出的標準目標為

$$t_x = \frac{G_x - P_x}{P_w}, \qquad t_y = \frac{G_y - P_y}{P_h}, \qquad t_w = \log \frac{G_w}{P_w}, \qquad t_h = \log \frac{G_h}{P_h}.$$

把位移除以參考框尺寸，讓目標具有**尺度不變性**（10 個像素的誤差，對 20 像素的框比對 400 像素的框嚴重得多）。寬高取**對數**，讓縮放變得對稱（放大兩倍與縮小一半分別是 $+\log 2$ 與 $-\log 2$），並保證解碼後的尺寸為正：$\hat{G}_w = P_w \exp(\hat{t}_w)$。實務上，目標常再除以固定的標準差，例如 $(0.1, 0.1, 0.2, 0.2)$，讓四個量的大小相近。

```
import numpy as np

def to_cxcywh(b):
    return np.stack([(b[:, 0] + b[:, 2]) / 2, (b[:, 1] + b[:, 3]) / 2,
                     b[:, 2] - b[:, 0], b[:, 3] - b[:, 1]], axis=1)
```

```
def to_xyxy(c):
    return np.stack([c[:, 0] - c[:, 2] / 2, c[:, 1] - c[:, 3] / 2,
                     c[:, 0] + c[:, 2] / 2, c[:, 1] + c[:, 3] / 2], axis=1)

def encode(anchors, gt, std=(0.1, 0.1, 0.2, 0.2)):
    """Regression targets (tx, ty, tw, th) of gt relative to anchors (both xyxy)."""
    P, G = to_cxcywh(anchors), to_cxcywh(gt)
    t = np.stack([(G[:, 0] - P[:, 0]) / P[:, 2],
                  (G[:, 1] - P[:, 1]) / P[:, 3],
                  np.log(G[:, 2] / P[:, 2]),
                  np.log(G[:, 3] / P[:, 3])], axis=1)
    return t / np.array(std)          # normalize so all four have similar scale

def decode(anchors, t, std=(0.1, 0.1, 0.2, 0.2)):
    P = to_cxcywh(anchors)
    t = t * np.array(std)
    c = np.stack([P[:, 0] + t[:, 0] * P[:, 2],
                  P[:, 1] + t[:, 1] * P[:, 3],
                  P[:, 2] * np.exp(t[:, 2]),
                  P[:, 3] * np.exp(t[:, 3])], axis=1)
    return to_xyxy(c)

anchors = np.array([[0, 0, 32, 32], [100, 100, 228, 164]], float)
gt      = np.array([[4, 2, 40, 30], [90, 110, 250, 170]], float)
t = encode(anchors, gt)
print(np.round(t, 3))
print(np.allclose(decode(anchors, t), gt))   # round trip
```

```
[[ 1.875  0.     0.589 -0.668]
 [ 0.469  1.25   1.116 -0.323]]
True
```

這些目標通常以 Fast R-CNN [9] 的 **smooth L1**（Huber）損失函數訓練：

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5\,x^2 & \text{if } |x| < 1, \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

逐一套用在每個座標差 $x = \hat{t} - t$ 上。誤差小時它像 L2（梯度平滑），誤差大時像 L1（在訓練初期對離群值穩健）。它的弱點是把四個座標當成彼此獨立的數字，而評估指標 IoU 卻把邊界框視為一個整體。這個落差催生了以 IoU 為基礎的損失函數。

IoU 及其變體

白話地說：IoU 問的是，在兩個框各自覆蓋的總面積中，有多少比例是兩者都覆蓋到的？完全吻合時為 1，兩框不相碰時為 0。

對兩個框 A 與 B，

$$\text{IoU}(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|},$$

其中 $|\cdot|$ 代表面積。IoU（交並比，intersection over union）具有尺度不變性，是評估的標尺（若一個偵測結果與某個尚未被匹配的真值框的 IoU 超過門檻，就算正確），也用於標籤指派與 NMS。圖 2 顯示這些門檻有多嚴格：把正方形沿兩軸各平移邊長的約 18%，IoU 就掉到 0.5。

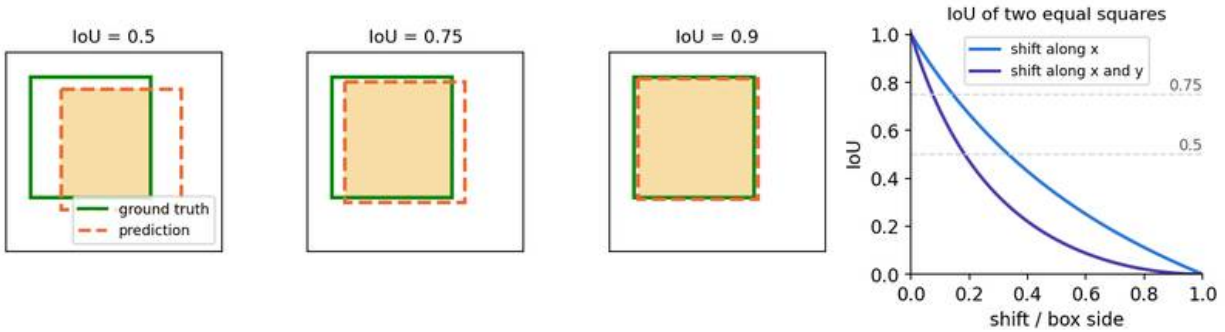


圖 2 — *IoU* 門檻長什麼樣子。左：*IoU* 為 *0.5*、*0.75*、*0.9* 的預測框與真值框。右：兩個相同正方形之一沿 x 軸、或沿 x 與 y 軸平移時的 *IoU*。

拿 $1-\text{IoU}$ 當損失函數有個缺陷：兩框不重疊時，不論相距多遠 IoU 都是 0，梯度也就是 0。常用的修正有三種：

- **GIoU** [11] 加入一個以同時包住兩框的最小框 C 為基礎的懲罰項：$\text{GIoU} = \text{IoU} - \dfrac{|C \setminus (A \cup B)|}{|C|}$，值域為 $(-1, 1]$，兩個不重疊的框離得越遠，它就繼續下降。
- **DIoU** [12] 直接懲罰中心點距離：$\text{DIoU} = \text{IoU} - \dfrac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}$，其中 ρ 是兩框中心 $\mathbf{b}$ 與 $\mathbf{b}^{gt}$ 的歐氏距離，c 是 C 的對角線長度。它比 GIoU 收斂得更快，因為即使一個框包住另一個框（此時 GIoU 的懲罰項消失），它仍會把中心點拉近。
- **CIoU** [12] 再加入長寬比項：$\text{CIoU} = \text{DIoU} - \alpha v$，其中 $v = \dfrac{4}{\pi^2}\left(\arctan\dfrac{w^{gt}}{h^{gt}} - \arctan\dfrac{w}{h}\right)^2$ 衡量長寬比的差異，$\alpha = \dfrac{v}{(1-\text{IoU})+v}$ 是權衡權重。

對應的損失函數分別是 $1-\text{GIoU}$、$1-\text{DIoU}$ 與 $1-\text{CIoU}$。

```
import numpy as np

def iou_family(a, b, eps=1e-9):
    """a, b: (N, 4) boxes as (x1, y1, x2, y2). Returns IoU, GIoU, DIoU, CIoU."""
    ix1, iy1 = np.maximum(a[:, 0], b[:, 0]), np.maximum(a[:, 1], b[:, 1])
    ix2, iy2 = np.minimum(a[:, 2], b[:, 2]), np.minimum(a[:, 3], b[:, 3])
    inter = np.clip(ix2 - ix1, 0, None) * np.clip(iy2 - iy1, 0, None)
    wa, ha = a[:, 2] - a[:, 0], a[:, 3] - a[:, 1]
    wb, hb = b[:, 2] - b[:, 0], b[:, 3] - b[:, 1]
    union = wa * ha + wb * hb - inter
    iou = inter / (union + eps)
    # smallest enclosing box C
    cx1, cy1 = np.minimum(a[:, 0], b[:, 0]), np.minimum(a[:, 1], b[:, 1])
    cx2, cy2 = np.maximum(a[:, 2], b[:, 2]), np.maximum(a[:, 3], b[:, 3])
    area_c = (cx2 - cx1) * (cy2 - cy1)
    giou = iou - (area_c - union) / (area_c + eps)
    # center distance over enclosing-box diagonal
```

```
    rho2 = ((a[:, 0] + a[:, 2]) - (b[:, 0] + b[:, 2])) ** 2 / 4 \
         + ((a[:, 1] + a[:, 3]) - (b[:, 1] + b[:, 3])) ** 2 / 4
    c2 = (cx2 - cx1) ** 2 + (cy2 - cy1) ** 2
    diou = iou - rho2 / (c2 + eps)
    v = 4 / np.pi**2 * (np.arctan(wb / hb) - np.arctan(wa / ha)) ** 2
    alpha = v / (1 - iou + v + eps)
    ciou = diou - alpha * v
    return iou, giou, diou, ciou

gt = np.array([[10, 10, 50, 50]] * 3, float)
pred = np.array([[20, 20, 60, 60],    # overlapping, shifted
                 [60, 10, 100, 50],   # disjoint, nearby
                 [200, 10, 240, 50]], float)  # disjoint, far away
for name, v in zip(["IoU", "GIoU", "DIoU", "CIoU"], iou_family(pred, gt)):
    print(f"{name:5s}", np.round(v, 3))
```

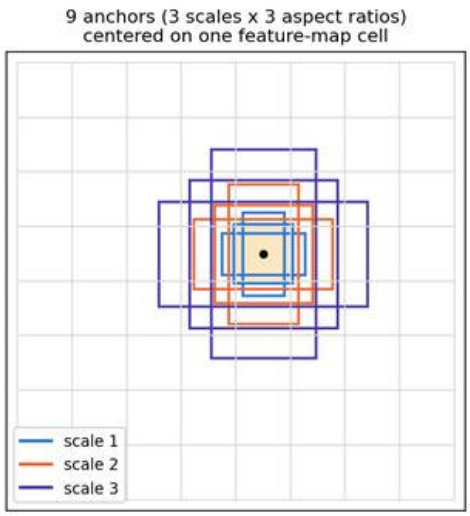
```
IoU   [0.391 0.    0.   ]
GIoU  [ 0.311 -0.111 -0.652]
DIoU  [ 0.351 -0.258 -0.662]
CIoU  [ 0.351 -0.258 -0.662]
```

兩個不重疊的預測框 IoU 都是 0，但 GIoU 與 DIoU 正確地指出比較遠的那個比較差。這裡 CIoU 等於 DIoU，因為所有框都是正方形，沒有長寬比差異。

錨框與標籤指派

白話地說：錨框式密集偵測器在特徵格點放置參考框，再於各位置預測物件分數與方框修正量。

在步幅為 s 的特徵圖上（每個格子涵蓋輸入影像的 $s \times s$ 像素），錨框的中心位於 $((i+\tfrac{1}{2})s, (j+\tfrac{1}{2})s)$，並有幾種尺度 σ 與長寬比 $r = h/w$，寬為 $w = \sigma/\sqrt{r}$、高為 $h = \sigma\sqrt{r}$，因此面積維持 σ^2。Faster R-CNN 在每個位置使用 3 種尺度 × 3 種長寬比 = 9 個錨框 [10]（圖 3 左）。



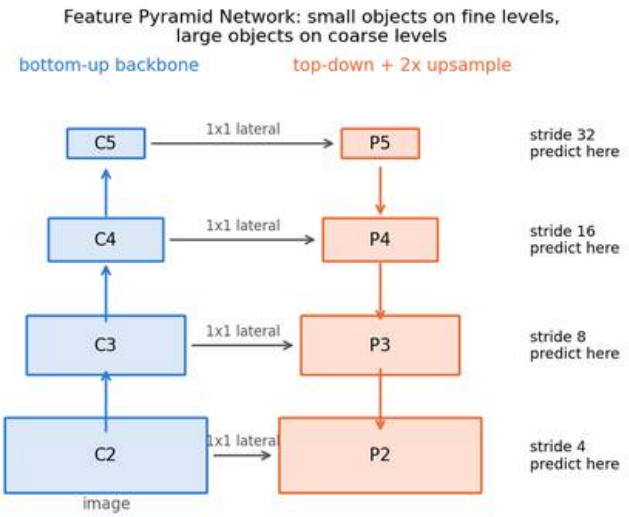


圖 3 — 左：特徵圖某一格上的九個錨框（3 種尺度 × 3 種長寬比）。右：特徵金字塔網路；每一層負責預測對應尺寸範圍的物件。

標籤指派（label assignment）決定哪個錨框要學習哪個物件。經典規則以 IoU 為準：在 Faster R-CNN 的區域提議網路中，若某錨框與某真值框的 IoU 至少 0.7（或它是該真值框最匹配的錨框），就是**正樣本**；若它與所有真值框的 IoU 都低於 0.3，就是**負樣本**；其餘忽略 [10]。RetinaNet 則使用 0.5 與 0.4 [15]。每個正樣本錨框以上述目標朝其匹配的框迴歸。後來發現，指派方式和架構一樣重要：無錨框偵測器（FCOS [29]）改用**位置**與物件尺寸來指派，而 DETR [30] 則以學習出的一對一匹配取代所有固定規則。

錨框也有實際成本：它帶來必須針對每個資料集調整的超參數（尺度、長寬比、門檻），也讓預測數量倍增。YOLO9000 [19] 用 $1 - \text{IoU}$ 當距離，對訓練框做 k-means 分群來決定錨框形狀。

僅線上版：以 *k-means* 決定錨框形狀：用 *IoU* 當距離 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-detection/notes/以-k-means-決定錨框形狀用-iou-當距離/*

非極大值抑制（NMS）與 Soft-NMS

白話地說：密集偵測器會在同一個物件上反應很多次（相鄰的錨框都看得到它）。**非極大值抑制**（non-maximum suppression, NMS）留下最有信心的框，刪掉與它重疊太多的其他框。

貪婪 NMS（逐類別進行）：依分數排序；反覆取出剩下分數最高的框 M，保留它，並移除所有 $\text{IoU}(M, b_i) > N_t$ 的框 b_i（N_t 通常為 0.5 到 0.7）。它的弱點在擁擠場景：兩個站得很近的真人可能重疊超過 N_t，分數較低的那個就被刪掉，召回率因而下降。

Soft-NMS [13] 把「刪除」改成「降低分數」。以高斯版本為例，

$$s_i \leftarrow s_i \exp\!\left(-\frac{\text{IoU}(M, b_i)^2}{\sigma}\right),$$

其中 s_i 是框 b_i 的分數，σ 控制重疊框被懲罰的速度。重疊很多時分數仍會大幅下降，但真正相鄰的物件會以較低的分數留下來，而不是直接消失。

```
import numpy as np

def iou_one_to_many(box, boxes):
    x1 = np.maximum(box[0], boxes[:, 0]); y1 = np.maximum(box[1], boxes[:, 1])
    x2 = np.minimum(box[2], boxes[:, 2]); y2 = np.minimum(box[3], boxes[:, 3])
    inter = np.clip(x2 - x1, 0, None) * np.clip(y2 - y1, 0, None)
    area = lambda b: (b[..., 2] - b[..., 0]) * (b[..., 3] - b[..., 1])
    return inter / (area(box) + area(boxes) - inter)

def nms(boxes, scores, thr=0.5):
    order = np.argsort(-scores)
    keep = []
    while order.size:
        i = order[0]
        keep.append(i)
        ious = iou_one_to_many(boxes[i], boxes[order[1:]])
        order = order[1:][ious <= thr]          # drop everything that overlaps too much
    return np.array(keep)
```

```
def soft_nms(boxes, scores, sigma=0.5, score_thr=0.001):
    scores = scores.copy()
    idx = np.arange(len(scores))
    keep = []
    while idx.size:
        j = np.argmax(scores[idx]); i = idx[j]
        keep.append(i)
        idx = np.delete(idx, j)
        ious = iou_one_to_many(boxes[i], boxes[idx])
        scores[idx] *= np.exp(-ious**2 / sigma)  # decay instead of delete
        idx = idx[scores[idx] > score_thr]
    return np.array(keep), scores

# two people standing close together + duplicates
boxes = np.array([[10, 10, 60, 110], [14, 12, 62, 112], [8, 6, 58, 104],
                  [40, 15, 90, 115], [43, 18, 92, 118]], float)
scores = np.array([0.95, 0.90, 0.80, 0.85, 0.70])
print("greedy NMS keeps:", nms(boxes, scores, 0.5))
keep, s = soft_nms(boxes, scores)
print("Soft-NMS order:", keep, "rescored:", np.round(s[keep], 3))
```

```
greedy NMS keeps: [0 3]
Soft-NMS order: [0 3 2 4 1] rescored: [0.95  0.761 0.183 0.145 0.058]
```

貪婪 NMS 為每個人留下一個框。Soft-NMS 保留所有框，但把重複的框排在兩個真實偵測結果之後很遠的位置；之後的分數門檻或 AP 計算會把它們當成低信心的猜測。圖 4 在合成場景上呈現同樣的效果。

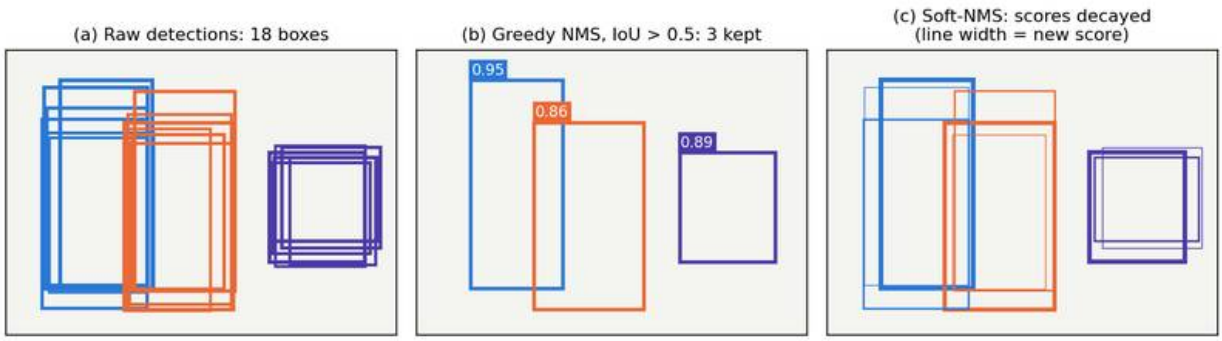


圖 *4* — 在合成偵測結果上做 *NMS*。(*a*) 三個物件周圍的 *18* 個原始框。(*b*) *IoU* 門檻 *0.5* 的貪婪 *NMS* 為每個物件留下一個框。(*c*) *Soft-NMS* 保留所有框，但降低重疊框的分數（線越細，分數越低）。

NMS 還有三個問題：它有一個要手動調整的門檻；它是循序、依資料而定的步驟，不容易在加速器上執行；而且它不屬於訓練的一部分，所以網路從來沒被教過要避免重複。去掉 NMS 是 DETR 與 YOLOv10 [24] 的主要動機之一。

特徵金字塔（FPN）

白話地說：小物件需要高解析度的特徵圖；大物件需要看過大量上下文的特徵。特徵金字塔讓每個尺度都同時擁有兩者。

CNN 骨幹網路本來就會產生步幅為 $4, 8, 16, 32$ 的金字塔 $\{C_2, C_3, C_4, C_5\}$，但淺層的語意較弱。**特徵金字塔網路**（Feature Pyramid Network, FPN）[14] 加上一條由上而下的路徑：從最粗的一層開始，上採樣 2 倍，再加上經 1×1 投影的骨幹橫向連接，

$$P_l = \mathrm{Conv}_{3\times3}\big(\mathrm{Conv}_{1\times1}(C_l) + \mathrm{Up}_{2\times}(P_{l+1})\big),$$

因此每個輸出層 P_l 的通道數相同，語意也都很強。在搭配 FPN 的二階段偵測器中，大小為 $w\times h$ 的感興趣區域從第 $l = \lfloor l_0 + \log_2(\sqrt{wh}/224)\rfloor$ 層擷取特徵（[14] 中 $l_0 = 4$），所以區域大一倍就往上移一層；密集偵測器也使用類似的「尺寸對應層級」規則。後來的頸部（neck）設計（額外的由下而上路徑、加權融合）都在精煉同一個想法；2017 年後幾乎每個偵測器都用上了它（圖 3 右）。

焦點損失與類別不平衡

白話地說：如果 99.9% 的候選都是背景，一般的損失函數會被數百萬個簡單的「這裡沒東西」樣本主導。焦點損失把模型已經答對的樣本「調小音量」。

令 $p \in [0,1]$ 為前景類別的預測機率，$y \in \{0,1\}$ 為標籤。若 $y=1$，定義 $p_t = p$，否則 $p_t = 1-p$。**焦點損失**（focal loss）[15] 為

$$\mathrm{FL}(p_t) = -\alpha_t (1-p_t)^{\gamma} \log(p_t),$$

其中 $\gamma \ge 0$ 是聚焦參數，α_t 是類別平衡權重（前景為 α，背景為 $1-\alpha$）。$\gamma = 0$ 時它就是加權交叉熵。以論文預設的 $\gamma = 2$、$\alpha = 0.25$，一個 $p_t = 0.99$ 的簡單負樣本相對於交叉熵被降權 $(0.01)^2 = 10^{-4}$ 倍，而困難樣本幾乎保留全部權重。損失會再除以正樣本錨框的數量。二階段偵測器用別的方法避開這個問題（提議階段先移除大部分背景，再以固定的前景：背景比例取樣），而 SSD [26] 使用**困難負樣本挖掘**（只保留損失最高的負樣本，比例約 3:1）。焦點損失第一次讓**密集**的一階段偵測器和二階段偵測器一樣準確。

二階段偵測器：R-CNN 家族

白話地說：先提出幾千個「可能有東西」的地方，再仔細看每一個，決定它**是什麼**以及邊緣**確切在哪裡**。

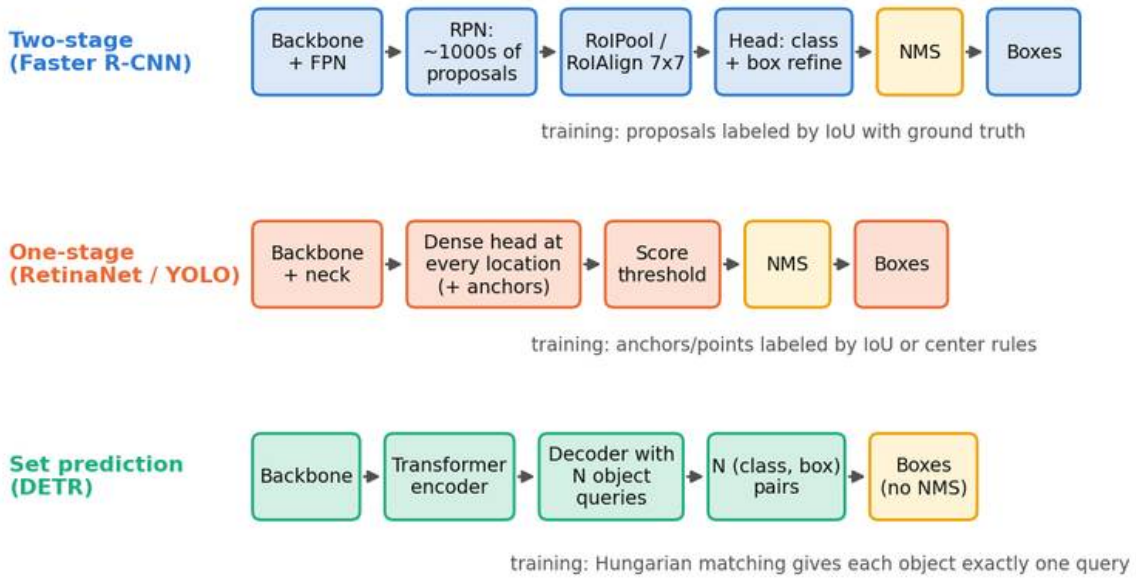


圖 5 — 三種偵測器流程。對二階段與一階段偵測器而言，*NMS*（黃色）是後處理步驟；*DETR* 式的集合預測透過學習一對一指派而把它移除。

R-CNN（2014） [8] 先用一個由下而上的分組演算法，為每張影像產生約 2,000 個與類別無關的候選區域（region proposal），把每個區域變形成固定大小，送進以分類預訓練的 CNN，以每類別一個的線性 SVM 為特徵評分，再用上述目標微調邊界框。它把 PASCAL VOC 2012 的 mean AP 相對於先前最佳結果提升了 30% 以上，並確立了「先在分類上預訓練，再在偵測上微調」這個至今仍是標準的做法。它非常慢，因為 CNN 要對每個候選區域各跑一次。

Fast R-CNN（2015） [9] 對整張影像只跑一次 CNN，再用 **RoI pooling** 從共享的特徵圖上切出每個候選區域：把區域投影到特徵圖上，分成固定的 $H \times W$ 網格（例如 7×7），每格做最大池化，所以不論形狀為何，每個候選區域都得到相同大小的特徵。整個網路以多任務損失訓練：

$$L = L_{\text{cls}}(p, u) + \lambda \, [u \geq 1] \, L_{\text{loc}}(t^u, v),$$

其中 p 是 $K+1$ 類的 softmax，u 是真實類別（背景為 0），$[u \geq 1]$ 只在前景候選區域為 1，t^u 是針對類別 u 預測的框偏移量，v 是目標偏移量，L_{loc} 是 smooth L1 損失。

Faster R-CNN（2015） [10] 讓候選區域本身也由學習產生。一個小型的**區域提議網路**（Region Proposal Network, RPN）在共享特徵圖上滑動，為每個位置的 9 個錨框預測「是否為物件」的分數與框偏移量。最好的候選區域（經 NMS 後）再送進 Fast R-CNN 的偵測頭。由於兩個階段共用骨幹，候選區域幾乎不花成本，整個偵測器也成為一個端到端訓練的網路。

Mask R-CNN（2017） [16] 加上一個小型全卷積分支，在每個 RoI 內為每個類別預測一張二值遮罩，因而得到實例分割。它的關鍵細節是 **RoIAlign**：RoI pooling 會把區域邊界和格子邊緣捨入到特徵網格上，使特徵最多偏移半個步幅。RoIAlign 保留實數座標，並以雙線性內插取樣每一格。這聽起來是小事，但像素級精確的遮罩需要它，高 IoU 下的框準確度也會提升。

僅線上版：*RoI pooling 與 RoIAlign：不捨入的取樣方式* — *https://blog.shyandram.dev/zh-tw/learn/*

deep-learning/object-detection/notes/roi-pooling-與-roialign 不捨入的取樣方式/

Cascade R-CNN（2018） [17] 處理一個微妙的問題。以 IoU ≥ 0.5 為正樣本訓練的偵測頭，學到的是「在 0.5 夠好」的框，在 0.75 卻不夠好。單純提高門檻會讓正樣本太少而過擬合（overfitting）。Cascade R-CNN 串接數個以遞增 IoU 門檻（0.5、0.6、0.7）訓練的偵測頭，每一級都精煉前一級的框，所以每一級拿到的訓練分布品質都更高。它在嚴格的 COCO 指標上帶來大幅提升。

二階段偵測器準確、有彈性（RoI 偵測頭可以預測遮罩、關鍵點或屬性），至今仍是許多實例分割任務的預設選擇，但每個 RoI 一次的偵測頭與 NMS 都要花時間。

一階段偵測器：YOLO、SSD 與 RetinaNet

白話地說：跳過提議步驟。在一次前向傳遞中，對特徵圖的每個位置做一次密集預測，再用 NMS 清理。

YOLO v1（2016） [18] 把影像切成 $S \times S$ 網格（$S = 7$）。每格預測 $B = 2$ 個框（各帶一個信心值）以及一組 C 個類別機率，因此輸出是單一個 $S \times S \times (5B + C)$ 張量（PASCAL VOC 的 20 類對應 $7 \times 7 \times 30$）。物件中心所在的格子負責該物件。它能即時執行，而且因為看得到整張影像，背景誤判比 Fast R-CNN 少；但它定位較不精確，也難以處理小物件與成群的物件，因為每格只能預測固定數量的框與一個類別。

YOLO 家族之後演進出許多版本 [19]–[25]。核心想法始終不變——在網格上做一次密集預測，再用 NMS 清理——每一版則改進其中一個部分：錨框與多尺度預測、更強的骨幹與訓練技巧、更好的標籤指派、注意力機制，最後是不再需要 NMS 的訓練方式（YOLOv10 [24]）。

僅線上版：*YOLO 家族逐版介紹 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-detection/notes/yolo-家族逐版介紹/*

SSD（2016） [26] 在**多個**不同解析度的特徵圖上接上小型卷積預測器，每層都有自己一組多種長寬比的預設框（default box，即錨框），處理尺度的能力遠勝 YOLO v1。它以困難負樣本挖掘與大量資料擴增訓練。

RetinaNet（2017） [15] 結合 FPN 骨幹、密集錨框（每個位置 9 個），以及分類與框迴歸兩個小型子網路，並以焦點損失訓練。論文的分析顯示，一階段偵測器先前落後二階段偵測器的主因是類別不平衡，而不是架構。

無錨框偵測器：用點取代框

白話地說：錨框是對物件長相的猜測。無錨框偵測器拿掉它們，改從**點**預測邊界框：角點、中心點，或物件內部的每一個像素。

CornerNet（2018） [27] 把物件偵測成一對關鍵點：一張左上角熱圖、一張右下角熱圖，以及每個角點的**嵌入向量**（embedding）。同一物件的兩個角點被訓練成嵌入相近，因此以嵌入距離來配對。由於角點常落在物件外面，論文提出**角點池化**（corner pooling），沿著列與行收集角點的證據。

CenterNet，〈Objects as Points〉（2019） [28] 以中心點代表每個物件。網路輸出一張熱圖 $\hat{Y} \in [0,1]^{\frac{W}{R} \times \frac{H}{R} \times K}$（輸出步幅 R，每類一個通道），以及每個位置的尺寸 (w, h) 與次像素偏移。真值中心以

高斯函數散佈在熱圖上，熱圖以逐像素的焦點損失訓練。推論時，峰值就是局部最大值，用 3×3 最大池化比較即可找到，因此**不需要以 IoU 為基礎的 NMS**。同樣的設計只要在中心點迴歸更多量，就能延伸到 3D 框與人體姿態。

FCOS（2019） [29] 把偵測當成分割來做。特徵圖上每個落在真值框內的位置都是正樣本，並迴歸到框四邊的距離：

$$l^* = x - x_1, \qquad t^* = y - y_1, \qquad r^* = x_2 - x, \qquad b^* = y_2 - y,$$

其中 (x, y) 是該位置的影像座標，(x_1, y_1, x_2, y_2) 是框。每一層 FPN 只處理最大迴歸距離落在自己範圍內的物件，這解決了框重疊時大部分的歧義。離物件中心遠的位置預測出的框品質差，因此 FCOS 加入**中心度**（centerness）分支：

$$\text{centerness} = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \cdot \frac{\min(t^*, b^*)}{\max(t^*, b^*)}},$$

它在中心為 1，往邊緣遞減；測試時乘上類別分數，讓偏離中心的框在 NMS 前排得較後面。FCOS 去掉了所有錨框超參數，表現追平甚至超越以錨框為基礎的 RetinaNet。

以 Transformer 做集合預測：DETR 及其後繼者

白話地說：與其預測上千個候選再刪掉重複的，不如直接預測一小**組**答案，並在訓練時讓每個真實物件剛好被一個預測認領。

DETR（2020） [30] 使用 CNN 骨幹、作用在特徵圖上的 Transformer 編碼器，以及一個 Transformer 解碼器；解碼器接收 N 個學習得到的**物件查詢**（object query，論文中為 100 個），平行輸出 N 組（類別、框）預測。沒有錨框、沒有候選區域，也沒有 NMS。由於 N 大於物件數，真值會用特殊的「無物件」類別 $\varnothing$ 補齊到 N 個。

關鍵在於以二**元匹配**（bipartite matching）訓練。令 $y = \{y_j\}$ 為補齊後的真值，$\hat{y} = \{\hat{y}_i\}_{i=1}^{N}$ 為預測。找出使總匹配成本最低的 N 元素排列 $\hat{\sigma}$：

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{j=1}^{N} \mathcal{L}_{\text{match}}(y_j, \hat{y}_{\sigma(j)}), \qquad \mathcal{L}_{\text{match}} = -\mathbb{1}_{\{c_j \neq \varnothing\}}\, \hat{p}_{\sigma(j)}(c_j) + \mathbb{1}_{\{c_j \neq \varnothing\}}\, \mathcal{L}_{\text{box}}(b_j, \hat{b}_{\sigma(j)}),$$

其中 $\mathfrak{S}_N$ 是所有排列的集合，c_j 與 b_j 是目標 j 的類別與框，$\hat{p}_{\sigma(j)}(c_j)$ 是被匹配的查詢對類別 c_j 的預測機率，而 $\mathcal{L}_{\text{box}} = \lambda_{L1}\|b_j - \hat{b}_{\sigma(j)}\|_1 + \lambda_{\text{iou}}(1 - \text{GIoU}(b_j, \hat{b}_{\sigma(j)}))$。這個指派問題可以用**匈牙利演算法**（Hungarian algorithm）[31] 在多項式時間內精確求解。接著，訓練損失（**集合損失**，set loss）使用匹配好的配對：

$$\mathcal{L}_{\text{Hungarian}} = \sum_{j=1}^{N} \Big[-\log \hat{p}_{\hat{\sigma}(j)}(c_j) + \mathbb{1}_{\{c_j \neq \varnothing\}}\, \mathcal{L}_{\text{box}}(b_j, \hat{b}_{\hat{\sigma}(j)}) \Big],$$

當 $c_j = \varnothing$ 時，對數機率項的權重降為十分之一，以平衡大量的「無物件」名額。因為每個物件只會匹配到一個查詢，重複的預測會被當成錯誤的「無物件」而受到**懲罰**，解碼器的自注意力因此學會抑制重複，這就是不需要 NMS 的原因。

用 SciPy 寫匹配步驟只需幾行。下面的成本以 DETR 的權重結合三項；`linear_sum_assignment` 可直接處理 $N \times M$ 的長方形矩陣（DETR 的 L1 項是在正規化的 (c_x, c_y, w, h) 上計算，這裡為了簡潔使用角點格式）。

```
import numpy as np
from scipy.optimize import linear_sum_assignment

def giou_matrix(a, b):
    x1 = np.maximum(a[:, None, 0], b[None, :, 0]); y1 = np.maximum(a[:, None, 1], b[None, :,
     ↪  1])
    x2 = np.minimum(a[:, None, 2], b[None, :, 2]); y2 = np.minimum(a[:, None, 3], b[None, :,
     ↪  3])
    inter = np.clip(x2 - x1, 0, None) * np.clip(y2 - y1, 0, None)
    area = lambda b: (b[:, 2] - b[:, 0]) * (b[:, 3] - b[:, 1])
    union = area(a)[:, None] + area(b)[None, :] - inter
    cx1 = np.minimum(a[:, None, 0], b[None, :, 0]); cy1 = np.minimum(a[:, None, 1], b[None,
     ↪  :, 1])
    cx2 = np.maximum(a[:, None, 2], b[None, :, 2]); cy2 = np.maximum(a[:, None, 3], b[None,
     ↪  :, 3])
    c = (cx2 - cx1) * (cy2 - cy1)
    return inter / union - (c - union) / c

rng = np.random.default_rng(1)
N, C = 6, 3                                      # 6 queries, 3 real classes (+ "no object")
logits = rng.normal(size=(N, C + 1))
prob = np.exp(logits) / np.exp(logits).sum(1, keepdims=True)
pred_boxes = np.array([[.10, .10, .40, .50], [.55, .20, .90, .60], [.12, .08, .42, .48],
                       [.30, .60, .60, .95], [.00, .00, .20, .20], [.50, .50, .70, .70]])
gt_boxes  = np.array([[.10, .10, .40, .50], [.32, .62, .62, .93], [.56, .22, .88, .62]])
gt_labels = np.array([0, 2, 1])

w_cls, w_l1, w_giou = 1.0, 5.0, 2.0              # weights used by DETR
cost = (-w_cls * prob[:, gt_labels]                          # (N, M) class term
        + w_l1 * np.abs(pred_boxes[:, None] - gt_boxes[None]).sum(-1)
        - w_giou * giou_matrix(pred_boxes, gt_boxes))
rows, cols = linear_sum_assignment(cost)         # works on rectangular N x M matrices
print("query -> gt:", dict(zip(rows.tolist(), cols.tolist())))
print("unmatched queries (trained as 'no object'):", sorted(set(range(N)) -
 ↪  set(rows.tolist())))
```

```
query -> gt: {0: 0, 1: 2, 3: 1}
unmatched queries (trained as 'no object'): [2, 4, 5]
```

查詢 2 幾乎是查詢 0 的複本，卻沒有被匹配，之後會被訓練成「無物件」。這正是取代 NMS 的機制。

代價。 DETR 需要非常長的訓練（數百個 epoch，Faster R-CNN 大約只要十幾個），小物件表現也差，因為在高解析度特徵圖上做全域注意力太昂貴，只能用單一一層粗特徵。後繼者逐一解決了這些問題。

三個後繼者指出了方向。**Deformable DETR** [32] 讓每個查詢只看橫跨多個特徵層級的少數取樣點，讓多尺度特徵負擔得起，也大幅縮短訓練。**DINO** [33]（不要和同名的自監督方法混淆）以

去雜訊訓練與更好的查詢初始化進一步加快收斂。**RT-DETR** [34] 讓這個設計快到足以即時使用，而且仍不需 NMS。

僅線上版：*DETR 後繼者細說* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-detection/notes/detr-後繼者細說/*

如今 DETR 路線與 YOLO 路線正在匯流：YOLOv10 借用一對一指派來去掉 NMS，即時 DETR 則借用高效的卷積頸部。

開放詞彙與定位偵測器

白話地說：封閉集合的偵測器只能找出它訓練過的 80 或 365 個類別。開放詞彙偵測器在**測試時以文字**接收類別清單，所以你可以直接要它找「紅色消防栓」或「堆高機」，不必重新訓練。

技巧在於把最後的分類層（一個由 K 個類別權重向量組成的固定矩陣）換成**文字嵌入**。對區域特徵 $\mathbf{z}_i$ 與提示詞（例如 “a photo of a forklift”）的文字嵌入 $\mathbf{e}_k$，類別分數是縮放後的餘弦相似度：

$$s_{ik} = \tau \cdot \frac{\mathbf{z}_i^\top \mathbf{e}_k}{\|\mathbf{z}_i\| \, \|\mathbf{e}_k\|},$$

其中 τ 是學習得到的溫度參數。任何能嵌入的片語都能變成一個類別。視覺端必須學會把區域與語言對齊，這需要遠比任何框標註資料集更大量的影像–文字資料。

- **OWL-ViT（2022）** [35] 從以影像–文字對比預訓練的 Transformer 影像編碼器與文字編碼器出發，移除最後的池化，在每個輸出 token 上接上輕量的框與類別偵測頭；類別偵測頭就是上面的文字嵌入。它支援零樣本（文字查詢）與單樣本（影像範例查詢）偵測，論文也顯示模型越大效果越穩定提升。
- **Grounding DINO（2024）** [36] 把語言深度融入 DINO 偵測器：影像與文字互相做交叉注意力的特徵增強器、語言引導的查詢選擇，以及跨模態解碼器。它在偵測、定位（片語對應區域）與圖說資料上預訓練，報告在 COCO 零樣本遷移（完全不用 COCO 訓練資料）上達 52.5 AP，在一組多樣化真實世界偵測資料集組成的基準上平均 AP 為 26.1。
- **YOLO-World（2024）** [37] 以視覺–語言路徑聚合網路（RepVL-PAN）與區域–文字對比預訓練，把開放詞彙帶進即時偵測器。它的**先提示、後偵測**（prompt-then-detect）模式會事先離線編碼使用者的詞彙，因此推論成本與封閉集合的 YOLO 相當。論文報告在 LVIS 上達 35.4 AP，在 V100 GPU 上 52.0 FPS。

可提示的基礎模型更進一步。SAM 3 [38]（2025）接受一個簡短名詞片語或影像範例作為**概念提示**（concept prompt），為影像與影片中**所有**符合的實例輸出遮罩與身分；作者建立了一個資料引擎，產生含有 400 萬個不重複概念標籤的資料集，並報告在其可提示概念分割基準上的準確度是既有系統的兩倍。在這類系統中，偵測成為一個通用「找出這個概念」模型的其中一種輸出，而不是單獨訓練的網路。

小物件與密集場景

白話地說：極小的物件只有很少的像素，步幅 32 的特徵可能把它壓縮到不到一個格子。人群之所以困難，是因為物件互相遮擋，而 NMS 會把相鄰的物件誤認為重複。

小物件。 COCO 把面積小於 32^2 像素的物件定義為「小」[40]。接近該面積上限且大致呈正方形的物件，在步幅 16 的特徵圖上約橫跨 2×2 個格寬；面積小並不能同時限制細長物件的兩條邊長。降取樣可能移除定位所需的細節。IoU 也變得嚴苛：在 10×10 的框上偏差 2 個像素，IoU 就可能跌破 0.7。常見的對策有高解析度特徵層（步幅 4 的 P_2）、特徵金字塔、推論時把影像切塊、提高輸入解析度、複製貼上與尺度擴增，以及不會讓小框缺少正樣本的指派規則。Cheng 等人的綜述與基準 [39] 整理了這些策略，並提出兩個大型小物件基準：SODA-D（24,828 張交通影像、278,433 個實例、9 個類別）與 SODA-A（2,513 張高解析度空拍影像、872,069 個實例、9 個類別）。

密集與遮擋場景。在人群中，真實物件彼此大量重疊，固定的 NMS 門檻只能在召回率與重複之間取捨。Soft-NMS [13] 有幫助；集合預測偵測器則透過學習哪些預測是重複的，完全避開門檻。遮擋也讓框的標註變得模糊（要標完整範圍，還是只標看得見的部分？），評估結果因此取決於協定。這些問題在追蹤中會再次出現，因為物件會消失又重新出現。

偵測器的評估

白話地說：把所有偵測結果依信心排序，由上往下走：每個偵測結果要不是命中（它和一個尚未被認領的真實物件夠吻合），就是誤報。記錄精確率與召回率如何變化；這條曲線下的面積就是平均精確率（Average Precision, AP）。

精確率、召回率與 AP

對單一類別與單一 IoU 門檻 θ，把測試集中所有偵測結果依分數由高到低排序。若某偵測結果與同類別、尚未被匹配的真值框 IoU 至少為 θ，就是**真陽性**（TP）；否則是**偽陽性**（FP）。每個真值框只能被匹配一次，重複的偵測就是這樣被懲罰的。取前 k 個偵測結果時，

$$\text{precision}(k) = \frac{\text{TP}(k)}{k}, \qquad \text{recall}(k) = \frac{\text{TP}(k)}{M},$$

其中 M 是該類別真值物件的數量。為了讓曲線單調，精確率以其上包絡線**內插**：$p_{\text{interp}}(r) = \max_{r' \geq r} p(r')$。接著：

- **VOC 11 點 AP**（PASCAL VOC 到 2009 年為止 [3]）：$\text{AP} = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} p_{\text{interp}}(r)$。
- **VOC 全點 AP**（VOC 2010 起 [41]）：內插曲線下的精確面積，$\text{AP} = \sum_n (r_{n+1} - r_n)\, p_{\text{interp}}(r_{n+1})$，對所有不同的召回值加總。
- **COCO AP** [40]：在 $0, 0.01, \ldots, 1$ 共 101 個召回點上取 p_{interp} 的平均。

圖 6 顯示一條 PR 曲線、它的包絡線，以及 AP 面積。

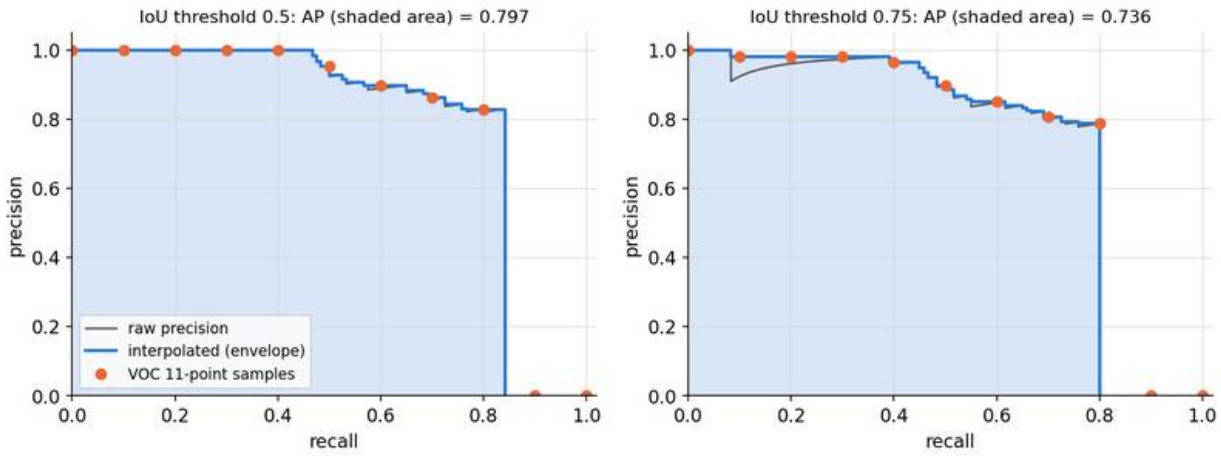


圖 *6* — 合成偵測結果在兩個 *IoU* 門檻下的精確率–召回率。內插包絡線下的陰影面積即全點 *AP*；橘點是 *VOC 11* 點取樣。精確率在召回率終止處降為 *0*，因為有些物件從未被偵測到。

COCO 協定

COCO 的主要指標寫作 **AP** 或 **mAP@[.5:.95]**，對 10 個 IoU 門檻 $\theta \in \{0.50, 0.55, \dots, 0.95\}$ 及全部 80 個類別取平均 [40]：

$$\mathrm{AP}_{\mathrm{COCO}} = \frac{1}{10\,K} \sum_{\theta} \sum_{k=1}^{K} \mathrm{AP}_k(\theta).$$

它獎勵的是精確定位，而不只是找到物件。配套指標有 AP_{50}（VOC 式指標）、AP_{75}（嚴格），以及針對小（面積 $< 32^2$）、中（32^2 到 96^2）、大（$> 96^2$）物件的 AP_S、AP_M、AP_L。在預設依類別評估的 COCO 程式中，100 個偵測結果的上限套用於每張影像的**每個類別**；偵測器本身可能另設更嚴格的總輸出數上限。平均召回率（AR）也在相同的逐類別評估下，使用最大偵測數 1、10、100 的設定 [40]。注意 COCO 把類別平均後的指標稱為「AP」；論文中的「mAP」通常指同一件事。

下面的程式碼在合成預測上從頭計算所有指標（真值加上抖動、約 15% 漏偵測，再加上一些隨機的誤報）。

```
import numpy as np

def box_iou(a, b):
    """Pairwise IoU between (N,4) and (M,4) xyxy boxes -> (N, M)."""
    x1 = np.maximum(a[:, None, 0], b[None, :, 0]); y1 = np.maximum(a[:, None, 1], b[None, :,
     ↪  1])
    x2 = np.minimum(a[:, None, 2], b[None, :, 2]); y2 = np.minimum(a[:, None, 3], b[None, :,
     ↪  3])
    inter = np.clip(x2 - x1, 0, None) * np.clip(y2 - y1, 0, None)
    area = lambda b: (b[:, 2] - b[:, 0]) * (b[:, 3] - b[:, 1])
    return inter / (area(a)[:, None] + area(b)[None, :] - inter)
```

```
def match(dets, gts, thr):
    """dets: list over images of (boxes, scores); gts: list of gt boxes.
    Returns scores, TP flags (sorted by score), and number of GT."""
    recs = []
    for img, ((db, ds), gb) in enumerate(zip(dets, gts)):
        for k in range(len(ds)):
            recs.append((ds[k], img, k))
    recs.sort(key=lambda r: -r[0])                      # global ranking by confidence
    used = [np.zeros(len(g), bool) for g in gts]
    tp = np.zeros(len(recs))
    for n, (s, img, k) in enumerate(recs):
        gb = gts[img]
        if len(gb) == 0:
            continue
        ious = box_iou(dets[img][0][k:k + 1], gb)[0]
        ious[used[img]] = -1                            # each GT can be matched once
        j = ious.argmax()
        if ious[j] >= thr:
            tp[n] = 1; used[img][j] = True
    return tp, sum(len(g) for g in gts)

def pr_curve(tp, n_gt):
    ctp, cfp = np.cumsum(tp), np.cumsum(1 - tp)
    return ctp / n_gt, ctp / (ctp + cfp)                # recall, precision

def ap(rec, prec, mode="all"):
    r = np.concatenate([[0], rec, [1]]); p = np.concatenate([[0], prec, [0]])
    p = np.maximum.accumulate(p[::-1])[::-1]            # monotone envelope
    if mode == "all":                                   # VOC2010+: exact area
        i = np.where(r[1:] != r[:-1])[0]
        return np.sum((r[i + 1] - r[i]) * p[i + 1])
    pts = np.linspace(0, 1, 11 if mode == "voc11" else 101)
    return np.mean([p[r >= t].max() if np.any(r >= t) else 0 for t in pts])

# synthetic data: 50 images, 1-4 objects each; detector = jittered GT + false positives
rng = np.random.default_rng(0)
gts, dets = [], []
for _ in range(50):
    n = rng.integers(1, 5)
    xy = rng.uniform(0, 200, (n, 2)); wh = rng.uniform(20, 80, (n, 2))
    g = np.hstack([xy, xy + wh]); gts.append(g)
    keep = rng.random(n) > 0.15                         # miss ~15% of objects
    jit = rng.normal(0, 0.06, (n, 4)) * np.hstack([wh, wh])
    db = (g + jit)[keep]; ds = rng.uniform(0.4, 1.0, keep.sum())
    m = rng.integers(0, 3)                              # 0-2 false positives
    fxy = rng.uniform(0, 200, (m, 2)); fwh = rng.uniform(20, 80, (m, 2))
    db = np.vstack([db, np.hstack([fxy, fxy + fwh])]); ds = np.concatenate([ds,
     ↪  rng.uniform(0, 0.7, m)])
    dets.append((db, ds))

for thr in (0.5, 0.75):
    tp, n_gt = match(dets, gts, thr)
    rec, prec = pr_curve(tp, n_gt)
    print(f"IoU {thr}: VOC11 {ap(rec, prec, 'voc11'):.3f}  all-point {ap(rec, prec):.3f}"
          f"  COCO-101 {ap(rec, prec, 'coco'):.3f}")
aps = [ap(*pr_curve(*match(dets, gts, t)), "coco") for t in np.arange(0.5, 0.96, 0.05)]
```

```
print("AP@[.5:.95] =", round(float(np.mean(aps)), 3))
```

```
IoU 0.5: VOC11 0.776  all-point 0.797  COCO-101 0.797
IoU 0.75: VOC11 0.749  all-point 0.736  COCO-101 0.738
AP@[.5:.95] = 0.532
```

可以看出三個教訓。三種內插規則得到相近但**不相同**的數字，所以不同協定算出的 AP 絕對不能互相比較。AP@[.5:.95] 比 AP50 低得多，因為抖動的框通不過嚴格門檻。而這只是單一類別；真正的 mAP 會對類別取平均，所以稀有類別和常見類別占同樣的份量。

AP 沒告訴你的事

AP 總結的是整個測試集上的排序品質，因此它不會替你選定操作門檻，也不會區分錯誤的**類型**：定位錯誤、與相似類別混淆、重複偵測、背景誤報，在 AP 裡看起來都一樣。診斷模型時，要依錯誤類型與物件大小拆開分析，並且實際去看影像。

資料集與基準

白話地說：偵測的進展由少數幾個公開資料集推動；每一個都改變了「好」的定義。

資料集	類別	規模（依論文）	標註	重要性
PASCAL VOC [3]	20	以今日標準來看很小	框（＋部分遮罩）	2005–2012 年的基準；AP50
MS COCO [4]	標註 91 種；偵測用 80 種	32.8 萬張影像、250 萬個標註實例	框＋實例遮罩	雜亂的日常場景、大量小物件；AP@[.5:.95]
Objects365 [42]	365	大規模、高品質的框	框	強大偵測器常用的預訓練資料集
Open Images V4 [43]	600 個有框類別	920 萬張影像；1,540 萬個框	框、影像標籤、關係	非常大；影像層級有 1.98 萬個概念
LVIS [44]	超過 1000	16.4 萬張影像中約 200 萬張遮罩	實例遮罩	長尾詞彙；稀有類別評估

有兩個趨勢很明顯。類別數持續增加（20、80，到數百乃至上千），隨之而來的是**長尾**問題：在 LVIS 中，許多類別只有少數幾個訓練樣本，忽略頻率的損失函數或取樣器在稀有類別上就會失敗。另外，最大的模型如今都先在 Objects365 或影像–文字資料上預訓練，再於 COCO 上微調，使得「只用 COCO 訓練」與「使用額外資料」成為兩個分開的排行榜。

速度、準確度與部署

白話地說：實務上你是在一條曲線上選一個點：更準通常代表更多運算。真正重要的是在**你的**硬體上、端到端的延遲。

- **FLOPs 不等於延遲**。FLOPs 計算的是算術量，但執行時間還取決於記憶體存取、運算子支援與平行度。深度可分離卷積與注意力在 FLOPs 上可能很便宜，在某些加速器上卻很慢。
- **端到端量測**。回報延遲時要包含前處理、網路**以及**後處理。NMS 的成本隨候選框數量增加（最壞情況與數量的平方成正比）、依資料而定，而且常在 CPU 上執行。RT-DETR 的論文 [34] 正是分析了這一點，這也是 RT-DETR 與 YOLOv10 [24] 等免 NMS 設計對部署很有吸引力的原因之一。
- **輸入解析度**是最強的調整旋鈕：邊長減半，運算量約減為四分之一，但受傷最重的是小物件。
- **量化**（quantization，例如 8 位元整數的權重與活化值）與**重參數化**（訓練時用多分支區塊，之後折疊成單一卷積，如 YOLOv7 [22] 所用）能在小幅損失準確度下降低延遲。框迴歸輸出與分數頭通常對量化最敏感，所以要用真實資料校準，並且檢查 AP75，而不只是 AP50。
- **批次大小 1** 是邊緣裝置的常見情境；以大批次量測的吞吐量數字不能直接套用。

一個你可以自己訓練的小型偵測器

為了親眼看到邊界框迴歸與 IoU 損失端到端地運作，下面的 PyTorch 程式訓練一個小 CNN，在帶雜訊的 32×32 影像中定位一個矩形或橢圓。這是單一物件偵測器：網路透過 sigmoid 直接迴歸 $[0,1]$ 範圍內的 (c_x, c_y, w, h)，以 L1 加上 GIoU 損失訓練，和 DETR 對匹配到的查詢所做的一樣。

```
import time, torch, torch.nn as nn
torch.manual_seed(0)
S = 32
torch.set_num_threads(1)  # avoid CPU oversubscription in the small teaching example

def make_batch(n):
    """One white rectangle or ellipse per 32x32 noisy image; target = (cx, cy, w, h) in [0,
     ↪  1]."""
    img = 0.1 * torch.randn(n, 1, S, S)
    wh = torch.rand(n, 2) * 0.4 + 0.15
    c = wh / 2 + torch.rand(n, 2) * (1 - wh)
    yy, xx = torch.meshgrid(torch.arange(S), torch.arange(S), indexing="ij")
    xx, yy = (xx[None] + 0.5) / S, (yy[None] + 0.5) / S
    dx = (xx - c[:, 0, None, None]) / (wh[:, 0, None, None] / 2)
    dy = (yy - c[:, 1, None, None]) / (wh[:, 1, None, None] / 2)
    rect = (dx.abs() <= 1) & (dy.abs() <= 1)
    ell = dx**2 + dy**2 <= 1
    shape = torch.where((torch.rand(n) < 0.5)[:, None, None], rect, ell)
    return img + shape[:, None].float(), torch.cat([c, wh], 1)

def xyxy(b):
    return torch.cat([b[:, :2] - b[:, 2:] / 2, b[:, :2] + b[:, 2:] / 2], 1)

def giou(p, g):
    p, g = xyxy(p), xyxy(g)
    lt, rb = torch.max(p[:, :2], g[:, :2]), torch.min(p[:, 2:], g[:, 2:])
    inter = (rb - lt).clamp(min=0).prod(1)
    area = lambda b: (b[:, 2:] - b[:, :2]).prod(1)
    union = area(p) + area(g) - inter
    c = (torch.max(p[:, 2:], g[:, 2:]) - torch.min(p[:, :2], g[:, :2])).prod(1)
    return inter / union, inter / union - (c - union) / c
```

```
net = nn.Sequential(
    nn.Conv2d(1, 16, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),   # 16
    nn.Conv2d(16, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),  # 8
    nn.Conv2d(32, 32, 3, padding=1), nn.ReLU(), nn.MaxPool2d(2),  # 4
    nn.Flatten(), nn.Linear(32 * 4 * 4, 128), nn.ReLU(),
    nn.Linear(128, 4), nn.Sigmoid())                               # (cx, cy, w, h) in [0, 1]
opt = torch.optim.Adam(net.parameters(), 2e-3)

t0 = time.time()
for step in range(500):
    x, y = make_batch(64)
    pred = net(x)
    iou, g = giou(pred, y)
    loss = (pred - y).abs().sum(1).mean() + (1 - g).mean()        # L1 + GIoU loss
    opt.zero_grad(); loss.backward(); opt.step()

with torch.no_grad():
    x, y = make_batch(1000)
    iou, _ = giou(net(x), y)
print(f"train time {time.time() - t0:.1f}s | test mean IoU {iou.mean():.3f}"
      f" | IoU>=0.5: {(iou >= 0.5).float().mean():.1%} | IoU>=0.75: {(iou >=
      ↪  0.75).float().mean():.1%}")
```

```
train time 25.3s | test mean IoU 0.882 | IoU>=0.5: 99.9% | IoU>=0.75: 97.3%
```

（以上為一次 CPU 執行結果；時間與最終分數可能隨硬體及數值函式庫而變。）因為恰好只有一個物件，所以沒有指派問題，也不需要 NMS。練習 6 會請你把它擴充到多個物件，而本篇所有困難的問題都會在那裡重新出現。

現代觀點

圖 7 以每篇引用文獻的發表年份，把本篇的方法放在時間軸上。

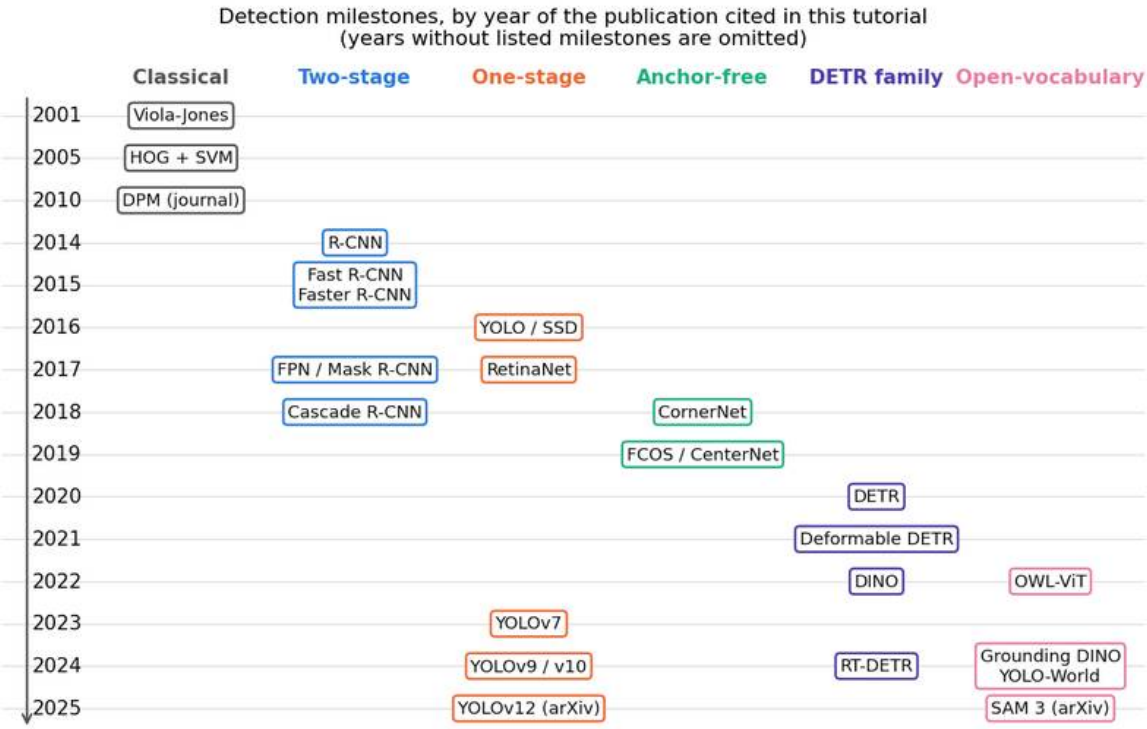


圖 7 — 依本篇引用文獻發表年份排列、按家族分組的里程碑。沒有列出里程碑的年份已省略。

綜述怎麼說

五篇綜述框出了這個領域。Zou 等人 [1] 提供涵蓋最廣的歷史，從 Viola–Jones 到深度偵測器，並指出大多數「新」想法都有更早的根源：多尺度掃描變成特徵金字塔，困難負樣本挖掘變成焦點損失。Liu 等人 [2] 是 CNN 時代最好的單一參考，並強調訓練策略、資料與評估和架構一樣重要。Shehzadi 等人 [45] 回顧近期對 DETR 的改進；Zhu 與 Chen [46] 整理開放詞彙方法，並提醒預訓練資料與測試資料的詞彙重疊會讓評估變得複雜；Cheng 等人 [39] 則顯示小物件仍遠比大物件困難。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-detection/papers/現代觀點綜述導讀/*

2023–2026 年的最先進技術

- **即時封閉集合偵測**如今是 YOLO 路線（YOLOv9 [23]、YOLOv10 [24]、YOLOv12 [25]）與即時 DETR（RT-DETR [34]）之間的競賽。兩條路線互相借鏡：YOLO 引入一對一指派與免 NMS 偵測頭，DETR 引入卷積頸部與混合編碼器。在即時預算下使用注意力已經負擔得起。
- **高準確度的封閉集合偵測**由 DETR 式模型主導，例如搭配大型骨幹與大規模預訓練（Objects365 [42] 及更多）的 DINO [33]。COCO 上的進步幅度已經很小，對最大的模型而言這個基準已接近飽和，因此注意力轉向更難的基準：LVIS [44]、開放詞彙與跨領域評估。

- **開放詞彙與定位偵測**（OWL-ViT [35]、Grounding DINO [36]、YOLO-World [37]）正走進實務：在許多應用中，你現在是去**提示**一個偵測器，而不是訓練一個，必要時再微調。
- **可提示的基礎模型**（SAM 3 [38]）以概念提示統一了偵測、分割與追蹤。

尚待解決的問題

1. **定位品質與信心分數**。分類分數與框品質分開訓練，兩者的相關性常常不高，所以「最好」的框可能被一個更有信心、卻更差的框抑制掉。
2. **小、擁擠與被遮擋的物件**仍遠未解決 [39]。
3. **長尾與開放世界辨識**。稀有類別與真正未見過的物件，以及知道模型**何時**不知道。
4. **評估**。COCO AP 隱藏了錯誤類型，在頂端已經飽和；開放詞彙的評估則取決於提示詞，以及預訓練資料裡原本就包含了什麼 [46]。
5. **在真實硬體上的效率**，包括免 NMS 設計、利於量化的偵測頭，以及公平的延遲報告。

重點整理

- 偵測預測數量不固定的方框、類別與分數集合。
- 方框參數化定義定位目標，指派則決定各預測學習哪個物件。
- NMS 抑制重複預測，擁擠場景讓這項決策更困難。
- 特徵金字塔處理尺度差異，焦點損失調整容易與困難樣本的貢獻。
- 比較架構時，分清候選區域細化、密集預測與一對一集合匹配。
- 開放詞彙偵測改變類別介面，仍須評估定位及詞彙設定。
- 報告 AP 定義、IoU 門檻、物件尺度與完整計時協定。

練習

1. **手算編碼**。一個錨框中心為 $(100, 100)$、大小為 64×32。一個真值框中心為 $(110, 96)$、大小為 80×40。在不做正規化的情況下計算 (t_x, t_y, t_w, t_h)，再把它們解碼回去。

   提示

   $t_x = 10/64 = 0.156$，$t_y = -4/32 = -0.125$，$t_w = \log(80/64) = 0.223$，$t_h = \log(40/32) = 0.223$。解碼：$c_x = 100 + 0.156 \cdot 64 = 110$，$w = 64e^{0.223} = 80$，$y$ 與 h 同理。

2. **IoU 與平移**。證明兩個相同的 $a \times a$ 正方形沿單一軸平移 d 時，$\text{IoU} = (a - d)/(a + d)$。平移多少時 IoU 為 0.5 與 0.75？再對兩軸等量平移的情況重做一次（圖 2 右）。

   提示

   交集為 $a(a - d)$，聯集為 $2a^2 - a(a - d) = a(a + d)$。IoU 0.5 時 $d = a/3$；IoU 0.75 時 $d = a/7$。斜向平移時 $\text{IoU} = (a - d)^2/(2a^2 - (a - d)^2)$，所以 IoU 0.5 時 $d = a(1 - \sqrt{2/3}) \approx 0.18a$。

3. **貪婪 NMS 何時失敗**。修改 NMS 範例，讓兩個不同的人以 IoU 0.6 重疊，然後分別以門檻 0.5 與 0.7 執行貪婪 NMS，以及 Soft-NMS。哪一種能找回兩個人？高門檻又帶來什麼新問題？

   提示

   門檻 0.5 時，分數較低的那個人被刪掉。門檻 0.7 時兩人都留下，但同一個人 IoU 介於 0.5 到 0.7 的重複框也會留下，增加誤報。Soft-NMS 以降低後的分數保留第二個人（原分數乘上 $e^{-0.36/0.5} \approx 0.49$），在 AP 中仍會計入召回。

4. **焦點損失的數字**。在 $\gamma = 2$、$\alpha = 0.25$ 下，對一個背景錨框計算 $p = 0.01$（簡單）與 $p = 0.6$（困難）時的焦點損失與一般交叉熵。兩者各自差了多少倍？

   提示

   背景時 $p_t = 1-p$，$\alpha_t = 0.75$。簡單：$\mathrm{CE} = -\log 0.99 = 0.01005$；$\mathrm{FL} = 0.75 \cdot 0.01^2 \cdot 0.01005 \approx 7.5 \times 10^{-7}$，約小了 1.3×10^4 倍。困難：$\mathrm{CE} = -\log 0.4 = 0.916$；$\mathrm{FL} = 0.75 \cdot 0.6^2 \cdot 0.916 \approx 0.247$，只小了約 3.7 倍。

5. **協定之間的差異**。使用 AP 程式碼，（藉由改變亂數種子或抖動量）找出 VOC 11 點 AP **高於**全點 AP 的情況，以及低於的情況。說明為什麼兩者沒有固定的大小關係。

   提示

   11 點規則在固定的召回值取樣包絡線，所以若包絡線剛好在取樣點右側很高、之後馬上下降，就會高估；反之則低估。上面的輸出已經同時呈現兩種情形：IoU 0.5 時 11 點的值較低，0.75 時較高。

6. **從一個物件到多個物件**。把小型 PyTorch 偵測器擴充到最多三個形狀的影像。方案 A：像 YOLO v1 一樣預測 8×8 的（物件分數、框）網格，把每個物件指派給其中心所在的格子。方案 B：預測 5 個（類別、框）名額，並像 DETR 一樣以匈牙利匹配訓練。比較兩者的失敗模式。

   提示

   方案 A 在兩個中心落在同一格時會失敗，也需要 NMS 去除相鄰格子的重複框。方案 B 不需要 NMS，但訓練較慢，而且訓練初期匹配不穩定（同一個物件可能在不同步驟被指派給不同名額）。對 detach 後的成本矩陣使用 `scipy.optimize.linear_sum_assignment`。

偵測結果是追蹤的原料。下一步——在影片影格之間串連邊界框並維持身分——請見 DLCV 07・物件追蹤。

參考文獻

1. Z. Zou, K. Chen, Z. Shi, Y. Guo and J. Ye, “Object detection in 20 years: A survey,” *Proceedings of the IEEE*, vol. 111, no. 3, 2023. doi
2. L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu and M. Pietikäinen, “Deep learning for generic object detection: A survey,” *International Journal of Computer Vision*, 2020. doi

3. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, “The PASCAL Visual Object Classes (VOC) challenge,” *International Journal of Computer Vision*, vol. 88, 2010. doi
4. T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick et al., “Microsoft COCO: Common objects in context,” in *Proc. ECCV*, 2014. doi
5. P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proc. IEEE CVPR*, 2001. doi
6. N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Proc. IEEE CVPR*, 2005. doi
7. P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE TPAMI*, 2010. doi
8. R. Girshick, J. Donahue, T. Darrell and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in *Proc. IEEE CVPR*, 2014. arXiv
9. R. Girshick, “Fast R-CNN,” in *Proc. IEEE ICCV*, 2015. arXiv
10. S. Ren, K. He, R. Girshick and J. Sun, “Faster R-CNN: Towards real-time object detection with region proposal networks,” *IEEE TPAMI*, 2017 (conference version 2015). doi
11. H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid and S. Savarese, “Generalized intersection over union: A metric and a loss for bounding box regression,” in *Proc. IEEE/CVF CVPR*, 2019. arXiv
12. Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye and D. Ren, “Distance-IoU loss: Faster and better learning for bounding box regression,” in *Proc. AAAI*, 2020. arXiv
13. N. Bodla, B. Singh, R. Chellappa and L. S. Davis, “Soft-NMS — Improving object detection with one line of code,” in *Proc. IEEE ICCV*, 2017. arXiv
14. T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, “Feature pyramid networks for object detection,” in *Proc. IEEE CVPR*, 2017. doi
15. T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, “Focal loss for dense object detection,” in *Proc. IEEE ICCV*, 2017. doi
16. K. He, G. Gkioxari, P. Dollár and R. Girshick, “Mask R-CNN,” in *Proc. IEEE ICCV*, 2017. doi
17. Z. Cai and N. Vasconcelos, “Cascade R-CNN: Delving into high quality object detection,” in *Proc. IEEE/CVF CVPR*, 2018. doi
18. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE CVPR*, 2016. doi
19. J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7263–7271, 2017. CVF ·arXiv
20. J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” tech. report, arXiv:1804.02767, 2018. arXiv
21. A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” arXiv:2004.10934, 2020. arXiv
22. C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” in *Proc. IEEE/CVF CVPR*, 2023. doi
23. C.-Y. Wang, I-H. Yeh and H.-Y. M. Liao, “YOLOv9: Learning what you want to learn using programmable gradient information,” in *Proc. ECCV*, 2024. doi
24. A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin et al., “YOLOv10: Real-time end-to-end object detection,” in *Proc. NeurIPS*, 2024. arXiv
25. Y. Tian, Q. Ye and D. Doermann, “YOLOv12: Attention-centric real-time object detectors,”

arXiv:2502.12524, 2025. arXiv
26. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single shot multibox detector,” in *Proc. ECCV*, 2016. arXiv
27. H. Law and J. Deng, “CornerNet: Detecting objects as paired keypoints,” in *Proc. ECCV*, 2018. doi
28. X. Zhou, D. Wang and P. Krähenbühl, “Objects as points,” arXiv:1904.07850, 2019. arXiv
29. Z. Tian, C. Shen, H. Chen and T. He, “FCOS: Fully convolutional one-stage object detection,” in *Proc. IEEE/CVF ICCV*, 2019. arXiv
30. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, “End-to-end object detection with transformers,” in *Proc. ECCV*, 2020. doi
31. H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, 1955. doi
32. X. Zhu, W. Su, L. Lu, B. Li, X. Wang and J. Dai, “Deformable DETR: Deformable transformers for end-to-end object detection,” in *Proc. ICLR*, 2021. arXiv
33. H. Zhang, F. Li, S. Liu, L. Zhang, H. Su, J. Zhu, L. M. Ni and H.-Y. Shum, “DINO: DETR with improved denoising anchor boxes for end-to-end object detection,” arXiv:2203.03605, 2022. arXiv
34. Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang et al., “DETRs beat YOLOs on real-time object detection,” in *Proc. IEEE/CVF CVPR*, 2024. doi
35. M. Minderer, A. Gritsenko, A. Stone, M. Neumann, D. Weissenborn et al., “Simple open-vocabulary object detection with vision transformers,” in *Proc. ECCV*, 2022. arXiv
36. S. Liu, Z. Zeng, T. Ren, F. Li, H. Zhang et al., “Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection,” in *Proc. ECCV*, 2024. doi
37. T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang et al., “YOLO-World: Real-time open-vocabulary object detection,” in *Proc. IEEE/CVF CVPR*, 2024. doi
38. N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, R. Hu et al., “SAM 3: Segment anything with concepts,” arXiv:2511.16719, 2025. arXiv
39. G. Cheng, X. Yuan, X. Yao, K. Yan, Q. Zeng et al., “Towards large-scale small object detection: Survey and benchmarks,” *IEEE TPAMI*, vol. 45, no. 11, 2023. arXiv
40. COCO Consortium, “COCO detection evaluation,” official evaluation description (12 metrics, 101-point interpolation). link ·official evaluator
41. M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, “The PASCAL Visual Object Classes challenge: A retrospective,” *International Journal of Computer Vision*, vol. 111, 2015. doi
42. S. Shao, Z. Li, T. Zhang, C. Peng et al., “Objects365: A large-scale, high-quality dataset for object detection,” in *Proc. IEEE/CVF ICCV*, 2019. doi
43. A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin et al., “The Open Images Dataset V4: Unified image classification, object detection, and visual relationship detection at scale,” *International Journal of Computer Vision*, 2020. arXiv
44. A. Gupta, P. Dollár and R. Girshick, “LVIS: A dataset for large vocabulary instance segmentation,” in *Proc. IEEE/CVF CVPR*, 2019. doi
45. T. Shehzadi, K. A. Hashmi, M. Liwicki, D. Stricker and M. Z. Afzal, “Object detection with transformers: A review,” *Sensors*, vol. 25, no. 19, 2025. doi
46. C. Zhu and L. Chen, “A survey on open-vocabulary detection and segmentation: Past, present,

and future," *IEEE TPAMI*, 2024. doi

DLCV 03A・以 YOLOv4 讀懂單階段偵測

從 YOLOv4 的架構、指派、訓練、後處理與實驗，逐步理解即時偵測，再選讀前後代 YOLO 與即時 DETR。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/

你將學到

- 追蹤 YOLOv4 的輸入、骨幹、頸部、輸出頭與方框解碼。
- 區分只影響訓練，以及影響推論成本的改動。
- 從消融辨認單項效果、交互作用與未測試元件。
- 在註明硬體、精度、輸入大小與計時範圍的條件下比較延遲。
- 透過論文導讀追蹤後續 YOLO 與即時 DETR 的設計變化。

先看全貌

YOLOv4 是一篇研究「配方」的論文：作者整理既有概念與修改，測試哪些訓練、架構技巧適合單張一般 GPU，再組成完整模型。因此骨幹、特徵融合、解碼、指派、增強與後處理都能在同一系統中看清楚；後續方法則提供替代決策。

> **用最簡單的話說**
>
> 想像廚師要做出一般家庭廚房也能完成的美味快餐。食譜技巧有些只改採買備料，上桌不增加工作；有些則每盤多花幾秒。廚師逐一測試，留下品質與速度合適的組合。YOLOv4 就像這本偵測食譜：「freebies」只改訓練，「specials」則增加少量推論工作。

單階段偵測器適用相機、機器人與工業檢測。YOLOv4 位於這條發展線中段，可藉它理解前代的網格、錨框、多尺度頭，以及後代取代的錨框、NMS、耦合頭。主文詳解 YOLOv4，前後代各有選讀導讀。

先讀 DLCV 03；其核心組件定義框迴歸、IoU、錨框指派、NMS、特徵金字塔與 focal loss。單階段、無錨框、DETR則介紹本章承接的家族。

先看整個家族

白話版。 YOLO 是多個團隊發展的家族；比較結果前，先確認版本、論文與實作。

時間線

圖 1 依首次公開年份與團隊分類，並標示審查狀態。**版本號不是單一血統**：YOLOv3 原作者停止後，不同團隊繼續取號；YOLOv4 的其中兩位作者後來提出 v7、v9 [1]、[2]。YOLOv5、v8、YOLO11 是**無論文的軟體發行** [3]、[4]、[5]。名稱也無統一管制：2026 年 8 月的「YOLOv14」預印本約兩週後撤回 [6]，數字更大本身不代表品質。

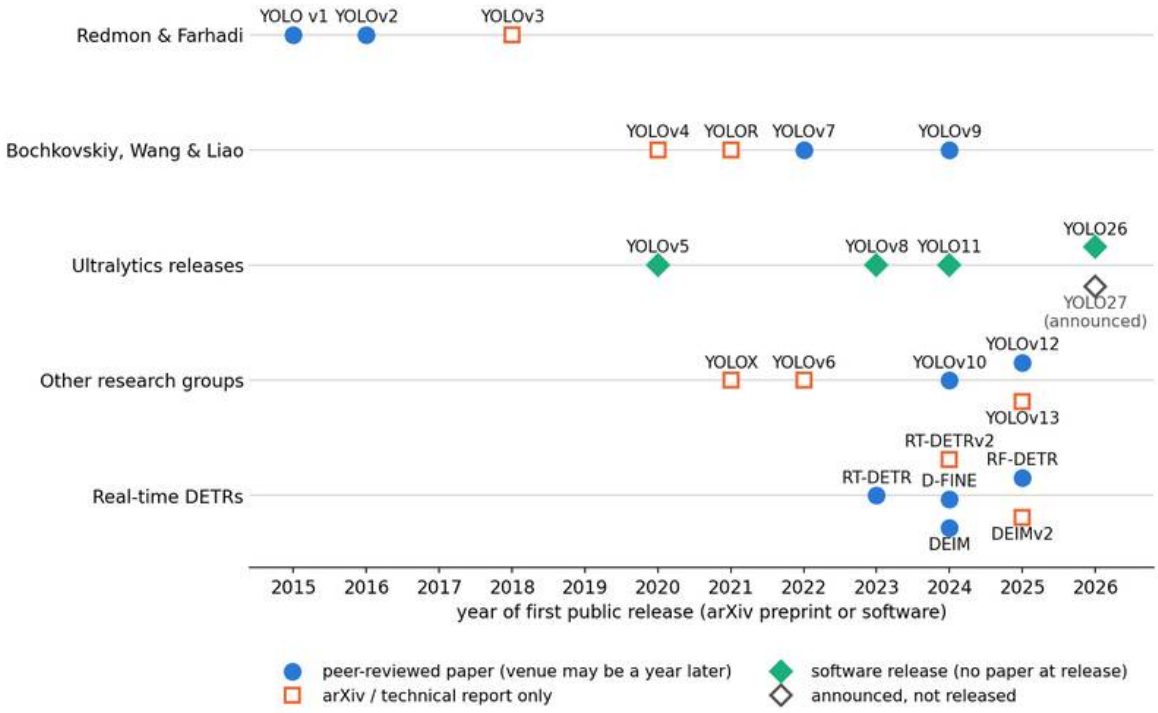


圖 *1* — 依首次公開年份排列。實心圓代表有審查論文，空心方形為預印本或技術報告，菱形為軟體，空心菱形則尚未發行。正式出版可能晚於預印本一年。

本章介紹的論文

下表分為 YOLOv4 前代、後續單階段偵測器、即時 DETR。即時 DETR 的架構不同，但已成為競爭選項；沒有論文的 Ultralytics 發行版放在補充資料。

論文	年份、場域	核心概念	對應 YOLOv4
前代			
YOLO v1 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo-v1 把偵測當成一次迴歸/) [7]	2016，CVPR	$S \times S$ 網格直接迴歸	架構
YOLOv2 ／ YOLO9000 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov2-與-yolo9000 錨框先驗與聯合訓練/) [8]	2017，CVPR	k-means 錨框、有限中心、多尺度訓練	頭

論文	年份、場域	核心概念	對應 YOLOv4
YOLOv3 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov3 三尺度與-darknet-53/) [9]	2018，技術報告	三尺度、Darknet-53、logistic 類別	頭
SSD (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/ssd 多尺度預設框/) [10]	2016，ECCV	多特徵圖預設框	架構
RetinaNet (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/retinanet-與-focal-loss/) [11]	2017，ICCV	Focal loss 處理前背景不均	損失
FCOS (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/fcos 逐像素無錨框偵測/) [12]	2019，ICCV	逐像素框與中心度	指派
CenterNet (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/centernet 把物件視為點/) [13]	2019，arXiv	物體中心熱圖，不用 IoU NMS	後處理
後續方法			
YOLOR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolor 顯式特徵與可學隱式表示/) [14]	2021，arXiv	與輸入無關的可學隱式張量	骨幹
YOLOX (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolox 無錨框-yolo-與-simota/) [15]	2021，arXiv	解耦無錨框頭、SimOTA	頭
YOLOv6 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/) [16]	2022，技術報告	重參數化、任務對齊、量化	骨幹

論文	年份、場域	核心概念	對應 YOLOv4
YOLOv7 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov7 可訓練的免費技巧/) [1]	2023，CVPR	E-ELAN、規劃重參數化、引導標籤	完整配方
YOLOv9 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov9 可編程梯度資訊/) [2]	2024，ECCV	PGI 輔助可逆分支、GELAN	骨幹
YOLOv10 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/) [17]	2024，NeurIPS	一對多／一對一雙指派、免 NMS	後處理
YOLOv12 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov12 以注意力為核心的-yolo/) [18]	2025，NeurIPS	區域注意力	注意力
YOLOv13 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov13 超圖關聯/) [19]	2025，arXiv	超圖特徵關聯	頸部
YOLO26 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo26 移除-dfl-的端到端-yolo/) [20]	2026，arXiv	預設免 NMS、移除 DFL、新最佳化器與小物件指派	指派
即時 *DETR*			
RT-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detr 即時-detr/) [21]	2024，CVPR	混合編碼、不確定性最小查詢	後處理
RT-DETRv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detrv2rt-detr-的實用配方/) [22]	2024，技術報告	各尺度取樣數、離散取樣	配方

論文	年份、場域	核心概念	對應 YOLOv4
D-FINE (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/d-fine 逐層精修框分布/) [23]	2025，ICLR	逐層精修框分布	損失
DEIM (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deim 密集一對一匹配/) [24]	2025，CVPR	密集一對一匹配	Mosaic
RF-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rf-detr 在-dinov2-骨幹上搜尋架構/) [25]	2026，ICLR	DINOv2 與權重共用架構搜尋	骨幹
DEIMv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deimv2 為即時-detr-調適-dinov3/) [26]	2025，arXiv	DINOv3 多尺度偵測調適	骨幹

第一次不用全部讀完

先讀**挑戰**、架構、框解碼與**結果**，再逐次補指派、損失、訓練技巧。各節先說用途才列公式，再用消融核對實測效果。能解釋完整輸入到框流程後，再選一篇前代、一篇後代導讀；完成練習不需先讀遍所有模型。

挑戰

白話版。 YOLOv4 測試既有技巧，希望兼顧準確偵測、即時推論與單張一般 GPU 訓練。

一張一般 GPU 可訓練的快速偵測器

Bochkovskiy、Wang、Liao 於 2020 年 4 月提出 [27]：最準網路常不即時，訓練又需多 GPU、大批次。他們以實際 GPU FPS 定義生產速度，而非只看理論運算量，目標是單張 1080 Ti 或 2080 Ti 即可訓練測試。

YOLOv3 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov3 三尺度與-darknet-53/) [9] 快，但原表在 608×608 僅 33.0% COCO AP [27]；EfficientDet [28] 運算量低、準確，卻不必然有高 GPU FPS。YOLOv4 希望兼得兩者優點。

即時的定義與 YOLOv4 計時方式

每幀處理時間短於幀間隔，才有機會即時。**延遲**是接收一幀到取得最終框的時間，**吞吐量**是成批處理時每秒完成幾幀。單幀為

$$T_{\text{e2e}} = T_{\text{pre}} + T_{\text{net}} + T_{\text{post}},$$

T_{pre} 含解碼、縮放、正規化、傳輸，T_{net} 是前向，T_{post} 含框解碼、閾值與 NMS。f FPS 留下 $1/f$ 秒，30 FPS 約 33.3 ms。批次 B 的吞吐量為 $B/T_{\text{batch}}(B)$，但各幀要等整批；即時相機關心**批次一**延遲。$1000/T$（T 以毫秒）轉 FPS 也只適用批次一。

速度表標註「batch=1、無 TensorRT」[29]，並依 Maxwell、Pascal、Volta 分組，避免直接混比 GPU [27]；30 FPS 以上標為即時。原文未清楚交代精度與是否包含 NMS；後來 YOLOX 以 V100、FP16、batch 1 說明對齊條件 [15]。讀結果時要保留這些限制。

為何測試 freebies 與 specials

許多改進只在特定模型、任務、小資料成立，因此原文將候選分兩袋再測 [27]：

- **Bag of freebies（BoF）**：只改訓練策略或增加訓練成本，推論速度不變，如增強、正規化、平滑、框損失與指派。
- **Bag of specials（BoS）**：增加少量推論成本，換取準確度，如感受野、注意力、特徵融合、啟動與 NMS 變體。

挑戰是找出符合速度與單 GPU 訓練限制的組合。以下依架構、訓練／推論技巧、實驗與未解問題說明。

方法

白話版。 YOLOv4 保留 YOLOv3 頭，加入 CSPDarknet53 與 SPP ／ PANet 頸部，並整合訓練及推論改良。

偵測器的輸入、骨幹、頸部與頭

白話版。骨幹把影像變成多尺度特徵，頸部交換各尺度資訊，頭在各格預測框與類別。

原文圖 2、2.1 節整理後續沿用的詞彙 [27]：

- **輸入**：影像、區塊或金字塔；YOLOv4 採單張正方形，邊長常為 416、512、608。
- **骨幹**：ImageNet [30] 分類預訓練網路；GPU 常用 VGG [31]、ResNet [32] 類，CPU 常採行動骨幹。
- **頸部**：連接骨幹與頭，透過 FPN [33]、PAN [34] 等上下路徑整合，SPP [35] 另擴大感受野。
- **頭**：密集頭在各位置預測，可有錨框（YOLO、SSD、RetinaNet）或無錨框（CenterNet、FCOS）；稀疏頭如兩階段 R-CNN，只處理提案集合。

最終選 CSPDarknet53 [36]、SPP [35] ／ PAN [34]、YOLOv3 頭 [9]。它是**有錨框單階段**：三個尺度，每格三錨框，各輸出四框值、一物件度、每類一分數。

正方形邊長 L、步幅 8、16、32 時，網格邊長為 $L/8, L/16, L/32$，預測總數為

$$N_{\text{pred}} = A\left[\left(\tfrac{L}{8}\right)^2 + \left(\tfrac{L}{16}\right)^2 + \left(\tfrac{L}{32}\right)^2\right],$$

每格 $A = 3$。$L = 608$ 時，$76^2 + 38^2 + 19^2 = 7{,}581$ 格，共 **22,743** 預測；416 時為 **10,647**。每預測 $4 + 1 + K$ 值，COCO 80 類即每尺度 $3 \times 85 = 255$ 通道，與公開設定一致 [37]。大量候選需閾值與 NMS，因此後處理成本重要。

骨幹：CSPDarknet53

白話版。骨幹須看得到大物件、容納多物件，且 GPU 上快。原文比較三種後，採 Darknet-53 加跨階段部分連接，讓每階段僅部分通道走昂貴區塊。

3.1 節認為偵測相較分類需更大輸入、更多層與容量，以處理小物件及多尺度。表 1 以 512 × 512、RTX 2070 比較 [27]：

骨幹	感受野	參數	BFLOPs	FPS（RTX 2070）
CSPResNeXt50	425 × 425	20.6 M	31	62
CSPDarknet53	725 × 725	27.6 M	52	66
EfficientNet-B3 [38]	1311 × 1311	12.0 M	11	26

EfficientNet-B3 約五分之一運算，FPS 卻不到一半，顯示 FLOPs 不預測 GPU 速度。CSPResNeXt50 分類較佳，CSPDarknet53 偵測較佳，見預訓練消融。後者有 29 個 3 × 3 卷積，前者 16 個，因此感受野較大。

CSP 的作用。 CSPNet [36] 以 DenseNet [39] 說明，也用於 ResNeXt [40] 等。DenseNet 各層接收之前輸出的串接；作者認為部分成本來自重複梯度資訊，這是設計解釋，不是所有層學相同特徵的證明。它將階段輸入分成兩通道群：

$$x_0 = [\,x_0',\ x_0''\,],$$

$[\cdot,\cdot]$ 表通道串接。只有 x_0'' 經 k 區塊產生 $x_1, \dots, x_k$，再以 1 × 1 轉換成 x_T；與未處理的 x_0' 串接後再轉換：

$$x_T = w_T * [\,x_0'', x_1, \dots, x_k\,], \qquad x_U = w_U * [\,x_0', x_T\,],$$

$*$ 表卷積，w_T, w_U 是轉換權重。分路可減少重複梯度、處理通道與記憶體交通 [36]。「最後融合」先轉區塊輸出再合旁路，可截斷部分重複梯度流且準確度代價小。套到 Darknet-53 即分割、處理、合併的殘差階段。

啟動。骨幹用 Mish，公開設定的頸部與頭用 LeakyReLU [37]。

頸部一：SPP 區塊

白話版。多種大小的最大池化，讓最粗圖的一格以低成本看到較大鄰域，再把結果疊為通道。

原始 SPP 用固定長度分類向量 [35]，不能原樣放全卷積偵測器。YOLOv4 沿用 YOLOv3 的保尺寸版本：$k \in \{1,5,9,13\}$、步幅一、適當填補，再串接 [27]。輸入 $C \times H \times W$ 時：

$$\mathrm{SPP}(F) = [\,F,\ \mathrm{MaxPool}_5(F),\ \mathrm{MaxPool}_9(F),\ \mathrm{MaxPool}_{13}(F)\,] \in \mathbb{R}^{4C\times H\times W},$$

$\mathrm{MaxPool}_k$ 是步幅一、same padding 的 $k\times k$ 池化，$k=1$ 即原圖。608 輸入的步幅 32 圖為 19×19，13 視窗覆蓋大部分。原文引用 YOLOv3-608 加 SPP 的 AP50 增 2.7%、計算增 0.5%，並指出感受野增加、速度影響小 [27]；公開設定有 5、9、13 三池化 [37]。

```
import numpy as np
from scipy.ndimage import maximum_filter

def spp(x, kernels=(5, 9, 13)):
    """YOLO-style SPP. x: (C, H, W). Stride-1 max pooling with 'same' padding keeps H and W;
    the input and the pooled copies are concatenated along the channel axis."""
    pooled = [maximum_filter(x, size=(1, k, k), mode="constant", cval=-np.inf) for k in
     ↪  kernels]
    return np.concatenate([x] + pooled, axis=0)

rng = np.random.default_rng(0)
x = rng.normal(size=(512, 19, 19))          # e.g. the stride-32 map of a 608 x 608 input
print("SPP:", x.shape, "->", spp(x).shape)

one = np.zeros((1, 19, 19)); one[0, 9, 9] = 1.0   # a single strong activation
y = spp(one)
print("cells reached by one activation per group:", [int((g > 0).sum()) for g in y])
```

```
SPP: (512, 19, 19) -> (2048, 19, 19)
cells reached by one activation per group: [1, 25, 81, 169]
```

通道變四倍，網格仍 19×19；一個強響應最多影響下一層 $13\times13=169$ 格，再以 1×1 縮通道。

頸部二：PANet 路徑聚合

白話版。深層偏語意、淺層偏位置。FPN 向下傳語意，PANet 再向上傳定位。YOLOv4 採雙向頸部，並把相加改為串接。

FPN [33] 以粗圖上採樣加細圖建立 P_3, P_4, P_5，見 DLCV 03。PANet [34] 為縮短低層到頂端路徑，另加由下而上增強；原文從 $N_2 = P_2$ 開始：

$$N_{i+1} = \mathrm{conv}_{3\times3}\Big(\mathrm{conv}_{3\times3}^{\text{stride }2}(N_i) + P_{i+1}\Big),$$

步幅二 3×3 把 N_i 尺寸減半，與同尺寸 P_{i+1} 逐元素相加，再卷積 [34]。YOLOv4 採步幅 8、16、32，N_3, N_4, N_5 接三頭，取代 YOLOv3 的 FPN [27]。

YOLOv4 修改。圖 6 把 PAN 相加改為通道**串接** $[\mathrm{down}(N_i), P_{i+1}]$ 再卷積 [27]。保留兩輸入供後層學權重，代價是較多通道。原文列為單 GPU 訓練調整，但未單獨消融相加對串接。

頭：YOLOv3 頭與框解碼

白話版。每格三模板，各預測位移、伸縮、物件度與類別。YOLOv4 保留頭，但修正中心接近格邊界時難以預測的數值問題。

解碼。延續 YOLOv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov2-與-yolo9000 錨框先驗與聯合訓練/) [8]、YOLOv3 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov3 三尺度與-darknet-53/) [9]。格左上 (c_x, c_y) 以網格為單位，錨寬高 p_w, p_h，輸出 (t_x, t_y, t_w, t_h)，框為

$$b_x = \sigma(t_x) + c_x, \qquad b_y = \sigma(t_y) + c_y, \qquad b_w = p_w\, e^{t_w}, \qquad b_h = p_h\, e^{t_h},$$

$\sigma(t) = 1/(1+e^{-t})$，(b_x, b_y) 是網格中心，乘步幅得像素；(b_w, b_h) 是尺寸。物件度為 $\sigma(t_o)$，類別各用 sigmoid 與 BCE，可多標籤 [9]。公開九錨框：步幅 8 為 $12\times16, 19\times36, 40\times28$；16 為 $36\times75, 76\times55, 72\times146$；32 為 $142\times110, 192\times243, 459\times401$ 像素 [37]。

網格敏感度。$\sigma(t_x) \in (0,1)$ 永不達端點，中心恰在邊界需 $t_x \to \pm\infty$，近邊界也需極大值。原文稱 grid sensitivity，提出乘大於一因子 [27]；程式再寫成

$$b_x = s\,\sigma(t_x) - \frac{s-1}{2} + c_x, \qquad b_y = s\,\sigma(t_y) - \frac{s-1}{2} + c_y,$$

$s \geq 1$ 為 `scale_x_y`，偏移 $-(s-1)/2$ 保持範圍居中，可跨格到 $(c_x - (s-1)/2, c_x + 1 + (s-1)/2)$ [37]。步幅 8、16、32 的 s 為 1.2、1.1、1.05。程式比較所需 t_x。

```
import numpy as np

sigmoid = lambda t: 1 / (1 + np.exp(-t))
logit = lambda p: np.log(p / (1 - p))

def decode(t, cell, anchor, stride, s=1.0):
    """One YOLOv3/YOLOv4 prediction t = (tx, ty, tw, th) -> box centre and size in pixels.
    s = 1 is the YOLOv3 formula; s > 1 is YOLOv4's grid-sensitivity fix."""
    tx, ty, tw, th = t
    bx = s * sigmoid(tx) - (s - 1) / 2 + cell[0]        # centre in grid units
    by = s * sigmoid(ty) - (s - 1) / 2 + cell[1]
    return bx * stride, by * stride, anchor[0] * np.exp(tw), anchor[1] * np.exp(th)

# cell (10, 7) of the stride-8 grid, anchor 36 x 75 pixels, raw outputs all zero
print("box (cx, cy, w, h):", np.round(decode((0, 0, 0, 0), (10, 7), (36, 75), 8), 1))

# which raw output tx puts the centre at a given position inside its cell?
for pos in (0.5, 0.9, 0.99, 1.0):
    row = []
    for s in (1.0, 1.2):
        p = (pos + (s - 1) / 2) / s                     # the sigmoid value that is needed
        row.append(f"s={s}: " + (f"{logit(p):5.2f}" if p < 1 else "  inf"))
    print(f"centre at {pos:4.2f} of the cell -> tx = " + " | ".join(row))
```

```
box (cx, cy, w, h): [84. 60. 36. 75.]
```

```
centre at 0.50 of the cell -> tx = s=1.0:  0.00 | s=1.2:  0.00
centre at 0.90 of the cell -> tx = s=1.0:  2.20 | s=1.2:  1.61
centre at 0.99 of the cell -> tx = s=1.0:  4.60 | s=1.2:  2.29
centre at 1.00 of the cell -> tx = s=1.0:   inf | s=1.2:  2.40
```

$s = 1.2$ 時，邊界只需 $t_x = 2.40$，不再無限大。消融表的「S」即此，見偵測訓練技巧。

訓練目標。三項為 CIoU 框損失、物件度 BCE、正樣本類別 BCE。未指派但已高度重疊真值的預測，不必強制當背景；YOLOv3 忽略 IoU 0.5 以上 [9]，公開 YOLOv4 `ignore_thresh=0.7` [37]。正樣本如何決定見下節。

標籤指派：每物件多個錨框

白話版。 YOLOv3 每物件只給最佳形狀錨框。YOLOv4 保留最佳匹配，再加入夠相似的錨框，增加監督。

YOLOv3 在中心所在格挑形狀 IoU 最高的單錨框 [9]。YOLOv4 先指派最佳，再加符合下式者：

$$\mathrm{IoU}(\text{truth}, \text{anchor}) > \tau$$

IoU 只比較共中心的寬高，τ 是閾值。基因搜尋選 $\tau = 0.213$，公開 `iou_thresh` 相同 [27]、[37]。九錨框跨三尺度，因此可同時在兩層學同一物件；無錨過閾值仍有最佳匹配。完整實作還須處理目標競爭同一預測。

```
import numpy as np

# the nine anchors (width, height in pixels) of yolov4.cfg; three per output stride
anchors = np.array([[12, 16], [19, 36], [40, 28], [36, 75], [76, 55], [72, 146],
                    [142, 110], [192, 243], [459, 401]], float)
stride = np.repeat([8, 16, 32], 3)

def shape_iou(wh, anchors):
    """IoU of boxes that share a centre: only width and height matter."""
    inter = np.minimum(wh[0], anchors[:, 0]) * np.minimum(wh[1], anchors[:, 1])
    return inter / (wh[0] * wh[1] + anchors.prod(1) - inter)

for wh in [(30, 60), (100, 90), (60, 120)]:
    iou = shape_iou(np.array(wh, float), anchors)
    pos = np.union1d([iou.argmax()], np.flatnonzero(iou > 0.213))
    print(f"object {wh[0]:3d}x{wh[1]:<3d} | best anchor {iou.argmax()} (IoU {iou.max():.2f})
    ↪  | "
          f"IoU > 0.213: anchors {pos.tolist()} on strides
           ↪  {sorted(set(stride[pos].tolist()))}")
```

```
object  30x60  | best anchor 3 (IoU 0.67) | IoU > 0.213: anchors [1, 2, 3, 4] on strides [8,
 ↪  16]
object 100x90  | best anchor 6 (IoU 0.58) | IoU > 0.213: anchors [3, 4, 5, 6] on strides [16,
 ↪  32]
object  60x120 | best anchor 5 (IoU 0.68) | IoU > 0.213: anchors [3, 4, 5, 6] on strides [16,
 ↪  32]
```

此例每物件由一個變四個正預測，訊號較多，也更需 NMS。消融「IT」即此。後代用當前預測指派，如 YOLOX（https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolox 無 錨 框-yolo-與-simota/）、YOLOv6（https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/），或以一對一免 NMS，如 YOLOv10（https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/）。DLCV 03另比較兩階段與 RetinaNet。

啟動：Mish

白話版。 YOLOv3 的 LeakyReLU 在零有折角；Mish 平滑、容許小負值，大正值近似恆等，原文發現有助分類與偵測。

Mish 定義為 [41]

$$\mathrm{Mish}(x) = x \tanh(\mathrm{softplus}(x)) = x \tanh(\ln(1 + e^x)),$$

$\mathrm{softplus}(x) = \ln(1 + e^x)$。大正值時 $\tanh \to 1$，Mish 近 x；大負值趨零，中間略負而非單調。LeakyReLU 為正側 x、其餘 $0.1x$ [7]。候選有 ReLU、LeakyReLU、PReLU、ReLU6、SELU、Swish [42]、Mish；原文排除難訓的 PReLU ／ SELU、偏量化的 ReLU6，並指出 Swish、Mish 連續可微 [27]。

```
import numpy as np

mish = lambda x: x * np.tanh(np.log1p(np.exp(x)))   # x * tanh(softplus(x))
leaky = lambda x: np.where(x > 0, x, 0.1 * x)         # the leaky rectifier of YOLO v1-v3

for v in (-3.0, -1.0, 0.0, 1.0, 3.0):
    print(f"x = {v:4.1f} | Mish {mish(v):7.4f} | leaky {float(leaky(v)):5.2f}")
x = np.linspace(-6, 6, 120001)
i = mish(x).argmin()
print(f"Mish minimum {mish(x[i]):.4f} at x = {x[i]:.3f}")
h = 1e-6
print(f"slope at 0: Mish {(mish(h) - mish(-h)) / (2 * h):.4f}; leaky jumps from 0.1 to 1")
```

```
x = -3.0 | Mish -0.1456 | leaky -0.30
x = -1.0 | Mish -0.3034 | leaky -0.10
x =  0.0 | Mish  0.0000 | leaky  0.00
x =  1.0 | Mish  0.8651 | leaky  1.00
x =  3.0 | Mish  2.9865 | leaky  3.00
Mish minimum -0.3088 at x = -1.192
slope at 0: Mish 0.6000; leaky jumps from 0.1 to 1
```

Mish 零點斜率 0.6，LeakyReLU 則由 0.1 跳到 1；最小值約 -0.31，超過 3 接近恆等。原文 CSPResNeXt-50 分類 top-1 **77.9%→78.9%**，Mish 預訓練 CSPDarknet53 的偵測最佳 [27]；代價是比整流多運算。

框損失：從 IoU 到 CIoU

白話版。分開算四座標誤差，忽略它們共同描述一個框。IoU 衡量重疊，CIoU 再加中心距離與長寬比。

原文回顧 [27]：未正規化座標平方差受單位影響。UnitBox 用 $-\log(\text{IoU})$ [43]；常見 $1-\text{IoU}$ 在零重疊仍有限、共同縮放不變，但框分離時梯度為零。GIoU [44] 加最小包圍框懲罰，DIoU／CIoU [45] 再加中心距離與比例。對預測 B、真值 B^{gt}：

$$\mathcal{L}_{\text{CIoU}} = 1 - \text{IoU} + \frac{\rho^2(\mathbf{b},\mathbf{b}^{gt})}{c^2} + \alpha v, \qquad v = \frac{4}{\pi^2}\Big(\arctan\frac{w^{gt}}{h^{gt}} - \arctan\frac{w}{h}\Big)^2, \qquad \alpha = \frac{v}{(1-\text{IoU})+v},$$

$\mathbf{b},\mathbf{b}^{gt}$ 為中心，ρ 為歐氏距離，c 為最小包圍框對角長，w,h,w^{gt},h^{gt} 為寬高，v 為比例差，α 讓重疊佳時比例項更重要 [45]。去末項為 DIoU，再去距離為 IoU。DLCV 03有四者比較；以下拆解 CIoU 三項。

```
import numpy as np

def iou_terms(a, b):
    """a, b: boxes (x1, y1, x2, y2). Returns IoU, the DIoU penalty rho^2/c^2 and the CIoU
     ↪  term alpha*v."""
    iw = max(0.0, min(a[2], b[2]) - max(a[0], b[0])); ih = max(0.0, min(a[3], b[3]) -
     ↪  max(a[1], b[1]))
    inter = iw * ih
    wa, ha, wb, hb = a[2] - a[0], a[3] - a[1], b[2] - b[0], b[3] - b[1]
    iou = inter / (wa * ha + wb * hb - inter)
    cw = max(a[2], b[2]) - min(a[0], b[0]); ch = max(a[3], b[3]) - min(a[1], b[1])
    rho2 = ((a[0] + a[2] - b[0] - b[2]) ** 2 + (a[1] + a[3] - b[1] - b[3]) ** 2) / 4
    v = 4 / np.pi ** 2 * (np.arctan(wb / hb) - np.arctan(wa / ha)) ** 2
    alpha = v / ((1 - iou) + v) if v > 0 else 0.0
    return iou, rho2 / (cw ** 2 + ch ** 2), alpha * v

gt = (100, 100, 200, 300)                                  # 100 x 200 pixels, tall
preds = {"centred, lying flat":       (50, 150, 250, 250),
         "shifted, right shape":      (125, 125, 225, 325),
         "far away":                  (300, 100, 400, 300)}
for name, p in preds.items():
    iou, d, av = iou_terms(p, gt)
    print(f"{name:26s} IoU {iou:.3f} | L_IoU {1 - iou:.3f} | L_DIoU {1 - iou + d:.3f} |
     ↪  L_CIoU {1 - iou + d + av:.3f}")
```

```
centred, lying flat        IoU 0.333 | L_IoU 0.667 | L_DIoU 0.667 | L_CIoU 0.700
shifted, right shape       IoU 0.488 | L_IoU 0.512 | L_DIoU 0.530 | L_CIoU 0.530
far away                   IoU 0.000 | L_IoU 1.000 | L_DIoU 1.308 | L_CIoU 1.308
```

同中心錯比例只有比例懲罰，同形狀位移只有距離懲罰；遠框 IoU 為零，但距離仍提供方向。YOLOv4 認為 CIoU 收斂與準確較佳 [27]，公開 `iou_loss=ciou`、`iou_normalizer=0.07` [37]。

後處理：NMS 與 DIoU-NMS

白話版。多個正樣本會重複預測同一物件，NMS 留最佳、刪高度重疊者，成本依場景而變。DIoU-NMS 降低中心相隔較遠的框被誤刪的機會。

貪婪 NMS。 DLCV 03定義：先取最高分，與剩餘框算 IoU、刪過閾值者、重複。n 候選若很少被刪，比較數接近

$$\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2},$$

k 是每次保留後的剩餘框數。若多為少數物件的重複框，會早刪而省工。因此成本依候選數、信心閾值與場景，不像固定形狀網路前向較穩定。範例的比較次數確定，時間則隨機器變。

```
import numpy as np

def nms_count(boxes, scores, thr=0.5):
    """Greedy NMS that also counts how many box-pair IoUs it computes."""
    order = np.argsort(-scores)
    keep, n_iou = [], 0
    while order.size:
        i, rest = order[0], order[1:]
        keep.append(i)
        n_iou += rest.size
        x1 = np.maximum(boxes[i, 0], boxes[rest, 0]); y1 = np.maximum(boxes[i, 1],
         ↪  boxes[rest, 1])
        x2 = np.minimum(boxes[i, 2], boxes[rest, 2]); y2 = np.minimum(boxes[i, 3],
         ↪  boxes[rest, 3])
        inter = np.clip(x2 - x1, 0, None) * np.clip(y2 - y1, 0, None)
        area = lambda b: (b[..., 2] - b[..., 0]) * (b[..., 3] - b[..., 1])
        iou = inter / (area(boxes[i]) + area(boxes[rest]) - inter)
        order = rest[iou <= thr]
    return len(keep), n_iou

def random_boxes(n, n_objects, rng, size=640):
    """n candidate boxes jittered around n_objects true objects."""
    centres = rng.uniform(40, size - 40, (n_objects, 2))
    owner = rng.integers(0, n_objects, n)
    c = centres[owner] + rng.normal(0, 6, (n, 2))
    wh = rng.uniform(30, 60, (n, 2))
    return np.hstack([c - wh / 2, c + wh / 2]), rng.random(n)

rng = np.random.default_rng(0)
for n_objects in (5, 500):                      # few objects vs a crowded scene
    for n in (100, 1000, 8400):
        b, s = random_boxes(n, n_objects, rng)
        kept, n_iou = nms_count(b, s)
        print(f"objects {n_objects:3d} | boxes {n:4d} | kept {kept:4d} | IoUs {n_iou:9,d}")
```

```
objects   5 | boxes  100 | kept   21 | IoUs       770
objects   5 | boxes 1000 | kept   43 | IoUs     7,809
objects   5 | boxes 8400 | kept   86 | IoUs    90,699
```

```
objects 500 | boxes  100 | kept   94 | IoUs     4,621
objects 500 | boxes 1000 | kept  513 | IoUs   209,490
objects 500 | boxes 8400 | kept 1259 | IoUs 3,039,292
```

同為 8,400 候選，擁擠例需要約 34 倍 IoU 比較。GPU 最佳化與候選上限會改時間，因此這只是教學分析，不是部署預測。圖 2 掃候選數；YOLOv10 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/)、RT-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detr 即時-detr/) 移除這項成本，後者也在真偵測器量測。

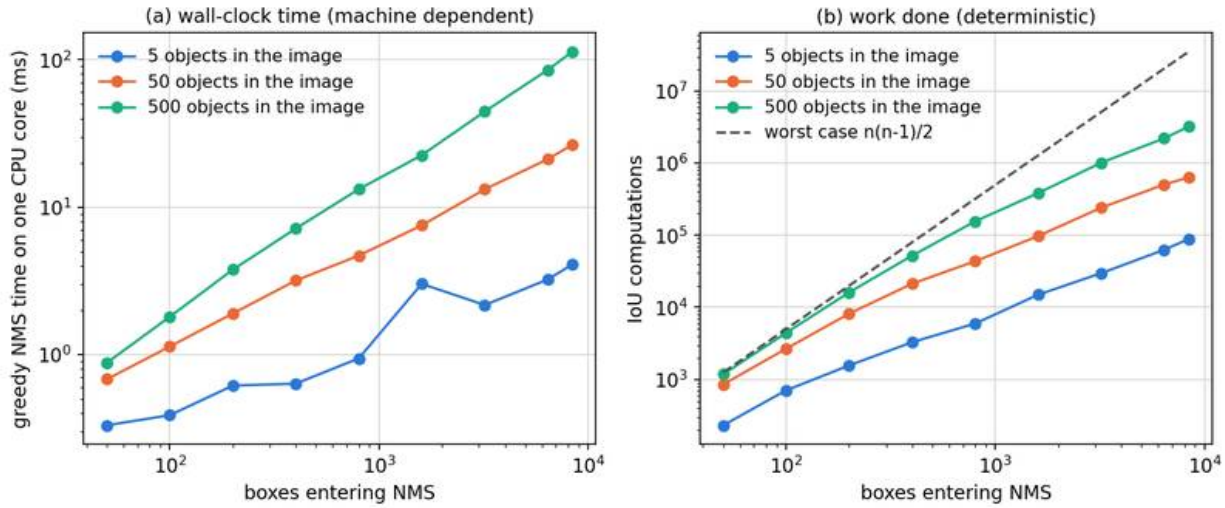


圖 2 — *NumPy* 合成候選的貪婪 *NMS*。(*a*) 單 *CPU* 核心時間依機器而變；(*b*) 比較次數確定，虛線為 $n(n-1)/2$ 最壞情況。

DIoU-NMS。單靠 IoU 可能把相鄰兩人當重複，DIoU 在判斷前減中心距離項 [45]：

$$s_i \leftarrow \begin{cases} s_i, & \mathrm{IoU}(M, B_i) - \dfrac{\rho^2(\mathbf{b}_M, \mathbf{b}_i)}{c^2} < \varepsilon, \\ 0, & \text{otherwise}, \end{cases}$$

s_i 是框 B_i 分數，零表示刪除；ρ 為中心 $\mathbf{b}_M, \mathbf{b}_i$ 距離，c 為包圍框對角，ε 是閾值。中心較遠的框較容易留下 [45]。YOLOv4 列為 detector BoS [27]，程式可獨立執行。

```
import numpy as np

def iou_terms(a, b):
    """a, b: boxes (x1, y1, x2, y2). Returns IoU, the DIoU penalty rho^2/c^2 and the CIoU
     ↪  term alpha*v."""
    iw = max(0.0, min(a[2], b[2]) - max(a[0], b[0])); ih = max(0.0, min(a[3], b[3]) -
     ↪  max(a[1], b[1]))
    inter = iw * ih
    wa, ha, wb, hb = a[2] - a[0], a[3] - a[1], b[2] - b[0], b[3] - b[1]
    iou = inter / (wa * ha + wb * hb - inter)
```

```
    cw = max(a[2], b[2]) - min(a[0], b[0]); ch = max(a[3], b[3]) - min(a[1], b[1])
    rho2 = ((a[0] + a[2] - b[0] - b[2]) ** 2 + (a[1] + a[3] - b[1] - b[3]) ** 2) / 4
    v = 4 / np.pi ** 2 * (np.arctan(wb / hb) - np.arctan(wa / ha)) ** 2
    alpha = v / ((1 - iou) + v) if v > 0 else 0.0
    return iou, rho2 / (cw ** 2 + ch ** 2), alpha * v


def nms(boxes, scores, thr=0.5, diou=False):
    order, keep = list(np.argsort(-scores)), []
    while order:
        m = order.pop(0); keep.append(int(m))
        crit = lambda j: iou_terms(boxes[m], boxes[j])[0] - (iou_terms(boxes[m], boxes[j])[1]
         ↪  if diou else 0)
        order = [j for j in order if crit(j) < thr]
    return keep

# two people side by side (centres 32 px apart) and one duplicate of the first person
boxes = np.array([[100, 100, 200, 200], [132, 100, 232, 200], [103, 100, 203, 200]], float)
scores = np.array([0.9, 0.8, 0.7])
print("greedy NMS keeps:", nms(boxes, scores), "| DIoU-NMS keeps:", nms(boxes, scores,
 ↪  diou=True))
```

```
greedy NMS keeps: [0] | DIoU-NMS keeps: [0, 1]
```

第二人 IoU 0.515 略過閾值，貪婪 NMS 刪除；減距離後 0.478，DIoU-NMS 保留，中心僅差三像素的重複框仍刪。效果只在部分重疊範圍，無法解所有擁擠歧義，原消融也未單獨量它。

資料增強：Mosaic 與 CutMix

白話版。 Mosaic 拼四圖，讓每個訓練畫面含不同背景與較小物件，是 YOLOv4 最廣為沿用的技巧。

原文整理光度、縮放、裁切、翻轉、旋轉與遮擋，也比較 MixUp [46] 的像素／標籤加權，及 CutMix [47] 的矩形貼上與面積混標籤。骨幹分類用 CutMix，偵測引入四圖 Mosaic；致謝 Glenn Jocher 提供 Mosaic、基因搜尋與網格修正概念 [27]。

原文理由：物件出現於非典型背景，且一張拼圖的 BN 統計來自四圖，可降低大批次需求 [27]，呼應單 GPU 目標。

原文只以圖 3 說明，縮放裁切依實作不同。以下示例選隨機分割點，讓四圖角落相接，裁掉各象限外區域，再裁切或丟棄框。

```
import numpy as np

def mosaic(images, boxes, size=416, rng=None, min_keep=0.25):
    """Place four images around a random split point (xc, yc) of a size x size canvas.
    images: four (H, W, 3) arrays; boxes: four (n, 4) arrays of (x1, y1, x2, y2) in pixels.
    Each image touches the split point with one corner; whatever falls outside its quadrant
     ↪  is cut.
```

```
    A box is kept if at least `min_keep` of its area survives the cut."""
    xc, yc = (int(v) for v in rng.uniform(0.3, 0.7, 2) * size)
    canvas = np.full((size, size, 3), 114, np.uint8)
    out, n_cut, n_drop = [], 0, 0
    for k, (img, bx) in enumerate(zip(images, boxes)):
        h, w = img.shape[:2]
        ox = xc - w if k in (0, 2) else xc          # where the image's top-left corner lands
        oy = yc - h if k in (0, 1) else yc
        rx0, rx1 = (0, xc) if k in (0, 2) else (xc, size)   # the quadrant it may fill
        ry0, ry1 = (0, yc) if k in (0, 1) else (yc, size)
        x0, x1, y0, y1 = max(rx0, ox), min(rx1, ox + w), max(ry0, oy), min(ry1, oy + h)
        canvas[y0:y1, x0:x1] = img[y0 - oy:y1 - oy, x0 - ox:x1 - ox]
        shifted = bx + [ox, oy, ox, oy]
        clipped = np.clip(shifted, [x0, y0, x0, y0], [x1, y1, x1, y1])
        area = lambda b: (b[:, 2] - b[:, 0]) * (b[:, 3] - b[:, 1])
        keep = area(clipped) >= min_keep * area(shifted)
        n_cut += int((keep & (area(clipped) < area(shifted))).sum()); n_drop +=
         ↪  int((~keep).sum())
        out.append(clipped[keep])
    return canvas, np.concatenate(out), (xc, yc), n_cut, n_drop

rng = np.random.default_rng(0)
imgs, bxs = [], []
for k in range(4):                                  # four synthetic 256 x 256 "photos"
    img = np.full((256, 256, 3), 60 * k + 40, np.uint8)
    b = []
    for _ in range(3):                              # three rectangles = three objects each
        x, y = rng.integers(0, 200, 2); w, h = rng.integers(30, 90, 2)
        x2, y2 = min(x + w, 256), min(y + h, 256)
        img[y:y2, x:x2] = rng.integers(0, 256, 3); b.append([x, y, x2, y2])
    imgs.append(img); bxs.append(np.array(b, float))

canvas, boxes, split, n_cut, n_drop = mosaic(imgs, bxs, rng=rng)
print("canvas", canvas.shape, "| split point", split)
print(f"objects in: {sum(len(b) for b in bxs)} | kept: {len(boxes)} (of which cut: {n_cut}) |
 ↪  dropped: {n_drop}")
```

```
canvas (416, 416, 3) | split point (223, 181)
objects in: 12 | kept: 9 (of which cut: 3) | dropped: 3
```

圖 3 中十二物件有三個只剩部分、三個幾乎全被裁。此處剩不到四分之一面積就丟，是教學實作選擇，非原文規則。後來 DEIM (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deim 密集一對一匹配/) 另用 Mosaic 增加每圖物件，給一對一更多正樣本。

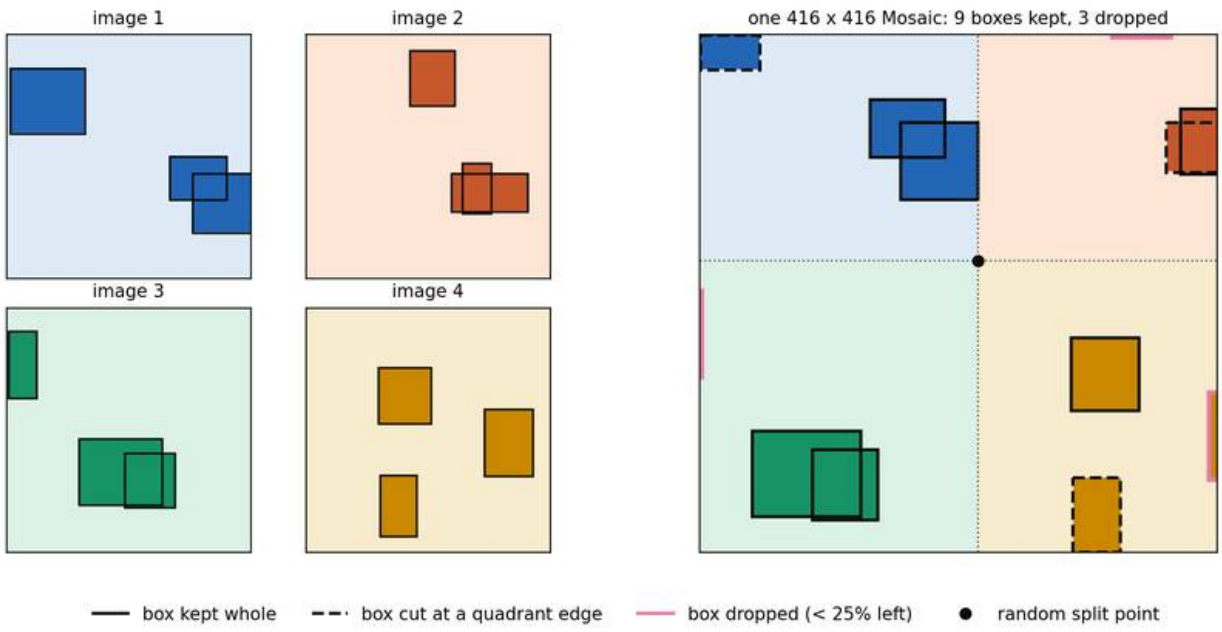


圖 3 — 本章 *Mosaic* 實作：各圖角落接隨機分割點，依象限裁切；框剩餘面積不到 25% 就丟棄。

自對抗訓練

白話版。SAT 先固定權重、改影像讓偵測器受騙，再正常訓練辨識改過的圖，是偵測器自己生成的增強。

原文描述兩次前反向：先對自己做攻擊，使期望物件難以被看見，再在修改圖上正常訓練 [27]。以下是本章示意，不是原文精確公式：

$$x' = x + \eta \nabla_x \mathcal{L}(f_\theta(x), y), \qquad \theta \leftarrow \theta - \lambda \nabla_\theta \mathcal{L}(f_\theta(x'), y),$$

x, y 為影像與框標籤，f_θ 為偵測器，$\mathcal{L}$ 為損失，η 為影像變化量，λ 為學習率。第一步固定權重、提高所選目標；一般偵測損失上升也可能只是定位變差，不一定隱藏物件。第二步正常訓練。SAT 增加訓練計算，不改推論；原文沒有單獨量時間，也無 SAT 消融欄，見證據限制。

正規化：DropBlock 與標籤平滑

白話版。正規化刻意讓訓練困難，減少記憶資料。DropBlock 關掉整塊特徵，標籤平滑則避免對類別過度確信。

Dropout [48] 獨立歸零，但相鄰卷積特徵高度相關，容易從鄰居補回。DropBlock [49] 在能放完整 $b \times b$ 區塊的位置抽 Bernoulli 種子，再歸零整塊、乘遮罩，依總位置數／保留數重縮放。$f \times f$ 特徵欲丟約 $1 - p_{\text{keep}}$，種子率為

$$\gamma = \frac{1 - p_{\text{keep}}}{b^2} \cdot \frac{f^2}{(f - b + 1)^2},$$

p_{keep} 是保留率，b 是區塊邊長，f 是特徵邊長，$(f-b+1)^2$ 為合法種子數；區塊會重疊，故為近似 [49]。原文還由保留率一逐漸降到目標，避免早期傷害。

```
import numpy as np

def dropblock_mask(feat, block, keep_prob, rng):
    """One DropBlock mask for a feat x feat map (Ghiasi et al., Algorithm 1)."""
    gamma = (1 - keep_prob) / block ** 2 * feat ** 2 / (feat - block + 1) ** 2
    half = block // 2
    seeds = np.zeros((feat, feat), bool)                        # only where a whole block
     ↪  fits
    seeds[half:feat - half, half:feat - half] = rng.random((feat - block + 1,) * 2) < gamma
    mask = np.ones((feat, feat))
    for i, j in zip(*np.nonzero(seeds)):
        mask[i - half:i + half + 1, j - half:j + half + 1] = 0.0
    return mask, gamma

rng = np.random.default_rng(0)
feat, block, keep = 38, 7, 0.9
masks = [dropblock_mask(feat, block, keep, rng)[0] for _ in range(2000)]
gamma = dropblock_mask(feat, block, keep, rng)[1]
dropped = 1 - np.mean([m.mean() for m in masks])
print(f"gamma = {gamma:.5f} | target drop rate {1 - keep:.2f} | measured {dropped:.3f}")
m = masks[0]
print("first mask: dropped units", int((m == 0).sum()), "| rescale factor", round(m.size /
 ↪  m.sum(), 3))
```

```
gamma = 0.00288 | target drop rate 0.10 | measured 0.094
first mask: dropped units 171 | rescale factor 1.134
```

38×38、7×7 區塊，種子率不到 0.3% 就丟約 9.4%，因重疊略低於 10% 目標。YOLOv4 把它列在骨幹及偵測 BoF，但未單獨消融 [27]。

類別標籤平滑。硬 one-hot 的交叉熵持續提高正確 logit，可能過度確信；平滑改為 [50]

$$q'(k) = (1-\epsilon)\,\delta_{k,y} + \frac{\epsilon}{K},$$

k 索引 K 類，y 為真類，$\delta_{k,y}$ 是相等指示，ϵ 為平滑量。YOLOv4 用獨立 sigmoid，LS 欄指 sigmoid 平滑 [27]；二元情況可令 $K=2$，正負目標變 $1-\epsilon/2, \epsilon/2$，原文未給 ϵ。分類 top-1 **77.9%→78.1%**，但單獨用於偵測 AP **38.0%→33.0%**，見消融。

正規化：跨迷你批次 CmBN

白話版。單 GPU 每次只放少量圖，BN 統計不穩。CmBN 在同一次權重更新的多個迷你批次間累積統計。

BN [51] 為 $\hat z = \gamma(z-\mu_{\mathcal B})/\sqrt{\sigma^2_{\mathcal B}+\epsilon}+\beta$；$z$ 是啟動，μ,σ^2 是當批均值變異數，γ,β 可學，ϵ 防除零。4 或 8 張統計較吵。CBN [52] 跨迭代，但權重改變，須用 Taylor 近似修正舊統計；跨 GPU 同步則不解單卡限制。

YOLOv4 批次 64，分成 8 或 4 張的 `subdivisions`，整批才更新 [27]、[37]。CmBN 只跨同批迷你批次。依圖 4，可示意第 t 個累積前 $1,\dots,t$ 個：

$$\mu^{(t)} = \frac{1}{t}\sum_{\tau=1}^{t} \mu_{\mathcal{B}_\tau}, \qquad \sigma^{2\,(t)} = \frac{1}{t}\sum_{\tau=1}^{t} \left(\sigma^2_{\mathcal{B}_\tau} + \mu^2_{\mathcal{B}_\tau}\right) - \left(\mu^{(t)}\right)^2,$$

$\mathcal{B}_\tau$ 是等大迷你批次，變異數為母體動差。只平均變異數會漏掉批次均值差；同批權重不變，所以不需跨更新補償。此為正確動差聚合示例，不是原文公式或完整程式規格。CBN 欄單獨 AP **38.4%**，基準 **38.0%** [27]。

注意力與融合：改良 SAM 與加權殘差

白話版。注意力重加權特徵。YOLOv4 簡化 CBAM 的空間注意力為逐元素閘門，另列 EfficientDet 的可學加權相加。

CBAM 的 SAM [53] 對 $F \in \mathbb{R}^{C\times H\times W}$ 沿通道取平均與最大，串成兩張圖再卷積：

$$M_s(F) = \sigma\big(f^{7\times 7}([\,\mathrm{AvgPool}_c(F);\ \mathrm{MaxPool}_c(F)\,])\big), \qquad F' = M_s(F)\otimes F,$$

$f^{7\times 7}$ 為卷積，σ 為 sigmoid，$\otimes$ 跨通道廣播相乘。原文估 SAM 額外計算約 0.1%、GPU 速度幾乎不變，相較代表性通道注意力約增 10% 時間 [27]。圖 5 改成直接卷積 F，閘門同形狀、逐元素縮放。CSPResNeXt50-PANet-SPP 加 SAM，AP **42.4%→42.7%**。

多輸入加權殘差 MiWRC。 EfficientDet 用非負可學權重做快速正規化融合 [28]：

$$O = \sum_i \frac{w_i}{\varepsilon + \sum_j w_j} I_i,$$

I_i 是輸入圖，$w_i \ge 0$ 由 ReLU 保證，$\varepsilon = 0.0001$ 防除零 [28]。YOLOv4 列為骨幹 BoS，未單獨量貢獻。

訓練排程：餘弦退火、基因搜尋與隨機尺寸

白話版。有些技巧只改學習率、超參數選法與訓練輸入尺寸，不改推論網路。

SGDR [54] 讓學習率沿半個餘弦下降：

$$\eta_t = \eta_{\min} + \tfrac{1}{2}(\eta_{\max} - \eta_{\min})\Big(1 + \cos\left(\pi\, T_{\mathrm{cur}}/T_i\right)\Big),$$

$\eta_{\max}, \eta_{\min}$ 是上下界，T_{cur} 是重啟後 epoch，T_i 是週期長。YOLOv4 的 CA 單用 AP **38.7%**，階梯基準 **38.0%** [27]。

基因搜尋 GA。短訓多組參數，保留較佳者，再突變、組合。原文用前 10% 時段搜尋；YOLOv3-SPP、GIoU、5,000 圖 min-val、300 epochs，選出學習率 **0.00261**、動量 **0.949**、指派閾值 **0.213**、框損失正規化 **0.07**。GA 單獨 AP **38.9%**，基準 **38.0%** [27]。

隨機形狀與動態批次。延續 YOLOv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov2-與-yolo9000 錨框先驗與聯合訓練/) 多尺度 [8]，改訓練解析

度，使一模型支援多尺寸。DM 在小圖自動加迷你批次以滿載 GPU，但單獨 AP 降至 **35.3%** [27]。

最佳化錨框 OA。依 512 × 512 重新估錨形狀，延續 YOLOv2 的 IoU k-means，見 DLCV 03 補充資料 [27]。

完整配方

白話版。 CSPDarknet53 ＋ SPP ＋ PAN ＋ YOLOv3 頭，再搭訓練技巧與少量推論模組。

原文 3.4 節列為 [27]：

部位	使用項目	類型
骨幹	CSPDarknet53	架構
頸部	SPP、串接版 PAN	架構
頭	YOLOv3，三尺度各三錨框	架構
骨幹 BoF	CutMix、Mosaic、DropBlock、標籤平滑	訓練
骨幹 BoS	Mish、CSP、MiWRC	推論有成本
偵測 BoF	CIoU、CmBN、DropBlock、Mosaic、SAT、網格修正、多錨框、餘弦退火、基因搜尋、隨機尺寸	訓練
偵測 BoS	Mish、SPP、改良 SAM ／ PAN、DIoU-NMS	推論有成本

後代保留整體形狀，替換個別決策；問題對照這些替換。

結果

白話版。 YOLOv4 在 COCO、608 輸入達 43.5% AP；416 版本在三種 GPU 世代都超過 30 FPS，但 608 在 Maxwell 只有 23 FPS。原比較依速度與準確度取捨，而非全尺寸皆即時。

訓練與評估設定

偵測在 COCO [55] 訓練、test-dev 2017 報 AP [56]：IoU 0.50–0.95 平均、AP50、AP75、小中大物件。分類則 ImageNet [30] 訓練、ILSVRC 2012 驗證 [27]。

偵測預設 500,500 步，學習率 0.01，400,000、450,000 步各乘 0.1，動量 0.9、衰減 0.0005；單 GPU、多尺度、批次 64、迷你批次 8 或 4。分類預設 8,000,000 步、批次 128 ／迷你 32、0.1 多項式衰減、1,000 暖身、動量 0.9、衰減 0.005；分類 BoF 多訓 50%，皆單張 1080 Ti 或 2080 Ti [27]。

主要比較

表 8、9、10 按 Maxwell、Pascal、Volta 分開不同顯卡，皆 batch 1、無 TensorRT [27]。YOLOv4 準確度相同，只重新測速度：

輸入	AP	AP50	AP75	APS	APM	APL	FPS Maxwell	FPS Pascal	FPS Volta
416	41.2	62.8	44.3	20.4	44.4	56.0	38	54	96
512	43.0	64.9	46.5	24.3	46.1	55.2	31	43	83
608	43.5	65.7	47.3	26.7	46.7	53.3	23	33	62

與其他列比較時：

- Maxwell 上 YOLOv3-608 AP **33.0%**、SPP **36.2%**，皆 20 FPS；YOLOv4-608 約增十個 AP 百分點且略快 [27]。
- Volta 上 EfficientDet-D0 至 D3 AP **33.8**、**39.6**、**43.0**、**45.8%**，FPS **62.5**、**50.0**、**41.7**、**23.8**。YOLOv4-512 同為 43.0% 卻 83 FPS；D3 更準但低於 30 FPS。這是相近準確度約快兩倍的依據 [27]。
- 表 9 更慢方法最高 **46.8% AP** ／ **1.3 FPS**；YOLOv4 並非宣稱全體最高 AP，而是即時範圍的取捨 [27]。

大物件 AP 隨輸入增大 **56.0→55.2→53.3**，小物件則 **20.4→26.7**；更大解析度並非各尺度都得利。摘要說 V100 約 65 FPS，表 10 卻列 62 FPS [27]，應採較明確表值。

速度：量測檢查清單

讀取或量測任何偵測延遲，應核對：

1. **硬體、引擎、精度一致**。 YOLOv4 有 GPU 世代與無 TensorRT，未明確精度。
2. **批次一**。大批次吞吐不能直接代表相機延遲。
3. **端到端邊界**。包含前處理、網路、NMS 等後處理，或清楚列排除。YOLOv4 未明確 NMS；RT-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detr 即 時-detr/)、YOLOv10 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/) 則在 COCO 圖片平均計入。
4. **同裝置**。同 GPU 世代不等於同卡；競爭列來自各論文，小 FPS 差不能視為同硬體測量。
5. **暖身與平均**。排除記憶體配置、核心選擇的初期時間，再平均多張。
6. **輸入與閾值**。 NMS 候選取決於信心閾值，低閾值算 AP 也會改時間。

消融實驗

白話版。原文先測分類骨幹、再測偵測 BoF ／ BoS、骨幹權重與批次。逐列閱讀才能分清強證據與尚未隔離的項目。

分類器消融

表 2、3 比較 ImageNet 的技巧、啟動與 top-1 ／ top-5 [27]。

CSPResNeXt-50 設定	Top-1	Top-5
基準	77.9	94.0
MixUp	77.2	94.0
CutMix	78.0	94.3
Mosaic	78.1	94.5
雙邊模糊	77.5	93.8
標籤平滑	78.1	94.4
Swish	64.5	86.0
Mish	78.9	94.5
CutMix ＋ Mosaic ＋平滑	78.5	94.8
再加 Mish	79.8	95.2

CSPDarknet-53 基準 **77.2** ／ **93.6**，三技巧 **77.8** ／ **94.4**，再加 Mish **78.7** ／ **94.8** [27]。這支持該設定的組合，MixUp、模糊未改善。Swish **64.5%** 異常低而原文未解釋，不應泛化為 Swish 本質差。

偵測器的訓練技巧

表 4 在 512 × 512、CSPResNeXt50-PANet-SPP 測 S 網格修正、M Mosaic、IT 多錨框、GA 搜尋、LS 平滑、CBN、CA 餘弦、DM 動態批次、OA 錨框與損失 [27]。

類別	設定	AP	AP50	AP75
基準	MSE、無技巧	38.0	60.0	40.8
單項、MSE	S	37.7	59.9	40.5
	M	39.1	61.8	42.0
	IT	36.9	59.7	39.4
	GA	38.9	61.7	41.9
	LS	33.0	55.4	35.4
	CBN	38.4	60.7	41.3
	CA	38.7	60.7	41.9
	DM	35.3	57.2	38.0
S ＋損失	GIoU ／ DIoU ／ CIoU	39.4 ／ 39.1 ／ 39.6	59.4 ／ 58.8 ／ 59.2	42.5 ／ 42.1 ／ 42.6
組合	S ＋ M ＋ IT ＋ GA、CIoU	41.5	64.0	44.8
	M ＋ GA ＋ OA、CIoU	36.1	56.5	38.4
	S ＋ M ＋ IT ＋ GA ＋ OA、MSE	40.3	64.0	43.1
	同組合、GIoU	42.4	64.4	45.9
	同組合、CIoU	42.4	64.4	45.9

表格支持以下觀察：

- Mosaic 單用增 **1.1 AP** 最多，GA、CA、CmBN 也有小增益。
- IoU 類優於 MSE；搭 S 時 CIoU **39.6** 略高於 GIoU **39.4**，最佳組合兩者同為 **42.4**。
- S、IT、DM、LS 單獨分別降 **0.3**、**1.1**、**2.7**、**5.0 AP**，但 S、IT 出現在佳組合。需更強訓練才能利用新增正樣本是合理解釋，非原文已分析結論。

- 最佳列沒有 LS、CBN、CA、DM，雖最終清單列部分項目；表內沒有全部 BoF 一起啟用的列。

偵測器的推論模組

表 5 基礎 SPP ／ PAN **42.4% AP**，加 SAM **42.7%**（AP50 64.6、AP75 46.3）；其他感受野、高斯不確定性頭、自適應融合組合降至 **41.1–41.8%** [27]。更多模組未必好；DIoU-NMS、偵測 Mish、MiWRC 沒有獨立列。

骨幹與預訓練權重

表 6 在相近設定、512 輸入比較初始化 [27]：

骨幹與分類初始化	AP	AP50	AP75
CSPResNeXt50，預設	42.4	64.4	45.9
CSPResNeXt50，BoF	42.3	64.3	45.7
CSPResNeXt50，BoF ＋ Mish	42.3	64.2	45.8
CSPDarknet53，BoF	42.4	64.5	46.0
CSPDarknet53，BoF ＋ Mish	43.0	64.9	46.5

CSPResNeXt50 分類 **77.9→79.8%** 不帶來較佳偵測，CSPDarknet53 卻有幫助。這支持骨幹選擇，也顯示分類準確度不是可靠偵測代理。

迷你批次大小

表 7 未加技巧時，608 CSPResNeXt50 以迷你 4 ／ 8 得 **37.1** ／ **38.4 AP**；完整 512 CSPDarknet53 得 **41.6** ／ **41.7** [27]。該設定下批次影響變小，支持單卡可訓。Mosaic、CmBN 為此設計，但本表未隔離兩者貢獻。

結論

YOLOv4 展示有錨框單階段偵測，可透過骨幹、頸部與測試過的技巧，在速度及準確度取得實用折衷、單張消費 GPU 訓練。Mosaic、IoU 損失、搜尋、正樣本、SPP、PAN、SAM 有不同程度實驗支持；有些流行技巧無效，最佳分類骨幹也未必最佳偵測。持久貢獻是組件詞彙與在偵測本身驗證配方的做法。

問題

白話版。錨框、共用頭、多正樣本與 NMS 都是後代改進點。證據也有限：部分組件未測、消融單次且使用 test-dev，原論文未經同儕審查。

方法限制

- **錨框依資料**。 COCO 的九模板未必適合新資料與尺寸，YOLOX (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolox 無錨框-yolo-與-simota/) 因此移除。
- **耦合頭**。分類定位共用卷積；YOLOX 顯示解耦可加速收斂。
- **固定指派**。形狀與中心規則不看目前預測，後來有 SimOTA、YOLOv6 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/) 的任務對齊。
- **依賴 NMS**。有閾值、場景相依成本與鄰近物件誤刪；YOLOv10 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/) 提供免 NMS 路線。
- **高解析度大物件**。 APL 在 416、608 分別 **56.0**、**53.3** [27]。

證據限制

- 原文是 arXiv 預印本 [27]，不同於有審查的 YOLOv7 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov7 可訓練的免費技巧/)。
- 消融在 test-dev、單次無變異數，小數點 AP 差可能屬訓練波動。
- SAT、偵測 DropBlock、DIoU-NMS、MiWRC、PAN 修改無獨立列，也無完整全技巧列。
- 精度與 NMS 計時不明，競品非同卡；摘要約 65 FPS 與表 62 也略異。

論文與公開設定的差異

查閱的 `cfg/yolov4.cfg` 使用階梯排程（400,000、450,000）、貪婪 NMS，沒有 SAM ／ DropBlock 層；有 Mosaic、CIoU、網格係數、0.213 指派。學習率 **0.0013** 也不是搜尋值 **0.00261** [37]。檔案可能後改，不能反推原表，但說明實務「YOLOv4」未必包含清單所有項，重現須核版本。

匯出、低精度與重參數化

訓練框架不是部署引擎。訓練好的偵測器通常會**匯出**成 ONNX [57] 這類交換格式，再由推論引擎針對目標裝置編譯，例如 NVIDIA GPU 的 TensorRT [29]。這一路上有三件事會改變：

- **精度**。 FP16 每值儲存是 FP32 一半，速度與 AP 依圖和引擎；INT8 用整數及尺度 [58]，需校準或 QAT。檢查全部輸出與嚴格 IoU AP。YOLOv6 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/) 另處理重參數化量化困難 [16]。
- **簡化運算圖**。 BN 可折入卷積，多分支重參數化也可融合，推導見補充資料。
- **算子與後處理**。引擎須支援 Mish 等算子；RT-DETRv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detrv2rt-detr-的實用配方/) 加離散取樣 [22]，YOLO26 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo26 移除-dfl-的端到端-yolo/) 移除分布頭以簡化匯出 [20]。免 NMS 仍需解碼、選分數、還原座標與計時。

僅線上版：把重參數化區塊折疊成一個卷積 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/notes/把重參數化區塊折疊成一個卷積/*

後續單階段方法改了什麼

下表對應 YOLOv4 組件；連結導讀有完整方法。

YOLOv4 組件	後續變化	論文
有錨框耦合頭	無錨框解耦頭	YOLOX (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolox 無錨框-yolo-與-simota/)
固定形狀指派	預測感知 SimOTA、任務對齊	YOLOX (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolox 無錨框-yolo-與-simota/)、YOLOv6 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/)
CSPDarknet53	推論折疊成卷積的重參數化	YOLOv6 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/)、YOLOv7 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov7 可訓練的免費技巧/)
只監督最終頭	訓練輔助頭與分支	YOLOv7 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov7 可訓練的免費技巧/)、YOLOv9 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov9 可編程梯度資訊/)
多正樣本與 NMS	雙指派與一對一免 NMS	YOLOv10 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/)、YOLO26 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo26 移除-dfl-的端到端-yolo/)
SAM 閘門	區域注意力、超圖關聯	YOLOv12 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov12 以注意力為核心的-yolo/)、YOLOv13 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov13 超圖關聯/)

YOLOv4 組件	後續變化	論文
四值 CIoU	分布框，再移除 DFL	YOLOv6 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov6 重參數化工業偵測器/)、YOLO26 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo26 移除-dfl-的端到端-yolo/)
SGD 排程	MuSGD 混合最佳化器	YOLO26 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo26 移除-dfl-的端到端-yolo/)

延續的則有 CSP、FPN–PAN、Mosaic、IoU 損失，以及實機測速習慣。

單階段之外：即時 DETR

白話版。 DETR 直接輸出物件集合、一物件一預測、免 NMS，但早期訓練推論較慢。即時 DETR 保留集合概念、重設昂貴部分，自 2023 起與 YOLO 競爭。

DLCV 03介紹 DETR [59]。RT-DETR 路線用 Hungarian 一對一匹配 [60]，每物件單預測，免 NMS。各有導讀：

論文	變化
RT-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detr 即時-detr/)	最粗圖注意力、卷積跨尺度、查詢選擇、可調解碼深度、含 NMS 比較
RT-DETRv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detrv2rt-detr-的實用配方/)	各尺度取樣、離散算子、動態增強與學習率
D-FINE (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/d-fine 逐層精修框分布/)	框分布逐層精修與末層蒸餾
DEIM (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deim 密集一對一匹配/)	Mosaic ／ MixUp 增正樣本、匹配感知損失
RF-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rf-detr 在-dinov2-骨幹上搜尋架構/)	DINOv2、權重共享搜尋、多速度品質選項
DEIMv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deimv2 為即時-detr-調適-dinov3/)	DINOv3 多尺度調適、小模型剪枝卷積

現代觀點

綜述怎麼說

A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS（Terven、Córdova-Esparza 與 Romero-González，*Machine Learning and Knowledge Extraction*，2023）[61] 依序走過 16 個版本，先從指標與後處理講起，再談每個版本的架構與訓練技巧。它的討論以四個軸整理這段歷史：錨框（YOLO v1 沒有，YOLOv2 加入，從 YOLOX 起又移除）、框架（先是 Darknet，後來是 PyTorch）、骨幹（Darknet，然後是跨階段部分連接、重參數化與架構搜尋），以及始終存在的速度–準確度取捨。它預期 COCO 會被更困難的基準取代，YOLO 模型也會擴展到追蹤與多模態任務。

YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions（Jegham、Koh、Abdelatti 與 Hendawi，arXiv，2024，2025 年修訂）[62] 是一份實驗性的比較，而不是歷史回顧：它在為了涵蓋不同物件大小、長寬比與小型單一類別物件而挑選的資料集上，評估從 YOLOv3 到 YOLOv12 的版本，並報告精確率、召回率、mAP、處理時間、GFLOPs 與模型大小。它的結論是沒有哪個版本到處都贏：在它的實驗中，YOLO11 給出最好的整體平衡，YOLOv10 速度快但對重疊物件較弱，YOLOv12 增加的複雜度沒有得到回報。請把這些視為一個團隊在其資料集上的結果，而不是通用的排名。

Ultralytics YOLO Evolution: An Overview of YOLO27, YOLO26, YOLO11, YOLOv8, and YOLOv5（Sapkota 與 Karkee，arXiv，2025，2026 年修訂）[63] 只回顧 Ultralytics 的發行，涵蓋架構、基準測試、部署（匯出與量化）與應用，並列出尚待解決的挑戰：密集場景、結合 CNN 與 Transformer、開放詞彙感知、領域泛化，以及硬體感知的最佳化。這是第三方的回顧，不是 Ultralytics 的出版物。

Shehzadi 等人的 *Object Detection with Transformers: A Review*（Sensors 2025）[64] 按骨幹、查詢與注意力組織 DETR 改進，提供即時系列脈絡。

2023–2026 年的最先進技術

- **免 NMS**。 YOLOv10 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov10 雙指派的免-nms-訓練/) [17]、YOLO26 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolo26 移除-dfl-的端到端-yolo/) [20] 與即時 DETR 都採用；YOLO27 的較大模型僅有宣布描述 [65]。
- **可負擔互動**。 YOLOv12 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov12 以注意力為核心的-yolo/) [18]、YOLOv13 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/yolov13 超圖關聯/) [19]、RT-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rt-detr 即時-detr/) [21] 各自限制注意力成本。
- **預訓練 ViT**。 RF-DETR (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/rf-detr 在-dinov2-骨幹上搜尋架構/) [25]、DEIMv2 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deimv2 為即時-detr-調適-dinov3/) [26] 使用 DINOv2 [66]、DINOv3 [67]，須分看準確、轉移與延遲。
- **訓練效率**。 DEIM (https://blog.shyandram.dev/zh-tw/learn/deep-learning/real-time-detection/papers/deim 密集一對一匹配/) [24]、YOLO26 也追求縮短訓練，延續單卡實用目

標。

尚未解決的問題

1. 缺乏跨裝置持續維護的端到端測試；RF-DETR 甚至顯示功率節流會改延遲。
2. 小型、擁擠物件仍難，YOLO26 指派與 DEIMv2 小物件結果都反映限制。
3. INT8 頭與注意力的準確度、算子支援仍需驗證。
4. 部分模型無審查，程式與論文可能不同，版號不等於品質。
5. 開放詞彙如何達到封閉類別 YOLO 的延遲與準確度。

重點整理

- 即時效能是在指定裝置上的端到端延遲預算，不是孤立的 FPS 數值。
- YOLOv4 結合 CSPDarknet53、SPP、改良 PANet 與 YOLOv3 輸出頭。
- 解讀成本時，區分僅用於訓練的技巧與推論元件。
- 消融對各元件的支持程度不同，單獨與組合使用的效果也可能不同。
- 分類準確率與運算量無法直接決定偵測準確率或實測速度。
- 後續偵測器重新設計輸出頭、指派、匹配與特徵融合；比較時須涵蓋完整訓練及推論設定。

練習

1. **預算的算術**。攝影機以 25 FPS 執行。前處理需要 3 ms，網路需要 18 ms，NMS 在 1 ms（空曠的街道）到 9 ms（人群）之間。系統在兩種情況下都跟得上嗎？留給追蹤的時間還有多少？哪一項改變對人群最有幫助？

   提示

   預算是 40 ms。總時間在空曠街道是 22 ms，在人群中是 30 ms，各剩 18 ms 與 10 ms。只有 NMS 這一項取決於場景，所以免 NMS 的偵測頭（或較高的信心閾值，代價是召回率）對人群最有幫助。

2. **數預測**。512 與 1024 輸入各有多少預測？若通過閾值比例相同、NMS 成本依候選平方增長，成本差幾倍？

   提示

   512 有 $64^2 + 32^2 + 16^2 = 5{,}376$ 格，乘三為 16,128；1024 為 64,512，四倍候選使最壞比較約十六倍。實際取決於刪除與場景。

3. **手算網格修正**。$s = 1.1$ 時中心偏移範圍為何？右邊界需要多少 t_x？修改程式驗證。

   提示

   範圍 $(-0.05, 1.05)$；右邊界需 sigmoid $1.05/1.1 \approx 0.9545$，故 $t_x \approx 3.04$。

4. **DropBlock 預算**。 19×19、區塊 7、目標丟 10%，γ 為何？量測並解釋與大圖差異。

   提示

   $\gamma = 0.1/49 \cdot 361/169 \approx 0.00436$。合法種子僅 13×13，有限尺寸、邊界與重疊使近似更明顯；可用實際遮罩檢查。

5. **更好的消融**。設計在其他技巧全開時量各項貢獻的實驗，要幾次訓練？除平均 AP 還報什麼？

   提示

   完整配方加逐項移除，約十餘設定，各多種子報標準差。用 val2017 而非 test-dev，報 AP、AP50、AP75、尺寸別 AP 與訓練時間，尤其 SAT 有額外成本。

6. **兩張速度表**。 A：A100、FP16、batch 32、僅前向 4.1 ms。B：T4、FP16、batch 1、含 NMS 端到端 6.3 ms。為何不能判 A 較快？如何公平重測？

   提示

   不同的 GPU、不同的批次大小（A 在批次 32 下的每張影像時間隱藏了延遲）、不同的計時階段（A 排除了前處理與後處理）。把兩個匯出的模型在同一個裝置上，用同一個引擎與精度，在批次 1、相同的輸入大小下執行，暖機之後對相同的影像做端到端計時，並使用你要部署的信心閾值。

偵測輸出可接追蹤，見 DLCV 07・物件追蹤。

參考文獻

1. C.-Y. Wang, A. Bochkovskiy and H.-Y. M. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” in *Proc. IEEE/CVF CVPR*, pp. 7464–7475, 2023. doi
2. C.-Y. Wang, I-H. Yeh and H.-Y. M. Liao, “YOLOv9: Learning what you want to learn using programmable gradient information,” in *Proc. ECCV*, 2024. doi
3. G. Jocher, “Ultralytics YOLOv5,” software (first released 2020; v7.0 released 22 November 2022). release ·documentation
4. G. Jocher, A. Chaurasia and J. Qiu, “Ultralytics YOLOv8,” software, version 8.0.0, 2023 (no paper). docs ·code
5. G. Jocher and J. Qiu, “Ultralytics YOLO11,” software, version 11.0.0, 2024 (no paper). docs · code
6. J. Lu, J. Jia, J. Yawl and C. Zhang, “YOLOv14: Adaptive real-time object detection for diverse imaging conditions,” arXiv:2608.04720, 2026 (withdrawn by the authors, 20 August 2026). arXiv
7. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You only look once: Unified, real-time object detection,” in *Proc. IEEE CVPR*, pp. 779–788, 2016. doi ·arXiv
8. J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” in *Proc. IEEE CVPR*, pp. 6517–6525, 2017. doi ·arXiv

9. J. Redmon and A. Farhadi, “YOLOv3: An incremental improvement,” tech. report, arXiv:1804.02767, 2018. arXiv
10. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, A. C. Berg, “SSD: Single Shot MultiBox Detector,” *Lecture Notes in Computer Science / Computer Vision – ECCV 2016*, 2016. doi · manuscript
11. T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, “Focal loss for dense object detection,” in *Proc. IEEE ICCV*, pp. 2999–3007, 2017. doi
12. Z. Tian, C. Shen, H. Chen, T. He, “FCOS: Fully Convolutional One-Stage Object Detection,” *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. doi ·manuscript
13. X. Zhou, D. Wang, P. Krähenbühl, “Objects as Points,” *arXiv preprint*, 2019. source
14. C. Wang, I. Yeh, H. M. Liao, “You Only Learn One Representation: Unified Network for Multiple Tasks,” *arXiv preprint*, 2021. source
15. Z. Ge, S. Liu, F. Wang, Z. Li and J. Sun, “YOLOX: Exceeding YOLO series in 2021,” arXiv:2107.08430, 2021. arXiv
16. C. Li, L. Li, H. Jiang, K. Weng, Y. Geng et al., “YOLOv6: A single-stage object detection framework for industrial applications,” tech. report, arXiv:2209.02976, 2022. arXiv
17. A. Wang, H. Chen, L. Liu, K. Chen, Z. Lin, J. Han and G. Ding, “YOLOv10: Real-time end-to-end object detection,” in *Proc. NeurIPS*, 2024. arXiv
18. Y. Tian, Q. Ye and D. Doermann, “YOLOv12: Attention-centric real-time object detectors,” in *Advances in Neural Information Processing Systems 38 (NeurIPS)*, 2025. proceedings ·arXiv
19. M. Lei, S. Li, Y. Wu, H. Hu, Y. Zhou et al., “YOLOv13: Real-time object detection with hypergraph-enhanced adaptive visual perception,” arXiv:2506.17733, 2025. arXiv
20. G. Jocher, J. Qiu, M. Liu, S. Lyu, F. C. Akyon and M. E. Kalfaoglu, “Ultralytics YOLO26: Unified real-time end-to-end vision models,” arXiv:2606.03748, 2026 (software released January 2026). arXiv ·docs
21. Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, J. Chen, “DETRs Beat YOLOs on Real-time Object Detection,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. source
22. W. Lv, Y. Zhao, Q. Chang, K. Huang, G. Wang and Y. Liu, “RT-DETRv2: Improved baseline with bag-of-freebies for real-time detection transformer,” arXiv:2407.17140, 2024. arXiv
23. Y. Peng, H. Li, P. Wu, Y. Zhang, X. Sun and F. Wu, “D-FINE: Redefine regression task in DETRs as fine-grained distribution refinement,” in *Proc. ICLR*, 2025. arXiv ·code
24. S. Huang, Z. Lu, X. Cun, Y. Yu, X. Zhou, X. Shen, “DEIM: DETR with Improved Matching for Fast Convergence,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. source
25. I. Robinson, P. Robicheaux, M. Popov, D. Ramanan and N. Peri, “RF-DETR: Neural architecture search for real-time detection transformers,” in *Proc. ICLR*, 2026. arXiv
26. S. Huang, Y. Hou, L. Liu, X. Yu and X. Shen, “Real-time object detection meets DINOv3,” arXiv:2509.20787, 2025 (revised 2026). arXiv
27. A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” arXiv:2004.10934, 2020. arXiv
28. M. Tan, R. Pang, Q. V. Le, “EfficientDet: Scalable and Efficient Object Detection,” *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. doi ·manuscript
29. NVIDIA, “TensorRT documentation,” official documentation. link

30. J. Deng, W. Dong, R. Socher, L. Li, Kai Li, Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. source
31. K. Simonyan, A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *arXiv preprint*, 2014. source
32. K. He, X. Zhang, S. Ren, J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv preprint*, 2015. source
33. T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, “Feature pyramid networks for object detection,” in *Proc. IEEE CVPR*, 2017. doi
34. S. Liu, L. Qi, H. Qin, J. Shi and J. Jia, “Path aggregation network for instance segmentation,” in *Proc. IEEE/CVF CVPR*, pp. 8759–8768, 2018. doi
35. K. He, X. Zhang, S. Ren, J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *Lecture Notes in Computer Science / Computer Vision – ECCV 2014*, 2014. doi ·manuscript
36. C.-Y. Wang, H.-Y. M. Liao, I-H. Yeh, Y.-H. Wu, P.-Y. Chen and J.-W. Hsieh, “CSPNet: A new backbone that can enhance learning capability of CNN,” *CVPR Workshops*, 2020. arXiv ·doi
37. A. Bochkovskiy and contributors, “Darknet: YOLOv4 configuration and detection layer,” official implementation, accessed 2 November 2026. configuration ·assignment and decoding
38. M. Tan, Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *ICML*, 2019. source ·proceedings
39. G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, “Densely Connected Convolutional Networks,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi ·manuscript
40. S. Xie, R. Girshick, P. Dollar, Z. Tu, K. He, “Aggregated Residual Transformations for Deep Neural Networks,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi ·manuscript
41. D. Misra, “Mish: A Self Regularized Non-Monotonic Activation Function,” *BMVC*, 2020. source
42. P. Ramachandran, B. Zoph, Q. V. Le, “Searching for Activation Functions,” *arXiv preprint*, 2017. source
43. J. Yu, Y. Jiang, Z. Wang, Z. Cao, T. Huang, “UnitBox: An Advanced Object Detection Network,” *ACM Multimedia*, 2016. source
44. H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, S. Savarese, “Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression,” *arXiv preprint*, 2019. source
45. Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye and D. Ren, “Distance-IoU loss: Faster and better learning for bounding box regression,” in *Proc. AAAI*, pp. 12993–13000, 2020. doi
46. H. Zhang, M. Cisse, Y. N. Dauphin and D. Lopez-Paz, “mixup: Beyond empirical risk minimization,” in *Proc. ICLR*, 2018. arXiv
47. S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, J. Choe, “CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features,” *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. doi ·manuscript
48. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, 2014. source
49. G. Ghiasi, T. Lin, Q. V. Le, “DropBlock: A regularization method for convolutional networks,”

NeurIPS, 2018. source ·proceedings
50. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi ·manuscript
51. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. ICML*, PMLR vol. 37, pp. 448–456, 2015. link
52. Z. Yao, Y. Cao, S. Zheng, G. Huang, S. Lin, "Cross-Iteration Batch Normalization," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. doi ·manuscript
53. S. Woo, J. Park, J. Lee, I. S. Kweon, "CBAM: Convolutional Block Attention Module," *Lecture Notes in Computer Science / Computer Vision – ECCV 2018*, 2018. doi ·manuscript
54. I. Loshchilov, F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," *ICLR*, 2017. source
55. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. ECCV*, pp. 740–755, 2014. doi
56. COCO Consortium, "COCO detection evaluation," official evaluation description. link
57. ONNX, "Open Neural Network Exchange," official project page. link
58. B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE/CVF CVPR*, pp. 2704–2713, 2018. doi
59. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. ECCV*, 2020. doi
60. H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955. doi
61. J. Terven, D.-M. Córdova-Esparza and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023. doi ·arXiv
62. N. Jegham, C. Y. Koh, M. Abdelatti and A. Hendawi, "YOLO evolution: A comprehensive benchmark and architectural review of YOLOv12, YOLO11, and their previous versions," arXiv:2411.00201, 2024 (revised 2025). arXiv
63. R. Sapkota and M. Karkee, "Ultralytics YOLO evolution: An overview of YOLO27, YOLO26, YOLO11, YOLOv8, and YOLOv5 object detectors for computer vision and pattern recognition," arXiv:2510.09653, 2025 (revised 2026). arXiv
64. T. Shehzadi, K. A. Hashmi, M. Liwicki, D. Stricker and M. Z. Afzal, "Object detection with transformers: A review," *Sensors*, vol. 25, no. 19, 2025. doi
65. Ultralytics, "YOLO27 (coming soon)," official documentation page, accessed 2 November 2026. docs
66. M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec et al., "DINOv2: Learning robust visual features without supervision," arXiv:2304.07193, 2023. arXiv
67. O. Siméoni, H. V. Vo, M. Seitzer, F. Baldassarre, M. Oquab et al., "DINOv3," arXiv:2508.10104, 2025. arXiv
68. X. Ding, X. Zhang, N. Ma, J. Han, G. Ding and J. Sun, "RepVGG: Making VGG-style ConvNets great again," in *Proc. IEEE/CVF CVPR*, pp. 13728–13737, 2021. doi
69. R. Girshick, "Fast R-CNN," *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. doi ·manuscript

70. S. Ren, K. He, R. Girshick, J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *arXiv preprint*, 2015. source
71. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” *International Journal of Computer Vision*, 2009. source
72. A. Shrivastava, A. Gupta, R. Girshick, “Training Region-Based Object Detectors with Online Hard Example Mining,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. doi ·manuscript
73. H. Zhang, Y. Wang, F. Dayoub and N. Sünderhauf, “VarifocalNet: An IoU-aware dense object detector,” in *Proc. IEEE/CVF CVPR*, pp. 8510–8519, 2021. doi
74. YOLOX contributors, “YOLOX detection head,” official implementation, accessed 2 November 2026. code
75. K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, Q. Tian, “CenterNet: Keypoint Triplets for Object Detection,” *arXiv preprint*, 2019. source
76. Z. Ge, S. Liu, Z. Li, O. Yoshie and J. Sun, “OTA: Optimal transport assignment for object detection,” in *Proc. IEEE/CVF CVPR*, pp. 303–312, 2021. doi
77. C. Li, L. Li, Y. Geng, et al., “YOLOv6 v3.0: A Full-Scale Reloading,” *arXiv preprint*, 2023. source
78. C. Feng, Y. Zhong, Y. Gao, M. R. Scott, W. Huang, “TOOD: Task-Aligned One-Stage Object Detection,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021. source
79. Z. Gevorgyan, “SIoU Loss: More Powerful Learning for Bounding Box Regression,” *arXiv preprint*, 2022. source
80. X. Li, W. Wang, L. Wu, S. Chen, X. Hu, J. Li, J. Tang and J. Yang, “Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection,” in *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020. proceedings
81. X. Ding, H. Chen, X. Zhang, K. Huang, J. Han, G. Ding, “Re-parameterizing Your Optimizers rather than Architectures,” *ICLR*, 2023. source
82. S. Zhang, C. Chi, Y. Yao, Z. Lei, S. Z. Li, “Bridging the Gap Between Anchor-Based and Anchor-Free Detection via Adaptive Training Sample Selection,” *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. doi ·manuscript
83. C. Wang, H. M. Liao, I. Yeh, “Designing Network Design Strategies Through Gradient Path Analysis,” *arXiv preprint*, 2022. source
84. T. Dao, D. Fu, S. Ermon, A. Rudra, C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness,” *Advances in Neural Information Processing Systems 35*, 2022. doi ·manuscript
85. D. P. Kingma, J. Ba, “Adam: A Method for Stochastic Optimization,” *ICLR*, 2015. source
86. K. Jordan, “Muon: An optimizer for hidden layers in neural networks,” technical blog, 2024. source
87. S. Shao, Z. Li, T. Zhang, C. Peng, G. Yu, X. Zhang, J. Li, J. Sun, “Objects365: A Large-Scale, High-Quality Dataset for Object Detection,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. source

DLCV 03B・三維物件偵測：光達、相機與鳥瞰視角

機器如何在三維空間中找出汽車、行人與單車騎士：點雲與座標系、從 PointNet 到 CenterPoint 的光達偵測器、純相機的鳥瞰視角方法、感測器融合、旋轉三維框與 IoU，以及 KITTI、nuScenes 與 Waymo 如何衡量結果。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/

你將學到

- 定義三維方框、座標框架與感測器校準。
- 比較點、體素、柱狀與混合表示。
- 理解僅相機的深度推論與多感測器融合。
- 編碼方向，並區分鳥瞰重疊與完整三維重疊。
- 解讀基準指標，並測試距離、天氣、校準與部署成本。

先看全貌

二維偵測器在影像上畫矩形。三**維偵測器**則把框放進真實世界：每個物體在幾公尺外、有多大、面向哪個方向，而對會移動的東西，還有它跑得多快。自駕車不會因為一個像素矩形而煞車；它煞車，是因為一位行人在前方 14 公尺處，正走進它的車道。

> **用最簡單的話說**
>
> 在黑暗的房間裡閉上眼睛，揮動一支只能告訴你各個方向牆壁有多遠的手電筒。過一陣子，你會得到一團勾勒出家具輪廓的點。三維偵測器看著這樣一團點，然後說：「這裡有張沙發，長 2 公尺，稍微朝左轉。」遠處東西上的點又少又散，所以遠處的沙發很難找。如果你手上只有一張照片、沒有這些點，你能把沙發看得很清楚，卻得猜它有多遠。

為什麼重要。三維偵測是自動駕駛感知系統的核心，它的輸出餵給追蹤、運動預測與規劃 [1]。機器人需要它來抓取與避開物體，擴增實境需要它把虛擬內容錨定在真實物體上。這項任務也逼出了二維視覺從未需要的新想法：處理無序點集的網路、稀疏卷積、鳥瞰視角表示，以及把平面影像「提升」到三維的方法。

本篇假設你已經熟悉 DLCV 03・物件偵測：從 R-CNN 到 DETR 的二維偵測詞彙：框的迴歸目標、IoU、錨框與標籤指派、NMS 與焦點損失（見其核心基本元件）、以中心為基礎的偵測（無錨框偵測器）、DETR 式的集合預測（以 Transformer 做集合預測），以及平均精確率（偵測器的評估）。我們沿用這些想法，只說明在三維中改變了什麼。兩篇綜述為這個領域搭起框架：Mao 等人 [1] 與 Qian 等人 [2]；我們在「現代觀點」中回頭談它們。

三維偵測問題

白話版。三維偵測器把感測觀測轉成方框、類別與分數；直立框設定描述偏航，不包含滾轉與俯仰。

精確版。三維偵測器輸出一個集合

$$\hat{Y} = \{(\hat{\mathbf{b}}_i, \hat{c}_i, \hat{s}_i, \hat{\mathbf{v}}_i)\}_{i=1}^{N}, \qquad \hat{\mathbf{b}}_i = (x, y, z, l, w, h, \theta),$$

其中 (x, y, z) 是框的中心（單位為公尺），(l, w, h) 是它的長、寬、高，θ 是**偏航角**（yaw，繞垂直軸的朝向角），$\hat{c}_i$ 是 K 個類別之一，$\hat{s}_i \in [0, 1]$ 是信心值，$\hat{\mathbf{v}}_i = (v_x, v_y)$ 是選用的地面平面速度（單位為公尺每秒），而 N 隨場景而異。這種 7 個數字的框稱為 **7-DoF**（自由度，degrees of freedom）框；例如 Waymo Open Dataset 就是這樣標註每個物體，長、寬、高沿著框自己的 x、y 與 z 軸量測 [3]。翻滾角與俯仰角假設為零，這對大致平坦道路上的用路人來說是合理的。

和二維偵測相比，有三件事改變了：

1. **框存在於公制空間**。尺寸不會隨距離縮小，所以一輛車不管在 5 m 還是 80 m 外，都約 4.5 m 長。這讓尺寸先驗（錨框、各類別的平均尺寸）遠比在影像中有用。
2. **朝向很重要**。二維框是與座標軸對齊的。三維框必須隨物體轉動，而錯誤的偏航角即使中心完全正確，也可能毀掉重疊程度。
3. **資料稀疏且不均勻**。光達看到的是表面而不是體積，而且多半是面向它的那幾面，遠處的物體只得到幾個點。

感測器與座標系

白話版。每個感測器都在自己的座標中量測。要把它們結合起來，我們用每個感測器已知的位置與朝向，也就是它的**校正**（calibration），把所有東西搬進一個固定在車上的共同座標系。

三種感測器占主導地位（圖 1，左）：

- **光達**（LiDAR）發出雷射脈衝並計時它們的回波。旋轉式的光達每次掃描回傳一個**點雲**（point cloud）：數萬個點 (x, y, z)，每個點帶一個**強度**（intensity，即反射率）。深度是直接且準確地量到的，但沒有顏色，而且點很稀疏。
- **相機**以低成本提供稠密的顏色與紋理，但沒有深度：一個像素只告訴你一條射線的方向，不告訴你物體在這條射線上的多遠處。
- **雷達**（radar）對每個物體只回傳寥寥幾個點，帶有距離，並透過都卜勒效應得到徑向速度。它在霧和雨中也能運作，但角解析度很差。

例如 nuScenes 的車子配有 6 台相機、5 個雷達與 1 個光達，涵蓋 360 度 [4]；Waymo 的車子配有五個光達與五台相機 [3]。

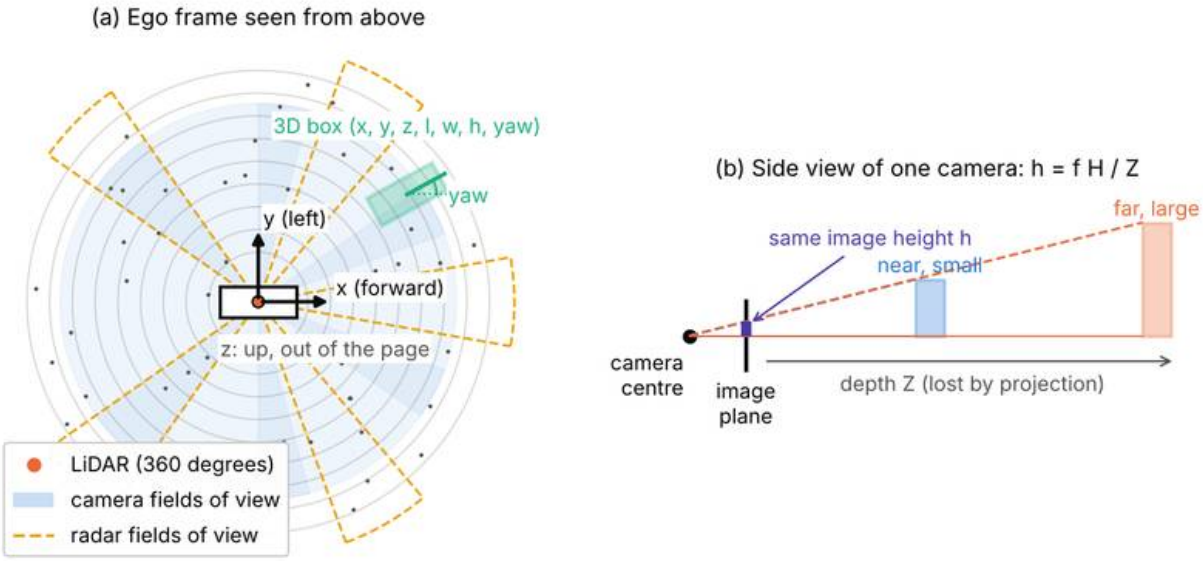


圖 *1* — (*a*) 從上方看的自車（*ego*，即本車）座標系：x 朝前、y 朝左、z 朝上，並標出一個光達、六台相機與數個雷達的視野，以及一個 *7-DoF* 框。(*b*) 單一相機為什麼會丟失深度：近處的小物體與遠處的大物體投影成相同的影像高度。

Waymo 資料集與本篇採用的慣例是右手系的**車輛座標系**（vehicle frame）：x 朝前、y 朝左、z 朝上；每個感測器有自己的座標系，一個稱為**外參**（extrinsics）的 4×4 矩陣把感測器座標映射到車輛座標系 [3]。用公式表示，感測器量到的一點 $\mathbf{p}_s$ 變成

$$\mathbf{p}_v = R\,\mathbf{p}_s + \mathbf{t},$$

其中 R 是 3×3 旋轉矩陣，$\mathbf{t}$ 是 3 維平移向量，兩者合起來就是該感測器的外參。相機另外還有**內參**（intrinsics）K，一個 3×3 矩陣，內含以像素為單位的焦距與影像中心。相機座標中的一點 $\mathbf{p}_c = (X, Y, Z)$（Z 指向鏡頭外）投影到像素

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z} K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix},$$

其中 (u, v) 是像素座標。除以 Z 是整件事的核心：射線 $\lambda(X, Y, Z)$ 上的每一點（$\lambda > 0$）都落在同一個像素上，所以深度就丟失了（圖 1，右）。校正在實務上很重要：如果 R 差了半度，60 m 外的一點會移動約半公尺，足以把行人的像素塗到他身後的牆上。

鳥瞰視角

白話版。鳥瞰視角由上方表示道路場景，方便在公尺空間比較物件大小與位置，但仍可能有垂直重疊。

在 BEV 中，我們捨棄 z，在 (x, y) 上保留一個二維網格，例如 0.16 m 的格子。高度變成特徵通道，而不是一條座標軸。因為物體在這個視角中保持公制尺寸、也不會互相遮擋，一個普通的二

維卷積偵測器就能在 BEV 網格上執行，下游的規劃器也是用這張地圖思考。代價是高度的細節被壓縮了；這對路上的車影響不大，對有桌子與架子的室內場景則影響較大。

點雲：網路實際看到的東西

白話版。點雲無固定順序且空間稀疏；遠處物件通常回波較少，表示方式須處理不均勻取樣。

一次光達掃描是一個集合 $P = \{\mathbf{p}_j\}_{j=1}^{M}$，其中 $\mathbf{p}_j = (x_j, y_j, z_j, r_j)$，$r_j$ 是強度，M 是點數，常常有數萬個。三個性質塑造了本篇的每一個網路：

1. **無序**。清單的任何排列描述的都是同一個場景，所以處理這些點的網路 f 必須是**排列不變**（permutation invariant）的：對索引的每個排列 π，都有 $f(\mathbf{p}_1, \dots, \mathbf{p}_M) = f(\mathbf{p}_{\pi(1)}, \dots, \mathbf{p}_{\pi(M)})$。
2. **稀疏**。點位於表面上。把車子周圍的空間切成 16 cm 的柱，超過 9/10 的柱是空的（我們在下面量測）。
3. **密度與距離有關**。旋轉式光達以固定的角度間隔 $\Delta\phi$（水平）與 $\Delta\epsilon$（垂直）發射射線。在距離 r 處，相鄰射線相隔約 $r\,\Delta\phi$ 與 $r\,\Delta\epsilon$，所以一個正對面積固定為 A 的物體大約會得到

$$n(r) \approx \frac{A}{r^2\,\Delta\phi\,\Delta\epsilon}$$

個點，其中 $n(r)$ 是點數的期望值，角度以弧度計。距離加倍，點數約減為四分之一。

下面的程式碼建立一個玩具旋轉式光達（32 條光束，每 0.2 度一條射線），並計算打到放在不同距離、大小如一輛車的框的射線數。它使用**平板法**（slab method）：如果一條射線在每一對平行面之間的區間互相重疊，它就打到了這個框。

```
import numpy as np

# A toy spinning LiDAR mounted 1.8 m above the ground: 32 beams between -16 and +10 degrees
# elevation, one ray every 0.2 degrees of azimuth (1800 rays per beam per turn).
elev = np.deg2rad(np.linspace(-16, 10, 32))
azim = np.deg2rad(np.arange(0, 360, 0.2))
E, A = np.meshgrid(elev, azim, indexing="ij")
dirs = np.stack([np.cos(E) * np.cos(A), np.cos(E) * np.sin(A), np.sin(E)], -1).reshape(-1, 3)
origin = np.array([0.0, 0.0, 1.8])

def hits_on_box(center, size):
    """Count rays whose first hit is an axis-aligned box (slab method)."""
    lo, hi = center - size / 2, center + size / 2
    with np.errstate(divide="ignore", invalid="ignore"):
        t1, t2 = (lo - origin) / dirs, (hi - origin) / dirs
    t_near = np.nanmax(np.minimum(t1, t2), axis=1)
    t_far = np.nanmin(np.maximum(t1, t2), axis=1)
    return int(np.sum((t_far >= t_near) & (t_far > 0)))

car = np.array([4.5, 1.8, 1.5])                      # length, width, height in metres
ranges = np.array([5, 10, 20, 40, 80])
counts = np.array([hits_on_box(np.array([r, 0, car[2] / 2]), car) for r in ranges])
for r, c in zip(ranges, counts):
    print(f"car at {r:2d} m: {c:5d} points")
```

```
slope = np.polyfit(np.log(ranges[1:]), np.log(counts[1:]), 1)[0]
print(f"log-log slope (10-80 m): {slope:.2f}")
```

```
car at  5 m:  2817 points
car at 10 m:   924 points
car at 20 m:   172 points
car at 40 m:    39 points
car at 80 m:     7 points
log-log slope (10-80 m): -2.33
```

點數從 5 m 處的數千個，掉到 80 m 處的 7 個。擬合出的斜率比 −2 略陡，因為在遠處，32 條光束中只有少數幾條會穿過 1.5 m 高的物體，垂直取樣變得很粗；擁有更多光束的真實感測器會更接近 $1/r^2$。圖 2 在整個場景上顯示同樣的效應：完全相同的車，證據卻天差地別。

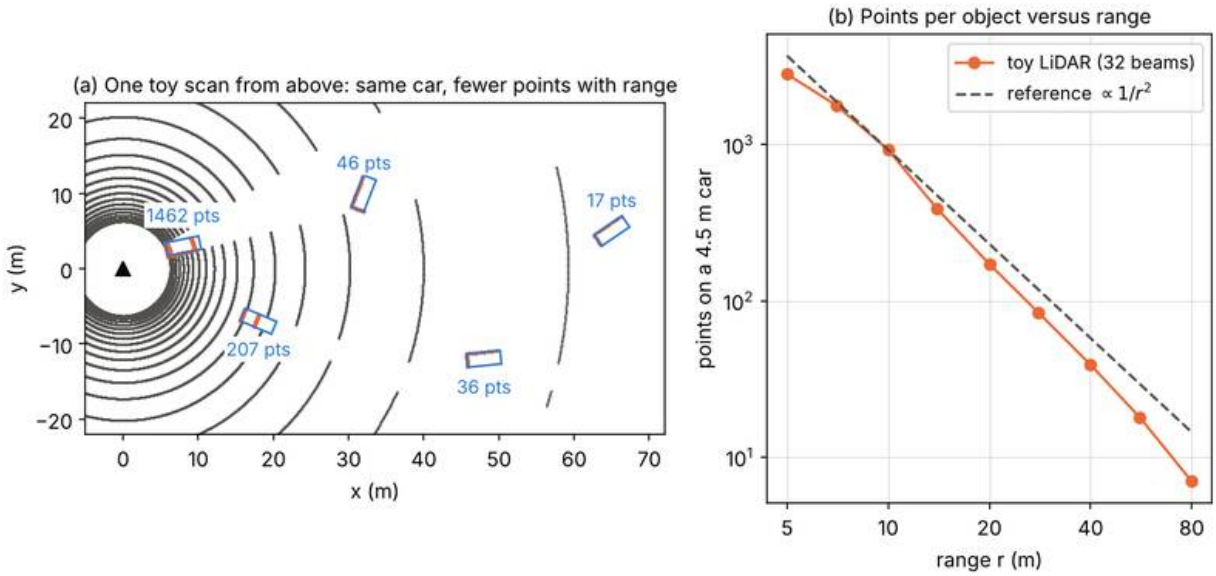


圖 2 — (*a*) 從上方看的玩具光線投射光達掃描：五輛相同的車，標上打到各車的點數。(*b*) 一輛 *4.5 m* 的車上的點數對距離的關係，附 $1/r^2$ 參考線。

傳統基準：地面移除、分群與框擬合

白話版。在深度學習之前，常見的光達流程是：把路面移除，把剩下的點分成一塊一塊，再在每一塊外面畫一個緊貼的矩形。它不需要訓練資料，也準確地告訴你後來的網路必須解決哪些問題。

三個步驟如下：

1. **用 RANSAC 移除地面** [5]。擬合一個平面 $\mathbf{n}^\top \mathbf{p} = d$（單位法向量 $\mathbf{n}$、偏移量 d）：重複隨機挑三個點、建出通過它們的平面，並計算**內點**（inlier）數，也就是滿足 $|\mathbf{n}^\top \mathbf{p} - d| < \tau$（$\tau$ 為容許

誤差）的點。內點最多的平面就是地面。（RANSAC 在 IP 12・特徵擷取的補充資料中有更詳細的說明。）

2. **用 DBSCAN 分群** [6]。在 BEV 中把剩下的點分組：兩點若距離小於 ε 就是鄰居，擁有至少 m 個鄰居的點是核心點，群從核心點長出去；孤立的點被標為雜訊。DBSCAN 不需要事先指定群數，適合物體數量未知的場景。（分群的一般介紹見 PR 07・分群、混合模型與 EM。）
3. **有向框擬合**。對每個群，嘗試各個偏航角 θ，把點旋轉到框的座標系，取緊貼的軸對齊矩形，並給它打分。沿著角度搜尋、配上擬合準則，就是搜尋式 L 形擬合（L-shape fitting）的想法 [7]。我們採用一個簡單的準則：每個點到最近矩形邊的平均距離。光達通常看到的是一個「L」（車子的兩個側面），而正確的偏航角會讓這些點落在兩條邊上。

```
import numpy as np
from sklearn.cluster import DBSCAN

rng = np.random.default_rng(0)

def box_surface(cx, cy, l, w, h, yaw, n):
    """n points on the sides of a yawed box that face the sensor at the origin."""
    c, s = np.cos(yaw), np.sin(yaw)
    pts = []
    for nx, ny, half, along in [(1, 0, l / 2, w), (-1, 0, l / 2, w), (0, 1, w / 2, l), (0,
     ↪ -1, w / 2, l)]:
        normal = np.array([c * nx - s * ny, s * nx + c * ny])
        mid = np.array([cx, cy]) + half * normal
        if normal @ mid >= 0:                       # face points away from the sensor: hidden
            continue
        t = rng.uniform(-along / 2, along / 2, n)
        tangent = np.array([-normal[1], normal[0]])
        xy = mid + t[:, None] * tangent
        pts.append(np.column_stack([xy, rng.uniform(0, h, n)]))
    return np.vstack(pts)

# ground plane z = 0 with slight noise, sampled uniformly in range and angle (density ~ 1/r)
r, a = rng.uniform(2, 60, 20000), rng.uniform(-np.pi / 2, np.pi / 2, 20000)
ground = np.column_stack([r * np.cos(a), r * np.sin(a), rng.normal(0, 0.03, r.size)])
objects = [(12, -4, 4.5, 1.8, 1.5, 0.3), (25, 6, 4.2, 1.7, 1.4, -1.0), (40, -10, 0.8, 0.6,
 ↪ 1.7, 0.0)]
cloud = [ground] + [box_surface(*o, n=int(40000 / np.hypot(o[0], o[1]) ** 2) + 5) for o in
 ↪ objects]
pts = np.vstack(cloud) + rng.normal(0, 0.02, (sum(len(c) for c in cloud), 3))

# 1) RANSAC plane fit: pick 3 points, count inliers within 0.1 m, keep the best plane
best = None
for _ in range(100):
    p = pts[rng.choice(len(pts), 3, replace=False)]
    n = np.cross(p[1] - p[0], p[2] - p[0])
    if np.linalg.norm(n) < 1e-9:
        continue
    n /= np.linalg.norm(n)
    inl = np.abs((pts - p[0]) @ n) < 0.1
    if best is None or inl.sum() > best.sum():
        best = inl
above = pts[~best]
print(f"{len(pts)} points, {best.sum()} ground inliers, {len(above)} left")
```

```
# 2) DBSCAN in the bird's-eye view
labels = DBSCAN(eps=0.7, min_samples=5).fit_predict(above[:, :2])

# 3) oriented box per cluster: search the yaw; score = mean distance of points to the nearest
 ↪  edge
for k in range(labels.max() + 1):
    q = above[labels == k]
    best_score, best_fit = np.inf, None
    for yaw in np.deg2rad(np.arange(0, 180, 1)):
        R = np.array([[np.cos(yaw), -np.sin(yaw)], [np.sin(yaw), np.cos(yaw)]])
        uv = q[:, :2] @ R                              # coordinates along the box axes
        lo, hi = uv.min(0), uv.max(0)
        edge_dist = np.minimum(np.minimum(uv - lo, hi - uv)[:, 0], np.minimum(uv - lo, hi -
         ↪  uv)[:, 1])
        if edge_dist.mean() < best_score and hi[0] - lo[0] >= hi[1] - lo[1]:   # x axis =
         ↪  long side
            best_score, best_fit = edge_dist.mean(), (R @ ((lo + hi) / 2), hi - lo,
             ↪  np.rad2deg(yaw))
    ctr, size, yaw = best_fit
    print(f"cluster {k}: {len(q):4d} pts, centre ({ctr[0]:5.1f}, {ctr[1]:5.1f}), "
          f"size {size[0]:.1f} x {size[1]:.1f} m, yaw {yaw:4.0f} deg, height {q[:,
           ↪  2].max():.1f} m")
```

```
20694 points, 19944 ground inliers, 750 left
cluster 0:  471 pts, centre ( 12.0,  -4.0), size 4.6 x 1.9 m, yaw   17 deg, height 1.5 m
cluster 1:  124 pts, centre ( 25.0,   6.1), size 4.1 x 1.9 m, yaw  123 deg, height 1.4 m
cluster 2:   53 pts, centre ( 40.0,  -9.9), size 0.8 x 0.6 m, yaw    0 deg, height 1.7 m
```

在這個合成場景上（真實物體：位於 $(12, -4)$、偏航角 17° 的一輛車，位於 $(25, 6)$、偏航角 $-57°$ 的一輛車（模 180° 即 123°），以及位於 $(40, -10)$、0.8×0.6 m 的一位行人），這個流程能把中心、尺寸與偏航角都還原得很好。圖 3 顯示結果。一個看似更簡單的準則，最小矩形面積，在這裡會失敗：L 形點集的凸包大致是一個三角形，沿著三角形長對角線的矩形能以相同的面積把它框住，所以面積準則可能選出偏差約 20° 的偏航角。

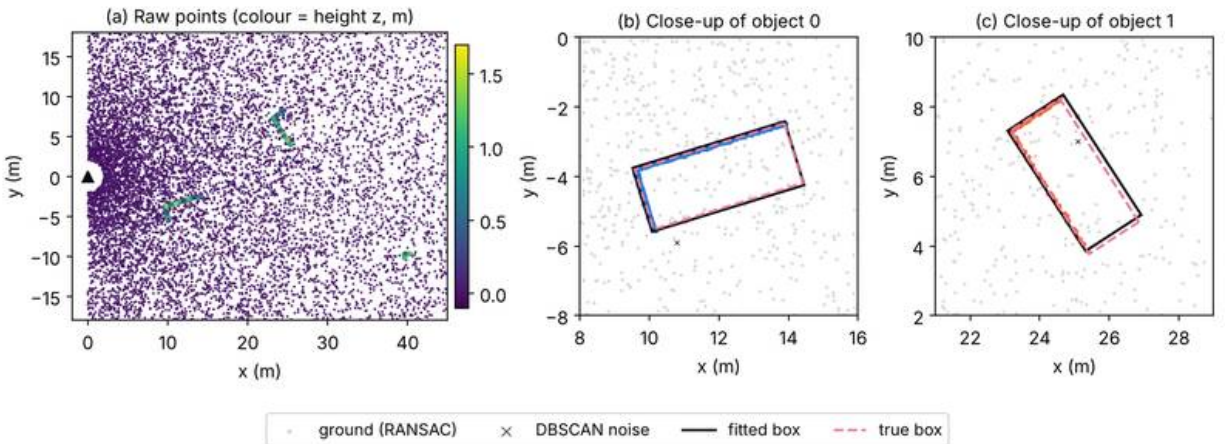


圖 3 — 傳統流程在合成場景上的結果。(*a*) 依高度著色的原始點。(*b, c*) *RANSAC* 地面移除與 *DBSCAN* 分群之後：物體點（彩色）、擬合的框（黑色）與真實的框（虛線）。光達只看到面向它的兩個側面。

這個基準也顯示了它的極限，而每一項極限都是學習的理由：

- **沒有語意**。一個群就只是「某個東西」。要分辨行人與電線桿、汽車與灌木，需要學習到的特徵。
- **只看到局部**。框只對可見的點擬合。從後面看到的遠處汽車只給出一條短線，所以它的長度未知。學習式偵測器用類別層級的尺寸先驗來補全看不到的部分。
- **合併與分裂**。兩個並肩走的人會合併成一個群；隔著縫隙看到的卡車會分裂成兩個。
- **朝向**。擬合只給出模 180° 的偏航角：它分不出車頭與車尾。

三維框：編碼、IoU 與 NMS

這些是 DLCV 03 基本元件的三維版本。我們只說明新的部分。

框的編碼與偏航角問題

白話版。和二維一樣，網路預測的是對一個參考框（錨框或一個點）的小幅修正。新的部分是角度：角度會繞回來，而且轉了 180° 的框涵蓋的空間完全相同，所以天真的角度損失會懲罰幾何上正確的答案。

VoxelNet [8]、SECOND [9] 與 PointPillars [10] 使用相同的殘差，作用在真值框（上標 g）與錨框（上標 a）之間：

$$\Delta x = \frac{x^g - x^a}{d^a}, \quad \Delta y = \frac{y^g - y^a}{d^a}, \quad \Delta z = \frac{z^g - z^a}{h^a}, \quad \Delta l = \log \frac{l^g}{l^a}, \quad \Delta w = \log \frac{w^g}{w^a}, \quad \Delta h = \log \frac{h^g}{h^a},$$

其中 $d^a = \sqrt{(l^a)^2 + (w^a)^2}$ 是錨框底面的對角線，用同一個長度把兩個方向的水平偏移正規化。這些就是 DLCV 03 的二維目標多加一個維度。

對於角度，VoxelNet 迴歸的是原始差值 $\theta^g - \theta^a$。SECOND [9] 指出這會造成衝突：預測正好差 π 時，框是一樣的，損失卻很大。它的**正弦誤差損失**（sine-error loss）

$$L_\theta = \text{SmoothL1}(\sin(\theta^p - \theta^t))$$

（θ^p 是預測角、θ^t 是目標角，SmoothL1 是 DLCV 03 的平滑 L1 損失：誤差小於 1 時是二次，大於 1 時是線性）在預測完全正確與預測翻轉了 π 時都是零。因為它分不出前後，SECOND 加了一個很小的**方向分類器**（direction classifier），有兩個分格，用來決定框指向哪一邊 [9]；PointPillars 使用同樣這一對損失 [10]。CenterPoint 則迴歸 $(\sin\theta, \cos\theta)$，再用 $\theta = \text{atan2}(\sin\theta, \cos\theta)$ 還原角度，這既連續又沒有歧義 [11]。

僅線上版：偏航角編碼比較：原始角度、正弦誤差與 (*sin*, *cos*) — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/notes/偏航角編碼比較原始角度正弦誤差與-sin-cos/*

旋轉框的 BEV IoU 與三維 IoU

白話版。 IoU 仍然是「重疊面積除以總面積」，但框現在是旋轉的，所以重疊部分是一個多邊形，不是矩形。在三維中，再把這個多邊形的面積乘上兩個高度區間的重疊長度。

對兩個直立的框 A 與 B，令 $A_{\text{bev}} \cap B_{\text{bev}}$ 為它們 BEV 矩形的交集（一個最多有 8 個角的凸多邊形），並令

$$o_z = \max\!\Big(0,\ \min\left(z_A + \tfrac{h_A}{2},\, z_B + \tfrac{h_B}{2}\right) - \max\left(z_A - \tfrac{h_A}{2},\, z_B - \tfrac{h_B}{2}\right)\Big)$$

為它們垂直範圍的重疊長度，其中 z 是框的中心高度，h 是它的高度。於是

$$\mathrm{IoU}_{\text{bev}} = \frac{|A_{\text{bev}} \cap B_{\text{bev}}|}{l_A w_A + l_B w_B - |A_{\text{bev}} \cap B_{\text{bev}}|}, \qquad \mathrm{IoU}_{\text{3D}} = \frac{|A_{\text{bev}} \cap B_{\text{bev}}|\, o_z}{l_A w_A h_A + l_B w_B h_B - |A_{\text{bev}} \cap B_{\text{bev}}|\, o_z},$$

其中 $|\cdot|$ 表示面積，l, w, h 是框的尺寸。兩個凸多邊形的交集多邊形，可以用 **Sutherland–Hodgman 裁切法**（clipping）[12] 算出：依序用第二個多邊形的每條邊所在的直線去切第一個多邊形，只保留內側的部分。接著，多邊形的面積由它各角 (x_k, y_k) 的**鞋帶公式**（shoelace formula）$\frac{1}{2}\left|\sum_k (x_k y_{k+1} - x_{k+1} y_k)\right|$ 得到，其中索引循環取值。

```
import numpy as np

def corners(box):
    """BEV corners (counter-clockwise) of box = (x, y, z, l, w, h, yaw); z is the box
     ↪  centre."""
    x, y, _, l, w, _, yaw = box
    c, s = np.cos(yaw), np.sin(yaw)
    local = np.array([[l, w], [-l, w], [-l, -w], [l, -w]]) / 2
    return local @ np.array([[c, s], [-s, c]]) + [x, y]

def clip(subject, clipper):
    """Sutherland-Hodgman: keep the part of polygon `subject` inside convex CCW polygon
     ↪  `clipper`."""
    out = list(subject)
    for i in range(len(clipper)):
        a, b = clipper[i], clipper[(i + 1) % len(clipper)]
        inside = lambda p: (b[0] - a[0]) * (p[1] - a[1]) - (b[1] - a[1]) * (p[0] - a[0]) >= 0
        pts, out = out, []
        for j in range(len(pts)):
            p, q = pts[j], pts[(j + 1) % len(pts)]
            if inside(q):
                if not inside(p):
                    out.append(cross_point(p, q, a, b))
                out.append(q)
            elif inside(p):
                out.append(cross_point(p, q, a, b))
        if not out:
            break
    return np.array(out)

def cross_point(p, q, a, b):
    """Intersection of segment p-q with the infinite line through a and b."""
    d1, d2 = q - p, b - a
```

```
    t = ((a[0] - p[0]) * d2[1] - (a[1] - p[1]) * d2[0]) / (d1[0] * d2[1] - d1[1] * d2[0])
    return p + t * d1

def area(poly):                                  # shoelace formula
    if len(poly) < 3:
        return 0.0
    x, y = poly[:, 0], poly[:, 1]
    return 0.5 * abs(x @ np.roll(y, -1) - y @ np.roll(x, -1))

def iou_bev_3d(a, b):
    inter = area(clip(corners(a), corners(b)))
    bev = inter / (a[3] * a[4] + b[3] * b[4] - inter)
    dz = max(0.0, min(a[2] + a[5] / 2, b[2] + b[5] / 2) - max(a[2] - a[5] / 2, b[2] - b[5] /
     ↪  2))
    inter3 = inter * dz
    return bev, inter3 / (a[3] * a[4] * a[5] + b[3] * b[4] * b[5] - inter3)

car = (0, 0, 0, 4, 2, 1.5, 0)
cases = {"identical":                (car, car),
         "rotated by 180 deg":       (car, (0, 0, 0, 4, 2, 1.5, np.pi)),
         "rotated by 90 deg":        (car, (0, 0, 0, 4, 2, 1.5, np.pi / 2)),
         "unit square vs 45 deg":    ((0, 0, 0, 1, 1, 1, 0), (0, 0, 0, 1, 1, 1, np.pi / 4)),
         "shifted 1 m along length": (car, (1, 0, 0, 4, 2, 1.5, 0)),
         "same BEV, raised 0.5 m":   (car, (0, 0, 0.5, 4, 2, 1.5, 0)),
         "far apart":                (car, (10, 0, 0, 4, 2, 1.5, 0.3))}
for name, (a, b) in cases.items():
    bev, iou3 = iou_bev_3d(np.array(a, float), np.array(b, float))
    print(f"{name:26s} BEV IoU {bev:.4f}   3D IoU {iou3:.4f}")
```

```
identical                  BEV IoU 1.0000   3D IoU 1.0000
rotated by 180 deg         BEV IoU 1.0000   3D IoU 1.0000
rotated by 90 deg          BEV IoU 0.3333   3D IoU 0.3333
unit square vs 45 deg      BEV IoU 0.7071   3D IoU 0.7071
shifted 1 m along length   BEV IoU 0.6000   3D IoU 0.6000
same BEV, raised 0.5 m     BEV IoU 1.0000   3D IoU 0.5000
far apart                  BEV IoU 0.0000   3D IoU 0.0000
```

每個值都與手算相符：4×2 的框對上自己旋轉 90° 後的框，重疊成一個 2×2 的正方形，所以 $\text{IoU}=4/(8+8-4)=1/3$；單位正方形對上自己旋轉 45° 後的版本，重疊成一個面積為 $2(\sqrt{2}-1)$ 的正八邊形，得到 $\sqrt{2}/2\approx 0.7071$；抬高了自身高度三分之一的框，BEV 重疊完整保留，三維重疊卻只剩一半。請注意第二行：轉了 180° 的框，IoU 是 1。IoU 對朝向的方向是盲的，這正是 Waymo 的指標要加上朝向項的原因（見下文）。

圖 4（右）顯示偏航角誤差對細長物體有多嚴苛。在中心與尺寸都完美的情況下，車子大小的框在偏航角誤差約 17° 時，IoU 就掉到 0.7 以下；巴士大小的框約在 8.5°；而方形的行人框永遠不會低於 0.7。

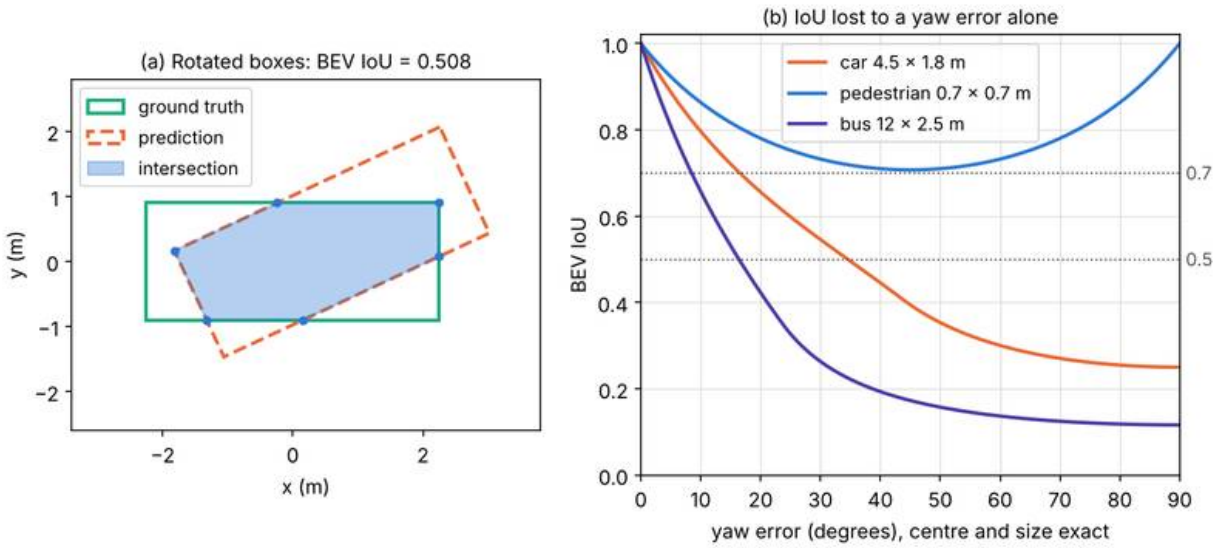


圖 4 — (*a*) 兩個旋轉的框；它們的交集是一個用裁切法求出的多邊形。(*b*) 單單偏航角誤差所損失的 *BEV IoU*，針對三種物體形狀，中心與尺寸皆精確。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

旋轉 NMS 及其替代方案

白話版。移除重複的框，做法和二維相同，但「重疊」現在必須在旋轉的框之間量測，這比較慢。兩種捷徑很常見：用軸對齊的框近似重疊，或是用中心峰值或集合預測，完全避開 NMS。

三維中的貪婪 NMS 依分數排序，並移除與某個已保留的框之間 BEV IoU 超過門檻的框，使用上面的旋轉 IoU。因為道路上的物體在 BEV 中很少重疊，門檻可以設得低。PointPillars 發現，對軸對齊的 BEV 框用門檻 0.5 做 NMS，結果與旋轉 NMS 相近，卻快得多 [10]。CenterPoint 從中心熱圖的局部極大值讀出物體，就像 CenterNet [13] 一樣，這在設計上就移除了大部分重複；論文仍在第二階段之前對這些峰值套用 NMS [11]。DETR 家族 [14] 的相機方法（例如 DETR3D）直接預測一組框，完全不需要 NMS [15]。

以光達為基礎的偵測器

白話版。光達偵測器的差別，主要在於它們如何把一個無序的點集變成網路能處理的東西：保留點本身（以點為基礎）、把點丟進三維網格（體素）、把點丟進垂直的柱（pillar），或是混用。在那之後，大多數方法會在 BEV 上使用二維偵測頭。

以點為基礎：PointNet、PointNet++ 與 PointRCNN

白話版。用同一個小網路處理每一個點，再用取最大值把所有點彙整起來。最大值不在乎輸入的順序，所以結果是排列不變的。

PointNet [16] 計算

$$f(\mathbf{p}_1, \dots, \mathbf{p}_M) = \gamma\Big(\max_{j=1,\dots,M} h(\mathbf{p}_j) \Big),$$

其中 h 是套用在每個點上的共用多層感知器，max 對所有點逐元素取最大（這是對稱函數，所以順序無關緊要），γ 是第二個網路，把彙整後的特徵映射到輸出。它先單獨看每個點，再一次看整個集合，所以會錯過局部結構。**PointNet++** [17] 用**集合抽象**（set abstraction）層修正這一點：用**最遠點取樣**（farthest point sampling，每個新的中心是離已選中心最遠的點）選出一個中心點子集，收集每個中心半徑內的鄰居（**球查詢**，ball query），再對每個鄰域執行一個小型 PointNet。把這些層疊起來，就得到一個局部特徵的階層，就像 CNN 逐步擴大的感受野。兩者主要都是骨幹網路：它們產生特徵，偵測頭另外建在上面。

PointRCNN [18] 是直接在點上運作的二階段偵測器。第一階段用 PointNet++ 骨幹把場景分割成前景點與背景點，並由下而上地從每個前景點提出一個三維框。第二階段在每個提議內部做點的池化，把它們轉換到提議自己的**標準座標**（canonical coordinates，經過平移與旋轉，使框與座標軸對齊），再精修這個框。以點為基礎的偵測器對小物體很準確，但在大型點雲上的取樣與鄰居搜尋很慢。

以體素為基礎：VoxelNet 與 SECOND

白話版。把空間切成小的三維格子（體素），用一個小型 PointNet 彙整每個格子裡的點，再對得到的三維網格做卷積。由於幾乎所有體素都是空的，只在有點的地方計算。

VoxelNet [8] 依體素把點分組，並讓每個點帶上它相對於體素質心的偏移，得到 7 維輸入 $(x, y, z, r, x - v_x, y - v_y, z - v_z)$，其中 (v_x, v_y, v_z) 是該體素內各點的平均。一疊**體素特徵編碼**（voxel feature encoding，VFE）層，每層是一個共用的全連接層，後接體素內的逐元素最大池化，把每個體素變成一個特徵向量。接著三維卷積處理這個網格，再由 Faster R-CNN [19] 風格的區域提議網路在 BEV 特徵圖上預測框。它是端到端的，而且很強，但稠密的三維卷積讓它很慢。

SECOND [9] 以**稀疏卷積**（sparse convolution）取代稠密的三維卷積，只儲存與計算有被占用的體素。它多半使用**子流形**（submanifold）稀疏卷積 [20]，其輸出只在輸入為活躍的地方才活躍，所以被占用位置的集合不會一層一層地長大。SECOND 也引入了正弦誤差角度損失與真值取樣資料增強（兩者都在本篇中說明），而這類稀疏體素骨幹後來被 PV-RCNN [21] 等偵測器採用。

僅線上版：稀疏卷積與子流形卷積 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/notes/稀疏卷積與子流形卷積/*

以柱為基礎：PointPillars

白話版。讓格子無限高。每一根垂直的柱（**pillar**）由一個小型 PointNet 彙整，結果是一張平面的「偽影像」，普通的二維 CNN 就能很快地處理它。

PointPillars [10] 有三個階段：柱特徵網路、二維卷積骨幹，以及 SSD 風格的偵測頭 [22]。柱中的每個點被裝飾成 $D = 9$ 維：(x, y, z, r)、它相對於柱內各點平均值的偏移 (x_c, y_c, z_c)，以及它在 x–y 平面上相對於柱中心的偏移 (x_p, y_p)。非空的柱被打包成形狀為 (D, P, N) 的稠密張量，每個樣本最多 P 根柱、每根柱最多 N 個點（論文在 0.16 m 網格下用 12,000 與 100），不足的補零，過

多的隨機子取樣。對每個點套用一個線性層、批次正規化與 ReLU，對 N 個點取最大值，每根柱得到一個 C 維特徵，再把這些特徵**散佈**（scatter）回它們的網格位置，形成 $C \times H \times W$ 的偽影像。論文報告完整流程為 62 Hz，較快的變體為 105 Hz [10]。

僅線上版：*PointPillars*：閱讀指南 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/papers/pointpillars 閱讀指南/*

下面的程式碼在一個玩具點雲上執行這個流程的輸入端：分格、9 維裝飾、稠密的 (P, N, D) 張量，以及散佈（PointNet 的權重在這裡是隨機的；真實模型中則是學出來的）。

```
import numpy as np

rng = np.random.default_rng(0)
# toy cloud: ground sampled uniformly in range and angle (so density falls as 1/r) + two
 ↪  "cars"
r, a = rng.uniform(2, 70, 10000), rng.uniform(-0.5, 0.5, 10000)
ground = np.column_stack([r * np.cos(a), r * np.sin(a), rng.normal(-1.7, 0.03, r.size)])
cars = [c + rng.uniform([-2.2, -0.9, -1.7], [2.2, 0.9, -0.2], (n, 3))
        for c, n in [(np.array([10.0, 2.0, 0.0]), 3000), (np.array([35.0, -6.0, 0.0]), 60)]]
xyz = np.vstack([ground] + cars)
pts = np.column_stack([xyz, rng.uniform(0, 1, len(xyz))])          # x, y, z, reflectance

# 1) bin into a bird's-eye-view grid of 0.16 m pillars over x in [0, 70.4), y in [-40, 40)
vx, x0, y0, W, H = 0.16, 0.0, -40.0, 440, 500
ix = np.floor((pts[:, 0] - x0) / vx).astype(int)
iy = np.floor((pts[:, 1] - y0) / vx).astype(int)
ok = (ix >= 0) & (ix < W) & (iy >= 0) & (iy < H)
pts, ix, iy = pts[ok], ix[ok], iy[ok]
pillar_id, inv, counts = np.unique(iy * W + ix, return_inverse=True, return_counts=True)
print(f"{len(pts)} points -> {len(pillar_id)} non-empty pillars out of {W * H} "
      f"({100 * (1 - len(pillar_id) / (W * H)):.1f}% empty); max {counts.max()} points in one
       ↪  pillar")

# 2) per-point 9-D features: x, y, z, r, offsets to the pillar mean, offsets to the pillar
 ↪  centre
mean = np.zeros((len(pillar_id), 3))
np.add.at(mean, inv, pts[:, :3])
mean /= counts[:, None]
centre = np.column_stack([(ix + 0.5) * vx + x0, (iy + 0.5) * vx + y0])
feat = np.column_stack([pts, pts[:, :3] - mean[inv], pts[:, :2] - centre])

# 3) dense tensor (P pillars, N points, D=9): pad with zeros, keep at most N points per
 ↪  pillar
P, N = len(pillar_id), 16          # PointPillars caps P at 12000 and N at 100, sampling at
 ↪  random
order = np.argsort(inv, kind="stable")
slot = np.arange(len(inv)) - np.searchsorted(inv[order], inv[order])   # rank inside its
 ↪  pillar
keep = slot < N
dense = np.zeros((P, N, 9))
dense[inv[order][keep], slot[keep]] = feat[order][keep]
print("dense pillar tensor:", dense.shape, "| points dropped by the cap N:",
 ↪  int((~keep).sum()))
```

```
# 4) a one-layer 'PointNet' (random weights here): linear + ReLU per point, max over points,
 ↪  scatter
Wt = rng.normal(0, 0.3, (9, 64))
pillar_feat = np.maximum(dense @ Wt, 0).max(axis=1)                    # (P, 64)
canvas = np.zeros((64, H * W))
canvas[:, pillar_id] = pillar_feat.T
print("pseudo-image:", canvas.reshape(64, H, W).shape)
```

```
13060 points -> 9481 non-empty pillars out of 220000 (95.7% empty); max 24 points in one
 ↪  pillar
dense pillar tensor: (9481, 16, 9) | points dropped by the cap N: 23
pseudo-image: (64, 500, 440)
```

在這個玩具中，22 萬個格子有超過 95% 是空的，接近 PointPillars 論文針對相同網格大小下真實 64 線掃描所報告的約 97% 稀疏度 [10]。這就是為什麼輸入步驟需要多少記憶體，取決於柱化與散佈，而不是卷積。圖 5 在一次光線投射的掃描上顯示這個網格。

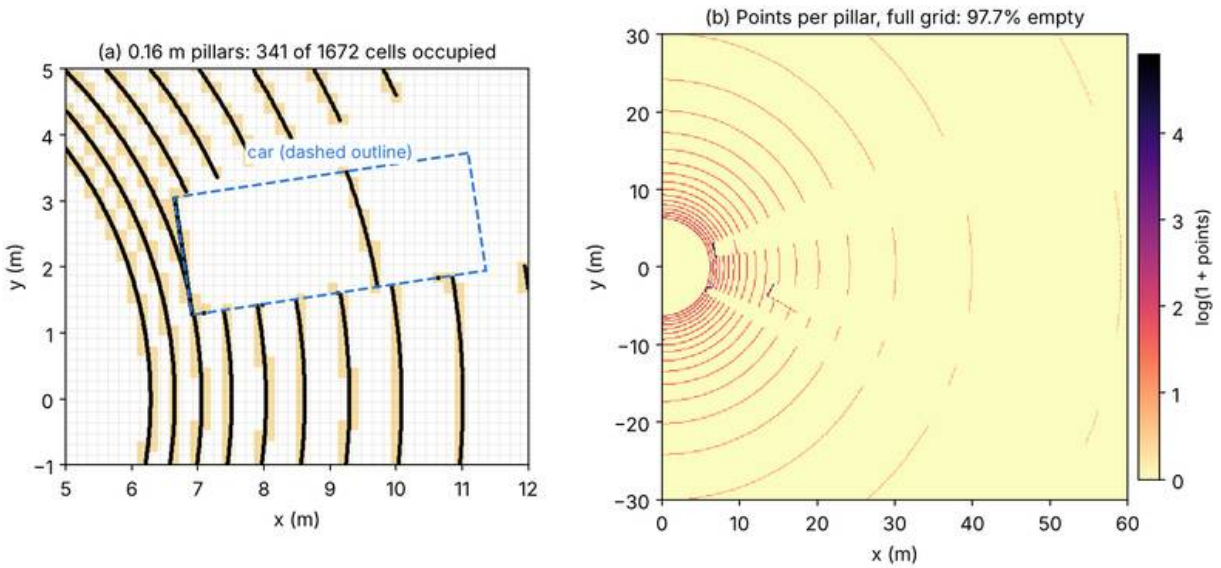


圖 *5* — 對一次玩具光線投射掃描做柱化。(*a*) 特寫中被占用的 *0.16 m* 柱（加陰影）。(*b*) 整個網格上每根柱的點數：*97.7%* 的柱是空的，被占用的柱隨距離變得稀疏。

以中心為基礎的偵測頭：CenterPoint

白話版。不使用多種朝向的錨框，而是把物體的中心當成 BEV 熱圖中的峰值找出來，再從峰值處的特徵讀出其他所有屬性。中心沒有朝向，所以錨框的旋轉問題就消失了。

CenterPoint [11] 保留標準的體素或柱骨幹，並以 CenterNet [13] 精神的中心偵測頭取代錨框偵測頭。熱圖每個類別一個通道，用焦點損失 [23] 對著畫在投影後框中心的高斯峰值訓練；因為物體在 BEV 中很小，高斯半徑被放大到至少 2 個格子。在每個中心，迴歸頭預測次體素偏移、離

地高度、三維尺寸、以 $(\sin\theta, \cos\theta)$ 表示的偏航角，以及二維速度。速度來自餵入連續兩次掃描，並讓追蹤變成在幀間對中心做貪婪的最近點配對。一個輕量的第二階段在框各面的中心取樣特徵，預測精修後的框，以及一個朝著該框與真值的三維 IoU 訓練的信心值。以中心為基礎的偵測頭也擴展到光達之外：例如相機偵測器 BEVDet 就使用 CenterPoint 的偵測頭 [24]。

點–體素混合：PV-RCNN

白話版。體素快，並能給出好的提議；點則保有精確的位置。用體素來提議，用點來精修。

PV-RCNN [21] 先執行稀疏三維體素 CNN 取得特徵與提議，再把多尺度的體素特徵彙整成一小組**關鍵點**（keypoints，即體素集合抽象）。對每個提議，**RoI 網格池化**（RoI-grid pooling）在框內的一個規則點網格上、以數種半徑收集關鍵點特徵，再由精修頭預測最終的框與信心值。在 Transformer 骨幹出現之前，這類二階段混合方法是最準確的光達偵測器之一。

點上的 Transformer 與序列模型

白話版。一輛車只占 150 m 場景中極小的一部分，所以像二維網路那樣縮小特徵圖，會丟掉細節。如果把注意力限制在被占用的體素上，它就能讓特徵在不縮小的情況下長距離互動。

SST [25] 觀察到降取樣在光達偵測中帶來的好處很小，因此全程只用單一步幅，並在鄰近的被占用體素之間使用注意力，來取回降取樣原本會提供的感受野；論文報告這對行人等小物體幫助最大。**DSVT** [26] 重新組織這種注意力，使它能在 GPU 上完全平行地執行，也容易部署。**LION** [27] 與 **Voxel Mamba** [28] 以線性時間的序列模型（線性遞迴網路與狀態空間模型）取代注意力，把體素當作長序列來讀。唯一要記住的一點是：這些骨幹用來換取長距離脈絡的成本，隨被占用體素的數量成長，而不是隨場景的大小成長。

僅線上版：稀疏體素上的注意力：視窗、集合與序列 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/notes/稀疏體素上的注意力視窗集合與序列/*

純相機的三維偵測

白話版。相機便宜，看得到顏色、紋理與標誌，但照片是平的。每個相機偵測器都必須回答同一個問題：它看到的每樣東西有多遠？本節先說明為什麼這很困難，再介紹從影像建出三維地圖的兩種主要方式：把像素推出到三維，或把影像特徵拉到三維位置。

為什麼單一相機不夠：深度歧義

白話版。近距離的一輛玩具車，與遠處的一輛真車，在照片上可能看起來一模一樣。要從單張影像得到深度，網路必須依賴一些線索，例如物體的典型尺寸、它接觸路面的位置，以及透視，而這些線索中的小誤差，到了遠處會變成以公尺計的大誤差。

對於焦距為 f（以像素計）的針孔相機，一個真實高度為 H、深度為 Z 的物體，在影像中高 $h = fH/Z$ 像素，所以 $Z = fH/h$。微分得到

$$\left|\frac{\partial Z}{\partial h}\right| = \frac{fH}{h^2} = \frac{Z^2}{fH},$$

所以一個像素的量測誤差所造成的深度誤差，隨距離的平方成長，而假設的尺寸 H 若有誤差，深度會被乘上相同的倍數。

```
import numpy as np

f, H = 1000.0, 1.5                      # focal length in pixels, true car height in metres
for Z in [10, 20, 40, 80]:
    h = f * H / Z                       # image height of the car in pixels
    z_pix = f * H / (h - 1)             # depth if we mis-measure the height by one pixel
    z_size = f * (1.1 * H) / h          # depth if we assume a car 10% taller than it is
    print(f"Z = {Z:2d} m: car is {h:6.2f} px tall | 1 px error -> {z_pix - Z:5.2f} m | "
          f"10% size error -> {z_size - Z:4.1f} m")
```

```
Z = 10 m: car is 150.00 px tall | 1 px error ->  0.07 m | 10% size error ->  1.0 m
Z = 20 m: car is  75.00 px tall | 1 px error ->  0.27 m | 10% size error ->  2.0 m
Z = 40 m: car is  37.50 px tall | 1 px error ->  1.10 m | 10% size error ->  4.0 m
Z = 80 m: car is  18.75 px tall | 1 px error ->  4.51 m | 10% size error ->  8.0 m
```

在 80 m 處，單單一個像素就能讓車子移動約 4.5 m，遠遠超出三維 IoU 0.7 能容忍的範圍。這就是為什麼單眼三維偵測器在遠處物體上遠遠落後光達。像 **FCOS3D** [29] 這樣的單眼方法，改造二維一階段偵測器（FCOS [30]），在投影後的三維中心迴歸深度、尺寸與朝向。**Pseudo-LiDAR** [31] 提出了一個有影響力的觀點：影像式與光達式偵測器之間的差距，很大一部分來自表示方式，而不只是深度品質；把估計出的深度圖轉成三維點雲，再對它執行光達偵測器，效果遠比把深度圖當成影像來處理好。

把影像提升到鳥瞰視角

白話版。每個像素不知道自己的深度，所以給它一個深度的機率分布，在每個可能的深度放一份它的特徵副本、以該機率加權，然後把所有這些三維特徵丟到 BEV 網格上。結果可以跨相機融合，並像光達的 BEV 圖那樣處理。

Lift-Splat-Shoot（LSS）[32] 把每張影像「提升」成特徵的視錐。在每個像素，網路預測一個脈絡向量 $\mathbf{c} \in \mathbb{R}^C$，以及一個在離散深度集合 $\mathcal{D}$ 上的分布 $\boldsymbol{\alpha}$（$\alpha_d \geq 0$，$\sum_{d \in \mathcal{D}} \alpha_d = 1$）。放在像素射線上深度 d 處的特徵是

$$\mathbf{c}_d = \alpha_d \, \mathbf{c},$$

也就是深度機率與脈絡的外積。有把握的深度把特徵放在一個地方；不確定的深度則把它沿著射線散開。「splat」步驟用相機外參與內參，把視錐中所有的點轉換到車輛座標系，再以加總池化放進 BEV 的柱中，並用累積和的技巧讓池化變快 [32]。論文評估的是 BEV 分割與規劃；**BEVDet** [24] 則用 BEV 編碼器與以中心為基礎的偵測頭，加上同時在影像與 BEV 空間進行的資料增強，把同一個視角轉換變成三維偵測器。**BEVDepth** [33] 發現只從偵測損失學到的深度分布品質很差，於是用投影到影像中的光達點所提供的深度明確監督它，並搭配一個感知相機參數的深度網路。

僅線上版：*Lift-Splat-Shoot*：閱讀指南 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/papers/lift-splat-shoot 閱讀指南/*

從三維回頭查詢影像：DETR3D、PETR 與 BEVFormer

白話版。反過來走。從三維的猜測出發，無論是物體查詢還是一個 BEV 格子的網格，用已知的相機矩陣把它們投影到影像中，再取出它們落點上的影像特徵。不需要深度圖。

DETR3D [15] 把每個物體查詢解碼成一個三維參考點，投影到所有相機中，以雙線性內插在那裡取樣影像特徵，並用它們精修查詢；訓練使用 DETR 的集合對集合損失 [14]，所以不需要 NMS。**PETR** [34] 則把三維位置編碼進影像特徵本身：把相機視錐離散成數個深度上的點，用相機參數轉換到三維，再通過一個小型網路產生三**維位置嵌入**（3D position embedding），加到影像特徵上；然後普通的 DETR 式查詢去關注這些具位置感知的特徵。**BEVFormer** [35] 保留一個由可學習的 **BEV 查詢**組成的網格，每個 BEV 格子一個。空間交叉注意力讓每個 BEV 查詢，用可變形注意力 [36]，在它的位置所投影到的影像區域中取樣特徵；時間自注意力則讓它去看前一幀的 BEV 特徵，這對速度和暫時被遮住的物體有幫助。它的 BEV 圖支援偵測與地圖分割的偵測頭。

事實證明，時間是相機深度的強力來源：幀與幀之間的運動就是一種立體視覺。**StreamPETR** [37] 在線上把物體查詢從一幀傳到下一幀，並報告在 nuScenes 上，純相機的結果可與光達方法相提並論；**Far3D** [38] 則以 2D 偵測產生的 3D 查詢，瞄準長距離的相機偵測。

融合光達、相機與雷達

白話版。光達擅長「在哪裡」，相機擅長「是什麼」。融合把兩者結合起來，而主要的設計問題是在哪裡結合：在原始點、在提議、在共用的 BEV 圖，或是透過注意力。

我們依融合發生的位置把方法分組；Wang 等人 [39] 給出更細緻的分類。

- **點層級（早期）融合。** **PointPainting** [40] 把每個光達點投影到影像語意分割結果上，並把類別分數附加到該點；然後任何光達偵測器都能在這個「上色」的點雲上執行。
- **提議層級融合。** **MV3D** [41] 從點雲的 BEV 圖產生三維提議，並融合來自 BEV、光達前視圖與影像的區域特徵。**Frustum PointNets** [42] 則由 2D 走向 3D：2D 偵測器在影像中找到物體，PointNet 只處理那個 2D 框後方視錐內的點。
- **BEV 層級融合。**兩個都叫 **BEVFusion** 的方法，把相機特徵轉成 BEV 圖（用 LSS 風格的視角轉換），再與光達 BEV 圖結合。其中一個來自 MIT [43]，以串接與卷積編碼器融合，支援偵測與地圖分割，並把相機到 BEV 的池化加速了 40 倍以上。另一個 [44] 讓相機分支獨立於光達，因此光達故障時偵測器仍能運作，並報告在模擬的光達故障下有很大的增益。
- **查詢層級融合。** **TransFusion** [45] 先從光達預測物體查詢，再用 Transformer 解碼器讓每個查詢去關注影像特徵。這種透過注意力的「軟」關聯，而不是把點硬投影到像素，讓它對不佳的影像條件與校正誤差更穩健。**UniTR** [46] 更進一步，用一個共用的 Transformer 骨幹處理相機與光達的 token。

雷達。雷達帶來速度資訊，以及對天氣的穩健性。**CenterFusion** [47] 在影像中偵測物體中心，透過圍繞每個偵測的視錐把雷達回波與它關聯起來，再用雷達特徵改善深度、旋轉與速度，使相機偵測器的 nuScenes 偵測分數大幅提高。

訓練：指派與資料增強

白話版。訓練三維偵測器和訓練二維很像，只有兩個轉折：指派使用旋轉的 BEV 重疊或中心熱圖；而資料增強可以把物體從一個場景切出來、貼進另一個場景，這在照片上不可能做得令人信服。

指派。以錨框為基礎的偵測頭在每個 BEV 格子放幾個錨框（每個類別一種尺寸，通常偏航角為 0° 與 90°），並依 BEV IoU 指派，就像 DLCV 03 的二維規則。例如 VoxelNet 的車輛錨框，BEV IoU 高於 0.6 為正樣本、低於 0.45 為負樣本 [8]。因為三維場景中的錨框遠多於物體，這些偵測頭使用焦點損失 [23]。以中心為基礎的偵測頭則以高斯熱圖目標取代 IoU 指派 [11]。

資料增強。光達場景數量不多（KITTI 有 7,481 筆訓練掃描 [48]），每個場景裡的物體更少。標準的資料增強有 [8], [9], [10]：

- **真值取樣**（ground-truth sampling）。把所有標註框以及框內的點建成資料庫，再把隨機挑選的物體貼進每個訓練場景 [9]。PointPillars 在 KITTI 上每個場景取樣 15 輛車、0 位行人與 8 位單車騎士 [10]。VoxelNet 在擾動框時就已使用碰撞測試，來剔除物理上不可能的結果 [8]，貼上時也需要相同的檢查。
- **逐物體擾動**。把每個框連同它的點一起旋轉與平移；VoxelNet 的旋轉角從 $[-\pi/10, \pi/10]$ 抽取，平移從標準差為 1 m 的零平均高斯分布抽取 [8]。
- **全域轉換**。繞垂直軸旋轉整個場景（VoxelNet 的角度從 $[-\pi/4, \pi/4]$ 抽取，模擬車子在轉彎）、以 $[0.95, 1.05]$ 中的倍率縮放，以及左右翻轉 [8], [10]。

這些做法很便宜，因為點雲不過就是一堆座標：把點與框一起旋轉，標註就能保持精確。麻煩在於跨感測器的一致性：貼進光達掃描的物體並不在相機影像裡，所以融合模型需要格外小心。

僅線上版：真值取樣細說 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/notes/真值取樣細說/*

資料集與指標

白話版。三個公開基準塑造了這個領域。KITTI 規模小、只看前方，指標是嚴格的三維 IoU。nuScenes 與 Waymo 規模大、涵蓋 360 度；nuScenes 以中心距離配對框並加上誤差項，Waymo 則依朝向吻合的程度，對每個正確的框加權。

資料集	感測器（取自論文）	規模（取自論文）	主要偵測指標
KITTI 3D [49], [48]	兩組立體相機（灰階與彩色）、一個 Velodyne HDL-64E 光達	7,481 筆訓練與 7,518 筆測試的影像與點雲；80,256 個標註物體	IoU 0.7（車）／0.5（行人、單車）的 3D AP，分簡單／中等／困難
nuScenes [4], [50]	6 台相機、5 個雷達、1 個光達，360°	1,000 個 20 秒的場景；23 個類別與 8 種屬性（偵測用 10 個類別）	以中心距離配對的 mAP；NDS
Waymo Open [3], [51]	5 個光達、5 台相機	1,150 個 20 秒的場景；約 1,200 萬個標註的三維光達物體	LEVEL_1 ／ LEVEL_2 的 mAPH

KITTI：分難度等級的 3D AP

在 KITTI 的三維基準上，一個偵測若與某個真值框的三維 IoU 至少為 0.7（車）或 0.5（行人與單車騎士），就是正確的 [48]。物體依其影像外觀被分成三個累積的難度等級 [48]：

- **簡單：** 2D 框高度至少 40 px、完全可見、截斷最多 15%。
- **中等：**高度至少 25 px、最多部分被遮擋、截斷最多 30%。
- **困難：**高度至少 25 px、最多「難以看見」、截斷最多 50%。

方法依中等難度的 AP 排名。自 2019 年 10 月 8 日起，基準依 Simonelli 等人的分析 [48], [52]，改用 40 個召回點而不是原本的 11 個來計算 AP；論文把它寫成 $\mathrm{AP}|_{R_{40}}$，兩種規則算出的數字不能拿來比較。標註只涵蓋前方相機可見的物體，所以 KITTI 的光達偵測器通常只保留投影到影像內的點 [10]。

nuScenes：以中心距離計算的 mAP 與 NDS

用距離配對。 nuScenes 不用 IoU 判斷偵測是否正確。一個預測會與地面平面上二維中心距離最小的真值物體配對，只要該距離低於門檻 [50]。AP 針對每個類別，在門檻 $\{0.5, 1, 2, 4\}$ m 下計算，並只在召回率與精確率都超過 0.1 的範圍內對精確率–召回率曲線積分，而

$$\mathrm{mAP} = \frac{1}{|\mathbb{C}|\,|\mathbb{D}|} \sum_{c \in \mathbb{C}} \sum_{d \in \mathbb{D}} \mathrm{AP}_{c,d},$$

其中 $\mathbb{C}$ 是 10 個偵測類別的集合，$\mathbb{D} = \{0.5, 1, 2, 4\}$ 是以公尺為單位的距離門檻集合 [50]。每個類別只在離自車的自己那一段範圍內評估（障礙物與交通錐為 30 m，自行車、機車與行人為 40 m，各種車輛類別為 50 m），而且每個樣本最多可以提交 500 個框 [50]。

為什麼用距離？同樣的中心誤差，IoU 對小物體的懲罰遠比對大物體重（圖 6）。例如，0.25 m 的誤差會讓 0.7 m 的行人框掉到 IoU 約 0.47，低於 0.5，而一輛車沿著車長方向平移同樣的量，IoU 仍接近 0.9。

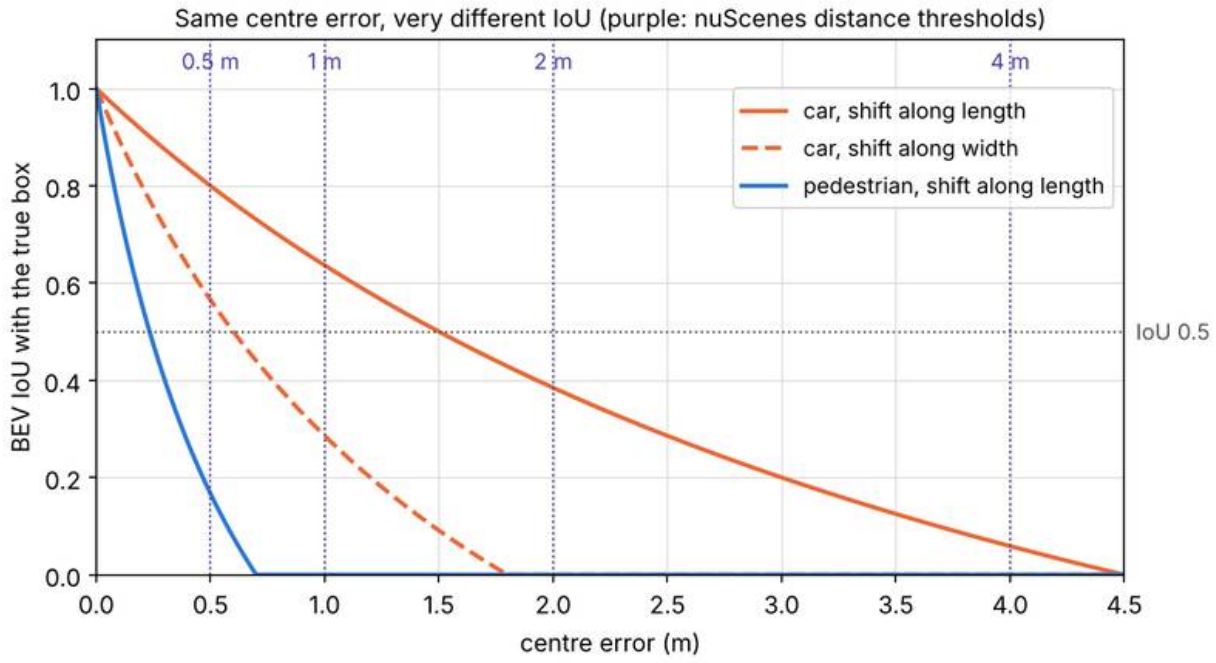


圖 *6* — 車（*4.5 × 1.8 m*）與行人（*0.7 × 0.7 m*）的 *BEV IoU* 對中心誤差，尺寸與偏航角皆精確。紫線是 *nuScenes* 的配對門檻。

下面的程式碼用 AP 說明同一件事。兩個類別得到相同的中心誤差（高斯分布，每個軸的標準差 0.25 m）和相同數量的偽陽性；只有尺寸不同。（我們計算的是一般的 101 點內插 AP，並略過 nuScenes 的 0.1 截斷，這不影響比較。）

```
import numpy as np

rng = np.random.default_rng(0)
size = {"car": (4.5, 1.8), "pedestrian": (0.7, 0.7)}     # BEV length x width in metres

def bev_iou(a, b):
    """Axis-aligned BEV IoU of boxes (x, y, l, w); yaw is 0 in this toy."""
    ox = max(0, min(a[0] + a[2] / 2, b[0] + b[2] / 2) - max(a[0] - a[2] / 2, b[0] - b[2] /
     ↪  2))
    oy = max(0, min(a[1] + a[3] / 2, b[1] + b[3] / 2) - max(a[1] - a[3] / 2, b[1] - b[3] /
     ↪  2))
    return ox * oy / (a[2] * a[3] + b[2] * b[3] - ox * oy)

def average_precision(gts, preds, is_match):
    """Greedy matching in score order; AP = mean interpolated precision at 101 recall
     ↪  points."""
    used, tp = np.zeros(len(gts), bool), []
    for p in sorted(preds, key=lambda p: -p[-1]):
        ok = [j for j in range(len(gts)) if not used[j] and is_match(p, gts[j])]
        if ok:
            j = min(ok, key=lambda j: np.hypot(p[0] - gts[j][0], p[1] - gts[j][1]))
            used[j] = True
```

```
            tp.append(bool(ok))
        tp = np.array(tp)
        recall = np.cumsum(tp) / len(gts)
        precision = np.cumsum(tp) / np.arange(1, len(tp) + 1)
        envelope = np.maximum.accumulate(precision[::-1])[::-1]
        return np.mean([envelope[recall >= r].max() if np.any(recall >= r) else 0
                        for r in np.linspace(0, 1, 101)])

    for cls, (l, w) in size.items():
        gts = [(x, y, l, w) for x, y in rng.uniform(-50, 50, (100, 2))]
        preds = [(g[0] + dx, g[1] + dy, l, w, s) for g, (dx, dy), s in
                 zip(gts, rng.normal(0, 0.25, (100, 2)), rng.uniform(0.3, 1, 100))]
        preds += [(x, y, l, w, s) for (x, y), s in zip(rng.uniform(-50, 50, (20, 2)),
         ↪  rng.uniform(0, 0.6, 20))]
        ap_iou = average_precision(gts, preds, lambda p, g: bev_iou(p, g) >= 0.5)
        ap_d = [average_precision(gts, preds, lambda p, g, t=t: np.hypot(p[0] - g[0], p[1] -
         ↪  g[1]) <= t)
                for t in (0.5, 1, 2, 4)]
        print(f"{cls:10s} AP(BEV IoU>=0.5) = {ap_iou:.3f} | AP(centre dist <= 0.5/1/2/4 m) = "
              f"{' / '.join(f'{a:.3f}' for a in ap_d)} | mean = {np.mean(ap_d):.3f}")
```

```
car        AP(BEV IoU>=0.5) = 0.951 | AP(centre dist <= 0.5/1/2/4 m) = 0.789 / 0.988 / 0.988
 ↪  / 0.990 | mean = 0.939
pedestrian AP(BEV IoU>=0.5) = 0.126 | AP(centre dist <= 0.5/1/2/4 m) = 0.791 / 0.979 / 0.979
 ↪  / 0.979 | mean = 0.932
```

在這次玩具實驗中，IoU 配對把行人偵測器評為幾乎沒用（AP 0.126），把車輛偵測器評為極好（0.951），雖然兩者的中心誤差完全相同。中心距離配對則給兩者大致相同的分數。距離配對並沒有忽略框的品質；它把品質移到另外的誤差項中。

真陽性誤差與 NDS。對於在 2 m 門檻下配對成功的偵測，nuScenes 量測五種**真陽性誤差**（TP errors）：平移誤差 ATE（中心距離，單位公尺）、尺度誤差 ASE（對齊中心與朝向之後的 $1-\text{IoU}$）、朝向誤差 AOE（最小偏航角差，單位弧度）、速度誤差 AVE（m/s）與屬性誤差 AAE（$1-$ 屬性準確率）[50]。它們的類別平均與 mAP 結合成 **nuScenes 偵測分數**（nuScenes detection score，NDS）：

$$\text{NDS} = \frac{1}{10}\Big[\,5\,\text{mAP} + \sum_{e \in \mathbb{E}} \big(1 - \min(1, e)\big)\Big],$$

其中 $\mathbb{E} = \{\text{mATE}, \text{mASE}, \text{mAOE}, \text{mAVE}, \text{mAAE}\}$ 是平均 TP 誤差的集合 [50]。NDS 有一半來自 mAP，另一半來自框的品質、朝向、速度與屬性，所以能把速度估得好的方法（靠多次掃描或多幀）能在 NDS 上獲得很大的增益。

僅線上版：一步步計算 *NDS* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/notes/一步步計算-nds/*

Waymo：以朝向加權的 AP

Waymo Open Dataset 使用三維 IoU 配對，車輛的門檻為 0.7，行人與單車騎士為 0.5，它的主要指標是 **APH**，即依朝向（heading）加權的平均精確率 [51]。每個真陽性不是算 1，而是算成一個朝向準確度

$$a_h = 1 - \frac{\min\left(|\tilde{\theta} - \theta|,\ 2\pi - |\tilde{\theta} - \theta|\right)}{\pi} \in [0, 1],$$

其中 $\tilde{\theta}$ 與 θ 分別是預測與真值的朝向，範圍在 $[-\pi, \pi]$ [3], [53]。因此一個正確但正好朝著反方向的框（$|\tilde{\theta} - \theta| = \pi$）算 0，這堵住了 IoU 留下的漏洞。真值框被分成兩個難度等級：**LEVEL_2** 是有 1 到 5 個光達點的框（或在釋出標註中被標為 LEVEL_2 的框），**LEVEL_1** 是其餘的框（沒有光達點的框不評估），而 LEVEL_2 的指標兩者都包含 [51]。它的 2020 年三維偵測挑戰賽以 **LEVEL_2 的 mAPH**（車輛、行人與單車騎士的平均）排名，並依距離細分結果：$[0, 35)$、$[35, 50)$ 與超過 50 m [51]。

效率與部署

白話版。車上的電腦必須在下一次掃描到來之前，早早完成感知，在嵌入式硬體上，每一次都是如此。表示方式決定了大部分的成本：有多少格子、每個格子多少個點，以及運算子能否在目標晶片上跑得快。

- **網格大小與範圍決定預算**。格子尺寸減半，BEV 格子數就變成四倍。PointPillars 正是這樣拿準確度換速度，從預設網格的 62 Hz 到較粗柱的 105 Hz [10]。
- **稀疏運算子需要支援**。稀疏卷積與點取樣原則上很有效率，但它們依賴自訂核心（kernel）；柱網路則跑在每個推論引擎都支援的稠密二維卷積上。DSVT 把部署明確當作設計目標，其池化不需要自訂的 CUDA 運算子，並報告部署後可即時推論 [26]。
- **相機的視角轉換很昂貴**。把每個像素提升到許多深度，會產生巨大的中間點集；BEVFusion 優化後的 BEV 池化，把這一步的延遲降低了 40 倍以上 [43]。
- **基準上的速度不是車上的速度**。 PointPillars 的論文指出，KITTI 只使用前方相機視野內的點，大約是一次掃描的 10%，而且桌上型 GPU 的計時不能直接換算到嵌入式電腦 [10]。要在目標硬體上、用完整 360 度的輸入、從頭到尾地量測。
- **後處理與時間輸入也要算進來**。旋轉 NMS 要處理很多框；多次掃描的輸入會讓點數成倍增加。nuScenes 把偵測方法可使用的過去感測器資料，限制在約 0.5 秒 [50]。

失敗模式

白話版。三維偵測器會在證據薄弱處（遠的、小的或被遮住的物體）、感測器受到干擾處（天氣、髒汙、校正不準），以及測試資料與訓練資料不同處（另一座城市、另一種感測器）失敗。

- **遠處與稀疏的物體**。點數大致隨 $1/r^2$ 下降（圖 2），所以遠處的物體只有幾個點，甚至一個也沒有。基準也承認這一點：Waymo 把含 1–5 個點或有明確難度標記的框歸為 LEVEL_2，而 LEVEL_2 評估同時包含兩個難度等級 [51]，而 nuScenes 只在 30–50 m 內評估每個類別 [50]。相機的深度誤差隨 Z^2 成長，對同樣這些物體傷害更大。

- **遮擋與截斷**。感測器只看到面向它的表面，而一輛停著的車會擋住它後面的行人。KITTI 的難度等級就是依遮擋與截斷定義的 [48]；時間模型能靠記住暫時被擋住的物體來幫忙 [35]。
- **容易混淆的類別**。 PointPillars 報告，行人與單車騎士常被彼此混淆，行人也會與電線桿和樹幹混淆 [10]。
- **天氣**。霧、雨和雪會散射雷射脈衝，產生虛假的近距離回波並丟掉遠處的回波。Hahner 等人在晴天的光達掃描上模擬基於物理的霧，並證明用它訓練能改善真實霧中的偵測 [54]。
- **校正與同步誤差**。把點硬投影到像素的融合，在外參漂移時會壞掉；TransFusion 的軟關聯正是為此設計的 [45]，而 LSS 則顯示，用帶雜訊的外參或丟棄相機來訓練，能讓相機 BEV 模型對它們更穩健 [32]。
- **常見的損壞**。一個針對 KITTI、nuScenes 與 Waymo 套用 27 種光達與相機損壞、測試 24 個偵測器的基準發現：與運動相關的損壞對所有模型傷害最大，光達–相機融合模型比較穩健，而純相機模型對影像損壞極其脆弱 [55]。
- **資料集之間的領域偏移**。在一個資料集上訓練的偵測器，到了另一個資料集可能大幅下滑。Wang 等人發現，德國與美國駕駛資料集之間的主要障礙，單純就是平均車輛尺寸不同，修正它就能彌補大部分差距 [56]。Waymo 論文本身也報告不同城市的資料之間有明顯的落差 [3]。不同的光達型號（光束數、安裝高度）也會讓點的分布偏移。

現代觀點

圖 7 依各篇引用的論文發表年份，把本篇的方法放在時間軸上。

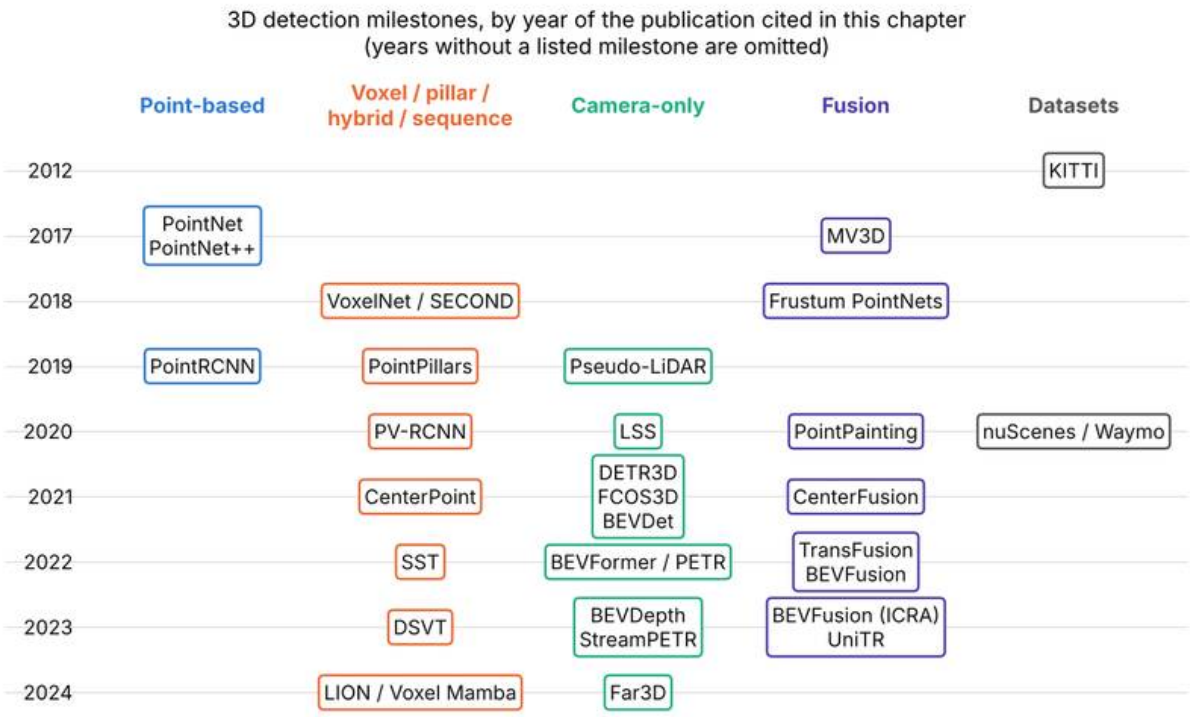


圖 7 — 依本篇所引論文的發表年份整理的里程碑，並按家族分組。沒有列出里程碑的年份省略。

綜述怎麼說

六篇綜述為這個領域搭起框架，各有自己的分類法。Mao 等人 [1] 給出自駕三維偵測最完整的地圖：光達方法依資料表示分類（點、網格、點–體素、距離影像），相機方法（單眼、立體、多視角）、融合、Transformer、時間與標註高效的偵測，並分析準確度與速度的趨勢。Qian 等人 [2] 涵蓋感測器、資料集與指標，並附有一個含執行時間、誤差與穩健性分析的案例研究。Guo 等人 [57] 回顧整個點雲深度學習，偵測只是其中一項任務。在相機方面，Ma 等人 [58] 與 Li 等人 [59] 綜述鳥瞰視角感知；後者把核心問題歸納為視角轉換、BEV 真值、多感測器流程，以及跨感測器配置的泛化，並補上一份實用的技巧「配方」。Wang 等人 [39] 則依融合階段、輸入與粒度來整理融合方法。兩個共通的結論：BEV 已成為各種感測器與任務共同的交會點，而標準基準上準確度的進步，已經超過穩健性與可部署性的進步。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-object-detection/papers/現代觀點綜述導讀/*

2023–2026 年的最先進技術

- **光達骨幹**已從稀疏卷積走向稀疏 Transformer，以及對序列化體素的線性時間序列模型（DSVT [26]、LION [27]、Voxel Mamba [28]）。Point Transformer V3 [60] 主張規模比繁複的設計更重要，並在多個資料集聯合訓練下進一步提升。
- **純相機偵測**已經縮小了與光達的差距，主要靠時間建模與稀疏查詢（StreamPETR [37]）以及更好的深度監督（BEVDepth [33]）；長距離相機偵測是活躍的方向（Far3D [38]）。
- 在共用的 BEV 或 token 空間中做**融合**（BEVFusion [43], [44]、UniTR [46]），是最強系統的標準配方，而對感測器缺失或劣化的穩健性，現在已是設計目標，而不是事後補救。
- **端到端駕駛**。偵測越來越常是為規劃而訓練的單一網路中的一個階段，例如 UniAD [61]，其中感知、預測與規劃任務透過統一的查詢介面溝通。

尚待解決的問題

1. **遠處、小型與罕見的物體**。遠距離的稀疏證據，以及訓練資料中罕見的類別（動物、碎片、特殊車輛），仍是弱點；封閉集合的資料集並未涵蓋長尾 [1]。
2. **穩健性**。天氣、感測器故障、校正漂移與運動損壞仍會造成大幅下降 [55], [54]。
3. **跨感測器與跨地點的泛化**。在一種光達型號、相機配置或城市上訓練的模型，常常在另一種上失敗 [56], [3]。
4. **符合駕駛需求的指標**。 AP 對 10 m 處漏掉的行人與 45 m 處漏掉的行人一視同仁；nuScenes 與 Waymo 加入了朝向、速度與距離細分，但正如 Mao 等人所指出，只最大化 AP 對整個駕駛系統來說未必是最佳的 [1]。
5. **標註**。三維框的標註很昂貴，這催生了半監督、自監督與自動標註的方法 [1]。

重點整理

- 解讀偵測前，先註明方框規約、單位與座標變換。
- 點雲稀疏性與隨距離變化的取樣，決定哪些表示有用。

- 鳥瞰視角簡化公尺空間推理，垂直結構與校準仍然重要。
- 僅相機偵測須推論深度；時間與額外感測器證據可協助，但也引入對齊要求。
- 重疊指標可能掩蓋方向錯誤，須檢查偏航規約與方向敏感指標。
- KITTI、nuScenes 與 Waymo 使用不同匹配及評分規則；應報告協定並量測部署流程。

練習

1. **距離上的點數**。某個光達的水平間隔為 0.1°、垂直間隔為 0.4°。用 $n(r) \approx A/(r^2 \Delta\phi \Delta\epsilon)$，估計一輛車 1.8 m × 1.5 m 的車尾，在 20 m 與 60 m 處各有多少個點。在 60 m 處，有幾條光束垂直穿過這輛車？

   提示

   換算成弧度：$\Delta\phi = 1.75 \times 10^{-3}$，$\Delta\epsilon = 6.98 \times 10^{-3}$。取 $A = 2.7\ \text{m}^2$，則 $n(20) \approx 2.7/(400 \cdot 1.22 \times 10^{-5}) \approx 550$，$n(60) \approx 62$。垂直方向上，射線在 60 m 處相隔 $r\,\Delta\epsilon \approx 0.42$ m，所以只有約 3–4 條光束穿過 1.5 m 高的物體；其餘的點來自細小的水平間隔。

2. **手算 IoU**。兩個 4 m × 2 m × 1.5 m 的框共用同一個中心；其中一個繞垂直軸旋轉 90°。計算 BEV IoU 與三維 IoU。接著把第二個框抬高 0.75 m，重新計算三維 IoU。用 IoU 的程式碼檢查你的答案。

   提示

   BEV 交集是 2 m × 2 m 的正方形：$\text{IoU}_{\text{bev}} = 4/(8 + 8 - 4) = 1/3$。高度相等時，三維 IoU 也是 $6/(12+12-6) = 1/3$。抬高 0.75 m 之後，高度重疊是 0.75 m，所以交集是 $3\ \text{m}^3$，$\text{IoU}_{\text{3D}} = 3/(24-3) \approx 0.143$。

3. **弄壞傳統流程**。在傳統基準的程式碼中，把行人移到 $(13.5, -2.2)$，貼著第一輛車（在我們的執行中，它的點與車的點相距不到約 0.15 m），再執行一次。DBSCAN 會怎麼做？改變 `eps` 來修正它，並說明太小的 `eps` 會對遠處的物體造成什麼新的失敗。

   提示

   在我們的執行中，`eps` 從 0.7 一路降到 0.15，都會把行人與車子合併成同一個群，得到錯誤的框（5.1 × 1.8 m、偏航角 28°，而不是 4.5 × 1.8 m、17°）。在 `eps=0.12` 時行人分開了，但 25 m 處的車已經碎成好幾塊；在 `eps=0.1` 時，連近處的車也裂成兩個可見的側面。遠處物體的點較少，所以點與點之間的間隙較大（在真實掃描器上，間隙以 $r\,\Delta\phi$ 的方式成長）。固定的 `eps` 無法適用所有距離，這是 $1/r^2$ 密度的直接後果。

4. **配對協定**。使用配對的程式碼，找出中心誤差的標準差為多少時，車的 AP（IoU ≥ 0.5）會等於行人在標準差 0.25 m 時的 AP。這對於比較用 IoU 配對算出的各類別 AP，說明了什麼？

   提示

   在我們的執行中，車的 IoU 型 AP 在標準差 0.75 m 時降到 0.117，約是行人的 0.25 m 的三倍（大致是車寬與行人寬的比值），與圖 6 一致。因此各類別的 IoU 型 AP 把偵測品質與物體大小混在一起，所以行人 AP 低，並不表示行人偵測器以公尺計的定位比較差。

5. **依朝向加權的 AP**。某個偵測器找到場景中全部 10 輛車，沒有偽陽性，IoU 也完美，但其中 3 輛朝向相反，2 輛偏差 30°。它的 AP 與 APH 各是多少（假設所有偵測的分數相同）？哪一個 nuScenes TP 誤差會揭露同樣的問題？

   提示

   AP 是 1，因為每個框都配對成功。朝向準確度：$5 \times 1 + 3 \times 0 + 2 \times (1 - 30/180) = 6.67$，所以 APH ≈ 0.667。在 nuScenes 中，朝向誤差 AOE（弧度）會上升，透過 $1 - \min(1, \text{mAOE})$ 這一項拉低 NDS。

6. **柱的預算**。在柱化的程式碼中，把網格改成 0.32 m 與 0.08 m（調整 `W` 與 `H`，使網格仍涵蓋 70.4 m × 80 m）。回報非空柱的數量、空格子的比例，以及被上限 N 丟掉的點數。把這個趨勢與 PointPillars 所報告的速度–準確度取捨連起來。

   提示

   在我們的執行中，0.32 m 得到 7,567 根非空柱（86.2% 為空），在 $N = 16$ 時丟掉 1,734 個點；0.08 m 得到 10,908 根柱（98.8% 為空），沒有丟掉任何點。較粗的柱表示柱比較少、比較滿：偽影像較小，被上限丟掉的點較多，所以較快但較不精確。較細的柱表示柱比較多、每根只有一兩個點，偽影像較大，空格子也較多。PointPillars 報告較粗的網格速度較高，損失主要出現在行人與單車騎士。

偵測結果是追蹤的輸入，追蹤把它們跨時間連起來；見 DLCV 07・物件追蹤：從單一物件到多物件。

參考文獻

1. J. Mao, S. Shi, X. Wang and H. Li, “3D object detection for autonomous driving: A comprehensive survey,” *International Journal of Computer Vision*, vol. 131, no. 8, 2023. doi
2. R. Qian, X. Lai and X. Li, “3D object detection for autonomous driving: A survey,” *Pattern Recognition*, vol. 130, 2022. doi
3. P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik et al., “Scalability in perception for autonomous driving: Waymo Open Dataset,” in *Proc. IEEE/CVF CVPR*, 2020. CVF
4. H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong et al., “nuScenes: A multimodal dataset for autonomous driving,” in *Proc. IEEE/CVF CVPR*, 2020. CVF
5. M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, 1981. doi
6. M. Ester, H.-P. Kriegel, J. Sander and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proc. KDD*, 1996. AAAI
7. X. Zhang, W. Xu, C. Dong and J. M. Dolan, “Efficient L-shape fitting for vehicle detection using laser scanners,” in *Proc. IEEE Intelligent Vehicles Symposium (IV)*, 2017. doi
8. Y. Zhou and O. Tuzel, “VoxelNet: End-to-end learning for point cloud based 3D object detection,” in *Proc. IEEE CVPR*, 2018. CVF
9. Y. Yan, Y. Mao and B. Li, “SECOND: Sparsely embedded convolutional detection,” *Sensors*, vol. 18, no. 10, 3337, 2018. doi

10. A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. IEEE/CVF CVPR*, 2019. CVF
11. T. Yin, X. Zhou and P. Krähenbühl, "Center-based 3D object detection and tracking," in *Proc. IEEE/CVF CVPR*, 2021. CVF
12. I. E. Sutherland and G. W. Hodgman, "Reentrant polygon clipping," *Communications of the ACM*, vol. 17, no. 1, 1974. doi
13. X. Zhou, D. Wang and P. Krähenbühl, "Objects as points," arXiv:1904.07850, 2019. arXiv
14. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. ECCV*, 2020. doi
15. Y. Wang, V. Guizilini, T. Zhang, Y. Wang, H. Zhao and J. Solomon, "DETR3D: 3D object detection from multi-view images via 3D-to-2D queries," in *Proc. Conference on Robot Learning (CoRL)*, 2021. arXiv
16. C. R. Qi, H. Su, K. Mo and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE CVPR*, 2017. arXiv
17. C. R. Qi, L. Yi, H. Su and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. NeurIPS
18. S. Shi, X. Wang and H. Li, "PointRCNN: 3D object proposal generation and detection from point cloud," in *Proc. IEEE/CVF CVPR*, 2019. arXiv
19. S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE TPAMI*, vol. 39, no. 6, 2017. doi
20. B. Graham, M. Engelcke and L. van der Maaten, "3D semantic segmentation with submanifold sparse convolutional networks," in *Proc. IEEE CVPR*, 2018. CVF
21. S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang and H. Li, "PV-RCNN: Point-voxel feature set abstraction for 3D object detection," in *Proc. IEEE/CVF CVPR*, 2020. arXiv
22. W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. ECCV*, 2016. arXiv
23. T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE ICCV*, 2017. doi
24. J. Huang, G. Huang, Z. Zhu, Y. Ye and D. Du, "BEVDet: High-performance multi-camera 3D object detection in bird-eye-view," arXiv:2112.11790, 2021. arXiv
25. L. Fan, Z. Pang, T. Zhang, Y.-X. Wang, H. Zhao, F. Wang, N. Wang and Z. Zhang, "Embracing single stride 3D object detector with sparse transformer," in *Proc. IEEE/CVF CVPR*, 2022. CVF
26. H. Wang, C. Shi, S. Shi, M. Lei, S. Wang, D. He, B. Schiele and L. Wang, "DSVT: Dynamic sparse voxel transformer with rotated sets," in *Proc. IEEE/CVF CVPR*, 2023. CVF
27. Z. Liu, J. Hou, X. Wang, X. Ye, J. Wang, H. Zhao and X. Bai, "LION: Linear group RNN for 3D object detection in point clouds," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. NeurIPS
28. G. Zhang, L. Fan, C. He, Z. Lei, Z. Zhang and L. Zhang, "Voxel Mamba: Group-free state space models for point cloud based 3D object detection," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. NeurIPS
29. T. Wang, X. Zhu, J. Pang and D. Lin, "FCOS3D: Fully convolutional one-stage monocular 3D object detection," in *Proc. IEEE/CVF ICCV Workshops*, 2021. arXiv
30. Z. Tian, C. Shen, H. Chen and T. He, "FCOS: Fully convolutional one-stage object detection,"

in *Proc. IEEE/CVF ICCV*, 2019. arXiv
31. Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell and K. Q. Weinberger, "Pseudo-LiDAR from visual depth estimation: Bridging the gap in 3D object detection for autonomous driving," in *Proc. IEEE/CVF CVPR*, 2019. arXiv
32. J. Philion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3D," in *Proc. ECCV*, 2020. arXiv
33. Y. Li, Z. Ge, G. Yu, J. Yang, Z. Wang, Y. Shi, J. Sun and Z. Li, "BEVDepth: Acquisition of reliable depth for multi-view 3D object detection," in *Proc. AAAI*, 2023. arXiv
34. Y. Liu, T. Wang, X. Zhang and J. Sun, "PETR: Position embedding transformation for multi-view 3D object detection," in *Proc. ECCV*, 2022. arXiv
35. Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Y. Qiao and J. Dai, "BEVFormer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers," in *Proc. ECCV*, 2022. arXiv
36. X. Zhu, W. Su, L. Lu, B. Li, X. Wang and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *Proc. ICLR*, 2021. arXiv
37. S. Wang, Y. Liu, T. Wang, Y. Li and X. Zhang, "Exploring object-centric temporal modeling for efficient multi-view 3D object detection," in *Proc. IEEE/CVF ICCV*, 2023. CVF
38. X. Jiang, S. Li, Y. Liu, S. Wang, F. Jia, T. Wang, L. Han and X. Zhang, "Far3D: Expanding the horizon for surround-view 3D object detection," in *Proc. AAAI*, 2024. arXiv
39. Y. Wang, Q. Mao, H. Zhu, J. Deng, Y. Zhang, J. Ji, H. Li and Y. Zhang, "Multi-modal 3D object detection in autonomous driving: A survey," *International Journal of Computer Vision*, vol. 131, no. 8, 2023. doi
40. S. Vora, A. H. Lang, B. Helou and O. Beijbom, "PointPainting: Sequential fusion for 3D object detection," in *Proc. IEEE/CVF CVPR*, 2020. arXiv
41. X. Chen, H. Ma, J. Wan, B. Li and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE CVPR*, 2017. arXiv
42. C. R. Qi, W. Liu, C. Wu, H. Su and L. J. Guibas, "Frustum PointNets for 3D object detection from RGB-D data," in *Proc. IEEE CVPR*, 2018. CVF
43. Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. Rus and S. Han, "BEVFusion: Multi-task multi-sensor fusion with unified bird's-eye view representation," in *Proc. IEEE ICRA*, 2023. arXiv
44. T. Liang, H. Xie, K. Yu, Z. Xia, Z. Lin, Y. Wang, T. Tang, B. Wang et al., "BEVFusion: A simple and robust LiDAR-camera fusion framework," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022. arXiv
45. X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu and C.-L. Tai, "TransFusion: Robust LiDAR-camera fusion for 3D object detection with transformers," in *Proc. IEEE/CVF CVPR*, 2022. arXiv
46. H. Wang, H. Tang, S. Shi, A. Li, Z. Li, B. Schiele and L. Wang, "UniTR: A unified and efficient multi-modal transformer for bird's-eye-view representation," in *Proc. IEEE/CVF ICCV*, 2023. arXiv
47. R. Nabati and H. Qi, "CenterFusion: Center-based radar and camera fusion for 3D object detection," in *Proc. IEEE/CVF WACV*, 2021. doi
48. A. Geiger, P. Lenz, C. Stiller and R. Urtasun, "3D object detection evaluation 2017," The KITTI Vision Benchmark Suite, official benchmark page (data sizes, IoU thresholds, difficulty levels, 40 recall positions). link

49. A. Geiger, P. Lenz and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *Proc. IEEE CVPR*, 2012. doi
50. nuScenes devkit, “nuScenes detection task,” official evaluation documentation (classes, detection ranges, mAP, TP metrics, NDS, submission rules). link
51. Waymo Open Dataset, “3D detection challenge (2020),” official challenge page (APH, IoU thresholds, LEVEL_1/LEVEL_2, range breakdown). link
52. A. Simonelli, S. Rota Bulò, L. Porzi, M. López-Antequera and P. Kontschieder, “Disentangling monocular 3D object detection,” in *Proc. IEEE/CVF ICCV*, 2019. doi
53. Waymo Open Dataset, `metrics_utils.cc`, official metric implementation (function `ComputeHeadingAccuracy`). code
54. M. Hahner, C. Sakaridis, D. Dai and L. Van Gool, “Fog simulation on real LiDAR point clouds for 3D object detection in adverse weather,” in *Proc. IEEE/CVF ICCV*, 2021. arXiv
55. Y. Dong, C. Kang, J. Zhang, Z. Zhu, Y. Wang, X. Yang, H. Su, X. Wei et al., “Benchmarking robustness of 3D object detection to common corruptions in autonomous driving,” in *Proc. IEEE/CVF CVPR*, 2023. arXiv
56. Y. Wang, X. Chen, Y. You, L. E. Li, B. Hariharan, M. Campbell, K. Q. Weinberger and W.-L. Chao, “Train in Germany, test in the USA: Making 3D object detectors generalize,” in *Proc. IEEE/CVF CVPR*, 2020. arXiv
57. Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu and M. Bennamoun, “Deep learning for 3D point clouds: A survey,” *IEEE TPAMI*, vol. 43, no. 12, 2021. doi
58. Y. Ma, T. Wang, X. Bai, H. Yang, Y. Hou, Y. Wang, Y. Qiao, R. Yang and X. Zhu, “Vision-centric BEV perception: A survey,” *IEEE TPAMI*, vol. 46, no. 12, 2024. doi
59. H. Li, C. Sima, J. Dai, W. Wang, L. Lu, H. Wang, J. Zeng, Z. Li et al., “Delving into the devils of bird’s-eye-view perception: A review, evaluation and recipe,” *IEEE TPAMI*, vol. 46, no. 4, 2024. doi
60. X. Wu, L. Jiang, P.-S. Wang, Z. Liu, X. Liu, Y. Qiao, W. Ouyang, T. He and H. Zhao, “Point Transformer V3: Simpler, faster, stronger,” in *Proc. IEEE/CVF CVPR*, 2024. arXiv
61. Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du et al., “Planning-oriented autonomous driving,” in *Proc. IEEE/CVF CVPR*, 2023. arXiv

DLCV 04・影像分割：語意、實例與全景分割

深度網路如何替每個像素貼標籤：三種分割任務、全卷積與編碼器–解碼器網路、空洞卷積與多尺度脈絡、損失函數與評估指標（mIoU、mask AP、全景品質），以及 MaskFormer 與 Mask2Former 的遮罩分類，還有可提示與開放詞彙分割。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/

你將學到

- 區分語意、實例與全景分割的輸出要求。
- 追蹤全卷積網路、解碼器與跳接中的解析度。
- 為類別不平衡與邊界敏感任務選擇損失。
- 計算並解讀 mIoU、mask AP 與全景品質。
- 比較遮罩分類、可提示分割與開放詞彙介面。

先看全貌

偵測器為每個物體畫一個方框（DLCV 03）。方框只是粗略的答案：它包含背景像素，也沒有說出物體的精確輪廓。**分割**（segmentation）給出精確的答案：它替每個像素決定「這個像素屬於什麼」。

> **用最簡單的話說**
>
> 給小朋友一頁街景的著色本和一盒蠟筆。**語意分割**是依「種類」上色：每輛車都塗藍色，每個人都塗橘色，馬路塗灰色，天空塗淺藍色。兩輛貼在一起的車會變成一整塊藍色。**實例分割**是在每輛車、每個人身上貼一張編號貼紙，並且只在該物體的輪廓裡面上色；馬路和天空留白。**全景分割**兩件事都做：頁面上每一小塊都有顏色，而且每輛車、每個人還各有自己的編號。

為什麼重要：自駕車需要知道哪些像素是可行駛的路面，而不只是「這個框裡某處有一條路」。放射科醫師量測腫瘤，需要它的精確面積。要把一個人從照片中移除的修圖軟體，需要這個人的輪廓。抓取物體的機器人、拍到農田與森林的衛星影像、醫學掃描，都需要逐像素的答案。

本篇是兩篇經典章節的深度學習續篇。IP 10 與 IP 11 用門檻、區域生長、圖切割（graph cut）、分水嶺與主動輪廓來分割影像，完全不需要訓練資料；本篇則是從有標註的範例中學出分割。DLCV 03 介紹過方框、IoU、錨框、特徵金字塔、focal loss 與 Mask R-CNN；這些想法我們直接沿用並連結回去，不重複說明。可提示的 Segment Anything 模型在 DLCV 01 與 ML 10 介紹過，這裡只說明它們在各種任務中的位置。三篇綜述勾勒出這個領域：Minaee 等人談 CNN 時代 [1]，Li 等人談以 Transformer 為基礎的分割 [2]，Zhou 等人談結合基礎模型的分割 [3]。我們會在「現代觀點」回頭討論。

三個任務，一張標籤圖

白話版。語意分割為像素分類，實例分割區分物件，全景分割則結合類別與適用的物件身分。

精確版。設 $\Omega = \{1, \dots, H\} \times \{1, \dots, W\}$ 為 $H \times W$ 影像的像素位置集合，$\{1, \dots, K\}$ 為類別標籤。

- **語意分割**預測一張標籤圖 $y : \Omega \to \{1, \dots, K\}$，每個像素 $\mathbf{x}$ 有一個類別 $y(\mathbf{x})$。它不區分同一類別的兩個物體。
- **實例分割**預測一個集合 $\{(m_j, c_j, s_j)\}_{j=1}^{N}$，其中 $m_j \in \{0,1\}^{H \times W}$ 是二值遮罩，c_j 是它的類別，$s_j \in [0,1]$ 是信心分數，N 隨影像而變。遮罩之間可以重疊，而且只會分割可數的物體類別。它就是「輸出遮罩而不是方框」的偵測。
- **全景分割** [4] 把每個像素 i 對應到一個數對 (l_i, z_i)，其中 l_i 是類別，z_i 是實例編號（instance id）。具有相同數對的像素組成一個**區段**（segment）。由於每個像素只帶一個數對，區段不能重疊，也不需要信心分數。

全景分割的論文沿用視覺領域的一個舊區分，替兩種類別命名。**Things** 是形狀明確、可以數的物體，例如人、車與動物。**Stuff** 是質地或材質相近的無定形區域，例如天空、馬路與草地 [4]。對 stuff 類別而言，實例編號會被忽略：一張影像裡所有的天空是同一個區段。哪些類別算 stuff、哪些算 things，由各資料集自行設計。含糊不清或不屬於任何類別的像素會得到特殊的 **void** 標籤，並且不參與評估。

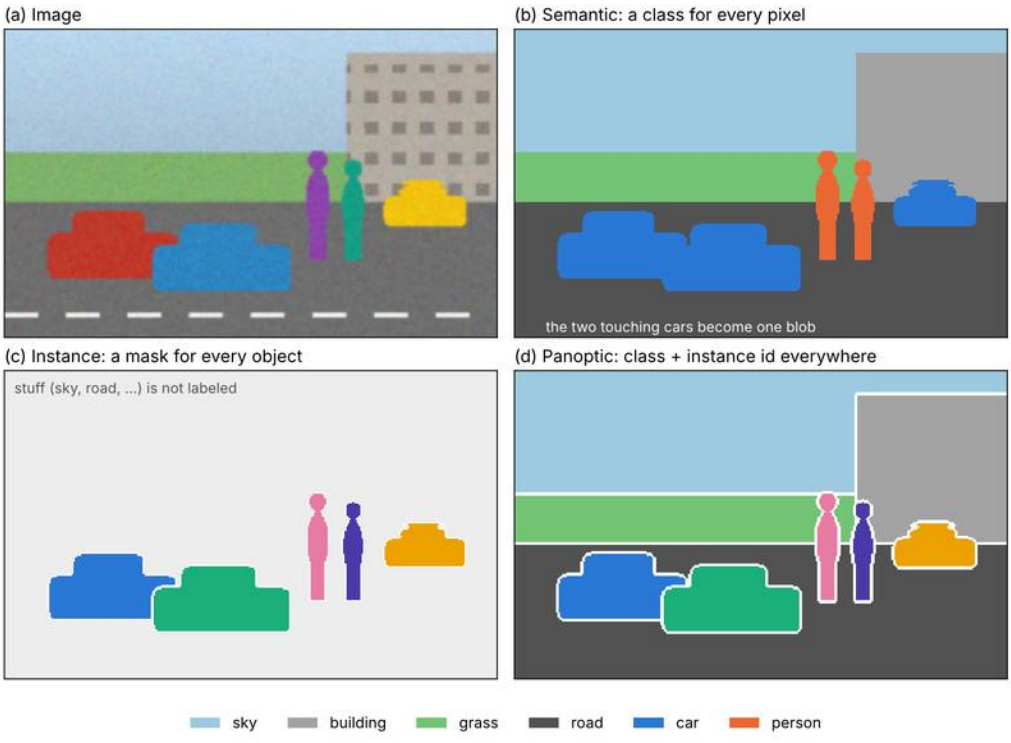


圖 *1* — 同一個合成場景上的三種任務。(*b*) 語意：兩輛貼在一起的車變成一個區域。(*c*) 實例：每個物體一個遮罩，*stuff* 不標註。(*d*) 全景：每個像素都有類別，而每個 *thing* 還有自己的身分。

	語意	實例	全景
標籤	每個像素一個類別	每個物體一個遮罩＋類別＋分數	每個像素一個（類別，實例編號）
涵蓋的類別	stuff 與 things	僅 things	stuff 與 things
允許重疊	否	是	否
信心分數	無	有	無
標準指標	mean IoU	mask AP	全景品質（PQ）

這張表說明了為什麼這三種任務長期各有各的方法：它們的輸出形狀不同，每個指標獎勵的東西也不同。本篇大致依這段歷史展開：先是語意分割的逐像素分類器，接著是以偵測器為基礎的實例分割，然後是迫使兩者會合的全景任務，最後是能處理三種任務的單一架構。

從分類器到逐像素分類器

白話版。保留空間特徵，不做全域池化；先預測類別分數格點，再上採樣至影像大小。

全卷積網路

Long、Shelhamer 與 Darrell 的**全卷積網路**（fully convolutional network, FCN）[5] 把分類網路的全連接層換成卷積。卷積可以處理任意大小的輸入，所以網路現在把 $H \times W$ 的影像對應成一張分數圖

$$Z \in \mathbb{R}^{K \times \frac{H}{s} \times \frac{W}{s}},$$

其中 K 是類別數，s 是**輸出步幅**（output stride），也就是輸入解析度與分數圖解析度的比值。分類骨幹網路會反覆地池化並加大步幅；對 FCN 當初採用的那些網路而言 $s = 32$，所以一張 512×512 的影像只會得到 16×16 的類別分數格點。每個分數向量 $Z[:, u, v]$ 概括了影像中很大的一塊視窗，這對辨認「是什麼」很好；但一格就涵蓋 32×32 個像素，對於說出邊界「確切落在哪裡」就很差。

FCN 把分數 Z 上取樣到輸入大小後，用逐像素交叉熵損失端到端地訓練整個網路，起點是為分類預訓練好的權重。為了找回細節，它融合更精細層的預測：步幅 32 的分數先上取樣 2 倍，再加上由步幅 16 那一層算出的分數圖，然後對步幅 8 再做一次。論文把三個版本稱為 FCN-32s、FCN-16s 與 FCN-8s，並顯示每多融合一個更精細的層，輪廓就更細緻 [5]。它的概念可以用一句話概括整個領域：把來自深而粗的層的「是什麼」，與來自淺而細的層的「在哪裡」結合起來。DLCV 01 在編碼器–解碼器那一節也介紹過同一個想法。

上取樣分數，不是標籤

白話版。如果先在低解析度決定類別，再把決定放大，每條邊界都會變成由 $s \times s$ 方塊組成的階梯。如果先把分數平滑地放大，再逐像素決定，邊界就會落在格點之間。

雙線性內插（bilinear interpolation）用最近的四個粗格點算出細位置的分數。對於位在粗座標 $(u + a, v + b)$ 的細像素，其中 u, v 是整數、$a, b \in [0, 1)$，

$$\tilde{Z}_k = (1-a)(1-b)\,Z_k[u,v] + a(1-b)\,Z_k[u+1,v] + (1-a)\,b\,Z_k[u,v+1] + a\,b\,Z_k[u+1,v+1],$$

其中 Z_k 是類別 k 的分數圖，$\tilde{Z}_k$ 是內插後的分數。預測的標籤是 $\hat{y} = \arg\max_k \tilde{Z}_k$。FCN 把上取樣實作成以雙線性內插初始化的步幅轉置卷積，並讓網路自己學習它 [5]；轉置卷積在 ML 03 說明過。後來的多數模型在最後一步直接使用固定的雙線性上取樣。

下面的實驗單獨觀察輸出步幅的影響。它建立一張**理想的**粗分數圖：每個 $s \times s$ 格子裡各類別所占的比例，也就是步幅為 s 的輸出頭（head）的理想輸出；然後比較「放大」與「決定」兩種先後順序。

```
import numpy as np
from scipy import ndimage as ndi

# Synthetic ground truth: 0 background, 1 large disk, 2 thin bar, 3 small square
H = W = 256
yy, xx = np.mgrid[0:H, 0:W]
gt = np.zeros((H, W), int)
gt[(yy - 120) ** 2 + (xx - 90) ** 2 < 60 ** 2] = 1
gt[40:216, 178:184] = 2                         # 6 pixels wide
gt[196:216, 60:80] = 3                          # 20 x 20 pixels
K = 4

def iou_per_class(gt, pred, K):
    return np.array([((gt == k) & (pred == k)).sum() / ((gt == k) | (pred == k)).sum()
                     for k in range(K)])

onehot = np.stack([(gt == k).astype(float) for k in range(K)])
print("stride | nearest labels: mIoU (per class) | bilinear scores: mIoU (per class)")
for s in [4, 8, 16, 32]:
    # ideal stride-s scores: the class fractions inside each s x s cell
    coarse = onehot.reshape(K, H // s, s, W // s, s).mean(axis=(2, 4))
    # (a) decide at low resolution, then enlarge the label map
    lab_nn = np.kron(coarse.argmax(0), np.ones((s, s), int))
    # (b) enlarge the scores with bilinear interpolation, then decide per pixel
    up = np.stack([ndi.zoom(c, s, order=1, mode="nearest", grid_mode=True) for c in coarse])
    lab_bi = up.argmax(0)
    a, b = iou_per_class(gt, lab_nn, K), iou_per_class(gt, lab_bi, K)
    print(f"{s:6d} | {a.mean():.3f} {np.round(a, 2)} | {b.mean():.3f} {np.round(b, 2)}")
```

```
stride | nearest labels: mIoU (per class) | bilinear scores: mIoU (per class)
     4 | 0.906 [0.99 0.97 0.67 1.  ] | 0.994 [1.   1.   0.99 0.99]
     8 | 0.825 [0.98 0.93 0.75 0.64] | 0.883 [0.99 0.99 0.7  0.86]
    16 | 0.565 [0.95 0.89 0.   0.41] | 0.578 [0.97 0.97 0.   0.37]
    32 | 0.421 [0.92 0.76 0.   0.  ] | 0.473 [0.96 0.93 0.   0.  ]
```

這個小實驗有三個教訓（數字來自上面的程式，不是來自任何論文）。第一，對分數做雙線性上取樣幾乎不花成本，而且對大圓盤而言總是至少一樣好。第二，沒有任何上取樣能找回粗格點從來沒看見的東西：在步幅 16 時，6 像素寬的長條在每個格子裡都是少數而消失了；在步幅 32 時，20 像素的方塊也消失了。細小的結構需要小的輸出步幅，或是能把精細特徵帶回來的解碼器。第三，mean IoU 對這些損失的懲罰很重，因為長條與方塊和背景的權重一樣（圖 2）。

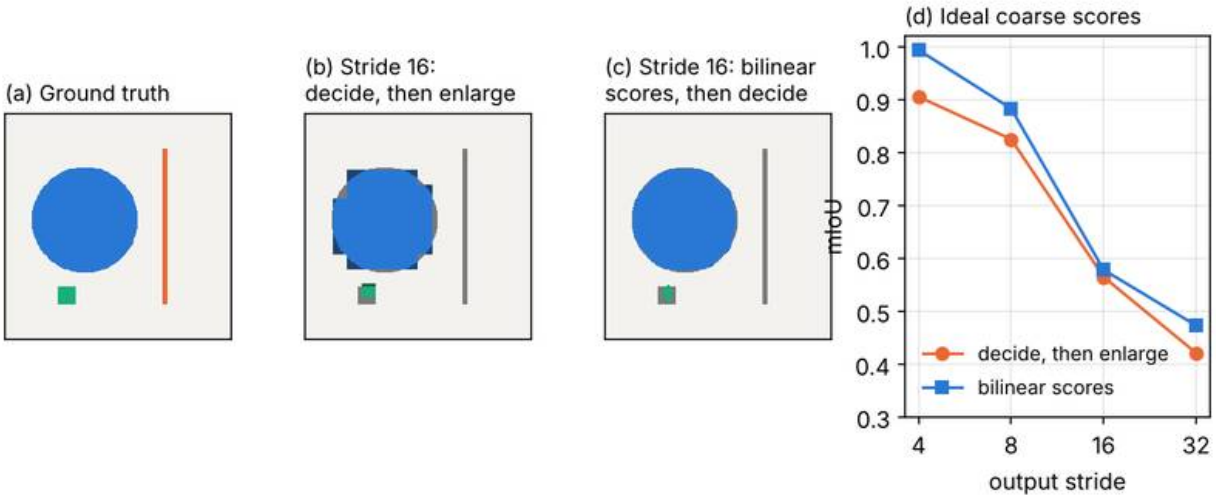


圖 2 — 即使有理想的粗分數，輸出步幅仍限制了分割模型能還原的內容。(*b, c*) 步幅 *16*；深色覆蓋處標示錯誤的像素。(*d*) 兩種上取樣順序在這張玩具圖上的 *mIoU*。

編碼器–解碼器與跳接

白話版。解碼器逐步提高解析度，並透過跳接重用對應的編碼器特徵以恢復細節。

U-Net [6] 把這個做法對稱化：一個由卷積與降取樣組成的收縮**編碼器**、一個由上取樣與卷積組成的擴張**解碼器**，並且在每一層用跳接（skip connection）把編碼器的特徵圖串接到解碼器。編碼器提供「是什麼」，跳接提供「在哪裡」。論文的設計選擇（無填補的卷積、重疊圖塊策略、為貼在一起的細胞所加的加權損失，以及強烈的彈性增強）在 IP 10 的 U-Net 導讀中說明。

選讀：*U-Net*：論文導讀（見 *IP 10* 的「論文導讀」頁）。論文的主張、有效卷積（*valid convolution*）的輸出尺寸算術，以及怎麼讀它的表格。

相反的設計則讓解碼器保持輕量。**DeepLabv3+** [7] 讓編碼器跑到輸出步幅 16，把特徵上取樣 4 倍，串接骨幹網路在步幅 4 的低階特徵（先經過 1×1 卷積減少通道數），再用幾個 3×3 卷積精修，最後再上取樣 4 倍。論文提出解碼器的動機，是指出單純的 16 倍雙線性上取樣，也就是「簡單的解碼器」（naive decoder），可能無法還原物體的細節。圖 3 比較三種形狀：FCN 少數幾張融合的分數圖、U-Net 完整的階梯，以及 DeepLab 的脈絡模組加上單一一條跳接。

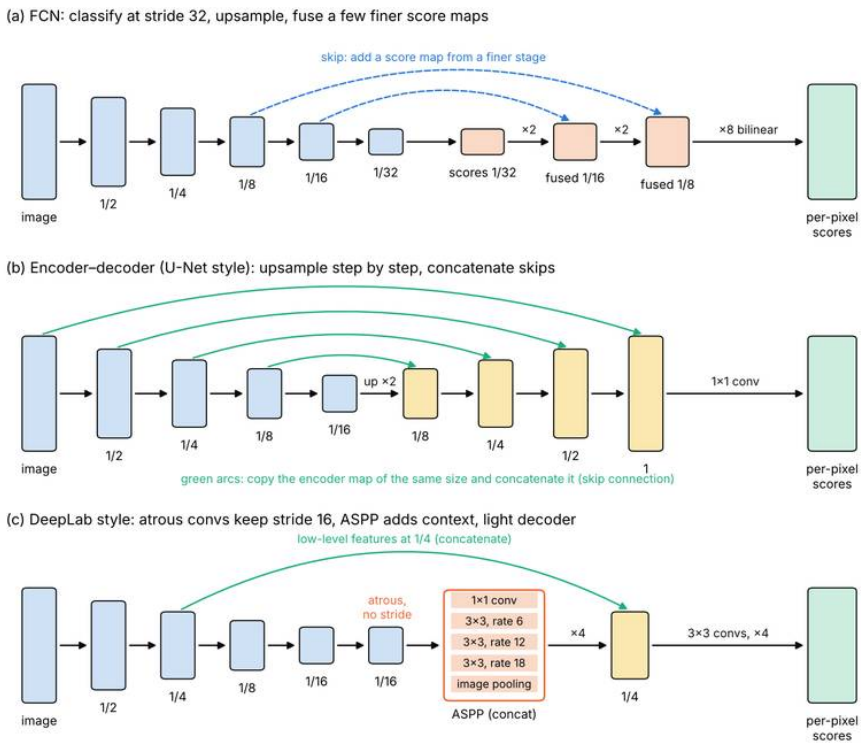


圖 3 — 從粗特徵回到像素的三種方法。(*a*) *FCN* 融合來自少數幾個較精細階段的分數圖。(*b*) *U-Net* 風格的編碼器–解碼器逐步上取樣，並串接大小相同的編碼器特徵圖。(*c*) *DeepLabv3+* 用空洞卷積維持步幅 *16*，用 *ASPP* 加入多尺度脈絡，並使用一條低階跳接。

不犧牲解析度的脈絡

白話版。要替一個像素貼標籤，需要兩樣方向相反的東西：寬廣的視野（這塊灰色是馬路還是牆？）和精細的解析度（馬路到底在哪裡結束？）。空洞卷積與池化金字塔能在不縮小特徵圖的情況下給你寬廣的視野。

空洞（atrous）卷積

降取樣是 CNN 擴大感受野（receptive field，一個輸出能看到的輸入區域）的慣用方法（DLCV 01 與 ML 03 有推導）。**空洞**（atrous；法文 “à trous”，意思是「有洞的」）卷積則是改把小卷積核的取樣點拉開。在一維情況下，DeepLab [8] 把輸入 x 與長度 K 的濾波器 w 做空洞卷積的輸出 y 定義為

$$y[i] = \sum_{k=1}^{K} x[i + r \cdot k]\, w[k],$$

其中 i 是輸出位置的索引，r 是**比率**（rate）：輸入每隔 r 個位置取樣一次。比率 $r = 1$ 就是一般的卷積。比率為 r 的 $k \times k$ 卷積核每邊涵蓋 $k_e = k + (k-1)(r-1)$ 個像素，而權重一樣是 k^2 個、計

算量也一樣 [8]。Yu 與 Koltun [9] 把同一個運算稱為膨脹卷積（dilated convolution），並證明堆疊膨脹率逐層加倍的卷積，能讓感受野呈指數成長而不損失解析度。

DeepLab 的做法，是把最後一個降取樣層的步幅設為 1，並把其後的卷積換成比率為 2 的空洞卷積，使它們用同樣的預訓練權重仍然看到同樣的影像區域；再多重複一個階段，密度又加倍 [8]。典型的模型輸出步幅是 8 或 16，而不是 32 [10]。代價是計算量與記憶體：在最後幾個階段，步幅 8 的位置數是步幅 16 的四倍。

```
import numpy as np

def atrous_1d(x, w, r):
    """y[i] = sum_k x[i + r*k] w[k] (DeepLab's Eq. 1), taps centred, zero padding."""
    K = len(w); c = K // 2
    xp = np.pad(x, r * c)
    return np.array([sum(xp[i + r * (k)] * w[k] for k in range(K)) for i in range(len(x))])

x = np.zeros(21); x[10] = 1.0                   # an impulse in the middle
w = np.array([1.0, 1.0, 1.0])
for r in [1, 2, 4]:
    print(f"rate {r}: taps at offsets {[(k - 1) * r for k in range(3)]},",
          "response", atrous_1d(x, w, r).astype(int))

def receptive_field(kernels, rates, strides):
    """Receptive field (in input pixels) of a stack of conv layers."""
    rf, jump = 1, 1                              # jump = distance between neighbouring
    ↪  outputs, in input pixels
    for k, r, s in zip(kernels, rates, strides):
        rf += (r * (k - 1)) * jump
        jump *= s
    return rf, jump

print("three 3x3, rates 1,1,1:", receptive_field([3, 3, 3], [1, 1, 1], [1, 1, 1]))
print("three 3x3, rates 1,2,4:", receptive_field([3, 3, 3], [1, 2, 4], [1, 1, 1]))
print("three 3x3, stride 2   :", receptive_field([3, 3, 3], [1, 1, 1], [2, 2, 2]))
```

```
rate 1: taps at offsets [-1, 0, 1], response [0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0]
rate 2: taps at offsets [-2, 0, 2], response [0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0]
rate 4: taps at offsets [-4, 0, 4], response [0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0]
three 3x3, rates 1,1,1: (7, 1)
three 3x3, rates 1,2,4: (15, 1)
three 3x3, stride 2   : (15, 8)
```

最後兩行就是空洞卷積的全部道理。比率為 1、2、4 的三層，看到的輸入像素數是 15，和三個步幅為 2 的層一模一樣；但它們每個輸入像素仍保有一個輸出（間隔 1），而不是每 8 個像素才一個（圖 4）。

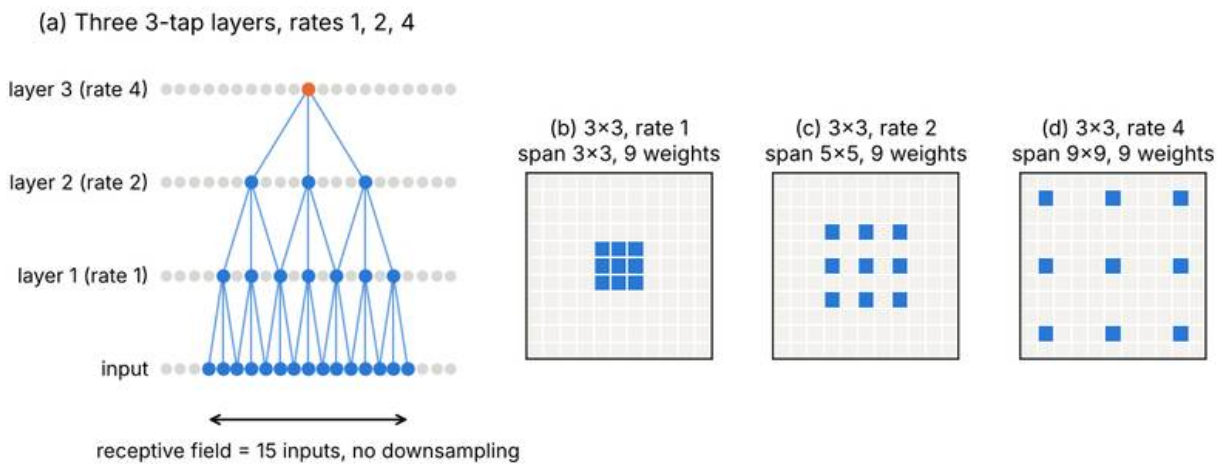


圖 *4* — 空洞卷積。(*a*) 堆疊比率 *1*、*2*、*4*，在完整解析度下涵蓋 *15* 個輸入。(*b–d*) 比率 *1*、*2*、*4* 的 *3×3* 卷積核，權重都是 *9* 個，但分別涵蓋 *3×3*、*5×5* 與 *9×9* 個像素。

膨脹有一個陷阱。單一膨脹層只讀取稀疏的輸入格點；而若堆疊的各層比率有共同因數，就會留下任何路徑都到不了的洞，形成棋盤格狀的瑕疵，Wang 等人稱之為**網格效應**（gridding）[11]。解法是選擇像 1、2、5 這樣能互相填補空隙的比率。

僅線上版：堆疊空洞卷積中的網格效應（*gridding*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/notes/堆疊空洞卷積中的網格效應 gridding/*

多尺度脈絡：ASPP 與金字塔池化

白話版。物體有各種大小，而一個像素是什麼，往往取決於它周圍是什麼。所以要同時透過好幾個大小不同的視窗看每個位置，其中包括一個看得到整張影像的視窗，再讓網路把這些答案結合起來。

空洞空間金字塔池化（atrous spatial pyramid pooling, ASPP）[8] 對同一張特徵圖平行地套用數個比率不同的空洞卷積，再融合它們的輸出，等於用好幾種有效尺度去探測影像。DeepLabv3 [10] 在輸出步幅 16 時，使用一個 1×1 卷積和三個比率為 6、12、18 的 3×3 卷積（步幅 8 時比率加倍），再加上一個**影像層級特徵**（image-level feature）：對整張特徵圖做全域平均池化、經過 1×1 卷積，再用雙線性內插放大回來。加入影像層級分支的原因是論文指出的一種邊界效應：當比率接近特徵圖的大小時，3×3 的取樣點會有越來越多落在特徵圖之外，濾波器退化成只用中心權重的 1×1 濾波器。下面的程式在 65×65 的特徵圖上數出這個現象。

```
import numpy as np

n = 65                                              # feature map is n x n
for r in [1, 6, 12, 18, 24, 36, 64]:
    pos = np.arange(n)
    inside_1d = np.array([[(0 <= p + r * d < n) for d in (-1, 0, 1)] for p in pos]).sum(1)
    mean_taps = np.mean(np.outer(inside_1d, inside_1d))   # taps inside the map, averaged
    ↪  over positions
```

```
    print(f"rate {r:2d}: mean valid taps {mean_taps:.2f} of 9")
```

```
rate  1: mean valid taps 8.82 of 9
rate  6: mean valid taps 7.93 of 9
rate 12: mean valid taps 6.92 of 9
rate 18: mean valid taps 5.98 of 9
rate 24: mean valid taps 5.11 of 9
rate 36: mean valid taps 3.58 of 9
rate 64: mean valid taps 1.06 of 9
```

在比率 64 時，一個「3×3」濾波器平均來說只剩下一個取樣點。全域池化是取得整張影像脈絡既便宜又可靠的方法。

金字塔場景解析網路（Pyramid Scene Parsing Network, PSPNet）[12] 用池化而不是膨脹達成同樣的目標。它的金字塔池化模組把最後的特徵圖池化成 1×1、2×2、3×3 與 6×6 的格點，各自用 1×1 卷積降低通道數，全部上取樣回來，再與原本的特徵串接。論文的動機來自 FCN 基線模型在場景解析上的錯誤，例如「關係不符」（mismatched relationship）：把河上的船標成車，儘管車子很少出現在河面上 [12]。

早期的 DeepLab 版本還會在網路之後，用全連接條件隨機場（conditional random field, CRF）來銳化邊界，它鼓勵顏色與位置相近的像素共用同一個標籤 [8]。DeepLabv3 不用它，也比先前的版本進步 [10]。

僅線上版：密集 *CRF* 後處理 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/notes/密集-crf-後處理/*

把注意力當成脈絡

Transformer 在每一層都讓每個 token 看得到所有其他 token，所以「脈絡」不再需要特別的模組（ML 05 說明了 Vision Transformer，即 ViT [13]）。問題出在解析度：一般的 ViT 只有一個粗糙的 token 格點，而對大量 token 做全域注意力的成本又很高。**SegFormer** [14] 是語意分割中被廣泛使用的解答。它的編碼器是階層式的，像 CNN 一樣產生多尺度特徵；使用縮減的注意力，縮短 key 與 value 的序列；而且不需要位置編碼，這樣在測試解析度與訓練時不同時，就不必內插位置編碼。它的解碼器只是幾層 MLP，用來融合多尺度特徵。階層式的視覺 Transformer 在 ML 05 說明。

密集預測的損失函數

白話版。訓練分割模型的方法，是把它逐像素的機率與真實的標籤圖相比。把分類損失在像素上取平均是預設做法，但當某個類別占了影像的 99%，這個平均幾乎只是在衡量那個容易的類別。

逐像素交叉熵。設 $p_{\mathbf{x}}(k)$ 為像素 $\mathbf{x}$ 上類別 k 的 softmax 機率，$y(\mathbf{x})$ 為真實類別，

$$\mathcal{L}_{\mathrm{CE}} = -\frac{1}{|\Omega'|}\sum_{\mathbf{x}\in\Omega'} w_{y(\mathbf{x})}\log p_{\mathbf{x}}(y(\mathbf{x})),$$

其中 Ω' 是有標註（非 void）的像素集合，w_k 是可選的類別權重，例如與類別出現頻率成反比。Void 像素直接不納入總和；多數框架用「ignore index」實作。

Dice 損失。 Milletari 等人 [15] 提出，與其重新加權樣本，不如直接最大化軟預測與二值真實標籤之間的 Dice 係數，因為感興趣的解剖結構常常只占醫學體積的一小部分，這時訓練很容易到處都預測成背景。對前景機率 p_i 與標籤 $g_i \in \{0,1\}$（共 N 個像素），他們的形式是

$$D = \frac{2\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2}, \qquad \mathcal{L}_{\text{Dice}} = 1 - D.$$

常見的變體在分母使用 $\sum p_i + \sum g_i$；當 p 與 g 都是二值時，兩者都等於重疊度量 $2|A \cap B|/(|A| + |B|)$，與 IoU 關係密切。對於多類別，Dice 損失逐類別計算再取平均。由於 D 是整張影像上的比值，小物體和大背景的份量一樣重。

Focal 損失。 Focal 損失 [16] 把每個像素的交叉熵乘上 $(1 - p_t)^{\gamma}$，其中 p_t 是真實類別的機率，讓容易的像素貢獻很少；它在 DLCV 03 中有推導。組合很常見：後面會談到的 MaskFormer，就用 focal 損失與 Dice 損失的加權和訓練每個二值遮罩 [17]。

下面的實驗在一張 64×64 的影像上比較這些損失，前景是一個 49 像素的圓盤，並對三個候選預測計算。「IoU@0.5」把機率以 0.5 為門檻二值化，再算出評估時會回報的 IoU。

```
import numpy as np

H = W = 64
yy, xx = np.mgrid[0:H, 0:W]
r2 = (yy - 32) ** 2 + (xx - 32) ** 2
g = (r2 <= 4 ** 2).astype(float)                # small foreground disk
print(f"foreground pixels: {int(g.sum())} of {g.size} ({100 * g.mean():.1f}%)")

def bce(p, g):
    return -np.mean(g * np.log(p) + (1 - g) * np.log(1 - p))

def weighted_bce(p, g):                          # weight each class by inverse frequency
    wf, wb = 0.5 / g.mean(), 0.5 / (1 - g.mean())
    return -np.mean(wf * g * np.log(p) + wb * (1 - g) * np.log(1 - p))

def focal(p, g, gamma=2.0):
    pt = np.where(g == 1, p, 1 - p)
    return -np.mean((1 - pt) ** gamma * np.log(pt))

def soft_dice_loss(p, g, eps=1e-6):
    return 1 - (2 * (p * g).sum() + eps) / (p.sum() + g.sum() + eps)

def iou_at_half(p, g):                           # the metric we actually report
    m = p > 0.5
    return (m & (g == 1)).sum() / (m | (g == 1)).sum()

preds = {
    "all background (p=0.05)":  np.full((H, W), 0.05),
    "right shape, timid (0.4)": np.where(g == 1, 0.4, 0.05),
    "bloated, confident (0.9)": np.where(r2 <= 8 ** 2, 0.9, 0.05),
}
```

```
print(f"{'prediction':26s} {'BCE':>6s} {'wBCE':>6s} {'focal':>6s} {'Dice':>6s}
 ↪  {'IoU@0.5':>8s}")
for name, p in preds.items():
    print(f"{name:26s} {bce(p, g):6.3f} {weighted_bce(p, g):6.3f} "
          f"{focal(p, g):6.3f} {soft_dice_loss(p, g):6.3f} {iou_at_half(p, g):8.3f}")

# Where does the BCE gradient come from? dL/dz = p - g per pixel (z = logit)
p = preds["all background (p=0.05)"]
grad = np.abs(p - g)
print(f"share of |BCE gradient| from background pixels: {grad[g == 0].sum() /
 ↪  grad.sum():.2f}")
```

```
foreground pixels: 49 of 4096 (1.2%)
prediction                   BCE   wBCE  focal   Dice  IoU@0.5
all background (p=0.05)     0.087  1.524  0.032  0.981    0.000
right shape, timid (0.4)    0.062  0.484  0.004  0.855    0.000
bloated, confident (0.9)    0.133  0.119  0.068  0.791    0.249
share of |BCE gradient| from background pixels: 0.81
```

逐列讀這張表（所有數字都來自這個小實驗）。一般的二元交叉熵（binary cross-entropy, BCE）寧可選「全部是背景」，也不選那個膨脹的預測；但膨脹的預測是唯一在 0.5 門檻下找到物體的。原因是膨脹的遮罩在大約 150 個背景像素上錯了，而這些像素的數量比物體還多。沒有類別權重 α 的 focal 損失，也做出同樣的選擇。有類別加權的 BCE 與 Dice 都把膨脹的預測排最好、「全部是背景」排最差，順序與 IoU 相同。最後一行說明了一般 BCE 為什麼會這樣：它 81% 的梯度來自本來就幾乎正確的背景像素。另外也請注意軟 Dice 對那個畏縮的預測有多嚴苛：在 $p = 0.05$ 的 4,047 個背景像素，讓分母增加約 202，是物體面積的四倍多，所以軟預測在背景上也必須很有把握。

實務上，交叉熵（常搭配類別權重或 focal 調變）是自然場景語意分割的預設；Dice 或 Dice ＋交叉熵的和，常用於有小結構的醫學影像；以遮罩為基礎的模型則用 Dice ＋ focal 訓練每個遮罩。也有同時懲罰邊界或拓撲錯誤的損失；其中三種以主動輪廓能量建立的，在 IP 11 的補充資料中有摘要。

僅線上版：軟 *Dice* 損失的梯度 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/notes/軟-dice-損失的梯度/*

衡量語意分割

白話版。在整個測試集上，數一數每個真實類別的像素被判成每個預測類別的各有多少。常見的分數都是這張表的不同摘要。標準的 mean IoU，對每個類別問：「在所有屬於這個類別、或被判成這個類別的像素中，有多少比例判對了？」再對類別取平均。

設 n_{ij} 為真實類別 i 被預測成類別 j 的像素數（在所有測試影像上累計），$t_i = \sum_j n_{ij}$ 為類別 i 的像素數。FCN [5] 定義了四個指標，類別數為 K：

$$\text{pixel accuracy} = \frac{\sum_i n_{ii}}{\sum_i t_i}, \qquad \text{mean accuracy} = \frac{1}{K}\sum_i \frac{n_{ii}}{t_i},$$

$$\text{mean IoU} = \frac{1}{K}\sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}}, \qquad \text{frequency-weighted IoU} = \frac{1}{\sum_k t_k}\sum_i t_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}}.$$

在類別 i 的 IoU 中，分子 n_{ii} 數的是真陽性，分母數的是真實像素與預測像素的聯集：t_i（真陽性加上漏掉的）加上 $\sum_j n_{ji} - n_{ii}$（誤報）。**Mean IoU（mIoU）**是 PASCAL VOC、Cityscapes 與 ADE20K 的標準指標。

```
import numpy as np

def confusion(gt, pred, K, ignore=255):
    """K x K matrix: rows = true class, columns = predicted class; 'ignore' pixels are
     ↪  skipped."""
    keep = gt != ignore
    return np.bincount(K * gt[keep] + pred[keep], minlength=K * K).reshape(K, K)

def seg_metrics(C):
    tp = np.diag(C).astype(float)
    t = C.sum(1)                    # pixels of each true class
    p = C.sum(0)                    # pixels predicted as each class
    iou = tp / (t + p - tp)
    return {"pixel acc": tp.sum() / C.sum(), "mean acc": np.mean(tp / t),
            "mIoU": iou.mean(), "fwIoU": (t * iou).sum() / t.sum(), "IoU": np.round(iou, 3)}

# Toy: road (0) covers most of the image, car (1) is medium, pole (2) is thin
H, W = 120, 160
gt = np.zeros((H, W), int)
gt[60:100, 40:90] = 1
gt[10:110, 130:133] = 2
pred = gt.copy()
pred[60:100, 86:90] = 0                          # car slightly too short
pred[10:110, 130:133] = 0                        # the pole is missed entirely
m = seg_metrics(confusion(gt, pred, 3))
for k, v in m.items():
    print(f"{k:9s}: {v:.3f}" if np.isscalar(v) else f"{k:9s}: {v}")
```

```
pixel acc: 0.976
mean acc : 0.640
mIoU     : 0.631
fwIoU    : 0.953
IoU      : [0.974 0.92  0.   ]
```

完全漏掉電線桿，讓像素準確率只掉 1.6%、在頻率加權 IoU 上幾乎沒有影響，卻讓 mean IoU 掉了三分之一。這就是為什麼對每個類別一視同仁的 mIoU 是主要數字。比較論文時有四個實務細節要注意：

- 混淆矩陣是在**整個資料集**上累計的，然後才計算 IoU。對每張影像各算 IoU 再平均，會得到不同的數字。

- Void 像素透過 ignore 標籤排除，如同上面的 `confusion`。
- mIoU 仍然偏袒類別內部的大物體：一輛大車的份量超過十輛小車。因此 Cityscapes 還報告實例層級的 **iIoU**，其中每個像素的貢獻，是以該類別的平均實例大小與其真實實例大小的比值來加權 [18]。
- 區域重疊對大物體的邊界品質不敏感，邊界上差幾個像素幾乎不會改變 IoU。**Boundary IoU** [19] 只在沿著輪廓的一條細帶中衡量重疊。

僅線上版：*Boundary IoU 詳解 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/notes/boundary-iou-詳解/*

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

實例分割

白話版。實例分割就是「附輪廓的偵測」。主流做法是先像偵測器那樣找出每個物體，再分割它方框的內部。

先偵測再分割：Mask R-CNN

Mask R-CNN [20] 在兩階段偵測器 Faster R-CNN [21] 上加了一個小型的全卷積遮罩分支；DLCV 03連同 RoIAlign 一起說明了 Faster R-CNN。對每個感興趣區域（region of interest），這個分支預測 K 個大小為 $m \times m$ 的二值遮罩，每個類別一個（論文建立在特徵金字塔網路 FPN [22] 上的版本是 28×28），損失使用逐像素 sigmoid，並且只對真實類別的遮罩計算平均二元交叉熵。因此遮罩預測與類別預測是**解耦**的：由方框分支選擇類別，不同類別的遮罩不互相競爭，這與 FCN 的逐像素 softmax 不同；論文的消融實驗發現，逐類別 sigmoid 比 softmax 帶來很大的提升 [20]。測試時，把預測類別的 $m \times m$ 遮罩縮放到方框大小，再以 0.5 為門檻二值化。

選讀：*RoI pooling 與 RoIAlign*（見 *DLCV 03* 的「補充資料」頁）。為什麼把方框捨入到特徵格點會讓遮罩位移，以及修正它的雙線性取樣。

固定的 28×28 格點也是 Mask R-CNN 的弱點：大物體的輪廓只由幾百個格子畫出，因此過度平滑。**PointRend** [23] 把遮罩預測當成電腦圖學中的算繪（rendering）：它從粗遮罩開始，反覆以雙線性內插上取樣，並且只在較密格點中最不確定的點（對二值遮罩來說，就是機率最接近 0.5 的點）重新計算預測，這些點大多沿著邊界分布。

另有其他路線。**由下而上**（bottom-up）的方法先預測逐像素的量，例如嵌入向量或到物體中心的距離，再把像素分組成實例；在預測的距離圖上做分水嶺，標記由網路找出，就是 IP 10 中標記控制分水嶺的學習版。還有**以查詢為基礎**（query-based）的方法，直接預測一組遮罩，詳見下面的遮罩分類一節。

Mask AP

實例分割的評分方式與偵測完全相同，都是 DLCV 03 的 COCO 平均精確度（average precision, AP），差別只在預測與真實物體之間的 IoU 是在遮罩上而不是方框上計算 [20]。這個指標每個遮

罩都需要一個信心分數，因為 AP 要對預測排序；而且它允許遮罩重疊：偵測器可以輸出數個互相重疊的假設，只有不相符的那些會被扣分。這就是為什麼實例輸出若不經額外處理，就不能直接當作全景圖使用。

全景分割

白話版。全景任務要求一幅一致的圖像：每個像素都有標籤，stuff 與 things 都一樣，沒有重疊，而且指標對天空區域和一輛車一視同仁。

任務與指標

Kirillov 等人 [4] 定義了這個任務（格式就是「三個任務，一張標籤圖」中給的那一種），並附上一個指標：**全景品質**（panoptic quality, PQ）。PQ 分兩步計算。

1. 配對。同一類別的預測區段 p 與真實區段 g，只有在它們的 IoU 嚴格大於 0.5 時才算配對。由於全景圖的區段不能重疊，這個配對是**唯一的**：如果 p_1 與 p_2 是互不相交的預測，則 $\mathrm{IoU}(p_1, g) + \mathrm{IoU}(p_2, g) \le 1$，因為每一個 $|p_i \cap g| / |p_i \cup g| \le |p_i \cap g| / |g|$，而且兩個交集沒有共同像素。所以每個 g 最多只有一個預測能超過 0.5，反過來也一樣 [4]。因此不需要最佳指派的步驟。

2. 評分。對每個類別，配對把區段分成真陽性 TP（配對成功的數對）、偽陽性 FP（未配對的預測）與偽陰性 FN（未配對的真實區段）。於是

$$\mathrm{PQ} = \frac{\sum_{(p,g)\in \mathrm{TP}} \mathrm{IoU}(p,g)}{|\mathrm{TP}| + \tfrac{1}{2}|\mathrm{FP}| + \tfrac{1}{2}|\mathrm{FN}|} = \underbrace{\frac{\sum_{(p,g)\in \mathrm{TP}} \mathrm{IoU}(p,g)}{|\mathrm{TP}|}}_{\text{segmentation quality (SQ)}} \times \underbrace{\frac{|\mathrm{TP}|}{|\mathrm{TP}| + \tfrac{1}{2}|\mathrm{FP}| + \tfrac{1}{2}|\mathrm{FN}|}}_{\text{recognition quality (RQ)}},$$

其中總和跨過配對成功的數對，$|\cdot|$ 計算集合的元素個數。**SQ** 是配對成功區段的平均 IoU，**RQ** 則是偵測中大家熟悉的 F1 分數。PQ 對每個類別分別計算再對類別取平均，所以稀有類別和常見類別的份量一樣；每個區段的份量也相同，不論面積大小。$\mathrm{PQ}^{\mathrm{St}}$ 與 $\mathrm{PQ}^{\mathrm{Th}}$ 分別只對 stuff 與 thing 類別取平均。配對之前，void 像素會先從預測中移除，而主要由 void（或同類別的「crowd」區域）組成的未配對預測，不算偽陽性 [4]。這兩個因子並不獨立，因為 SQ 只在配對成功的區段上衡量，所以一定要把三個數字都報告出來。

下面的程式在圖 5 的玩具場景上實作這個配對與評分（void 的處理簡化到最簡單的形式）。

```
import numpy as np

def panoptic_quality(gt_seg, gt_cls, pr_seg, pr_cls):
    """gt_seg/pr_seg: integer segment-id maps (0 = void); *_cls: dict segment id -> class.
    Matches segments of the same class with IoU > 0.5 and returns per-class (PQ, SQ, RQ, TP,
     ↪  FP, FN)."""
    out = {}
    for c in sorted(set(gt_cls.values()) | set(pr_cls.values())):
        g_ids = [i for i, k in gt_cls.items() if k == c]
        p_ids = [i for i, k in pr_cls.items() if k == c]
        ious, matched_p = [], set()
        for gi in g_ids:
```

```
            G = gt_seg == gi
            for pj in p_ids:
                P = (pr_seg == pj) & (gt_seg != 0)          # ignore predicted pixels on void
                iou = (G & P).sum() / (G | P).sum()
                if iou > 0.5:                                # unique by the matching theorem
                    ious.append(iou); matched_p.add(pj)
        tp = len(ious); fn = len(g_ids) - tp; fp = len(p_ids) - len(matched_p)
        sq = np.mean(ious) if tp else 0.0
        rq = tp / (tp + 0.5 * fp + 0.5 * fn)
        out[c] = (sq * rq, sq, rq, tp, fp, fn)
    return out

H, W = 100, 200
SKY, ROAD, CAR, PERSON = 0, 1, 2, 3
def scene(boxes, sky_rows):
    seg = np.full((H, W), 2); seg[:sky_rows] = 1          # segment 1 = sky, 2 = road
    cls = {1: SKY, 2: ROAD}
    for sid, (c, r0, r1, c0, c1) in enumerate(boxes, start=3):
        seg[r0:r1, c0:c1] = sid; cls[sid] = c
    return seg, cls

gt_seg, gt_cls = scene([(CAR, 50, 80, 10, 55), (CAR, 50, 80, 65, 110),
                        (CAR, 38, 52, 125, 150), (PERSON, 30, 80, 165, 185)], sky_rows=30)
pr_seg, pr_cls = scene([(CAR, 52, 80, 13, 55),             # car A, slightly off
                        (CAR, 55, 80, 65, 96),             # car B, too small
                        (CAR, 84, 97, 110, 150),           # a car that is not there
                        (PERSON, 30, 52, 165, 185),        # the person, split in two
                        (PERSON, 52, 80, 165, 185)], sky_rows=33)   # car C is missed
names = ["sky", "road", "car", "person"]
res = panoptic_quality(gt_seg, gt_cls, pr_seg, pr_cls)
for c, (pq, sq, rq, tp, fp, fn) in res.items():
    print(f"{names[c]:6s} PQ {pq:.3f} = SQ {sq:.3f} x RQ {rq:.3f}  (TP {tp}, FP {fp}, FN
     ↪ {fn})")
pqs = np.array([v[0] for v in res.values()])
print(f"PQ {pqs.mean():.3f} | PQ_stuff {pqs[:2].mean():.3f} | PQ_things
 ↪ {pqs[2:].mean():.3f}")
```

```
sky    PQ 0.917 = SQ 0.917 x RQ 1.000  (TP 1, FP 0, FN 0)
road   PQ 0.805 = SQ 0.805 x RQ 1.000  (TP 1, FP 0, FN 0)
car    PQ 0.482 = SQ 0.723 x RQ 0.667  (TP 2, FP 1, FN 1)
person PQ 0.373 = SQ 0.560 x RQ 0.667  (TP 1, FP 1, FN 0)
PQ 0.644 | PQ_stuff 0.861 | PQ_things 0.428
```

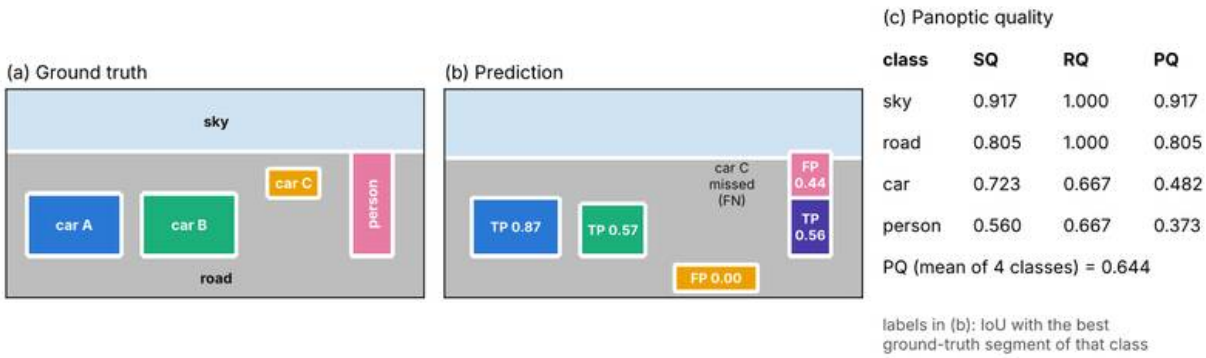


圖 *5* — 玩具場景上的全景品質。每個預測區段都標上它與同類別真實區段的最佳 *IoU*；只有 *IoU* 大於 *0.5* 才算配對。

這個小例子顯示 PQ 如何解讀錯誤（數字來自這次執行）。被切成兩塊的人仍然得到一個配對，因為下半塊涵蓋了他的 56%，但上半塊變成偽陽性：把一個物體切開，傷到的是辨識品質，而不只是分割品質。車 B 的 IoU 是 0.57，被算作完全辨識出來，但它偏低的 IoU 拉低了車子的 SQ。漏掉一輛車與憑空多出一輛車，各讓分母多半個區段。

早期全景系統：合併兩個輸出

白話版。最早的全景系統，就是把語意分割模型與實例分割模型各跑一次，再把結果拼貼在一起。

全景論文自己的基線 [4] 用一種類似非極大值抑制（non-maximum suppression）的做法來做。它依信心分數對實例遮罩排序，丟掉分數低的，然後沿著清單往下走：每個遮罩要扣掉前面遮罩已經占走的像素，只有剩下的部分夠多才保留。留下來的 thing 區段再與語意圖結合，同時被 thing 與 stuff 類別宣稱的像素歸給 thing。**Panoptic FPN** [24] 接著把兩個任務放進同一個網路：一個帶特徵金字塔網路的 Mask R-CNN（FPN [22]，見 DLCV 03），加上一個輕量的語意分支，把所有金字塔層級上取樣到 1/4 尺度後相加。語意分支預測 stuff 類別，再加上一個代表所有 thing 像素的「other」類別，這樣兩個分支就不會為了物體互相爭奪。

僅線上版：*Panoptic Segmentation*：論文導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/papers/panoptic-segmentation* 論文導讀/

遮罩分類：一個模型解決三種任務

白話版。與其問「這個像素是什麼類別？」，不如問「有哪些區域，而每個區域是什麼？」模型預測固定數量的候選遮罩，並替每個遮罩給出一個類別（或「什麼都不是」）。語意圖、一組實例與全景圖，就只是用不同方式讀同一份清單。

MaskFormer

Cheng、Schwing 與 Kirillov [17] 把 FCN 的做法稱為**逐像素分類**（per-pixel classification），把 Mask R-CNN 與 DETR 的做法稱為**遮罩分類**（mask classification）。他們的觀察是：遮罩分類的通用性足以用在語意分割。**MaskFormer** 預測一組 N 個機率–遮罩數對 $\{(p_i, m_i)\}_{i=1}^{N}$，其中 p_i 是 K 個類別加上一個「無物體」標籤 $\emptyset$ 上的分布，$m_i \in [0,1]^{H \times W}$ 是軟遮罩。三個模組產生它們：

1. 一個骨幹網路與一個**像素解碼器**（pixel decoder）產生逐像素嵌入 $\mathcal{E}_{\text{pixel}} \in \mathbb{R}^{C_E \times H \times W}$，其中 C_E 是嵌入維度。
2. 一個 **Transformer 解碼器**把 N 個可學習的查詢（query）轉成 N 個逐區段嵌入，方法是像 DETR [25] 那樣對影像特徵做注意力。
3. 一個線性分類器給出 p_i，一個小型 MLP 給出遮罩嵌入 $\mathcal{E}_{\text{mask}}[:, i] \in \mathbb{R}^{C_E}$。遮罩是一個內積再經過 sigmoid，

$$m_i[h, w] = \sigma(\mathcal{E}_{\text{mask}}[:, i]^{\top}\, \mathcal{E}_{\text{pixel}}[:, h, w]),$$

其中 σ 是 logistic sigmoid，(h, w) 是像素位置。

訓練時，預測與真實區段做一對一配對，如同 DETR 對方框做的那樣（DLCV 03），只是配對成本由類別機率與遮罩損失構成，而不是方框。配對到的預測得到交叉熵類別損失與 focal + Dice 遮罩損失；沒配對到的則被訓練去預測 $\emptyset$ [17]。

推論時，兩條規則以不同方式讀同一份輸出 [17]。**語意推論**對查詢做邊際化（marginalize），$\hat{y}[h, w] = \arg\max_c \sum_i p_i(c)\, m_i[h, w]$，其中 c 遍歷 K 個真實類別。**全景推論**把每個像素指派給一個查詢，$\arg\max_{i:\, c_i \neq \emptyset} p_i(c_i)\, m_i[h, w]$，其中 c_i 是查詢 i 最可能的類別，因此同一類別的兩個查詢會給出兩個實例。下面的程式在一個小小的 6×10 例子上跑這兩條規則。

```
import numpy as np

rng = np.random.default_rng(0)
# Ground-truth layout of a tiny 6 x 10 image: sky on top, road below, two cars on the road
region = np.zeros((6, 10), int)               # 0 sky, 1 road, 2 car A, 3 car B
region[2:] = 1
region[3:5, 1:4] = 2
region[3:5, 6:9] = 3

# Pixel decoder output: a C_E = 4 dimensional embedding per pixel (here: noisy one-hot of the
 ↪  region)
E_pixel = 6 * np.eye(4)[region].transpose(2, 0, 1) - 3 + 0.5 * rng.standard_normal((4, 6,
 ↪  10))

# Transformer decoder output for N = 5 queries: a mask embedding and class probabilities
# over K = 3 classes (sky, road, car) plus "no object" (last column)
E_mask = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1], [.2, .2, .2, .2]],
 ↪  float)
p = np.array([[.90, .05, .03, .02],          # query 0: sky
              [.05, .85, .05, .05],          # query 1: road
              [.02, .08, .85, .05],          # query 2: a car
              [.03, .07, .75, .15],          # query 3: another car
              [.05, .05, .05, .85]])         # query 4: nothing
```

```
m = 1 / (1 + np.exp(-np.einsum("nc,chw->nhw", E_mask, E_pixel)))   # N masks, sigmoid(dot
 ↪  product)

# Semantic inference: sum over queries of p_i(c) * m_i, then argmax over the K real classes
sem = np.einsum("nk,nhw->khw", p[:, :3], m).argmax(0)
# Panoptic inference: each pixel goes to the query maximizing p_i(c_i) * m_i (no-object
 ↪  queries dropped)
c_i = p.argmax(1)
keep = c_i != 3
score = p[np.arange(5), c_i][:, None, None] * m
seg = np.where(keep[:, None, None], score, -1).argmax(0)
print("semantic (0 sky, 1 road, 2 car):\n", sem)
print("panoptic segment = query index:\n", seg)
print("query classes:", c_i, "-> car segments:", [int(i) for i in np.unique(seg) if c_i[i] ==
 ↪  2])
```

```
semantic (0 sky, 1 road, 2 car):
 [[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1]
 [1 2 2 2 1 1 2 2 2 1]
 [1 2 2 2 1 1 2 2 2 1]
 [1 1 1 1 1 1 1 1 1 1]]
panoptic segment = query index:
 [[0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0]
 [1 1 1 1 1 1 1 1 1 1]
 [1 2 2 2 1 1 3 3 3 1]
 [1 2 2 2 1 1 3 3 3 1]
 [1 1 1 1 1 1 1 1 1 1]]
query classes: [0 1 2 2 3] -> car segments: [2, 3]
```

語意圖把兩輛車合併成類別 2；全景圖則把它們分開，成為區段 2 與 3，而兩者來自完全相同的網路輸出。論文報告 MaskFormer 在 Cityscapes（19 個類別）上與逐像素分類相當，在詞彙量較大的資料集上則更好 [17]。

Mask2Former 與其後繼者

Mask2Former [26] 保留 MaskFormer 的整體架構（骨幹網路、像素解碼器、Transformer 解碼器），改動解碼器，使它訓練得更快，並且能把實例分割得很好。它的關鍵元件是**遮罩注意力**（masked attention）：每個查詢只對前一層所預測的遮罩內部的像素做注意力。在第 l 層，查詢特徵為 $X_l \in \mathbb{R}^{N\times C}$，

$$X_l = \mathrm{softmax}(\mathcal{M}_{l-1} + Q_l K_l^\top) V_l + X_{l-1}, \qquad \mathcal{M}_{l-1}(x,y) = \begin{cases} 0 & \text{if } M_{l-1}(x,y) = 1, \\ -\infty & \text{otherwise,} \end{cases}$$

其中 Q_l 是 X_{l-1} 的線性投影，K_l 與 V_l 是影像特徵的投影，M_{l-1} 是前一層的遮罩預測，經縮放並以 0.5 為門檻二值化。在 softmax 之前加上 $-\infty$，會讓遮罩之外的注意力權重為零。作者的假設

是：局部特徵就足以更新一個查詢，而先前模型收斂緩慢，是因為交叉注意力必須先學會把焦點放在物體上。另外兩項改動，一是在連續的解碼器層中輪流使用像素解碼器 1/32、1/16 與 1/8 的特徵圖，以利小物體；二是在取樣的點上而不是完整遮罩上計算遮罩損失，論文報告這讓訓練記憶體減少為三分之一 [26]。

僅線上版：*Mask2Former*：論文導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-segmentation/papers/mask2former* 論文導讀/

後來有兩個方向建立在這個設計上。**OneFormer** [27] 用單一模型訓練三種任務：它從全景標註推導出語意與實例標籤，每張訓練影像隨機抽一個任務，並以「任務 token」作為查詢的條件，讓同一組權重服務所有任務。**EoMT** [28] 則把任務專用的部分全部拿掉。作者主張，經過大量預訓練的大型一般 ViT，能自行學會轉接器（adapter）、像素解碼器與 Transformer 解碼器原本提供的東西；他們在最後幾個 ViT 區塊之前，把可學習的查詢串接到圖塊 token 上，讓這些區塊同時處理兩者，並報告了與較複雜模型相近的準確率，而且速度快很多（用 ViT-L 時最多快四倍）。

可提示與開放詞彙分割

白話版。上面的模型只分割訓練時固定的類別清單。兩個較新的家族放寬了這份清單：可提示模型分割你指到的任何東西；開放詞彙模型分割你叫得出名字的任何東西。

可提示分割。Segment Anything Model（SAM）[29] 針對點、方框或粗略遮罩的提示傳回一個遮罩；遇到含糊的點（是襯衫還是人？），它會傳回數個有效的遮罩及其信心分數。它在 SA-1B 上訓練，這是一個用模型參與迴圈（model-in-the-loop）的資料引擎建成的資料集，在 1,100 萬張影像上有超過 10 億個遮罩。SAM 2 [30] 用串流記憶（streaming memory）把它擴展到影片，SAM 3 [31] 則加入**可提示概念分割**（promptable concept segmentation）：給一個簡短的名詞片語或影像範例，它會傳回所有相符實例的遮罩與身分。這些模型在 DLCV 01 與 ML 10 有說明，SAM 2 當作追蹤器則在 DLCV 07。以本篇的三種任務來看：SAM 的遮罩**沒有類別**，所以單獨來看，它既不是語意也不是全景的輸出；偵測器給的方框加上 SAM，可以得到實例遮罩；而 SAM 3 的概念提示則讓它成為開放詞彙的實例分割模型。閱讀這類模型的「零樣本」（zero-shot）結果時，要檢查提示是從哪裡來的：從真實方框推導出來的提示，給了模型很多資訊。

開放詞彙分割。關鍵成分是 CLIP [32] 這類視覺–語言模型，它的文字編碼器把任何類別名稱對應到與影像特徵相同的空間裡的嵌入（ML 10 有介紹）。把分割模型固定的分類器權重換成類別名稱的文字嵌入，詞彙就變成測試時才輸入的清單。**LSeg** [33] 做的正是這件事：Transformer 影像編碼器產生逐像素嵌入，文字編碼器嵌入標籤名稱，訓練則讓每個像素的嵌入與其類別的文字嵌入對齊，所以測試時不必重新訓練就能使用沒見過的標籤。**CAT-Seg** [34] 則改為建立一個**代價體積**（cost volume），裡面是 CLIP 影像與文字嵌入之間的餘弦相似度，並加以聚合；作者發現這讓他們能微調 CLIP 的編碼器而不會過擬合到訓練類別，而這個問題曾使多數先前的方法選擇凍結 CLIP。Zhu 與 Chen 的綜述 [35] 提醒，評估「未見過」的類別很困難，因為影像–文字預訓練資料可能早已包含它們。

資料集與基準

白話版。逐像素標註很昂貴，所以由少數幾個公開資料集撐起整個領域。每個資料集都固定了類別清單、stuff ／ thing 的劃分，以及評估方式。

資料集	內容	規模與類別（來自論文）	任務
PASCAL VOC 2012 [36]	日常物體	20 個物體類別＋背景（如 [8] 所述）	語意
Cityscapes [18]	來自 50 個城市的街景	5,000 張精細標註與 20,000 張粗略標註的影像；30 個類別，其中 19 個用於評估；8 個有實例標籤 [4]	語意、實例、全景
ADE20K [37]	多樣的場景、物體與部件	訓練／驗證／測試各 20,210 / 2,000 / 3,000 張影像；基準取前 150 個類別（SceneParse150）；全景用 100 個 things ＋ 50 個 stuff [4]	語意、全景
COCO [38] + COCO-Stuff [39]	日常場景	80 個 thing 類別的實例遮罩 [24]；COCO-Stuff 為 COCO 2017 全部 164K 張影像加上 91 個 stuff 類別的逐像素標籤；COCO 全景有 80 個 things ＋ 53 個 stuff [17]	實例、語意、全景
Mapillary Vistas [40]	世界各地的街道層級影像	25k 張影像；28 個 stuff ＋ 37 個 thing 類別 [4]	語意、實例、全景

有三個事實影響每一項結果。第一，**標註昂貴且有雜訊**：全景論文的人類一致性研究，讓兩份對同一批影像的標註互相評分，在它檢視的三個資料集上 PQ 值都遠低於 100 [4]。第二，**類別清單很小**，與世界上的詞彙量相比，這正是上面開放詞彙模型的動機。第三，**領域不同**：Cityscapes 是在 50 個城市錄製的，主要在德國，而且刻意避開惡劣天氣 [18]，所以用它訓練的模型到了別處，會遇到不同的路面標線、天氣與相機。駕駛用的鳥瞰圖（bird's-eye-view）地圖分割，是在地平面格點而不是影像上預測標籤，在 DLCV 03B 說明。

失效模式與實務建議

白話版。分割模型會在可預期的地方失敗：結構細小之處、邊界、稀有類別，以及與訓練集不同的影像。

- **細小的結構**（電線桿、電線、遠處的行人）在大的輸出步幅下會消失，如第一個實驗所示。使用輸出步幅 8，或帶有高解析度跳接的解碼器，並且評估時看每個類別的 IoU，而不只是 mIoU。
- **邊界**會變得平滑或呈方塊狀。對分數做雙線性上取樣、解碼器、PointRend [23] 這類以點為基礎的精修，以及對邊界敏感的指標 [19]，都能處理這個問題。
- **類別不平衡**讓網路偏向頻繁的類別。加權交叉熵、focal 或 Dice 項，以及取樣含有稀有類別的影像，都有幫助（見損失函數的實驗）。
- **脈絡錯誤**：某個區域被標成在周圍環境下說不通的類別（[12] 中被標成車的船）。全域脈絡模組與注意力能減少這類錯誤。
- **破碎的遮罩與破洞**：逐像素分類器沒有形狀的概念。對預測遮罩做形態學清理（開運算、閉運算、填洞；見 IP 09）是便宜而常見的後處理步驟。
- **高解析度輸入**會超過 GPU 記憶體。在重疊的圖塊上執行再混合分數，與 U-Net 的重疊圖塊是同一個想法。

- **領域偏移**：換到新的城市、天氣、感測器或醫學掃描儀，效能會下降。要用留出的領域來衡量，而不只是留出的資料切分。

現代觀點

綜述怎麼說

Minaee 等人 [1] 把語意與實例分割的 CNN 時代整理成十個家族：全卷積網路、搭配圖模型（如 CRF）的卷積模型、編碼器–解碼器、多尺度與金字塔網路、用於實例的 R-CNN 系模型、膨脹卷積與 DeepLab 家族、遞迴網路、注意力模型、生成式與對抗式模型，以及結合主動輪廓的卷積模型。他們也回顧了資料集與已發表的結果。大多數家族都出現在本篇各節，以 FCN、U-Net、PSPNet、DeepLab 與 Mask R-CNN 為例。

Li 等人 [2] 圍繞著一個**整體架構**（meta-architecture）來整理以 Transformer 為基礎的分割，這個架構有特徵擷取器、物體查詢與 Transformer 解碼器；他們依方法修改了哪個元件來分類：更好的骨幹網路、解碼器中的交叉注意力設計、最佳化查詢的方式，等等。他們也涵蓋影片與點雲分割、基礎模型的調校、效率與醫學分割，並在共同的基準上重新評估許多方法。

Zhou 等人 [3] 回顧基礎模型時代的 300 多種方法，把領域分成**通用**分割（語意、實例、全景）與**可提示**分割（互動式、指稱式與少樣本）。一個核心主題是：分割的知識會在沒有用遮罩訓練的模型中自然浮現，例如 CLIP、文字到影像的擴散模型與自監督 ViT，免訓練的方法就利用這一點。**Zhu 與 Chen** [35] 則聚焦於開放詞彙的偵測與分割，以及公平評估「新」類別有多困難。

2023–2026 的最新技術

- **通用的查詢式架構**是封閉詞彙基準上的預設：Mask2Former [26] 與它的後代如 OneFormer [27]，用一種設計處理語意、實例與全景分割；在速度或簡單性重要的地方，專門的逐像素模型仍然很常見（SegFormer [14]、DeepLab 風格的模型 [7]）。
- **更簡單的模型搭配更大的預訓練**：EoMT [28] 主張，把算力花在擴大一般 ViT 編碼器上，比加上解碼器與轉接器更划算。
- **可提示的基礎模型**（SAM [29]、SAM 2 [30]、SAM 3 [31]）把與類別無關的遮罩、影片傳播與概念提示變成可重複使用的元件，它們的資料引擎也改變了分割資料集的建造方式。
- **開放詞彙分割**建立在視覺–語言模型上（LSeg [33]、CAT-Seg [34]）；在許多應用中，現在是用類別名稱來提示模型，而不是重新訓練。

開放問題

1. **邊界、小物體與細小結構。** Mask2Former 的作者承認它仍然不擅長小物體 [26]，而區域指標會掩蓋大物體上的邊界錯誤 [19]。
2. **一個模型，只訓練一次。**通用架構仍常常針對每個任務與資料集分別訓練；只用全景標籤訓練的 Mask2Former，在實例與語意指標上，略差於針對各任務分別訓練的版本 [26]。
3. **開放詞彙的評估**取決於提示，以及預訓練資料包含了什麼 [35]。

4. **資料**。逐像素標註很昂貴；綜述指出可擴展的資料引擎、由生成式模型產生的合成影像–遮罩配對，以及弱監督或無監督，是可行的方向，但各有自己的領域差距 [3], [2]。
5. **效率**。基礎模型的分割器很笨重；蒸餾、壓縮與參數高效率的調校，是尚未解決的工程問題 [3]。
6. **理解「自然浮現」**。為什麼沒有用遮罩訓練的模型會學會分割，目前還沒有好的解釋 [3]。

重點整理

- 先定義輸出需要類別、物件身分，或兩者兼具。
- 輸出步幅限制空間細節；應先上採樣分數，再決定標籤。
- 帶跳接的解碼器恢復細節，空洞卷積、池化與注意力提供上下文。
- 交叉熵、焦點與 Dice 損失對錯誤的加權不同，須配合任務不平衡與指標選擇。
- mIoU、mask AP 與全景品質評估不同輸出結構，不能互換。
- 遮罩分類可支援多種任務；提示與詞彙介面會改變需要評估的內容。

練習

1. **用手算輸出步幅**。一張路景是 1024×2048 像素，電線桿寬 6 個像素。在理想的粗分數（各格的類別比例）與 argmax 決策下，電線桿在哪些輸出步幅還有機會存活？為什麼它相對於格點的位置很重要？把第一段程式中的長條移動一兩個像素來檢驗你的答案。

   提示

   電線桿只有在涵蓋超過格子寬度一半時才贏得那一格，所以 $s < 12$；$s = 8$ 時，只有在它的 6 欄中至少有 5 欄落在同一個 8 像素格子裡的那些格子，它才存活。$s = 4$ 時，它至少總是涵蓋一整個格子。移動長條會改變它主導哪些格子，這就是為什麼程式最近鄰標籤那一欄中，長條在步幅 4 的 IoU 是 0.67 而不是 1（剛好各占一半的平手，歸給背景，也就是第一個類別）。

2. **空洞卷積的算術**。(a) 比率 12 的 3×3 卷積核，有效大小是多少？(b) 骨幹網路的一個階段常以步幅 2 開始，後面接 3×3 卷積。如果拿掉步幅，改用比率 2，輸出步幅、後續各層的感受野，以及那些層的計算成本，分別會怎麼變？

   提示

   (a) $k_e = 3 + 2 \cdot 11 = 25$。(b) 輸出步幅減半（例如 32 變成 16），之後每一層涵蓋的影像區域與先前相同（比率補償了較密的格點），而且之後的各層現在要在四倍多的位置上運算，所以成本約增加為四倍。

3. **空遮罩下的損失**。修改損失函數的程式，讓影像中沒有前景。對「all background (p=0.05)」這個預測，每個損失各傳回什麼？沒有 `eps` 項時，軟 Dice 損失會發生什麼事？實作上通常怎麼避免這個問題？

提示

當 $g = 0$ 處處成立，Dice 的分子是 0（加上 `eps`），分母是 $\sum p_i = 204.8$，所以無論預測多好，損失都幾乎是 1；若處處 $p = 0$，沒有 `eps` 時是 0/0，有 `eps` 時變成 $1 - \text{eps}/\text{eps} = 0$。當兩個遮罩都沒有任何像素時，`iou_at_half` 也會 0 除以 0。常見的修正：在分子與分母加平滑常數、對整個批次而不是每張影像計算 Dice，或是把 Dice 與交叉熵結合。

4. **用手算 PQ**。在 PQ 的程式中，把兩塊人的預測換成單一個涵蓋第 32 到 80 列的預測。在執行程式之前，先預測新的人的 SQ、RQ 與 PQ。然後讓車 B 變大，過度分割到它的 IoU 掉到 0.5 以下；車子的 TP、FP 與 FN 個數怎麼變？

提示

單一的人預測，IoU 是 $48 \cdot 20/(50 \cdot 20) = 0.96$，所以 SQ = 0.96、RQ = 1、PQ = 0.96。如果車 B 的 IoU 降到 0.5 或以下，這一對就不再算配對：車子的 TP 從 2 降到 1，FP 與 FN 各增加 1。

5. **語意與全景之間的互相轉換**。用遮罩分類的程式：(a) 從全景區段圖推導出語意圖，並檢查它等於 `sem`；(b) 讓查詢 3 以 0.6 的機率預測「road」而不是「car」，並解釋兩個輸出。(c) 為什麼一般而言，語意圖不能轉回全景圖？

提示

(a) 把每個區段編號對應到它的類別，`c_i[seg]`。(b) 查詢 3 的遮罩仍然涵蓋車 B，所以在全景圖中車 B 仍是區段 3，但這個區段現在被標成 road；由於 stuff 類別在一張影像中只有一個區段，最後的輸出會把它和馬路合併。在語意圖中，車 B 也變成 road。兩種輸出裡都只剩下一輛車。(c) 語意圖沒有實例編號：貼在一起的同類物體形成一個區域，要把它分開，需要圖裡沒有的資訊。

參考文獻

1. S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, “Image segmentation using deep learning: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, 2022. doi ·arXiv
2. X. Li, H. Ding, H. Yuan, W. Zhang, J. Pang, G. Cheng, K. Chen, Z. Liu and C. C. Loy, “Transformer-based visual segmentation: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, 2024. doi ·arXiv
3. T. Zhou, W. Xia, F. Zhang, B. Chang, W. Wang, Y. Yuan, E. Konukoglu and D. Cremers, “Image segmentation in foundation model era: A survey,” arXiv:2408.12957, 2024. arXiv
4. A. Kirillov, K. He, R. Girshick, C. Rother and P. Dollár, “Panoptic segmentation,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. doi ·arXiv
5. J. Long, E. Shelhamer and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. doi · arXiv
6. O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional networks for biomedical image segmentation,” in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, LNCS vol. 9351, 2015. doi

7. L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proc. European Conference on Computer Vision* (*ECCV*), 2018. doi ·arXiv
8. L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, 2018. doi · arXiv
9. F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *Proc. International Conference on Learning Representations* (*ICLR*), 2016. arXiv
10. L.-C. Chen, G. Papandreou, F. Schroff and H. Adam, "Rethinking atrous convolution for semantic image segmentation," arXiv:1706.05587, 2017. arXiv
11. P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou and G. Cottrell, "Understanding convolution for semantic segmentation," in *Proc. IEEE Winter Conference on Applications of Computer Vision* (*WACV*), 2018. doi ·arXiv
12. H. Zhao, J. Shi, X. Qi, X. Wang and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2017. doi ·arXiv
13. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. International Conference on Learning Representations* (*ICLR*), 2021. arXiv
14. E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez and P. Luo, "SegFormer: Simple and efficient design for semantic segmentation with transformers," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2021. NeurIPS ·arXiv
15. F. Milletari, N. Navab and S.-A. Ahmadi, "V-Net: Fully convolutional neural networks for volumetric medical image segmentation," in *Proc. Fourth International Conference on 3D Vision* (*3DV*), 2016. doi ·arXiv
16. T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE International Conference on Computer Vision* (*ICCV*), 2017. doi
17. B. Cheng, A. G. Schwing and A. Kirillov, "Per-pixel classification is not all you need for semantic segmentation," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2021. NeurIPS · arXiv
18. M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The Cityscapes dataset for semantic urban scene understanding," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2016. doi ·arXiv
19. B. Cheng, R. Girshick, P. Dollár, A. C. Berg and A. Kirillov, "Boundary IoU: Improving object-centric image segmentation evaluation," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2021. doi ·arXiv
20. K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," in *Proc. IEEE International Conference on Computer Vision* (*ICCV*), 2017. doi ·arXiv
21. S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, 2017. doi
22. T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2017. doi

23. A. Kirillov, Y. Wu, K. He and R. Girshick, "PointRend: Image segmentation as rendering," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. doi ·arXiv
24. A. Kirillov, R. Girshick, K. He and P. Dollár, "Panoptic feature pyramid networks," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. doi ·arXiv
25. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. European Conference on Computer Vision (ECCV)*, 2020. doi ·arXiv
26. B. Cheng, I. Misra, A. G. Schwing, A. Kirillov and R. Girdhar, "Masked-attention mask transformer for universal image segmentation," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. doi ·arXiv
27. J. Jain, J. Li, M. Chiu, A. Hassani, N. Orlov and H. Shi, "OneFormer: One transformer to rule universal image segmentation," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. doi ·arXiv
28. T. Kerssies, N. Cavagnero, A. Hermans, N. Norouzi, G. Averta, B. Leibe, G. Dubbelman and D. de Geus, "Your ViT is secretly an image segmentation model," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. CVF ·arXiv
29. A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson et al., "Segment anything," in *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. doi ·arXiv
30. N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma et al., "SAM 2: Segment anything in images and videos," in *Proc. International Conference on Learning Representations (ICLR)*, 2025. OpenReview ·arXiv
31. N. Carion, L. Gustafson, Y.-T. Hu, S. Debnath, R. Hu, D. Suris et al., "SAM 3: Segment anything with concepts," in *Proc. International Conference on Learning Representations (ICLR)*, 2026. OpenReview ·arXiv
32. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal et al., "Learning transferable visual models from natural language supervision," in *Proc. International Conference on Machine Learning (ICML)*, PMLR 139, 2021. PMLR
33. B. Li, K. Q. Weinberger, S. Belongie, V. Koltun and R. Ranftl, "Language-driven semantic segmentation," in *Proc. International Conference on Learning Representations (ICLR)*, 2022. OpenReview ·arXiv
34. S. Cho, H. Shin, S. Hong, A. Arnab, P. H. Seo and S. Kim, "CAT-Seg: Cost aggregation for open-vocabulary semantic segmentation," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. doi ·arXiv
35. C. Zhu and L. Chen, "A survey on open-vocabulary detection and segmentation: Past, present, and future," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, 2024. doi
36. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, 2010. doi
37. B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso and A. Torralba, "Scene parsing through ADE20K dataset," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi ·CVF
38. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan et al., "Microsoft COCO: Common objects in context," in *Proc. European Conference on Computer Vision (ECCV)*, 2014. doi ·

arXiv
39. H. Caesar, J. Uijlings and V. Ferrari, “COCO-Stuff: Thing and stuff classes in context,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. doi ·arXiv
40. G. Neuhold, T. Ollmann, S. Rota Bulò and P. Kontschieder, “The Mapillary Vistas dataset for semantic understanding of street scenes,” in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017. doi
41. P. Krähenbühl and V. Koltun, “Efficient inference in fully connected CRFs with Gaussian edge potentials,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2011. NeurIPS · arXiv

DLCV 05・人體姿態與關鍵點估計

深度網路如何找出影像中每個人的關節：骨架與可見性、熱圖與迴歸及子像素解碼、由上而下與由下而上的多人流程、以 OKS 為基礎的 AP 與 PCK，以及透過提升、三角測量與人體模型取得的 3D 姿態、資料集與失效模式。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/

你將學到

- 以具名關鍵點、可見性標籤與骨架連線表示姿態。
- 比較座標迴歸與熱圖，包含次像素解碼。
- 選擇由上而下、由下而上或直接多人預測。
- 解讀以 OKS 為基礎的 AP、PCK ／ PCKh 與三維關節誤差。
- 理解三維提升、三角測量與身體模型擬合，再診斷常見錯誤。

先看全貌

偵測器會在每個人身上畫一個框（DLCV 03）。框只說明人在哪裡，卻沒說他在做什麼：站著、揮手、跌倒或踢球，框看起來都差不多。**姿態估計**（pose estimation）再往前一步。它找出身體上少數有意義的點，也就是**關鍵點**（鼻子、肩膀、手肘、手腕、臀部、膝蓋、腳踝等），再把它們連成火柴人。

> **用最簡單的話說**
>
> 還記得連連看的圖畫嗎？姿態估計玩的是相反的遊戲。電腦看著一張人物照片，必須把點補回去：肩膀、手肘、手腕、臀部、膝蓋、腳踝各放一個點，而且每個點的名字都要對。然後把點連起來，你就得到一個動作跟真人一樣的火柴人。難的地方，你自己做也會覺得難：手藏在背後、兩個朋友抱在一起而分不清哪隻手臂是誰的，還有要記得哪一邊才是他們的左邊。

為什麼重要。火柴人是身體動作的精簡摘要。運動分析、物理治療、跌倒偵測與健身 App 都從它量測關節角度。動畫、遊戲與擴增實境用它來驅動虛擬角色。機器人與自駕車用它預測行人接下來會往哪走。神經科學家不用貼標記，就能追蹤小鼠與果蠅的關節 [1]。同樣的想法，熱圖、關鍵點分組與關鍵點指標，也被拿去處理人臉、手、動物與物件的部位。

本篇在人物偵測、非極大值抑制（non-maximum suppression, NMS）與 AP 的做法上，建立在 DLCV 03 之上；在密集預測與上取樣上，建立在 DLCV 04 之上；這些內容我們直接連結，不再重複。隨時間追蹤姿態會用到 DLCV 07 的關聯方法，3D 三角測量則用到 DLCV 11 的相機幾何。兩篇綜述勾勒出這個領域：Zheng 等人 [2] 涵蓋以深度學習做的 2D 與 3D 姿態，Chen 等人 [3] 則依任務整理單目方法。我們會在「現代觀點」回頭談這兩篇。

什麼是姿態

白話版。姿態由具名身體點、座標與可見性標籤組成，再以預定義骨架連接。

精確版。**骨架**（skeleton）是一張圖 $G = (V, E)$。頂點 $V = \{1, \dots, K\}$ 是 K 種關鍵點類型（例如「左手腕」），邊 $E \subset V \times V$ 是連接它們的**肢段**（limb）。一個人的 2D 姿態是

$$P = \{(\mathbf{x}_k, v_k)\}_{k=1}^{K}, \qquad \mathbf{x}_k = (x_k, y_k) \in \mathbb{R}^2,$$

其中 $\mathbf{x}_k$ 是關鍵點 k 在影像中的位置（單位為像素），v_k 是它的**可見性旗標**（visibility flag）。COCO 關鍵點格式用 $v_k = 0$ 表示「未標註」，$v_k = 1$ 表示「已標註但看不見」（被遮擋，位置由標註者推測），$v_k = 2$ 表示「已標註且可見」[4]。3D 姿態則把 $\mathbf{x}_k$ 換成點 $\mathbf{X}_k \in \mathbb{R}^3$，通常以毫米為單位，位於相機座標系或世界座標系。

COCO 的人物骨架有 $K = 17$ 個關鍵點：鼻子、雙眼、雙耳、肩膀、手肘、手腕、臀部、膝蓋與腳踝 [4]。圖 1 在一個合成的火柴人上標出它們。有兩個慣例會讓每個初學者踩到：

- **左與右是指那個人的，不是觀看者的**。面對相機的人，他的左肩在影像的右側。因此把影像水平翻轉，會讓每個左側關鍵點變成右側關鍵點，資料增強時除了鏡射座標，也必須對調標籤。
- **看不見不等於未標註**。被遮擋的手腕（$v = 1$）仍有位置，也仍會被評估；未標註的關鍵點（$v = 0$）則直接被忽略。

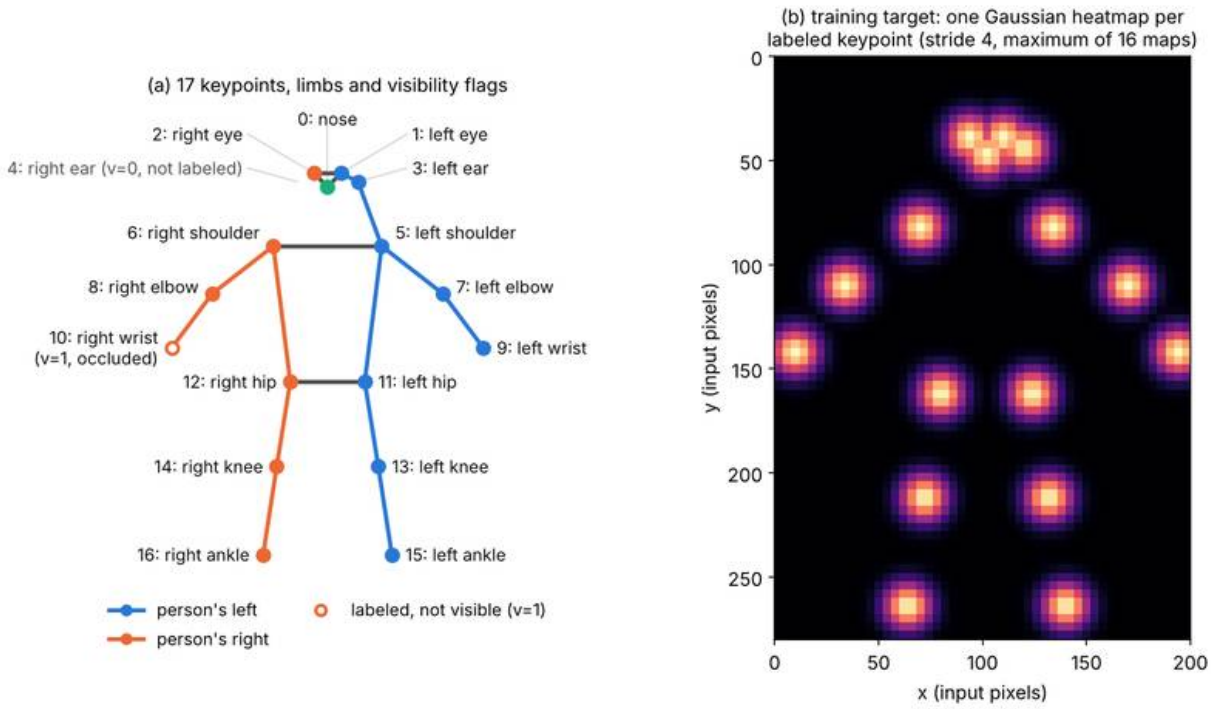


圖 *1* — (*a*) 合成人物上的 *17* 個關鍵點、*COCO* 風格骨架，人物的左側為藍色、右側為橘色，並標出兩種不可見的情況。(*b*) 對應的熱圖網路訓練目標：在輸出步幅 *4* 下，每個已標註的關鍵點有一個高斯光斑（*16* 張圖以逐像素最大值合併顯示）。

姿態估計與相鄰任務的比較

有四個相關任務很容易混淆：

任務	輸出	在本課程的位置
通用關鍵點偵測	在另一張影像中容易再次找到的角點與斑點，附描述子；沒有語意名稱	IP 12
物件偵測	每個物件一個框、類別與分數	DLCV 03
實例分割	每個物件一張像素遮罩	DLCV 04
姿態估計	每個物件一組具名關鍵點，由固定骨架連接	本篇

IP 12 的關鍵點是由影像決定的：哪裡剛好有角點或斑點就在哪裡。這裡的關鍵點是由任務決定的：即使左手肘落在一片平滑、沒有任何角點的袖子上，也必須找出來。這就是為什麼語意關鍵點要從有標註的範例學習，而不是用公式偵測。

這個任務有好幾種變體。**單人 2D** 估計假設每個輸入裁切區域只有一個人。**多人 2D** 估計還得判斷有幾個人，以及每個關鍵點屬於誰。**3D** 估計要還原深度。**全身**（whole-body）估計加上腳、手與臉部的特徵點，**網格重建**（mesh recovery）則輸出完整的人體表面，而不是稀疏的骨架 [5]。

輸出關鍵點的兩種方式

白話版。網路可以直接說出數字（「手腕在 $x = 140$、$y = 128$」），這是迴歸；也可以畫一張圖，在手腕所在之處發亮，這是熱圖。在深度學習的大部分時期，畫圖贏了，因為卷積網路畫圖比報數字拿手。

直接迴歸

DeepPose [6] 是早期以迴歸為基礎的深度姿態估計器。卷積網路把影像對應到 $2K$ 個數字的向量，也就是以人物框正規化後的關鍵點座標，並用平方誤差訓練，

$$\mathcal{L}_{\text{reg}} = \sum_{k=1}^{K} \|\hat{\mathbf{x}}_k - \mathbf{x}_k\|_2^2,$$

其中 $\hat{\mathbf{x}}_k$ 是關鍵點 k 的預測位置，$\mathbf{x}_k$ 是真值位置（皆為正規化座標）。DeepPose 以一串後續的迴歸器修正它的初步猜測，這些迴歸器作用在每個關節周圍解析度更高的裁切區域上。

迴歸簡單、快速，也沒有解析度上限；但全連接的輸出丟掉了卷積擅長處理的空間配置，因此多年來迴歸在準確度上落後熱圖。

高斯熱圖

Tompson 等人 [7] 改為要求網路針對每種關鍵點類型輸出一張影像，也就是一張**熱圖**，在關鍵點附近高、其他地方低。關鍵點 k 的目標是以真實位置為中心的高斯光斑：

$$H_k^*(\mathbf{p}) = \exp\!\left(-\frac{\|\mathbf{p} - \mathbf{x}_k/s\|_2^2}{2\sigma^2}\right),$$

其中 $\mathbf{p}$ 是熱圖的一個格子（整數座標），s 是**輸出步幅**（output stride，一個熱圖格子涵蓋幾個輸入像素，通常為 4），$\mathbf{x}_k/s$ 是以熱圖單位表示的真實關鍵點位置，σ 是光斑的寬度（以格子為單位）。網路用預測熱圖 $\hat{H}_k$ 與目標之間逐像素的均方誤差來訓練，

$$\mathcal{L}_{\text{hm}} = \frac{1}{K\,|\Omega|}\sum_{k=1}^{K}\sum_{\mathbf{p}\in\Omega} w_k\big(\hat{H}_k(\mathbf{p}) - H_k^*(\mathbf{p})\big)^2,$$

其中 Ω 是熱圖格子的集合，w_k 對未標註的關鍵點為 0，其餘為 1。堆疊沙漏網路（stacked hourglass network）沿用 Tompson 等人的作法使用這種監督，取 $\sigma = 1$ 像素 [8]。

為什麼這比迴歸好這麼多？熱圖**符合卷積的運作方式**：它是密集的，而且具平移等變性（人移動，光斑就跟著移動），所以每個輸出格子回答的都是局部問題：「這裡有手腕嗎？」它**能表達疑慮**：兩個合理的手腕位置會變成兩個較弱的光斑，而單一迴歸座標會落在兩者之間。而且峰值高度就是每個關鍵點**免費的信心分數**。

代價是解析度：在步幅 4 下，峰值只能落在間距 4 像素的格點上，而且 K 張圖所需的記憶體遠多於 $2K$ 個數字。密集預測網路如何恢復解析度，見 DLCV 04；下一節說明格點造成多少損失，以及如何把大部分損失補回來。

解碼熱圖：argmax、量化誤差與子像素修正

白話版。熱圖最亮格子提供粗略位置，相鄰反應則協助細化到格點之間。

最簡單的解碼器取最亮的格子，再對應回影像：

$$\hat{\mathbf{x}}_k = s \cdot \arg\max_{\mathbf{p}\in\Omega} \hat{H}_k(\mathbf{p}).$$

如果真實位置在格子內均勻隨機分布，這個解碼器在每個軸上的誤差會均勻分布在 $[-s/2, s/2]$，預期的歐幾里得誤差約為 $0.383\,s$（單位正方形中心到其中一個隨機點的平均距離，$(\sqrt{2} + \ln(1+\sqrt{2}))/6 \approx 0.383$，再乘以 s）。在步幅 4 下，那就是約 1.5 像素的誤差，而且是在網路還沒犯任何錯之前。對一個 100 像素高的人來說，這已足以在稍後介紹的 OKS 指標的嚴格門檻上丟分。

常見的改進有三種：

- **四分之一偏移**。把估計值從峰值朝較亮的鄰格方向移動四分之一個格子。堆疊沙漏論文加入這一步來改善高精度門檻下的結果 [8]，SimpleBaseline 也把它當作標準作法 [9]。
- **對數熱圖上的 Taylor 步驟**。高斯的對數是二次函數（拋物線），所以可以由最亮格子上的值、斜率與曲率，精確求出它的最大值位置：$\hat{\mathbf{p}} = \mathbf{m} - \mathbf{D}''(\mathbf{m})^{-1}\mathbf{D}'(\mathbf{m})$，其中 $\mathbf{m}$ 是最亮的格子，$\mathbf{D}'$ 與 $\mathbf{D}''$ 是 $\log \hat{H}_k$ 在該處的梯度與 Hessian，以有限差分估計。這是 DARK [10] 的解碼步驟。DARK 在解碼前還會先對預測熱圖做高斯平滑，因為真實的預測並非完美的高斯；它也把訓練目標以精確（未四捨五入）的關鍵點位置為中心，這正是我們 H_k^* 公式已經採用的作法。
- **Soft-argmax（積分迴歸）**。把熱圖當成機率分布，輸出它的平均值：$\hat{\mathbf{x}}_k = s\sum_{\mathbf{p}} \mathbf{p}\,\tilde{H}_k(\mathbf{p})$，其中 $\tilde{H}_k = \text{softmax}(\hat{H}_k)$ 是對所有格子取的 softmax。Sun 等人 [11] 證明這能把熱圖變成可微分的迴歸輸出，所以網路可以直接針對座標做端到端訓練，沒有量化問題。

DARK 的消融實驗讓這個效果的大小很具體：對 HRNet-W32 模型、小尺寸 128 × 96 輸入、在 COCO 驗證集上，純 argmax 的 AP 為 61.2，四分之一偏移為 66.9，它的 Taylor 解碼為 68.4 [10]。單靠解碼器就讓 AP 動了 7 點以上。

僅線上版：以 *Taylor* 步驟做子像素解碼 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/notes/以-taylor-步驟做子像素解碼/*

這段程式碼把 500 個隨機關鍵點編碼成步幅 1、2、4、8 的熱圖，加入少量雜訊來模擬不完美的預測，再用各種方法解碼。所有數字都來自這次合成實驗。

```
import numpy as np

def encode(xy, shape, stride, sigma=2.0):
    """Gaussian heatmap of size shape=(H, W) cells for a keypoint at image position xy."""
    cx, cy = xy[0] / stride, xy[1] / stride            # exact (unrounded) center in cells
    ys, xs = np.mgrid[0:shape[0], 0:shape[1]]
    return np.exp(-((xs - cx) ** 2 + (ys - cy) ** 2) / (2 * sigma ** 2))

def peak(h):
    y, x = np.unravel_index(np.argmax(h), h.shape)
    return x, y

def decode_argmax(h, stride):
    x, y = peak(h)
    return np.array([x, y], float) * stride

def decode_quarter(h, stride):
    """Move a quarter cell from the peak toward the higher neighbour on each axis."""
    x, y = peak(h)
    p = np.array([x, y], float)
    if 0 < x < h.shape[1] - 1: p[0] += 0.25 * np.sign(h[y, x + 1] - h[y, x - 1])
    if 0 < y < h.shape[0] - 1: p[1] += 0.25 * np.sign(h[y + 1, x] - h[y - 1, x])
    return p * stride

def decode_taylor(h, stride):
    """One Newton step on log h at the peak: offset = -Hessian^-1 * gradient."""
    x, y = peak(h)
    p = np.array([x, y], float)
    if 1 <= x < h.shape[1] - 1 and 1 <= y < h.shape[0] - 1:
        L = np.log(np.maximum(h[y - 1:y + 2, x - 1:x + 2], 1e-10))
        g = np.array([L[1, 2] - L[1, 0], L[2, 1] - L[0, 1]]) / 2
        hxx = L[1, 2] - 2 * L[1, 1] + L[1, 0]
        hyy = L[2, 1] - 2 * L[1, 1] + L[0, 1]
        hxy = (L[2, 2] - L[2, 0] - L[0, 2] + L[0, 0]) / 4
        Hs = np.array([[hxx, hxy], [hxy, hyy]])
        if np.linalg.det(Hs) > 0 and hxx < 0:          # a proper maximum
            p += -np.linalg.solve(Hs, g)
    return p * stride

def decode_soft(logits, stride):
    """Soft-argmax: the expected position under softmax(logits)."""
    w = np.exp(logits - logits.max()); w /= w.sum()
    ys, xs = np.mgrid[0:w.shape[0], 0:w.shape[1]]
    return np.array([(w * xs).sum(), (w * ys).sum()]) * stride
```

```
rng = np.random.default_rng(0)
W, H = 192, 256                                    # input crop size in pixels (width,
 ↪  height)
pts = rng.uniform([16, 16], [W - 16, H - 16], (500, 2))

def mean_error(decoder, s, noise=0.01, use_log=False):
    shape, e = (H // s, W // s), 0.0
    for xy in pts:
        h = encode(xy, shape, s)
        h = np.clip(h + rng.normal(0, noise, h.shape), 1e-6, None)   # an imperfect
         ↪  "prediction"
        e += np.linalg.norm(decoder(np.log(h) if use_log else h, s) - xy)
    return e / len(pts)

print("stride  argmax  quarter  taylor   (mean error in input pixels)")
for s in (1, 2, 4, 8):
    print(f"{s:6d}  {mean_error(decode_argmax, s):6.2f}  {mean_error(decode_quarter,
     ↪  s):7.2f}"
          f"  {mean_error(decode_taylor, s):6.2f}")
print(f"soft-argmax, stride 4: clean map {mean_error(decode_soft, 4, 0.0, True):.2f},"
      f" noisy map {mean_error(decode_soft, 4, 0.01, True):.2f}")
```

```
stride  argmax  quarter  taylor   (mean error in input pixels)
     1    0.39     0.20    0.05
     2    0.80     0.40    0.11
     4    1.54     0.78    0.21
     8    3.20     1.59    0.43
soft-argmax, stride 4: clean map 0.04, noisy map 24.42
```

可以看出四個重點，圖 2 畫出前三個。

1. **argmax 誤差隨步幅線性成長**，並緊貼 $0.383\,s$ 的預測（步幅 1 到 8 分別是 0.38、0.77、1.53、3.06）。
2. **四分之一偏移讓誤差減半**，而且不花任何成本。
3. **Taylor 步驟去除了剩下的大部分誤差**，因為高斯的對數確實是拋物線；剩下的來自加入的雜訊。
4. **Soft-argmax 在乾淨的熱圖上是精確的，但在有雜訊的熱圖上嚴重失敗**。當 logits 等於 $\log \hat{H}$ 時，softmax 權重正比於 $\hat{H}$，而散布在數千個格子上的微小雜訊底，總量會超過狹窄的峰值，把平均值拉向圖的中心。積分迴歸是讓網路透過 soft-argmax 一起訓練，迫使網路在關鍵點以外幾乎不放任何質量；若把 soft-argmax 事後套用在以 MSE 訓練的熱圖上，它就是個很差的解碼器。

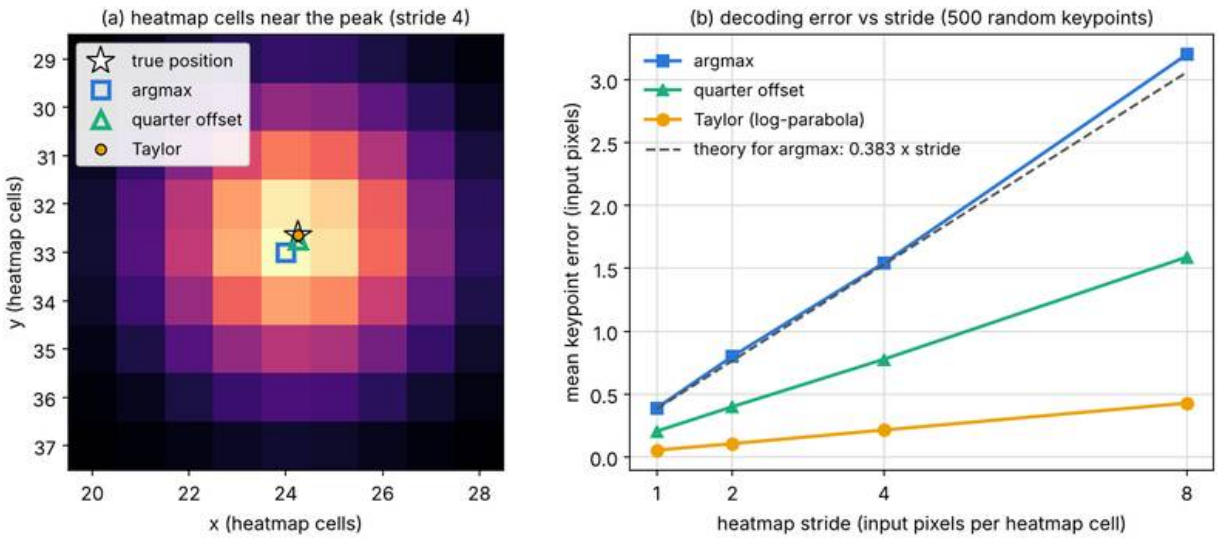


圖 2 — 解碼熱圖。(*a*) 步幅 *4* 下的峰值附近，最亮的格子（方塊）沒有對準真實位置（星號）；四分之一偏移（三角形）移動了一部分；*Taylor* 步驟（圓點）剛好落在上面。(*b*) 上述程式碼合成實驗中，各解碼器對 *500* 個隨機關鍵點的平均誤差；虛線是 *argmax* 的預測值，即步幅的 *0.383* 倍。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

其他輸出表示法

有兩種較新的表示法介於熱圖與單純迴歸之間。**座標分類**（SimCC [12]）對每個關鍵點預測兩個一維分布，分別針對 x 與 y，每個像素分成多個 bin，不需大型 2D 圖就能降低量化誤差。**以似然為基礎的迴歸**（RLE [13]）維持直接迴歸，但在訓練中學習誤差分布的形狀，其論文報告迴歸在多人姿態上勝過熱圖方法。兩者都是本篇稍後單階段與基於 query 的模型的自然輸出形式。

僅線上版：座標分類與以似然為基礎的迴歸 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/notes/座標分類與以似然為基礎的迴歸/*

單人熱圖網路

白話版。精確定位需要高解析度細節，辨認正確關節還需要身體上下文。

輸入是**人物裁切區域**（person crop）：圍著一個人的框，略微放大、補邊成固定的長寬比，再縮放成固定尺寸，例如 256 × 192（高 × 寬）。輸出是步幅 4 的 K 張熱圖，這個例子裡是 64 × 48。

- **Convolutional Pose Machines（CPM）** [14] 堆疊數個階段。每個階段看到影像特徵以及上一階段的熱圖（信念圖），所以後面的階段可以用其他關節的位置當脈絡，例如「手肘位於肩膀與手腕之間」。每個階段都有監督（**中間監督**，intermediate supervision），對抗深層堆疊中的梯度消失。

- **堆疊沙漏網路**（stacked hourglass）[8] 把對稱的編碼器–解碼器（「沙漏」）模組重複多次，每次之後都有中間監督。每個沙漏先池化到非常粗的解析度以蒐集全身脈絡，再用跳接上取樣回來，這和 DLCV 04 為分割描述的編碼器–解碼器是同一個想法。
- **SimpleBaseline** [9] 證明這些機制大多沒有必要。它採用 ImageNet 預訓練的 ResNet（DLCV 02），在最後一個階段之上加三層反卷積（轉置卷積）層，對 256×192 的輸入產生 64×48 的熱圖。它的消融實驗發現，三層反卷積（熱圖 64×48）比兩層（32×24）高 2.5 AP，直接量到了上面討論的解析度代價。
- **HRNet** [15] 讓高解析度分支貫穿整個網路，而不是到最後才恢復解析度。它逐步加入較低解析度的分支，讓它們**平行**執行，並反覆在所有解析度之間交換資訊（多尺度融合）。熱圖從高解析度分支讀出，而這個分支已經多次接收過粗尺度的脈絡。本篇後面有好幾個方法建立在 HRNet 之上：DARK 用 HRNet 模型回報它的解碼結果，HigherHRNet 與 DEKR 則把它擴展到由下而上的姿態。
- **ViTPose** [16] 把卷積骨幹換成不分層級的單純視覺 Transformer（ML 05），使用 16×16 的 patch，後面接由幾層反卷積（甚至只有一層上取樣加一層卷積）組成的輕量解碼器。論文把它從 1 億擴大到 10 億參數，顯示簡單的設計加上大型預訓練 Transformer，創造了新的準確度–吞吐量取捨。

僅線上版：*Simple Baselines（Xiao 等人，2018）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/papers/simple-baselinesxiao-等人 2018/*

以上每個模型都有兩個重要的訓練細節。**資料增強**會對裁切區域做旋轉、縮放與翻轉；水平翻轉時必須對調左右關鍵點的通道。**翻轉測試**（flip testing）讓模型分別處理裁切區域與它的鏡像，把鏡像輸出的通道換回來，再把熱圖平均；堆疊沙漏論文報告這讓驗證集平均提升約 1% [8]，後來的模型也把它當作標準作法 [9]。

僅線上版：翻轉測試與左右對調的記帳 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/notes/翻轉測試與左右對調的記帳/*

多人：由上而下與由下而上

白話版。照片裡有好幾個人時，有兩種工作順序。**由上而下**：先找出每個人，再在每個人的框裡找關節。**由下而上**：先找出圖中所有的關節，再判斷哪些關節屬於同一個人。第三類方法則兩件事一次做完。

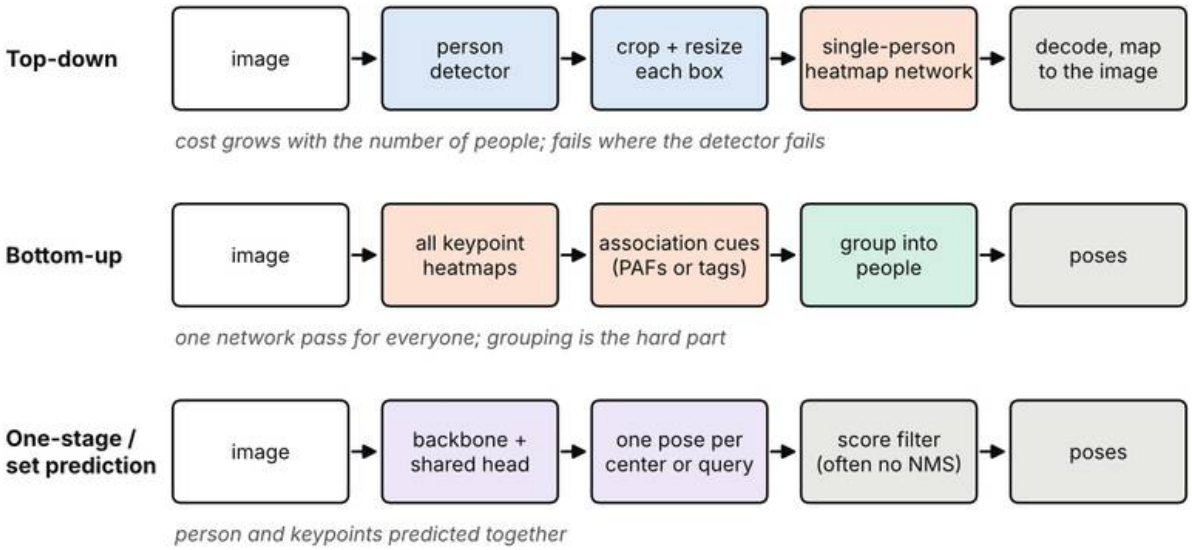


圖 3 — 三種多人流程。由上而下對每個偵測到的框執行單人模型；由下而上一次找出所有關鍵點再分組；單階段與集合預測模型一次輸出每個人的完整姿態。

由上而下：先偵測，再估計

人物偵測器（DLCV 03）提出框；每個框被裁切、縮放，再送進 SimpleBaseline、HRNet 或 ViTPose 這類單人網路；熱圖經過解碼，關鍵點再對應回影像座標。Mask R-CNN [17] 把同樣的想法建進偵測器：它的關鍵點分支在每個感興趣區域內預測 K 張大小為 $m \times m$ 的遮罩，每張都以 m^2 類 softmax 訓練，目標是關鍵點所在的單一「one-hot」像素（RoI 的機制見 DLCV 03）。

由上而下的方法很準確，因為每個人在關鍵點網路看到之前，都被縮放到同一個舒適的尺度，這對小尺寸與中等尺寸的人幫助最大 [18]。它的弱點來自流程本身：

- **成本隨人數成長**，因為關鍵點網路每個框都要跑一次。
- **偵測器失敗的地方它也會失敗**。漏掉的人就沒有姿態；NMS（DLCV 03）可能刪掉兩個重疊者其中一個；而一個框裡若有兩個人，單人網路會有兩個左手腕可選。OpenPose 的作者在一個 COCO 驗證子集上量測過：同一個單人模型用真值框得到 62.7 AP，用 SSD 偵測器的框只有 52.7 AP [18]。

由下而上：先偵測所有關鍵點，再分組

由下而上的模型對整張影像只跑一次，輸出 K 張包含所有人關鍵點的熱圖。峰值用局部最大值檢驗找出，這相當於每種關鍵點各做一次 NMS。困難的部分是**分組**（grouping）：決定哪個左手肘該跟哪個左手腕配對。

部分親和場（PAF）。OpenPose [18] 除了熱圖之外，還對每種肢段預測一個 2D 向量場。在第 k 個人的肢段 c 上的像素，目標是指向該肢段方向的單位向量；其他地方為零：

$$\mathbf{L}^{*}_{c,k}(\mathbf{p}) = \begin{cases} \mathbf{v} = \dfrac{\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}}{\|\mathbf{x}_{j_2,k} - \mathbf{x}_{j_1,k}\|_2} & \text{if } \mathbf{p} \text{ lies on limb } c \text{ of person } k, \\ \mathbf{0} & \text{otherwise,} \end{cases}$$

其中 j_1 與 j_2 是肢段 c 連接的兩種關鍵點類型（例如左手肘與左手腕），$\mathbf{x}_{j,k}$ 是第 k 個人的關鍵點 j 的位置，而「在肢段上」的意思是離它們之間的線段不超過 σ_l（肢段寬度，單位為像素）。幾個人的肢段重疊處，目標取平均。要為兩個偵測到的關鍵點 $\mathbf{d}_{j_1}$ 與 $\mathbf{d}_{j_2}$ 之間的候選連接打分，OpenPose 沿著連接兩者的線段對預測場 $\mathbf{L}_c$ 做積分：

$$E = \int_0^1 \mathbf{L}_c(\mathbf{p}(u)) \cdot \frac{\mathbf{d}_{j_2} - \mathbf{d}_{j_1}}{\|\mathbf{d}_{j_2} - \mathbf{d}_{j_1}\|_2}\, du, \qquad \mathbf{p}(u) = (1-u)\,\mathbf{d}_{j_1} + u\,\mathbf{d}_{j_2},$$

其中 u 沿著線段變化，積分用幾個等間距的 u 取樣值來近似。當線段上的場和線段方向一致時，E 就高，而這只有在兩個關鍵點真的是同一個肢段的兩端時才會發生。同時為所有肢段找最佳指派是 NP-hard，所以 OpenPose 做了鬆弛：只保留樹狀骨架的邊，對每種肢段各解一次**二分圖匹配**（bipartite matching），例如用匈牙利演算法（見 DLCV 07），再把共用關鍵點的肢段接成人。

僅線上版：*OpenPose*：部分親和場（*Cao* 等人，*2017*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/papers/openpose 部分親和場 cao-等人 2017/*

這段程式碼為兩個並肩站著、左前臂交叉的人建立真值 PAF，在關鍵點位置加入少量雜訊，再逐個肢段分組，一次用 PAF 分數，一次只用單純的距離（總肢段長度最短）。座標是合成的。

```
import numpy as np
from scipy.optimize import linear_sum_assignment

# 13 body keypoints and the 12 limbs of a tree that connects them
J = ["nose", "l_sho", "r_sho", "l_elb", "r_elb", "l_wri", "r_wri",
     "l_hip", "r_hip", "l_knee", "r_knee", "l_ank", "r_ank"]
LIMBS = [(0, 1), (0, 2), (1, 3), (3, 5), (2, 4), (4, 6),
         (1, 7), (2, 8), (7, 9), (9, 11), (8, 10), (10, 12)]

# two people side by side; their left forearms cross (synthetic coordinates)
A = np.array([[40, 20], [52, 38], [28, 38], [70, 50], [24, 62], [100, 74], [22, 86],
              [48, 80], [32, 80], [50, 110], [30, 110], [52, 140], [28, 140]], float)
B = np.array([[72, 24], [84, 42], [62, 42], [98, 58], [56, 64], [74, 76], [52, 86],
              [80, 84], [66, 84], [82, 114], [64, 114], [84, 144], [62, 144]], float)
people = [A, B]
H, W, width = 160, 120, 3.0

def render_paf(c):
    """Ground-truth part affinity field of limb c: unit vectors near each person's limb."""
    ys, xs = np.mgrid[0:H, 0:W]
    field, count = np.zeros((H, W, 2)), np.zeros((H, W))
    for P in people:
        a, b = P[LIMBS[c][0]], P[LIMBS[c][1]]
        v = (b - a) / np.linalg.norm(b - a)
        along = (xs - a[0]) * v[0] + (ys - a[1]) * v[1]
        across = np.abs(-(xs - a[0]) * v[1] + (ys - a[1]) * v[0])
```

```
        on = (along >= 0) & (along <= np.linalg.norm(b - a)) & (across <= width)
        field[on] += v; count[on] += 1
    return field / np.maximum(count, 1)[..., None]          # average where limbs overlap

def limb_score(field, p, q, n=10):
    """Line integral of the field along the segment p -> q, approximated by n samples."""
    d = (q - p) / np.linalg.norm(q - p)
    u = np.linspace(0, 1, n)[:, None]
    pts = np.round(p + u * (q - p)).astype(int)
    return np.mean(field[pts[:, 1], pts[:, 0]] @ d)

rng = np.random.default_rng(0)
cands = [np.stack([A[j], B[j]]) + rng.normal(0, 1.0, (2, 2)) for j in range(13)]  #
 ↪  detections

ok_paf = ok_near = 0
for c, (j1, j2) in enumerate(LIMBS):
    field = render_paf(c)
    S = np.array([[limb_score(field, p, q) for q in cands[j2]] for p in cands[j1]])
    D = np.linalg.norm(cands[j1][:, None] - cands[j2][None], axis=2)
    if (j1, j2) == (3, 5):                                  # the crossing forearms
        print("left elbow -> left wrist, PAF scores (rows: elbows A, B; cols: wrists A, B)")
        print(np.round(S, 2))
        print("distances in pixels"); print(np.round(D, 1))
    r, k = linear_sum_assignment(-S)                        # maximize total affinity
    ok_paf += np.sum(k == r)                                # candidate i belongs to person
     ↪  i
    r, k = linear_sum_assignment(D)                         # baseline: shortest total
     ↪  length
    ok_near += np.sum(k == r)
print(f"correct limbs with PAF scores:     {ok_paf}/{2 * len(LIMBS)}")
print(f"correct limbs with distance only:  {ok_near}/{2 * len(LIMBS)}")
```

```
left elbow -> left wrist, PAF scores (rows: elbows A, B; cols: wrists A, B)
[[0.67 0.04]
 [0.29 0.78]]
distances in pixels
[[41.1 27.2]
 [18.4 30.2]]
correct limbs with PAF scores:     24/24
correct limbs with distance only:  22/24
```

印出的矩陣顯示只靠幾何會失敗。每個手肘都更靠近另一個人的手腕（27.2 與 18.4 像素），而不是自己的（41.1 與 30.2），所以依距離配對會把兩條前臂都接錯。PAF 分數只在真正的肢段上高（0.67 與 0.78），因為那裡的場指向線段的方向；在錯誤的配對上則很低。圖 4 畫出兩種結果。請注意這裡的 PAF 是真值；訓練好的網路預測的場會比較模糊，方法的準確度取決於它預測得多好。

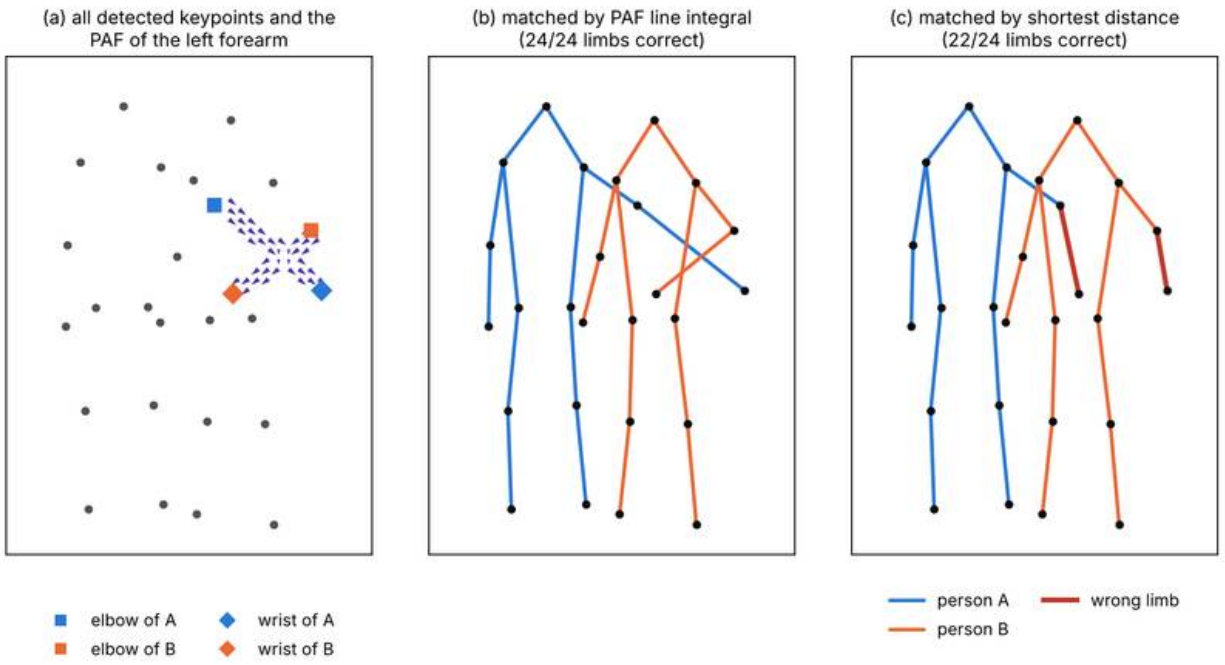


圖 4 — 在上述程式碼的合成場景上做由下而上分組。(*a*) 偵測到的關鍵點與左前臂的 *PAF*。(*b*) 以 *PAF* 線積分配對，每個肢段都還原正確。(*c*) 只用距離配對，會把交叉的兩條前臂對調（紅色）。

關聯嵌入（associative embedding）。 Newell 等人 [19] 完全不用肢段場。網路在每張關鍵點熱圖旁邊輸出一張 **tag 圖**：每個像素一個實數。訓練時讓同一個人的關鍵點 tag 彼此相等，不同人的 tag 彼此相異。令 $h_k(\mathbf{x})$ 為關鍵點類型 k 在像素 $\mathbf{x}$ 處的 tag，$\mathbf{x}_{nk}$ 為第 n 個人的關鍵點 k 的真值位置，共 N 個人，$\bar{h}_n = \frac{1}{K}\sum_k h_k(\mathbf{x}_{nk})$ 為第 n 個人的平均 tag，則分組損失為

$$\mathcal{L}_g = \frac{1}{N}\sum_{n}\sum_{k}\left(\bar{h}_n - h_k(\mathbf{x}_{nk})\right)^2 + \frac{1}{N^2}\sum_{n}\sum_{n'}\exp\!\Big(-\frac{1}{2\sigma^2}\left(\bar{h}_n - \bar{h}_{n'}\right)^2\Big),$$

其中第一項把每個人的 tag 拉向他們的平均，第二項把不同人的平均推開，懲罰會隨兩者距離增加而衰減（σ 決定多遠才算「夠遠」）。tag 的絕對值不重要，只有差值重要。測試時，關鍵點一個關節接一個關節地加入人物，配對 tag 接近且偵測信心高的那些。CornerNet 借用了完全相同的想法來配對框的角點（DLCV 03）。

尺度。由下而上的方法以影像本身的尺度看所有人，所以小尺寸的人只佔到很少的熱圖格子；OpenPose 在中等尺寸的人上不如由上而下的方法準確，原因就在這裡 [18]。HigherHRNet [20] 加入解析度更高的熱圖（HRNet 的輸出用轉置卷積上取樣），並對多個解析度做監督與聚合；其論文報告在 CrowdPose 測試集上勝過所有由上而下的方法，那裡的框並不可靠。DEKR [21] 回到**密集迴歸**，每個關鍵點有一個分支，其自適應卷積專注在該關鍵點的區域，並報告在 COCO 與 CrowdPose 上優於偵測加分組的方法。

單階段與集合預測姿態

第三類方法直接為每個人預測完整的姿態，不需要獨立的分組步驟。

- **以中心為基礎。** CenterNet（「Objects as Points」）[22] 在熱圖上把每個人偵測為一個中心點，並迴歸從中心到各關鍵點的 $2K$ 個偏移量；接著把每個迴歸出的關鍵點吸附到另一張關鍵點熱圖上最近的峰值，所以中心扮演了分組的線索。
- **即時密集頭。** RTMO [23] 把姿態頭放進 YOLO 風格的單階段偵測器，用一對一維熱圖（座標分類）表示每個關鍵點，目標是以單階段的速度達到由上而下的準確度。
- **集合預測。** DETR 風格的 Transformer 偵測器（DLCV 03）輸出固定數量的 query，並與真值一對一匹配，所以不需要 NMS 或分組。ED-Pose [24] 以兩個明確的框偵測階段把它套用到姿態上：人物解碼器偵測人，人到關鍵點解碼器則把每個關鍵點當成一個要偵測的小框。它用 L1 迴歸損失訓練、不用熱圖，並報告在 COCO 上，使用相同骨幹時優於以熱圖為基礎的由上而下方法。

選擇哪種流程

	由上而下	由下而上	單階段／集合預測
主要步驟	先偵測人，再逐一估計	先偵測所有關鍵點，再分組	直接預測每個人的姿態
成本與人數的關係	線性成長	網路成本幾乎固定；分組成本會成長	幾乎固定
小尺寸的人	好（每個裁切區域都會縮放）	較難（所有人共用一個影像尺度）	取決於特徵金字塔
擁擠與重疊	框會重疊；NMS 可能刪掉人	分組可能把人混在一起	query 或中心可能碰撞
典型用途	人少且分得開時要求高準確度	同時有很多人；即時系統	端到端訓練；即時

衡量姿態準確度

白話版。預測的關鍵點只要離真實位置近，就算「好」；但「近」必須取決於人的大小，也取決於是哪一種關鍵點：在很小的人的鼻子上偏 5 像素是大錯；在很大的人的臀部上偏 5 像素就不是。

PCK 與 PCKh

正確關鍵點百分比（percentage of correct keypoints, PCK）把預測關鍵點落在真值的門檻距離之內視為正確，門檻是人物大小的某個比例。Yang 與 Ramanan 使用人物邊界框大小的某個比例 [25]。MPII 的 **PCKh** [26] 則改用**頭部線段長度**的 50%，因為人彎腰或伸展時頭的大小不會改變。形式上，

$$\mathrm{PCKh@}\alpha = \frac{1}{\sum_{n,k} \delta(v_{nk} > 0)} \sum_{n,k} \delta(v_{nk} > 0)\; \mathbb{1}\big[\|\hat{\mathbf{x}}_{nk} - \mathbf{x}_{nk}\|_2 \le \alpha\, \ell_n\big],$$

其中 n 是人的索引，k 是關鍵點的索引，ℓ_n 是第 n 個人的頭部線段長度，α 是比例（標準 PCKh 取 $\alpha = 0.5$），$\delta(\cdot)$ 與 $\mathbb{1}[\cdot]$ 在條件成立時為 1，否則為 0，且只計入已標註的關鍵點。PCK 假設人已經被找到；它不會懲罰漏掉或憑空多出的人，這就是為什麼多人基準使用 AP 類型的指標。

物件關鍵點相似度與關鍵點 AP

COCO 的指標仿照框偵測（DLCV 03）。在框 AP 中，偵測與真值框的 IoU 超過門檻時就算配對成功。在關鍵點 AP 中，預測姿態與真值人物的**物件關鍵點相似度**（object keypoint similarity, OKS）超過門檻時才算配對成功 [4]：

$$\mathrm{OKS} = \frac{\sum_i \exp\!\left(-d_i^2/(2s^2\kappa_i^2)\right)\delta(v_i>0)}{\sum_i \delta(v_i>0)},$$

其中 i 遍歷所有關鍵點，d_i 是預測與真值關鍵點 i 之間的歐幾里得距離，v_i 是真值的可見性旗標，s 是物件尺度，定義為該人物分割面積的平方根，κ_i 是各關鍵點的常數。每一項都是誤差的未正規化高斯，標準差為 $s\kappa_i$，所以完美的關鍵點為 1，誤差變大時趨近 0；OKS 對已標註的關鍵點取平均。預測的可見性旗標不會被使用。

常數從哪裡來。 COCO 取 5,000 張驗證影像，由兩組人各標註一次，對每種關鍵點類型量測人類標註者相對於人物尺度的分歧程度，$\sigma_i^2 = E[d_i^2/s^2]$，再設 $\kappa_i = 2\sigma_i$ [4]。σ_i 的值為：鼻子 0.026、眼睛 0.025、耳朵 0.035、肩膀 0.079、手肘 0.072、手腕 0.062、臀部 0.107、膝蓋 0.087、腳踝 0.089 [4], [27]。眼睛容易精確定位；臀部藏在衣服下面就不容易，所以這個指標對臀部比較寬容。取 $\kappa_i = 2\sigma_i$ 時，一、二、三個標註者標準差的誤差，相似度分別是 $e^{-1/8} \approx 0.88$、$e^{-1/2} \approx 0.61$ 與 $e^{-9/8} \approx 0.32$，而人類標註者彼此之間的 OKS 中位數約為 0.91 [4]。

從 OKS 到 AP。其餘部分就是 DLCV 03 的 AP：依人物層級的分數排序所有預測姿態，把每個姿態配對到 OKS 最高且尚未被配對的真值人物，再在 OKS 門檻 $0.50, 0.55, \ldots, 0.95$ 下計算 AP。主要指標 AP 是這十個值的平均。附屬指標有 AP^{50}、AP^{75}、中等尺寸的人 AP^{M}（面積介於 32^2 與 96^2 之間）、大尺寸的人 AP^{L}（面積大於 96^2），以及對應的 AR 值。每張影像最多計分 20 個姿態，而小尺寸的人沒有關鍵點標註 [4]。

僅線上版：深入 *OKS*：未標註的人、擁擠區域與尺度 *s* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/notes/深入-oks 未標註的人擁擠區域與尺度-s/*

這段程式碼用 COCO 的常數，對一個約 200 像素高的合成人物計算 OKS。

```
import numpy as np

NAMES = ["nose", "l_eye", "r_eye", "l_ear", "r_ear", "l_shoulder", "r_shoulder",
         "l_elbow", "r_elbow", "l_wrist", "r_wrist", "l_hip", "r_hip",
         "l_knee", "r_knee", "l_ankle", "r_ankle"]
SIGMAS = np.array([.26, .25, .25, .35, .35, .79, .79, .72, .72, .62, .62,
                   1.07, 1.07, .87, .87, .89, .89]) / 10.0       # COCO per-keypoint
                    ↪  constants

def oks(pred, gt, vis, area):
    """pred, gt: (17, 2) pixel coordinates; vis: (17,) COCO flags; area: segment area
     ↪  (s^2)."""
    d2 = ((pred - gt) ** 2).sum(1)
    kappa = 2 * SIGMAS
    ks = np.exp(-d2 / (2 * area * kappa ** 2))                 # per-keypoint similarity
    labeled = vis > 0
    return ks[labeled].mean(), ks

# a standing person about 200 px tall, facing the camera (synthetic coordinates)
```

```
gt = np.array([[100, 30], [106, 25], [94, 25], [112, 28], [88, 28], [125, 60], [75, 60],
               [135, 95], [65, 95], [140, 128], [60, 128], [115, 130], [85, 130],
               [117, 175], [83, 175], [118, 220], [82, 220]], float)
vis = np.full(17, 2); vis[3] = 0                                    # left ear not labeled
area = 0.6 * (70 * 200)                                             # segment area: ~60% of its
 ↪  box

rng = np.random.default_rng(1)
jitter = gt + rng.normal(0, 3, gt.shape)                            # every keypoint off by ~3 px
 ↪  per axis
flipped = jitter.copy(); flipped[[9, 10]] = flipped[[10, 9]]    # left/right wrists swapped
print(f"perfect          OKS = {oks(gt, gt, vis, area)[0]:.3f}")
print(f"3 px jitter      OKS = {oks(jitter, gt, vis, area)[0]:.3f}")
print(f"+ wrists swapped OKS = {oks(flipped, gt, vis, area)[0]:.3f}")

# the same 5-pixel error costs very different amounts on different keypoints
for k in (0, 5, 11):
    p = gt.copy(); p[k, 0] += 5
    print(f"5 px error on {NAMES[k]:10s}: similarity {oks(p, gt, vis, area)[1][k]:.3f}")
```

```
perfect          OKS = 1.000
3 px jitter      OKS = 0.933
+ wrists swapped OKS = 0.810
5 px error on nose      : similarity 0.577
5 px error on l_shoulder: similarity 0.942
5 px error on l_hip     : similarity 0.968
```

這個合成範例有三個重點。到處有小抖動，代價很小（0.933）。單一的手腕**左右對調**會讓兩個手腕的相似度都變成幾乎為零，OKS 降到 0.810：這個姿態在門檻到 0.80 為止仍算正確，但在 0.85 以上就失敗，所以十個 AP 門檻中大約損失三分之一。而同樣 5 像素的誤差，在鼻子上要花掉 0.42 的相似度，在臀部上只花 0.03。圖 5 顯示每個關鍵點的容忍半徑與相似度曲線。

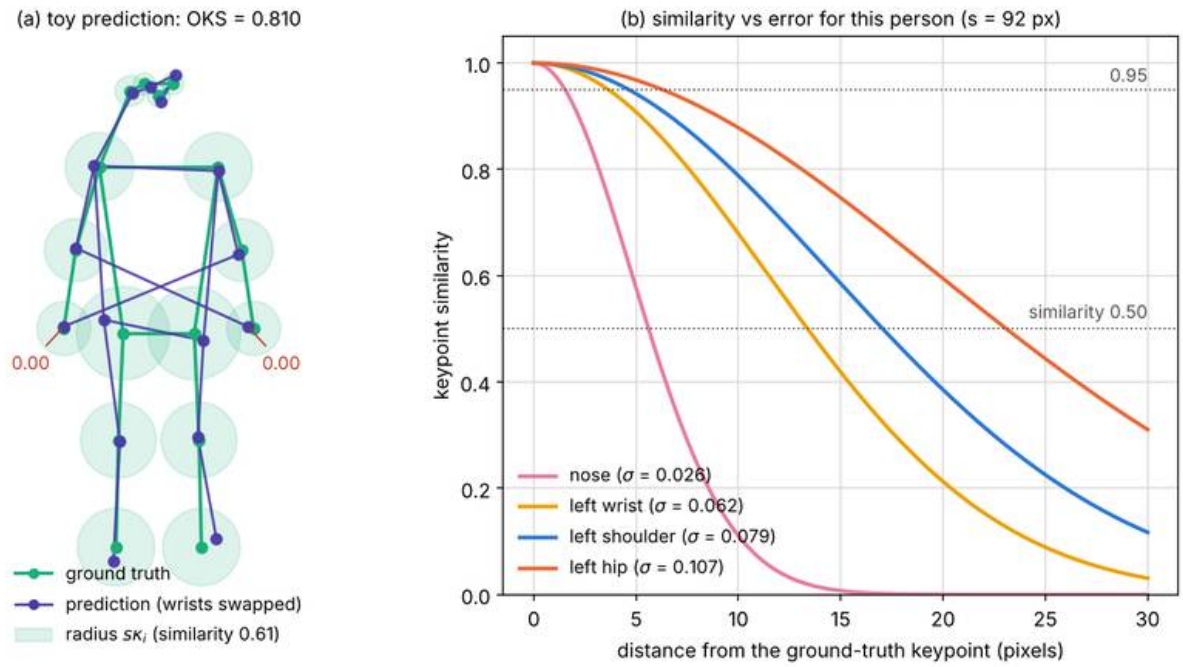


圖 *5* — 上述程式碼中合成人物的 *OKS*。(*a*) 半徑為 $s\kappa_i$ 的圓，關鍵點在圓上的相似度是 *0.61*；預測有約 *3* 像素的抖動，並且手腕對調。(*b*) 四種關鍵點類型的相似度隨誤差變化的函數，對應此人物的尺度 $s \approx 92$ 像素。

3D 姿態的指標

3D 姿態的標準指標是**平均每關節位置誤差**（mean per-joint position error, MPJPE），即預測與真值 3D 關節之間歐幾里得距離的平均，

$$\text{MPJPE} = \frac{1}{K}\sum_{k=1}^{K} \|\hat{\mathbf{X}}_k - \mathbf{X}_k\|_2,$$

以毫米為單位，並對所有測試影格取平均，其中 $\hat{\mathbf{X}}_k$ 與 $\mathbf{X}_k$ 是關節 k 的預測與真實 3D 位置。Human3.6M 上常見兩種協定：把預測與真值的根（中央臀部）關節對齊後計算誤差，稱為協定 #1；再額外把預測骨架剛性對齊到真值後計算誤差，稱為協定 #2 [28]；第二種指標也稱為 PA-MPJPE 或重建誤差 [2]。第二種會掩蓋全域旋轉與尺度的誤差，所以兩個數字絕對不能互相比較。

數字沒告訴你的事

和框 AP 一樣，關鍵點 AP 混合了好幾種誤差。Ronchi 與 Perona [29] 把關鍵點定位誤差分成四種：**抖動**（jitter，靠近正確的關鍵點，但超出人類誤差範圍）、**反轉**（inversion，落在這個人自己的左右對應點或另一個關鍵點上）、**對調**（swap，落在另一個人的關鍵點上）與**漏失**（miss，附近沒有任何關鍵點）。他們發現誤差最取決於可見關鍵點的數量，以及附近其他人造成的雜亂。連

COCO 經過兩次標註的資料也有 1–2% 的離群值，多半是左右混淆或標到錯的人 [4]。診斷模型時要依誤差類型，而不是只看 AP。

資料集與基準

白話版。少數幾個資料集定義了什麼叫「好」：COCO 代表日常照片中的多人，MPII 代表廣泛的活動，CrowdPose 代表嚴重重疊，Human3.6M 代表 3D，另外還有較新的資料集涵蓋全身、影片與動物。

資料集	內容（取自論文或官方頁面）	指標	為什麼重要
COCO keypoints [30], [4]	日常照片中人物的 17 個身體關鍵點；小尺寸的人未標註	OKS AP	主要的多人基準
MPII Human Pose [26]	40,522 個有標註的人，影像來自 YouTube 影片，以超過 800 種活動為查詢詞蒐集	PCKh	活動多樣；有單人與多人賽道
CrowdPose [31]	20,000 張影像、約 80,000 人、14 個關鍵點，挑選成擁擠程度均勻分布	OKS AP	擁擠場景
COCO-WholeBody [32]	每人 133 個特徵點：臉部 68、手部 42、身體與腳部 23	OKS AP	臉、手與腳一起處理
PoseTrack [33]	含人物軌跡與關節姿態的影片	AP、MOTA	影片中的姿態估計與追蹤
Human3.6M [34]	11 位演員、17 種情境的 360 萬個 3D 姿態，由 4 台校正過的相機搭配動作捕捉記錄	MPJPE	最廣泛使用的 3D 基準
AP-10K [35]	23 個科、54 種哺乳動物的 10,015 張影像，以 COCO 風格標註	AP	動物姿態與跨物種泛化

CrowdPose 值得細看，因為它對擁擠程度有精確的定義。對第 i 個人，令 N_a^i 為落在其框內的自己的關節數，N_b^i 為落在同一個框內的其他人的關節數；有 n 個人的影像的**擁擠指數**（Crowd Index）為 $\frac{1}{n}\sum_i N_b^i / N_a^i$ [31]。常見資料集以不擁擠的影像為主；CrowdPose 的挑選方式讓這個指數在 0 與 1 之間近乎均勻分布，其作者發現強模型在擁擠影像上的準確度比不擁擠的影像低約 20 AP [31]。

閱讀結果時有兩點要小心。Human3.6M 是只有少數演員的室內資料集 [2], [34]，所以在那裡 MPJPE 低，不代表在室外影片上也有好結果。而在 COCO 上，結果取決於由上而下方法所用的人物偵測器（見 Simple Baselines 導讀 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/papers/simple-baselinesxiao-等人 2018/)）、輸入解析度與額外的訓練資料，所以只有在相同設定下才能比較數字。

人體之外

白話版。熱圖網路裡沒有任何東西是專屬於人的。換掉關鍵點的清單和訓練資料，同一套機制就能找出臉的角落、手的關節，或狗的爪子。

- **全身**。 COCO-WholeBody [32] 在身體關鍵點之外加入腳、手與密集的臉部特徵點，這在同一個人身上造成極大的尺度範圍；它的基準模型 ZoomNet 圍繞身體的階層結構設計來處理這一點。OpenPose 的期刊版本 [36] 加入了腳部關鍵點。
- **動物**。 AP-10K [35] 對 54 種哺乳動物的姿態做基準測試，並檢驗對未見過的動物科別的泛化能力。在神經科學中，DeepLabCut [1] 以預訓練網路做遷移學習，在其實驗中用約 200 個標註影格，就能追蹤小鼠與果蠅使用者自訂的身體部位，取代反光標記。
- **物件**。同樣的表述適用於任何具有一致特徵點的類別，例如汽車的車輪與車燈，前提是關鍵點的定義一致，並且像處理人體那樣處理該類別的左右對稱。

從 2D 到 3D 姿態

白話版。照片把世界壓平了：從單張影像看，指向相機的手和放在臀部的手可能看起來幾乎一樣。要把深度找回來有三種方法：憑經驗猜（提升）、用兩台以上的相機量（三角測量），或擬合一個只允許合理體型的人體模型。

提升 2D 關鍵點

Martinez 等人 [28] 證明，單目 3D 姿態估計中令人意外大的一部分，只要 2D 關鍵點已知就解決了。他們的模型取偵測到的 2D 姿態的 $2K$ 個座標，用簡單的全連接網路（含批次正規化、dropout 與殘差連接的線性層）對應到 $3K$ 個座標。在 Human3.6M 上訓練後，給定真值 2D 關鍵點時，它大幅超越先前最佳的方法，這暗示剩餘的困難大部分在 2D 偵測，而不是提升。提升也是有歧義的：好幾個 3D 姿態會投影成同一個 2D 姿態，而只有單一輸出的網路只能傳回其中一個。

VideoPose3D [37] 用擴張時間卷積提升整段 2D 姿態序列，所以相鄰影格之間的運動有助於解決深度與抖動問題。它還提出**反投影**（back-projection），這是一種半監督損失：對沒有標註的影片估計 3D 姿態，並要求它們投影回偵測到的 2D 關鍵點。

多視角三角測量

有兩台以上校正過的相機時，深度就不再是猜測。每台相機的 2D 關鍵點在空間中定義一條射線，3D 關節就是這些射線（幾乎）相交的那一點。這個幾何就是 DLCV 11 的三角測量，逐個關節套用：相機矩陣就是該篇的針孔模型，線性（DLT）解則推導於它的補充資料。可學習的三角測量 [38] 讓這一步成為網路的一部分：它依影像預測的信心為每個視角加權，所以關節被遮擋的視角貢獻較少；它的第二種變體則把所有視角的 2D 特徵聚合成 3D 體積，再用 3D 卷積精修。

這段程式碼把一個合成的 13 關節骨架（站在兩台朝內對準的相機前方 4 公尺處）投影到兩張影像中，在 2D 關鍵點上加入 2 像素的雜訊，用 DLT 對每個關節做三角測量，並在三種相機間距（**基線**，baseline）下回報 MPJPE。數字來自這次合成實驗，沒有做根對齊。

```
import numpy as np

def camera(K, angle_deg, center):
    """3x4 projection P = K [R | t] for a camera at `center` rotated about the y axis."""
    a = np.deg2rad(angle_deg)
    R = np.array([[np.cos(a), 0, -np.sin(a)], [0, 1, 0], [np.sin(a), 0, np.cos(a)]])
    return K @ np.hstack([R, -R @ center[:, None]])

def project(P, X):
    x = (P @ np.hstack([X, np.ones((len(X), 1))]).T).T
    return x[:, :2] / x[:, 2:]

def triangulate(Ps, xs):
    """Linear (DLT) triangulation of one point seen in several views."""
    A = []
    for P, (u, v) in zip(Ps, xs):
        A += [u * P[2] - P[0], v * P[2] - P[1]]
    X = np.linalg.svd(np.array(A))[2][-1]
    return X[:3] / X[3]

# a 13-joint 3D skeleton in millimetres, standing 4 m in front of the cameras (synthetic)
skel = np.array([[0, -1650, 0], [170, -1400, 0], [-170, -1400, 0], [250, -1120, 50], [-250,
 ↪  -1120, 50],
                 [280, -860, 120], [-280, -860, 120], [100, -900, 0], [-100, -900, 0],
                 [110, -480, 30], [-110, -480, 30], [110, -60, 0], [-110, -60, 0]], float)
skel += [0, 900, 4000]
K = np.array([[1000, 0, 640], [0, 1000, 360], [0, 0, 1]], float)

rng = np.random.default_rng(0)
for baseline in (200, 1000, 3000):                          # distance between the cameras
 ↪  (mm)
    half = baseline / 2
    Ps = [camera(K, np.degrees(np.arctan2(half, 4000)), np.array([-half, 0, 0.])),
          camera(K, -np.degrees(np.arctan2(half, 4000)), np.array([half, 0, 0.]))]
    errs = []
    for trial in range(200):                                    # 2D detections with 2 px noise
        obs = [project(P, skel) + rng.normal(0, 2.0, (13, 2)) for P in Ps]
        est = np.array([triangulate(Ps, [o[j] for o in obs]) for j in range(13)])
        errs.append(np.linalg.norm(est - skel, axis=1).mean())   # MPJPE of this trial
    print(f"baseline {baseline:4d} mm: MPJPE = {np.mean(errs):6.1f} mm")
```

```
baseline  200 mm: MPJPE =  183.7 mm
baseline 1000 mm: MPJPE =   37.9 mm
baseline 3000 mm: MPJPE =   17.4 mm
```

相機相距 20 公分時，2 像素的影像誤差在 3D 中變成約 18 公分，幾乎全部在深度方向（同一次實驗的逐軸檢查顯示深度方向 182 毫米，橫向分別為 9 與 18 毫米）：射線幾乎平行，所以微小的位移會讓交點沿著射線移動很遠。較寬的基線讓射線以較大的角度相交，把誤差降低為十分之一。

參數化人體模型與網格重建

骨架完全沒有說明體型，也沒有任何機制阻止提升網路預測出每個影格長度都在變的前臂。**參數化人體模型**（parametric body model）用一小組參數描述人體，解決了這兩個問題。最常用的是 SMPL [39]。它從一個有 $N = 6890$ 個頂點的模板網格，以及有 $K = 23$ 個關節的骨架出發，產生擺好姿態的人體：

$$M(\boldsymbol{\beta}, \boldsymbol{\theta}) = W(T_P(\boldsymbol{\beta}, \boldsymbol{\theta}), J(\boldsymbol{\beta}), \boldsymbol{\theta}, \mathcal{W}), \qquad T_P(\boldsymbol{\beta}, \boldsymbol{\theta}) = \bar{\mathbf{T}} + B_S(\boldsymbol{\beta}) + B_P(\boldsymbol{\theta}),$$

其中 $\boldsymbol{\beta}$ 是**體型**參數（身體形狀變異主要模式的係數），$\boldsymbol{\theta}$ 是**姿態**參數（每個關節相對其父關節的軸角旋轉，加上全域朝向：$3 \times 23 + 3 = 72$ 個數字），$\bar{\mathbf{T}}$ 是模板網格，B_S 與 B_P 是頂點偏移量，分別建模個人差異與隨姿態變化的形變（例如肌肉隆起），$J(\boldsymbol{\beta})$ 是由上了體型的網格迴歸出的關節位置，$\mathcal{W}$ 是混合權重，W 是標準的混合蒙皮（blend-skinning）函數，讓每個頂點繞著關節旋轉。關鍵點是從網格上讀出的，所以擬合好的 SMPL 人體永遠有一致的骨長。

人體網格重建（human mesh recovery, HMR）[40] 直接從影像迴歸 SMPL 參數與相機。它的主要損失是模型關節對 2D 關鍵點標籤的**重投影誤差**，這使得可以在野外影像上訓練；另有一個用真實 3D 人體訓練的對抗式先驗，會排除不合理的參數。HMR 2.0 是 4DHumans [41] 的一部分，它讓網路完全以 Transformer 為基礎，並在影片中追蹤重建出的人物度過遮擋。

僅線上版：*SMPL 細說* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/notes/smpl-細說/*

失效模式與實務建議

白話版。姿態模型失敗的地方，也是人會猶豫的地方：被藏起來的關節、纏在一起的人、鏡像混淆、被畫面切掉的身體、訓練中很少見的姿勢，以及模糊。

- **遮擋**。被遮擋的關節（$v = 1$）有標註但看不到，所以網路必須從脈絡推斷；自遮擋讓雙腿特別容易發生左右混淆 [29]。
- **擁擠**。重疊的人會讓偵測器失靈、讓分組混淆，並造成**對調**誤差 [29]；擁擠指數升高時準確度就下降 [31]。
- **左右翻轉**。正面與背面看起來很像，所以模型可能把一條肢段或整個人鏡像（**反轉**）；翻轉增強時忘了對調通道，也會造成系統性的同類錯誤。
- **截斷**。畫面外的關節沒有標註或只能猜測，局部人物的裁切區域也很難對中。
- **罕見姿勢**。體操、舞蹈或躺著的人在訓練資料中很少見；HMR 2.0 的作者特別指出這是先前方法吃力的情況 [41]。
- **動態模糊、低解析度與領域偏移**。快速的肢體會被拖影，小尺寸的人每條肢段只有很少像素，而以成人照片訓練的模型對嬰兒或藝術作品的遷移效果不佳 [2]。

本篇帶出的實務建議是：使用無偏的編碼器與子像素解碼器；檢查翻轉時有對調左右通道；回報由上而下結果所用的偵測器與輸入尺寸；分開評估擁擠與不擁擠的影像；依類型拆解誤差；影片中則在時間上平滑或追蹤關鍵點。以偵測為基礎的追蹤（tracking-by-detection）與 DLCV 07 的關聯方法可以直接沿用：例如 SimpleBaseline 用光流把姿態傳播到下一個影格，並用以 OKS 為基礎的相似度來配對 [9]，PoseTrack 則提供了基準 [33]。追蹤個別表面點而不是具名關節，則是 DLCV 07 的點追蹤問題。

現代觀點

綜述怎麼說

Zheng 等人 [2] 回顧 2014 年以來超過 250 篇以深度學習做 2D 與 3D 姿態的論文，組織方式與本篇相同：迴歸對熱圖、由上而下對由下而上，3D 則分為提升、網格重建與多視角方法。他們的未來方向包括領域適應、人與場景的約束，以及時間平滑度的指標。

Chen 等人 [3] 把單目方法整理成四種任務（2D 或 3D、單人或多人），列出被證明有效的設計技巧（多階段結構、中間監督、多尺度融合、身體結構約束），並指出遮擋與擁擠是主要尚未解決的挑戰。

Tian 等人 [5] 綜述以人體模型做的單目 3D 網格重建，對比以最佳化為基礎的擬合與迴歸網路，並指出缺乏野外 3D 標籤是核心困難。

僅線上版：「現代觀點」綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/pose-keypoints/papers/現代觀點綜述導讀/*

2023–2026 年的最先進技術

- **放大規模的由上而下 Transformer 姿態**。 ViTPose [16] 證明單純的視覺 Transformer 搭配簡單的解碼器可以良好地擴大規模（論文中從 1 億到 10 億參數），其最大的模型在發表時刷新了 COCO 關鍵點基準的最佳紀錄。
- **以人為中心的基礎模型**。 Sapiens [42] 以自監督方式，在超過 3 億張人類影像上預訓練 3 億到 20 億參數的視覺 Transformer，再微調到 2D 姿態、身體部位分割、深度與表面法向量；其作者報告，在相同的運算預算下，這種針對人的預訓練改善了全部四項任務。
- **即時系統**。 RTMPose [43]（由上而下、座標分類，針對 CPU、GPU 與行動裝置部署而設計）與 RTMO [23]（單階段）瞄準的是應用的延遲預算，而不是排行榜的巔峰。
- **端到端多人模型**。 ED-Pose [24] 這類基於 query 的模型去除了 NMS、分組與熱圖；ED-Pose 報告在相同骨幹下，COCO 準確度高於以熱圖為基礎的由上而下方法。
- **影片中的 3D 人體**。 HMR 2.0 [41] 這類 Transformer 網格重建，能從單目影片重建並追蹤完整的 3D 人體，讓這個領域從稀疏的骨架邁向完整的人體表面。

尚待解決的問題

1. **擁擠與嚴重遮擋**仍是最大的準確度落差 [31], [3]。
2. **野外 3D**。實驗室以外很難取得準確的 3D 標籤 [5]，而最廣泛使用的 3D 基準 Human3.6M 是室內資料集 [2]，所以單目 3D 方法的訓練與測試環境，離實際使用的環境很遠。
3. **時間一致性**。 MPJPE 這類逐影格的指標無法衡量抖動，也無法衡量隨時間的物理合理性 [2]。
4. **跨領域與跨物種的泛化**：嬰兒、藝術作品、不尋常的視角，以及未見過的動物科別 [2], [35]。
5. **評估**。 OKS 的常數是依 COCO 人物上標註者的分歧程度量測的 [4]，不會自動適用於其他關鍵點集合；而 AP 會掩蓋反轉與對調這類誤差類型 [29]。

重點整理

- 關鍵點名稱、左右規約與可見性標籤須一致。
- 熱圖解析度與解碼影響定位，只取最亮格子會遺失次像素資訊。
- 精確關節定位需要細緻空間特徵，以及足以辨識部位的上下文。
- 由上而下方法承接偵測錯誤與逐人成本，由下而上方法則需要分組。
- 註明各指標的尺度正規化、可見性規則與座標規約。
- 三維提升有多解，三角測量需要校準，身體模型則加入形狀假設；應依類型檢查錯誤。

練習

1. **手算量化**。一個手腕在裁切區域中位於 $x = 101.3$ 像素，熱圖步幅為 4。峰值在哪一欄熱圖？argmax 解碼傳回什麼，誤差是多少？四分之一偏移傳回什麼？在無雜訊的高斯上做精確的 Taylor 步驟會傳回什麼？

   提示

   以熱圖單位計，手腕在 $101.3/4 = 25.325$，所以峰值在第 25 欄，argmax 傳回 $25 \times 4 = 100$ 像素，誤差 1.3 像素。右鄰格（第 26 欄）比左鄰格更接近 25.325，所以四分之一偏移移到 25.25，對應 101.0 像素（誤差 0.3）。在無雜訊的高斯上，對數熱圖是精確的拋物線，所以 Taylor 步驟傳回 25.325，也就是 101.3 像素。

2. **手算 OKS**。某人的分割面積是 10,000 像素 ^{2}，所以 $s = 100$。預測的膝蓋離真值 10 像素。用 COCO 的常數計算它的關鍵點相似度。再對偏了 10 像素的鼻子重算一次。你寧願犯哪一種誤差？為什麼 COCO 對兩者的處理差這麼多？

   提示

   膝蓋：$\kappa = 2 \times 0.087 = 0.174$，$s\kappa = 17.4$，相似度 $= \exp(-100/(2 \times 17.4^2)) = \exp(-0.165) \approx 0.85$。鼻子：$\kappa = 0.052$，$s\kappa = 5.2$，相似度 $= \exp(-100/54.08) \approx 0.16$。標註者放置鼻子的一致性大約是膝蓋的三倍，而 COCO 依這種分歧程度來調整容忍度。

3. **由上而下何時變得太慢？**人物偵測器每張影像花 20 ms，單人網路每個裁切區域花 5 ms。由下而上的網路每張影像花 40 ms，另外每個人需要 1 ms 的分組時間。每張影像多少人時，由下而上比較快？除了速度之外，請說出在擁擠的火車站中偏好其中一種流程的兩個理由。

   提示

   由上而下的成本是 $20 + 5N$ ms，由下而上是 $40 + N$ ms，所以當 $4N > 20$，也就是超過 5 個人時，由下而上比較快。在擁擠的車站裡，由上而下受害於重疊的框與 NMS 刪掉人，但因為每個裁切區域都會縮放，所以處理遠處的小尺寸的人比較好；由下而上的網路成本固定，但必須對嚴重重疊的肢段分組，而且小尺寸的人只佔到很少的熱圖格子。

4. **破壞部分親和場**。在分組程式碼中，移動人物 B，讓他的左前臂與人物 A 的左前臂位在同一條線上並互相重疊（一個人站在另一個人後面）。兩個正確配對與兩個錯誤配對的 PAF 分數會怎麼樣？為什麼？還能用什麼額外線索解決這個情況？

提示

兩個肢段重疊處，目標場是兩個相同單位向量的平均，所以它沿著共用的那條線。這條線上任何手肘到手腕的候選線段，不論對錯，都與場對齊，得到相近的分數，所以配對變得有歧義。可以幫上忙的線索有：肢段相對於各自身體其他部分的長度、像關聯嵌入那樣的 tag、每個人的其他肢段（採全域而非逐肢段的解析），或影片中的時間連續性。

5. **相機看不見的對調**。修改三角測量的程式碼，讓兩台相機都瞄準骨架的中心，並在第二台相機中對調左右膝蓋的 2D 位置（2D 偵測器的反轉誤差）。對左膝蓋做三角測量，計算它在兩個視角中的平均重投影誤差，分別比較有對調與沒有對調的情況。接著把第二台相機移到第一台上方 1 公尺處（而不是旁邊）再做一次。解釋兩者的差異。

提示

在我們對這個實驗的執行中，相機並排時，不管有沒有對調，重投影誤差都約 1 像素，但對調後的膝蓋落點離真值超過半公尺。兩個膝蓋高度幾乎相同，水平基線下對極線幾乎是水平的，所以對調膝蓋是把觀測點移到它的對極線上：兩條射線仍然相交，只是深度錯了。第二台相機在第一台上方時，對極線變成幾乎垂直，對調會把觀測點移離它的線，重投影誤差跳到幾十像素，因此能標示出這個錯誤。所以多視角姿態的相機配置不應該把所有相機放在同一條水平線上。

6. **忘了對調通道**。你做了翻轉測試，但在平均之前忘了對調鏡像熱圖的左右通道。對一個面向相機、雙臂張開的人，平均後的左手腕熱圖看起來如何？argmax 解碼傳回什麼？

提示

想想網路在鏡像影像裡看到什麼：一個面向相機的人，所以它把影像右側的手腕標成「左手腕」。那個手腕其實是那個人的右手腕，被鏡射了。把圖翻回來卻不對調通道，就會在左手腕通道中，把光斑放在真正的右手腕上。平均後有兩個高度只有一半的光斑，每個手腕上各一個，argmax 挑到稍微高一點的那個，所以大約有一半的時間，左手腕會被回報在右手腕的位置：這是反轉誤差。

參考文獻

1. A. Mathis, P. Mamidanna, K. M. Cury, T. Abe, V. N. Murthy, M. W. Mathis and M. Bethge, “DeepLabCut: Markerless pose estimation of user-defined body parts with deep learning,” *Nature Neuroscience*, vol. 21, no. 9, 2018. doi ·arXiv
2. C. Zheng, W. Wu, C. Chen, T. Yang, S. Zhu, J. Shen, N. Kehtarnavaz and M. Shah, “Deep learning-based human pose estimation: A survey,” *ACM Computing Surveys*, vol. 56, no. 1, 2024. doi ·arXiv
3. Y. Chen, Y. Tian and M. He, “Monocular human pose estimation: A survey of deep learning-based methods,” *Computer Vision and Image Understanding*, vol. 192, 102897, 2020. doi ·arXiv
4. COCO Consortium, “Keypoint evaluation,” COCO dataset official documentation (accessed 2026-11-01). cocodataset.org
5. Y. Tian, H. Zhang, Y. Liu and L. Wang, “Recovering 3D human mesh from monocular images: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 12, 2023. doi ·arXiv

6. A. Toshev and C. Szegedy, “DeepPose: Human pose estimation via deep neural networks,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2014. doi ·arXiv
7. J. Tompson, A. Jain, Y. LeCun and C. Bregler, “Joint training of a convolutional network and a graphical model for human pose estimation,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2014. NeurIPS ·arXiv
8. A. Newell, K. Yang and J. Deng, “Stacked hourglass networks for human pose estimation,” in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, 2016. doi ·arXiv
9. B. Xiao, H. Wu and Y. Wei, “Simple baselines for human pose estimation and tracking,” in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, 2018. doi ·arXiv
10. F. Zhang, X. Zhu, H. Dai, M. Ye and C. Zhu, “Distribution-aware coordinate representation for human pose estimation,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2020. doi ·arXiv
11. X. Sun, B. Xiao, F. Wei, S. Liang and Y. Wei, “Integral human pose regression,” in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, 2018. doi ·arXiv
12. Y. Li, S. Yang, P. Liu, S. Zhang, Y. Wang, Z. Wang, W. Yang and S.-T. Xia, “SimCC: A simple coordinate classification perspective for human pose estimation,” in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, 2022. doi ·arXiv
13. J. Li, S. Bian, A. Zeng, C. Wang, B. Pang, W. Liu and C. Lu, “Human pose regression with residual log-likelihood estimation,” in *Proc. IEEE/CVF International Conference on Computer Vision* (*ICCV*), 2021. doi ·arXiv
14. S.-E. Wei, V. Ramakrishna, T. Kanade and Y. Sheikh, “Convolutional pose machines,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2016. doi ·arXiv
15. K. Sun, B. Xiao, D. Liu and J. Wang, “Deep high-resolution representation learning for human pose estimation,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2019. doi ·arXiv
16. Y. Xu, J. Zhang, Q. Zhang and D. Tao, “ViTPose: Simple vision transformer baselines for human pose estimation,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022. NeurIPS ·arXiv
17. K. He, G. Gkioxari, P. Dollár and R. Girshick, “Mask R-CNN,” in *Proc. IEEE International Conference on Computer Vision* (*ICCV*), 2017. doi ·arXiv
18. Z. Cao, T. Simon, S.-E. Wei and Y. Sheikh, “Realtime multi-person 2D pose estimation using part affinity fields,” in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2017. doi ·arXiv
19. A. Newell, Z. Huang and J. Deng, “Associative embedding: End-to-end learning for joint detection and grouping,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2017. NeurIPS ·arXiv
20. B. Cheng, B. Xiao, J. Wang, H. Shi, T. S. Huang and L. Zhang, “HigherHRNet: Scale-aware representation learning for bottom-up human pose estimation,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2020. doi ·arXiv
21. Z. Geng, K. Sun, B. Xiao, Z. Zhang and J. Wang, “Bottom-up human pose estimation via disentangled keypoint regression,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2021. doi ·arXiv
22. X. Zhou, D. Wang and P. Krähenbühl, “Objects as points,” arXiv:1904.07850, 2019. arXiv
23. P. Lu, T. Jiang, Y. Li, X. Li, K. Chen and W. Yang, “RTMO: Towards high-performance one-stage

real-time multi-person pose estimation," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2024. doi ·arXiv
24. J. Yang, A. Zeng, S. Liu, F. Li, R. Zhang and L. Zhang, "Explicit box detection unifies end-to-end multi-person pose estimation," in *Proc. International Conference on Learning Representations* (*ICLR*), 2023. OpenReview ·arXiv
25. Y. Yang and D. Ramanan, "Articulated human detection with flexible mixtures of parts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, 2013. doi
26. M. Andriluka, L. Pishchulin, P. Gehler and B. Schiele, "2D human pose estimation: New benchmark and state of the art analysis," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2014. doi ·CVF
27. COCO Consortium, `cocoeval.py` (keypoint OKS computation and the per-keypoint sigmas), official COCO API source code (accessed 2026-11-01). GitHub
28. J. Martinez, R. Hossain, J. Romero and J. J. Little, "A simple yet effective baseline for 3d human pose estimation," in *Proc. IEEE International Conference on Computer Vision* (*ICCV*), 2017. doi · arXiv
29. M. R. Ronchi and P. Perona, "Benchmarking and error diagnosis in multi-instance pose estimation," in *Proc. IEEE International Conference on Computer Vision* (*ICCV*), 2017. doi ·arXiv
30. T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, 2014. doi ·arXiv
31. J. Li, C. Wang, H. Zhu, Y. Mao, H.-S. Fang and C. Lu, "CrowdPose: Efficient crowded scenes pose estimation and a new benchmark," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2019. doi ·arXiv
32. S. Jin, L. Xu, J. Xu, C. Wang, W. Liu, C. Qian, W. Ouyang and P. Luo, "Whole-body human pose estimation in the wild," in *Proc. European Conference on Computer Vision* (*ECCV*), LNCS, 2020. doi ·arXiv
33. M. Andriluka, U. Iqbal, E. Insafutdinov, L. Pishchulin, A. Milan, J. Gall and B. Schiele, "PoseTrack: A benchmark for human pose estimation and tracking," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2018. doi ·arXiv
34. C. Ionescu, D. Papava, V. Olaru and C. Sminchisescu, "Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, 2014. doi ·project page
35. H. Yu, Y. Xu, J. Zhang, W. Zhao, Z. Guan and D. Tao, "AP-10K: A benchmark for animal pose estimation in the wild," in *Advances in Neural Information Processing Systems* (*NeurIPS*) *Datasets and Benchmarks Track*, 2021. NeurIPS ·arXiv
36. Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei and Y. Sheikh, "OpenPose: Realtime multi-person 2D pose estimation using part affinity fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, 2021. doi ·arXiv
37. D. Pavllo, C. Feichtenhofer, D. Grangier and M. Auli, "3D human pose estimation in video with temporal convolutions and semi-supervised training," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2019. doi ·arXiv
38. K. Iskakov, E. Burkov, V. Lempitsky and Y. Malkov, "Learnable triangulation of human pose," in *Proc. IEEE/CVF International Conference on Computer Vision* (*ICCV*), 2019. doi ·arXiv
39. M. Loper, N. Mahmood, J. Romero, G. Pons-Moll and M. J. Black, "SMPL: A skinned multi-

person linear model," *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, vol. 34, no. 6, 2015. doi

40. A. Kanazawa, M. J. Black, D. W. Jacobs and J. Malik, "End-to-end recovery of human shape and pose," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. doi ·arXiv
41. S. Goel, G. Pavlakos, J. Rajasegaran, A. Kanazawa and J. Malik, "Humans in 4D: Reconstructing and tracking humans with transformers," in *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023. doi ·arXiv
42. R. Khirodkar, T. Bagautdinov, J. Martinez, S. Zhaoen, A. James, P. Selednik, S. Anderson and S. Saito, "Sapiens: Foundation for human vision models," in *Proc. European Conference on Computer Vision (ECCV)*, LNCS, 2024. doi ·arXiv
43. T. Jiang, P. Lu, L. Zhang, N. Ma, R. Han, C. Lyu, Y. Li and K. Chen, "RTMPose: Real-time multi-person pose estimation based on MMPose," arXiv:2303.07399, 2023. arXiv

DLCV 06・影像中的文字：OCR 與文件理解

機器如何讀出照片、掃描檔與螢幕截圖裡的文字：偵測、辨識與版面管線，以及每個階段在哪裡失敗；迴歸式與分割式文字偵測器、CTC、注意力與 Transformer 辨識器、端到端文字讀取、LayoutLM 式與免 OCR 的文件模型；CER、WER 與 H-mean；標準基準資料集，以及真正重要的失效模式。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/

你將學到

- 區分文字偵測、辨識、版面分析與文件理解。
- 比較幾何式與分割式文字偵測器。
- 使用 CTC、注意力或 Transformer 模型解碼字元序列。
- 理解版面感知與不依賴獨立 OCR 的文件模型。
- 以適當協定評估辨識、偵測與下游擷取。

先看全貌

影像裡到處都是文字：路標、商品標籤、車牌、收據、表單、投影片與螢幕截圖。讀出它們，就是把像素變成程式可以搜尋、翻譯、檢查或儲存的字串，這件事稱為**光學字元辨識**（optical character recognition，OCR）。困難的版本是一般照片裡的文字，稱為**場景文字**（scene text）閱讀；有結構的版本是頁面的掃描檔與 PDF，稱為**文件**（document）分析。

> **用最簡單的話說**
>
> 想想你怎麼讀牆上的海報。你的眼睛先找出有字的區塊（這是**偵測**）。接著你湊近看其中一塊，把字母一個個唸出來，遇到模糊的字就憑「這個詞通常長什麼樣」來猜（這是**辨識**）。最後你才弄懂海報在說什麼：最上面的大字是標題，最下面的小字是日期，兩欄文字要一欄讀完再讀下一欄（這是**版面與理解**）。電腦也做同樣三件事，而且每一件都可能出錯。

為什麼要獨立成一篇？一個單字又長又細，常常旋轉或彎曲，還緊貼著鄰居；它的「類別」不是固定清單裡的標籤，而是一個字母表上的任意字串，這個字母表可能有好幾萬個符號；而且一個字元寫錯就可能改變意思（發票上的「1046」與「1O46」）。所以這個領域有自己的偵測器、序列辨識器、指標與基準資料集。現在又有一類模型會讀完整頁並直接寫出結構化文字，讓整條管線變成可有可無，卻也帶來新的風險：影像並不支持、但讀起來很流暢的輸出。圖 1 畫出這條管線，以及每個階段常見的失敗。

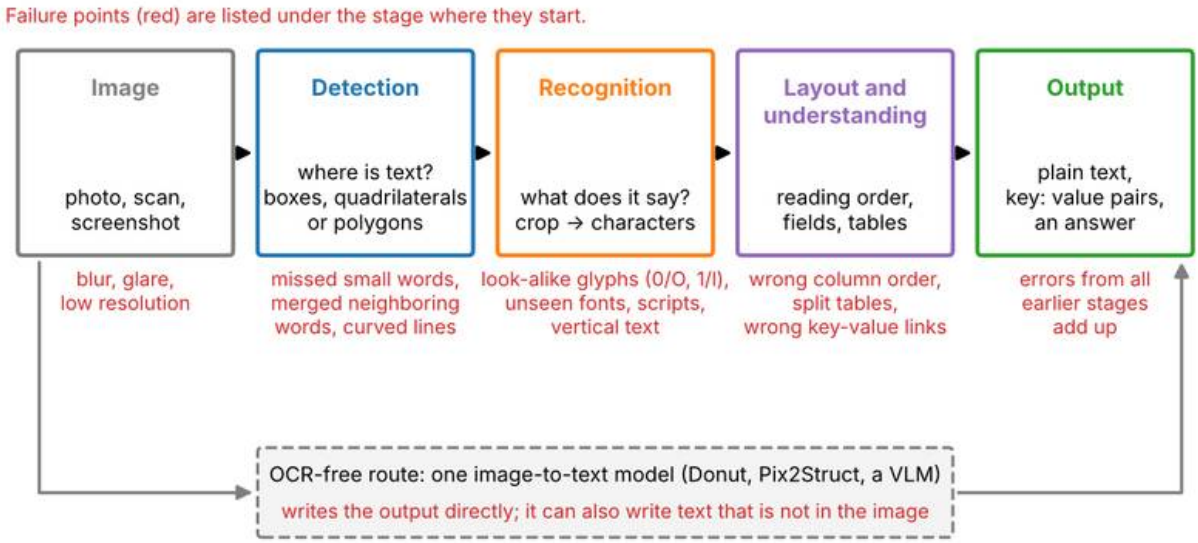


圖 *1* — 閱讀管線。每個階段都有自己常見的失敗（紅字），前面階段犯的錯後面無法修正。免 *OCR* 模型用單一的影像到文字模型取代中間的階段。

本篇假設你已經讀過 DLCV 03，知道邊界框、IoU、NMS 與精確率–召回率。其他概念用連結帶過、不重複講：CTC 演算法（PR 06 的 CTC 補充資料）、編輯距離（PR 06）、注意力（NLP 07）與視覺語言模型（ML 10）。Long 等人 [1] 與 Chen 等人 [2] 的綜述整理了場景文字；Cui 等人 [3] 與 Zhang 等人 [4] 則涵蓋文件。

閱讀管線與出錯的地方

白話版。傳統的 OCR 系統是一條鏈：找出文字區域、逐一讀出來，再把讀到的內容排好順序、賦予意義。每一環都會把自己的錯誤傳給下一環。

場景文字與文件文字

這兩種情境困難的地方不一樣。

	場景文字	文件文字
來源	照片、影片、街景	掃描檔、PDF、螢幕截圖、用手機拍的頁面
數量	每張影像幾個字	每頁數百到數千字
幾何	任意方向、透視、曲線	大多是水平行；有欄與表格
外觀	裝飾字體、雜亂背景、反光、模糊	小字、雜訊、印章、手寫
主要困難	找到並讀出每個單字	閱讀順序、結構與意義

基準資料集也依此分成兩類。與 ICDAR（International Conference on Document Analysis and Recognition，國際文件分析與辨識會議）同時舉辦的 Robust Reading 競賽，定義了許多標準的

場景文字基準：ICDAR 2013 是對焦清楚、大多水平的英文文字 [5]；ICDAR 2015 的 *Incidental Scene Text* 挑戰賽 [6] 用穿戴式相機（Google Glass）拍攝，單字常常很小、模糊又旋轉 [7]。文件基準則要求找出版面區域、表單欄位，或回答關於頁面的問題（見文件理解）。

錯誤從四個地方開始

知道錯誤出在哪個階段，比較容易修。

1. **影像**。模糊、反光、低解析度與壓縮會把證據抹掉。像素裡沒有的字元，後面的階段都救不回來；若有生成模型「救回」了它，那只是猜的（見失效模式）。
2. **偵測**。漏掉的單字就永遠消失了；兩個單字被合併成一個框，會被讀成一個字串；框切掉最後一個字母，就會產生刪除錯誤。
3. **辨識**。長得像的字形（0/O、1/l/I、rn/m）、沒見過的字體與文字系統，以及直排文字，都會造成替換錯誤。
4. **版面與理解**。字都對但順序錯、表格被橫著讀而不是直著讀、或是把數值接到錯的鍵上，都會讓完美的字元得到錯誤的答案。

一個好習慣：先單獨評估每個階段（例如用真值裁切圖評估辨識），再評估整條鏈，這樣才知道該改進哪一個階段。

深度學習之前：閾值與連通元件

白話版。在乾淨頁面可對深色像素門檻化、分組連通區域並擬合方框；不均勻背景與相接字元會揭露此基準的限制。

傳統做法有三個步驟。**二值化**（binarization）拿像素和閾值比較，把每個像素標成墨水或紙。單一的全域閾值，例如用 Otsu 方法 [8] 選出的那個（推導見 IP 10），在光線不均時會失敗，所以文件系統改用局部閾值，在每個像素周圍的視窗內計算。Sauvola 方法 [9] 是為文件影像設計的，很常用：

$$T(x,y) = m(x,y)\left[1 + k\left(\frac{s(x,y)}{R} - 1\right)\right],$$

其中 $T(x,y)$ 是像素 (x,y) 處的閾值，$m(x,y)$ 與 $s(x,y)$ 是以該像素為中心、$w \times w$ 視窗內灰階值的平均與標準差，R 是標準差的動態範圍，$k > 0$（scikit-image 的預設值是 0.2）控制局部對比會把閾值壓低多少。比 T 暗的像素就是墨水。接著**分組**（grouping）把墨水像素併成**連通元件**（connected components，IP 09），通常會先做一次水平膨脹，把單字內字母之間的空隙接起來。**取框**（box extraction）則取每個連通元件的外接框。

下面的程式用 matplotlib 畫一張小發票，把左側調暗、加上雜訊，再用全域與局部閾值各跑一次這套做法。它用文字基準使用的精確率、召回率與 H-mean 為框打分（精確定義見評估 OCR）：若預測框和某個尚未被配對的真值單字的 IoU 至少 0.5，就算正確。

```
import numpy as np
import matplotlib
matplotlib.use("Agg")
```

```
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from skimage.filters import threshold_otsu, threshold_sauvola

H, W = 200, 560
WORDS = [("INVOICE", 20, 18, 22), ("No.", 200, 24, 14), ("1046", 245, 24, 14),
         ("Date:", 330, 24, 14), ("2026-11-01", 385, 24, 14),
         ("Item", 20, 80, 13), ("Qty", 260, 80, 13), ("Price", 400, 80, 13),
         ("Paper", 20, 110, 13), ("12", 265, 110, 13), ("3.50", 405, 110, 13),
         ("Total", 20, 160, 15), ("42.00", 395, 160, 15)]     # (word, x, y, font size)

def render(words):
    """Draw words with matplotlib; return a gray image in [0, 1] and word boxes."""
    fig = plt.figure(figsize=(W / 100, H / 100), dpi=100)
    texts = [fig.text(x / W, 1 - y / H, s, fontsize=fs, va="top") for s, x, y, fs in words]
    fig.canvas.draw()
    img = np.asarray(fig.canvas.buffer_rgba())[..., :3].mean(axis=2) / 255.0
    r = fig.canvas.get_renderer()
    boxes = [[b.x0, H - b.y1, b.x1, H - b.y0]
             for b in (t.get_window_extent(r) for t in texts)]
    plt.close(fig)
    return img, np.array(boxes)

def iou(a, b):
    w = min(a[2], b[2]) - max(a[0], b[0]); h = min(a[3], b[3]) - max(a[1], b[1])
    inter = max(w, 0) * max(h, 0)
    area = lambda r: (r[2] - r[0]) * (r[3] - r[1])
    return inter / (area(a) + area(b) - inter)

def detect(img, thresh, gap=9):
    ink = img < thresh                                    # dark pixels = ink
    ink = ndi.binary_opening(ink, np.ones((2, 2)))       # drop isolated noise pixels
    words = ndi.binary_dilation(ink, np.ones((3, gap)))  # bridge gaps between letters
    lab, _ = ndi.label(words)
    out = []
    for y, x in ndi.find_objects(lab):
        if (y.stop - y.start) * (x.stop - x.start) > 40:   # ignore tiny blobs
            out.append([x.start, y.start, x.stop, y.stop])
    return np.array(out, float)

def h_mean(pred, gt, thr=0.5):
    used, tp = set(), 0
    for p in pred:                                        # greedy one-to-one matching
        cand = [(iou(p, g), j) for j, g in enumerate(gt) if j not in used]
        best = max(cand, default=(0, -1))
        if best[0] >= thr:
            used.add(best[1]); tp += 1
    P, R = tp / max(len(pred), 1), tp / len(gt)
    return P, R, 2 * P * R / max(P + R, 1e-9)

rng = np.random.default_rng(0)
clean, gt = render(WORDS)
shade = 0.45 + 0.55 * np.linspace(0, 1, W)[None, :]      # light falls off to the left
img = np.clip(clean * shade + rng.normal(0, 0.04, clean.shape), 0, 1)

t_sau = threshold_sauvola(img, window_size=31)
```

```
for name, t, gap in [("Otsu, gap 9", threshold_otsu(img), 9), ("Sauvola, gap 3", t_sau, 3),
                     ("Sauvola, gap 9", t_sau, 9), ("Sauvola, gap 25", t_sau, 25)]:
    pred = detect(img, t, gap)
    P, R, F = h_mean(pred, gt)
    print(f"{name:16s} boxes={len(pred):2d}  P={P:.2f}  R={R:.2f}  H-mean={F:.2f}")
```

```
Otsu, gap 9      boxes= 8  P=0.62  R=0.38  H-mean=0.48
Sauvola, gap 3   boxes=33  P=0.15  R=0.38  H-mean=0.22
Sauvola, gap 9   boxes=13  P=1.00  R=1.00  H-mean=1.00
Sauvola, gap 25  boxes=11  P=0.64  R=0.54  H-mean=0.58
```

圖 2 顯示發生了什麼事。全域閾值把整個暗的左側都標成墨水，那裡的單字就消失在一個巨大的假框裡。局部閾值找回了每個單字，但只在人工挑選的某個間隙值下成立：膨脹太窄會把單字切成字母，太寬又會把「No.」與「1046」、「Date:」與日期合併。這裡所有數字都來自這個合成的玩具範例。

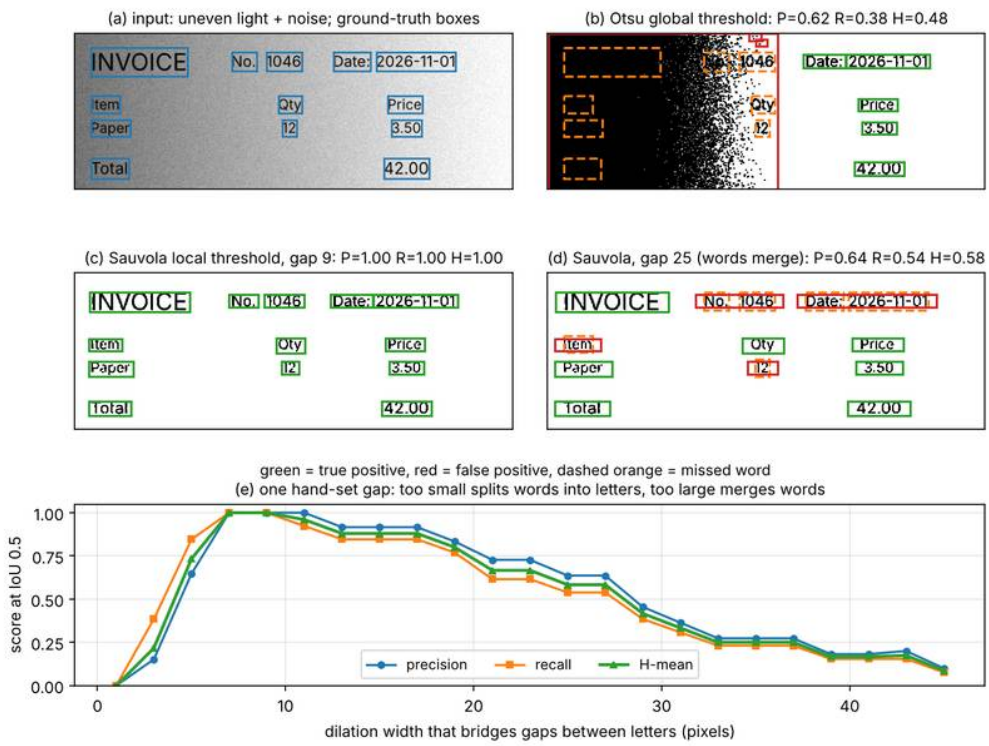


圖 2 — 合成發票上的傳統偵測（玩具資料）。(*a*) 輸入與真值。(*b*) 全域閾值在光線不均時失敗。(*c*) 局部（*Sauvola*）閾值搭配 *9* 像素的間隙，找到每個單字。(*d*) *25* 像素的間隙把鄰居合併。(*e*) *IoU 0.5* 下的分數對間隙作圖：同一條固定規則無法適用於每種字體大小與間距。

這個實驗正是學習式文字偵測器的動機。真實影像同時包含各種字體大小、紋理背景上的文字、旋轉，以及單字之間比其他單字內字母之間還近的情況。網路可以從上下文與外觀學會文字在哪裡，也能學會把鄰居分開。

文字偵測

白話版。文字偵測器須以幾何預測或文字遮罩，處理細長、旋轉、彎曲及間距很近的區域。

為什麼文字不只是另一種物件

DLCV 03 的偵測器預測的是正立（與座標軸對齊）的框。對文字來說，這在三個地方會出問題。

- **極端的長寬比**。一行文字可能比它的高度寬很多倍，遠遠超出為一般物件調好的錨框形狀。
- **方向與彎曲**。招牌與包裝上的字會旋轉、有透視變形，或沿著曲線排列。所以基準資料集標註的是四邊形（ICDAR 2015 [6]）或多邊形（Total-Text [10]、CTW1500 [11]），而且是在多邊形之間而不是矩形之間計算 IoU。
- **緊鄰的單字**。一行上的單字只隔幾個像素；把它們合併，或讓 NMS 刪掉其中一個，都會降低召回率。

前兩個問題合在一起特別糟。下面的程式在一個細長單字旋轉時，計算它和最緊貼它的正立框之間的 IoU。

```
import numpy as np
from skimage.draw import polygon

def rotated_word(cx, cy, w, h, deg):
    """Corners of a w x h word box rotated by deg around its center (x, y)."""
    t = np.deg2rad(deg)
    R = np.array([[np.cos(t), -np.sin(t)], [np.sin(t), np.cos(t)]])
    corners = np.array([[-w, -h], [w, -h], [w, h], [-w, h]]) / 2
    return corners @ R.T + [cx, cy]

def mask(quad, shape=(300, 300)):
    m = np.zeros(shape, bool)
    rr, cc = polygon(quad[:, 1], quad[:, 0], shape)
    m[rr, cc] = True
    return m

for deg in [0, 10, 20, 30, 45]:
    quad = rotated_word(150, 150, 160, 24, deg)          # a long, thin word
    x0, y0 = quad.min(axis=0); x1, y1 = quad.max(axis=0)
    aabb = np.array([[x0, y0], [x1, y0], [x1, y1], [x0, y1]])  # tightest upright box
    a, b = mask(quad), mask(aabb)
    print(f"angle {deg:2d} deg: IoU(word, its upright box) = {(a & b).sum() / (a |
     ↪  b).sum():.2f}")
```

```
angle  0 deg: IoU(word, its upright box) = 1.00
angle 10 deg: IoU(word, its upright box) = 0.47
angle 20 deg: IoU(word, its upright box) = 0.31
angle 30 deg: IoU(word, its upright box) = 0.25
angle 45 deg: IoU(word, its upright box) = 0.22
```

對一個旋轉 θ 的 $w \times h$ 矩形，精確值是 $wh/[(w\cos\theta + h\sin\theta)(w\sin\theta + h\cos\theta)]$，其中 w 與 h 是單字的寬與高，分母是它的正立外接框的面積。才轉 10 度，最好的正立框就通不過常用的 IoU 0.5 門檻了（圖 3）。因此文字偵測器輸出的是旋轉矩形、四邊形或多邊形；旋轉形狀的 IoU 怎麼算，和 DLCV 03B 為鳥瞰圖框解決的多邊形裁切是同一個問題。

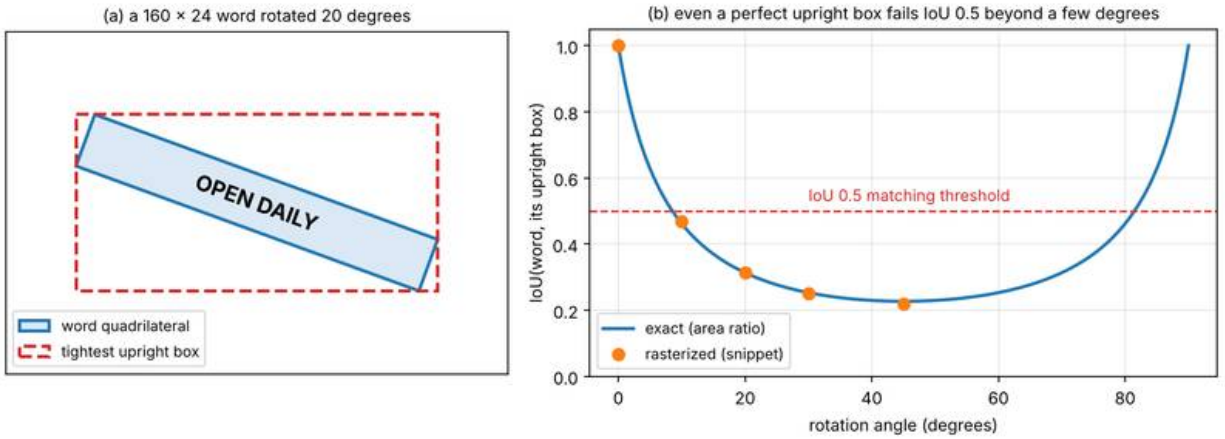


圖 3 — 為什麼文字偵測器要預測四邊形。(*a*) 一個旋轉 *20°* 的 *160 × 24* 單字，以及緊貼它的正立框。(*b*) 兩者的 *IoU* 對角度作圖：超過約 *8.5°* 之後，即使是完美的正立框，在 *IoU 0.5* 下也算漏檢。

以迴歸為基礎的偵測器：EAST

白話版。各文字像素預測信心與單字幾何，再把一致的高信心預測合併成偵測結果。

EAST（Efficient and Accurate Scene Text detector）[12] 是一個全卷積網路，帶有 U 形的特徵合併分支，在降低取樣後的圖上每個位置輸出一個 $[0,1]$ 之間的**分數**（這個像素是否在縮小後的單字內？）以及一個**幾何**。它提出兩種幾何。**RBOX** 有五個通道：像素到旋轉矩形四邊的距離，以及矩形的旋轉角。**QUAD** 有八個通道：像素到四邊形四個角的位移 $(\Delta x_i, \Delta y_i)$，$i = 1, \dots, 4$。因為有幾萬個像素在投票，EAST 先逐列合併相鄰的幾何，以分數為權重做平均（**局部感知 NMS**，locality-aware NMS），然後才跑標準 NMS [12]。一次網路前向傳遞，就取代了早期系統中各個候選分組步驟。它的弱點是感受距離：像素必須預測可能遠在數百像素外的邊緣，而且四邊形無法順著彎曲的單字。

以分割為基礎的偵測器：PSENet 與 DBNet

白話版。不要要求每個像素預測整個單字的形狀，只問「這個像素是文字嗎？」這樣得到的遮罩可以跟著任何曲線走。困難在於單字貼在一起時，要怎麼把遮罩切成單字，所以這些偵測器為每個單字預測一個更細的核心，讓核心彼此分開，事後再把它們長回去。

網路是類似 DLCV 04 的編碼器–解碼器，預測逐像素的文字**機率圖** P；訣竅在標籤上。面積為 A、周長為 L 的每個真值多邊形 G，都向內縮小一個位移

$$D = \frac{A\,(1 - r^2)}{L},$$

其中 $r \in (0,1)$ 是縮小比例；縮小後的多邊形（單字的**核心**，kernel）是正樣本區域。即使單字互相貼在一起，相鄰單字的核心之間仍有間隔，所以預測核心圖的連通元件就是單字實例；之後再把每個膨脹回原本大小。

PSENet（Progressive Scale Expansion Network）[13] 為每個單字預測多個不同縮小比例的核心，並用廣度優先搜尋，把最小、彼此分得開的核心長進較大的核心裡，這樣一個像素被某個單字佔走之後，鄰居就搶不走。代價是後處理緩慢又得依序進行。

DBNet [14] 只保留一個核心，並讓二值化這一步本身可以訓練。標準的二值化 $B = \mathbf{1}[P \ge t]$ 使用固定閾值 t，幾乎處處梯度為零，無法訓練。DBNet 另外預測第二張圖，即**閾值圖** T，並使用近似的階梯函數

$$\hat{B}_{i,j} = \frac{1}{1 + e^{-k\,(P_{i,j} - T_{i,j})}},$$

其中 $\hat{B}_{i,j}$ 是像素 (i,j) 處的近似二值，$P_{i,j}$ 是預測的文字機率，$T_{i,j}$ 是學到的閾值，k 是放大係數，設為 50 [14]。令 $x = P_{i,j} - T_{i,j}$、$f(x) = \hat{B}_{i,j}$，文字像素的二元交叉熵 $\ell_+ = -\log f(x)$ 的導數是 $\partial \ell_+ / \partial x = -k\,(1 - f(x))$；背景像素的 $\ell_- = -\log(1 - f(x))$，導數是 $\partial \ell_- / \partial x = k f(x)$。梯度被放大 k 倍，並且在像素落在閾值錯誤那一側時最大。閾值圖被監督成在每個單字邊界周圍的一圈帶狀區域內偏高，所以單字相接處的二值化很嚴格，單字內部則很寬鬆。

下面的程式用數值方法檢查導數，並在一維剖面上示範：兩個單字之間的閾值調高後可以把它們分開，而固定閾值會把它們合併。

```
import numpy as np

def db(P, T, k=50.0):
    """Differentiable binarization: an approximate step at the learned threshold T."""
    return 1.0 / (1.0 + np.exp(-k * (P - T)))

x = np.array([-0.10, -0.02, 0.0, 0.02, 0.10])            # x = P - T at five pixels
for k in [1.0, 50.0]:
    f = db(x, 0.0, k)
    g_pos = -k * (1 - f)       # d/dx of -log f       (pixel labeled text)
    g_neg = k * f              # d/dx of -log(1 - f)  (pixel labeled background)
    # check the text-pixel gradient by finite differences
    eps = 1e-6
    num = (-np.log(db(x + eps, 0, k)) + np.log(db(x - eps, 0, k))) / (2 * eps)
    assert np.allclose(num, g_pos, rtol=1e-4)
    print(f"k={k:4.0f}  B_hat={np.round(f, 3)}  dl+/dx={np.round(g_pos, 2)}
     ↪  dl-/dx={np.round(g_neg, 2)}")

# Two words whose probability map P does not drop to 0 between them.
P = np.array([0.1, 0.9, 0.95, 0.9, 0.55, 0.5, 0.55, 0.9, 0.95, 0.9, 0.1])
T = np.array([0.3, 0.3, 0.3, 0.3, 0.7, 0.7, 0.7, 0.3, 0.3, 0.3, 0.3])   # high at the gap
print("fixed t=0.3:", (P >= 0.3).astype(int))
print("DB, k=50   :", (db(P, T) > 0.5).astype(int))
```

```
k=   1  B_hat=[0.475 0.495 0.5   0.505 0.525]  dl+/dx=[-0.52 -0.5  -0.5  -0.5  -0.48]
 ↪  dl-/dx=[0.48 0.5  0.5  0.5  0.52]
k=  50  B_hat=[0.007 0.269 0.5   0.731 0.993]  dl+/dx=[-49.67 -36.55 -25.   -13.45  -0.33]
 ↪  dl-/dx=[ 0.33 13.45 25.   36.55 49.67]
```

```
fixed t=0.3: [0 1 1 1 1 1 1 1 1 1 0]
DB, k=50   : [0 1 1 1 0 0 0 1 1 1 0]
```

$k = 1$ 時，「二值」圖是在 0.5 附近的灰色模糊，每個像素得到的梯度都一樣小；$k = 50$ 時它幾乎是二值的，比閾值低 0.1 的文字像素會得到約 -50 的梯度，而判對的像素幾乎沒有梯度。圖 4 在二維中重現這個分離。

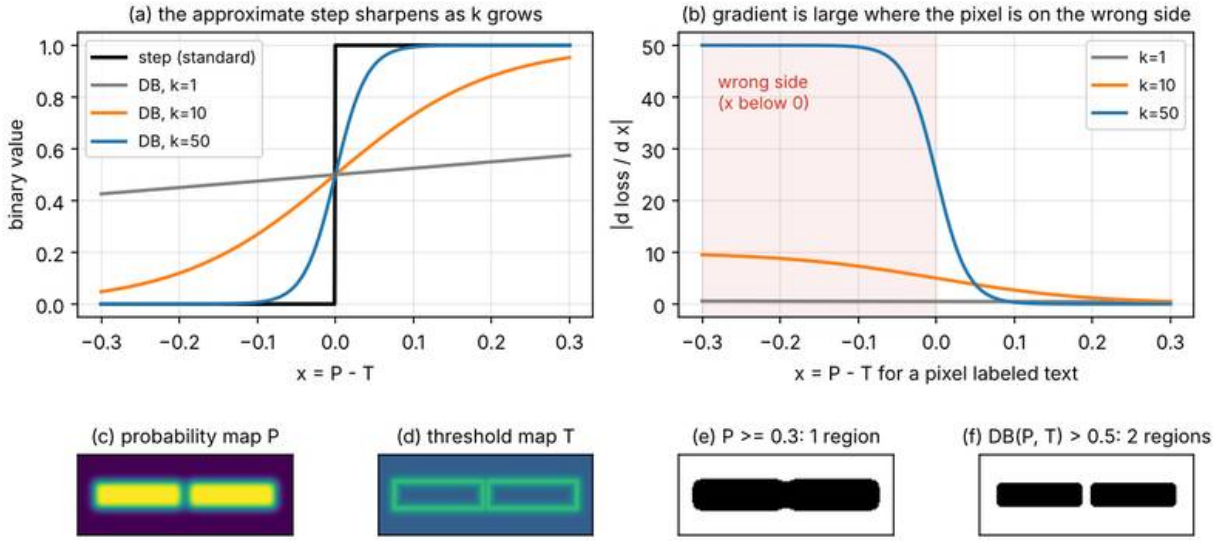


圖 *4* — 可微分二值化。(*a*) 不同 k 值的近似階梯函數。(*b*) 文字像素的梯度在閾值錯誤那一側被 k 放大。(*c*)–(*f*) 玩具圖：在 *4* 像素間隙上模糊的機率圖，用固定閾值會把兩個單字合併；而在邊界處偏高的閾值圖（這裡是手工做的，在 *DBNet* 中是學出來的）能把它們分開。

推論時 DBNet 丟掉閾值分支：它用固定的 0.2 對 P 二值化，取連通元件，並把每個區域膨脹 $D' = A'r'/L'$，其中 A' 與 L' 是該區域的面積與周長，$r' = 1.5$ [14]。所以閾值圖主要是訓練時的工具，用來讓 P 本身變得銳利；簡單的後處理讓偵測器保持快速。

僅線上版：縮小與放大文字多邊形 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/notes/縮小與放大文字多邊形/*

僅線上版：*DBNet* 導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/papers/dbnet-導讀/*

字元層級與彎曲文字偵測器

有些偵測器先找字母，再把鄰近的字母黏成單字。CRAFT [15] 預測**區域分數**（region score，字元的中心）與**親和分數**（affinity score，同一單字內兩個字元之間），並把字元連成任意形狀的單字；因為真實資料集只標註單字，它用一個中間模型來估計真實影像上的字元標籤。像 CTW1500 [11] 一併提出的多邊形偵測器，則是沿著彎曲的文字行迴歸許多邊界點。更豐富的形狀模型更能貼合不規則文字，但需要更複雜的輸出或後處理。

文字辨識

白話版。辨識的輸入是一個裁下來的單字或文字行，輸出是它的字元。現代的做法把裁切圖當成序列來讀：從左到右滑過去，描述每一小條，再把這一串小條轉成一串字元，完全不必把單字切成一個個字母。

把辨識當成序列預測

早期系統把單字切成字元再逐一分類，字母黏在一起或斷開時就會失敗。**CRNN**（Convolutional Recurrent Neural Network）[16] 讓辨識不再需要切割，並且可以端到端訓練。裁切圖先縮放到固定高度；CNN 把它變成特徵圖，特徵圖的各個欄構成**特徵序列**，每一欄描述影像中的一條細長直條；深層雙向 LSTM（見 ML 04）加入左右兩側的上下文；轉錄層再把每一欄的預測變成標籤序列。使用矩形的池化視窗時，100×32 的輸入會產生 25 個畫格的序列，比大多數英文單字都長 [16]。沒有任何東西告訴網路哪個畫格屬於哪個字元；CTC 處理的就是這個缺少的對齊。

CTC：不需對齊的訓練

白話版。每個畫格，網路可以說一個字元，或說「沒有新東西」（**空白**，blank）。讀輸出時，把重複的字元合併、再刪掉空白。訓練時，則把所有逐畫格、最後讀出來等於正確單字的「故事」的機率加起來。

連結時序分類（connectionist temporal classification，CTC）[17] 定義如下：對每個符號 s（字母表中的字元或空白 $\emptyset$）在畫格 t 的逐畫格機率 $q_t(s)$，一個合併映射 $\mathcal{B}$（先合併重複，再移除空白），以及給定輸入 x 時標籤序列 y 的機率

$$p(y \mid x) = \sum_{z:\ \mathcal{B}(z)=y} \prod_{t=1}^{T} q_t(z_t),$$

其中 $z = (z_1, \dots, z_T)$ 是一種對齊（每個畫格一個符號），T 是畫格數，總和涵蓋所有合併後等於 y 的對齊。訓練時最小化 $-\log p(y \mid x)$，用前向遞迴計算，長度為 L 的標籤需要 $O(TL)$。完整的推導與暴力檢查見 PR 06 的 CTC 補充資料；這裡只需要知道它對解碼意味著什麼。

最便宜的解碼器是**貪婪**（greedy，最佳路徑）解碼：每個畫格取最可能的符號，再合併。CRNN 在不使用詞典的模式下正是用這個近似 [16]。但最可能的路徑不一定會合併成最可能的文字，因為一段文字的機率分散在許多條路徑上。下面的程式建了一個五畫格的玩具範例，讓這件事看得見。

```
import itertools
import numpy as np

alphabet = ["-", "c", "a", "t"]                # index 0 is the CTC blank
# Per-frame probabilities from a recognizer (rows: 5 frames, columns: symbols).
probs = np.array([[0.10, 0.80, 0.05, 0.05],
                  [0.20, 0.05, 0.70, 0.05],
                  [0.60, 0.05, 0.30, 0.05],
                  [0.55, 0.02, 0.03, 0.40],
```

```
                  [0.55, 0.02, 0.03, 0.40]])

def collapse(path):
    """CTC map B: merge repeated symbols, then remove blanks."""
    merged = [s for i, s in enumerate(path) if i == 0 or s != path[i - 1]]
    return "".join(alphabet[s] for s in merged if s != 0)

greedy = probs.argmax(axis=1)                    # best symbol at every frame
print("greedy path:", "".join(alphabet[s] for s in greedy), "->", repr(collapse(greedy)))

# Exact: add up every alignment that collapses to the same text (4**5 = 1024 paths).
total = {}
for path in itertools.product(range(4), repeat=len(probs)):
    p = np.prod(probs[np.arange(len(probs)), path])
    total[collapse(path)] = total.get(collapse(path), 0.0) + p
for text, p in sorted(total.items(), key=lambda kv: -kv[1])[:3]:
    print(f"p({text!r}) = {p:.3f}")
print("collapse('caa-at') =", repr(collapse([1, 2, 2, 0, 2, 3])),
      " collapse('caaat') =", repr(collapse([1, 2, 2, 2, 3])))
```

```
greedy path: ca--- -> 'ca'
p('cat') = 0.363
p('ca') = 0.179
p('ct') = 0.091
collapse('caa-at') = 'caat'  collapse('caaat') = 'cat'
```

「t」在任何一個畫格都不是最可能的單一符號，所以貪婪解碼把它丟掉了；但在第 4 或第 5 畫格含有「t」的所有對齊加起來，機率是「ca」的兩倍（圖 5）。最後一行顯示空白的第二個用途：重複的字母只有在兩份之間隔著空白時才能保留。實際的解碼器使用**前綴束搜尋**（prefix beam search），它保留最好的幾個部分轉錄，並把合併後得到同一前綴的對齊併在一起。

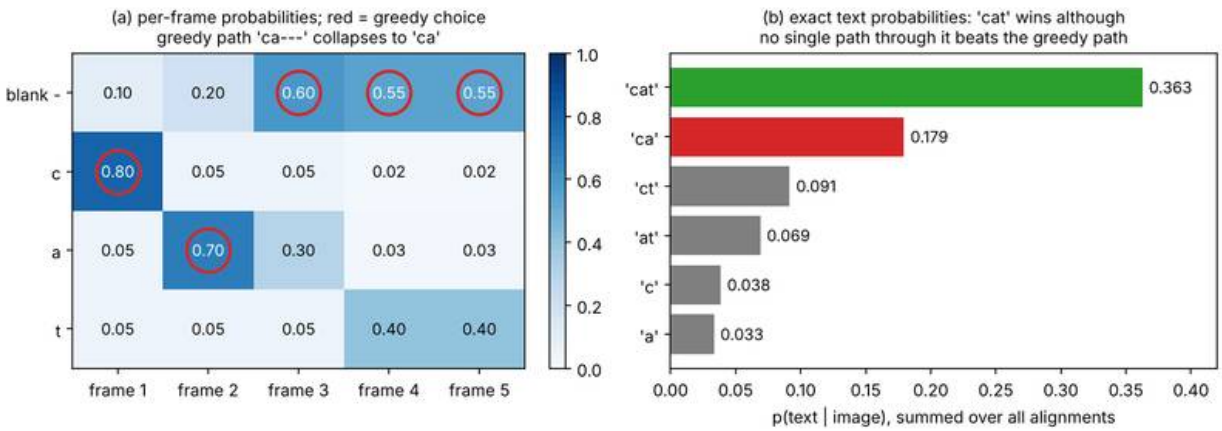


圖 5 — 玩具矩陣上的 *CTC* 解碼。(*a*) 貪婪解碼選出「*ca—*」，合併後得到「*ca*」。(*b*) 把全部 *1,024* 種對齊加總，「*cat*」的機率是「*ca*」的兩倍，儘管每一條讀成「*cat*」的路徑，都比貪婪路徑的機率低。

僅線上版：*CTC 前綴束搜尋，一步一步來* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/notes/ctc-前綴束搜尋一步一步來/*

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

CTC 有兩個內建的限制（見 PR 06 的補充資料）：給定影像，各畫格的輸出彼此獨立，所以模型無法以已經輸出的字元為條件；而且對齊是單調的，所以閱讀順序必須與畫格順序一致。直排或彎曲的文字會打破第二點，除非先矯正。

注意力解碼器與矯正

白話版。注意力解碼器一次寫一個字元。每寫一個字元之前，它會回頭看影像特徵並選擇要看哪裡；它也記得自己已經寫了什麼，所以行為像一個小小的拼字語言模型。

注意力解碼器（attention decoder）就是把 NLP 07 的編碼器–解碼器用在影像特徵上：第 i 步它對特徵序列做注意力，並根據結果與 $y_1, \dots, y_{i-1}$ 預測字元 y_i，直到結束符號。它拋棄了 CTC 的獨立性假設，代價是必須依序解碼。**ASTER** [18] 把這樣的解碼器和**矯正網路**（rectification network）配成一對：薄板樣條（thin-plate spline，TPS）轉換由影像預測、只透過辨識損失來訓練，在讀之前先把彎曲或有透視變形的單字扭正成水平的直線。

Baek 等人 [7] 指出，當時大多數辨識器都符合同一個四階段樣板：**轉換**（無或 TPS）、**特徵擷取**、**序列建模**（無或 BiLSTM）與**預測**（CTC 或注意力）；他們在相同資料上重新訓練所有組合，發現注意力主要增加的是速度成本，而且先前的比較因訓練集與測試子集不同而失真。

僅線上版：*Baek 等人：What Is Wrong With Scene Text Recognition Model Comparisons?（ICCV 2019）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/papers/baek-等人-what-is-wrong-with-scene-text-recognition-model-comparisonsiccv-2019/*

Transformer 辨識器

TrOCR [19] 用預訓練的 Transformer 取代 CNN 與 RNN。影像縮放到 384 × 384 並切成 16 × 16 的圖塊，和 Vision Transformer 一樣（ML 05）；影像編碼器由 DeiT 或 BEiT 初始化來編碼這些圖塊，文字解碼器由 RoBERTa 或 MiniLM 初始化，以自迴歸方式產生**詞片**（wordpieces，子詞符號），並用束搜尋解碼 [19]。它先在合成文字影像上預訓練，再在印刷、手寫與場景文字上微調：辨識如今從通用的視覺與語言預訓練中借來骨幹與解碼器。

PARSeq [20] 用許多隨機的輸出位置順序訓練同一個 Transformer 解碼器（**排列語言建模**，permutation language modeling），所以同一組權重既能由左到右解碼、也能不用上下文平行解碼所有字元，還能利用兩側的上下文來修正第一次的猜測。

語言模型：幫助與傷害

一個「懂英文」的辨識器能修正模糊的字母：「PLEA_E」幾乎肯定是「PLEASE」。這種知識可以是**隱含的**（自迴歸解碼器以先前的字元為條件）、**詞典**（CRNN 可以把輸出限制在一份單字清單

內 [16]），或**明確的語言模型**：在 ABINet [21] 中，一個獨立的雙向語言模型會迭代修正視覺模型的輸出。

同樣的先驗也會傷人。不是語言的文字，像序號、代碼、價格、人名與混合文字，正是 OCR 最重要的地方，也是語言模型把正確讀法「修正」成錯誤的地方。PARSeq 的作者指出，不看影像的外部語言模型有這種風險 [20]；OCRBench 則報告，大型多模態模型在**非語意文字**（隨機字元字串）、手寫、多語言文字與數學式上很弱 [22]。當內容不是自然語言時，請在有語言先驗與沒有語言先驗兩種情況下各量一次。

僅線上版：語言模型在辨識器裡的位置 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/notes/語言模型在辨識器裡的位置/*

訓練資料：合成文字

有標註的真實裁切圖很稀少，而只用**合成文字**訓練的辨識器，在各個場景文字資料集上都能有不錯的泛化 [1]。MJSynth [23] 用各種字體、陰影、扭曲與雜訊算繪單字（890 萬張單字影像）；SynthText [24] 依場景的幾何把文字貼進自然照片（550 萬個裁切單字）[7]。兩者一起訓練比單用任何一個都好，而且各取 20% 的組合樣本，就勝過任何一個完整資料集，所以多樣性可能比數量更重要 [7]。

端到端文字讀取（text spotting）

白話版。不是先跑偵測器、再對每個裁切圖跑另一個辨識器，而是由一個網路兩件事都做，並共用特徵。讀得出來有助於找到（讀起來像單字的區域，多半是文字），找到也有助於讀出來。

文字讀取器（text spotter）直接輸出（形狀，轉錄）配對。FOTS [25] 在密集的逐像素偵測分支與用 CTC 解碼的辨識分支之間共用卷積特徵，靠的是 **RoIRotate**：這個運算子把每個偵測到的四邊形的特徵區域裁下來並旋轉成正立、固定高度的長條。Mask TextSpotter [26] 擴充 Mask R-CNN（見 DLCV 04 的實例分割），為整個單字與個別字元都加上遮罩，所以不需要序列模型就能讀彎曲文字。ABCNet [27] 用參數化的貝茲曲線（Bézier curve）擬合每個彎曲的文字實例，並用 **BezierAlign**（RoIAlign 的彎曲版）沿著曲線取樣特徵。端到端評估要問的是：每個真值單字是否既被定位、也被讀對，我們在評估 OCR中定義它。

文件理解

白話版。在頁面上，讀出單字往往是最容易的部分。問題是哪些字屬於同一組、要依什麼順序讀、它們是什麼意思：哪個數字是總額、哪一行是標題、哪個儲存格在哪一欄。

版面分析與閱讀順序

版面分析（layout analysis）找出頁面的各個區域——段落、標題、清單、表格、圖片——通常是對頁面影像做物件偵測或實例分割。PubLayNet [28] 把超過一百萬篇 PubMed Central 文章的 XML 與 PDF 版本對起來，自動標註了超過 36 萬張頁面影像，共五個類別（文字、標題、清單、

表格、圖片）。DocLayNet [29] 有 80,863 頁來源多樣的人工標註頁面，11 個類別，採 COCO 格式，因為只用科學文章訓練的模型，轉移到報告、手冊與表單時表現很差。

有了區域與單字之後，系統還必須產生**閱讀順序**（reading order）。最直覺的規則「先由上到下，再由左到右」，連最簡單的多欄頁面都會失敗，下面的程式在一份兩欄的公告上示範。

```
import numpy as np

# Word boxes (x0, y0, x1, y1) on a two-column page: each column is one sentence.
left = "Pay the invoice within thirty days of receipt".split()
right = "Late payments carry a two percent monthly fee".split()
words, boxes = [], []
for col, text in [(0, left), (1, right)]:
    for k, w in enumerate(text):
        row, pos = divmod(k, 3)                      # three words per line
        x0 = 40 + 300 * col + 80 * pos
        y0 = 100 + 30 * row + 3 * col                # columns not perfectly aligned
        words.append(w); boxes.append([x0, y0, x0 + 70, y0 + 20])
boxes = np.array(boxes, float)

def naive_order(boxes, line_tol=10):
    """Top-to-bottom, then left-to-right, treating boxes within line_tol as one line."""
    line = np.round(boxes[:, 1] / line_tol)
    return np.lexsort((boxes[:, 0], line))

def column_order(boxes, line_tol=10):
    """Split the page at the widest empty vertical strip, then read column by column."""
    x_min, x_max = int(boxes[:, 0].min()), int(boxes[:, 2].max())
    covered = np.zeros(x_max - x_min, bool)
    for x0, _, x1, _ in boxes.astype(int):
        covered[x0 - x_min:x1 - x_min] = True
    edges = np.flatnonzero(np.diff(np.r_[1, covered.astype(int), 1]))  # empty-run bounds
    starts, ends = edges[::2], edges[1::2]
    widest = np.argmax(ends - starts)
    split = x_min + (starts[widest] + ends[widest]) / 2
    col = (boxes[:, 0] > split).astype(int)
    line = np.round(boxes[:, 1] / line_tol)
    return np.lexsort((boxes[:, 0], line, col))

print("naive :", " ".join(words[i] for i in naive_order(boxes)))
print("column:", " ".join(words[i] for i in column_order(boxes)))
```

```
naive : Pay the invoice Late payments carry within thirty days a two percent of receipt
 ↪  monthly fee
column: Pay the invoice within thirty days of receipt Late payments carry a two percent
 ↪  monthly fee
```

天真排序的輸出中，每個單字都是對的，文字卻毫無用處。欄規則修好了這個頁面，但它又是另一條人工規則，遇到表格或圖片就會失敗（圖 6）；學習式的版面與文件模型，很大程度上就是為了取代這類規則而出現。

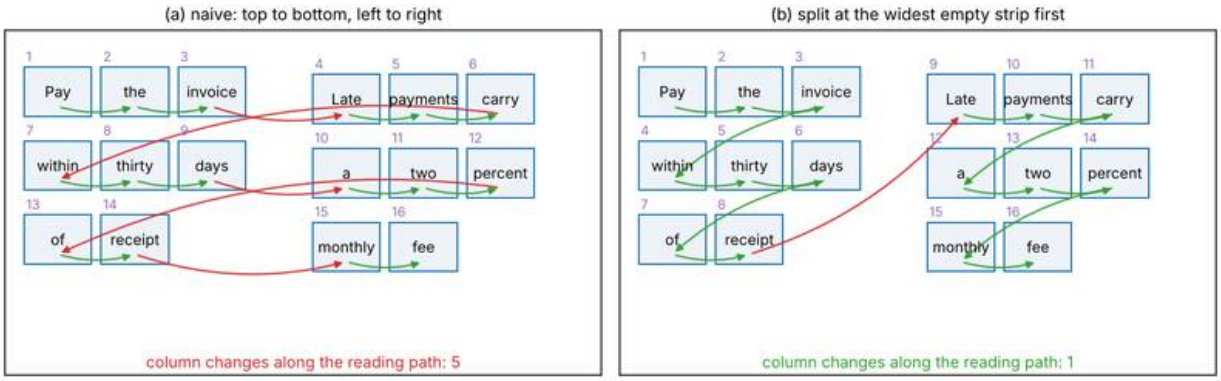


圖 *6* — 合成兩欄頁面上的閱讀順序。(*a*) 先依行再依 x 排序，會讓兩欄交錯（換欄 *5* 次）。(*b*) 先在最寬的空白直條處切開，得到預期的順序（換欄 *1* 次），但只適用於這類版面。

文字加版面：LayoutLM 家族

白話版。不只把頁面上的單字交給語言模型，也告訴它每個單字在哪裡。這樣，收據右下角跟在「Total」後面的數字，就和段落中間的同一個單字意義不同。

LayoutLM [30] 從 BERT（NLP 10）出發，在每個單字的符號嵌入上加一個二**維位置嵌入**。每個單字都帶著它的 OCR 框 (x_0, y_0, x_1, y_1)——左上角與右下角——並重新縮放成 0 到 1,000 的整數「虛擬」座標，讓不同大小的頁面共用同一個尺度。四個嵌入查表結果相加，其中 x_0 與 x_1 共用一張表，y_0 與 y_1 共用另一張：

$$\mathbf{h}_i = \mathbf{e}_{\text{tok}}(w_i) + \mathbf{e}_{\text{pos}}(i) + \mathbf{X}(x_{0,i}) + \mathbf{Y}(y_{0,i}) + \mathbf{X}(x_{1,i}) + \mathbf{Y}(y_{1,i}),$$

其中 $\mathbf{h}_i$ 是第 i 個單字 w_i 的輸入向量，$\mathbf{e}_{\text{tok}}$ 與 $\mathbf{e}_{\text{pos}}$ 是 BERT 的符號與一維位置嵌入，$\mathbf{X}$、$\mathbf{Y}$ 是水平與垂直座標學到的查表。預訓練時遮住一些單字但保留它們的二維位置，讓模型從文字與版面上下文預測它們（**遮罩視覺語言模型**，masked visual-language model），資料是約 1,100 萬頁用開源 Tesseract 引擎做 OCR 的掃描頁面 [30]、[31]。LayoutLM 在下游任務中還會加入每個單字影像區域的視覺特徵，由 Faster R-CNN 模型算出 [30]。LayoutLMv3 [32] 則把頁面影像切成圖塊直接輸入，同時遮住文字符號與影像圖塊，並加入單字–圖塊對齊的目標，使一個模型既能處理以文字為主的任務（表單、收據、問答），也能處理以影像為主的任務（文件分類、版面分析）。

關鍵資訊擷取（key information extraction）——從頁面中取出「發票號碼」、「日期」與「總額」——因此就是對 OCR 單字做符號分類，如 NLP 06 的序列標註，再加上把鍵連到值。FUNSD [33] 是這類的小型基準：199 張真實、有雜訊的掃描表單，語意實體標成問題、答案、標頭或其他，實體之間有連結。

這個家族的弱點在輸入：OCR 引擎漏掉或讀錯的字它無法修正，而且它繼承了引擎的閱讀順序。

免 OCR 模型：Donut 與 Pix2Struct

白話版。跳過 OCR 引擎。訓練一個模型看頁面影像，然後一個符號一個符號地寫出想要的輸出——完整文字、JSON 紀錄或答案——就像影像標題生成模型寫標題那樣。

Donut（Document understanding transformer）[34] 由一個 Swin Transformer 編碼器（ML 05）與一個 BART 解碼器組成。它先預訓練去按閱讀順序讀出整頁的所有文字——這是一項「偽 OCR」任務，資料是 1,100 萬份英文掃描文件，加上它的 SynthDoG 產生器產生的中文、日文、韓文與英文合成頁面——再微調成輸出一串符號，這串符號可以一對一轉成 JSON，用於分類、資訊擷取或問答 [34]。這避開了執行 OCR 的成本、OCR 引擎在不同語言與文件類型上的僵化，以及 OCR 錯誤的傳遞。**Pix2Struct** [35] 的預訓練任務是把被遮住的網頁截圖解析成簡化的 HTML，在一個目標裡同時混合了 OCR、語言建模與影像標題生成；它接受可變解析度的輸入，並把問題直接算繪在影像上。

代價是解析度與可驗證性。Donut 使用 2560 × 1920 的輸入，並報告在小字很多的文件上，準確率隨輸入尺寸快速上升，運算成本也隨之增加 [34]；而且因為模型是生成而不是抄寫，輸出的每個字元都無法對應到影像的某個區域。

僅線上版：*Donut 導讀* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/papers/donut-導讀/*

文件用的視覺語言模型

通用的多模態大型語言模型（ML 10）現在能回答關於文件的問題，也能抄寫頁面；讀小字要靠切塊或動態解析度（示意圖沒畫出的事）。有三點要注意。第一，流暢的回答不等於抄寫：要拿它和偵測結果核對，或要求它給座標再驗證座標。第二，生成式讀取器有自己的失效模式：針對學術文件的 Nougat [36] 報告它有時會退化成一直重複同一句話（在 1.5% 的測試頁面上，領域外更常見），而且參考文獻清單與章節標題中的數字，有時會被跳過或被幻覺出來。第三，要量測具體的能力：OCRBench [22] 蒐集了 29 個資料集，分別測試文字辨識、場景文字與文件問答、關鍵資訊擷取，以及手寫數學式，而不是把 OCR 藏在一個總分裡。

手寫、多種文字與中日韓文字

白話版。手寫因人而異，字母還會連在一起；許多文字系統的符號數量遠超過拉丁字母；有些，例如中文，還可以直排。這幾點都打破了英文場景文字模型默默做的某個假設。

手寫

離線手寫辨識是從掃描檔讀出手寫的文字行。手寫英文句子的 IAM 資料庫 [37]，用 TrOCR 作者的話來說，是手寫文字辨識最常用的資料集；TrOCR 使用 6,161 行訓練、966 行驗證與 2,915 行測試的切分，並報告區分大小寫的 CER [19]。行層級、用 CTC 或注意力的辨識器可以原封不動地使用；困難在於不同書寫者之間的差異、連筆的字母，以及有標註的資料有限，這也是合成資料預訓練和大型預訓練解碼器有幫助的原因。對於中文手寫，CASIA 線上與離線資料庫 [38] 同時提供筆跡軌跡（線上）與掃描影像（離線）辨識所需的資料。

大字元集與直排文字

中文與日文使用的字母表有幾萬個符號，而拉丁文字只有幾十個 [1]。RCTW-17 列出了中文文字的不同之處：字元集大得多、詞與詞之間沒有空格，以及許多字元由好幾個不相連的部分組成 [39]。每一點都有後果。辨識器的輸出層隨字母表變大，而且大多數字元都很罕見：Chinese Text in the Wild 資料集在 32,285 張街景影像中標註了 1,018,402 個字元實例，涵蓋 3,850 種不同字元，其中大多數很少用 [40]。不相連的部分會破壞連通元件分組。沒有空格，系統就以行與字元而不是單字來運作（見 NLP 01）。

直排文字是另一個困難：中文除了橫排，也可以直排，像 CTW 這類街景資料集就含有直排文字 [40]。W3C 的 *Requirements for Chinese Text Layout* 描述了直排的編排方式：字元由上到下、行由右到左，而夾在其中的拉丁單字，不是直立擺放，就是順時針旋轉 90° [41]。假設畫格由左到右的辨識器，若拿到橫放的裁切圖，會把這一欄讀成亂碼；Baek 等人把直排文字列為他們測試的全部 24 個辨識器共同的失敗案例之一 [7]。解法是先決定每條偵測到的文字行的方向，再旋轉或重排之後才去讀。

僅線上版：閱讀直排與混合方向的文字 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/notes/閱讀直排與混合方向的文字/*

同時有多種文字

多語言系統還必須判斷自己看到的是哪一種文字系統。ICDAR 2019 多語言挑戰賽（MLT-2019）[42] 有 20,000 張真實影像，含 10 種語言的文字，並包含文字系統辨識任務；這些語言涵蓋七種文字系統 [2]。

評估 OCR

白話版。像校對員那樣數錯誤：要插入、刪除或更改多少字元，才能把輸出變成正確答案。對偵測，則像物件偵測那樣數：找到的框有多少是對的，真實單字被找到多少。

由編輯距離算出 CER 與 WER

參考字串 r 與辨識字串 h 之間的**編輯距離**（edit distance，Levenshtein 距離）是把一個變成另一個所需的最少插入、刪除與替換次數；Wagner–Fischer 動態規劃可以算出它 [43]（推導見 PR 06）。沿著最佳對齊回溯整張表，就能把總數拆成 S 次替換、D 次刪除與 I 次插入。於是

$$\mathrm{CER} = \frac{S + D + I}{N}, \qquad \mathrm{WER} = \frac{S_w + D_w + I_w}{N_w},$$

其中 N 是參考字串的字元數，S_w、D_w、I_w 與 N_w 是以單字（以空格切開的序列）計算的同樣數量。兩者都是錯誤率：0 表示完美，輸出含有大量插入時可以超過 1。還有其他慣例：對裁切單字的場景文字辨識，通常報告**單字準確率**（word accuracy，完全讀對的裁切圖比例），有些論文把 1 減去它稱為詞錯誤率，而多語言競賽報告的是 1 減去**正規化編輯距離**（normalized edit distance，NED）[2]。務必說明你報告的是哪一種。

```
import numpy as np

def edit_ops(ref, hyp):
    """Levenshtein alignment of two sequences; returns (S, D, I) counts."""
    n, m = len(ref), len(hyp)
    E = np.zeros((n + 1, m + 1), int)
    E[:, 0], E[0, :] = np.arange(n + 1), np.arange(m + 1)
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            E[i, j] = min(E[i - 1, j] + 1, E[i, j - 1] + 1,
                          E[i - 1, j - 1] + (ref[i - 1] != hyp[j - 1]))
    S = D = I = 0
    i, j = n, m
    while i > 0 or j > 0:                       # walk back along one optimal path
        if i > 0 and j > 0 and E[i, j] == E[i - 1, j - 1] + (ref[i - 1] != hyp[j - 1]):
            S += ref[i - 1] != hyp[j - 1]; i, j = i - 1, j - 1
        elif i > 0 and E[i, j] == E[i - 1, j] + 1:
            D += 1; i -= 1
        else:
            I += 1; j -= 1
    return S, D, I

def rates(ref, hyp):
    c = edit_ops(ref, hyp)
    w = edit_ops(ref.split(), hyp.split())
    return c, sum(c) / len(ref), w, sum(w) / len(ref.split())

ref = "Total due 25.80 EUR by 1 May"
for hyp in ["Total due 25.80 EUR by 1 May",
            "Tota1 due 25.8O EUR by l May",      # three look-alike glyphs
            "Total due25.80 EUR by 1 May",       # one missing space
            "Total due 25.80 EUR"]:              # last words not read
    c, cer, w, wer = rates(ref, hyp)
    print(f"{hyp:30s} chars S,D,I={c} CER={cer:.3f} | words S,D,I={w} WER={wer:.3f}")
```

```
Total due 25.80 EUR by 1 May   chars S,D,I=(0, 0, 0) CER=0.000 | words S,D,I=(0, 0, 0)
 ↪  WER=0.000
Tota1 due 25.8O EUR by l May   chars S,D,I=(3, 0, 0) CER=0.107 | words S,D,I=(3, 0, 0)
 ↪  WER=0.429
Total due25.80 EUR by 1 May    chars S,D,I=(0, 1, 0) CER=0.036 | words S,D,I=(1, 1, 0)
 ↪  WER=0.286
Total due 25.80 EUR            chars S,D,I=(0, 9, 0) CER=0.321 | words S,D,I=(0, 3, 0)
 ↪  WER=0.429
```

三個長得像的替換，花掉 28 個字元中的 3 個（CER 0.107），卻毀了 7 個單字中的 3 個（WER 0.429），而且其中一個還把金額變成非數字。少一個空格只花一個字元，卻造成兩個單字錯誤；丟掉最後三個單字，則花了九個字元，含空格。CER 告訴你要修多少；WER，或者更好的是像「總額對不對？」這樣的任務檢查，才告訴你輸出能不能用（圖 7）。

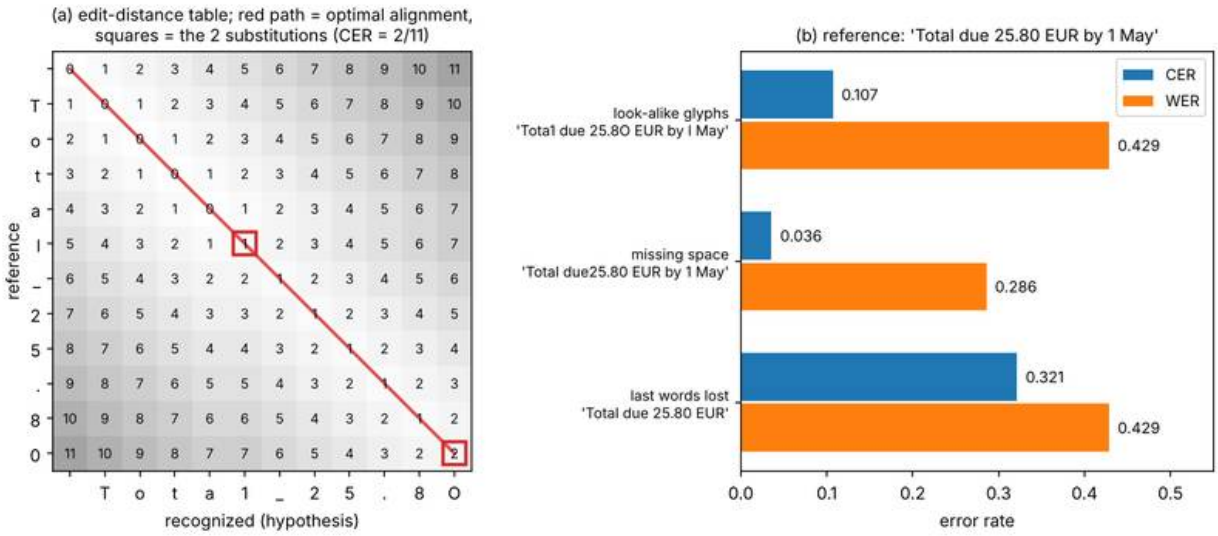


圖 7 — *CER* 與 *WER*。(*a*) 一對短字串的編輯距離表；紅色路徑是一條含兩次替換的最佳對齊，所以 $CER = 2/11$。(*b*) 對同一個參考字串，很小的字元錯誤也可能造成很大的詞錯誤率。

偵測：IoU 0.5 下的精確率、召回率與 H-mean

文字偵測使用 DLCV 03 的精確率與召回率，只是有一條配對規則和一個彙總數字。在 Robust Reading 競賽使用的協定中（此處依 MLT-2019 的描述），偵測若與某個真值四邊形的 IoU 超過 0.5，就是真陽性；每個真值與每個偵測最多只能配對一次；而與標為「不在乎」（do not care，無法辨認或超出範圍的文字）的區域重疊超過該區域面積一半的偵測，在配對前會被移除 [42]。計數是在整個測試集上累加的，而且

$$P = \frac{\mathrm{TP}}{|\mathcal{D}|}, \qquad R = \frac{\mathrm{TP}}{|\mathcal{G}|}, \qquad H = \frac{2PR}{P + R},$$

其中 TP 是配對成功的組數，$|\mathcal{D}|$ 是過濾後的偵測數，$|\mathcal{G}|$ 是不屬於「不在乎」的真值單字數，H 是 **H-mean**（精確率與召回率的調和平均，即 F-score）。和 COCO AP 不同，H-mean 是精確率–召回率曲線上的單一點，位於提交者自己選的信心閾值；Long 等人建議同時報告 AP [1]。ICDAR 2013 用的是不同的、以重疊為基礎的協定，所以它的數字不能和以 IoU 為基礎的數字比較 [1]。

端到端與文件指標

端到端文字讀取先如上所述把偵測配對到真值，再比較轉錄內容，可以對每個單字比（**端到端辨識**），也可以只對給定詞彙中的單字比（**單字定位**，word spotting）；結果強烈依賴那份詞彙的大小 [2]。文件基準使用任務指標：**ANLS**（average normalized Levenshtein similarity，平均正規化 Levenshtein 相似度），這是用於 DocVQA 答案的、以編輯距離為基礎的分數，設計成不會把細小的 OCR 失誤當成完全失敗 [44]；對擷取出的 JSON 紀錄，則用欄位層級的 F1 與以樹編輯距離為基礎的準確率 [34]；版面區域則用 COCO 式的 mAP [29]。

僅線上版：評估協定的細節 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/notes/評估協定的細節/*

資料集與基準

下表列出本篇提到的基準資料集。大小依引用的論文所述。

基準	任務	內容
ICDAR 2013 [5]	偵測、辨識、端到端	對焦的場景文字，大多是水平的英文；462 張影像 [10]
ICDAR 2015 Incidental [6]	偵測、辨識、端到端	穿戴式相機影像；偵測用 1,000 張訓練／500 張測試 [12]；四邊形標註
Total-Text [10]	偵測、文字讀取	1,555 張影像（1,255 ／ 300）、9,330 個單字、多邊形；水平、多方向與彎曲
CTW1500 [11]	彎曲文字偵測	1,500 張影像（1,000 ／ 500）、超過 10,000 個多邊形文字標註
IIIT5K [45]	裁切單字辨識	取自網路影像搜尋的單字；2,000 訓練／3,000 測試 [7]
SVT [46]	裁切單字辨識	Google 街景；257 訓練／647 測試單字 [7]
MLT-2019 [42]	多語言偵測、文字系統辨識、端到端	20,000 張影像、10 種語言
RCTW-17 [39]、CTW [40]	中文偵測與辨識	12,263 張影像；32,285 張有字元層級標註的影像
IAM [37]、CASIA-HWDB [38]	手寫	英文句子；中文線上與離線手寫
FUNSD [33]	表單理解	199 張掃描表單；問題／答案／標頭／其他實體與連結
PubLayNet [28]、DocLayNet [29]	版面分析	超過 36 萬頁自動標註的文章頁面（5 類）；80,863 頁人工標註頁面（11 類）
DocVQA [44]	文件問答	超過 12,000 張文件影像上的 50,000 個問題；ANLS
OCRBench [22]、OmniDocBench [47]	多模態模型的 OCR 能力；頁面解析	29 個資料集；9 種文件來源、19 種版面類別

所有基準都有兩點要注意。不同論文曾在同一基準的不同子集上評估（Baek 等人發現 IC03 與 IC15 各有兩個版本在使用），所以比較數字之前要先檢查子集 [7]。而且標籤並不完美：Baek 等人估計，常用辨識基準的標註錯誤率，在忽略特殊字元時是 1.3%，把特殊字元算進去時升到 6.1%，連大小寫也算進去時是 24.1% [7]。

失效模式

白話版。 OCR 會在像素不再帶有答案時失敗，在形狀或文字系統和訓練時不同時失敗，也會在模型用看似合理的東西填補空缺時失敗。最後一種最危險，因為輸出看起來很完美。

長得像的字形與低解析度。許多字元只差幾個像素：0 與 O、1、l 與 I、5 與 S、8 與 B、「rn」與「m」。下面的程式為 15 個字元做了一個小小的範本讀取器，並顯示每個字元的像素數下降時，它如何退化。

```
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from skimage.transform import resize

CHARS = "0O8B5S1lIZ2aekx"

def glyph(c):
    """Render one character (ink = 1, paper = 0) on a 48 x 48 canvas."""
    fig = plt.figure(figsize=(0.48, 0.48), dpi=100)
    fig.text(0.5, 0.5, c, fontsize=26, ha="center", va="center")
    fig.canvas.draw()
    img = 1 - np.asarray(fig.canvas.buffer_rgba())[..., :3].mean(axis=2) / 255.0
    plt.close(fig)
    return img

def ncc(a, b):
    a, b = a - a.mean(), b - b.mean()
    return (a * b).sum() / np.sqrt((a * a).sum() * (b * b).sum())

templates = {c: glyph(c) for c in CHARS}
rng = np.random.default_rng(0)
for h in [48, 12, 8, 6]:                          # pixels per character cell
    errors, total = {}, 0
    for c, img in templates.items():
        for _ in range(20):
            low = resize(img, (h, h), anti_aliasing=True)       # camera sees few pixels
            low = low + rng.normal(0, 0.05, low.shape)          # sensor noise
            back = resize(low, (48, 48), order=1)               # what the reader gets
            guess = max(CHARS, key=lambda t: ncc(back, templates[t]))
            total += 1
            if guess != c:
                errors[(c, guess)] = errors.get((c, guess), 0) + 1
    top = sorted(errors.items(), key=lambda kv: -kv[1])[:3]
    print(f"{h:2d}px  accuracy={1 - sum(errors.values()) / total:.2f}  most confused: "
          + ", ".join(f"{a}->{b} ({n})" for (a, b), n in top))
```

```
48px  accuracy=1.00  most confused:
12px  accuracy=0.93  most confused: l->I (20)
 8px  accuracy=0.84  most confused: I->1 (13), l->1 (12), 0->O (6)
 6px  accuracy=0.53  most confused: B->8 (20), 0->8 (15), S->8 (12)
```

小寫「l」與大寫「I」在 12 像素時就已經混淆，因為這個無襯線字體把兩者都畫成一根單純的直條（圖 8）；到 6 像素時，有一半的讀法是錯的。深度辨識器遠比範本比對好，但沒有任何辨識器能分開字體畫得一模一樣的字形；只有上下文能。超解析度可能有幫助（DLCV 09A），但請看下面的最後一項。

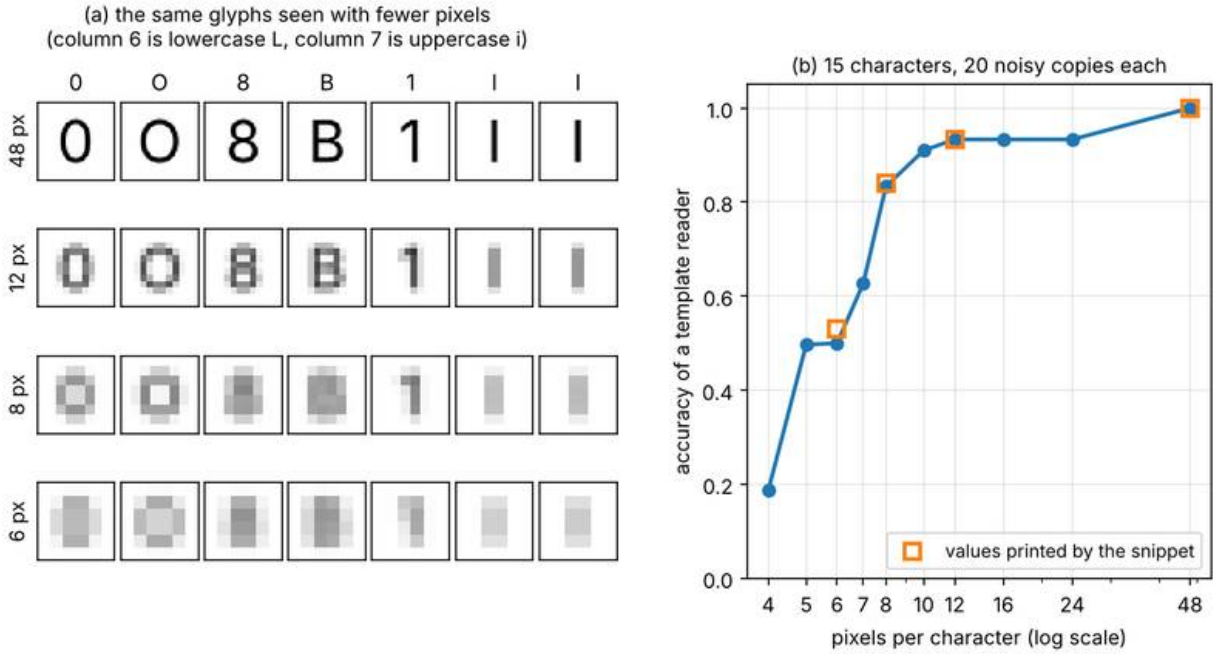


圖 *8* — 低解析度下長得像的字形（玩具範本讀取器，*15* 個字元，每個 *20* 份帶雜訊的副本）。(*a*) 同樣的字形，像素越來越少。(*b*) 準確率對解析度作圖；方塊標出程式碼印出的數值。

其他常見的失敗，各自標出它從哪個階段開始（圖 1）：

- **彎曲、旋轉與有透視的文字**（偵測與辨識）：正立框會漏掉它們（圖 3）；水平的辨識器除非先矯正，否則會讀錯。
- **特殊字體與風格**（辨識）：書法體的品牌名稱與店家招牌，是 Baek 等人研究中全部 24 個模型都失敗的案例之一，同樣的還有直排文字、特殊字元、嚴重遮擋與低解析度 [7]。
- **實例被合併或被拆開**（偵測）：圖 2 的間隙取捨的學習版；分割式偵測器用縮小的核心與學到的閾值來對付它。
- **版面錯誤**（理解）：字對但順序錯（圖 6）、表格被橫著讀、數值接到錯的鍵上。
- **分布偏移**（所有階段）：只用文章頁面訓練的版面模型，在較多樣的文件上會退化 [29]；DLCV 14 討論一般性的偏移。
- **幻覺文字**（任何生成階段）：生成式讀取器可能輸出影像裡沒有的文字——重複的句子、捏造的章節編號 [36]、被「修正」過的序號。上游也會發生同樣的事：生成式復原或超解析度模型，可能把看不清的字母變成乾淨、清楚卻錯誤的字母，如 DLCV 09B 所討論的；生成影像中的文字也常常拼錯（DLCV 12）。凡是必須正確的資訊，例如金額、姓名或車牌，不要對經過生成式增強的影像跑 OCR，除非拿原圖核對。

現代觀點

綜述怎麼說

有四篇回顧涵蓋了本篇的範圍。**Long 等人** [1] 依一個預測涵蓋單字的多少來整理場景文字偵測（整個單字，相對於字元、片段或核心），並把辨識分成以 CTC 為基礎與編碼器–解碼器兩類方法；他們認為跨資料集的泛化與大字母表是尚未解決的問題。**Chen 等人** [2] 專注於辨識，依管線階段組織，並整理了資料集與評估協定。**Cui 等人** [3] 回顧從規則式分析到預訓練 LayoutLM 式模型的文件任務；**Zhang 等人** [4] 把文件解析分成模組化管線與統一的 VLM 式模型，並指出複雜版面、VLM 解析的可靠度與推論成本是主要挑戰。OCRBench [22] 補充了這些回顧，評估大型多模態模型在文字相關任務上的表現。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/text-in-images/papers/現代觀點綜述導讀/*

2023–2026 年的最新進展

- **管線仍是自然的選擇**，當座標、速度或可稽核性重要時：DBNet 式的偵測器、注意力或 Transformer 辨識器，加上一個版面模型，各自獨立評估與修正。
- **端到端的頁面讀取器正在快速成熟**。一次就把頁面變成 Markdown 或結構化文字的模型，現在已涵蓋公式、表格與圖表：針對學術論文的 Nougat [36]；GOT 把純文字、公式、表格、圖表與樂譜都當成「字元」，用一個編碼器–解碼器模型處理 [48]；olmOCR 是一個微調過的視覺語言模型，用來大規模線性化 PDF，並附帶自己的困難頁面基準 [49]；以及 DeepSeek-OCR，它研究一頁最少可以壓縮成多少個視覺符號、同時仍然可讀 [50]。這些報告全都來自作者自己。
- **評估正在迎頭趕上**。 OmniDocBench [47]（CVPR 2025）在來自九種文件來源的頁面上評估管線與端到端模型，有 19 個版面類別與 15 個屬性標籤，所以結果可以按文件類型與難度拆開看，而不是只有一個分數。

尚待解決的問題

1. **忠實度**。生成式讀取器可能產生頁面上沒有的文字，而且它們的錯誤看起來像正確的文字。字元層級的接地（每個輸出符號由哪些像素支持）與校準過的信心，還不是標準配備。
2. **非語意與罕見文字**。代碼、數字、人名、大字母表中的罕見字元與混合文字，正是語言先驗幫助最小、錯誤代價最大的地方 [22]。
3. **複雜頁面上的版面與閱讀順序**——有浮動圖片的多欄文章、巢狀表格、含手寫欄位的表單——對管線與端到端模型來說都依然困難；對複雜版面的穩健性是 Zhang 等人 [4] 點名的主要挑戰之一。
4. **跨領域與跨文字系統的泛化**，包含直排與彎曲文字，並且不必為每個領域蒐集新的標籤 [1]。
5. **可比較的評估**。不同的子集、指標（CER、單字準確率、NED、ANLS）與輸出正規化，讓已發表的數字很難比較 [7]。

重點整理

- 分別評估閱讀流程各階段與完整管線，上游錯誤會向後傳遞。
- 旋轉、彎曲與密集文字，需要一般直立方框未必能表達的幾何。

- 縮小文字區域有助分開相接實例，門檻化與區域擴張則影響最終偵測。
- 辨識是序列預測；解碼與語言先驗有助自然詞彙，也可能扭曲代碼或數字。
- 版面與 OCR-free 模型改變錯誤傳遞路徑，解析度限制與虛構文字仍須檢查。
- 註明 CER ／ WER 版本、偵測匹配規則、忽略區域與下游任務指標。

練習

1. **間隙與字體大小**。在偵測程式中，把 `WORDS` 裡每個字體大小先乘以 0.6、再乘以 1.6，重新跑膨脹間隙的掃描。最佳間隙怎麼移動？這對於一張同時含有兩種大小的影像、只用一條固定的後處理規則意味著什麼？

   提示

   最佳間隙會隨字母間距、也就是隨字體大小縮放，所以同時含有兩種大小的影像，沒有單一的好間隙。學習式偵測器直接預測單字區域或縮小的核心，就避開了這個問題。

2. **手算正立框**。證明旋轉 θ 的 $w \times h$ 矩形與其最緊貼的正立框之間的 IoU 是 $wh/\big[(w\cos\theta + h\sin\theta)(w\sin\theta + h\cos\theta)\big]$。對 $w = 160$、$h = 24$，求出它降到 0.5 的角度，並與正方形單字（$w = h$）比較。

   提示

   旋轉後的矩形完全在它的正立框內，所以交集是 wh，聯集就是框的面積。對 $w/h = 160/24$ 數值求解，得到約 8.5°。對正方形，45° 時的最小值恰好是 $1/2$，所以正方形永遠不會低於 0.5：這個問題是細長形狀（如文字行）特有的。

3. **貪婪與精確的 CTC**。把 CTC 矩陣最後兩列改成：兩個畫格中空白與「t」的機率都是 0.65 與 0.30。「cat」還是最可能的文字嗎？接著讓第 4、5 畫格中「t」的機率 p_t 從 0.05 變到 0.45，空白佔 $0.95 - p_t$。在哪些值上，貪婪與精確解碼的結果一致？

   提示

   重用暴力迴圈。在 $p_t = 0.30$ 時，「cat」（約 0.294）仍勝過「ca」（約 0.249）。貪婪解碼在 p_t 超過空白（大於 0.475）之前都回傳「ca」，但精確答案在 $p_t \approx 0.27$ 時就已經換成「cat」；兩個值之間兩者不一致。低於交叉點時，兩者都得到「ca」。

4. **藏著災難的 CER**。構造一個 40 個字元的參考字串，以及兩個 CER 同為 0.05 的辨識結果，其中一個對發票處理系統無害，另一個卻是致命的。提出一個能區分兩者的欄位層級指標。

   提示

   一種是在很長的品項描述單字裡錯兩個字（「Paper A4 recyled」），另一種是在總額裡錯兩個字（「42.00」讀成「4Z.0O」）。對關鍵欄位（發票號碼、日期、總額）做欄位層級的完全比對，或是 Donut 報告的、對擷取欄位計算的 F1，都能區分這兩者；CER 與 WER 對每個字元或單字一視同仁。

5. **閱讀順序又壞了。**在閱讀順序程式的兩欄公告上方，加一個橫跨兩欄的標題「PAYMENT TERMS」。`column_order` 會產生什麼？修改它，讓橫跨整個寬度的行先被讀，並描述一種仍然能打敗你的規則的版面。

提示

標題跨過了分割線，並依其左邊緣被分到左欄；放在最上面剛好沒問題，但放在中間的整行寬就不行了。把與分割線重疊的框當成獨立的一條帶，一條帶一條帶地讀。表格，或在某一欄內的圖片，仍然會打破以空白直條為基礎的規則。

參考文獻

1. S. Long, X. He and C. Yao, “Scene text detection and recognition: The deep learning era,” *International Journal of Computer Vision*, vol. 129, pp. 161–184, 2021. doi
2. X. Chen, L. Jin, Y. Zhu, C. Luo and T. Wang, “Text recognition in the wild: A survey,” *ACM Computing Surveys*, vol. 54, no. 2, 2021. doi
3. L. Cui, Y. Xu, T. Lv and F. Wei, “Document AI: Benchmarks, models and applications,” arXiv: 2111.08609, 2021. arXiv
4. Q. Zhang, B. Wang, V. S.-J. Huang, J. Zhang, Z. Wang, H. Liang, C. He and W. Zhang, “Document parsing unveiled: Techniques, challenges, and prospects for structured information extraction,” arXiv:2410.21169, 2024. arXiv
5. D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. Gomez i Bigorda, S. Robles Mestre, J. Mas et al., “ICDAR 2013 Robust Reading Competition,” in *Proc. International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1484–1493, 2013. doi
6. D. Karatzas, L. Gomez-Bigorda, A. Nicolaou, S. Ghosh, A. Bagdanov, M. Iwamura, J. Matas et al., “ICDAR 2015 competition on Robust Reading,” in *Proc. International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1156–1160, 2015. doi
7. J. Baek, G. Kim, J. Lee, S. Park, D. Han, S. Yun, S. J. Oh and H. Lee, “What is wrong with scene text recognition model comparisons? Dataset and model analysis,” in *Proc. IEEE/CVF ICCV*, pp. 4714–4722, 2019. doi
8. N. Otsu, “A threshold selection method from gray-level histograms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. doi
9. J. Sauvola and M. Pietikäinen, “Adaptive document image binarization,” *Pattern Recognition*, vol. 33, no. 2, pp. 225–236, 2000. doi
10. C. K. Ch’ng and C. S. Chan, “Total-Text: A comprehensive dataset for scene text detection and recognition,” in *Proc. ICDAR*, pp. 935–942, 2017. doi
11. Y. Liu, L. Jin, S. Zhang, C. Luo and S. Zhang, “Curved scene text detection via transverse and longitudinal sequence connection,” *Pattern Recognition*, vol. 90, pp. 337–345, 2019. doi
12. X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He and J. Liang, “EAST: An efficient and accurate scene text detector,” in *Proc. IEEE CVPR*, pp. 2642–2651, 2017. doi
13. W. Wang, E. Xie, X. Li, W. Hou, T. Lu, G. Yu and S. Shao, “Shape robust text detection with progressive scale expansion network,” in *Proc. IEEE/CVF CVPR*, pp. 9328–9337, 2019. doi

14. M. Liao, Z. Wan, C. Yao, K. Chen and X. Bai, “Real-time scene text detection with differentiable binarization,” in *Proc. AAAI Conference on Artificial Intelligence*, vol. 34, pp. 11474–11481, 2020. doi
15. Y. Baek, B. Lee, D. Han, S. Yun and H. Lee, “Character region awareness for text detection,” in *Proc. IEEE/CVF CVPR*, pp. 9357–9366, 2019. doi
16. B. Shi, X. Bai and C. Yao, “An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition,” *IEEE TPAMI*, vol. 39, no. 11, pp. 2298–2304, 2017. doi
17. A. Graves, S. Fernández, F. Gomez and J. Schmidhuber, “Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks,” in *Proc. ICML*, pp. 369–376, 2006. doi
18. B. Shi, M. Yang, X. Wang, P. Lyu, C. Yao and X. Bai, “ASTER: An attentional scene text recognizer with flexible rectification,” *IEEE TPAMI*, vol. 41, no. 9, pp. 2035–2048, 2019. doi
19. M. Li, T. Lv, J. Chen, L. Cui, Y. Lu, D. Florencio, C. Zhang, Z. Li and F. Wei, “TrOCR: Transformer-based optical character recognition with pre-trained models,” in *Proc. AAAI Conference on Artificial Intelligence*, vol. 37, pp. 13094–13102, 2023. doi
20. D. Bautista and R. Atienza, “Scene text recognition with permuted autoregressive sequence models,” in *Proc. ECCV*, pp. 178–196, 2022. doi
21. S. Fang, H. Xie, Y. Wang, Z. Mao and Y. Zhang, “Read like humans: Autonomous, bidirectional and iterative language modeling for scene text recognition,” in *Proc. IEEE/CVF CVPR*, pp. 7094–7103, 2021. doi
22. Y. Liu, Z. Li, M. Huang, B. Yang, W. Yu, C. Li, X.-C. Yin, C.-L. Liu, L. Jin and X. Bai, “OCRBench: On the hidden mystery of OCR in large multimodal models,” *Science China Information Sciences*, vol. 67, 2024. doi
23. M. Jaderberg, K. Simonyan, A. Vedaldi and A. Zisserman, “Synthetic data and artificial neural networks for natural scene text recognition,” arXiv:1406.2227, 2014. arXiv
24. A. Gupta, A. Vedaldi and A. Zisserman, “Synthetic data for text localisation in natural images,” in *Proc. IEEE CVPR*, pp. 2315–2324, 2016. doi
25. X. Liu, D. Liang, S. Yan, D. Chen, Y. Qiao and J. Yan, “FOTS: Fast oriented text spotting with a unified network,” in *Proc. IEEE/CVF CVPR*, pp. 5676–5685, 2018. doi
26. P. Lyu, M. Liao, C. Yao, W. Wu and X. Bai, “Mask TextSpotter: An end-to-end trainable neural network for spotting text with arbitrary shapes,” in *Proc. ECCV*, pp. 71–88, 2018. doi
27. Y. Liu, H. Chen, C. Shen, T. He, L. Jin and L. Wang, “ABCNet: Real-time scene text spotting with adaptive Bezier-curve network,” in *Proc. IEEE/CVF CVPR*, pp. 9806–9815, 2020. doi
28. X. Zhong, J. Tang and A. Jimeno Yepes, “PubLayNet: Largest dataset ever for document layout analysis,” in *Proc. ICDAR*, pp. 1015–1022, 2019. doi
29. B. Pfitzmann, C. Auer, M. Dolfi, A. S. Nassar and P. Staar, “DocLayNet: A large human-annotated dataset for document-layout segmentation,” in *Proc. ACM SIGKDD*, pp. 3743–3751, 2022. doi
30. Y. Xu, M. Li, L. Cui, S. Huang, F. Wei and M. Zhou, “LayoutLM: Pre-training of text and layout for document image understanding,” in *Proc. ACM SIGKDD*, pp. 1192–1200, 2020. doi
31. R. Smith, “An overview of the Tesseract OCR engine,” in *Proc. ICDAR*, vol. 2, pp. 629–633, 2007. doi
32. Y. Huang, T. Lv, L. Cui, Y. Lu and F. Wei, “LayoutLMv3: Pre-training for document AI with

unified text and image masking,” in *Proc. ACM International Conference on Multimedia*, pp. 4083–4091, 2022. doi
33. G. Jaume, H. K. Ekenel and J.-P. Thiran, “FUNSD: A dataset for form understanding in noisy scanned documents,” in *Proc. ICDAR Workshops (OST)*, 2019. doi
34. G. Kim, T. Hong, M. Yim, J. Nam, J. Park, J. Yim, W. Hwang, S. Yun, D. Han and S. Park, “OCR-free document understanding transformer,” in *Proc. ECCV*, pp. 498–517, 2022. doi
35. K. Lee, M. Joshi, I. R. Turc, H. Hu, F. Liu, J. M. Eisenschlos, U. Khandelwal, P. Shaw, M.-W. Chang and K. Toutanova, “Pix2Struct: Screenshot parsing as pretraining for visual language understanding,” in *Proc. ICML*, PMLR vol. 202, pp. 18893–18912, 2023. PMLR
36. L. Blecher, G. Cucurull, T. Scialom and R. Stojnic, “Nougat: Neural optical understanding for academic documents,” in *Proc. ICLR*, 2024. OpenReview ·arXiv
37. U.-V. Marti and H. Bunke, “The IAM-database: An English sentence database for offline handwriting recognition,” *International Journal on Document Analysis and Recognition*, vol. 5, pp. 39–46, 2002. doi
38. C.-L. Liu, F. Yin, D.-H. Wang and Q.-F. Wang, “CASIA online and offline Chinese handwriting databases,” in *Proc. ICDAR*, pp. 37–41, 2011. doi
39. B. Shi, C. Yao, M. Liao, M. Yang, P. Xu, L. Cui, S. Belongie, S. Lu and X. Bai, “ICDAR2017 competition on reading Chinese text in the wild (RCTW-17),” in *Proc. ICDAR*, pp. 1429–1434, 2017. doi ·arXiv
40. T.-L. Yuan, Z. Zhu, K. Xu, C.-J. Li, T.-J. Mu and S.-M. Hu, “A large Chinese text dataset in the wild,” *Journal of Computer Science and Technology*, vol. 34, pp. 509–521, 2019. doi ·arXiv
41. F. Xue and R. Ishida (eds.), “Requirements for Chinese Text Layout,” W3C Group Note (draft of 7 October 2026). W3C
42. N. Nayef, Y. Patel, M. Busta, P. N. Chowdhury, D. Karatzas, W. Khlif, J. Matas et al., “ICDAR2019 Robust Reading Challenge on multi-lingual scene text detection and recognition — RRC-MLT-2019,” in *Proc. ICDAR*, pp. 1582–1587, 2019. doi ·arXiv
43. R. A. Wagner and M. J. Fischer, “The string-to-string correction problem,” *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974. doi
44. M. Mathew, D. Karatzas and C. V. Jawahar, “DocVQA: A dataset for VQA on document images,” in *Proc. IEEE/CVF WACV*, pp. 2199–2208, 2021. doi
45. A. Mishra, K. Alahari and C. V. Jawahar, “Scene text recognition using higher order language priors,” in *Proc. BMVC*, 2012. doi
46. K. Wang, B. Babenko and S. Belongie, “End-to-end scene text recognition,” in *Proc. IEEE ICCV*, pp. 1457–1464, 2011. doi
47. L. Ouyang, Y. Qu, H. Zhou, J. Zhu, R. Zhang, Q. Lin, B. Wang, Z. Zhao et al., “OmniDocBench: Benchmarking diverse PDF document parsing with comprehensive annotations,” in *Proc. IEEE/CVF CVPR*, pp. 24838–24848, 2025. CVF
48. H. Wei, C. Liu, J. Chen, J. Wang, L. Kong, Y. Xu, Z. Ge, L. Zhao et al., “General OCR theory: Towards OCR-2.0 via a unified end-to-end model,” arXiv:2409.01704, 2024. arXiv
49. J. Poznanski, A. Rangapur, J. Borchardt, J. Dunkelberger, R. Huff, D. Lin, C. Wilhelm et al., “olmOCR: Unlocking trillions of tokens in PDFs with vision language models,” arXiv:2502.18443, 2025. arXiv
50. H. Wei, Y. Sun and Y. Li, “DeepSeek-OCR: Contexts optical compression,” arXiv:2510.18234, 2025. arXiv

51. C. Wolf and J.-M. Jolion, “Object count/area graphs for the evaluation of object detection and segmentation algorithms,” *International Journal of Document Analysis and Recognition*, vol. 8, pp. 280–296, 2006. doi

DLCV 07・物件追蹤：從單一物件到多物件

電腦視覺如何在影片中跟住物體：重識別、從相關濾波器到 Transformer 與 SAM 2 的單物件追蹤，以及結合卡爾曼濾波器、匈牙利演算法、端到端模型與 MOTA ／ IDF1 ／ HOTA 指標的多物件追蹤。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-tracking/

你將學到

- 區分運動估計與身分關聯。
- 區分單物件、多物件、多相機追蹤與再辨識。
- 比較樣板匹配、先偵測再追蹤與學習式查詢方法。
- 選擇能分別揭露偵測與身分錯誤的指標。
- 執行小型追蹤器，診斷漏偵測、軌跡斷裂與身分切換。

先看全貌

偵測器看一張圖，告訴你「這裡有三個人」。追蹤器看一段影片，告訴你「這個人和一秒前那個是**同一個**，而那一位剛走到柱子後面，等一下會再出來」。追蹤就是在偵測之上加入時間與記憶。

> **用最簡單的話說**
>
> 想像你和朋友一起看足球賽，各自挑一位最喜歡的球員。在一張照片裡找出場上所有球員，是**偵測**。眼睛一直盯著**你的**球員，看她跑、轉身、被別人擋住一下又出現，是**單物件追蹤**。幫**每一位**球員都貼上號碼貼紙，而且就算兩個人撞在一起，貼紙也絕對不會貼錯，是**多物件追蹤**。電視切到另一台攝影機時，你還認得出你的球員，就是**重識別**。

為什麼重要：自駕車要靠追蹤才知道行人是**正走向**馬路，而不只是站在路邊；運動分析要靠追蹤才算得出每位球員跑了多遠；倉儲要靠追蹤來清點堆高機和機器人；影片剪輯要靠追蹤讓遮罩黏在移動的物體上。這些應用不只要知道東西**在哪裡**，還要知道**誰是誰**、隨時間怎麼變化。

> **關於本篇**
>
> 本篇由先前在 Medium 發表的文章〈物件追蹤 Object Tracking 簡介〉[1] 擴充而成，保留原文架構，並補上技術細節、可執行的追蹤器與最新的文獻回顧。原文請見這裡。

相關篇章。 DLCV 05 介紹人體姿態與關鍵點估計；追蹤器常把姿態當作不依賴外觀的線索；DLCV 08 則辨識人在一段時間內做了什麼（動作辨識）。

從分類、偵測到追蹤

白話版。分類回答「圖裡**是什麼**」；偵測回答「是什麼、**在哪裡**」；追蹤回答「是什麼、在哪裡、**是哪一個**，而且隨時間持續」。

[1] 的這條路線，正好是電腦視覺任務的一張地圖：

1. **影像分類**把影像 I 對應到一個標籤 y。
2. **物件偵測**把影像對應到一組帶有類別與信心分數的框，$\mathcal{D} = \{(b_j, c_j, s_j)\}_{j=1}^{N}$，其中 b_j 是邊界框，c_j 是類別，$s_j \in [0,1]$ 是分數。偵測模仿的是人類視覺判斷畫面中目標位置的能力，詳細內容請見 DLCV 03・物件偵測。
3. **物件追蹤**輸入影格序列 $I_1, \dots, I_T$，輸出**軌跡**：對每個物件 k，給出它在可見影格集合 $\mathcal{T}_k$ 上的一串框（或遮罩）$(b_t^k)_{t \in \mathcal{T}_k}$，而且全部帶著同一個身分 k。

新加入的關鍵元素是身分標籤。偵測是無狀態的：第 t 格與第 $t+1$ 格各自獨立處理。追蹤則是有狀態的：它要把資訊帶到下一格。

追蹤在做什麼？

白話版。追蹤器預測運動，並在外觀改變與短暫遮擋下維持身分。

更精確地說，每個追蹤器都在解兩個互相耦合的子問題：

- **隨時間定位**（運動感知）：估計每個目標在每一格中的位置，最好能用過去預測未來。這讓追蹤器能撐過模糊、漏偵測與短暫遮擋。
- **維持身分**（資料關聯，data association）：決定第 t 格的哪個觀測屬於第 $t-1$ 格的哪個目標。這能防止兩個交錯而過的人互換標籤。

單物件追蹤因為只有一個身分，主要力氣花在第一個問題；多物件追蹤主要在處理第二個問題；重識別則是**完全沒有**運動資訊時的第二個問題。

[1] 對各類方法列出的困難，都來自這兩件事：外觀變化（姿態、光照、尺度、視角）、被其他物體或場景遮擋、長得很像的干擾物、快速或非線性的運動、攝影機本身的移動，以及目標進出畫面。

追蹤的分類

白話版。追蹤器可以依照**追什麼**（一個或多個物件、一台或多台攝影機）來分，也可以依照**怎麼維持身分**（只靠外觀，或是逐格跟著運動）來分。

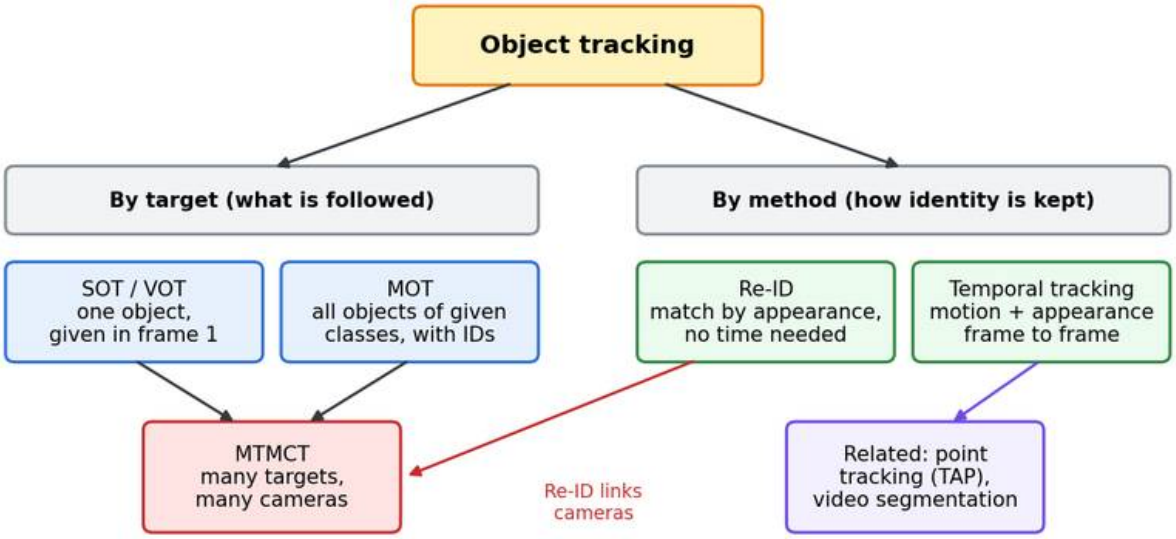


圖 *1* — 追蹤的分類。依目標：單物件（*SOT*，又稱 *VOT*）與多物件（*MOT*）。依方法：只靠外觀的重識別，以及時序追蹤。兩者結合後最困難的情境是多目標跨攝影機追蹤（*MTMCT*）。

依照 [1]：

- **依作法**。重識別（*Re-ID*）只靠外觀（有時加上姿態），在時間相隔很遠、或來自不同攝影機的影像之間比對同一個目標。時序物件追蹤則在同一段影片中逐格跟隨目標，可以利用運動的連續性。
- **依目標**。視覺物件追蹤（*visual object tracking, VOT*）又稱單物件追蹤（*single-object tracking, SOT*），追蹤第一格中以框指定的任意一個物件。多物件追蹤（*multi-object tracking, MOT*）則在**連續影格**中追蹤指定類別的所有物件（例如每一位行人），並為每個物件維持一致的 ID。
- **終極目標**。多目標跨攝影機追蹤（*Multi-Target Multi-Camera Tracking, MTMCT*）在整個攝影機網路中追蹤多個目標，當一個人離開某台攝影機、出現在另一台時，仍維持相同身分。它結合了單一攝影機內的 MOT 與攝影機之間的 Re-ID。

傳統文獻與大多數基準資料集都以**行人**為主要目標 [1]，因為監控與自駕車讓「人」成為最有價值的追蹤類別。新的基準則加入了舞者、運動員、車輛與開放詞彙類別。

重識別（Re-ID）

白話版。 Re-ID 是一個搜尋問題：給你某台攝影機拍到的一個人，請在其他攝影機拍到的照片中找出同一個人，並由最像到最不像排序。

問題定義

Re-ID 被定義成**檢索**（retrieval）問題 [7]。查詢影像（query）q 與由其他攝影機偵測出的行人圖庫（gallery）$\mathcal{G} = \{g_1, \dots, g_M\}$，都經由網路 f_θ 映射成嵌入向量，再依距離排序圖庫，通常使用餘弦距離

$$d(q, g) = 1 - \frac{f_\theta(q)^\top f_\theta(g)}{\|f_\theta(q)\| \, \|f_\theta(g)\|},$$

其中 $f_\theta(\cdot) \in \mathbb{R}^D$ 是參數為 θ 的網路輸出的 D 維嵌入。訓練的目標是讓同一身分的嵌入靠近、不同身分的嵌入遠離。常見做法是把身分分類損失（對 C 個訓練身分做交叉熵）與三**元組損失**（triplet loss）結合：

$$\mathcal{L}_{\text{tri}} = \left[\, d(a, p) - d(a, n) + m \,\right]_+,$$

其中 a 是錨點影像，p 是正樣本（同一身分），n 是負樣本（不同身分），$m > 0$ 是邊界（margin），$[x]_+ = \max(x, 0)$。測試時的身分從未在訓練中出現，所以 Re-ID 是開放集合（open-set）問題：網路必須學到一般性的「是不是同一個人」。

挑戰

[1] 把困難分成**資料因素**與**環境因素**。資料因素：標註身分數量有限、偵測器給出的框有雜訊、身體只拍到一部分、解析度低。環境因素：視角與各攝影機色彩響應不同、光照、遮擋、雜亂背景，以及換了衣服的人。Re-ID 也帶來明顯的**隱私**疑慮，這也是一些監控型資料集被撤下的原因之一（見下文）。原文把城市級監控系統列為主要應用；如今 Re-ID 同樣常見於 MOT 追蹤器裡的外觀模組，以及零售與運動分析。

代表模型

下面三個模型都沿用上面的檢索架構，差別在於怎麼訓練嵌入網路 f_θ。一個普通的 ResNet-50 搭配一組訓練「技巧」，就是強大且可重現的基線（strong baseline）[14]；質心三元組損失（centroid triplet loss）拿影像和各身分的**質心**比較，而不是和單張影像比較 [15]；CLIP-ReID 則把視覺語言模型 CLIP 用在沒有文字標籤的 Re-ID 上 [16]。要記住的一點：把標準骨幹網路仔細訓練好就已經很強，所以新的 Re-ID 想法應該拿這樣的基線來比較。

僅線上版：更多 *Re-ID* 模型：強基線、質心與 *CLIP-ReID* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-tracking/notes/更多-re-id-模型強基線質心與-clip-reid/*

資料集

資料集	內容	狀態
Market-1501 [12]	1,501 個身分、6 台攝影機、偵測器裁切的框	可取得
MSMT17 [13]	4,101 個身分、15 台室內外攝影機	可取得

資料集	內容	狀態
DukeMTMC [10]（及 DukeMTMC-reID）	8 台同步的校園攝影機	**2019 年已撤下**

DukeMTMC 與跨攝影機追蹤的 IDF1 指標一起被提出 [10]，其中的 Re-ID 子集也被廣泛使用。由於在校園拍攝行人引發的隱私與倫理疑慮，作者已於 2019 年撤下該資料集，不應再下載或使用；Peng 等人 [11] 追蹤了它在撤下後仍以衍生版本流傳的情形，並主張資料集需要更好的管理（stewardship）。

指標：CMC rank-k 與 mAP

累積匹配特性（cumulative matching characteristic, CMC）在 rank k 的值，是「第一個正確匹配出現在排序後圖庫前 k 名」的查詢比例：

$$\mathrm{CMC}(k) = \frac{1}{|Q|}\sum_{q \in Q} \mathbb{1}\left[\, r_q \le k \,\right],$$

其中 Q 是查詢集合，r_q 是查詢 q 的第一個正確匹配的名次。*Rank-1* 準確率就是 $\mathrm{CMC}(1)$。由於一個查詢在圖庫中通常有好幾個正確匹配，只看 rank-k 會忽略**其他**正確匹配排得好不好。**平均精確度均值**（mean average precision, mAP）補上了這一點：

$$\mathrm{AP}(q) = \frac{1}{G_q}\sum_{i=1}^{G_q} \frac{i}{r_{q,i}}, \qquad \mathrm{mAP} = \frac{1}{|Q|}\sum_{q} \mathrm{AP}(q),$$

其中 G_q 是 q 在圖庫中的正確匹配數，$r_{q,i}$ 是第 i 個正確匹配的名次，所以 $i / r_{q,i}$ 就是找到第 i 個正確匹配那一刻的精確度。

```
import numpy as np

def cmc_map(dist, q_ids, g_ids, max_rank=5):
    """dist: (Q, G) distances between query and gallery embeddings."""
    order = np.argsort(dist, axis=1)                 # closest gallery items first
    match = g_ids[order] == q_ids[:, None]           # (Q, G) True where same person
    cmc = (np.cumsum(match, axis=1) > 0).mean(axis=0)[:max_rank]
    aps = []
    for row in match:
        hits = np.flatnonzero(row)                   # ranks (0-based) of true matches
        precision_at_hits = np.arange(1, len(hits) + 1) / (hits + 1)
        aps.append(precision_at_hits.mean())
    return cmc, np.mean(aps)

rng = np.random.default_rng(0)
centres = rng.normal(size=(50, 64))                  # 50 identities, 64-D embeddings
g_ids = np.repeat(np.arange(50), 4)                  # 4 gallery images per identity
q_ids = np.arange(50)
gallery = centres[g_ids] + 1.6 * rng.normal(size=(200, 64))
query = centres[q_ids] + 1.6 * rng.normal(size=(50, 64))
```

```
norm = lambda x: x / np.linalg.norm(x, axis=1, keepdims=True)
dist = 1 - norm(query) @ norm(gallery).T            # cosine distance
cmc, mAP = cmc_map(dist, q_ids, g_ids)
print("rank-1 %.2f  rank-5 %.2f  mAP %.2f" % (cmc[0], cmc[4], mAP))
```

在這個合成圖庫（每個身分 4 張）上，程式輸出 `rank-1 0.76  rank-5 0.92  mAP 0.57`：大多數查詢都能在前面找到一**張**正確影像，但另外三張散落在後面，只有 mAP 看得出來。

單物件追蹤

白話版。單物件追蹤接收初始目標框，之後須在各影格定位同一目標。

問題定義

給定影格 $I_1, \dots, I_T$ 與初始框 b_1，單物件追蹤器輸出 $\hat{b}_2, \dots, \hat{b}_T$。目標**與類別無關**：可能是人、鳥或咖啡杯。因此 SOT 是一種 *one-shot* 學習：目標唯一的範例就是第一格的影像塊，也就是**模板**（template）z。在每個新影格，追蹤器在上一個位置附近的搜尋區域 x 中尋找目標。[1] 列出的挑戰是：複雜的外觀變化、移動模式，以及光照、遮擋等環境變化。

傳統方法：相關濾波器

相關濾波器是一張在搜尋視窗上滑動的模板，與影像最吻合的位置得分最高。在深度學習之前，最快的追蹤器會線上學習這樣一個**相關濾波器**（correlation filter）。MOSSE [17] 在頻率域學習濾波器 H，讓它與訓練影像塊 F_i 做相關後，在目標中心產生尖銳的高斯峰 G_i：

$$\min_{H^*} \sum_i |F_i \odot H^* - G_i|^2 \quad \Rightarrow \quad H^* = \frac{\sum_i G_i \odot F_i^*}{\sum_i F_i \odot F_i^*},$$

其中大寫字母代表二維離散傅立葉轉換，$\odot$ 是逐元素相乘，* 是複數共軛，除法也是逐元素進行。因為相關運算在頻率域變成乘法（見IP 04），訓練與偵測只需要幾次 FFT，在 CPU 上就能達到每秒數百格。KCF [18] 指出，用影像塊的**所有循環位移**來訓練，會得到一個可以被 DFT 對角化的循環矩陣，於是核嶺迴歸（kernel ridge regression）變成逐元素運算，也能使用 HOG 等多通道特徵。它們的弱點是搜尋視窗固定、循環假設造成的邊界效應，以及人工設計的特徵。

僅線上版：*MOSSE：在傅立葉域學習的相關濾波器 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-tracking/notes/mosse 在傅立葉域學習的相關濾波器/*

孿生網路追蹤器

白話版。離線用大量影片訓練一個網路，只回答一個問題：「模板出現在這張搜尋影像的哪裡？」測試時直接凍結使用。

SiamFC [19] 把**同一個**全卷積網路 φ 同時套用在模板 z 與搜尋區域 x 上，再把兩張特徵圖做互相關：

$$f(z, x) = \varphi(z) \star \varphi(x) + b\,\mathbb{1},$$

其中 $\star$ 是互相關（模板特徵扮演卷積核），b 是可學習的偏差，$\mathbb{1}$ 是全為 1 的圖。輸出是一張分數圖，峰值位置就是新位置。訓練時使用邏輯損失（logistic loss），真實中心附近標為正、其他位置標為負。由於不做線上更新，它速度快、也不容易漂移，但只能預測位置，尺度則靠搜尋一個小型影像金字塔來處理。

SiamRPN [20] 在孿生特徵之後加上**區域提議網路**（region proposal network）：一個分支判斷錨框是目標還是背景，另一個分支迴歸框的偏移量。追蹤因此變成以模板定義類別的「one-shot 偵測」，能輸出長寬比會變化的精確框。

Transformer 單流追蹤器

2022 年後，主流從孿生 CNN 轉向 Transformer。關鍵差別在於模板與搜尋區域**在哪裡**互動（圖 2）。雙流（two-stream）設計中，兩者分別編碼，最後才比較一次。在 OSTrack [21] 這類**單流**（one-stream）設計中，z 與 x 的影像 patch 被串成同一個 token 序列，由同一個 Vision Transformer 處理，因此每一層注意力都同時在擷取特徵、也在建立模板與搜尋區域之間的關係。OSTrack 也會在前幾層丟掉明顯是背景的搜尋 token 以節省計算。SeqTrack [22] 更進一步簡化：由編碼器—解碼器 Transformer 以自迴歸方式，把框的四個座標當成離散 token 序列逐一生成，只用一般的交叉熵訓練——與語言模型的做法相同。

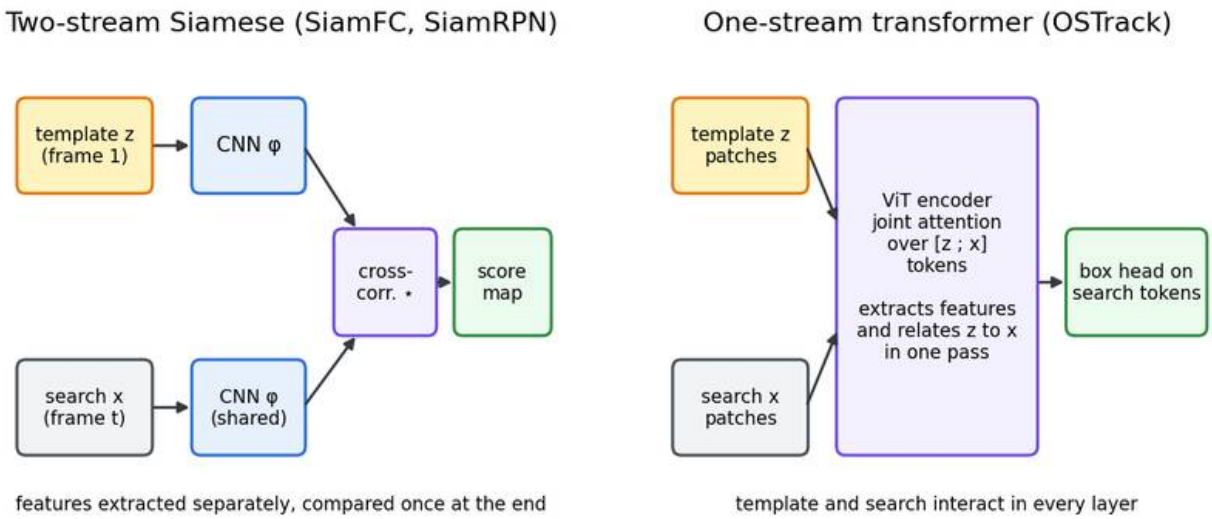


圖 2 — 雙流孿生追蹤器只在最後比較一次模板與搜尋特徵；單流 *Transformer* 追蹤器讓兩者在每一層都互動。

用影片分割模型做追蹤

白話版。提示式影片分割可用像素遮罩追蹤選定物件；身分與時間一致性仍然重要。

SAM 2 [23] 把 Segment Anything 模型延伸到影片，加入**串流記憶**（streaming memory）：過去影格的特徵與預測遮罩，連同被提示的影格，都存在記憶庫中，供目前影格做注意力。只要在一格中用點、框或遮罩提示，就能得到貫穿整段影片的「masklet」，行為就像輸出遮罩的 SOT 追蹤器。SAMURAI [24] 不需重新訓練就把 SAM 2 改造成追蹤器：它加入卡爾曼濾波器運動模型，從 SAM 2 的多個候選遮罩中挑選，並用考慮運動的規則決定哪些過去影格進入記憶，這在有相似干擾物的擁擠場景特別有幫助。SAM 3 [25] 加入**可提示概念分割**（promptable concept segmentation）：給一個名詞片語，例如「黃色校車」，或一張範例影像，它就能在影片中偵測、分割並追蹤**所有**符合的實例，並給每個實例各自的身分。這已經是一種開放詞彙 MOT，SOT 與 MOT 的界線也因此更加模糊。

基準資料集與指標

SOT 基準的差別在於影片長度、規模，以及測試物件的類別是否出現在訓練資料中。兩個經典分數每一格只問一個簡單的問題：預測框和真實框重疊得夠不夠多？它的中心夠不夠近？

基準	特色
OTB [26]	早期的標準基準，每段序列都有屬性標記（遮擋、快速運動等）；提出 success plot 與 precision plot
VOT challenge [53]	自 2013 年起每年舉辦的競賽，採用失敗後重新初始化的評估協定；現為 VOTS，同時評估追蹤**與**分割
LaSOT [27]	長時追蹤：1,400 段序列、超過 350 萬格，每一格都由人工標註
GOT-10k [28]	one-shot 協定：訓練與測試的物件類別互不重疊
TrackingNet [29]	大規模、真實場景影片，以獨立的評估伺服器評分

兩個經典指標來自 OTB [26]：

$$\text{Success}(\tau) = \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}[\,\text{IoU}(\hat{b}_t, b_t) > \tau], \qquad \text{AUC} = \int_0^1 \text{Success}(\tau)\, d\tau,$$

$$\text{Precision}(\delta) = \frac{1}{T}\sum_{t=1}^{T} \mathbb{1}[\|\hat{c}_t - c_t\|_2 \le \delta],$$

其中 $\hat{b}_t, b_t$ 是預測框與真實框，τ 是 IoU 門檻，$\hat{c}_t, c_t$ 是兩者的中心，δ 是像素距離，慣例取 20。success plot 的 AUC 是 OTB 與 LaSOT 的主要數字。GOT-10k 則報告**平均重疊**（average overlap，即平均 IoU），以及 $\tau = 0.5$ 與 0.75 時的成功率。

```
import numpy as np

def iou_xywh(a, b):
    """IoU between rows of a and b, boxes as [x, y, w, h] (top-left corner)."""
    x1 = np.maximum(a[:, 0], b[:, 0]); y1 = np.maximum(a[:, 1], b[:, 1])
    x2 = np.minimum(a[:, 0] + a[:, 2], b[:, 0] + b[:, 2])
    y2 = np.minimum(a[:, 1] + a[:, 3], b[:, 1] + b[:, 3])
    inter = np.clip(x2 - x1, 0, None) * np.clip(y2 - y1, 0, None)
    return inter / (a[:, 2] * a[:, 3] + b[:, 2] * b[:, 3] - inter)
```

```
def success_auc(pred, gt, thresholds=np.linspace(0, 1, 21)):
    ious = iou_xywh(pred, gt)
    curve = np.array([(ious > t).mean() for t in thresholds])
    return curve.mean()                              # area under the success plot

def precision_at(pred, gt, px=20):
    c_pred = pred[:, :2] + pred[:, 2:] / 2; c_gt = gt[:, :2] + gt[:, 2:] / 2
    return (np.linalg.norm(c_pred - c_gt, axis=1) <= px).mean()

rng = np.random.default_rng(0)
gt = np.c_[np.linspace(50, 400, 300), np.full(300, 80.0), np.full(300, 40.0), np.full(300,
 ↪  60.0)]
pred = gt + rng.normal(0, 4, gt.shape)
pred[200:230, :2] += 120                             # 30 frames lost on a distractor
print("success AUC %.3f   precision@20px %.3f" % (success_auc(pred, gt), precision_at(pred,
 ↪  gt)))
```

輸出為 `success AUC 0.647   precision@20px 0.900`：跟到干擾物上的 30 格讓 precision 掉了 10 個百分點；而其他影格上的小幅抖動對 AUC 的影響大得多，因為 AUC 也會獎勵貼得很緊的框。

多物件追蹤

白話版。在每一格找出所有感興趣的物件，並給每個物件一個號碼，只要它還在畫面中，號碼就不變——即使物件彼此交錯、互相遮擋，或離開後又回來。

MOT 是在**連續影格**中同時追蹤多個目標，並為每個目標維持一致的 ID；單一畫格本身只能做偵測。[1] 列出的挑戰包括：頻繁遮擋、軌跡何時開始與結束、目標之間外觀相似，以及目標之間的相互影響（人群中的人會一起移動、彼此遮擋）。

形式上，給定每一格的偵測 $\mathcal{D}_t$，MOT 要找出一組軌跡 $\mathcal{T} = \{\tau_k\}$，把真實的偵測分組、丟掉誤報，並補上漏掉的偵測。離線（offline）方法對整段影片一起求解，常寫成圖或網路流問題；本節著重的**線上**（online）方法，必須只用第 $1..t$ 格在第 t 格就做出決定，這正是即時應用所需要的。

「先偵測再追蹤」流程

十年來的主流設計是**先偵測再追蹤**（tracking-by-detection）[2]、[4]：每一格都跑偵測器，再把偵測結果跨時間連起來（圖 3）。每一格經過四個步驟：

1. **偵測**。偵測器（Faster R-CNN、YOLO 系列、DETR 系列；見 DLCV 03）輸出框與分數。
2. **預測**。運動模型把每條現有軌跡往前推到現在應該在的位置。
3. **描述**。（可選）用 Re-ID 網路把每個偵測框裁切出的影像轉成嵌入向量。
4. **關聯**。在預測的軌跡與偵測之間建立代價矩陣，解指派問題。匹配到的軌跡被更新；沒匹配到的偵測可能開啟新軌跡（**出生**）；太久沒匹配到的軌跡被刪除（**死亡**）。

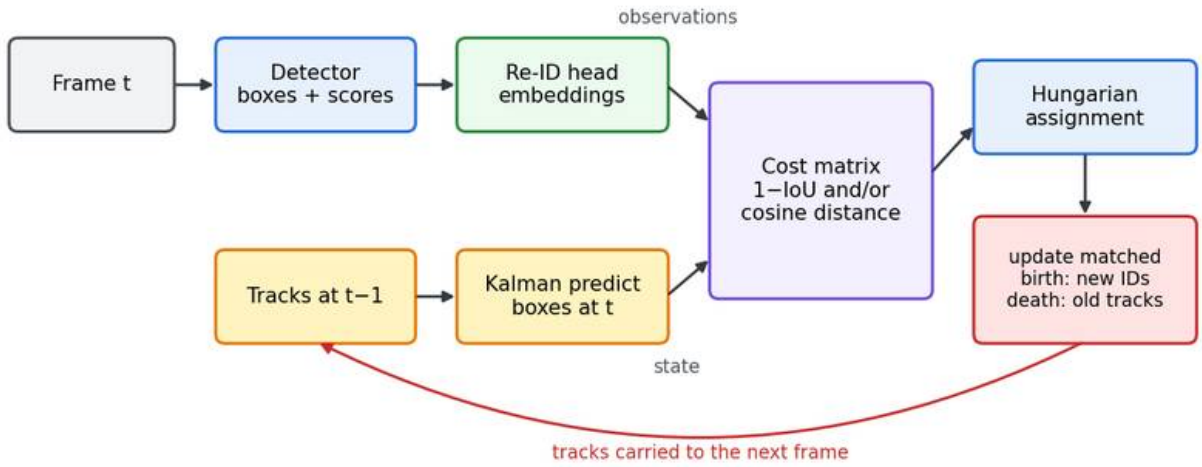


圖 3 — 線上的先偵測再追蹤流程。觀測（框、嵌入）與狀態（預測的軌跡）在代價矩陣中會合，指派結果再更新下一格要用的狀態。

運動模型：卡爾曼濾波器

白話版。先根據物體上一刻的位置與速度猜它現在在哪；再把這個猜測往偵測結果拉一點，而且比較不確定的那一方，就比較不被相信。

幾乎所有先偵測再追蹤的系統，都使用等速模型的**線性卡爾曼濾波器**（Kalman filter）[30]。在我們的程式中，一條軌跡的狀態是

$$\mathbf{x} = [\,c_x,\ c_y,\ w,\ h,\ v_x,\ v_y\,]^\top,$$

其中 (c_x, c_y) 是框的中心，w, h 是寬與高，(v_x, v_y) 是中心的速度（像素／格）。（SORT 使用中心、面積與長寬比，而不是 w, h [32]；BoT-SORT 又改回 w 與 h [36]。）模型分兩步。

預測（時間更新）：

$$\hat{\mathbf{x}}_{t|t-1} = \mathbf{F}\,\hat{\mathbf{x}}_{t-1|t-1},$$
$$\mathbf{P}_{t|t-1} = \mathbf{F}\,\mathbf{P}_{t-1|t-1}\,\mathbf{F}^\top + \mathbf{Q},$$

其中 $\mathbf{F}$ 是 6×6 的狀態轉移矩陣（單位矩陣再加上幾個 1，把 v_x, v_y 加到 c_x, c_y 上，因為 $\Delta t = 1$ 格），$\mathbf{P}$ 是狀態共變異數（我們的不確定性），$\mathbf{Q}$ 是過程雜訊共變異數（真實運動可以偏離等速多少）。

更新（量測更新），使用匹配到的偵測 $\mathbf{z}_t = [c_x, c_y, w, h]^\top$：

$$\mathbf{S}_t = \mathbf{H}\,\mathbf{P}_{t|t-1}\,\mathbf{H}^\top + \mathbf{R}, \qquad \mathbf{K}_t = \mathbf{P}_{t|t-1}\,\mathbf{H}^\top\,\mathbf{S}_t^{-1},$$
$$\hat{\mathbf{x}}_{t|t} = \hat{\mathbf{x}}_{t|t-1} + \mathbf{K}_t(\mathbf{z}_t - \mathbf{H}\,\hat{\mathbf{x}}_{t|t-1}), \qquad \mathbf{P}_{t|t} = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\,\mathbf{P}_{t|t-1},$$

其中 $\mathbf{H} = [\,\mathbf{I}_4\ \ \mathbf{0}\,]$ 取出狀態中可觀測的部分，$\mathbf{R}$ 是偵測器的量測雜訊共變異數，$\mathbf{z}_t - \mathbf{H}\hat{\mathbf{x}}_{t|t-1}$ 是**創新量**（innovation，也就是「意外程度」），$\mathbf{S}_t$ 是它的共變異數，$\mathbf{K}_t$ 是**卡爾曼增益**。若預測非常不

確定（**P** 很大），更新會幾乎把估計值拉到偵測結果上；若偵測器雜訊很大（**R** 很大），就比較相信預測。偵測缺漏時，追蹤器只做預測步驟（稱為「滑行」，coasting），**P** 會持續變大（圖 4）。

```
import numpy as np
from scipy.optimize import linear_sum_assignment

# ---------- 1. constant-velocity Kalman filter ----------
class KalmanBox:
    """State x = [cx, cy, w, h, vx, vy]; measurement z = [cx, cy, w, h]."""
    def __init__(self, z, q=1.0, r=4.0):
        self.x = np.r_[z, 0.0, 0.0]
        self.P = np.diag([r, r, r, r, 100.0, 100.0])    # unknown velocity: large variance
        self.F = np.eye(6); self.F[0, 4] = self.F[1, 5] = 1.0   # dt = 1 frame
        self.H = np.eye(4, 6)
        self.Q = q * np.diag([1, 1, 0.1, 0.1, 0.5, 0.5])
        self.R = r * np.eye(4)

    def predict(self):
        self.x = self.F @ self.x
        self.P = self.F @ self.P @ self.F.T + self.Q
        return self.x[:4]

    def update(self, z):
        S = self.H @ self.P @ self.H.T + self.R          # innovation covariance
        K = self.P @ self.H.T @ np.linalg.inv(S)          # Kalman gain
        self.x = self.x + K @ (z - self.H @ self.x)
        self.P = (np.eye(6) - K @ self.H) @ self.P
```

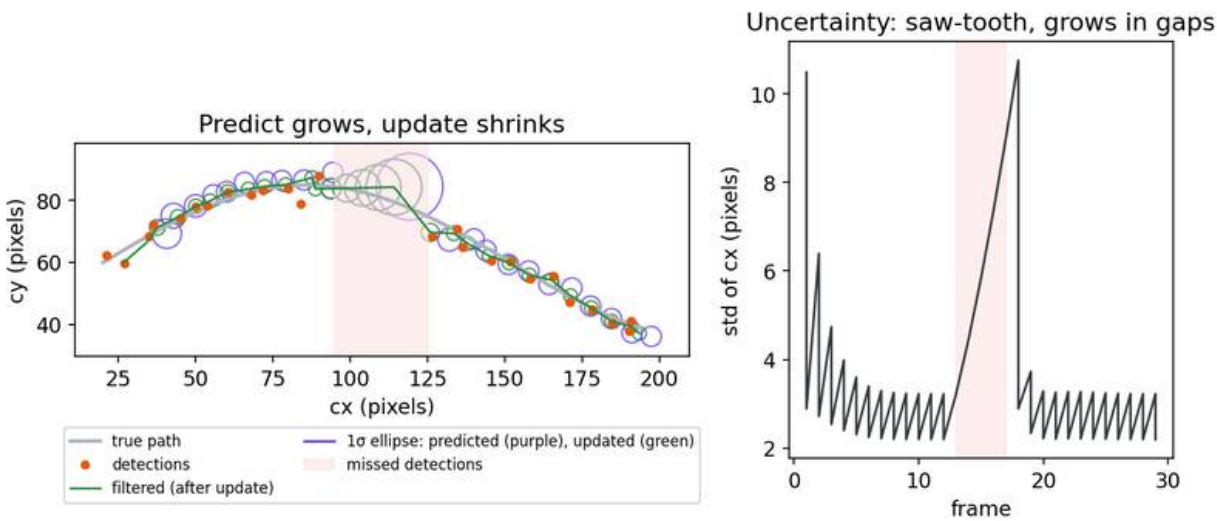


圖 *4* — 在合成軌跡上的卡爾曼預測與更新。每次預測都讓不確定性膨脹，每次更新都讓它縮小。連續五格漏偵測時（粉紅色區域），濾波器沿直線滑行，不確定性持續上升——身分錯誤正是在這種時候發生。

在模型假設下，創新量 $\boldsymbol{\nu}$ 的平方**馬氏距離**（Mahalanobis distance）$d_M^2 = \boldsymbol{\nu}^\top \mathbf{S}^{-1} \boldsymbol{\nu}$ 服從自由度 4 的

χ^2 分布，所以 DeepSORT [33] 用它來**閘控**（gating，直接禁止不合理的匹配）。等速假設也是最大的弱點：舞者、運動員，以及任何由移動攝影機拍攝的畫面，都會違反它。

外觀嵌入

Re-ID 網路（見上一節）把每個偵測裁切影像轉成單位向量 $\mathbf{e}_j$。每條軌跡保有外觀記憶，例如指數移動平均 $\mathbf{m}_k \leftarrow \alpha\,\mathbf{m}_k + (1-\alpha)\,\mathbf{e}_j$ 後再重新正規化，其中 $\alpha \in [0,1)$（我們用 0.9）。外觀代價就是餘弦距離 $1 - \mathbf{m}_k^\top \mathbf{e}_j$。在長時間遮擋後運動預測已經沒用時，正是靠外觀把身分找回來。

關聯：代價矩陣與匈牙利演算法

白話版。為每一組「軌跡 k 配上偵測 j」打一個「有多不合適」的分數，做成一張表，再選出一對一配對中總分最小的那一組。

有 K 條預測軌跡與 N 個偵測時，建立代價矩陣 $\mathbf{C} \in \mathbb{R}^{K \times N}$，例如

$$C_{kj} = \lambda\,(1 - \mathrm{IoU}(\hat{b}_k, b_j)) + (1-\lambda)\,(1 - \mathbf{m}_k^\top \mathbf{e}_j),$$

其中 $\hat{b}_k$ 是軌跡 k 的預測框，b_j 是偵測 j 的框，$\lambda \in [0,1]$ 是運動與外觀的權重（$\lambda = 1$ 就是 SORT 的純 IoU）。接著解**線性指派問題**

$$\min_{\mathbf{X} \in \{0,1\}^{K \times N}} \sum_{k,j} C_{kj} X_{kj} \quad \text{s.t.} \quad \sum_j X_{kj} \le 1,\ \sum_k X_{kj} \le 1,$$

讓每條軌跡最多拿一個偵測，反之亦然。**匈牙利演算法**（Hungarian algorithm）[31] 能在多項式時間內精確求解（$n = \max(K,N)$ 時為 $O(n^3)$）；SciPy 的 `linear_sum_assignment` 實作了一個快速的變形。最後，代價超過門檻的配對會被拒絕，實務上的閘控就是這樣做的。

僅線上版：匈牙利演算法：最佳的一對一指派 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-tracking/notes/匈牙利演算法最佳的一對一指派/*

```
# ---------- 2. IoU and Hungarian association ----------
def iou_matrix(a, b):
    """a: (N,4), b: (M,4) boxes as [cx, cy, w, h] -> (N,M) IoU."""
    a1, a2 = a[:, None, :2] - a[:, None, 2:] / 2, a[:, None, :2] + a[:, None, 2:] / 2
    b1, b2 = b[None, :, :2] - b[None, :, 2:] / 2, b[None, :, :2] + b[None, :, 2:] / 2
    wh = np.clip(np.minimum(a2, b2) - np.maximum(a1, b1), 0, None)
    inter = wh[..., 0] * wh[..., 1]
    area = lambda x: x[..., 2] * x[..., 3]
    return inter / (area(a)[:, None] + area(b)[None, :] - inter + 1e-9)

def associate(cost, max_cost):
    """Hungarian assignment; pairs whose cost exceeds max_cost are rejected."""
    if cost.size == 0:
        return [], list(range(cost.shape[0])), list(range(cost.shape[1]))
    rows, cols = linear_sum_assignment(cost)
    pairs = [(r, c) for r, c in zip(rows, cols) if cost[r, c] <= max_cost]
    mr = {r for r, _ in pairs}; mc = {c for _, c in pairs}
    return (pairs, [r for r in range(cost.shape[0]) if r not in mr],
```

```
            [c for c in range(cost.shape[1]) if c not in mc])
```

在合成場景上的完整追蹤器

為了看清楚每個想法帶來什麼，我們產生一個 80 格、5 位行人的場景，刻意放入幾種經典的失敗情境：

- 物件 0 與 1 **交錯而過**，重疊時位在後方的物件 1 只得到**低信心**偵測；
- 物件 2 與 3 相遇後**反彈回去**（非線性運動，像舞者），重疊時後方那位同樣只有低信心偵測；
- 物件 4 走到**柱子後方** 8 格，這段期間偵測都是低信心；
- 每個偵測都有 5% 機率漏掉，另外偶爾會出現低分的誤報。

每個偵測也帶有一個 16 維外觀向量：所屬物件固定的單位向量加上雜訊。

```
# ---------- 3. synthetic scene ----------
def make_scene(T=80, seed=0):
    rng = np.random.default_rng(seed)
    W, H = 30.0, 70.0
    t = np.arange(T, dtype=float)
    tr = {}
    # pair 1: plain crossing, B is behind A (occluded while they overlap)
    tr[0] = np.c_[80 + 5 * t, np.full(T, 120.0)]
    tr[1] = np.c_[480 - 5 * t, np.full(T, 128.0)]
    # pair 2: they meet and bounce back (non-linear motion, like dancers)
    m = 40
    x2 = np.where(t < m, 120 + 4 * t, 120 + 4 * m - 4 * (t - m))
    x3 = np.where(t < m, 440 - 4 * t, 440 - 4 * m + 4 * (t - m))
    tr[2] = np.c_[x2 + 10, np.full(T, 260.0)]
    tr[3] = np.c_[x3 - 10, np.full(T, 266.0)]
    # walker passing behind a pole (frames 30-37): detections become low-score
    tr[4] = np.c_[60 + 3 * t, 40 + 1.0 * t]
    gt = []                                   # per frame list of (id, box)
    dets = []                                 # per frame list of (box, score, gt_id)
    feats = {k: (lambda v: v / np.linalg.norm(v))(rng.normal(size=16)) for k in tr}
    for f in range(T):
        g, d = [], []
        boxes = {k: np.r_[tr[k][f], W, H] for k in tr}
        for k, b in boxes.items():
            g.append((k, b))
            score = rng.uniform(0.6, 0.95)
            if k == 1 and iou_matrix(b[None], boxes[0][None])[0, 0] > 0.2:
                score = rng.uniform(0.15, 0.45)       # occluded by object 0
            if k == 3 and iou_matrix(b[None], boxes[2][None])[0, 0] > 0.2:
                score = rng.uniform(0.15, 0.45)       # occluded by object 2
            if k == 4 and 30 <= f <= 37:
                score = rng.uniform(0.15, 0.45)       # behind the pole
            if rng.random() < 0.05:
                continue                               # random missed detection
            noisy = b + np.r_[rng.normal(0, 2.0, 2), rng.normal(0, 1.0, 2)]
            e = feats[k] + 0.15 * rng.normal(size=16)
            d.append((noisy, score, k, e / np.linalg.norm(e)))
```

```
        if rng.random() < 0.3:                          # low-score false positive
            fp = np.r_[rng.uniform(50, 550), rng.uniform(50, 320), W, H]
            e = rng.normal(size=16)
            d.append((fp, rng.uniform(0.1, 0.4), -1, e / np.linalg.norm(e)))
        gt.append(g); dets.append(d)
    return gt, dets
```

下面的追蹤器有三個開關。全部關閉時就是 **SORT 風格**：卡爾曼預測、IoU 代價、只對高分偵測（分數 ≥ 0.5）做匈牙利匹配，連續漏 2 格就刪除軌跡。`use_low=True` 加入 **BYTE** 風格的第二輪：沒配到的軌跡再只用 IoU 去配低分偵測。`use_app=True` 在第一輪加入**外觀**代價，並加上第三輪：純靠外觀，把遺失 30 格以內的軌跡重新識別回來。

```
# ---------- 4. SORT-style tracker with optional BYTE and appearance ----------
class Track:
    def __init__(self, tid, z, emb):
        self.id, self.kf, self.emb = tid, KalmanBox(z), emb
        self.lost = 0

def run_tracker(dets, use_low=False, use_app=False, hi=0.5, max_age=2,
                app_age=30, lam=0.5):
    tracks, out, next_id = [], [], 0
    for d in dets:
        for tk in tracks:
            tk.kf.predict()
        boxes = np.array([x[0] for x in d]).reshape(-1, 4)
        scores = np.array([x[1] for x in d])
        embs = np.array([x[3] for x in d]).reshape(-1, 16)
        high = np.where(scores >= hi)[0]
        low = np.where((scores < hi) & (scores >= 0.1))[0]
        active = [tk for tk in tracks if tk.lost <= max_age]
        pred = np.array([tk.kf.x[:4] for tk in active]).reshape(-1, 4)
        # stage 1: high-score detections vs active tracks
        cost = 1 - iou_matrix(pred, boxes[high])
        gate = 0.7
        if use_app and len(active) and len(high):
            tE = np.array([tk.emb for tk in active])
            app = 1 - tE @ embs[high].T            # cosine distance
            cost = np.where(cost > 0.9, 9.0, lam * cost + (1 - lam) * app)
            gate = 0.6
        pairs, ut, ud = associate(cost, gate)
        matched = [(active[r], high[c]) for r, c in pairs]
        rest_tracks = [active[r] for r in ut]
        rest_high = [high[c] for c in ud]
        # stage 2 (BYTE): remaining tracks vs low-score detections, IoU only
        if use_low and rest_tracks and len(low):
            p2 = np.array([tk.kf.x[:4] for tk in rest_tracks])
            pairs2, ut2, _ = associate(1 - iou_matrix(p2, boxes[low]), 0.5)
            matched += [(rest_tracks[r], low[c]) for r, c in pairs2]
            rest_tracks = [rest_tracks[r] for r in ut2]
        # stage 3 (appearance only): long-lost tracks vs leftover high-score dets
        if use_app and rest_high:
            lost = [tk for tk in tracks if max_age < tk.lost <= app_age]
            if lost:
```

```
                app = 1 - np.array([tk.emb for tk in lost]) @ embs[rest_high].T
                pairs3, _, ud3 = associate(app, 0.3)
                matched += [(lost[r], rest_high[c]) for r, c in pairs3]
                rest_high = [rest_high[c] for c in ud3]
        for tk, j in matched:
            tk.kf.update(boxes[j]); tk.lost = 0
            tk.emb = 0.9 * tk.emb + 0.1 * embs[j]; tk.emb /= np.linalg.norm(tk.emb)
        hit = {id(tk) for tk, _ in matched}
        for tk in tracks:
            if id(tk) not in hit:
                tk.lost += 1
        for j in rest_high:                              # birth: unmatched high-score
            tracks.append(Track(next_id, boxes[j], embs[j])); next_id += 1
        tracks = [tk for tk in tracks if tk.lost <= (app_age if use_app else max_age)]
        out.append([(tk.id, tk.kf.x[:4].copy()) for tk in tracks if tk.lost == 0])
    return out
```

指標程式（MOTA、IDF1、HOTA，下文說明）與實驗：

```
# ---------- 5. metrics: MOTA, IDF1, HOTA ----------
def frame_match(g, p, thr):
    if not g or not p:
        return []
    iou = iou_matrix(np.array([b for _, b in g]), np.array([b for _, b in p]))
    r, c = linear_sum_assignment(-iou)
    return [(g[i][0], p[j][0]) for i, j in zip(r, c) if iou[i, j] >= thr]

def mota(gt, hyp, thr=0.5):
    fn = fp = idsw = 0; ngt = 0; last = {}
    for g, p in zip(gt, hyp):
        m = frame_match(g, p, thr)
        ngt += len(g); fn += len(g) - len(m); fp += len(p) - len(m)
        for gi, pi in m:
            if gi in last and last[gi] != pi:
                idsw += 1
            last[gi] = pi
    return 1 - (fn + fp + idsw) / ngt, idsw

def idf1(gt, hyp, thr=0.5):
    gids = sorted({i for g in gt for i, _ in g}); pids = sorted({i for p in hyp for i, _ in
    ↪  p})
    gi = {k: n for n, k in enumerate(gids)}; pi = {k: n for n, k in enumerate(pids)}
    ov = np.zeros((len(gids), len(pids)))
    for g, p in zip(gt, hyp):
        if g and p:
            iou = iou_matrix(np.array([b for _, b in g]), np.array([b for _, b in p]))
            for a in range(len(g)):
                for b in range(len(p)):
                    ov[gi[g[a][0]], pi[p[b][0]]] += iou[a, b] >= thr
    r, c = linear_sum_assignment(-ov)
    idtp = ov[r, c].sum()
    n_gt = sum(len(g) for g in gt); n_p = sum(len(p) for p in hyp)
    return 2 * idtp / (n_gt + n_p)
```

```
def hota(gt, hyp, alphas=np.arange(0.05, 0.96, 0.05)):
    res = []
    for a in alphas:
        pairs = []; n_gt = n_p = 0
        for g, p in zip(gt, hyp):
            pairs += frame_match(g, p, a); n_gt += len(g); n_p += len(p)
        tp = len(pairs)
        if tp == 0:
            res.append((0, 0, 0)); continue
        cnt = {}
        for q in pairs:
            cnt[q] = cnt.get(q, 0) + 1
        gcount = {}; pcount = {}
        for g in gt:
            for i, _ in g: gcount[i] = gcount.get(i, 0) + 1
        for p in hyp:
            for i, _ in p: pcount[i] = pcount.get(i, 0) + 1
        ass = sum(cnt[q] / (gcount[q[0]] + pcount[q[1]] - cnt[q]) for q in pairs) / tp
        det = tp / (n_gt + n_p - tp)
        res.append((np.sqrt(det * ass), det, ass))
    return np.array(res).mean(axis=0)   # HOTA, DetA, AssA averaged over alpha

gt, dets = make_scene(seed=3)
for name, kw in [("IoU only", {}), ("+ low-score pass", dict(use_low=True)),
                 ("+ appearance", dict(use_low=True, use_app=True))]:
    hyp = run_tracker(dets, **kw)
    m, sw = mota(gt, hyp)
    print(f"{name:18s} MOTA {m:.2f}  IDF1 {idf1(gt, hyp):.2f}  "
          f"HOTA {hota(gt, hyp)[0]:.2f}  ID switches {sw}")
```

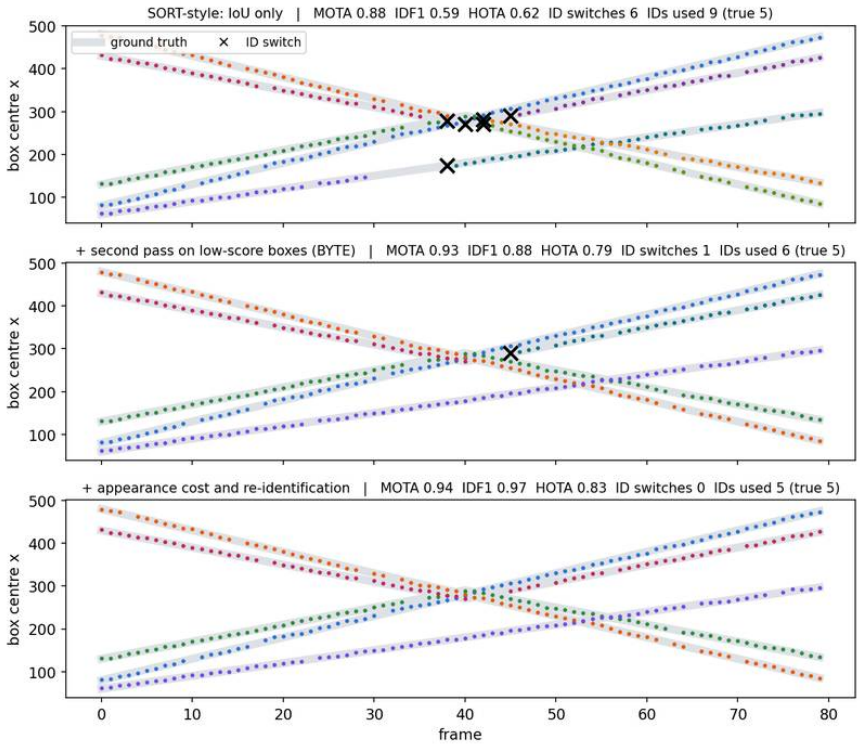


圖 5 — 同一組合成偵測，用三種方式追蹤。顏色代表預測 *ID*，黑色叉號標示 *ID* 切換。低分關聯消除了遮擋造成的斷裂；外觀則修正了反彈情境，因為等速預測此時指向錯誤的方向。

程式輸出：

追蹤器	MOTA	IDF1	HOTA	ID 切換	使用的 ID 數（真實：5）
SORT 風格，只用 IoU	0.88	0.59	0.62	6	9
+ BYTE 低分偵測第二輪	0.93	0.88	0.79	1	6
+ 外觀與重新識別	0.94	0.97	0.83	0	5

仔細看這張表。只用 IoU 的追蹤器會弄丟被遮擋的物件，因為它們的偵測分數低於門檻，軌跡死掉，再出現時就換了**新 ID**（軌跡斷裂，fragmentation）。低分偵測的第二輪讓這些軌跡活下來，幾乎修好了所有問題，但修不了反彈：卡爾曼濾波器預測兩位舞者會繼續往前走，所以重疊之後其中一位沒被配到，稍後以新 ID 重新出現。只有外觀能解決這個情況。也請注意 MOTA 的變化（0.88 → 0.94）比 IDF1（0.59 → 0.97）小得多：MOTA 主要是在衡量偵測，我們會在指標一節再回來談。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

SORT 家族

這些想法正好對應到最常被引用的線上追蹤器：

追蹤器	核心想法
SORT [32]	卡爾曼濾波器 + IoU 代價 + 匈牙利指派；刻意極簡、速度非常快。證明了偵測器的品質主導追蹤品質。
DeepSORT [33]	加入在行人 Re-ID 資料集上訓練的 CNN 外觀描述子、馬氏距離閘控，以及優先處理最近才看到之軌跡的**匹配級聯**（matching cascade）。能撐過更長的遮擋，ID 切換更少。
ByteTrack [34]	關聯**每一個**偵測框：先配高分框，再把剩下的軌跡只用 IoU 去配**低分**框。被遮擋的物件常常分數偏低，因此能被救回而不是被丟掉；而低分的背景框配不到軌跡，就被捨棄。
OC-SORT [35]	「以觀測為中心」：遺失的軌跡被重新找到時，沿著上一次與這一次觀測之間的虛擬軌跡重跑卡爾曼更新（修正滑行期間累積的誤差），並在代價中加入運動方向一致性項。在 DanceTrack 這類非線性運動上特別強。
BoT-SORT [36]	ByteTrack 式的關聯，再加上攝影機運動補償（全域影像對齊）、以 w, h 取代長寬比的卡爾曼狀態，以及融合 IoU 與 Re-ID 的代價。

這個家族的教訓是：在強偵測器之上**精心設計的關聯規則**，在行人基準上非常難被超越。Adžemović 2025 年的綜述 [4] 也得到同樣的結論：在密集、運動大致線性的場景，啟發式方法領先；運動複雜時，學習式關聯表現較好。

聯合偵測與嵌入，以及以偵測器追蹤

對每個裁切框都另外跑一次 Re-ID 網路很昂貴。**JDE** [37] 在單階段偵測器上加一個嵌入頭，一次前向傳遞就同時輸出框與外觀向量。**FairMOT** [38] 指出這種共享對 Re-ID 並「不公平」：以錨框為基礎的偵測器產生的特徵對身分而言很模糊，而且訓練被偵測任務主導。它改用無錨框、CenterNet 風格的偵測器，在高解析度特徵圖上放兩個對等的分支（偵測與 Re-ID）。原文也提到的 **Tracktor** [39]，則在大多數情況下省去顯式關聯：把上一格的框送進偵測器的框迴歸頭，讓每個框被拉到物件的新位置，並保留原本的身分。

以 query 追蹤：端到端 MOT

白話版。不再手寫「預測、比較、配對」，而是給 Transformer 每個物件一個記憶格，讓它自己學會每一格怎麼更新這個記憶格。

DETR 風格的偵測器（見 DLCV 03）用 **query** 表示物件。MOTR [40] 把這延伸到影片：每個偵測到的物件的輸出嵌入，變成一個**軌跡 query**（track query）傳到下一格，持續預測同一個物件；新的**偵測 query** 則負責找出新出現的物件。身分由 query 隱式攜帶，推論時沒有卡爾曼濾波器，也沒有匈牙利匹配。訓練時使用考慮軌跡的標籤指派（tracklet-aware label assignment）與在片段上平均的損失。它的弱點是偵測品質：同一個解碼器既要偵測又要關聯，兩個任務互相衝突。MOTRv2 [41] 以另一個預訓練偵測器（YOLOX）的提議框作為錨點來引導 MOTR，緩解了這個

衝突。MOTIP [42]（CVPR 2025）把關聯重新定義為 **ID 預測**：給定到目前為止的軌跡，每條軌跡帶有一個從可學習 ID 字典中取出的 ID 標籤，解碼器把每個新偵測分類成既有的某個 ID 或「新物件」。這是 in-context、可端到端訓練的，而且只需要物件層級的特徵。

開放詞彙 MOT

傳統 MOT 只能追蹤訓練集中出現過的類別。**OVTrack** [43]（CVPR 2023）透過從視覺語言模型（CLIP）蒸餾知識來做分類與關聯，並用擴散模型**幻想生成**（hallucinate）的影像對來學習穩健的外觀特徵，藉此追蹤任意類別。SAM 3 的概念提示 [25] 則從分割這一側朝同一個目標前進。

資料集

資料集	領域	考驗什麼
MOT17 [44]	行人，固定與移動攝影機	標準行人基準；以 MOT16 的序列重新發布為 MOT17
MOT20 [45]	非常擁擠的行人場景	嚴重遮擋、小目標
DanceTrack [46]	群舞	外觀一致、運動多變且非線性；無法靠外觀區分人
SportsMOT [47]	籃球、排球、足球（240 段序列）	快速且速度多變的運動，外觀相似但仍可區分
BDD100K MOT [48]	行車影片	從移動車輛上做多類別追蹤（汽車、行人、自行車騎士等）

從 MOT17 轉向 DanceTrack 與 SportsMOT，改變了這個領域的重點：在 MOT17 上，好的偵測器加上 IoU 匹配就能走很遠；但在 DanceTrack 上，運動模型與學習式關聯才是關鍵。

指標：MOTA、IDF1、HOTA

白話版。追蹤器有兩種失敗方式：漏掉物件或憑空多出物件（偵測錯誤），或是把誰是誰搞混（關聯錯誤）。不同指標對這兩者的權重不同。

三個指標都先在每一格、以某個 IoU 門檻把預測與真實值配對。令 GT_t、FN_t、FP_t 與 IDSW_t 分別是第 t 格的真實物件數、漏偵測數、誤報數與身分切換數。所謂身分切換，是指某個真實物件這次配到的預測 ID，與它上一次配到的不同。

MOTA（CLEAR MOT [9]）：

$$\mathrm{MOTA} = 1 - \frac{\sum_t (\mathrm{FN}_t + \mathrm{FP}_t + \mathrm{IDSW}_t)}{\sum_t \mathrm{GT}_t}.$$

它的範圍是 $-\infty$ 到 1。由於偵測數量通常遠多於切換次數，FN 與 FP 主導了 MOTA：它主要是偵測分數。而且切換只在發生的那一刻被計算一次，所以一個把兩個 ID 換掉後維持 1,000 格的追蹤器，與一個 1 格後就換回來的追蹤器，付出的代價相同。

IDF1 [10] 在真實軌跡與預測軌跡之間做**全域**的一對一匹配（同樣用匈牙利演算法），最大化兩者一致的影格數，然後

$$\mathrm{IDF1} = \frac{2\,\mathrm{IDTP}}{2\,\mathrm{IDTP} + \mathrm{IDFP} + \mathrm{IDFN}},$$

其中 IDTP 計算那些配到「與其真實身分對應之預測軌跡」的偵測，IDFP、IDFN 則是其餘的預測與真實偵測。IDF1 衡量追蹤器能維持正確身分多久，所以主要是關聯分數——但因為匹配是全域的，它的行為有時並不直觀，例如偵測變好時 IDF1 反而下降。

HOTA（Higher Order Tracking Accuracy [8]）的提出，正是因為上述兩個指標既不平衡、也無法拆解。對一個定位門檻 α，以每格配對得到真陽性（TP）、FN 與 FP：

$$\mathrm{DetA}_\alpha = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FN}| + |\mathrm{FP}|}, \qquad \mathrm{AssA}_\alpha = \frac{1}{|\mathrm{TP}|}\sum_{c \in \mathrm{TP}} \frac{|\mathrm{TPA}(c)|}{|\mathrm{TPA}(c)| + |\mathrm{FNA}(c)| + |\mathrm{FPA}(c)|},$$

$$\mathrm{HOTA}_\alpha = \sqrt{\mathrm{DetA}_\alpha \cdot \mathrm{AssA}_\alpha}, \qquad \mathrm{HOTA} = \frac{1}{19}\sum_{\alpha \in \{0.05, 0.10, \ldots, 0.95\}} \mathrm{HOTA}_\alpha.$$

對一個配對 c（真實 ID g、預測 ID p），$\mathrm{TPA}(c)$ 是整段影片中同樣把 g 配到 p 的所有 TP；$\mathrm{FNA}(c)$ 是 g 被配到其他 ID 或漏掉的偵測；$\mathrm{FPA}(c)$ 是 p 被配到其他物件或沒配到任何物件的偵測。幾何平均代表追蹤器必須在偵測與關聯**兩方面**都好才能得高分；DetA 與 AssA（以及定位準確度 LocA）也可以分開報告，用來診斷問題。對 α 取平均也會獎勵精確的框。（我們的程式用的是簡化版：每格最大化 IoU 的匹配；正式定義使用的匹配分數還會偏好過去已經關聯過的配對。）

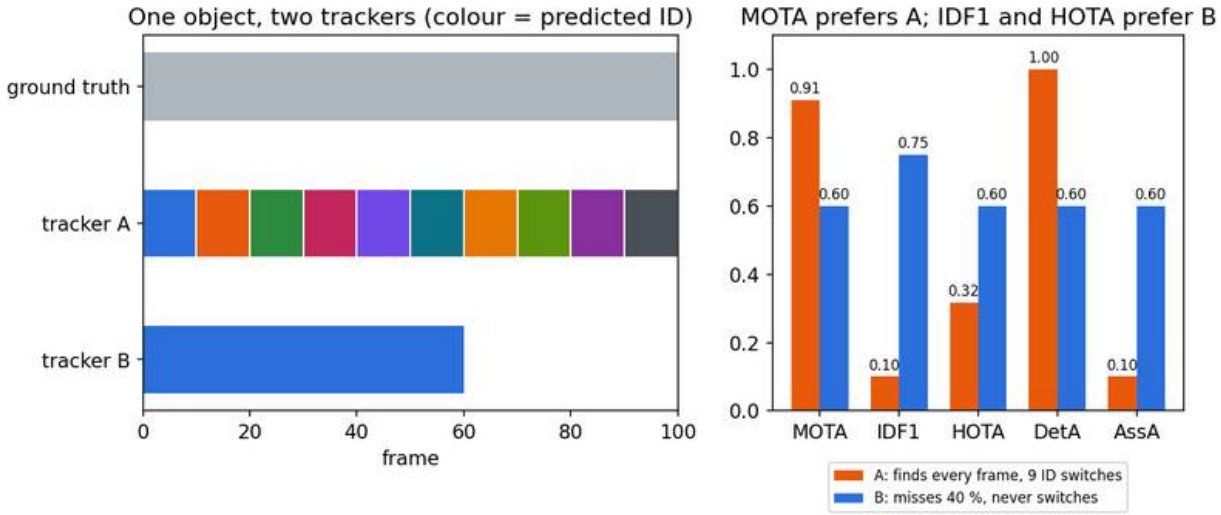


圖 *6 — HOTA* 為什麼被提出。追蹤器 *A* 每格都偵測到，但換了九次身分；追蹤器 *B* 漏掉 *40%* 的影格，但從不換身分。*MOTA* 認為 *A* 遠遠領先；*IDF1* 與 *HOTA* 認為 *B* 較好，而 *DetA* ／ *AssA* 正好說明原因。

圖 6 是用上面的函式算出來的：追蹤器 A 的 $\mathrm{MOTA} = 1 - 9/100 = 0.91$，$\mathrm{AssA} = 0.10$，所以 $\mathrm{HOTA} = \sqrt{1.0 \times 0.10} \approx 0.32$；追蹤器 B 的 MOTA、DetA、AssA 與 HOTA 都是 0.60，$\mathrm{IDF1} = 2 \cdot 60/(100 + 60) = 0.75$。如今 MOT17、MOT20、DanceTrack 與 SportsMOT 上的論文會同時報告 HOTA、IDF1 與 MOTA，而 HOTA 通常是主要的排名指標。

點追蹤：相關的新趨勢

白話版。不追整個物件，而是追單一個**點**——臉頰上的一顆雀斑、箱子的一個角——貫穿整段影片，即使它短暫被擋住也一樣。

追蹤任意點（tracking any point, TAP）的問題是：給定第 t_0 格的查詢點 (x, y)，求它在其他每一格的位置與是否可見。它把光流（兩格之間）推廣到長時間、能處理遮擋的軌跡，也是IP 12傳統關鍵點追蹤的自然延伸。代表模型有 TAPIR [49] 與 CoTracker 系列 [50]、[51]；後者**聯合**追蹤許多點，所以被遮擋的點可以從還看得見的鄰近點推斷出來。點追蹤器現在被用作物件追蹤器中的運動線索，也用在影片編輯與機器人上。

僅線上版：點追蹤模型：*TAPIR* 與 *CoTracker* 系列 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-tracking/notes/點追蹤模型 tapir-與-cotracker-系列/*

多目標跨攝影機追蹤（MTMCT）

白話版。很多台攝影機看著同一個地方。先在每台攝影機裡追蹤每個人，再判斷離開 3 號攝影機的那位女士，就是走進 5 號攝影機的同一位。

MTMCT 就是 [1] 所說的終極目標。典型的流程是：先在每台攝影機內做 MOT 得到**短軌跡**（tracklet），用 Re-ID 特徵描述每條短軌跡，再利用外觀、攝影機拓樸（哪些攝影機相連、彼此之間的移動時間），以及在已校正攝影機下的地面 3D 位置，把不同攝影機的短軌跡分群。IDF1 原本就是為這個情境提出的 [10]。**AI City Challenge** 是主要的常態性基準。第九屆（2025）[52] 的 Track 1 是倉儲型環境中的多類別 **3D** 跨攝影機追蹤，類別包括人、人形機器人、自主移動機器人與堆高機，並提供攝影機校正資料與 3D 框標註。從影像上的框走向多視角、已校正的 3D 軌跡，正是 MTMCT 前進的方向。

現代觀點

白話版。現在有兩類追蹤器並存：精心設計的「偵測、預測、匹配」流程，以及端到端學習關聯的模型。在它們周圍，追蹤正擴展到文字指定的任意類別、單一個點，以及許多台已校正的攝影機。

綜述論文怎麼說

- **Luo 等人 [3]**（*Artificial Intelligence*，2021）：經典的 MOT 綜述；形式化定義問題，並逐一分析各個元件，不依賴任何網路。
- **Ciaparrone 等人 [2]**（*Neurocomputing*，2020）：把深度 MOT 分成四個階段（偵測、特徵擷取／運動預測、親和度、關聯）；大部分進步來自偵測品質與學習式外觀特徵。
- **Adžemović [4]**（arXiv，2025）：比較先偵測再追蹤的各類方法與端到端追蹤器；啟發式追蹤器在擁擠、運動大致線性的場景領先，學習式關聯則在複雜運動上勝出。
- **Marvasti-Zadeh 等人 [5]**（*IEEE T-ITS*，2022）與 **Thangavel（Kugarajeevan）等人 [6]**（*IEEE Access*，2023）：深度與 Transformer SOT；單流設計（OSTrack 類）成為主流模式。

- **Ye 等人 [7]**（*IEEE TPAMI*，2022）：封閉世界與開放世界 Re-ID，並提出 AGW 基線與 mINP 指標。
- **Luiten 等人 [8]**（*IJCV*，2020）：不是綜述，但它對 MOTA 與 IDF1 的重新檢視，正是目前排行榜以 HOTA 排名的原因。

僅線上版：追蹤綜述論文的細節 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/object-tracking/papers/追蹤綜述論文的細節/*

2024–2026 的技術現況

- **SOT** 由 OSTrack ／ SeqTrack 一脈的單流 Transformer 追蹤器主導，而分割基礎模型的角色越來越重：加上運動感知記憶的 SAM 2（SAMURAI），不需針對追蹤訓練就有很強的 zero-shot 追蹤能力 [23]、[24]。
- **MOT** 兩個陣營並存。啟發式的先偵測再追蹤（ByteTrack、OC-SORT、BoT-SORT 及其後繼者）仍是實務上的預設選擇：簡單、快速、模組化，大部分工作由強大的偵測器完成。端到端追蹤器（MOTR、MOTRv2、MOTIP）則在運動複雜、外觀模糊的情境領先，例如 DanceTrack 與 SportsMOT [4]、[42]。
- **開放詞彙與可提示追蹤**正進入主流模型：OVTrack 建立了這個任務，SAM 3 則能依文字概念偵測、分割並追蹤帶有身分的實例 [25]、[43]。
- **點追蹤**（TAPIR、CoTracker、CoTracker3）已成熟為通用的運動基本元件 [49]–[51]。
- **MTMCT** 正走向已校正、多類別的 3D 追蹤（AI City Challenge 2025）[52]。

尚待解決的問題

- **長時間遮擋與重新進場**。幾秒之後運動模型就沒用了，只能靠外觀維持身分；但在服裝一致（DanceTrack、運動比賽）或跨攝影機時，外觀並不可靠。
- **非線性運動與攝影機運動**。等速卡爾曼濾波器仍是預設，也是舞蹈、運動與行車資料上的弱點；學習式運動模型還沒有在所有情境都明顯更好。
- **端到端與啟發式關聯之爭**。端到端追蹤器優雅、在複雜運動上勝出，但需要大量影片訓練資料與計算，在擁擠行人場景上仍輸給簡單的啟發式方法。如何設計出兩邊都最好的統一架構，仍是開放問題。
- **開放詞彙與長尾追蹤**。稀有類別，以及能推廣到訓練中從未見過之類別的外觀特徵。
- **評估**。 HOTA 解決了很多問題，但基準仍以行人為主，而跨數小時影片與多台攝影機的身分層級評估依然困難。**隱私與資料集倫理**（DukeMTMC 的教訓）也限制了能蒐集哪些資料。

重點整理

- 追蹤在跟隨位置時維持身分，再辨識則排序外觀匹配。
- 已初始化的單物件追蹤器與線上多物件追蹤器，解決不同的輸入／輸出問題。
- 先偵測再追蹤結合位置預測、匹配、更新及軌跡建立／終止決策。
- 運動、外觀與低信心偵測提供互補證據，也各有失敗情況。
- 使用符合追蹤任務的指標，分開評估偵測與關聯品質。
- 檢查遮擋、長時間缺失、相機變化與身分切換，不只依賴總分。

練習

1. **卡爾曼增益的直覺**。對一維等位置模型，先驗變異數為 P、量測雜訊為 R，證明卡爾曼增益為 $K = P/(P+R)$，且後驗變異數為 $PR/(P+R)$。若每次預測都讓 P 增加 Q，連續五格沒有偵測之後 K 會怎麼變？

   提示

   $H = 1$ 時，$S = P + R$，$K = P/S$。後驗變異數為 $(1-K)P = PR/(P+R)$，比 P 與 R 都小。滑行五步後先驗變成 $P + 5Q$，K 往 1 靠近：濾波器會非常相信下一個偵測，這也是為什麼遮擋後一次錯誤的匹配會讓軌跡大幅移動。

2. **手算匈牙利演算法**。軌跡 A、B 與偵測 1、2 的預測 IoU 為 $\begin{bmatrix} 0.6 & 0.5 \\ 0.55 & 0.0 \end{bmatrix}$。貪婪匹配（IoU 最高者優先）的結果是什麼？以 $1 - \text{IoU}$ 為代價的匈牙利演算法結果又是什麼？哪一個的 IoU 總和較大？

   提示

   貪婪法先選 A–1（0.6），剩下 B–2 的 IoU 為 0（被拒絕），總和 0.6。匈牙利演算法選 A–2 與 B–1，總和 $0.5 + 0.55 = 1.05$。貪婪法讓軌跡 B 沒配到，並為偵測 2 開一個新 ID——這正是 ID 切換出現的方式之一。

3. **把追蹤器弄壞**。在合成場景中修改 `make_scene` 裡的反彈，讓物件 2 與 3 直接穿過彼此（速度不反轉）。三個追蹤器中哪一個進步最多？為什麼？

   提示

   運動變成線性後，等速預測在重疊期間都是正確的。以 `seed=3` 來說，低分偵測追蹤器的 IDF1 從 0.88 提升到約 0.96，也用了正確的 5 個 ID，幾乎追上外觀追蹤器（同樣約 0.96）；只用 IoU 的追蹤器仍會讓被遮擋的物件斷裂。外觀最有幫助的時候，正是運動非線性的時候，這也是 DanceTrack 的設計初衷。

4. **指標計算**。一段影片中只有一個物件，共 200 格。追蹤器每一格都偵測到它，但第 1–100 格用 ID 1，第 101–200 格用 ID 2。假設框完全正確，在單一門檻下計算 MOTA、IDF1、DetA、AssA 與 HOTA。

   提示

   $\text{MOTA} = 1 - 1/200 = 0.995$。最佳全域 ID 匹配涵蓋 100 格：$\text{IDF1} = 2 \cdot 100/(200+200) = 0.5$。DetA $= 1$。每個 TP 的 $|\text{TPA}| = 100$、$|\text{FNA}| = 100$、$|\text{FPA}| = 0$，所以 AssA $= 0.5$，HOTA $= \sqrt{0.5} \approx 0.71$。一次切換幾乎不影響 MOTA，卻讓 AssA 減半。

5. **Re-ID 指標**。在 Re-ID 程式中，把雜訊從 1.6 改成 2.2，再改成 1.0。rank-1 與 mAP 怎麼變？哪一個比較敏感？請用 AP 的定義解釋。

提示

雜訊變大時 mAP 下降得比 rank-1 快，因為 rank-1 只需要最容易的那一個正確匹配排在第一，而 AP 會懲罰每一個被往後推的正確匹配。每個查詢有 4 個正確匹配時，最難的那一個通常決定了 AP 損失多少。

6. **設計題**。你要在倉庫中用 12 台已校正的攝影機追蹤堆高機與工作人員。用本篇介紹的元件畫出一個流程，並說明你會報告哪個指標、為什麼。

提示

每台攝影機做偵測，搭配 ByteTrack ／ BoT-SORT 風格的追蹤器；利用校正參數把框投影到地面；跨攝影機關聯先看 3D 位置與時間，再看 Re-ID 外觀（工作人員可能穿著一樣的背心）；兩個類別分開處理。報告 HOTA（平衡，且可拆成 DetA ／ AssA）與 IDF1（跨攝影機的長時身分），與 AI City Challenge 的設定一致。

參考文獻

1. Shyandram，〈物件追蹤 Object Tracking 簡介〉，Medium 文章，2023；2026 年更新。連結
2. G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri and F. Herrera, “Deep learning in video multi-object tracking: A survey,” *Neurocomputing*, 2020. arXiv:1907.12740
3. W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu and T.-K. Kim, “Multiple object tracking: A literature review,” *Artificial Intelligence*, vol. 293, 2021 (arXiv 2014). arXiv:1409.7618
4. M. Adžemović, “Deep learning-based multi-object tracking: A comprehensive survey from foundations to state-of-the-art,” arXiv:2506.13457, 2025. arXiv
5. S. M. Marvasti-Zadeh, L. Cheng, H. Ghanei-Yakhdan and S. Kasaei, “Deep learning for visual tracking: A comprehensive survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, 2022. arXiv:1912.00535
6. J. Thangavel (Kugarajeevan), T. Kokul, A. Ramanan and S. Fernando, “Transformers in single object tracking: An experimental survey,” *IEEE Access*, vol. 11, 2023. arXiv:2302.11867
7. M. Ye, J. Shen, G. Lin, T. Xiang, L. Shao and S. C. H. Hoi, “Deep learning for person re-identification: A survey and outlook,” *IEEE TPAMI*, vol. 44, 2022. arXiv:2001.04193
8. J. Luiten, A. Ošep, P. Dendorfer, P. Torr, A. Geiger, L. Leal-Taixé and B. Leibe, “HOTA: A higher order metric for evaluating multi-object tracking,” *International Journal of Computer Vision*, 2020. arXiv:2009.07736
9. K. Bernardin and R. Stiefelhagen, “Evaluating multiple object tracking performance: The CLEAR MOT metrics,” *EURASIP Journal on Image and Video Processing*, 2008. doi
10. E. Ristani, F. Solera, R. S. Zou, R. Cucchiara and C. Tomasi, “Performance measures and a data set for multi-target, multi-camera tracking,” *ECCV Workshop on Benchmarking Multi-Target Tracking*, 2016. arXiv:1609.01775
11. K. Peng, A. Mathur and A. Narayanan, “Mitigating dataset harms requires stewardship: Lessons from 1000 papers,” arXiv:2108.02922, 2021. arXiv
12. L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang and Q. Tian, “Scalable person re-identification: A benchmark,” *ICCV*, 2015. doi

13. L. Wei, S. Zhang, W. Gao and Q. Tian, “Person transfer GAN to bridge domain gap for person re-identification,” *CVPR*, 2018. arXiv:1711.08565
14. H. Luo, Y. Gu, X. Liao, S. Lai and W. Jiang, “Bag of tricks and a strong baseline for deep person re-identification,” *CVPR Workshops*, 2019. arXiv:1903.07071
15. M. Wieczorek, B. Rychalska and J. Dąbrowski, “On the unreasonable effectiveness of centroids in image retrieval,” arXiv:2104.13643, 2021. arXiv
16. S. Li, L. Sun and Q. Li, “CLIP-ReID: Exploiting vision-language model for image re-identification without concrete text labels,” *AAAI*, 2023. arXiv:2211.13977
17. D. S. Bolme, J. R. Beveridge, B. A. Draper and Y. M. Lui, “Visual object tracking using adaptive correlation filters,” *CVPR*, 2010. doi
18. J. F. Henriques, R. Caseiro, P. Martins and J. Batista, “High-speed tracking with kernelized correlation filters,” *IEEE TPAMI*, vol. 37, 2015. arXiv:1404.7584
19. L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi and P. H. S. Torr, “Fully-convolutional Siamese networks for object tracking,” arXiv:1606.09549, 2016 (ECCV 2016 workshops). arXiv
20. B. Li, J. Yan, W. Wu, Z. Zhu and X. Hu, “High performance visual tracking with Siamese region proposal network,” *CVPR*, 2018. doi
21. B. Ye, H. Chang, B. Ma, S. Shan and X. Chen, “Joint feature learning and relation modeling for tracking: A one-stream framework” (OSTrack), *ECCV*, 2022. arXiv:2203.11991
22. X. Chen, H. Peng, D. Wang, H. Lu and H. Hu, “SeqTrack: Sequence to sequence learning for visual object tracking,” *CVPR*, 2023 (the arXiv entry was later extended as SeqTrackv2). arXiv:2304.14394
23. N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma et al., “SAM 2: Segment anything in images and videos,” *ICLR*, 2025. arXiv:2408.00714
24. C.-Y. Yang, H.-W. Huang, W. Chai, Z. Jiang and J.-N. Hwang, “SAMURAI: Adapting segment anything model for zero-shot visual tracking with motion-aware memory,” arXiv:2411.11922, 2024. arXiv
25. N. Carion, L. Gustafson, Y.-T. Hu et al., “SAM 3: Segment anything with concepts,” *ICLR*, 2026. arXiv:2511.16719
26. Y. Wu, J. Lim and M.-H. Yang, “Object tracking benchmark,” *IEEE TPAMI*, vol. 37, 2015. doi
27. H. Fan, L. Lin, F. Yang, P. Chu, G. Deng, S. Yu et al., “LaSOT: A high-quality benchmark for large-scale single object tracking,” *CVPR*, 2019. arXiv:1809.07845
28. L. Huang, X. Zhao and K. Huang, “GOT-10k: A large high-diversity benchmark for generic object tracking in the wild,” *IEEE TPAMI*, vol. 43, 2021. arXiv:1810.11981
29. M. Müller, A. Bibi, S. Giancola, S. Al-Subaihi and B. Ghanem, “TrackingNet: A large-scale dataset and benchmark for object tracking in the wild,” arXiv:1803.10794, 2018 (ECCV 2018). arXiv
30. R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960. doi
31. H. W. Kuhn, “The Hungarian method for the assignment problem,” *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955. doi
32. A. Bewley, Z. Ge, L. Ott, F. Ramos and B. Upcroft, “Simple online and realtime tracking,” *ICIP*, 2016. arXiv:1602.00763
33. N. Wojke, A. Bewley and D. Paulus, “Simple online and realtime tracking with a deep association metric,” *ICIP*, 2017. doi

34. Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu and X. Wang, “ByteTrack: Multi-object tracking by associating every detection box,” *ECCV*, 2022. arXiv:2110.06864
35. J. Cao, J. Pang, X. Weng, R. Khirodkar and K. Kitani, “Observation-centric SORT: Rethinking SORT for robust multi-object tracking,” *CVPR*, 2023. arXiv:2203.14360
36. N. Aharon, R. Orfaig and B.-Z. Bobrovsky, “BoT-SORT: Robust associations multi-pedestrian tracking,” arXiv:2206.14651, 2022. arXiv
37. Z. Wang, L. Zheng, Y. Liu, Y. Li and S. Wang, “Towards real-time multi-object tracking” (JDE), *ECCV*, 2020. arXiv:1909.12605
38. Y. Zhang, C. Wang, X. Wang, W. Zeng and W. Liu, “FairMOT: On the fairness of detection and re-identification in multiple object tracking,” *IJCV*, 2021. arXiv:2004.01888
39. P. Bergmann, T. Meinhardt and L. Leal-Taixé, “Tracking without bells and whistles” (Tracktor), *ICCV*, 2019. arXiv:1903.05625
40. F. Zeng, B. Dong, Y. Zhang, T. Wang, X. Zhang and Y. Wei, “MOTR: End-to-end multiple-object tracking with transformer,” *ECCV*, 2022. arXiv:2105.03247
41. Y. Zhang, T. Wang and X. Zhang, “MOTRv2: Bootstrapping end-to-end multi-object tracking by pretrained object detectors,” *CVPR*, 2023. arXiv:2211.09791
42. R. Gao, J. Qi and L. Wang, “Multiple object tracking as ID prediction” (MOTIP), *CVPR*, 2025. arXiv:2403.16848
43. S. Li, T. Fischer, L. Ke, H. Ding, M. Danelljan and F. Yu, “OVTrack: Open-vocabulary multiple object tracking,” *CVPR*, 2023. arXiv:2304.08408
44. A. Milan, L. Leal-Taixé, I. Reid, S. Roth and K. Schindler, “MOT16: A benchmark for multi-object tracking,” arXiv:1603.00831, 2016 (MOT16/MOT17). arXiv
45. P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler and L. Leal-Taixé, “MOT20: A benchmark for multi object tracking in crowded scenes,” arXiv:2003.09003, 2020. arXiv
46. P. Sun, J. Cao, Y. Jiang, Z. Yuan, S. Bai, K. Kitani and P. Luo, “DanceTrack: Multi-object tracking in uniform appearance and diverse motion,” *CVPR*, 2022. arXiv:2111.14690
47. Y. Cui, C. Zeng, X. Zhao, Y. Yang, G. Wu and L. Wang, “SportsMOT: A large multi-object tracking dataset in multiple sports scenes,” *ICCV*, 2023. arXiv:2304.05170
48. F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan and T. Darrell, “BDD100K: A diverse driving dataset for heterogeneous multitask learning,” *CVPR*, 2020. arXiv:1805.04687
49. C. Doersch, Y. Yang, M. Vecerik, D. Gokay, A. Gupta, Y. Aytar et al., “TAPIR: Tracking any point with per-frame initialization and temporal refinement,” *ICCV*, 2023. arXiv:2306.08637
50. N. Karaev, I. Rocco, B. Graham, N. Neverova, A. Vedaldi and C. Rupprecht, “CoTracker: It is better to track together,” *ECCV*, 2024. arXiv:2307.07635
51. N. Karaev, I. Makarov, J. Wang, N. Neverova, A. Vedaldi and C. Rupprecht, “CoTracker3: Simpler and better point tracking by pseudo-labelling real videos,” arXiv:2410.11831, 2024. arXiv
52. Z. Tang, S. Wang, D. C. Anastasiu et al., “The 9th AI City Challenge,” *ICCV Workshops*, 2025. arXiv:2508.13564
53. VOT Challenge organizers, “The Visual Object Tracking (VOT) challenge series,” website, 2013–present. votchallenge.net
54. R. Jonker and A. Volgenant, “A shortest augmenting path algorithm for dense and sparse linear assignment problems,” *Computing*, vol. 38, no. 4, pp. 325–340, 1987. doi
55. D. F. Crouse, “On implementing 2D rectangular assignment algorithms,” *IEEE Transactions on*

Aerospace and Electronic Systems, vol. 52, no. 4, pp. 1679–1696, 2016. doi

DLCV 08・影片理解與動作辨識

深度網路如何辨識影片裡發生了什麼事：任務（片段分類、時序與時空動作定位、檢索與問答）、為什麼時間難學、雙流、3D、(2+1)D、SlowFast、TSN/TSM 與影片 Transformer 架構、遮罩式影片預訓練、評估方式、資料集，以及場景偏差等失效模式。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/

你將學到

- 為影片任務配對片段、區間、人員層級或語言輸出。
- 區分時間推理與外觀捷徑。
- 比較光流分支、時間卷積、取樣與注意力。
- 理解遮罩影片預訓練與影片語言遷移。
- 評估效能與成本時，控制影格、視圖、解析度及預訓練。

先看全貌

影像分類器看一張圖，說出圖裡有什麼。影片模型還得理解發生了什麼事：誰做了什麼、先後順序、持續多久。多出來的輸入是時間軸，也帶來影像沒有的兩個問題。第一，時間很貴：30 影格／秒的 10 秒片段就有 300 張影像。第二，時間很容易被忽略：網路常常光看球場就能猜出「打網球」，根本不必看揮拍的動作。

> **用最簡單的話說**
>
> 看一張門的照片。有人正在開門，還是關門？你分不出來。現在把前後連續拍的幾張照片像翻頁動畫一樣翻過去，答案就很明顯了。影片模型要學的是讀這本翻頁動畫，而不是只看其中一頁。麻煩的是它常常會作弊：如果每張「游泳」的照片都有泳池，它只要找藍色的水就行，永遠不必學游泳長什麼樣子。

影片理解很重要，因為大部分視覺資料都是影片：手機錄影、監視器與交通攝影機、運動轉播、機器人、汽車與醫療處置。辨識動作是許多應用的基礎：搜尋影片典藏、分析運動、監控安全、協助居家生活與控制機器人。這個領域也產生了擴散到深度學習其他角落的想法，例如把預訓練的 2D 網路膨脹成 3D、把空間與時間分解開來，以及極高遮罩比例的遮罩式預訓練。

本篇假設你已熟悉 DLCV 02 的影像分類、訓練與評估。它會重用 DLCV 03 的偵測指標，以及 ML 05 的注意力。有五篇綜述為這個領域勾勒出輪廓：Zhu 等人談深度動作辨識 [1]、Kong 與 Fu 談辨識與預測 [2]、Selva 等人談影片 Transformer [3]、Madan 等人談影片基礎模型 [4]，以及 Tang 等人談以大型語言模型做影片理解 [5]。我們會在「現代觀點」回頭討論它們。

相關章節。在時間上追蹤物件是 DLCV 07・物件追蹤；骨架式辨識所用的人體關鍵點在 DLCV 05；光流的估計在 DLCV 11A；用時空注意力與影片自編碼器生成影片在 DLCV 12A；預測場景的未來以便規劃則在 DLCV 13・世界模型。

影片理解的任務

白話版。影片輸出可以是片段標籤、動作區間、人員偵測或語言答案；選模型前先定義輸出。

精確版。影片是一個張量 $V \in \mathbb{R}^{T\times H\times W\times 3}$，其中 T 是影格數，$H \times W$ 是影格大小，3 是色彩通道。模型很少一次看到整個 V，它看到的是一個**片段**（clip）$x \in \mathbb{R}^{T'\times H'\times W'\times 3}$，由 $T' \ll T$ 張取樣出來的影格組成，並已縮放與裁切成 $H' \times W'$。圖 1 畫出四類任務。

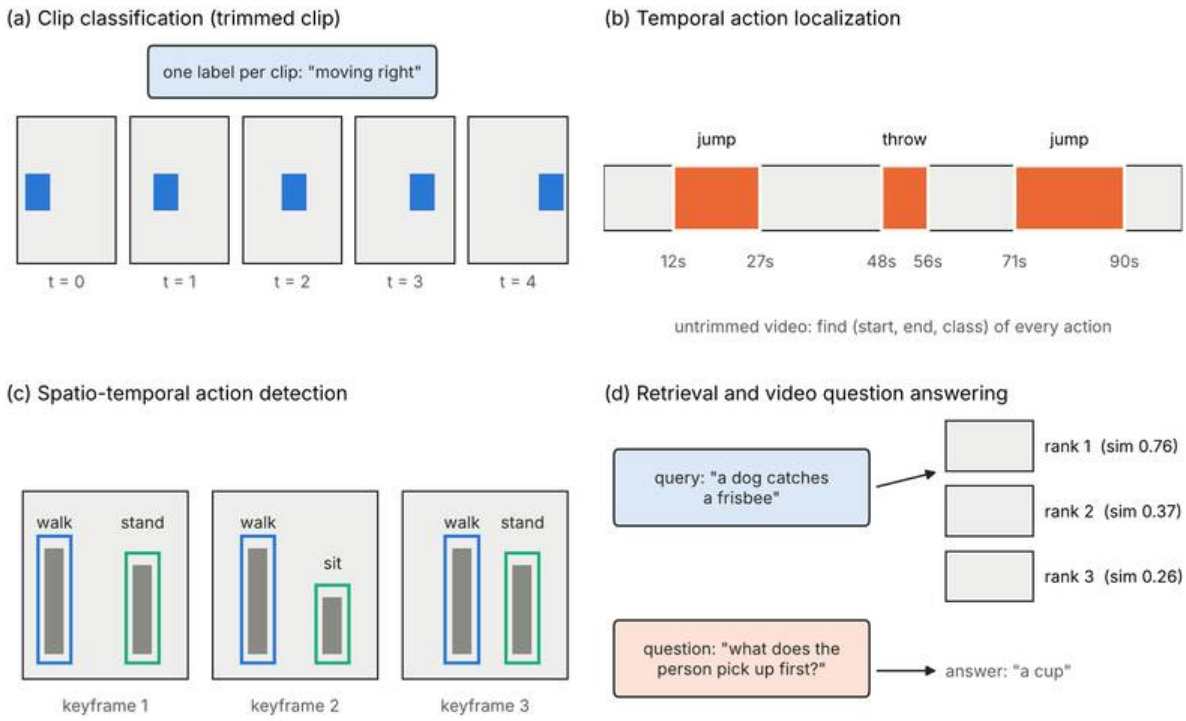


圖 *1* — 影片的主要任務：(*a*) 每段已裁剪片段一個標籤；(*b*) 對未裁剪影片中的每個動作給出開始、結束與類別；(*c*) 對每張關鍵影格中的每個人給一個框與動作標籤；(*d*) 依文字查詢對影片排序，或回答關於影片的問題。

1. **片段（動作）分類**。給定一段含有單一動作的已裁剪片段，預測它的類別 $y \in \{1, \dots, K\}$。以交叉熵訓練，並以 top-1 與 top-5 準確率評估，與 DLCV 02 完全相同。有些資料集是**多標籤**的（同時有好幾個動作），以各類別的平均精確率均值（mean average precision, mAP）評估。
2. **時序動作定位**（temporal action localization，也稱時序動作偵測）。給定一支未裁剪的影片，輸出一組片段 $\{(s_i, e_i, c_i, p_i)\}$，分別是開始時間 s_i、結束時間 e_i、類別 c_i 與信心分數 p_i。這就是沿著一個維度做的物件偵測，以數個**時序 IoU**（temporal IoU）門檻下的 mAP 來評估。
3. **時空動作偵測**。找出誰在哪裡、何時做了什麼：在每張有標註的關鍵影格中給出人物框，每個框帶一個或多個動作標籤。以固定框 IoU 門檻下的影格層級 mAP 評估。
4. **影片–語言任務**。**文字到影片檢索**（text-to-video retrieval）依影片與一個句子的相似度為影片集合排序，以 Recall@k 評估（正確影片落在前 k 名的查詢所占的比例）。**影片問答**（video question answering）產生簡短的答案，常是從多個選項中挑選，以準確率計分。

相鄰的問題在別處討論。在各影格之間連結同一個物件是追蹤（DLCV 07）。預測接下來發生什麼是動作預期或世界建模（DLCV 13）。替每張影格的每個像素貼標籤則是影片分割。

為什麼影片很難

白話版。有三件事讓影片比影像更難：標籤常常取決於順序與運動、資料集讓模型可以省掉這些功夫，而且輸入量非常大。

時間很重要，但資料不一定逼你使用它

有些動作只能靠運動分辨：「開門」與「關門」、「把東西推向左」與「把東西推向右」、「坐下」與「站起來」。這些動作的單張影格是模稜兩可的。許多其他動作靠單張影格就能認出，因為有**場景**與**物件**：游泳池、馬、吉他。從網路蒐集的資料集裡，有很多動作屬於第二類。

這件事很早就看得出來。Karpathy 等人用 100 萬支 YouTube 影片、487 個運動類別的資料集訓練 CNN，比較了幾種跨時間融合資訊的方式；他們最好的時空網路只比單影格模型略有進步 [6]。後來的研究直接量測這個問題。Li 等人定義了資料集的**表徵偏差**（representation bias），並顯示常見的動作資料集偏向靜態線索（物件、場景與人物）；他們也建立了 Diving48，一個包含 48 種細緻跳水類別的競技跳水資料集，以降低這種偏差 [7]。Sevilla-Lara 等人請人們辨認影格被打亂順序的影片中的動作，只保留變得無法辨認的類別；模型能不能使用時間，就在這個「時序」子集上顯現出來 [8]。Something-Something 則是為了捕捉動作的細部物理面向，而不是高層概念而建立的：它的 174 個類別是像「把東西從左推到右」這樣的描述樣板 [9]，VideoMAE 的作者稱之為以運動為中心 [10]。

網路用手邊最容易的線索解決訓練任務，這個一般現象就是 DLCV 02 的**捷徑學習**（shortcut learning）。在影片裡，最容易的線索通常是場景。

一個單張影格毫無資訊的資料集

下面的程式碼用合成片段把這件事具體化。一個小方塊向左或向右移動，標籤就是方向。方塊在中間影格的位置，在兩種標籤下分布相同，所以**單看中間影格，完全得不到標籤的資訊**。接著我們加上「場景」：背景亮度在 90% 的訓練片段中與標籤一致。三個邏輯迴歸分類器分別看到 (1) 中間影格的像素、(2) 一個運動特徵，以及 (3) 兩者。

運動特徵是經典的亮度恆常性估計（見 DLCV 11A）。如果某個圖樣每影格位移 u 個像素，則 $I_{t+1}(x) \approx I_t(x-u)$，所以時間差分為 $I_{t+1} - I_t \approx -u\,\partial_x I_t$。對整張影格以最小平方法擬合 u，得到

$$\hat{u}_t = -\frac{\sum_{x,y} \left(I_{t+1}(x,y) - I_t(x,y)\right)\partial_x I_t(x,y)}{\sum_{x,y} \left(\partial_x I_t(x,y)\right)^2},$$

其中 $I_t(x,y)$ 是第 t 張影格在第 x 欄、第 y 列的亮度，$\partial_x I_t$ 是它的水平導數，$\hat{u}_t$ 是第 t 與 $t+1$ 張影格之間估計的水平運動（正值表示向右）。它是一個時間導數濾波器和一個空間導數濾波器的乘積，這正是 3D 卷積能學到的那類圖樣。

```
import numpy as np
from sklearn.linear_model import LogisticRegression

def make_clips(n, T=8, S=20, noise=0.5, scene_bias=0.5, seed=0):
    """A 3x3 square moves 1 px/frame left (y=0) or right (y=1) on a noisy background.
    scene_bias = P(background brightness agrees with the label); 0.5 = no bias."""
    rng = np.random.default_rng(seed)
    y = rng.integers(0, 2, n)
    X = noise * rng.standard_normal((n, T, S, S))
    for i in range(n):
        step = 1 if y[i] == 1 else -1
        c_mid = rng.integers(T // 2, S - 3 - T // 2 + 1)   # middle-frame position: same for
        ↪  both labels
        r = rng.integers(0, S - 3 + 1)
        agree = rng.random() < scene_bias
        X[i] += 0.3 * (y[i] if agree else 1 - y[i])       # the "scene": background
        ↪  brightness
        for t in range(T):
            c = c_mid + step * (t - T // 2)
            X[i, t, r:r + 3, c:c + 3] += 1.0
    return X, y

def frame_features(X):     # appearance only: the pixels of the middle frame
    return X[:, X.shape[1] // 2].reshape(len(X), -1)

def motion_features(X):    # temporal difference times horizontal gradient, per frame pair
    dt = X[:, 1:] - X[:, :-1]
    dx = np.gradient(X[:, :-1], axis=3)
    u = -(dt * dx).sum(axis=(2, 3)) / ((dx ** 2).sum(axis=(2, 3)) + 1e-6)
    return u                                              # one motion estimate per frame pair

data = {"train": make_clips(2000, scene_bias=0.9, seed=1),  # scene agrees with the label 90%
        "same": make_clips(1000, scene_bias=0.9, seed=2),   # test: same distribution
        "no cue": make_clips(1000, scene_bias=0.5, seed=3), # test: scene says nothing
        "shift": make_clips(1000, scene_bias=0.1, seed=4)}  # test: scene misleads 90%
for name, f in [("one frame", frame_features), ("motion", motion_features),
                ("frame+motion", lambda X: np.c_[frame_features(X), motion_features(X)])]:
    clf = LogisticRegression(C=0.1, max_iter=5000).fit(f(data["train"][0]), data["train"][1])
    acc = {k: clf.score(f(X), y) for k, (X, y) in data.items() if k != "train"}
    print(f"{name:13s}", "  ".join(f"{k} {v:.3f}" for k, v in acc.items()))
```

```
one frame     same 0.867  no cue 0.495  shift 0.131
motion        same 0.945  no cue 0.929  shift 0.941
frame+motion  same 0.889  no cue 0.522  shift 0.163
```

（這個區塊與圖 2 的所有數字都來自這次玩具實驗。）在與訓練相同分布的測試資料上，單影格模型看起來不錯（0.867），因為場景有 90% 的機率能預測標籤。拿掉場景線索，它就掉到隨機猜測的水準（0.495）；讓場景產生誤導，它就跌到 0.131。運動模型在每種情況下都約在 0.93–0.94，因為它用的是真正決定標籤的線索。組合模型是個警訊：它有運動特徵，卻學成依賴數百個亮度像素，在分布偏移下失敗（0.163）。它對場景的依賴程度，取決於特徵尺度與正則化（練習 1），但從訓練分布抽出來的基準，永遠不會暴露這個問題。

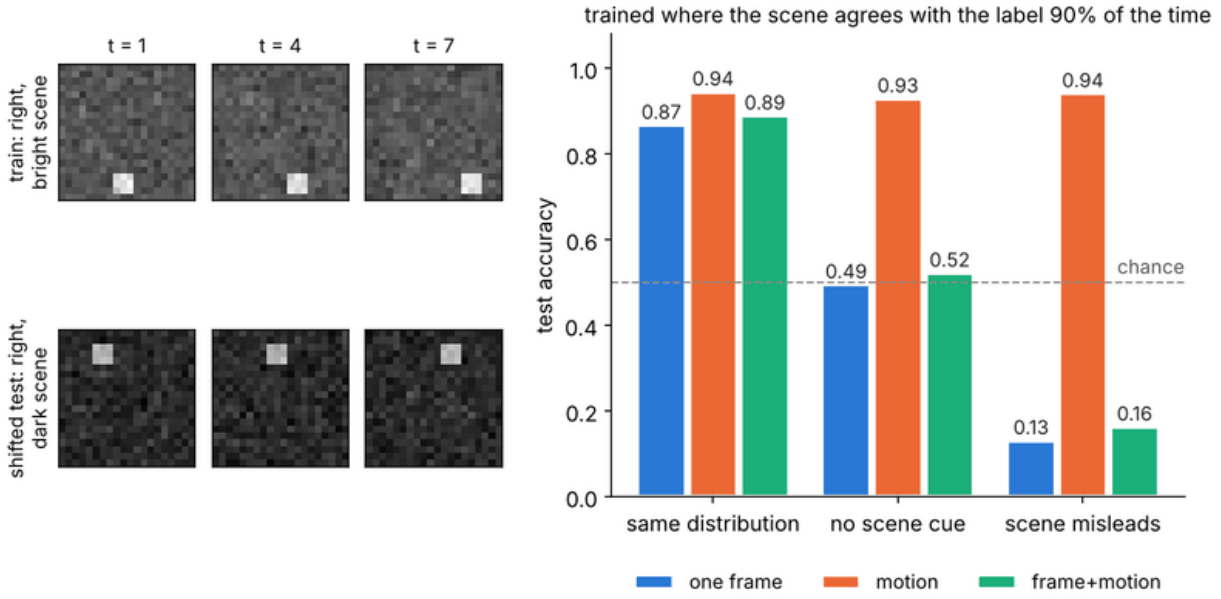


圖 2 — 場景偏差玩具實驗。左：訓練片段（亮場景）與偏移後的測試片段（暗場景）的影格；為了看得見方塊，這裡的雜訊比實驗中少。右：三個分類器在三個測試集上的準確率。只有運動模型在場景不再與標籤一致時仍然撐得住。

運算量與記憶體

對逐影格的網路，一段 T' 張影格的片段，成本大約是一張影像的 T' 倍；對會混合影格的模型則更多。對一個卷積，設輸入通道 C_{in}、輸出通道 C_{out}、核大小 $k_t \times k_h \times k_w$（時間 × 高 × 寬），在 $T \times H \times W$ 的輸入上用步幅 1 與「same」補零，

$$\text{params} = C_{\text{in}}\, C_{\text{out}}\, k_t k_h k_w, \qquad \text{MACs} = \text{params} \times THW,$$

其中 MACs 是乘加運算（multiply–accumulate）的次數（每次一個乘法加一個加法）。因此 $3 \times 3 \times 3$ 的核，參數量與運算量都是對同樣影格施加 3×3 核的 3 倍。對 Transformer 來說，問題在於 token 數 N：自注意力要比較 N^2 對查詢–鍵，而 N 與影格數成線性成長。下一節的程式碼會把這兩者量化出來。

如何讓影像網路學會時間

白話版。影片模型透過池化、光流、時間卷積、多影格率或時空注意力結合影格特徵。

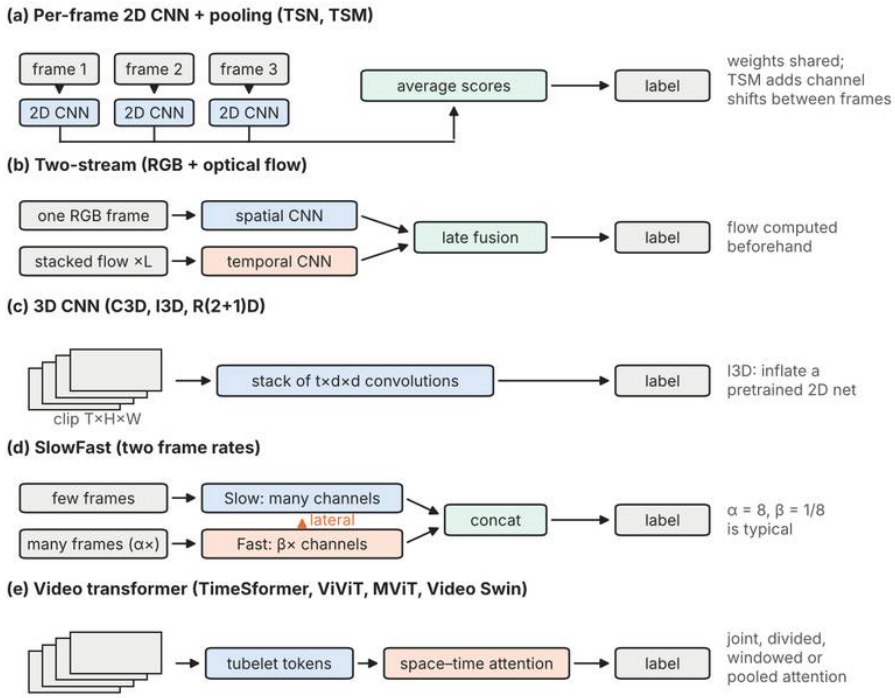


圖 3 — 把時間加進影像網路的五種方式。(*a*)–(*d*) 屬於卷積式；(*e*) 屬於注意力式。實際的系統會組合它們，例如雙流 *I3D*，或使用 *SlowFast* 式多尺度特徵的 *Transformer*。

逐影格網路與後期融合

最簡單的影片模型，是把影像分類器 f 套用在 K 張影格 $x_{t_1}, \dots, x_{t_K}$ 上，再把類別分數平均：

$$\hat{y} = \operatorname{softmax}\Big(\frac{1}{K}\sum_{k=1}^{K} f(x_{t_k})\Big),$$

其中 $f(x) \in \mathbb{R}^C$ 是單張影格的類別分數向量（logits），C 是類別數。這個模型對順序視而不見：打亂影格順序，答案不變，所以它分不出「打開」與「關上」。它在偏重場景的資料集上仍是很強的基準，而且成本低，可以從任何預訓練的影像骨幹開始（DLCV 02）。在逐影格特徵上接 LSTM [11] 這類循環網路，可以加入順序；這種設計在 ML 04 討論。

雙流網路：外觀加光流

白話版。雙流模型結合彩色影格的外觀與光流場的運動資訊。

Simonyan 與 Zisserman 的雙流 ConvNet [12] 有一條**空間流**，是對單張 RGB 影格的影像 CNN，以及一條**時間流**，是輸入為 L 對連續影格之稠密光流的 CNN。每個光流場 d_t 有水平分量 d_t^x 與垂直分量 d_t^y，所以影格 τ 的堆疊輸入有 $2L$ 個通道：

$$I_\tau(u,v,2k-1) = d^x_{\tau+k-1}(u,v), \qquad I_\tau(u,v,2k) = d^y_{\tau+k-1}(u,v), \qquad k = 1,\dots,L,$$

其中 (u,v) 是像素位置，$d^x_t(u,v)$ 與 $d^y_t(u,v)$ 是該像素從第 t 張到第 $t+1$ 張影格的水平與垂直位移。兩條流分開訓練，在後期融合：對 softmax 分數取平均，或在堆疊的分數上用線性 SVM [12]。這篇論文顯示，即使訓練資料有限，作用在堆疊光流上的 ConvNet 表現也很好，而且兩條流是互補的 [12]。

它的弱點是成本：必須在訓練前先為每張影格計算（通常還要儲存）稠密光流，用的是傳統求解器或光流網路（用光流做扭曲見 DLCV 09A.3）。光流也不是針對辨識任務學出來的。Temporal Segment Networks [13] 與 I3D [14] 仍使用光流流；SlowFast（見下）後來直接從原始影格端到端學出運動，不用光流，其作者在實驗上發現它比雙流方法更有效 [15]。

3D 卷積：C3D 與 I3D 膨脹

白話版。讓每個濾波器都變成一個小立方體，不只在空間上滑動，也在時間上滑動。這樣網路就能直接從像素學到運動偵測器，例如「一條向右移動的亮邊」。

核為 $w \in \mathbb{R}^{k_t \times k_h \times k_w}$ 的 3D 卷積，對一個輸入通道與一個輸出通道，計算

$$y(t,i,j) = \sum_{a=0}^{k_t-1} \sum_{b=0}^{k_h-1} \sum_{c=0}^{k_w-1} w(a,b,c)\, x(t+a, i+b, j+c),$$

其中 $x(t,i,j)$ 是第 t 張影格、第 i 列、第 j 欄的輸入（實際上會對所有輸入通道求和，邊界也要補零）。**C3D** [16] 在一個大型影片資料集上訓練了由這種層組成的網路，發現所有層都用小的 $3 \times 3 \times 3$ 核的同質架構是最好的之一；用它的特徵加線性分類器，在數個基準上都很強。

從頭訓練 3D 網路所需的資料，遠比 2015 年有的多。Carreira 與 Zisserman 在 2017 年同時解決了這兩個問題 [14]。他們提出了 **Kinetics**，一個含有 400 個人類動作類別的資料集，每個類別至少 400 段 YouTube 片段，每段約 10 秒 [17]；以及 **Inflated 3D ConvNet（I3D）**：拿一個成功的 2D 影像架構，把每個 $N \times N$ 的濾波器與池化核膨脹成 $N \times N \times N$。為了從 ImageNet [18] 預訓練的權重開始，他們讓膨脹後的網路在「無聊影片」（同一張影像在時間上重複）上的反應，與 2D 網路在該影像上的反應相同：把每個 2D 濾波器沿時間重複 N 次，再除以 N [14]。設 w_{2D} 為預訓練的濾波器，

$$w_{3D}(a,b,c) = \frac{1}{N}\, w_{2D}(b,c), \qquad a = 0,\dots,N-1,$$

所以對於在時間上為常數的輸入，N 個時間分接點加起來，恰好等於 2D 的反應。下面的程式碼用數值驗證這一點。

```
import numpy as np
from scipy.ndimage import correlate

rng = np.random.default_rng(0)
w2d = rng.standard_normal((3, 3))            # a "pretrained" 2D filter
N = 3                                        # temporal size of the inflated filter
w3d = np.repeat(w2d[None] / N, N, axis=0)    # (N, 3, 3): copy N times, divide by N

img = rng.standard_normal((32, 32))
```

```
boring = np.repeat(img[None], 8, axis=0)     # an 8-frame "video" of one still image

r2d = correlate(img, w2d, mode="nearest")
r3d = correlate(boring, w3d, mode="nearest")  # 3D filtering over (t, y, x)
print("max |3D response on any frame - 2D response| =", np.abs(r3d - r2d[None]).max())
```

```
max |3D response on any frame - 2D response| = 1.7763568394002505e-15
```

差異在浮點數捨入的等級。接著，訓練會在濾波器學習運動的過程中，把它們從「在時間上為常數」推開。論文的最終模型是**雙流 I3D**：一個 I3D 吃 RGB、一個吃光流，預測取平均，因為作者仍然覺得光流流很有價值 [14]。在 Kinetics 上預訓練後，I3D 在小型的 UCF101 與 HMDB51 基準上大幅進步 [14]，這讓「在 Kinetics 上預訓練、再到目標資料上微調」成為影片的標準配方。同樣的膨脹想法也出現在影片生成器中（DLCV 12A），而 I3D 特徵是 Fréchet Video Distance [19] 背後的特徵網路（DLCV 12A）。

僅線上版：*Quo Vadis：I3D 與 Kinetics 資料集（Carreira 與 Zisserman，2017）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/papers/quo-vadisi3d-與-kinetics-資料集 carreira-與-zisserman2017/*

分解式 (2+1)D 卷積

白話版。立方體形的濾波器可以拆成兩個比較便宜的步驟：先用一個平面濾波器在每張影格內部看，再用一個細長的濾波器在每個位置上跨影格看。只要中間寬度選對，拆開後參數量不變，卻多了一個非線性，也比較容易訓練。

Tran 等人 [20] 在殘差網路中比較了幾種時空卷積：只用 2D、全部用 3D、混合式（只在早期層用 3D，或只在後期層用 3D），以及分解式的 **(2+1)D** 區塊。(2+1)D 區塊把 N_i 個大小為 $N_{i-1} \times t \times d \times d$ 的濾波器，換成 M_i 個大小為 $N_{i-1} \times 1 \times d \times d$ 的空間濾波器、一個 ReLU，以及 N_i 個大小為 $M_i \times t \times 1 \times 1$ 的時間濾波器。這裡 N_{i-1} 與 N_i 是第 i 層的輸入與輸出通道數，t 與 d 是時間與空間的核大小，M_i 是中間空間的寬度，選擇的方式是讓參數量與 3D 層一致 [20]：

$$M_i = \left\lfloor \frac{t\, d^2\, N_{i-1}\, N_i}{d^2\, N_{i-1} + t\, N_i} \right\rfloor .$$

論文提出兩個好處：多出來的 ReLU 讓非線性的數量在相同參數量下加倍，而分解讓最佳化變容易，表現為比同樣大小的 3D 網路有更低的訓練誤差 [20]。它是影像 CNN 中深度可分離技巧（ML 03）在影片上的對應：把一個大運算沿著不同的軸拆成較便宜的運算。

程式碼計算一個 64 到 64 通道的層，在 16 張影格的片段、56×56 特徵圖上的參數量與乘加次數，並順便算出影片 Transformer 的 token 數作為比較。

```
import math

def conv_cost(c_in, c_out, kt, kh, kw, T, H, W):
    """Parameters and multiply-adds of one convolution (stride 1, 'same' padding, no
    ↪  bias)."""
```

```
    params = c_in * c_out * kt * kh * kw
    return params, params * T * H * W

c, t, d = 64, 3, 3                          # channels in = out, temporal and spatial kernel
 ↪  size
T, H, W = 16, 56, 56                        # a 16-frame clip, 56x56 feature maps
M = math.floor(t * d * d * c * c / (d * d * c + t * c))   # R(2+1)D's matched middle width

layers = {
    "2D  (1x3x3), per frame": [conv_cost(c, c, 1, d, d, T, H, W)],
    "3D  (3x3x3)":            [conv_cost(c, c, t, d, d, T, H, W)],
    f"(2+1)D, M={M}":         [conv_cost(c, M, 1, d, d, T, H, W), conv_cost(M, c, t, 1, 1, T,
     ↪  H, W)],
}
for name, parts in layers.items():
    p = sum(x[0] for x in parts); f = sum(x[1] for x in parts)
    print(f"{name:24s} params {p:>8,d}   GMACs {f / 1e9:5.2f}")

# a video transformer's token count with 2x16x16 tubelets on a 16x224x224 clip
n_tok = (16 // 2) * (224 // 16) ** 2
print(f"tokens {n_tok}, joint attention pairs {n_tok**2:,}, "
      f"divided (time, then space) pairs {n_tok * 8 + n_tok * 196:,}")
```

```
2D  (1x3x3), per frame   params   36,864   GMACs  1.85
3D  (3x3x3)              params  110,592   GMACs  5.55
(2+1)D, M=144            params  110,592   GMACs  5.55
tokens 1568, joint attention pairs 2,458,624, divided (time, then space) pairs 319,872
```

在輸入與輸出寬度相等時，$M = 144$ 使 (2+1)D 區塊的參數量與運算量恰好與 3D 層相同；它的好處在表達能力與最佳化，而不在成本。選擇較小的 M，就可以得到更便宜的拆法。圖 4 畫出這個分解。

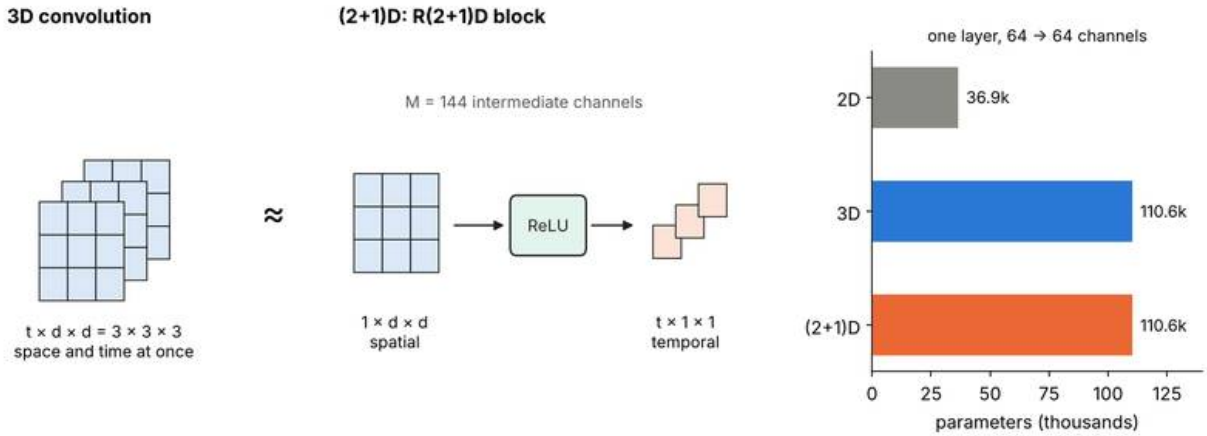


圖 *4 — 3D* 卷積與它的 *(2+1)D* 分解。用上面公式得到的匹配中間寬度 M，兩者的參數量相同；*(2+1)D* 區塊在空間與時間步驟之間多了一個非線性。

僅線上版：選擇 $(2+1)D$ 區塊的中間寬度 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/notes/選擇-21d-區塊的中間寬度/*

低成本的時序建模：TSN 取樣與時序位移模組

白話版。兩個技巧讓普通的 2D 網路也能便宜地處理長影片。TSN 看的是分散在整支影片的幾張影格，而不是某一刻附近的許多影格。TSM 讓相鄰影格在網路內部交換少數幾個特徵通道，幾乎不花成本就能跨時間混合資訊。

Temporal Segment Networks（TSN） [13] 把影片分成 K 個等長的區段，從每段取一小段短片段（訓練時取隨機位置），用同一個網路替每個片段打分，再用**區段共識**（segmental consensus）合併分數，例如在 softmax 之前取平均。損失是對共識計算的，所以是整支影片在監督網路，而不只是一個短視窗 [13]。論文的預設是 $K = 3$ 個區段 [13]。這種**稀疏取樣**用固定且少量的影格涵蓋長距離結構（圖 5）。

```
import numpy as np

def tsn_indices(n_frames, K, train, rng=None):
    """TSN-style sparse sampling: split the video into K equal segments, take one frame
    per segment (random position when training, the segment centre when testing)."""
    edges = np.linspace(0, n_frames, K + 1)
    if train:
        return np.floor(edges[:-1] + rng.random(K) * np.diff(edges)).astype(int)
    return np.floor((edges[:-1] + edges[1:]) / 2).astype(int)

def dense_clip(n_frames, length, stride, start):
    """A dense clip: `length` frames, `stride` apart, starting at `start`."""
    return start + stride * np.arange(length)

rng = np.random.default_rng(0)
n = 300                                          # a 10-second video at 30 fps
print("TSN train:", tsn_indices(n, 8, True, rng))
print("TSN test :", tsn_indices(n, 8, False))
print("dense 8x8:", dense_clip(n, 8, 8, start=100))   # 8 frames, every 8th frame
# multi-view testing: 10 clip start positions spread uniformly over the video
starts = np.linspace(0, n - 8 * 8, 10).astype(int)
print("10 clip starts:", starts)
```

```
TSN train: [ 23  47  76 113 180 221 247 289]
TSN test : [ 18  56  93 131 168 206 243 281]
dense 8x8: [100 108 116 124 132 140 148 156]
10 clip starts: [  0  26  52  78 104 131 157 183 209 236]
```

八張稀疏影格橫跨整支影片；稠密片段則以細節涵蓋大約 2 秒。（測試時取區段中心，是後來實作中常見的選擇；原始 TSN 論文是用 25 張影格、各做 10 個裁切來測試 [13]。）稠密片段是 3D 網路與 SlowFast 所吃的輸入；最後一行，分散在整支影片的十個片段，就是下面評估一節要討論的多視角測試協定。

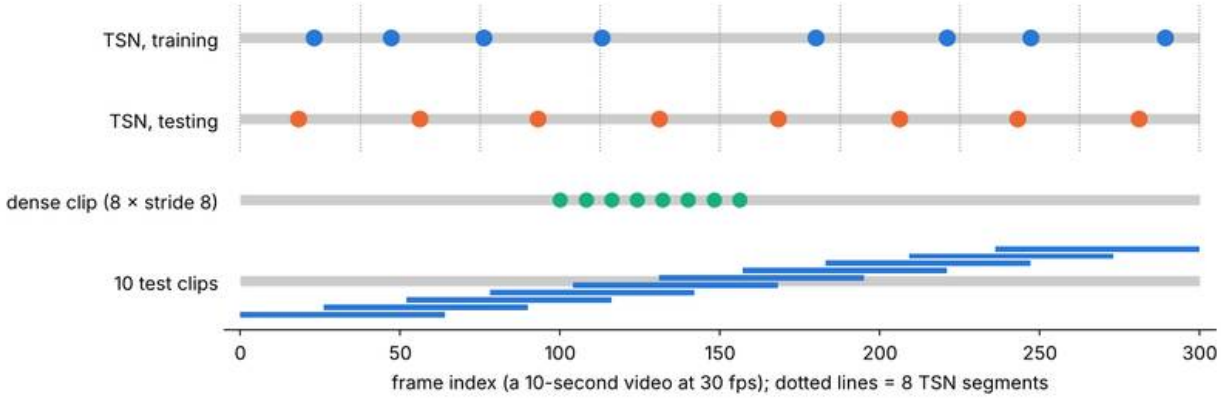


圖 5 — 影格取樣，來自上面的程式碼：*TSN* 式取樣每個區段取一張影格（訓練時隨機，測試時取中心）；稠密片段以固定步幅取連續的影格；多視角測試則平均分散在整支影片的數個片段的預測。

時序位移模組（Temporal Shift Module, TSM） [21] 在 2D 網路中加入真正的時間交互作用，而且完全不增加參數，也不增加乘加運算。在每個殘差區塊內，特徵張量的一部分通道沿時間向前位移一張影格，另一部分向後位移一張影格，其餘留在原處；接下來的 2D 卷積就會混合來自三張不同影格的通道。位移太多通道，延遲（資料搬移）與準確率（每張影格剩下的空間資訊變少）都會變差，所以論文位移 1/4 的通道（每個方向 1/8），並把位移放在殘差分支上 [21]。對即時影片，單向版本只從過去位移到現在的影格 [21]。

僅線上版：時序位移是一種稀疏的時序卷積 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/notes/時序位移是一種稀疏的時序卷積/*

SlowFast：兩種影格率

白話版。東西是什麼，變化得很慢；東西怎麼動，可能變化得很快。所以用兩條路徑：一條很重、只看少數影格來理解場景；一條很輕、看很多影格來捕捉運動。

SlowFast 網路 [15] 有一條 **Slow 路徑**，以較大的時間步幅 τ 取樣 T 張影格；以及一條 **Fast 路徑**，以步幅 τ/α 取樣 αT 張影格，所以在時間上比 Slow 路徑密 α 倍。Fast 路徑只有 Slow 路徑通道數的 β 倍。典型的值是 $\alpha = 8$ 與 $\beta = 1/8$ [15]。由於一層的成本大致與通道數的平方成正比，在論文的實例中，Fast 路徑只占總運算量的大約 20% [15]。**橫向連接**（lateral connection）在每個階段把 Fast 的特徵送進 Slow 路徑；為了讓形狀匹配，它們把 αT 張 Fast 影格轉換成 T 張，例如用時間步幅的 3D 卷積，這是論文的預設做法 [15]。

有兩個設計細節帶有通用的教訓。Slow 路徑只在最後兩個階段使用大於 1 的時間核，因為作者發現，當影格步幅很大時，早期層的時間卷積反而有害：除非空間感受野也很大，否則快速移動的內容在相隔很久的時間步之間幾乎沒有相關性 [15]。而且 SlowFast **不使用光流**：它直接從原始影格端到端學習運動 [15]。它在 Kinetics、Charades 與 AVA 的分類與偵測上報告了很強的結果 [15]。

僅線上版：*SlowFast* 網路（*Feichtenhofer* 等人，*2019*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/papers/slowfast-網路 feichtenhofer-等人 2019/*

還有兩個相關的想法補完了 CNN 家族。**Non-local 區塊** [22] 把每個位置的反應，算成對所有時空位置的加權和，也就是插在 3D CNN 中的一種注意力層，用來捕捉長距離相依。**X3D** [23] 從一個很小的 2D 影像網路出發，一次沿一個軸（空間、時間、寬度與深度）擴張它，來尋找低成本且準確的網路；一個令人意外的發現是，高時空解析度的網路可以非常窄 [23]。

影片 Transformer：TimeSformer、ViViT、MViT 與 Video Swin

白話版。把片段切成小小的時空塊，每塊變成一個 token，讓 Transformer 決定哪些塊該看哪些塊。困難在於一段片段有幾千個 token，所以各種設計主要的差別，在於每個 token 可以注意哪些 token。

Vision Transformer [24]（ML 05）把影像變成小塊。影片 Transformer 則在時空中做同樣的事。用 **tubelet 嵌入**（tubelet embedding）[25]，把互不重疊、大小為 $t \times p \times p$ 像素的管狀區塊做線性投影（等價於步幅等於核大小的 3D 卷積），於是 $T \times H \times W$ 的片段變成

$$N = \frac{T}{t} \cdot \frac{H}{p} \cdot \frac{W}{p}$$

個 token，其中 t 是 tubelet 的影格長度，p 是小塊邊長的像素數。例如 VideoMAE 的 ViT-Base，吃 $16 \times 224 \times 224$ 的片段與 $2 \times 16 \times 16$ 的方塊，得到 $8 \times 196 = 1568$ 個 token [10]。**聯合**時空注意力要比較全部 N^2 對，上面的程式碼算出每層約 246 萬對。有四種設計可以馴服它：

- **TimeSformer** [26] 比較了幾種自注意力方案，發現它試過的方案中以**分離式注意力**（divided attention）最好：在每個區塊中，一個 token 先注意其他影格中相同空間位置的 token（時間注意力），再注意自己這張影格中的 token（空間注意力）[26]。在上面的設定中，這大約是 32 萬對，而不是 246 萬對。它也能處理超過一分鐘的片段 [26]。聯合與分離式注意力的成本比，在 DLCV 12A 中推導，那裡也說明了為什麼影片生成器常偏好完整注意力。
- **ViViT** [25] 提出四種模型：聯合時空注意力；**分解式編碼器**（對每張影格用一個空間 Transformer，再對影格嵌入用一個時間 Transformer，是一種後期融合）；每個區塊內的分解式自注意力；以及分解式點積注意力，其中一半的注意力頭注意空間，一半注意時間。它也顯示了如何正則化這些模型，並用預訓練的影像模型初始化它們，使它們可以在相對小的資料集上訓練 [25]。
- **MViT** [27] 帶回 CNN 的多尺度階層：各階段從高時空解析度、少量通道開始，逐步增加通道並降低解析度，並使用**池化注意力**（pooling attention），也就是在注意力內部對查詢、鍵與值序列做池化 [27]。
- **Video Swin Transformer** [28] 主張**局部性**：注意力只在局部 3D 視窗內計算，視窗在相鄰層之間位移，讓資訊可以跨過視窗邊界。它延伸了影像的 Swin Transformer [29]（DLCV 02），並從預訓練的影像權重開始，在速度與準確率的取捨上優於全域注意力 [28]。

注意力不是建模長片段的唯一方式。**VideoMamba** [30] 把 Mamba 狀態空間模型 [31]（其成本與 token 數成線性成長）調整到影片上；底層的選擇性掃描在 ML 04 說明。

不用標籤學習影片表示

白話版。標註影片又慢又貴，但沒有標籤的影片非常多。所以把片段的大部分遮起來，訓練網路把它補回來；要補得好，它就必須學會東西長什麼樣子、怎麼移動。

VideoMAE [10] 把遮罩式自編碼（MAE [32]，見 ML 06）用到影片上。它遮住時空方塊，只用 ViT 編碼可見的部分，再用輕量解碼器重建被遮住部分的像素。有兩件事是影片特有的。第一，相鄰影格高度冗餘，所以遮罩比例可以極高：90% 到 95% 仍然很有效 [10]。第二，如果每張影格獨立抽遮罩，被遮住的小塊常常可以從相鄰影格的同一位置直接複製過來，這種**資訊洩漏**（information leakage）讓任務變得太簡單。所以 VideoMAE 使用**管狀遮罩**（tube masking）：每張影格都用同一個空間遮罩 [10]。下面的片段量測這種洩漏。

```
import numpy as np

rng = np.random.default_rng(0)
T, P, ratio = 8, 196, 0.9               # 8 token frames of 14x14 patches, 90% masked

def leak(mask):
    """Fraction of masked tokens whose same-position token in a neighbouring frame is
     ↪  visible."""
    vis = ~mask
    nb = np.zeros_like(mask)
    nb[1:] |= vis[:-1]; nb[:-1] |= vis[1:]
    return (mask & nb).sum() / mask.sum()

random_mask = rng.random((T, P)) < ratio                  # independent per token
tube_mask = np.repeat((rng.random(P) < ratio)[None], T, 0) # same spatial mask in every frame
print(f"random masking: {random_mask.mean():.2f} masked, leak {leak(random_mask):.2f}")
print(f"tube masking  : {tube_mask.mean():.2f} masked, leak {leak(tube_mask):.2f}")
```

```
random masking: 0.90 masked, leak 0.16
tube masking  : 0.89 masked, leak 0.00
```

使用獨立遮罩時，大約每六個被遮住的 token 中，就有一個在相鄰影格有可見的同位置 token（對中間的影格，期望值為 $1 - 0.9^2 = 0.19$；第一張與最後一張影格只有一個鄰居）。管狀遮罩完全消除了這個捷徑，這正是重點：預訓練任務只有在它的捷徑都被堵住時才有用。VideoMAE 也報告，在只有幾千支影片的小資料集上，不需額外資料就能有很強的結果；發現對這類預訓練而言，資料品質比資料數量更重要；並指出預訓練資料與目標資料之間的領域偏移是個重要的議題 [10]。**VideoMAE V2** [33] 以**雙重遮罩**（dual masking）把這個方法擴展到十億參數的 ViT，雙重遮罩也會遮住解碼器的輸入，以節省運算。

第二條路線預測的是特徵而不是像素。**V-JEPA** [34] 只用特徵預測的目標來訓練影片模型，不用預訓練的影像編碼器、文字、負樣本或重建，使用大約 200 萬支影片，並用凍結的骨幹在以運動為主與以外觀為主的任務上評估它們 [34]。它的後繼者 V-JEPA 2 [35] 與 JEPA 的想法，在 ML 06 與 DLCV 13 中討論。

影片與語言（簡介）

白話版。不使用固定的動作名稱清單，而是用自由文字描述影片。把影片和句子放在同一個空間中的模型，可以用描述來搜尋影片；把影片 token 送進語言模型的模型，可以回答關於影片的問題。

檢索。 CLIP 式的影像–文字模型 [36]（ML 10）可以遷移到影片：**CLIP4Clip** [37] 把 CLIP 端到端地遷移到文字–影片檢索，並研究影像特徵是否足夠、在影片–文字資料上做後預訓練（post-pretraining）有什麼幫助，以及如何建模影格之間的相依關係，使用的基準如 **MSR-VTT**，這是一個包含 1 萬段網路影片片段與 20 萬組片段–句子配對的集合 [38]。**影片基礎模型**如 **InternVideo2** [39]，在一個漸進式的訓練方案中結合了遮罩式影片建模、影片–文字對比學習與下一個 token 預測，影片編碼器最大達 60 億個參數 [39]。

問答。多模態大型語言模型（ML 10）接受取樣出來的影格作為視覺 token。長影片會讓它們吃力：**Video-MME** [40] 在 900 支人工標註、長度從 11 秒到 1 小時的影片（2,700 組問答配對）上評估它們，字幕與音訊是可選的輸入，並發現處理更長的序列仍是弱點 [40]。一個實務上的提醒，類比場景偏差：在把時間理解的功勞歸給模型之前，先檢查問題是否光靠單張影格、或光靠問題文字就能回答；單影格基準是很便宜的健全性檢查。

在時間與空間中定位動作

白話版。真實的影片沒有被裁剪過。一支料理影片包含「切」、「攪拌」與「倒」，中間夾著很長的無聊片段。時序定位找出每個動作從哪裡開始、到哪裡結束；時空偵測還會說是哪個人在做。

時序動作定位

這個任務沿著時間軸呼應物件偵測（DLCV 03），方法也是如此。片段層級的骨幹（常常是凍結的，例如 I3D 或 SlowFast 特徵）沿時間產生一個特徵序列；定位頭再把它轉成帶分數的區段。

- **以提案為基礎（兩階段）**。先產生不分類別的候選區段，再對它們分類。**BMN** [41] 把一個提案表示成一對開始與結束邊界，並在一張**邊界匹配信心圖**（boundary-matching confidence map）中一次為所有稠密分布的配對打分；結合動作分類器後就得到偵測結果 [41]。
- **無錨框、單階段**。 **ActionFormer** [42] 對時間上的每個時刻分類，並迴歸到動作開始與結束的距離，不需要提案，也不需要預先定義的錨框視窗。它使用具有局部自注意力的多尺度 Transformer 與一個輕量解碼器 [42]，是 DLCV 03 無錨框偵測器在時間上的對應。重複的區段用時間上的非極大值抑制移除，與框的做法相同（ActionFormer 用的是軟式變體 [42]）。

時序 IoU 與 mAP。對預測區段 $\hat{g} = [\hat{s}, \hat{e}]$ 與真實區段 $g = [s, e]$，

$$\mathrm{tIoU}(\hat{g}, g) = \frac{\max\left(0, \min(\hat{e}, e) - \max(\hat{s}, s)\right)}{(\hat{e} - \hat{s}) + (e - s) - \max\left(0, \min(\hat{e}, e) - \max(\hat{s}, s)\right)},$$

其中分子是重疊的長度，分母是聯集的長度。對某一類別與門檻 θ，把預測依分數排序；若某個預測與該類別尚未被配對的真實區段的 tIoU 至少為 θ，它就是真陽性。平均精確率（average precision, AP）是所得精確率–召回率曲線下的面積，mAP 是對各類別的 AP 取平均，而**平均**

mAP 則再對多個門檻取平均。常見的有兩種慣例：THUMOS14 對 tIoU $\{0.3, 0.4, \dots, 0.7\}$ 取平均，ActivityNet 1.3 則對 $\{0.5, 0.55, \dots, 0.95\}$ 取平均 [42]。程式碼在合成的偵測結果上計算單一類別的 AP（找到 85% 的動作，帶有邊界誤差，再加上隨機的誤報）。

```
import numpy as np

def tiou(seg, segs):
    """Temporal IoU of one segment [s, e] with an array of segments (n, 2)."""
    inter = np.clip(np.minimum(seg[1], segs[:, 1]) - np.maximum(seg[0], segs[:, 0]), 0, None)
    union = (seg[1] - seg[0]) + (segs[:, 1] - segs[:, 0]) - inter
    return inter / union

def average_precision(preds, gts, thr):
    """preds: list of (video, start, end, score); gts: dict video -> (m, 2) array.
    Greedy matching in score order; each ground truth can be matched once."""
    n_gt = sum(len(g) for g in gts.values())
    used = {v: np.zeros(len(g), bool) for v, g in gts.items()}
    tp = []
    for v, s, e, _ in sorted(preds, key=lambda p: -p[3]):
        ious = tiou(np.array([s, e]), gts[v])
        ious[used[v]] = -1                      # already matched ground truths are off
        ↪ limits
        j = int(np.argmax(ious))
        hit = ious[j] >= thr
        if hit:
            used[v][j] = True
        tp.append(hit)
    tp = np.array(tp, float)
    recall = np.cumsum(tp) / n_gt
    precision = np.cumsum(tp) / np.arange(1, len(tp) + 1)
    precision = np.maximum.accumulate(precision[::-1])[::-1]   # monotone envelope
    return np.sum(np.diff(np.r_[0, recall]) * precision)

rng = np.random.default_rng(0)
gts, preds = {}, []
for v in range(20):                             # 20 untrimmed videos, 1-3 actions each
    starts = np.sort(rng.uniform(0, 100, rng.integers(1, 4)))
    g = np.c_[starts, starts + rng.uniform(4, 15, len(starts))]
    gts[v] = g
    for s, e in g:
        if rng.random() < 0.85:                 # the detector finds 85% of actions...
            j = rng.normal(0, 0.2 * (e - s), 2)  # ...with boundary errors
            preds.append((v, s + j[0], e + j[1], rng.uniform(0.5, 1.0)))
    for _ in range(rng.integers(0, 3)):         # plus some false alarms
        s = rng.uniform(0, 100)
        preds.append((v, s, s + rng.uniform(3, 12), rng.uniform(0.0, 0.7)))

thrs = [0.3, 0.4, 0.5, 0.6, 0.7]
aps = [average_precision(preds, gts, t) for t in thrs]
print("AP at tIoU:", "  ".join(f"{t}: {a:.3f}" for t, a in zip(thrs, aps)))
print(f"average over [0.3:0.1:0.7]: {np.mean(aps):.3f}")
```

```
AP at tIoU: 0.3: 0.731  0.4: 0.696  0.5: 0.696  0.6: 0.604  0.7: 0.284
```

```
average over [0.3:0.1:0.7]: 0.602
```

（玩具數字。）到 tIoU 0.5 為止，AP 主要受 15% 漏掉的動作與誤報所限制；到 0.7 時它崩落，因為約為動作長度 20% 的邊界誤差已經不夠好。動作的邊界本來就模稜兩可（「倒」是從什麼時候開始的？），這就是為什麼嚴格的門檻即使對人來說也很難，也是為什麼基準會報告多個門檻。

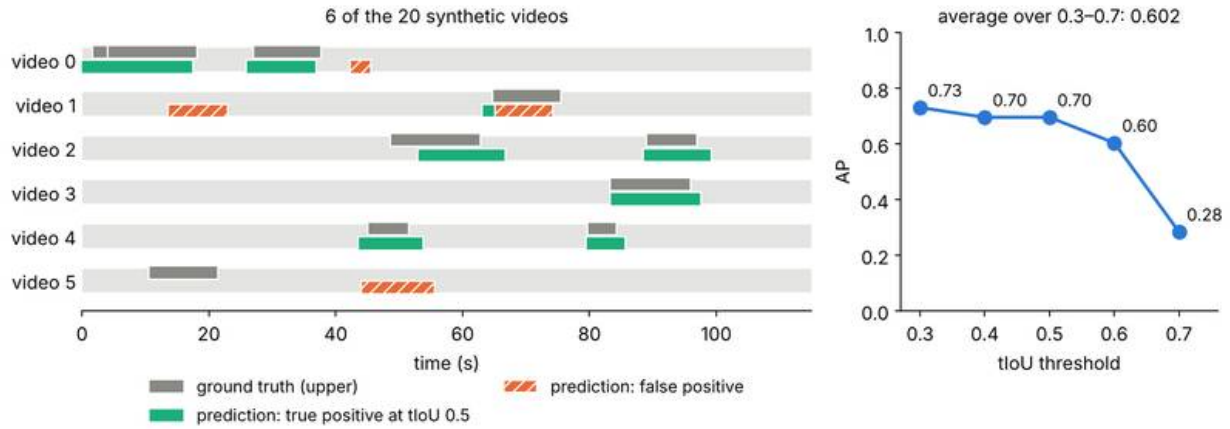


圖 *6* — 上面程式碼的時序 *AP* 玩具實驗。左：*20* 支影片中六支的真實動作（灰）與預測，依是否在 *tIoU 0.5* 下配對成功而上色。右：*tIoU* 門檻變嚴格時，*AP* 下降。

僅線上版：時序定位的平均精確率，一步一步來 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/notes/時序定位的平均精確率一步一步來/*

時空動作偵測

這裡模型必須說出哪個人做了哪個動作。在 **AVA** [43] 中，從 15 分鐘電影片段以 1 Hz 取樣出的關鍵影格裡，每個人都有一個框，以及來自 80 個原子動作（14 個姿勢類別、49 個人–物互動與 17 個人–人互動）的一個或多個標籤 [43]。常見的流程是偵測加上影片分類，如 SlowFast 的 AVA 偵測器 [15]：現成的人物偵測器（DLCV 03）在關鍵影格上提出框，影片骨幹為它周圍的一段片段計算時空特徵圖，對每個框內的特徵做池化，再由每個類別各一個 sigmoid 分類器給出多標籤預測，因為一個人可以同時「站著」、「對話」又「拿著東西」。指標是 **IoU 0.5 下的影格 mAP**（frame-mAP）：對所有關鍵影格偵測，逐動作類別計算 AP，偵測只有在它的框與該類別的某個真實框的 IoU 至少為 0.5 時才算數，再對各類別取平均 [43]。把偵測結果在時間上連成管狀序列，則借用追蹤的關聯方法（DLCV 07）。

骨架式動作辨識

模型也可以不讀像素，而是讀**姿態**：身體關節隨時間變化的 2D 或 3D 座標，可以像 DLCV 05 那樣估計，或用深度感測器擷取，例如含 60 個動作類別的 NTU RGB+D 資料集 [44]。**ST-GCN** [45] 建立一張圖，節點是每張影格中的關節，空間邊沿著骨頭連接，時間邊則連接連續影格中的同一個關節，並在這張圖上做圖卷積 [45]。骨架很精簡，不受衣著與背景影響，因此較不容易出

現場景偏差：在 **Mimetics** 資料集（沒有一般的物件與場景、只做出模仿動作）上，以姿態為基礎的模型比 3D CNN 表現好得多 [46]。代價是骨架會漏掉物件（「喝」與「吃」），而且依賴姿態估計器的誤差。

僅線上版：骨架序列上的圖卷積（*ST-GCN*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/notes/骨架序列上的圖卷積 st-gcn/*

訓練與評估影片分類器

白話版。你怎麼挑影格、測試時平均多少個視角、報告在哪個資料集上，對數字的影響不亞於許多架構上的改動。比較兩篇論文之前，先讀它們的協定。

取樣與訓練

有三個選擇決定了輸入：**影格數** T'、影格之間的**步幅**（影格率），以及稠密與稀疏（TSN 式）取樣。以運動為中心的資料集獎勵稠密、細緻的取樣；以場景為中心的資料集可以容忍稀疏取樣。例如 VideoMAE 在 Something-Something V2 上使用 TSN 式均勻取樣，在 Kinetics-400 上使用稠密取樣 [10]。空間資料增強與影像相同（隨機縮放裁切、翻轉），但有一個影片特有的陷阱：**水平翻轉會改變標籤**，對與方向相關的類別，如「把東西從左推到右」就是如此。其餘訓練則依 DLCV 02。初始化通常來自影像模型（膨脹，或預訓練的 ViT），或來自自監督的影片模型。

多視角測試

測試時，一段片段很少涵蓋整支影片，所以預測會在 V_t 個時間片段與 V_s 個空間裁切上取平均：

$$\hat{p}(y \mid \text{video}) = \frac{1}{V_t V_s} \sum_{i=1}^{V_t} \sum_{j=1}^{V_s} \text{softmax}\left(f(x_{ij})\right)_y,$$

其中 x_{ij} 是在第 j 個空間位置裁切的第 i 個片段。例如 SlowFast 沿著影片均勻取樣 10 個片段，每個再取 3 個 256×256 的裁切，共 30 個視角，並把成本報告為**每個視角的 GFLOPs × 視角數**，同時指出一些早期工作用了超過 100 個視角 [15]。VideoMAE 在 Something-Something V2 上用 2 個片段 × 3 個裁切評估，在 Kinetics-400 上用 5 × 3 [10]。總推論成本是每個視角的成本乘以視角數，所以每個視角看起來便宜的模型，實務上可能很貴。兩者都要報告，而且只在相同協定下比較模型。

資料集與基準

資料集	任務	內容（依其作者的描述）
UCF101 [47]	片段分類	101 個動作類別，超過 1.3 萬段片段，27 小時使用者上傳的影片
HMDB51 [48]	片段分類	來自電影與網路影片的 51 個動作類別；規模小且刻意設計得很難

資料集	任務	內容（依其作者的描述）
Kinetics-400 [17]	片段分類	400 個人類動作類別，每類至少 400 段片段，每段約 10 秒，來自 YouTube
Something-Something V2 [9], [10]	片段分類	174 個以運動為中心、定義為描述樣板的類別；約 16.9 萬支訓練與 2 萬支驗證影片
Diving48 [7]	片段分類	超過 18,000 段片段，48 種細緻的跳水類別，為降低靜態偏差而建立
Charades [49]	多標籤分類、定位	267 人錄製的 9,848 支日常室內活動影片，157 個動作類別
THUMOS14 [50]	時序定位	未裁剪的影片，包含沒有目標動作的背景影片
ActivityNet [51]	未裁剪分類、定位	針對未裁剪網路影片中人類活動的大規模基準
AVA [43]	時空偵測	430 段 15 分鐘電影片段中的 80 個原子動作，158 萬個動作標籤
EPIC-KITCHENS-100 [52], [53]	第一人稱辨識、偵測、預期	45 個環境中 100 小時未編排的頭戴式影片，9 萬個動作
NTU RGB+D [44]	骨架／ RGB-D 辨識	超過 5.6 萬個樣本，60 個動作類別，40 位受試者
MSR-VTT [38]	檢索、描述生成	1 萬段網路片段，20 萬組片段–句子配對
Video-MME [40]	多模態 LLM 的影片問答	900 支 11 秒到 1 小時的影片，2,700 組問答配對

UCF101 與 HMDB51 報告在三**個官方訓練／測試切分**上的平均準確率 [12], [13]；Kinetics 與 Something-Something 報告其驗證集上的 top-1 與 top-5；Charades 與 AVA 報告 mAP。有兩個提醒。第一，Kinetics 與 Something-Something 獎勵的東西不同：VideoMAE 的作者形容 Kinetics 的影片大多靜止且與場景有關，而 Something-Something V2 則以運動為中心 [10]。模型的排名可能在兩者之間改變。第二，EPIC-KITCHENS-100 的作者設計了一個「時間考驗」（test of time）：拿在 2018 年蒐集的影片訓練的模型，去測兩年後新蒐集的影片，並設有一個非監督式領域適應挑戰賽 [53]。像這樣的基準，量測的是單一保留切分量測不到的穩健性。

失效模式

白話版。影片模型會以幾種可預測的方式失敗。大多數都是因為模型用了一個剛好在訓練資料上有效的線索。

1. **場景與物件偏差**。模型從網球場預測「打網球」。證據：常見資料集偏向靜態線索的表徵偏差 [7]；模型在沒有一般物件與場景的模仿動作上失敗 [46]。補救方式包括去偏差損失，例如對抗式場景分類器，以及懲罰在人物被遮掉的影片上仍做出高信心預測的損失 [54]、以運動為中心的訓練資料 [9]，以及姿態輸入 [46]。
2. **時序捷徑**。模型忽略影格順序。測試：打亂或反轉影格。如果準確率幾乎沒變，模型（或基準）就沒有用到時間；Sevilla-Lara 等人就是這樣建立他們的時序類別子集 [8]。
3. **時間脈絡太短**。大多數片段模型一次只看幾秒。以分鐘計的動作（「組裝家具」）或關於整部電影的問題，需要長距離記憶，即使對大型多模態模型，這仍是弱點 [40]。

4. **領域偏移**。第三人稱的 YouTube 影片、第一人稱的廚房、夜間監視器與醫療內視鏡看起來截然不同。在其中一種上訓練的模型，遷移到另一種時並不完美；EPIC-KITCHENS-100 用沒看過的廚房與較晚錄製的影片量測這件事 [52], [53]。DLCV 02 的分佈偏移測試也適用。
5. **含糊的標籤與邊界**。動作會重疊、開始與結束模糊，而且取決於意圖。多標籤資料集（Charades、AVA）與多個 tIoU 門檻承認了這一點；單標籤準確率則把它藏起來。
6. **不公平的比較**。視角數、影格數、解析度與預訓練資料各不相同，讓報告的數字無法比較 [1]。務必檢查結果表中的協定那一行。

兩個便宜的診斷就能抓出大部分問題：**單影格基準**（有多少分數根本不需要時間？）與**打亂影格測試**（順序對模型有影響嗎？）。在信任影片模型之前，先把兩個都跑過。

現代觀點

綜述論文

五篇綜述從不同角度涵蓋這個領域。**Zhu 等人** [1] 回顧了超過 200 篇深度動作辨識的論文，介紹 17 個資料集，依時間順序呈現模型（早期嘗試、雙流網路、3D 卷積、高效率運算模型），在同一套協定下以公開的程式碼對常用方法做基準測試，並指出這個領域的主要挑戰：長距離時序建模、運算成本，以及因資料集與協定不同而無法比較的結果。**Kong 與 Fu** [2] 同時涵蓋辨識（為完整的動作貼標籤）與預測（從動作的前段預期動作），包含資料集與評估協定。**Selva 等人** [3] 依輸入如何 token 化、如何減少冗餘並重新引入歸納偏差、如何捕捉長期動態，以及如何訓練，來整理影片 Transformer，並發現它們在動作分類上即使運算成本較低，也勝過 3D CNN。**Madan 等人** [4] 把超過 200 個影片基礎模型分成三組來分析（改造成影片用途的影像模型、以影片為基礎的模型，以及跨模態的通用模型），並報告在大多數影片理解任務上，影像模型勝過以影片為基礎的模型，而通用的多模態模型表現最好。**Tang 等人** [5] 依影片如何送到語言模型（影片分析器、影片嵌入器，或兩者皆有），以及語言模型的角色（摘要器、管理者、文字解碼器、迴歸器或隱藏層），來分類以大型語言模型做影片理解的方法。

僅線上版：「現代觀點」綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-understanding/papers/現代觀點綜述導讀/*

2023–2026 年的最新進展

- **自監督預訓練是常見的起點**。遮罩式影片建模（VideoMAE [10]、VideoMAE V2 [33]）與特徵預測（V-JEPA [34]）產生的編碼器，可以為下游任務微調或探測；VideoMAE V2 的作者稱規模是這類基礎模型的首要因素 [33]。
- **影片基礎模型統一了各種任務**。如 InternVideo2 [39] 的模型，用遮罩式建模、影片–文字對比與下一個 token 預測訓練同一個編碼器，並在辨識、檢索與對話上評估。Madan 等人發現改造後的影像模型依然非常有競爭力 [4]，這提醒我們，影片理解的很大一部分仍然是外觀。
- **多模態 LLM 接手開放式理解**。影片問答、描述生成與時序定位（grounding），越來越多由帶有視覺 token 的語言模型處理，並在 Video-MME [40] 等基準上評估。瓶頸在於長影片的視覺 token 數量。
- **高效率的長距離模型**。成本呈線性的序列模型如 VideoMamba [30]，以及串流式設計（精神上類似線上 TSM [21]），針對的是長影片與即時影片。

- **定位建立在預訓練編碼器之上**。時序定位器執行在預訓練影片模型的特徵上；例如 ActionFormer 在 THUMOS14 上使用在 Kinetics 上預訓練的雙流 I3D 特徵 [42]，所以更好的編碼器與更好的定位頭會一起提升結果。

尚未解決的問題

1. **真正的時序推理**。光靠單張影格或語言先驗不夠的基準，以及會使用順序、因果與持續時間的模型。
2. **長影片**。數小時影片的記憶體與 token 預算，以及對長距離理解的評估。
3. **細粒度與第一人稱動作**。手、小物件與意圖；跨視角與環境的遷移。
4. **高效率與線上辨識**。在裝置上的低延遲，並誠實報告成本（FLOPs × 視角數、真實延遲）。
5. **評估**。可比較的協定、網路規模預訓練資料對基準的汙染，以及獎勵時間理解而不是外觀的指標。

重點整理

- 依輸出選擇指標：片段標籤、動作區間、人員方框或檢索結果。
- 高分可能反映場景線索，而非運動理解。
- 光流、時間卷積與注意力以不同方式納入時間，應檢查各自能讀取哪些資訊。
- 取樣與多速率分支在時間涵蓋範圍及計算量之間取捨。
- 預訓練若要學到有用結構，遮罩須避免容易的時間資訊洩漏。
- 比較時固定測試視圖與預訓練，再測試單影格基準與打亂影格輸入。

練習

1. **組合模型作弊到什麼程度？**在場景偏差的程式碼中，把特徵標準化（在 pipeline 中使用 `sklearn.preprocessing.StandardScaler`），並讓 `C` 在 0.01、0.1 與 1 之間變化。影格加運動模型的「no cue」與「shift」準確率如何變化？為什麼標準化能幫助它使用運動特徵？

   提示

   運動特徵的數值範圍，比 400 個像素特徵小，所以在 L2 懲罰下，模型覺得把權重分散到許多像素上比較划算。在我們的執行中，標準化後影格加運動模型的「no cue」／「shift」準確率，在 `C = 0.01` 時為 0.752/0.547，在 `C = 0.1` 時為 0.824/0.685，在 `C = 1` 時為 0.842/0.708，仍遠低於只用運動的模型。縮放降低了捷徑，但只要場景在訓練中仍然有預測力，捷徑就不會消失。

2. **手算膨脹**。一個 2D 濾波器 $w_{2D} = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$（水平導數）被膨脹到 $N = 3$ 張影格。寫出 w_{3D}，並計算它在某一位置的反應，分別對 (a) 每張影格都是同一列 $[5, 3, 1]$ 的無聊影片，以及 (b) 三張影格的列分別為 $[5, 3, 1]$、$[3, 1, 0]$ 與 $[1, 0, 0]$ 的影片。兩者的差異告訴你訓練必須改變什麼？

   提示

w_{3D} 有三個時間切片，每個都是 $\frac{1}{3}[1, 0, -1]$。(a) 每個切片給出 $\frac{1}{3}(5-1) = \frac{4}{3}$，總和為 4，與 2D 的反應相同。(b) 三個切片給出 $\frac{4}{3}$、$\frac{3}{3}$ 與 $\frac{1}{3}$，總和為 $\frac{8}{3}$。膨脹後的濾波器是對影格取平均，所以它只能透過模糊來反映運動；要變成運動偵測器，訓練必須讓各時間切片不同（例如第一個與最後一個的正負號相反）。

3. **寬度不相等的 (2+1)D**。對輸入 $N_{i-1} = 64$、輸出 $N_i = 128$ 個通道、$t = 3$、$d = 3$ 的層，計算 M_i、3D 的參數量與 (2+1)D 的參數量。接著選 $M = 64$，計算節省了多少。

   提示

   $M_i = \lfloor 3 \cdot 9 \cdot 64 \cdot 128 / (9 \cdot 64 + 3 \cdot 128) \rfloor = \lfloor 221184 / 960 \rfloor = 230$。3D：$64 \cdot 128 \cdot 27 = 221{,}184$。$M = 230$ 的 (2+1)D：$64 \cdot 230 \cdot 9 + 230 \cdot 128 \cdot 3 = 132{,}480 + 88{,}320 = 220{,}800$（因為向下取整，所以略少）。$M = 64$ 時：$36{,}864 + 24{,}576 = 61{,}440$，約為 3D 層的 28%。

4. **視角與成本**。模型 A 每個視角成本 36 GFLOPs，用 10×3 個視角測試；模型 B 每個視角成本 180 GFLOPs，用 1×3 個視角測試。每支影片哪個比較貴？設計一個公平的比較。

   提示

   A：$36 \times 30 = 1080$ GFLOPs；B：$180 \times 3 = 540$ GFLOPs，所以雖然 A 每個視角便宜五倍，但每支影片的成本卻是兩倍。公平的比較，是在相同的影格與解析度下，對每個模型的數種視角數，畫出準確率對總推論成本的曲線。

5. **打亂影格測試**。擴充場景偏差的程式碼：把每個測試片段的影格順序打亂，每個片段用新的排列（`rng = np.random.default_rng(5)`，然後 `np.stack([x[rng.permutation(8)] for x in X])`），再重新評估三個用有序片段訓練的模型。執行前先預測結果。

   提示

   單影格模型仍然只讀場景，所以幾乎沒變（在我們的執行中，三個測試集上為 0.868 / 0.499 / 0.127）。運動模型每對影格的估計現在有隨機的正負號，掉到隨機猜測的水準（0.522 / 0.500 / 0.466）。影格加運動模型沒有動（0.864 / 0.497 / 0.133），這暴露了它從來沒有依賴順序。一個在打亂後準確率依然不變的模型，並沒有使用時間。

6. **手算 tIoU**。一個真實動作從 10 秒持續到 20 秒。預測：P1 [12, 22]，分數 0.9；P2 [9, 19]，分數 0.8；P3 [30, 35]，分數 0.7。計算各自的 tIoU，在 $\theta = 0.5$ 與 $\theta = 0.75$ 下標出 TP/FP，並計算兩個門檻下的 AP。

   提示

   P1：重疊 8，聯集 12，tIoU 0.667。P2：重疊 9，聯集 11，tIoU 0.818。P3：0。在 0.5 下：P1 是 TP（分數順序中第一個），P2 是重複預測（FP），P3 是 FP；在召回率 1 時精確率為 1，所以 AP = 1。在 0.75 下：P1 是 FP，P2 是 TP（召回率 1 時精確率為 1/2），P3 是 FP；AP = 0.5。在嚴格的門檻下，分數較高的較差框可能讓 AP 損失一半。

參考文獻

1. Y. Zhu, X. Li, C. Liu, M. Zolfaghari, Y. Xiong, C. Wu, Z. Zhang, J. Tighe et al., “A comprehensive study of deep video action recognition,” technical report, arXiv:2012.06567, 2020. arXiv
2. Y. Kong and Y. Fu, “Human action recognition and prediction: A survey,” *International Journal of Computer Vision*, vol. 130, no. 5, pp. 1366–1401, 2022. doi
3. J. Selva, A. S. Johansen, S. Escalera, K. Nasrollahi, T. B. Moeslund and A. Clapés, “Video transformers: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 12922–12943, 2023. doi
4. N. Madan, A. Moegelmose, R. Modi, Y. S. Rawat and T. B. Moeslund, “Foundation models for video understanding: A survey,” arXiv:2405.03770, 2024. arXiv
5. Y. Tang, J. Bi, S. Xu, L. Song, S. Liang, T. Wang, D. Zhang, J. An et al., “Video understanding with large language models: A survey,” arXiv:2312.17432, 2023 (accepted to *IEEE Transactions on Circuits and Systems for Video Technology* according to the arXiv record). arXiv
6. A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Proc. IEEE CVPR*, pp. 1725–1732, 2014. CVF
7. Y. Li, Y. Li and N. Vasconcelos, “RESOUND: Towards action recognition without representation bias,” in *Proc. ECCV*, 2018. CVF
8. L. Sevilla-Lara, S. Zha, Z. Yan, V. Goswami, M. Feiszli and L. Torresani, “Only time can tell: Discovering temporal data for temporal modeling,” in *Proc. IEEE WACV*, 2021. doi
9. R. Goyal, S. Ebrahimi Kahou, V. Michalski, J. Materzyńska, S. Westphal, H. Kim, V. Haenel, I. Fruend et al., “The ‘something something’ video database for learning and evaluating visual common sense,” in *Proc. IEEE ICCV*, 2017. CVF
10. Z. Tong, Y. Song, J. Wang and L. Wang, “VideoMAE: Masked autoencoders are data-efficient learners for self-supervised video pre-training,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2022. NeurIPS ·arXiv
11. S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. doi
12. K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2014. NeurIPS ·arXiv
13. L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang and L. Van Gool, “Temporal segment networks: Towards good practices for deep action recognition,” in *Proc. ECCV*, pp. 20–36, 2016. doi ·arXiv
14. J. Carreira and A. Zisserman, “Quo vadis, action recognition? A new model and the Kinetics dataset,” in *Proc. IEEE CVPR*, 2017. CVF ·arXiv
15. C. Feichtenhofer, H. Fan, J. Malik and K. He, “SlowFast networks for video recognition,” in *Proc. IEEE/CVF ICCV*, 2019. CVF ·arXiv
16. D. Tran, L. Bourdev, R. Fergus, L. Torresani and M. Paluri, “Learning spatiotemporal features with 3D convolutional networks,” in *Proc. IEEE ICCV*, pp. 4489–4497, 2015. CVF
17. W. Kay, J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green et al., “The Kinetics human action video dataset,” arXiv:1705.06950, 2017. arXiv
18. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE CVPR*, 2009. doi

19. T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski and S. Gelly, “Towards accurate generative models of video: A new metric & challenges,” arXiv:1812.01717, 2018. arXiv
20. D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun and M. Paluri, “A closer look at spatiotemporal convolutions for action recognition,” in *Proc. IEEE/CVF CVPR*, 2018. CVF ·arXiv
21. J. Lin, C. Gan and S. Han, “TSM: Temporal shift module for efficient video understanding,” in *Proc. IEEE/CVF ICCV*, 2019. CVF ·arXiv
22. X. Wang, R. Girshick, A. Gupta and K. He, “Non-local neural networks,” in *Proc. IEEE/CVF CVPR*, 2018. CVF
23. C. Feichtenhofer, “X3D: Expanding architectures for efficient video recognition,” in *Proc. IEEE/CVF CVPR*, 2020. CVF
24. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” in *Proc. ICLR*, 2021. arXiv
25. A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić and C. Schmid, “ViViT: A video vision transformer,” in *Proc. IEEE/CVF ICCV*, 2021. CVF ·arXiv
26. G. Bertasius, H. Wang and L. Torresani, “Is space-time attention all you need for video understanding?,” in *Proc. ICML*, PMLR 139, 2021. PMLR ·arXiv
27. H. Fan, B. Xiong, K. Mangalam, Y. Li, Z. Yan, J. Malik and C. Feichtenhofer, “Multiscale vision transformers,” in *Proc. IEEE/CVF ICCV*, 2021. CVF ·arXiv
28. Z. Liu, J. Ning, Y. Cao, Y. Wei, Z. Zhang, S. Lin and H. Hu, “Video Swin Transformer,” in *Proc. IEEE/CVF CVPR*, 2022. CVF ·arXiv
29. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, “Swin Transformer: Hierarchical vision transformer using shifted windows,” in *Proc. IEEE/CVF ICCV*, 2021. doi
30. K. Li, X. Li, Y. Wang, Y. He, Y. Wang, L. Wang and Y. Qiao, “VideoMamba: State space model for efficient video understanding,” in *Proc. ECCV*, pp. 237–255, 2024. doi ·arXiv
31. A. Gu and T. Dao, “Mamba: Linear-time sequence modeling with selective state spaces,” in *Proc. Conference on Language Modeling* (*COLM*), 2024. arXiv
32. K. He, X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, “Masked autoencoders are scalable vision learners,” in *Proc. IEEE/CVF CVPR*, 2022. doi
33. L. Wang, B. Huang, Z. Zhao, Z. Tong, Y. He, Y. Wang, Y. Wang and Y. Qiao, “VideoMAE V2: Scaling video masked autoencoders with dual masking,” in *Proc. IEEE/CVF CVPR*, 2023. CVF ·arXiv
34. A. Bardes, Q. Garrido, J. Ponce, X. Chen, M. Rabbat, Y. LeCun, M. Assran and N. Ballas, “Revisiting feature prediction for learning visual representations from video,” *Transactions on Machine Learning Research* (*TMLR*), 2024. OpenReview ·arXiv
35. M. Assran, A. Bardes, D. Fan, Q. Garrido, R. Howes et al., “V-JEPA 2: Self-supervised video models enable understanding, prediction and planning,” arXiv:2506.09985, 2025. arXiv
36. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal et al., “Learning transferable visual models from natural language supervision,” in *Proc. ICML*, PMLR 139, 2021. PMLR
37. H. Luo, L. Ji, M. Zhong, Y. Chen, W. Lei, N. Duan and T. Li, “CLIP4Clip: An empirical study of CLIP for end to end video clip retrieval and captioning,” *Neurocomputing*, vol. 508, pp. 293–304, 2022. doi ·arXiv
38. J. Xu, T. Mei, T. Yao and Y. Rui, “MSR-VTT: A large video description dataset for bridging video

and language," in *Proc. IEEE CVPR*, 2016. CVF
39. Y. Wang, K. Li, X. Li, J. Yu, Y. He, G. Chen, B. Pei, R. Zheng et al., "InternVideo2: Scaling foundation models for multimodal video understanding," in *Proc. ECCV*, pp. 396–416, 2024. doi ·arXiv
40. C. Fu, Y. Dai, Y. Luo, L. Li, S. Ren, R. Zhang, Z. Wang, C. Zhou et al., "Video-MME: The first-ever comprehensive evaluation benchmark of multi-modal LLMs in video analysis," in *Proc. IEEE/CVF CVPR*, 2025. CVF ·arXiv
41. T. Lin, X. Liu, X. Li, E. Ding and S. Wen, "BMN: Boundary-matching network for temporal action proposal generation," in *Proc. IEEE/CVF ICCV*, 2019. CVF
42. C.-L. Zhang, J. Wu and Y. Li, "ActionFormer: Localizing moments of actions with transformers," in *Proc. ECCV*, pp. 492–510, 2022. doi ·arXiv
43. C. Gu, C. Sun, D. A. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici et al., "AVA: A video dataset of spatio-temporally localized atomic visual actions," in *Proc. IEEE/CVF CVPR*, 2018. CVF ·arXiv
44. A. Shahroudy, J. Liu, T.-T. Ng and G. Wang, "NTU RGB+D: A large scale dataset for 3D human activity analysis," in *Proc. IEEE CVPR*, 2016. doi ·arXiv
45. S. Yan, Y. Xiong and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proc. AAAI*, vol. 32, no. 1, 2018. doi ·arXiv
46. P. Weinzaepfel and G. Rogez, "Mimetics: Towards understanding human actions out of context," *International Journal of Computer Vision*, vol. 129, no. 5, pp. 1675–1690, 2021. doi ·arXiv
47. K. Soomro, A. R. Zamir and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild," technical report CRCV-TR-12-01, arXiv:1212.0402, 2012. arXiv
48. H. Kuehne, H. Jhuang, E. Garrote, T. Poggio and T. Serre, "HMDB: A large video database for human motion recognition," in *Proc. IEEE ICCV*, pp. 2556–2563, 2011. doi
49. G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev and A. Gupta, "Hollywood in homes: Crowdsourcing data collection for activity understanding," in *Proc. ECCV*, pp. 510–526, 2016. doi ·arXiv
50. H. Idrees, A. R. Zamir, Y.-G. Jiang, A. Gorban, I. Laptev, R. Sukthankar and M. Shah, "The THUMOS challenge on action recognition for videos 'in the wild'," *Computer Vision and Image Understanding*, vol. 155, pp. 1–23, 2017. doi
51. F. Caba Heilbron, V. Escorcia, B. Ghanem and J. C. Niebles, "ActivityNet: A large-scale video benchmark for human activity understanding," in *Proc. IEEE CVPR*, pp. 961–970, 2015. CVF
52. D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro et al., "Scaling egocentric vision: The EPIC-KITCHENS dataset," in *Proc. ECCV*, 2018. CVF
53. D. Damen, H. Doughty, G. M. Farinella, A. Furnari, E. Kazakos, J. Ma, D. Moltisanti, J. Munro et al., "Rescaling egocentric vision: Collection, pipeline and challenges for EPIC-KITCHENS-100," *International Journal of Computer Vision*, vol. 130, no. 1, 2022. doi ·arXiv
54. J. Choi, C. Gao, J. C. E. Messou and J.-B. Huang, "Why can't I dance in the mall? Learning to mitigate scene bias in action recognition," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2019. NeurIPS ·arXiv
55. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE CVPR*, 2015. doi

DLCV 09・深度影像復原：低階視覺

同一個退化模型、許多個任務：深度網路如何去除模糊、雜訊、低解析度、霧、雨、雪與暗光，從 SRCNN、DnCNN 一路到 Restormer、all-in-one 模型與擴散先驗。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/

你將學到

- 以觀測模型及未知量描述影像復原。
- 建立配對資料與合成退化，再評估真實世界落差。
- 比較任務專用骨幹，以及受退化資訊或指令條件化的模型。
- 選擇損失，並區分忠實度、感知品質與量測一致性。
- 使用適當資料集與協定評估，並辨認缺失細節的不確定性。

先看全貌

低階視覺處理的是像素本身。在系統能辨認人臉、讀出路牌或偵測車輛之前，影像得先清楚到「看得見」。影像復原模型就是流程中負責這一步的部分：拿到一張受損的照片（太小、雜訊多、模糊、起霧、下雨、太暗），輸出對乾淨場景更好的估計。

> **用最簡單的話說**
>
> 想像有人拿給你一張被洗衣機洗過的照片：糊掉了、有斑點、也褪色了。你這輩子看過成千上萬張正常的照片，所以你猜得出它**原本**長什麼樣子：人臉有兩隻眼睛、建築物的邊是直的、天空是平滑的。復原網路做的是同一件事。它看過幾百萬組「洗之前／洗之後」的照片，學會了乾淨照片長什麼樣、洗衣機又會怎麼破壞它，再同時用這兩種知識把損傷還原。

為什麼重要：每支手機相機在你看到照片之前，都已經做過去雜訊、去馬賽克（demosaicking）與超解析度。衛星、顯微鏡、醫學與監視影像在量測之前都要先復原。影音串流服務會把低位元率的畫面放大。自駕車需要偵測器在雨霧中照常運作。從研究的角度看，許多架構想法也是先在復原任務上被驗證的：殘差學習、通道注意力、高效率 Transformer 與擴散先驗，都有知名的復原版本。

> **關於本篇**
>
> 本篇改寫並擴充自先前的文章〈Low-Level Vision Task-Image Restoration 簡介〉[1]。保留原文的順序與觀點，把 2026 年的更新與更正直接寫進正文，並補上數學、程式碼與綜述文獻。

本篇假設你已熟悉 CNN 與反向傳播（DLCV 01），以及傳統影像復原（IP 05），後者詳細介紹了雜訊模型、Wiener 濾波與退化模型。這裡會直接連結過去，不再重複。

相關篇章。有三篇深入本篇的部分主題：DLCV 09A（單張影像超解析度）、DLCV 09A.3（影片超解析度）與 DLCV 09B（以 GAN 與擴散先驗做生成式復原）。影像融合見 DLCV 10。DLCV 01B 說明影像復原網路為什麼和分類網路的積木組法不同。

什麼是低階視覺？

白話版。高階視覺問的是「圖裡有什麼？」（分類、偵測）；低階視覺問的是「這張圖逐個像素來看，實際上長什麼樣？」前者關於**理解**，後者關於**感知**。

原文 [1] 中的比喻，來自人與外界互動的方式：我們先**感知**（光打在視網膜上，看見邊緣、顏色與對比），再**理解**（那是一個杯子）。兩者缺一不可。如果感知很差（杯子很模糊），理解也會跟著受影響。低階視覺任務就是為了讓感知重新變好而存在：當我們說一個東西「很模糊」，代表感官對它的判定並不靈敏，於是就有了把它變「清晰」的去模糊任務。

精確版。低階視覺任務把一張影像映射成內容（通常）相同的另一張影像：

$$f_\theta : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^{H'\times W'\times C'},$$

輸出是逐像素的估計，而不是一個標籤。H, W, C 是輸入的高、寬與通道數，H', W', C' 是輸出的（超解析度倍率為 s 時 $H' = sH$，其他情況通常相等）。這帶來三個影響整個領域的特性：

1. **輸出密集且高解析度**。每個像素都重要，所以架構不能大幅降採樣，必須保留細節，這讓高解析度下的運算成本很高。
2. **監督訊號很容易合成**。拿一張乾淨影像，用已知的方式把它弄壞，就得到一組輸入–目標配對。這是相對於辨識任務（需要人工標註）的巨大優勢，也是這個領域最大弱點的來源：合成的損傷終究不是真實的損傷。
3. 「**正確答案**」**並不唯一**。很多張乾淨影像都和同一張受損影像相容。哪一張「最好」，取決於你重視的是忠於真實還是看起來自然，後面的評估章節會詳細討論。

這類任務範圍很廣：去模糊、超解析度、去雜訊、修補（inpainting）、色彩校正、紅外線與可見光融合等等。過去大家常在 Papers with Code 上依任務瀏覽分類與排行榜，但它已停止營運，原網址現在會直接轉址到 Hugging Face 的 Trending Papers 頁面，依任務分類的 SOTA 頁面已無法瀏覽。現在追蹤進展的可靠方式，是每年的 **NTIRE** 挑戰賽報告（New Trends in Image Restoration and Enhancement，CVPR 的一個 workshop），本篇接近結尾時會介紹 [2], [3]。

一個問題、許多面貌：退化模型

白話版。本篇的每一個任務都可以寫成：「相機看到乾淨的場景 x，某種東西破壞了它，我們拿到的是 y。」改變的只有那個「破壞」。只要能把破壞寫下來，就能說出這是哪個任務。

統一的觀點

$$y = \mathcal{D}(x) + n,$$

其中 x 是未知的乾淨影像，$\mathcal{D}$ 是退化運算子（可能是非線性的，也可能未知），n 是雜訊，y 是觀測到的影像。復原就是從 y 估計 x。以下是原文 [1] 與本篇後面用到的具體形式，圖 1 把每一種套用在同一張照片上。

模糊（課本上的模型）。和 IP 05 一樣，

$$g(u,v) = (h * f)(u,v) + n(u,v),$$

其中 f 是清晰影像，h 是點擴散函數（模糊核），$*$ 是二維卷積，n 是加性雜訊，g 是觀測影像，(u, v) 是像素座標。去雜訊是 $h = \delta$（單位脈衝，也就是沒有模糊）的特例。

超解析度。低解析度影像是高解析度影像經過模糊、次取樣再加上雜訊的結果：

$$y = (x * k) \downarrow_s + n,$$

其中 k 是模糊（抗鋸齒）核，$\downarrow_s$ 表示每個方向每 s 個像素取一個，s 是放大倍率（2、3、4.....）。經典的「bicubic」基準設定則把這一切換成 MATLAB 風格的雙三次降採樣，並且不加雜訊。

壓縮。$y = \mathrm{JPEG}_q(x)$，q 是品質因子。JPEG 會量化 8×8 區塊的 DCT 係數（見 IP 08），產生區塊效應與振鈴。它既非加性也非線性，所以通常直接當成黑盒子放進退化流程。

霧（大氣散射模型）。

$$I(p) = J(p)\, t(p) + A\,(1 - t(p)), \qquad t(p) = e^{-\beta d(p)},$$

其中 p 是像素位置，I 是有霧影像，J 是無霧的場景輻射，A 是全域大氣光（霧的顏色），$t \in (0, 1]$ 是透射率，β 是散射係數，d 是場景深度。第一項是被衰減的場景；第二項是被散射進相機的「空氣光」（airlight）。越遠的物體（d 大）t 越小，逐漸融入 A。

雨。雨線（rain streak）常用加性模型：

$$I = B + R,$$

其中 B 是無雨背景，R 是雨層。附著在鏡頭或擋風玻璃上的雨滴（raindrop）行為不同：它會遮住並折射背景，所以常用的模型是 $I = (1 - M) \odot B + R$，其中 M 是雨滴的二值遮罩，$\odot$ 是逐元素乘積，R 是穿過雨滴的光 [39]。

低光（Retinex）。Retinex 理論 [5] 把影像分解成反射率與光照：

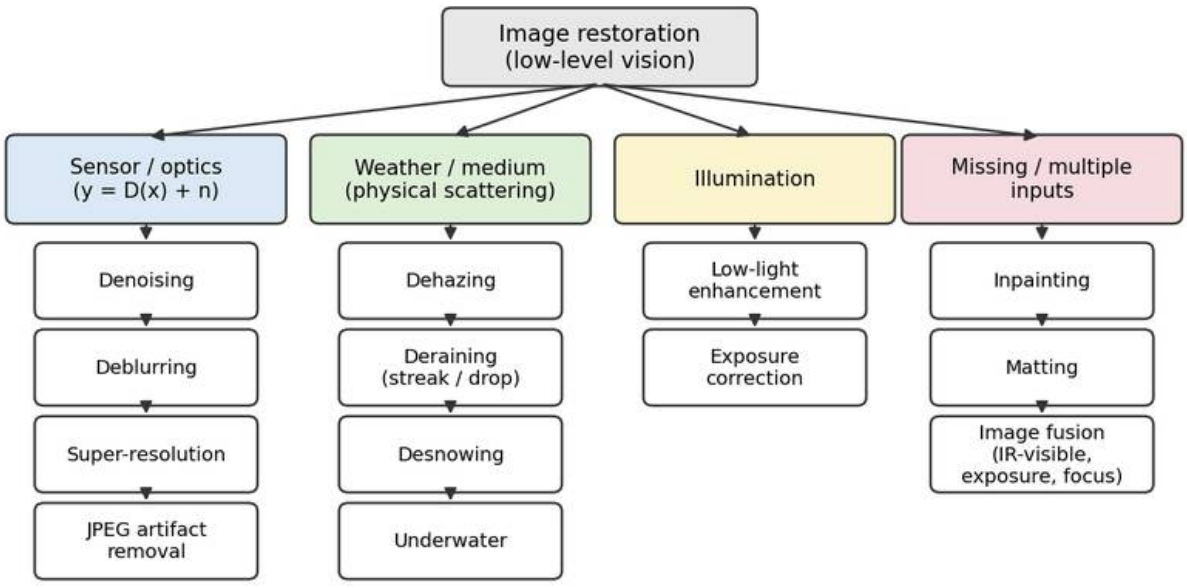
$$S = R \cdot L,$$

其中 S 是觀測影像，R 是反射率（物體表面本身的顏色與紋理，這是我們要的），L 是光照，$\cdot$ 是逐元素乘積。暗照片的 L 很小；增強就是估計 L 並把它提亮，同時原本藏在暗處的雜訊也會被放大。



圖 *1* — 同一張乾淨影像 x 經過本節各種退化模型。每個復原任務都想還原其中一種（或幾種）退化。

這些模型都不完全精確，卻有相同的形狀：乾淨影像進去，運算子加上雜訊出來。這正是原文 [1] 的核心觀點：雖然不是很嚴謹，但透過退化模型來看，它們本質上都可以用同一種方式「理解」，**屬於同一種問題**，同一套工具（而且常常是同一個網路）都適用。圖 2 依照損傷的來源把它們分類。



All-in-one models try to cover several boxes with one network

圖 2 — 依退化來源整理的低階視覺任務分類。*All-in-one* 模型試圖用單一網路涵蓋多個方框。

為什麼困難：病態問題

上面幾乎每一種 $\mathcal{D}$，都有許多不同的 x 會產生相同的 y。降採樣 4 倍丟掉每 16 個像素中的 15 個；模糊把高頻壓到幾乎為零；$t \approx 0$ 的濃霧會抹去遠處的物體。這個反問題是**病態的**（ill-posed），我們需要一個關於乾淨影像長相的**先驗**（prior）。用貝氏（MAP）形式寫：

$$\hat{x} = \arg\min_{x} \; \tfrac{1}{2}\|y - \mathcal{D}(x)\|_2^2 + \lambda\, \Phi(x),$$

第一項（資料保真項）要求 $\hat{x}$ 必須能解釋觀測，Φ 是編碼先驗的正則項（自然的影像 Φ 小），$\lambda > 0$ 平衡兩者。整個領域的歷史，就是一部尋找更好先驗 Φ 的歷史。

深度學習之前：手工先驗的時代

在深度學習興起之前，每個任務都有自己精心推導的先驗，成敗取決於對問題是否有足夠充分的物理或統計知識（prior knowledge）。去雜訊用局部平滑、非局部自相似與稀疏性；去模糊用 Wiener 與限制最小平方濾波，加上重尾的梯度先驗；除霧用暗通道先驗 [25]；超解析度用插值與以範例為基礎的字典；低光增強用直方圖等化與 Retinex 分解。IP 05 涵蓋了這套工具的核心（雜訊模型、平均／中位數／自適應濾波、反濾波、Wiener 與 CLS 濾波），結尾也簡短回顧了走向學習式去雜訊的歷史。[1] 的看法是：深度學習逐漸解決並超越這些模型能做到的程度之後，關鍵不再是為每個任務建立強大的理論先驗，而是強大的**學習機制與特徵擷取方法**。退化模型並沒有消失，它從求解器搬進了**資料產生器**。

深度學習的問題設定

白話版。做出大量（受損, 乾淨）的影像對，訓練網路把受損的變成乾淨的。大部分的苦工都在讓損傷看起來夠真實。

以成對資料做監督式訓練

給定乾淨影像集合 $\{x_i\}$ 與一個退化取樣器，建立配對 (y_i, x_i)，其中 $y_i = \mathcal{D}_i(x_i) + n_i$，然後求

$$\theta^\star = \arg\min_{\theta} \; \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(y_i),\, x_i),$$

其中 f_θ 是參數為 θ 的復原網路，N 是訓練對數量，ℓ 是損失函數（L1、L2、感知、對抗……見損失函數一節）。訓練時使用隨機裁切的小塊（patch，例如 64×64 到 256×256）並搭配翻轉與旋轉，因為卷積與視窗注意力網路具有平移等變性，而全解析度影像也放不進 GPU 記憶體。

有一個事實可以解釋後面很多現象：用 L2 損失時，最佳網路輸出的是**後驗平均** $f^\star(y) = \mathbb{E}[x \mid y]$。當許多清晰影像都與 y 相容時，它們的平均是模糊的。這就是為什麼以 MSE 訓練的超解析度看起來很平滑，也是後來引入 GAN 與擴散模型的原因。

合成退化流程

配對資料有三種來源。(1) **合成**：用已知模型（雙三次降採樣、高斯雜訊、高斯模糊、JPEG）弄壞乾淨影像。便宜且無限量，但受限於模型有多真實。(2) **實拍**：同一個場景拍兩次，例如短曝光與長曝光，或把連拍平均成乾淨參考（如真實雜訊資料集 SIDD [22]），或把高速攝影的連續影格平均，得到真實的動態模糊（GoPro [44]）。真實但昂貴，且侷限於特定相機。(3) **非成對／自監督**：只用受損影像學習，靠 GAN、循環一致性或雜訊統計。今天大多數強模型都用豐富退化流程產生的合成配對訓練，再同時在合成與實拍資料上評估。

典型的「經典」流程是模糊 → 降採樣 → 雜訊 → JPEG，模仿相機與圖片分享網站依序破壞照片的順序：

```
import numpy as np, cv2, torch, torch.nn.functional as F
from skimage import data, img_as_float32

def gaussian_kernel(sigma, size=21):
    ax = torch.arange(size) - size // 2
    g = torch.exp(-ax**2 / (2 * sigma**2))
    k = torch.outer(g, g)
    return k / k.sum()

def degrade(x, scale=4, sigma_blur=1.6, sigma_noise=0.03, jpeg_q=40, rng=None):
    """x: float tensor (C,H,W) in [0,1]. Returns the low-quality image y = D(x) + n."""
    rng = rng or np.random.default_rng(0)
    C = x.shape[0]
    k = gaussian_kernel(sigma_blur).to(x)[None, None].repeat(C, 1, 1, 1)
    y = F.conv2d(F.pad(x[None], (10,) * 4, mode="reflect"), k, groups=C)   # 1) blur  h * x
    y = y[..., ::scale, ::scale]                                           # 2) downsample
     ↪  (.)↓s
    y = y + sigma_noise * torch.from_numpy(rng.standard_normal(y.shape).astype(np.float32))
     ↪  # 3) noise n
    y = y.clamp(0, 1)[0]
    img = (y.permute(1, 2, 0).numpy() * 255).round().astype(np.uint8)      # 4) JPEG round
     ↪  trip
    ok, buf = cv2.imencode(".jpg", img[..., ::-1], [cv2.IMWRITE_JPEG_QUALITY, jpeg_q])
    img = cv2.imdecode(buf, cv2.IMREAD_COLOR)[..., ::-1]
    return torch.from_numpy(img.copy()).permute(2, 0, 1).float() / 255

x = torch.from_numpy(img_as_float32(data.astronaut())).permute(2, 0, 1)    # (3, 512, 512)
y = degrade(x)
print(x.shape, "->", y.shape)        # torch.Size([3, 512, 512]) -> torch.Size([3, 128, 128])
```

實際訓練時，每個參數（模糊核的寬度與形狀、雜訊強度與種類、JPEG 品質，甚至順序）都會**對每張影像隨機取樣**，讓網路看到的是一整個退化分布，而不是單一一種。

真實世界與盲退化

只用一種固定 $\mathcal{D}$（例如雙三次 ×4）訓練的模型，遇到真實照片就會失效，因為真實退化是未知的、隨位置變化的，而且是多種效應的混合。這就是**盲**（blind，或稱真實世界）復原問題。大致有兩種策略：

- **把合成分布做得夠寬，寬到能涵蓋真實情況**。Real-ESRGAN [4] 把經典流程**連續套用兩次**（「高階」退化，每一輪都隨機取樣模糊核、縮放、雜訊與 JPEG），並加入 sinc 濾波器來模擬振鈴與過衝。即使只用合成資料訓練，它在真實低品質影像上的泛化也比雙三次訓練的模型好得多。
- **引入強大的生成先驗**。預訓練的文字生成影像擴散模型已經知道自然影像長什麼樣。DiffBIR [16]、SUPIR [17] 與 OSEDiff [18] 都以受損輸入作為條件來驅動這類模型，讓先驗「補上」合理的細節。這是真實世界超解析度近年的主要趨勢，後面會再討論。

第三種互補的策略是**估計退化本身**（雜訊強度、模糊核、退化種類），再餵給網路，例如 FFDNet 的雜訊強度圖 [21] 與各種 all-in-one 模型。在盲超解析度中，未知的主要是 $y = (x * k)\downarrow_s + n$ 裡的模糊核 k；有一整個系列的方法把模糊核交給網路當條件、在迴圈中估計並修正它，或只從測試影像本身估計它 [87]–[91]。要記住的結論是：明確估計模糊核的方法在退化真的是「模糊、降採樣、雜訊」時準確又可解釋，但錯誤的模糊核會傳到輸出裡。所以大多數真實世界模型，例如 BSRGAN [92]、Real-ESRGAN [4] 與擴散先驗方法，改從隨機化的退化流程中**隱式**地學習退化。

僅線上版：盲超解析度：以模糊核為條件與估計模糊核 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/盲超解析度以模糊核為條件與估計模糊核/*

僅線上版：即插即用先驗與半二次分裂 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/即插即用先驗與半二次分裂/*

超解析度（Super-Resolution, SR）

白話版。把小圖變大**而且**變清楚，也就是預測缺少的高解析度像素，而不只是把現有的像素拉伸開來。

本節只做概覽；DLCV 09A・影像超解析度會深入介紹超解析度：退化模型、傳統與學習式方法、感知式與擴散式超解析度、真實世界超解析度，以及如何評估。

問題設定

單張影像超解析度（SISR）在 $y = (x * k)\downarrow_s + n$ 之下，從 $y \in \mathbb{R}^{H\times W\times C}$ 估計 $x \in \mathbb{R}^{sH\times sW\times C}$。傳統做法是**插值法**（interpolation）：最近鄰（nearest neighbor）、雙線性（bilinear）與雙三次（bicubic）。其中雙三次插值以 4×4 鄰域擬合三次多項式，最為知名，至今仍被廣泛使用，既是基準方法，也是產生低解析度測試輸入的標準方式。插值只會重新分配既有資訊，無法重建被 $\downarrow_s$ 丟掉的高頻。

網路怎麼「變大」：三種上採樣策略

輸出比輸入大，所以每個 SR 網路都得決定解析度**在哪裡**提升。

a. 先上採樣（先插值）。SRCNN [6] 先用雙三次插值把輸入放大到目標尺寸，再接三層卷積（區塊擷取、非線性映射、重建）。簡單直觀，但每一層都在高解析度上運算，速度慢，網路還得清理插值造成的瑕疵。

b. 轉置卷積（反卷積，deconvolution）。FSRCNN [7] 在低解析度上運算，只在最後用學習得到的轉置卷積放大，速度比 SRCNN 快很多。步長無法整除核大小的轉置卷積會產生不均勻的重

疊，容易出現**棋盤狀瑕疵**（checkerboard artifact）[9]，所以現在很少被當作 SR 的最終上採樣器；不過在其他類型的神經網路中（例如分割與生成模型的解碼器）仍然相當常見。

c. 子像素卷積（pixel shuffle）。 ESPCN [8] 在低解析度上計算 Cs^2 個通道，再重新排列成 s 倍大的影像：

$$\mathrm{PS}(T)_{c,\,sh+i,\,sw+j} = T_{cs^2+si+j,\,h,\,w}, \qquad 0 \le i,j < s,$$

其中 T 是有 Cs^2 個通道的低解析度特徵張量，(h,w) 是低解析度位置，(i,j) 是子像素偏移，c 是輸出通道。所有學習都在低解析度進行，重新排列本身沒有參數。這是現代 SR 網路的主流選擇（`torch.nn.PixelShuffle`）。

里程碑與目前的主流家族

白話版。 SR 網路先是變深，接著學會畫出逼真的紋理，最近則借用了 Transformer 與大型預訓練生成模型。

主要的幾步：**SRCNN** [6] 是第一個端到端的 SR CNN；**EDSR** [10] 與 **RCAN** [11] 把 SR 網路做得非常深（EDSR 拿掉批次正規化，RCAN 加入通道注意力）；**SRGAN** [12] 與 **ESRGAN** [13] 加入對抗損失與感知損失，產生銳利、逼真的紋理，而不是 MSE 的模糊平均。目前的主流家族是 **SwinIR** [14]、**HAT** [15] 這類 Transformer，MambaIR [49] 這類狀態空間模型，以及**擴散先驗**（DiffBIR [16]、SUPIR [17] 與一步式的 OSEDiff [18]）；擴散先驗在真實照片上能產生更有說服力的紋理，但也可能捏造出場景中不存在的細節。模型通常以 **DIV2K** [19] 訓練，並以亮度通道上的 PSNR 與 SSIM 比較；沒有真值的真實世界 SR 則依賴感知與無參考指標（見評估指標一節）。大約在 2018 年，輸出已經遠勝插值，如何**評判** SR 便成為核心問題（見感知–失真一節）。

僅線上版：超解析度的里程碑、主流家族與基準 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/超解析度的里程碑主流家族與基準/*

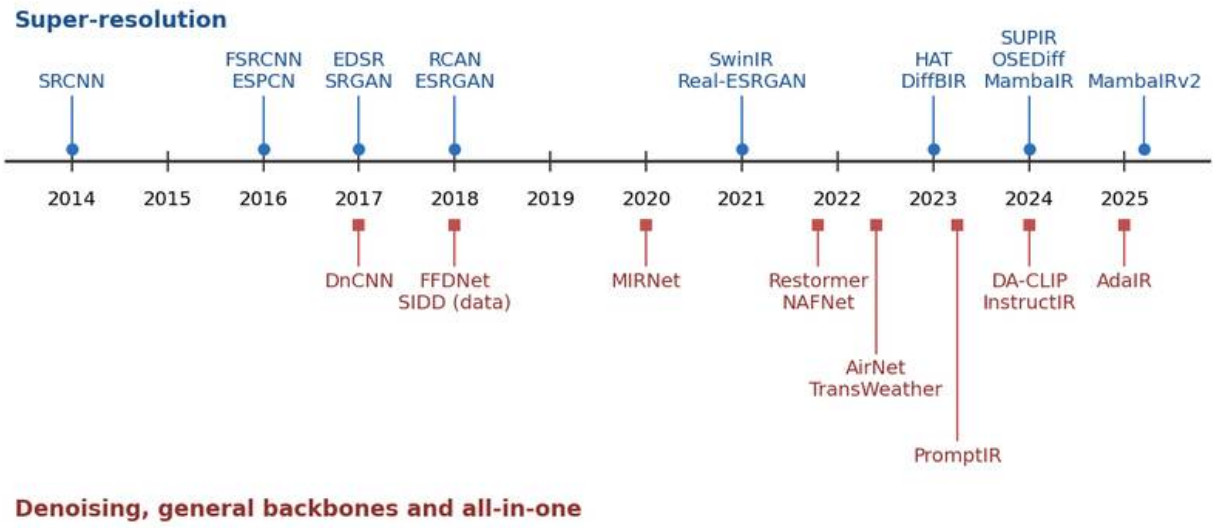


圖 3 — 超解析度（上）以及去雜訊、通用骨幹與 *all-in-one* 復原（下）的里程碑，依發表年份排列（同一年內的位置只是為了方便閱讀）。

去雜訊（Denoising）

白話版。把照片上隨機的顆粒去掉，同時不要把真正的紋理也抹平。雜訊是最單純的退化，所以去雜訊成了大多數深度復原想法最先出現的試驗場。

問題設定

$y = x + n$。合成設定中 $n \sim \mathcal{N}(0, \sigma^2 I)$（加性白高斯雜訊，AWGN，標準差 σ 通常以 0–255 的尺度表示，例如 $\sigma = 25$）。真實相機的雜訊不一樣：它和訊號相關（光子散粒雜訊隨亮度增加），去馬賽克後在空間上具相關性，還會被相機的影像處理流程改變形狀。針對暗影像寫出的卜瓦松–高斯（Poisson–Gaussian）雜訊模型，請見 DLCV 09C。

里程碑

DnCNN [20] 做了兩個後來成為標準的選擇。第一是**殘差學習**（residual learning）：網路 $\mathcal{R}_\theta$ 不直接預測 x，而是預測雜訊，輸出為

$$\hat{x} = y - \mathcal{R}_\theta(y),$$

其中 $\mathcal{R}_\theta(y) \approx n$。殘差（雜訊）比乾淨影像「更容易學」，使得搭配批次正規化的深層純 CNN 能訓練得很好。單一 DnCNN 也能針對一段範圍的雜訊強度訓練（盲高斯去雜訊），而且同一個網路也能處理 SR 與 JPEG 去區塊。

FFDNet [21] 把**雜訊強度圖** M（與影像同尺寸，每個像素的值為 σ）和降採樣後的輸入一起餵給網路。測試時調整 M，可以在去雜訊與保留細節之間取捨；非均勻的 M 則能處理隨位置變化的雜訊。它的速度快到可以在 CPU 上執行。

真實雜訊。 **SIDD** 資料集 [22] 提供真實的智慧型手機雜訊影像，乾淨參考影像則透過仔細拍攝連拍並加以處理而得。只用 AWGN 訓練的模型在它上面表現很差，這推動了整個領域走向真實的雜訊合成與實拍訓練資料。

通用骨幹。 MIRNet [30]、**Restormer** [23] 與 **NAFNet** [24] 是通用的復原架構，在 SIDD 以及去模糊等任務上都報告了最先進的結果，會在骨幹網路一節介紹。

程式碼：一個小小的殘差去雜訊網路

下面的程式在四張 `skimage` 影像隨機切出的 32×32 小塊上訓練一個五層、DnCNN 風格的網路，每一步都加上新的高斯雜訊，然後在一張從沒看過的影像上測試。在筆電 CPU 上大約一分鐘內就能跑完（我們在負載很重的機器上量到約一分鐘）。

```
import time, numpy as np, torch, torch.nn as nn
from skimage import data, img_as_float32
from skimage.color import rgb2gray

torch.manual_seed(0); torch.set_num_threads(2)
rng = np.random.default_rng(0)
```

```
sigma = 25 / 255                                    # noise level (on a [0,1] scale)
train_imgs = [img_as_float32(im) for im in (data.camera(), data.coins(), data.moon(),
 ↪  data.page())]
test_img = img_as_float32(rgb2gray(data.astronaut()))   # held out

def random_patches(n=32, p=32):
    batch = []
    for _ in range(n):
        im = train_imgs[rng.integers(len(train_imgs))]
        i, j = rng.integers(0, im.shape[0] - p), rng.integers(0, im.shape[1] - p)
        batch.append(im[i:i + p, j:j + p])
    return torch.from_numpy(np.stack(batch))[:, None]   # (n, 1, p, p)

class TinyDnCNN(nn.Module):
    """conv-ReLU, (depth-2) x conv-BN-ReLU, conv. Predicts the NOISE, not the image."""
    def __init__(self, depth=5, ch=32):
        super().__init__()
        layers = [nn.Conv2d(1, ch, 3, padding=1), nn.ReLU(inplace=True)]
        for _ in range(depth - 2):
            layers += [nn.Conv2d(ch, ch, 3, padding=1), nn.BatchNorm2d(ch),
             ↪  nn.ReLU(inplace=True)]
        layers += [nn.Conv2d(ch, 1, 3, padding=1)]
        self.body = nn.Sequential(*layers)
    def forward(self, y):
        return y - self.body(y)                         # x_hat = y - R(y): residual learning

net = TinyDnCNN()
opt = torch.optim.Adam(net.parameters(), lr=2e-3)
t0 = time.time()
for step in range(300):
    x = random_patches()
    y = x + sigma * torch.randn_like(x)                 # synthetic pairs, made on the fly
    loss = nn.functional.l1_loss(net(y), x)
    opt.zero_grad(); loss.backward(); opt.step()
print(f"trained in {time.time() - t0:.0f} s, last L1 = {loss.item():.4f}")

net.eval()
with torch.no_grad():
    x = torch.from_numpy(test_img)[None, None]
    y = x + sigma * torch.randn_like(x)
    x_hat = net(y).clamp(0, 1)
mse = lambda a, b: torch.mean((a - b) ** 2).item()
print(f"PSNR noisy {10*np.log10(1/mse(y.clamp(0,1), x)):.2f} dB -> denoised
 ↪  {10*np.log10(1/mse(x_hat, x)):.2f} dB")
# e.g. PSNR noisy 20.74 dB -> denoised 28.64 dB
```

即使是這個只用四張圖、訓練 300 個小步驟的玩具模型，在沒看過的照片上也能提升約 8 dB，而且勝過手動調整的高斯濾波（圖 4）。拖動滑桿，比較雜訊輸入與小網路的輸出。



雜訊（$\sigma = 25$）



小型殘差 *CNN*

滑桿比較：顆粒明顯的灰階太空人臉部，與小型 CNN 去雜訊後乾淨但略顯平滑的同一區塊。

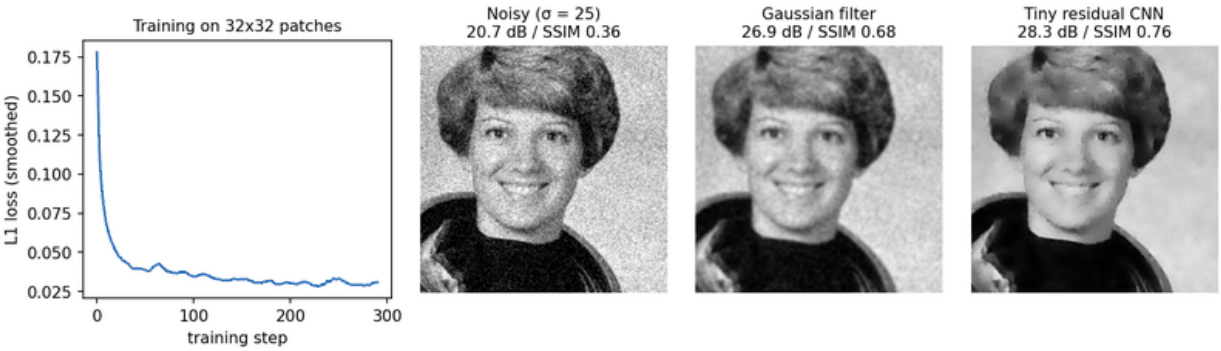


圖 *4* — 小型殘差去雜訊網路。*PSNR* 與 *SSIM* 是在完整的 *512* × *512* 測試影像上計算。*CNN* 去掉的雜訊比高斯濾波多，邊緣也更銳利，但同時把細緻紋理壓平了，這是 *MSE* ／ *L1* 訓練的典型行為。不同次執行、不同機器的數值會略有差異。

除霧（Dehazing）

白話版。去霧旨在恢復受大氣散射遮蔽的對比與顏色，遠處區域尤其明顯。

傳統做法與暗通道先驗

傳統方法先估計 $I = Jt + A(1-t)$ 中的未知量（透射率圖 t 與大氣光 A），再代回公式求解。最困難的部分，就是想出能把這些未知量確定下來的可靠關係。最知名的是何凱明等人的**暗通道先驗**（dark channel prior）[25]：在無霧的戶外影像中，大多數局部區塊裡至少有一個像素在某個顏色通道上非常暗（陰影、深色或鮮豔的物體）：

$$J^{\text{dark}}(p) = \min_{c\in\{r,g,b\}} \Big(\min_{q\in\Omega(p)} J^c(q) \Big) \approx 0,$$

其中 $\Omega(p)$ 是以 p 為中心的小區塊，J^c 是顏色通道 c。對霧的模型做同樣的雙重取最小值，再利用 $J^{\text{dark}} \approx 0$，就得到透射率估計

$$\tilde{t}(p) = 1 - \omega \min_c \Big(\min_{q\in\Omega(p)} \frac{I^c(q)}{A^c} \Big),$$

其中 $\omega < 1$（論文中取 0.95）刻意保留一點霧，以維持深度感。接著 $\hat{J} = (I - A)/\max(\tilde{t}, t_0) + A$，小小的 t_0 用來避免除以零。這個先驗在大片與大氣光相近的明亮區域（白牆、天空）會失效，而以區塊為單位得到的粗糙 t 也必須再精修，才能貼合邊緣。

深度除霧：兩個流派

原文 [1] 把深度除霧分成兩派，這個分法至今仍很貼切。

1. 物理引導（「套公式」）。學出未知參數，再反轉模型。**DehazeNet** [26] 用 CNN（搭配 Maxout 單元與有界 ReLU）從有霧區塊迴歸出透射率 t。**AOD-Net** [27] 進一步簡化大氣散射模型，把 t 與 A 合併成單一函數 K：

$$J(p) = K(p)\,I(p) - K(p) + b, \qquad K(p) = \frac{\frac{1}{t(p)}(I(p) - A) + (A - b)}{I(p) - 1},$$

其中 b 是常數偏移。一個輕量 CNN 預測 K，於是整個網路是端到端的，還可以接在偵測器前面一起訓練。

2. 強大的通用架構（「超級模型」）。跳過物理，直接用強大的網路學 $I \mapsto J$。**GridDehazeNet** [28] 使用以注意力為基礎的多尺度網格；**FFA-Net** [29] 堆疊通道注意力與像素注意力區塊，並做自適應特徵融合。這類設計的學習能力很強，換句話說，很有機會也適用於其他退化任務，這正是 MIRNet [30] 等通用復原網路走的路。之後出現了 Transformer 除霧模型：**DehazeFormer** [31] 修改正規化層、激活函數與空間聚合方式，把 Swin Transformer 調整成適合除霧。

資料集

- **RESIDE** [32]：大型基準，包含以深度與散射模型合成的室內外有霧影像，以及用於任務導向評估的真實有霧影像。它的 SOTS 測試集是最常被引用的除霧基準。
- **O-HAZE** [33] 與 **Dense-Haze** [34]：以專業造霧機拍攝的真實有霧／無霧配對，分別是戶外場景與濃霧場景。規模小，但能突顯合成與真實之間的落差。
- **KITTI** [35] **不是**除霧資料集。它是在 Karlsruhe 一帶錄製的自駕車情境視覺基準（立體視覺、光流、里程計、3D 物件偵測、追蹤）。部分除霧研究會在它的影像上合成霧氣（它有光達與立體視覺提供的深度），或用它評估除霧對下游偵測的影響。

除雪（Desnowing）

白話版。去掉飄落的雪。看起來像除雨的親戚，但行為上也很像除霧。

除雪比看起來困難。雪花的大小與透明度差異極大，從不透明的白色團塊到半透明的條紋都有，而大雪還會造成類似霧的**朦朧效應**（veiling effect）。所以除雪模型不只要處理呈現為白點的「雪」，也得處理朦朧的霧氣。**DesnowNet** [36] 推出 **Snow100K** 資料集（在真實影像上合成雪，另附真實雪景影像），以及一個同時估計雪的半透明程度與其殘留的網路。**JSTASR** [37] 明確建模雪的大小與透明度，並加入去除朦朧效應的步驟。現在除雪通常被放進 TransWeather [52] 這類「惡劣天氣」all-in-one 模型中一起研究。

除雨（Deraining）

白話版。把照片裡的雨去掉。「雨」其實指兩種很不一樣的東西：在空中落下的細雨線，以及停在鏡頭或擋風玻璃上的雨滴。

雨線與雨滴

- **雨線**（rain streak）是疊加在場景上、細而亮、有方向性的線條，模型為 $I = B + R$。遠處的雨線累積起來會形成類似霧的面紗。方法的目標是把兩層分開。一個早期具影響力的深度方法同時偵測並去除雨 [38]，並提供了被廣泛使用的輕雨與大雨合成資料集。
- **雨滴**（raindrop）附著在玻璃表面，像一個個小透鏡：遮住並折射背景、常常失焦，而且沒有一致的方向。Qian 等人的注意力 GAN [39] 用注意力圖（雨滴在哪裡？）同時引導生成器與判別器，並推出一個透過玻璃拍攝、有無水滴成對的雨滴資料集。

走向真實場景

大多數除雨資料集是合成的，用合成雨線訓練的模型面對真實的雨常常失效。**NTIRE 2025 Day and Night Raindrop Removal for Dual-Focused Images** 挑戰賽 [40] 反映了研究重心轉向真實資料：它使用真實拍攝的 **Raindrop Clarity** 資料集，涵蓋白天與夜晚，以及對焦在雨滴與對焦在背景兩種情況（訓練 14,139 張、驗證 240 張、測試 731 張）。對焦設定很關鍵：相機對焦在雨滴上時，背景也是模糊的，於是去除雨滴有一部分變成了去模糊。

低光增強（簡介）

白話版。讓暗照片看起來曝光正常、細節可見、色調自然，同時不要把雜訊放大。

訓練時對比暗影像與同一場景正常曝光的參考影像，例如隨 **RetinexNet** [41] 推出、包含低光／正常光配對的 **LOL** 資料集。傳統方法以對比度調整（直方圖等化）與 Retinex 模型為主（較少見的還有反轉大氣散射模型）。深度方法有兩條路線：受 Retinex 啟發、分解 $S = R \cdot L$ 再調整 L 的網路（RetinexNet [41]），以及特別設計的通用網路。**Retinexformer** [42]（ICCV 2023）結合兩者：單階段的 Retinex 公式加上光照引導的注意力，作者稱它是第一個以 Transformer 為基礎的低光增強方法。低光增強本身的內容（零參考方法、曲線估計、雜訊、色彩、影片）多到值得獨立一章：見 DLCV 09C。

影像融合（簡介）

白話版。把同一場景、各有所長的幾張影像組合成一張，讓結果同時擁有每一張的優點，資訊更豐富多樣。

原文 [1] 提到三種常見設定：**紅外線與可見光**融合（紅外線提供熱目標，可見光相機提供紋理與色彩）、**多重曝光**融合（結合曝光不足與過度照片的細節，得到類似 HDR 的結果），以及**多焦點**融合（由對焦在不同深度的照片合成全景深影像）。與其他任務不同，融合通常沒有真實的「正確融合影像」，所以深度融合依賴保留來源影像梯度與強度的非監督式損失，或使用 GAN。Zhang 等人的綜述 [43] 依這些情境與學習策略整理了深度融合方法。

去模糊（Deblurring）

白話版。從被手震、物體移動或失焦弄糊的影像中，還原出清晰的影像。

問題設定

去模糊就是 $g = h * f + n$ 的直接應用。模糊核 h 已知時稱為**非盲**（non-blind）去模糊（DIP 第 5 章中 Wiener 與 CLS 濾波的設定，以及學習式非盲求解器）；h 未知時稱為**盲**（blind）去模糊。動態場景中的真實動態模糊更麻煩：不同物體的運動不同、深度也不同，所以根本沒有單一的全域 h。

盲去模糊：估計模糊核

白話版。盲去模糊須推估清晰影像與模糊；交替方法固定其中一項，再改善另一項估計。

聯合問題。對空間上均勻的模糊，盲反卷積（blind deconvolution）要解

$$(\hat{f}, \hat{h}) = \arg\min_{f,h} \|h * f - g\|_2^2 + \lambda_f\, \phi(f) + \lambda_h\, \psi(h) \quad \text{subject to} \quad h \geq 0,\ \ \textstyle\sum_{u,v} h(u,v) = 1,$$

其中 g 是模糊影像，f 是未知的清晰影像，h 是未知的模糊核（只在一個小視窗內非零，手震通常是 15×15 到 75×75 像素），ϕ 是影像先驗（通常是稀疏梯度懲罰，因為自然影像大多是平坦區域加上少數強邊緣），ψ 是模糊核先驗，$\lambda_f, \lambda_h > 0$ 是它們的權重。限制條件表示模糊核只是把光重新分配：它非負，而且保持總亮度不變。即使如此，模糊核也只能還原到差一個小平移的程度，因為把 f 往一個方向平移、h 往反方向平移，$h * f$ 不變。

為什麼這麼病態。未知數（f 和 h）遠多於方程式，很多組解都能完全解釋 g。更糟的是，最明顯的那一組是錯的：$h = \delta$（沒有模糊）且 $f = g$ 完美符合資料。Levin 等人 [74] 指出，在常用的稀疏梯度先驗下，模糊影像的先驗代價往往比清晰影像更低，因為模糊會讓梯度變小，而 ϕ 偏好小梯度。所以對 (f, h) 做天真的聯合 MAP 估計，常會得到「沒有模糊」的解。他們的分析指向一個較適定的目標：只估計模糊核，對所有可能產生 g 的清晰影像取平均。模糊核的未知數比像素觀測少得多，所以它比整張影像確定得多；一旦知道 $\hat{h}$，再用非盲反卷積就能還原 f。

傳統的解法。實務上有效的方法，不是近似上述對影像的平均（Fergus 等人 [75] 的變分貝氏），就是**預測並挑選清晰邊緣**，只用可靠的梯度估計模糊核（Cho 與 Lee [77] 用衝擊濾波器把目前的估計銳化，下面的程式也這麼做；Xu 與 Jia [78] 只保留有資訊量的邊緣），或是改變先驗，讓清晰的影像勝出（Krishnan 等人 [79]、Pan 等人 [80]）。它們都共用下面這個迴圈。

標準流程。幾乎所有傳統盲去模糊方法都遵循同一個迴圈。在影像金字塔最粗的一層，從 delta 模糊核開始，交替進行

$$f^{(t+1)} = \arg\min_{f} \|h^{(t)} * f - g\|_2^2 + \lambda_f\, \phi(f),$$

$$h^{(t+1)} = \arg\min_{h \geq 0,\ \sum h = 1} \|h * \nabla \tilde{f}^{(t+1)} - \nabla g\|_2^2 + \lambda_h \|h\|_2^2,$$

其中 t 是迭代次數，∇ 是影像梯度，$\tilde{f}$ 是經過邊緣預測或挑選後的潛在影像（銳化過，弱梯度設為零）。模糊核這一步在梯度域進行，因為卷積與微分可以交換，$\nabla(h * f) = h * \nabla f$，而模糊的資訊主要藏在梯度裡；搭配二次懲罰時，它在頻域有封閉解，之後再投影回限制條件。迭代幾次後，

把模糊核放大到金字塔的下一層（**由粗到細**，coarse-to-fine），因為大模糊在粗尺度上只有幾個像素寬，這樣可以避開不好的局部極小值。最後固定 $\hat{h}$，用**非盲反卷積**產生輸出影像：可以是 IP 05 的 Wiener 或限制最小平方濾波，也可以是超拉普拉斯（hyper-Laplacian）先驗或學習式的非盲求解器。

深度學習與模糊核。神經網路以三種方式進入盲去模糊：**先預測模糊核**（或反卷積濾波器），再以傳統方法反卷積 [81]–[83]；用未經訓練的生成網路、不靠訓練資料，**對單張照片同時擬合模糊核與影像**，例如 SelfDeblur [85]；或是**略過模糊核**，端到端地把模糊影像映射成清晰影像，例如 Nah 等人 [44] 的多尺度網路、DeblurGAN [86]、Restormer 與 NAFNet。

僅線上版：傳統與學習式的盲去模糊方法 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/傳統與學習式的盲去模糊方法/*

僅線上版：*Deep Image Prior*：網路結構即先驗 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/deep-image-prior 網路結構即先驗/*

哪一種比較好？以模糊核為基礎的方法適合**空間均勻**的模糊，提供**可解釋**、可以檢查也可以重複使用的模糊核，但它們較慢，遇到隨位置變化的模糊就會失效。不估計模糊核的網路能處理物體運動與深度造成的**非均勻**模糊，也是 GoPro 等動態場景基準上目前最好的方法，但它們只能泛化到訓練時看過的模糊，也不給出明確的模糊核。Zhang 等人的綜述 [69] 整理了這個轉變（完整的比較在同一則補充資料中）。

程式碼：一個交替式盲反卷積的玩具範例。下面的程式用一個彎曲的 15×15 手震模糊核把 256×256 的 `camera` 影像弄糊，然後交替進行封閉形式的影像步驟（帶二次梯度懲罰、類似 CLS 的濾波器）與模糊核步驟（只用經衝擊濾波後最強的 2% 梯度做最小平方，再投影到 $h \ge 0$、$\sum h = 1$）。在 CPU 上約一秒跑完。

```
import numpy as np
from numpy.fft import fft2, ifft2
from scipy.ndimage import convolve, gaussian_filter, laplace
from skimage import data, img_as_float, transform

rng = np.random.default_rng(0)
f_true = transform.resize(img_as_float(data.camera()), (256, 256), anti_aliasing=True)
K = 15                                              # kernel support K x K
h_true = np.zeros((K, K))
for t in np.linspace(0, 1, 40):                     # a curved "camera-shake" path
    h_true[round(3 + 8 * t), round(2 + 10 * t + 2 * np.sin(3 * t))] += 1
h_true = gaussian_filter(h_true, 0.5); h_true /= h_true.sum()
g = convolve(f_true, h_true, mode="wrap") + 0.005 * rng.standard_normal(f_true.shape)

def otf(h, shape):                                  # centered kernel -> transfer function
    p = np.zeros(shape); p[:h.shape[0], :h.shape[1]] = h
    return fft2(np.roll(p, (-(h.shape[0] // 2), -(h.shape[1] // 2)), (0, 1)))

G = fft2(g)
Dx, Dy = otf(np.array([[0, -1, 1]]), g.shape), otf(np.array([[0], [-1], [1]]), g.shape)

def image_step(h, lam):        # f = argmin ||h*f - g||^2 + lam ||grad f||^2  (closed form,
 ↪  CLS-like)
    H = otf(h, g.shape)
    return np.real(ifft2(np.conj(H) * G / (np.abs(H)**2 + lam * (np.abs(Dx)**2 +
     ↪  np.abs(Dy)**2))))
```

```
def shock(f, n=5, dt=0.5):     # edge prediction: a shock filter turns soft edges into steps
    f = gaussian_filter(f, 0.7)
    for _ in range(n):
        gy, gx = np.gradient(f)
        f = f - dt * np.sign(laplace(f)) * np.hypot(gx, gy)
    return f

def kernel_step(f, keep=0.02, beta=1e-3):   # least squares in the gradient domain, strong
 ↪  edges only
    num = den = 0
    for D in (Dx, Dy):
        fx = np.real(ifft2(D * fft2(f)))
        fx[np.abs(fx) < np.quantile(np.abs(fx), 1 - keep)] = 0
        Fx = fft2(fx)
        num, den = num + np.conj(Fx) * D * G, den + np.abs(Fx)**2
    h = np.roll(np.real(ifft2(num / (den + beta))), (K // 2, K // 2), (0, 1))[:K, :K]
    h = np.maximum(h, 0)                              # project onto h >= 0 ...
    return h / h.sum()                                # ... and sum(h) = 1

def best_shift(fn):            # blind deconvolution recovers h (and f) only up to a small
 ↪  translation
    return min(fn(dy, dx) for dy in range(-3, 4) for dx in range(-3, 4))
kernel_error = lambda h: best_shift(lambda dy, dx: np.linalg.norm(np.roll(h, (dy, dx), (0,
 ↪  1)) - h_true)) / np.linalg.norm(h_true)
psnr = lambda f: -best_shift(lambda dy, dx: 10 * np.log10(np.mean((np.clip(np.roll(f, (dy,
 ↪  dx), (0, 1)), 0, 1) - f_true)[8:-8, 8:-8] ** 2)))

h = np.zeros((K, K)); h[K // 2, K // 2] = 1
h = gaussian_filter(h, 1.0); h /= h.sum()            # start from an almost-delta kernel ("no
 ↪  blur")
print(f"kernel error at start: {kernel_error(h):.2f}")
for _ in range(30):                                  # alternate: image update, then kernel
 ↪  update
    h = kernel_step(shock(image_step(h, lam=0.02)))
print(f"kernel error after 30 alternations: {kernel_error(h):.2f}")
print(f"PSNR  blurry {psnr(g):.2f} dB | estimated kernel {psnr(image_step(h, 1e-2)):.2f} dB"
      f" | true kernel {psnr(image_step(h_true, 1e-2)):.2f} dB")
# kernel error at start: 1.19
# kernel error after 30 alternations: 0.57
# PSNR  blurry 22.59 dB | estimated kernel 27.11 dB | true kernel 29.67 dB
```

模糊核的相對誤差大約減半，用估計的模糊核做最後的非盲步驟，也補回了模糊輸入與「真實模糊核」結果之間大部分的差距。但要誠實看待這個玩具範例：模糊是合成的、均勻的、循環的（所以 FFT 模型完全成立），雜訊很低，參數（`lam`、`keep`、衝擊濾波的次數）也是針對這張影像調的。程式沒有影像金字塔，所以更大的模糊核會失敗；面對真實照片時，還需要處理邊界、飽和像素，並使用穩健的損失。兩種誤差都是在最佳的小平移之後才量測，原因就是上面說的平移不確定性。

資料與里程碑

Nah 等人 [44]（CVPR 2017）用 **GoPro** 資料集處理這個問題：他們以高速攝影機拍攝清晰的影片，把連續影格平均起來，得到真實的模糊影像，並以中間那一格作為清晰目標。他們的網路是**多尺度**的：先還原粗略版本，再逐步在更細的尺度上精修，類似傳統盲去模糊由粗到細的策略，但完全不需要估計模糊核。GoPro 至今仍是最常用的去模糊基準。

今天最常被比較的基準是通用骨幹 **Restormer** [23] 與 **NAFNet** [24]。NAFNet 有一個驚人的發現：常用的非線性啟用函數（ReLU、GELU、Sigmoid）其實不是必要的，可以用兩半特徵的逐元素相乘（「SimpleGate」）取代，或直接移除，得到一個「無傳統非線性啟用函數」（nonlinear activation free）的網路，不但更簡單，在 GoPro 與 SIDD 上也更強。特徵相乘仍是非線性，名稱指的是傳統啟用函數。Zhang 等人的去模糊綜述 [69] 涵蓋了更廣的範圍（盲與非盲、影片、失焦、損失函數）。

影像修補（Inpainting，簡介）

白話版。把缺失或破損的區域（刮痕、被移除的物體、破洞）補上合理的內容。

修補是最接近影像**生成**的復原任務：在大破洞裡沒有東西可以「還原」，只能創造出和周圍相符的內容。它的退化是一個遮罩，$y = m \odot x$，其中 m 在破洞處為 0、其他地方為 1。前陣子相當知名的**遮罩自編碼器**（masked autoencoder, MAE）[45] 也可以歸類於此，可視為把修補當作預訓練任務：遮住 75% 的影像區塊，訓練 ViT 把它們重建回來，結果學到了很強的視覺表徵。實務上的修補方面，**LaMa** [46] 使用具備全影像感受野的快速傅立葉卷積，並以大型遮罩訓練，能在高解析度下填補大破洞；如今，以擴散模型為基礎的修補是大範圍、語意層級編輯的預設做法。

去背（Matting，簡介）

白話版。把前景（一個人、他的頭髮）連同柔和、半透明的邊緣一起摳出來，好貼到新背景上。

去背假設每個像素是前景與背景的混合：

$$I = \alpha F + (1 - \alpha) B,$$

其中 $\alpha \in [0, 1]$ 是不透明度（alpha matte），F 是前景顏色，B 是背景顏色。每個像素有七個未知數（α 與兩組 RGB 顏色），卻只有三個觀測值，嚴重不足，所以傳統方法要求使用者提供三**色圖**（trimap：確定前景、確定背景、不確定區域）。它也可以視為分割的一種軟性變形。**Background Matting** [47] 用一張不含主體的背景照片取代三色圖，先在合成影像上訓練網路，再以對抗損失（判斷新合成圖看起來是否自然）在真實影像上訓練。

通用骨幹網路

白話版。在上述所有任務中，總是那幾種網路設計反覆勝出。它們是通用的「影像到影像引擎」，可以拿來訓練上面任何一種退化。

骨幹	核心想法	為什麼適合復原
U-Net [48]	編碼器–解碼器，每個尺度都有跳躍連接	降採樣路徑提供大感受野，跳躍連接保留細節
SwinIR [14]	在全解析度上堆疊位移視窗自注意力的殘差群組	權重隨內容自適應；SR、去雜訊、JPEG 表現強
Restormer [23]	由 Transformer 區塊組成的 U-Net；注意力在**通道**維度計算（轉置注意力），搭配門控前饋網路	成本與像素數呈線性，可擴展到大影像
NAFNet [24]	由簡單卷積區塊組成的 U-Net，使用 LayerNorm、SimpleGate 與簡化通道注意力；移除傳統啟用函數，閘門仍有非線性	每單位 FLOP 的準確度極高；預設的「簡單基準」
MambaIR / MambaIRv2 [49], [50]	選擇性狀態空間（Mamba）掃描，加上局部增強與通道注意力；v2 加入「注意力式」非因果掃描	以線性成本取得全域感受野

反覆出現的設計理由。(1) **全域殘差**：預測 $x - y$（或 x 減去上採樣的 y），讓網路只學缺少的部分。(2) 自 EDSR [10] 以來，大多數 SR／復原網路**不用批次正規化**：訓練小塊與完整測試影像的 BN 統計量不同，會損害色彩與數值範圍的保真度。(3) 以可控成本取得**大感受野**：多尺度 U-Net、視窗或通道注意力、狀態空間掃描。復原既需要上下文（分辨紋理與雜訊），也需要全解析度的細節。(4) 用**注意力或門控**做內容自適應處理：通道注意力（RCAN）、門控（NAFNet、Restormer）。原文的「Extension」清單預言了 ViT、Flow、GAN、Diffusion 等新架構；到了 2026 年，狀態空間模型也加入 CNN 與 Transformer，成為第三種骨幹選項 [49], [50]。

All-in-one 與通用復原

白話版。不要為雨、霧、雜訊各訓練一個模型，而是訓練**一個**模型，讓它自己判斷影像哪裡出了問題，再把它修好。

動機

原文的「Extension」一節把 all-in-one 復原列為 2023 年最熱門的題目之一，大約從 2021 年開始。動機很實際：在實際部署的系統中，我們事先不知道某張影像有哪一種退化（或哪幾種混合），各種天氣也可能隨時出現或同時發生。一整組任務專屬模型需要前面再接一個可靠的分類器，還浪費記憶體；一個能自我調整的模型方便得多。困難在於**任務干擾**：不同退化需要不同、甚至相反的操作（去雜訊是平滑，去模糊是銳化），所以直接做多任務訓練的效果會輸給專用模型。

代表方法

- **AirNet** [51]（CVPR 2022）：以對比學習訓練的退化編碼器，以成對修復資料與對比裁切樣本學到影像「**如何**」受損，並用它引導復原網路；它的雜訊＋雨＋霧設定成為標準的三任務基準。
- **TransWeather** [52]（CVPR 2022）：一個 Transformer 處理雨、霧、雪，使用可學習的天氣類型查詢。

- **PromptIR** [53]（NeurIPS 2023）：輕量的提示區塊把退化相關的資訊注入 Restormer 風格的解碼器。
- **AdaIR** [54]（ICLR 2025）：不同退化影響不同的頻帶，所以分別自適應地調整低頻與高頻特徵。
- **DA-CLIP** [55]（ICLR 2024）：調整過的 CLIP 模型提供退化嵌入，用來引導復原網路。
- **InstructIR** [56]（ECCV 2024）：以一句自然語言指令（「把雨去掉」）作為網路的條件。

僅線上版：*All-in-one* 方法的細節 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/all-in-one-方法的細節/*

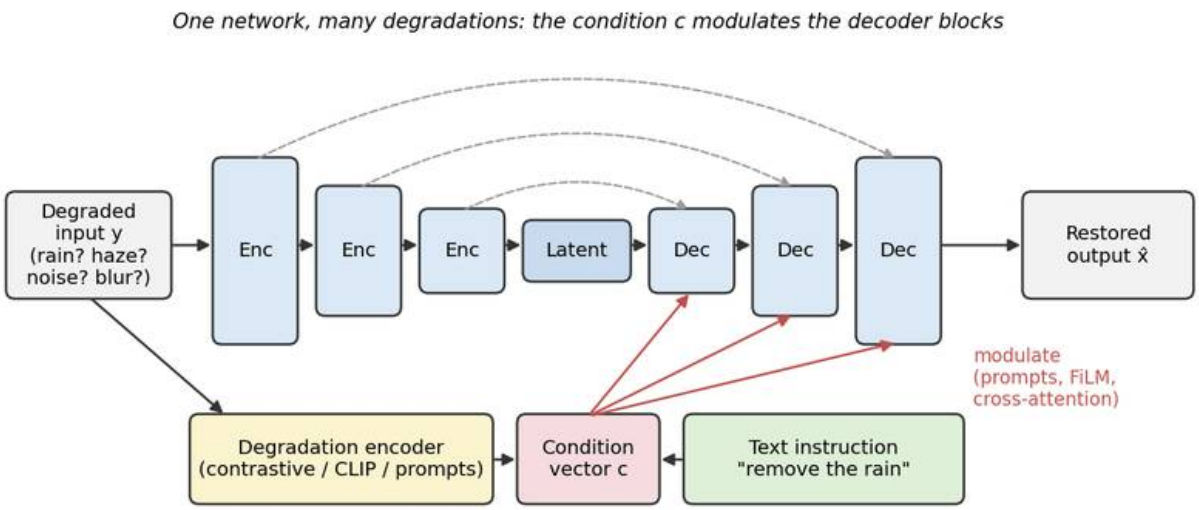


圖 5 — *All-in-one* 復原背後的共同樣板：編碼器–解碼器骨幹，加上一個能感知退化的條件 c。c 可以從影像推論（*AirNet*、*PromptIR*、*AdaIR*、*DA-CLIP*），也可以由文字給定（*InstructIR*），用來調變解碼器。

原文的動機仍是核心；改變的是「**由誰**決定退化類型」。它從固定的任務標籤，轉移到學習出的提示、視覺語言嵌入，最後到人類的指令，背後往往有大型預訓練模型支撐。All-in-one 綜述 [72] 對此有詳細討論。

生成式與工具式擴充。 AutoDIR [119] 結合自動品質評估、潛在擴散與結構修正；PromptFix [122] 結合使用者指令、視覺輔助描述與高頻保留。AgenticIR [120]、4KAgent [121] 則安排並評估預訓練工具；規劃迴圈與多個候選的成本，與整合骨幹單次前向運算不同。

僅線上版：*Restormer*：以通道關係取代逐像素注意力 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/restormer 以通道關係取代逐像素注意力/*

僅線上版：*AirNet*：先推估退化，再調整修復方式 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/airnet 先推估退化再調整修復方式/*；*TransWeather*：用學得的天候查詢共用修復模型 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/transweather 用學得的天候查詢共用修復模型/*；*PromptIR*：依輸入產生特徵 *prompt*，不需要文字指令 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/promptir 依輸入產生特徵-prompt 不需要文字指令/*；*AdaIR*：自適應分離頻率，再交換互補資訊

— https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/adair 自適應分離頻率再交換互補資訊/

僅線上版：*DA-CLIP*：把內容引導與退化引導分開 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/da-clip 把內容引導與退化引導分開/；*InstructIR*：以人類指令引導影像特徵路由 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/instructir 以人類指令引導影像特徵路由/；*AutoDIR*：診斷退化、引導擴散，再修正結構 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/autodir 診斷退化引導擴散再修正結構/；*PromptFix*：結合指令、視覺診斷與高頻保留 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/promptfix 結合指令視覺診斷與高頻保留/

僅線上版：*AgenticIR*：規劃、檢查，再修訂修復工具順序 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/agenticir 規劃檢查再修訂修復工具順序/；*4KAgent*：依忠實度目標選擇修復專家 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/4kagent 依忠實度目標選擇修復專家/

損失函數

白話版。損失函數決定了什麼叫「好」。像素損失獎勵平均而言接近真實；感知與對抗損失獎勵看起來真實；頻率損失獎勵把細節做對。

像素損失。令 $\hat{x} = f_\theta(y)$，

$$\ell_2 = \frac{1}{P}\sum_p (\hat{x}_p - x_p)^2, \qquad \ell_1 = \frac{1}{P}\sum_p |\hat{x}_p - x_p|, \qquad \ell_\varepsilon = \frac{1}{P}\sum_p \sqrt{(\hat{x}_p - x_p)^2 + \varepsilon^2},$$

其中 P 是像素值的總數，ε 是一個小常數，讓最後一個成為平滑、可微分的 L1 版本。L2 直接最佳化 PSNR，但對大誤差懲罰很重，而且會收斂到後驗平均。Zhao 等人 [57] 證明在架構固定的情況下，光是換損失函數就能明顯改變結果，而且 L1 與受感知啟發的損失（包括和 SSIM 的組合）常常勝過 L2，甚至在與 L2 相關的指標上也是。L1 是現代大多數 SR 與復原論文的預設選擇。

感知（特徵）損失。在預訓練分類網路 ϕ（通常是 VGG）的特徵空間中比較影像 [58]：

$$\ell_{\text{feat}} = \frac{1}{C_j H_j W_j}\|\phi_j(\hat{x}) - \phi_j(x)\|_2^2,$$

其中 ϕ_j 是第 j 層的特徵圖，有 C_j 個通道、尺寸為 $H_j \times W_j$。它能容忍小的位移，獎勵的是邊緣與紋理吻合，而不是數值完全相同。

對抗損失。判別器 D_ψ 學習分辨復原影像與真實影像，復原網路則學著騙過它，例如 $\ell_{\text{adv}} = -\log D_\psi(\hat{x})$。這會把輸出推向自然影像的流形上（SRGAN [12]、ESRGAN [13]、Real-ESRGAN [4]），代價是可能出現憑空捏造的紋理。實務上以 GAN 為基礎的復原網路會使用加權和，例如 $\ell = \ell_1 + \lambda_p \ell_{\text{feat}} + \lambda_a \ell_{\text{adv}}$，權重為 λ_p, λ_a。

頻率損失。網路傾向先擬合低頻，對細節擬合不足。在傅立葉轉換 $\mathcal{F}$ 上加一項損失，例如對複數值或振幅計算 $\|\mathcal{F}(\hat{x}) - \mathcal{F}(x)\|_1$，就能直接懲罰缺少的高頻。**焦點頻率損失**（focal frequency loss）[59] 更進一步，自適應地加重目前最難重建的頻率成分。頻率項現在在去模糊與 all-in-one 模型中很常見。

評估指標

白話版。「復原結果是否接近真實？」與「它看起來像不像真的照片？」是兩個不同的問題，由不同的指標回答，而且你無法同時把兩者做到最好。

全參考指標

PSNR（峰值訊雜比）：

$$\mathrm{PSNR}(x, \hat{x}) = 10 \log_{10} \frac{\mathrm{MAX}^2}{\mathrm{MSE}(x, \hat{x})},$$

其中 MAX 是像素可能的最大值（1 或 255），MSE 是均方誤差。單位是 dB；提升 1 dB 約等於 MSE 降低 21%。

SSIM（結構相似度）[60] 比較局部平均、變異數與共變異數：

$$\mathrm{SSIM}(a, b) = \frac{(2\mu_a\mu_b + C_1)(2\sigma_{ab} + C_2)}{(\mu_a^2 + \mu_b^2 + C_1)(\sigma_a^2 + \sigma_b^2 + C_2)},$$

在兩張影像的局部視窗 a、b 上計算再取平均，其中 μ 是局部平均，σ^2 是局部變異數，σ_{ab} 是局部共變異數，C_1, C_2 是維持數值穩定的小常數。它和人眼感受到的結構相關性比 PSNR 好，但仍然偏好平滑的結果。

LPIPS [61] 衡量深度網路特徵之間的距離（依人類相似度判斷校準）。越低越好，它會獎勵 PSNR 所懲罰的逼真紋理。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, img_as_float32
from skimage.metrics import peak_signal_noise_ratio, structural_similarity

x = img_as_float32(data.camera())
y = np.clip(x + np.random.default_rng(0).normal(0, 0.1, x.shape), 0, 1).astype(np.float32)
x_hat = ndi.gaussian_filter(y, 1.0)

def psnr(ref, est, peak=1.0):
    mse = np.mean((ref.astype(np.float64) - est) ** 2)
    return 10 * np.log10(peak ** 2 / mse)

print(f"PSNR  ours {psnr(x, x_hat):.2f} dB | skimage {peak_signal_noise_ratio(x, x_hat,
 ↪  data_range=1):.2f} dB")
print(f"SSIM  {structural_similarity(x, x_hat, data_range=1):.3f}")
# PSNR  ours 27.20 dB | skimage 27.20 dB
# SSIM  0.639
```

一定要寫清楚計算慣例（RGB 或 Y 通道、邊界裁切、數值範圍、是否先四捨五入成 8 位元）：光是這些差異，就足以讓論文中的 PSNR 相差零點幾 dB。

無參考指標

白話版。無參考指標只從復原影像估計品質，適合沒有乾淨參考的情況，但無法證明還原細節正確。

全參考、部分參考與無參考

影像品質評估（image quality assessment, IQA）方法依照它們對原始影像 x 知道多少來分類：

- **全參考（full-reference, FR）**：完整的 x 可用，指標逐像素或逐特徵比較 $\hat{x}$ 與 x（PSNR、SSIM [60]、LPIPS [61]）。合成基準用的就是這一類。
- **部分參考（reduced-reference, RR）**：只有 x 的少數摘要特徵可用（例如隨壓縮視訊一起傳送的統計量），指標在 $\hat{x}$ 上計算同樣的特徵再比較。
- **無參考（no-reference, NR，盲）**：只有 $\hat{x}$ 可用。指標必須自帶一套「什麼叫好」的模型。

復原需要 NR 指標有三個理由。(1) **真實測試影像沒有真值**：真實世界 SR、去雨、除霧都在沒有人擁有乾淨版本的照片上評估。(2) **參考影像本身可能不完美，或只是眾多合理答案之一**：GAN 或擴散模型的輸出可能看起來非常好，PSNR 卻很低，因為它的紋理與參考影像不同（見下方的感知–失真權衡）。(3) **NR 分數可以在執行時引導系統**，例如在沒有參考影像時，從多個輸出或參數中挑一個。代價是：NR 指標是人類意見的模型，而每個模型都有盲點。

如何驗證 NR 指標：與人類意見的相關性

白話版。只有當一個 NR 指標對影像的排序和人一致時，我們才信任它。所以研究者收集大量影像的大量人類評分，再檢查指標分數與評分有多一致。

平均意見分數。在主觀實驗中，每張影像 i 由多位觀察者評分，平均分就是它的**平均意見分數**（mean opinion score），$\mathrm{MOS}_i = \frac{1}{R}\sum_{r=1}^{R} s_{ir}$，其中 s_{ir} 是評分者 r 給的分數，R 是評分者人數。有些資料集公布的是**差異**分數（DMOS），以失真影像與其參考影像評分之差為基礎；DMOS 通常越低越好。

SRCC（Spearman 等級相關係數）衡量指標是否把影像排成和人相同的順序：

$$\mathrm{SRCC} = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n\,(n^2-1)},$$

其中 n 是測試影像數，d_i 是影像 i 依預測分數的名次與依 MOS 的名次之差（這個簡式假設沒有同名次；有同名次時改算名次的 Pearson 相關）。SRCC 對分數做任何單調轉換都不變，因此它衡量的是**預測的單調性**。

PLCC（Pearson 線性相關係數）衡量分數之間有多線性地一致：

$$\mathrm{PLCC} = \frac{\sum_{i=1}^{n}(q_i-\bar{q})(m_i-\bar{m})}{\sqrt{\sum_{i=1}^{n}(q_i-\bar{q})^2}\,\sqrt{\sum_{i=1}^{n}(m_i-\bar{m})^2}},$$

其中 q_i 是指標給影像 i 的分數，m_i 是它的 MOS，$\bar{q}$、$\bar{m}$ 是兩者的平均。由於各指標的尺度是任意的，常見做法是先用一個擬合出的單調函數（通常是 logistic 函數）轉換 q_i，再在轉換後的分數上

算 PLCC，因此 PLCC 衡量的是校準後的**預測準確度**。兩者都落在 $[-1,1]$。對 NIQE 這種「越低越好」的指標，與 MOS 的原始相關係數是負的，論文會報告其絕對值。指標通常在資料集的一部分上訓練、在其餘部分測試，而跨資料集的相關性才是對泛化能力更誠實的檢驗。

人類評分資料集。 NR 指標在附有人類評分的影像品質資料集上訓練與測試。對復原而言，重點是失真從哪裡來：合成損壞（LIVE [104]、TID2013 [105]、KADID-10k [106]）、真實相機照片（KonIQ-10k [107]、SPAQ [108]），或是復原演算法本身的輸出（PIPAL [109]）。只有最後這一類資料集，才包含復原論文真正需要評分的那種影像。

僅線上版：人類評分的 *IQA* 資料集 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/人類評分的-iqa-資料集/*

第一類：自然場景統計（NSS）模型

白話版。乾淨的照片有一些隱藏的統計習慣。把照片的局部亮度與局部對比去掉之後，剩下的部分會呈現某種典型形狀的鐘形直方圖，而且相鄰的剩餘值彼此只有微弱的關聯。模糊、雜訊與壓縮會以各自特有的方式改變這個形狀。NSS 指標就是量測這個形狀，再問它有多不尋常。

MSCN 係數。對灰階影像 I，**去均值對比正規化**（mean-subtracted contrast-normalized, MSCN）係數為 [94]

$$\hat{I}(i,j)=\frac{I(i,j)-\mu(i,j)}{\sigma(i,j)+C},\qquad \mu(i,j)=\sum_{k,l} w_{k,l}\,I(i+k,j+l),\qquad \sigma(i,j)=\sqrt{\sum_{k,l} w_{k,l}\,\big(I(i+k,j+l)-\mu(i,j)\big)^2},$$

其中 (i,j) 是像素位置，w 是權重總和為 1 的小型圓對稱高斯視窗，μ 與 σ 是局部加權平均與標準差，C 是一個小常數（8 位元影像取 1），避免在平坦區域除以零。這是一種除法正規化，大致模仿人類早期視覺中的對比增益控制。

廣義高斯擬合。$\hat{I}$ 的直方圖以零均值的廣義高斯分布（generalized Gaussian distribution, GGD）建模：

$$f(t;\alpha,\sigma^2)=\frac{\alpha}{2\beta\,\Gamma(1/\alpha)}\exp\!\Big(-(|t|/\beta)^{\alpha}\Big),\qquad \beta=\sigma\sqrt{\frac{\Gamma(1/\alpha)}{\Gamma(3/\alpha)}},$$

其中 t 是一個 MSCN 值，$\alpha>0$ 是**形狀**參數（$\alpha=2$ 為高斯，$\alpha=1$ 為拉普拉斯；α 越小直方圖越尖、尾巴越重，越大則頂部越平），σ^2 是變異數，β 是對應的尺度，Γ 是 gamma 函數。估計 α 的簡單方法是**動差匹配**（moment matching）：比值 $\mathbb{E}[t^2]/(\mathbb{E}|t|)^2=\Gamma(1/\alpha)\,\Gamma(3/\alpha)/\Gamma(2/\alpha)^2$ 只與 α 有關，所以先從資料算出左邊，再搜尋能對上的 α。

BRISQUE [94]（Blind/Referenceless Image Spatial Quality Evaluator）計算 MSCN 係數的 GGD 形狀與變異數，再加上對相鄰 MSCN 係數乘積（水平、垂直與兩個對角方向）擬合非對稱 GGD 得到的參數，並在兩個影像尺度上重複。一個以合成失真影像的人類評分訓練的支持向量迴歸器（SVR），把這些特徵對應成品質分數。越低越好。它是**需要意見分數的**（opinion-aware）：訓練時需要人類評分。

NIQE [62]（Natural Image Quality Evaluator）則**不需要意見分數**（opinion-unaware）：它從未看過人類評分或失真影像。它在清晰的區塊上計算類似的 NSS 特徵，對一組原始影像的特徵擬合一個均值 ν_1、共變異數 Σ_1 的多變量高斯（multivariate Gaussian, MVG），對測試影像的區塊擬合另一個 MVG (ν_2,Σ_2)，然後報告

$$D(\nu_1, \nu_2, \Sigma_1, \Sigma_2) = \sqrt{(\nu_1 - \nu_2)^\top \left(\frac{\Sigma_1 + \Sigma_2}{2}\right)^{-1} (\nu_1 - \nu_2)},$$

也就是兩個高斯之間類似 Mahalanobis 距離的量（$^\top$ 為轉置）。越低越好；零代表「統計上和原始影像集一樣」。

還有兩個在復原論文中常見的親戚。**PIQE** [96] 逐區塊套用 MSCN 統計量，不需要意見分數，還能給出影像哪裡看起來失真的分布圖。2018 年 PIRM 超解析挑戰賽 [63] 的**感知指數**（Perceptual Index, PI）把 NIQE 與 Ma 等人 [95] 從超解析影像的人類評分學出的分數結合起來；PI 越低代表感知品質越好，許多感知 SR 論文至今仍報告它。

僅線上版：*PIQE 與感知指數 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/piqe-與感知指數/*

程式：實際看看 MSCN 統計量。下面的程式在 `camera` 及其模糊、加雜訊、過度銳化的版本上，計算 MSCN 係數、以動差匹配估計的 GGD 形狀、MSCN 變異數，以及水平相鄰係數的相關（BRISQUE 成對特徵的簡化版）：

```
import numpy as np
from scipy.ndimage import gaussian_filter
from scipy.special import gamma
from skimage import data, img_as_float

def mscn(img, sigma=7/6, C=1.0):
    """Mean-subtracted contrast-normalized coefficients of a grayscale image in [0, 255]."""
    mu = gaussian_filter(img, sigma)
    sd = np.sqrt(np.abs(gaussian_filter(img * img, sigma) - mu * mu))
    return (img - mu) / (sd + C)

def ggd_shape(x):
    """Moment-matching estimate of the generalized Gaussian shape alpha (2 = Gaussian)."""
    alphas = np.arange(0.2, 10.0, 0.001)
    r_theory = gamma(1 / alphas) * gamma(3 / alphas) / gamma(2 / alphas) ** 2
    r_data = np.mean(x ** 2) / np.mean(np.abs(x)) ** 2
    return alphas[np.argmin((r_theory - r_data) ** 2)], np.mean(x ** 2)

rng = np.random.default_rng(0)
clean = img_as_float(data.camera()) * 255
versions = {
    "clean": clean,
    "blur (sigma=2)": gaussian_filter(clean, 2),
    "noise (sigma=20)": np.clip(clean + rng.normal(0, 20, clean.shape), 0, 255),
    "over-sharpened": np.clip(clean + 2.0 * (clean - gaussian_filter(clean, 2)), 0, 255),
}
for name, img in versions.items():
    m = mscn(img)
    alpha, var = ggd_shape(m)
    rho = np.mean(m[:, :-1] * m[:, 1:]) / var           # correlation of horizontal neighbors
    print(f"{name:17s} GGD shape {alpha:4.2f} | MSCN variance {var:4.2f} | neighbor corr.
     ↪ {rho:+.2f}")
# clean             GGD shape 1.56 | MSCN variance 0.28 | neighbor corr. +0.12
# blur (sigma=2)    GGD shape 0.87 | MSCN variance 0.04 | neighbor corr. +0.89
# noise (sigma=20)  GGD shape 3.12 | MSCN variance 0.68 | neighbor corr. -0.10
```

```
# over-sharpened    GGD shape 2.35 | MSCN variance 0.43 | neighbor corr. +0.09
```

每種失真都把特徵推往自己的方向（圖 6）。**模糊**去掉局部對比，大部分 MSCN 值縮向零：變異數很小、直方圖非常尖（α 小）、相鄰值高度相關。**雜訊**加入彼此獨立的能量：變異數變大、直方圖頂部變平（α 大於 2）、相鄰相關接近零（略為負值，因為減去局部平均會讓相鄰值彼此牽連）。**過度銳化**則把特徵推往和雜訊相同的方向。這正是 NSS 指標用於復原時的核心弱點：它分不出復原出來的紋理和憑空加上的高頻能量。另外要注意，絕對數值取決於影像內容與 C（乾淨的 `camera` 影像並不是單位高斯），這就是為什麼 BRISQUE 需要訓練迴歸器、NIQE 需要原始參考影像集，而不是使用固定門檻。

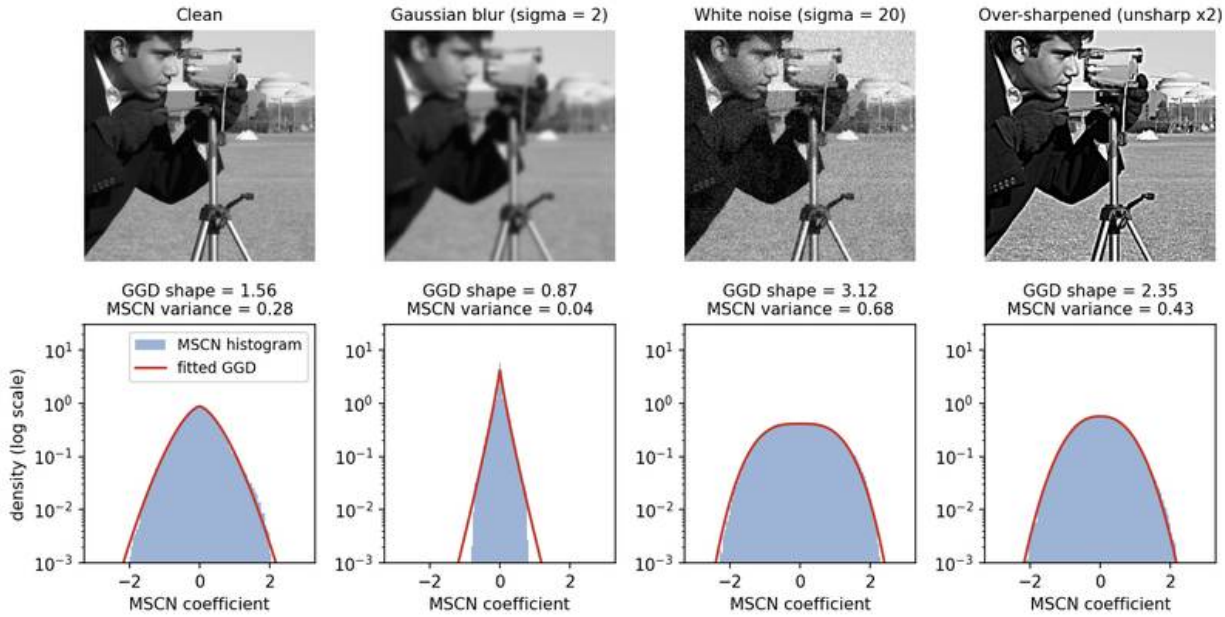


圖 *6 — MSCN 統計量，BRISQUE、NIQE 與 PIQE 的基礎。模糊讓直方圖變窄變尖；雜訊與過度銳化都讓它變寬變平，因此 NSS 指標會把兩者看成相似的變化。*

第二類：學習式深度模型

白話版。不用手工設計的統計量，而是訓練神經網路直接預測人類評分。深度特徵更能掌握內容、語意與複雜的瑕疵，但模型只懂它訓練時看過的影像與意見。

五個常用的例子：**NIMA** [97] 預測人類評分的整個分布，而不是單一數字；**HyperIQA** [98] 依畫面內容調整品質判斷規則，因為同樣的模糊出現在天空上和臉上，給人的感受不同；**MUSIQ** [99] 以原始解析度與長寬比讀取影像，不會因縮放而改變正要被評估的瑕疵；**MANIQA** [100] 是一個 transformer，贏得 NTIRE 2022 感知 IQA 挑戰賽的無參考賽道，該賽事資料包含復原與 GAN 的輸出；**TOPIQ** [101] 則讓高階語意決定哪些低階失真比較重要。

這些模型都是**需要意見分數的**，輸出「越高越好」的分數，而且都依賴訓練資料集：同一架構在 KonIQ-10k 上訓練與在 PIPAL 上訓練，可能會給復原方法不同的排名。

僅線上版：學習式 *NR* 評估器如何運作 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/學習式-nr-評估器如何運作/*

第三類：視覺–語言與大型多模態模型

白話版。用大量圖文資料訓練出的模型，已經「知道」好照片和壞照片長什麼樣子。你可以直接問它們：把影像和文字提示比較，或乾脆請一個聊天機器人式的模型打分數。

CLIP-IQA [102] 比較影像的 CLIP 嵌入與一對反義提示（例如「Good photo.」與「Bad photo.」）的嵌入，再把兩個相似度轉成 0 到 1 之間的分數（越高越好）。基本版本完全不需要意見分數；換用其他提示對（「Sharp photo.」/「Blurry photo.」）可以探測單一屬性，而提示的措辭會改變分數。**Q-Align** [103] 微調一個大型多模態模型，讓它像人類評分者一樣以 bad 到 excellent 的評分等級回答，再把它對這五個字的機率轉成分數。這類模型跨資料集表現強，但運算量大，也會繼承基礎模型的偏誤。

僅線上版：*CLIP-IQA* 與 *Q-Align* 的分數 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/clip-iqa-與-q-align-的分數/*

分布層級的指標

有些指標評的是一組輸出而非單張影像。**FID** [64] 比較復原結果集合與一組真實影像在 Inception 特徵上的分布；它原本是為生成模型設計的，現在用於擴散 SR 這類生成式復原，但它需要大量影像，而且無法告訴你任何單張影像的好壞。原文也提到的 **IS**（Inception Score）衡量類別預測的信心與多樣性，比較適合類別條件式生成而非復原，如今在復原任務中很少使用。

使用者研究與成對比較

白話版。好不好看，最終由人決定。兩種標準的問法是：請人在量表上打分數，或請人從兩張中選出比較好的一張。這樣的研究也是上述每個 NR 指標用來驗證的標尺，更是新方法能提出的最有力證據。

平均意見分數（MOS）。N 位評分者各自給一張影像一個固定量表上的分數 s_n（常見為 1 到 5），MOS 就是這些分數的平均，定義同上面驗證指標時的說明。ITU-R BT.500 [111] 為這類測試訂定標準：觀看條件、評分量表、單刺激與雙刺激以及比較式方法，還有篩除不可靠評分者的程序。

成對比較。絕對量表在不同評分者之間會漂移；從兩張中選出比較好的一張比較容易，也比較一致。**Bradley–Terry** 模型 [112] 把大量這樣的選擇轉換成每個方法 i 的一個強度 $\pi_i > 0$，擬合的目標是讓方法 i 勝過方法 j 的機率等於 $\pi_i/(\pi_i + \pi_j)$。依擬合出的強度排序，就得到一個一致的排名，即使每位評分者一次只比較過兩張影像。

僅線上版：擬合 *Bradley–Terry* 模型 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/擬合-bradleyterry-模型/*

良好的做法：隱藏方法名稱、隨機化順序與左右位置、同時顯示輸入以提供脈絡、使用足夠多的影像與評分者、回報信賴區間或顯著性，並說明顯示器與觀看條件。只有少數評分者與精挑細選影像的研究，只能算展示，不能算證據。

常見陷阱

- **指標可以被刻意迎合**。一個會加顆粒或銳化邊緣的復原模型可以改善 NSS 分數（上面的程式就顯示了這點），卻沒有變得更忠實。任何可微分的指標（以及任何學習式 IQA 網路）一旦被拿來當訓練損失函數或用於模型選擇，都可能被直接最佳化，網路學會的是討好指標而不是討好人。如果一個方法在損失函數中用了 CLIP 或某個 IQA 網路，再用同一個模型評估它就不是獨立的評估。
- **領域不匹配**。大多數需要意見分數的指標是在合成失真（LIVE、TID2013、KADID-10k）或相機照片（KonIQ-10k、SPAQ）上訓練的。復原輸出的瑕疵不一樣：憑空生成的紋理、過度平滑、GAN 特有的圖樣。這種不匹配正是 PIPAL [109] 明確的動機：其作者發現既有的 IQA 方法無法公平評估以 GAN 為基礎的復原。低光源的問題則在於曝光與色彩，而大多數 NSS 特徵會把它們正規化掉；低光源專屬的討論請見 DLCV 09C。
- **指標彼此不一致**。 NSS、學習式與視覺–語言指標常把同一批方法排出不同的順序，而且都可能與人的判斷不同。分數也取決於實作、權重檔、色彩空間、輸入尺寸以及影像是否被縮放，所以從不同論文抄來的數字往往無法直接比較。
- 「**NR 品質較高**」**不等於**「**較正確**」。 NR 指標無法知道一張清晰的臉或一行文字是否和真實場景相符。憑空生成的車牌也可能拿到滿分。

建議做法

1. 有參考影像時，以 FR 指標（PSNR、SSIM、LPIPS）為主，NR 指標作為補充；並在感知–失真平面上標出你的方法（見下一小節）。
2. 在沒有參考影像的真實影像上，報告**多個來自不同類別的 NR 指標**（例如 NIQE 或 PI、一兩個學習式模型如 MUSIQ、MANIQA 或 TOPIQ，以及一個視覺–語言指標如 CLIP-IQA 或 Q-Align），並註明每個指標是越高還是越低越好。
3. 使用同一套實作並寫明版本與權重檔。開源的 **IQA-PyTorch** 工具箱（`pyiqa`）[110] 以統一介面重新實作了本節大多數的指標（NIQE、BRISQUE、PIQE、Ma 分數與 PI、NIMA、HyperIQA、MUSIQ、MANIQA、TOPIQ、CLIP-IQA、Q-Align，以及 FR 指標）。
4. 不要對著裁判調參：至少保留一個完全沒參與訓練或模型選擇的指標。
5. 重要的主張要有**使用者研究**佐證（平均意見分數或成對比較，盲測且隨機化）；上面的使用者研究小節介紹了 MOS 與 Bradley–Terry 成對比較的流程。指標之間或指標與人不一致時，要明白說出來。

總覽表

四個類別一覽：

類別	例子	需要人類評分嗎？	方向	常見盲點
NSS	NIQE、BRISQUE、PIQE、PI	NIQE 與 PIQE 不需要；BRISQUE 與 PI（透過 Ma）需要	越低越好	獎勵添加的顆粒與銳化；把曝光與色彩正規化掉
學習式深度模型	NIMA、HyperIQA、MUSIQ、MANIQA、TOPIQ	需要	越高越好	取決於訓練資料；復原瑕疵不同於訓練時的失真

類別	例子	需要人類評分嗎？	方向	常見盲點
視覺–語言	CLIP-IQA、Q-Align	CLIP-IQA（零樣本）不需要；Q-Align 需要	越高越好	提示措辭；基礎模型的偏誤；運算量大
分布層級	FID	不需要（使用預訓練 Inception 特徵）	越低越好	需要大量影像；無法評判單張影像

僅線上版：逐一指標的總覽表 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/notes/逐一指標的總覽表/*

感知–失真權衡

Blau 與 Michaeli [65] 把這個矛盾形式化。令 $\Delta(x, \hat{x})$ 為任意失真度量（MSE、$1 - $ SSIM……），$d(p_X, p_{\hat{X}})$ 為真實影像分布與復原影像分布之間的散度（感知品質：d 越小，輸出在統計上越無法與真實影像區分）。**感知–失真函數**定義為

$$P(D) = \min_{p_{\hat{X}|Y}} d(p_X, p_{\hat{X}}) \quad \text{subject to} \quad \mathbb{E}[\Delta(X, \hat{X})] \le D,$$

其中最小值取遍所有（可能帶隨機性的）復原方法。他們證明 $P(D)$ 是非遞增的，而且（當散度對第二個引數為凸時）是凸函數：原點附近有一塊無法達到的區域。一個方法不可能同時擁有最小失真與完美的真實感。特別是 MMSE 估計 $\mathbb{E}[x \mid y]$ 的失真最低，但輸出分布模糊、不真實；想看起來真實，就必須接受較高的 MSE。GAN（以及現在的擴散模型）是沿著這條邊界移動的原則性方法。

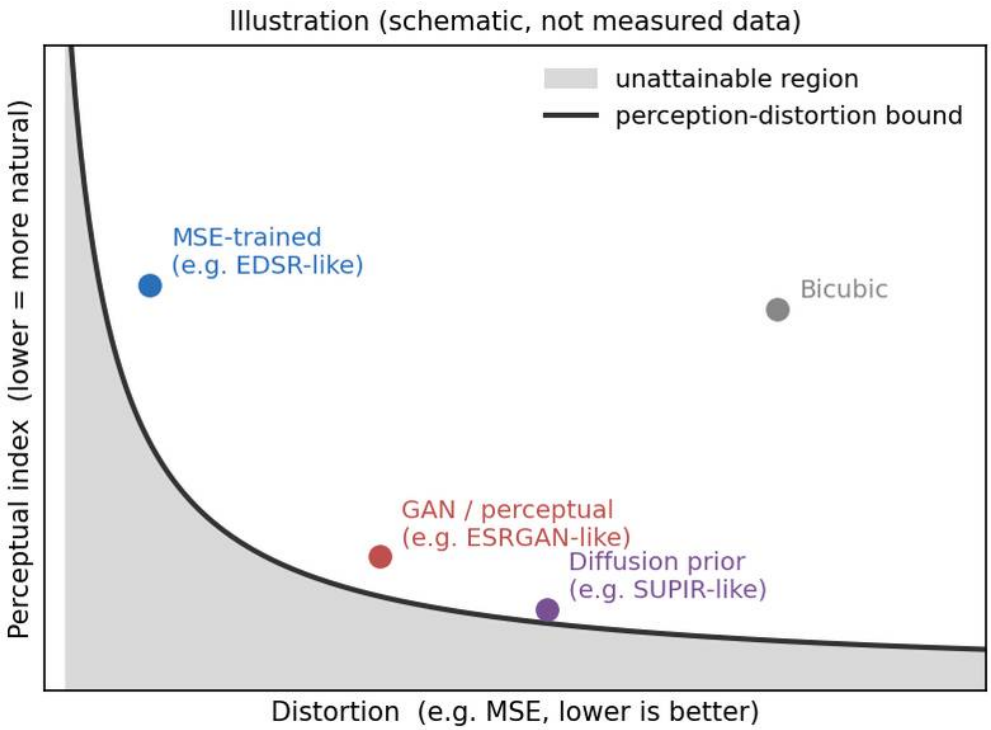


圖 7 — 感知–失真權衡的示意圖 [65]（非實測資料）。邊界是凸且遞減的；各方法在保真度（左）與真實感（下）之間取捨。

這正是原文評估指標一節的更新：擴散模型能「生成」看似合理但不一定忠於原圖的細節，保真度（PSNR ／ SSIM）與感知品質之間的落差因此更受重視。**NTIRE 2025** ×4 SR 挑戰賽就分成兩個賽道：以 PSNR 排名的 restoration track，以及以感知分數排名的 perceptual track [3]；2024 年的版本則以 PSNR 排名 [2]。實務建議：同時報告一個失真指標與一個感知指標，並說明你的方法瞄準權衡曲線上的哪個位置。

NTIRE 挑戰賽

白話版。每年 CVPR 的一個 workshop 會在固定資料集上舉辦公開競賽，並發表報告說明勝出的想法：這是了解「現在什麼有用」最簡單的方式。

既然依任務分類的排行榜已經消失，NTIRE 挑戰賽報告就是追蹤復原領域的好管道。每份報告都會說明任務、資料、評估方式，以及名列前茅隊伍的方法。本篇引用的例子：

- **NTIRE 2017 SR** [19]：推出 DIV2K；EDSR [10] 奪冠。
- **NTIRE 2024 ×4 SR** [2]：在 DIV2K 上做雙三次 ×4，以 PSNR 排名。
- **NTIRE 2025 ×4 SR** [3]：restoration（PSNR）與 perceptual 兩個賽道分開評比。
- **NTIRE 2025 雨滴去除** [40]：真實的白天與夜晚雨滴影像，並有雙重對焦設定。
- **NTIRE 2024 與 2025 低光增強**：這兩屆低光挑戰賽的摘要請見 DLCV 09C。

近年的報告有兩個共同模式：勝出的作品通常是上述骨幹（HAT、Restormer、NAFNet 或 Mamba 風格，有時再加上擴散先驗）的集成或大型混合模型，並使用大量資料與算力訓練；而挑戰賽也越來越常使用真實拍攝的資料與感知賽道。

現代觀點

原文最後列出了延伸方向（醫學影像、多模態、非監督、新架構、all-in-one、低階結合高階、即時與低運算、水下）。其中大多數現在都是活躍的研究領域。以下依任務整理綜述文獻，再談 2023–2026 年的最新進展與尚未解決的問題。

綜述論文

- **超解析度**（Wang、Chen 與 Hoi [66]）：把 SR 模型描述成在上採樣位置與運算、網路設計、損失函數與訓練策略上的選擇，並點名非監督／真實世界 SR 為主要的開放方向。
- **去雜訊**（Tian 等人 [67]；Elad、Kawar 與 Vaksman [68]）：瓶頸往往是雜訊模型的真實程度；去雜訊器如今是其他反問題與擴散模型的基本元件，前沿也移向從後驗分布取樣。
- **去模糊**（Zhang 等人 [69]）：動態場景去模糊已從估計模糊核，轉向以 GoPro 這類實拍配對訓練的端到端多尺度網路。
- **除霧**（Gui 等人 [70]）與**除雨**（Yang 等人 [71]）：物理引導與無物理的設計並存；真實的退化模型與真實世界的評估，比在合成資料上繼續微調架構更重要。

- **All-in-one**（Jiang 等人 [72]）：依架構、學習範式與核心創新分類；各論文的任務組合不一致，讓比較變得困難，對未見過與混合退化的泛化仍是開放問題。
- **擴散式復原**（Li 等人 [73]）：感知品質最好，但成本高，也有保真度與捏造細節的疑慮；蒸餾成少步驟與更好的評估是關鍵方向。

僅線上版：綜述論文的詳細介紹 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-restoration/papers/綜述論文的詳細介紹/*

2023–2026 年的最新進展

1. **All-in-one 與盲復原成為主流設定**，不再是小眾題目。方法從學習式退化表徵（AirNet）發展到提示（PromptIR）、頻率感知的自適應（AdaIR）與語言引導（DA-CLIP、InstructIR）。
2. **以預訓練擴散模型作為生成先驗**，大幅改變了真實世界 SR 與盲復原（DiffBIR、SUPIR），一步式變體（OSEDiff）則把推論成本從數十、數百步降到一步。
3. **狀態空間模型**（MambaIR、MambaIRv2）加入 CNN 與 Transformer，成為以線性成本取得全域感受野的骨幹選項。
4. **評估方式跟上了生成能力**：挑戰賽把保真度賽道與感知賽道分開 [3]，真實拍攝的資料集（SIDD、O-HAZE、Dense-Haze、Raindrop Clarity）也和合成基準一起成為標準。

尚未解決的問題

- **真實世界的泛化**。合成流程仍然漏掉真實相機的影像處理、隨位置變化的模糊與混合天氣；模型遇到沒見過的組合時表現會下滑。
- **保真度 vs. 捏造**。生成先驗可能憑空創造文字、人臉或紋理。在醫學、鑑識與科學用途中這是不能接受的，而我們還缺少衡量「忠實又清晰」的好指標。
- **評估**。 PSNR ／ SSIM 獎勵模糊；無參考指標可以被刻意迎合；人類評測成本高。對生成式復原而言，指標與人類偏好的一致性仍然很弱。
- **效率**。在手機與相機上做高解析度、即時（以及影片）復原仍是很強的限制，對 Transformer 與擴散模型尤其如此。
- **All-in-one 模型的任務干擾與擴展**，以及如何在不重新訓練的情況下加入新的退化。
- **低階結合高階**。讓復原幫助下游偵測或分割，而不是給人看，是另一種目標（任務導向評估，例如 RESIDE [32]）。

原文的觀點

原文以兩個至今仍成立的觀察作結。第一，由於每個任務都簡單明瞭，現階段 SOTA 的比拼大多靠特殊的架構取勝，也就代表大量的算力與複雜的實驗設計；但這些任務在應用上都相當實用且容易上手，所以在**應用面**與**任務的組合**上還有很多著墨空間（all-in-one 復原正是往這裡發展）。第二，相較於高階任務，這個領域的學習成本不高：入門時很容易想像模型的架構與樣貌（受損的影像進去、更乾淨的影像出來），進而接觸更進一步的內容，是進入深度學習研究的好入口。

重點整理

- 選擇復原器及解讀輸出前，先說明退化假設。
- 訓練配對與退化合成決定網路學習還原什麼。
- 讓解析度路徑、感受野與正規化配合復原任務。
- 條件化可協助單一模型處理多種退化；未知與混合退化須分開測試。
- 像素準確度、感知真實感與輸入一致性回答不同問題，應搭配報告。
- 盲復原可以估計、依退化條件化，或隱式處理退化；聯合估計可能偏向退化解 [74]。
- 使用符合任務的資料集，並把無參考分數視為可能失準的品質估計，而非細節已還原的證明。

練習

1. **退化的順序**。用 `degrade` 函式，在相同參數下比較 (a) 模糊 → 降採樣 → 雜訊 → JPEG 與 (b) 雜訊 → 模糊 → 降採樣 → JPEG。從視覺上，以及在平坦區塊量測雜訊標準差，說明為什麼順序會影響網路要學的東西。

   提示

   在 (b) 中，模糊與降採樣會過濾雜訊：標準差變小，而且在空間上變得相關（變成平滑的斑塊，而不是逐像素的顆粒）。在 (a) 中，雜訊在低解析度下仍然是白雜訊。只用 (a) 訓練的網路，測試時遇到相關雜訊會去除不足。這也是 Real-ESRGAN 隨機化並重複各個階段的原因之一。

2. **手算 pixel shuffle**。一個特徵張量的形狀為 $(Cs^2, H, W) = (4, 2, 2)$，$s = 2$、$C = 1$，第 k 個通道全部填入常數 k。用 SR 一節的公式寫出 pixel shuffle 的 4×4 輸出，再用 `torch.nn.PixelShuffle(2)` 驗證。

   提示

   輸出像素 $(2h + i, 2w + j)$ 取自通道 $2i + j$。所以每個 2×2 區塊都是 $\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$，$4 \times 4$ 的輸出就是把這個區塊重複四次。每個輸出區塊都從同一個低解析度位置上的四個通道各取一個值。

3. **殘差還是直接預測？**把 `TinyDnCNN.forward` 改成回傳 `self.body(y)`（直接預測乾淨影像），以相同的隨機種子與步數訓練兩個版本，比較 PSNR 與損失曲線。再用 $\sigma = 5/255$ 重做一次。殘差學習在什麼情況下幫助最大？

   提示

   在訓練步數很少時，殘差版本通常收斂更快、PSNR 更高，因為它在初始化時就已經大約輸出 y（PSNR 不差），只需要學一個小修正。低雜訊時直接預測的版本最吃力，因為它必須透過好幾層 ReLU 學出一個近乎恆等的映射。

4. **用數字算暗通道**。某個有霧像素 $I = (0.75, 0.78, 0.80)$，其區塊內所有通道 I^c/A^c 的最小值為 0.70，$A = (0.9, 0.9, 0.9)$。取 $\omega = 0.95$、$t_0 = 0.1$，計算這個像素的 $\tilde{t}$ 與還原後的 $\hat{J}$。

提示

$\tilde{t} = 1 - 0.95 \times 0.70 = 0.335$。接著 $\hat{J}^c = (I^c - 0.9)/0.335 + 0.9$，約為 $(0.452, 0.542, 0.601)$。輸出比輸入更暗、飽和度更高，這正是去掉霧的面紗後該有的樣子。

5. **感知 vs. 失真**。把小型去雜訊網路的輸出再加上少量新的高斯雜訊（$\sigma = 3/255$），計算加入前後的 PSNR 與 SSIM。雖然兩個指標都下降了，影像看起來卻可能**更**自然（比較不像塑膠）。把這個現象和感知–失真函數 $P(D)$ 連結起來。

提示

近似 MMSE 的輸出位於低失真的位置，但離自然影像分布很遠（太平滑）。加入顆粒會讓輸出分布更接近真實照片（d 變小），代價是 MSE 變高：這是沿著（或朝向）凸邊界的移動，而不是越過它。

6. **設計一個 all-in-one 實驗**。你要為雜訊（$\sigma \in [0, 50]$）、雨線與霧建立單一模型。說明訓練資料的混合方式、如何為網路提供條件（測試時沒有標籤）、要和哪些基準比較，以及如何偵測任務干擾。

提示

每個任務各自取資料集並平衡取樣；加入一個與網路一起訓練的退化編碼器（如 AirNet 的對比學習，或 PromptIR 的提示）。與三個使用相同骨幹、相同大小的單任務模型比較，也和不加條件的版本比較。任務干擾的跡象是：all-in-one 模型在某個任務上落後單任務模型，在其他任務上卻打平；另外也要在訓練時沒見過的混合退化（雨＋霧）上測試。

7. **動手估計模糊核**。在盲反卷積的程式中，(a) 拿掉 `shock(...)` 並設定 `keep=1.0`（使用全部梯度），比較模糊核誤差與最後的 PSNR；(b) 改從 3×3 的方框模糊核開始，而不是接近 delta 的模糊核；(c) 把 `h_true` 換成 31×31 視窗中一條長 25 像素的直線運動軌跡。用 Levin 等人 [74] 的討論與由粗到細的策略解釋每個結果。

提示

(a) 使用全部梯度時，平滑的潛在影像中大量的弱梯度會主導最小平方擬合，估計不再改善（我們的執行結果中，模糊核誤差不降反升，最後的 PSNR 也停留在模糊輸入的水準）。只挑選少數強而且預測良好的邊緣，模糊核步驟才有資訊量，這正是實務上對付天真聯合 MAP「沒有模糊」偏差的方法。(b) 這個迴圈是非凸的交替最佳化，起點很重要；不同的起點可能收斂到不同的局部極小值。(c) 在全解析度下，從 delta 起點很難找到大的模糊核：我們的執行結果中誤差下降得少很多，也只補回一小部分「真實模糊核」的 PSNR。先在縮小的影像上估計模糊核（那時它只有幾個像素長），再把結果放大，正是影像金字塔的用途。

8. **驗證一個 NR 指標**。五張復原影像的平均意見分數為 MOS $= (4.1, 3.2, 2.5, 1.8, 3.9)$，NIQE 分數為 $(3.4, 4.2, 5.1, 6.3, 3.0)$。(a) 手算 SRCC，並解釋它的正負號。(b) 如果改報告 $\log(\text{NIQE})$，SRCC 會改變嗎？PLCC 呢？(c) 把 MSCN 程式用在一張去雜訊後再過度銳化的影像上，解釋為什麼 NSS 指標可能給它比單純去雜訊的輸出更好的分數。

提示

(a) 由小到大排名時，影像 1–5 的 MOS 名次為 $(5,3,2,1,4)$，NIQE 名次為 $(2,3,4,5,1)$。名次差為 $(3,0,-2,-4,3)$，所以 $\sum d_i^2 = 38$，SRCC $= 1 - 6\cdot 38/(5\cdot 24) = -0.9$。符號為負，是因為 NIQE 越低代表品質越好；報告時取 $|\text{SRCC}| = 0.9$。只有影像 1 與影像 5 的順序對調。(b) SRCC 只依賴名次，任何像 log 這樣的單調轉換都不會改變它；PLCC 則會改變，這也是為什麼 PLCC 通常在擬合單調對應之後才計算。(c) 去雜訊會留下平滑的影像，其 MSCN 統計量類似輕微模糊（變異數低、直方圖尖）。銳化把高頻能量加回來，把統計量推回接近自然影像的樣子，不論加上的細節是不是真的。

參考文獻

1. Shyandram, “Low-Level Vision Task-Image Restoration 簡介,” Medium post, 2023; updated 2026. link
2. Z. Chen, Z. Wu, E. Zamfir, K. Zhang, Y. Zhang, R. Timofte et al., “NTIRE 2024 Challenge on Image Super-Resolution (×4): Methods and Results,” in *Proc. IEEE/CVF CVPR Workshops*, 2024. arXiv
3. Z. Chen, K. Liu, J. Gong, J. Wang, L. Sun et al., “NTIRE 2025 Challenge on Image Super-Resolution (×4): Methods and Results,” in *Proc. IEEE/CVF CVPR Workshops*, 2025. arXiv
4. X. Wang, L. Xie, C. Dong and Y. Shan, “Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data,” in *Proc. IEEE/CVF ICCV Workshops*, 2021. arXiv
5. E. H. Land and J. J. McCann, “Lightness and Retinex Theory,” *Journal of the Optical Society of America*, vol. 61, no. 1, pp. 1–11, 1971. doi
6. C. Dong, C. C. Loy, K. He and X. Tang, “Image Super-Resolution Using Deep Convolutional Networks,” *IEEE TPAMI*, 2016 (conference version ECCV 2014). arXiv
7. C. Dong, C. C. Loy and X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network,” in *Proc. ECCV*, 2016. arXiv
8. W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert and Z. Wang, “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network,” in *Proc. IEEE CVPR*, 2016. arXiv
9. A. Odena, V. Dumoulin and C. Olah, “Deconvolution and Checkerboard Artifacts,” *Distill*, 2016. link
10. B. Lim, S. Son, H. Kim, S. Nah and K. M. Lee, “Enhanced Deep Residual Networks for Single Image Super-Resolution,” in *Proc. IEEE CVPR Workshops*, 2017. arXiv
11. Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong and Y. Fu, “Image Super-Resolution Using Very Deep Residual Channel Attention Networks,” in *Proc. ECCV*, 2018. arXiv
12. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” in *Proc. IEEE CVPR*, 2017. arXiv
13. X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, C. C. Loy, Y. Qiao and X. Tang, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” in *Proc. ECCV Workshops*, 2018. arXiv
14. J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool and R. Timofte, “SwinIR: Image Restoration Using Swin Transformer,” in *Proc. IEEE/CVF ICCV Workshops*, 2021. arXiv
15. X. Chen, X. Wang, J. Zhou, Y. Qiao and C. Dong, “Activating More Pixels in Image Super-Resolution Transformer,” in *Proc. IEEE/CVF CVPR*, 2023. arXiv

16. X. Lin, J. He, Z. Chen, Z. Lyu, B. Dai, F. Yu, W. Ouyang, Y. Qiao and C. Dong, "DiffBIR: Towards Blind Image Restoration with Generative Diffusion Prior," arXiv:2308.15070, 2023. arXiv
17. F. Yu, J. Gu, Z. Li, J. Hu, X. Kong, X. Wang, J. He, Y. Qiao and C. Dong, "Scaling Up to Excellence: Practicing Model Scaling for Photo-Realistic Image Restoration In the Wild," in *Proc. IEEE/CVF CVPR*, 2024. arXiv
18. R. Wu, L. Sun, Z. Ma and L. Zhang, "One-Step Effective Diffusion Network for Real-World Image Super-Resolution," in *Proc. NeurIPS*, 2024. arXiv
19. E. Agustsson and R. Timofte, "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study," in *Proc. IEEE CVPR Workshops*, 2017. doi
20. K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017. arXiv
21. K. Zhang, W. Zuo and L. Zhang, "FFDNet: Toward a Fast and Flexible Solution for CNN-Based Image Denoising," *IEEE Transactions on Image Processing*, 2018. arXiv
22. A. Abdelhamed, S. Lin and M. S. Brown, "A High-Quality Denoising Dataset for Smartphone Cameras," in *Proc. IEEE/CVF CVPR*, 2018. doi
23. S. W. Zamir, A. Arora, S. Khan, M. Hayat, F. S. Khan and M.-H. Yang, "Restormer: Efficient Transformer for High-Resolution Image Restoration," in *Proc. IEEE/CVF CVPR*, 2022. arXiv
24. L. Chen, X. Chu, X. Zhang and J. Sun, "Simple Baselines for Image Restoration," in *Proc. ECCV*, 2022. arXiv
25. K. He, J. Sun and X. Tang, "Single Image Haze Removal Using Dark Channel Prior," in *Proc. IEEE CVPR*, 2009. doi
26. B. Cai, X. Xu, K. Jia, C. Qing and D. Tao, "DehazeNet: An End-to-End System for Single Image Haze Removal," *IEEE Transactions on Image Processing*, 2016. arXiv
27. B. Li, X. Peng, Z. Wang, J. Xu and D. Feng, "AOD-Net: All-in-One Dehazing Network," in *Proc. IEEE ICCV*, 2017. doi ·arXiv
28. X. Liu, Y. Ma, Z. Shi and J. Chen, "GridDehazeNet: Attention-Based Multi-Scale Network for Image Dehazing," in *Proc. IEEE/CVF ICCV*, 2019. arXiv
29. X. Qin, Z. Wang, Y. Bai, X. Xie and H. Jia, "FFA-Net: Feature Fusion Attention Network for Single Image Dehazing," in *Proc. AAAI*, 2020. arXiv
30. S. W. Zamir, A. Arora, S. Khan, M. Hayat, F. S. Khan, M.-H. Yang and L. Shao, "Learning Enriched Features for Real Image Restoration and Enhancement," in *Proc. ECCV*, 2020. arXiv
31. Y. Song, Z. He, H. Qian and X. Du, "Vision Transformers for Single Image Dehazing," *IEEE Transactions on Image Processing*, vol. 32, 2023. arXiv
32. B. Li, W. Ren, D. Fu, D. Tao, D. Feng, W. Zeng and Z. Wang, "Benchmarking Single-Image Dehazing and Beyond," *IEEE Transactions on Image Processing*, 2019. arXiv
33. C. O. Ancuti, C. Ancuti, R. Timofte and C. De Vleeschouwer, "O-HAZE: A Dehazing Benchmark with Real Hazy and Haze-Free Outdoor Images," in *Proc. IEEE/CVF CVPR Workshops*, 2018. doi
34. C. O. Ancuti, C. Ancuti, M. Sbert and R. Timofte, "Dense-Haze: A Benchmark for Image Dehazing with Dense-Haze and Haze-Free Images," in *Proc. IEEE ICIP*, 2019. doi
35. A. Geiger, P. Lenz and R. Urtasun, "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. IEEE CVPR*, 2012. project page
36. Y.-F. Liu, D.-W. Jaw, S.-C. Huang and J.-N. Hwang, "DesnowNet: Context-Aware Deep Net-

work for Snow Removal," *IEEE Transactions on Image Processing*, 2018. arXiv
37. W.-T. Chen, H.-Y. Fang, J.-J. Ding, C.-C. Tsai and S.-Y. Kuo, "JSTASR: Joint Size and Transparency-Aware Snow Removal Algorithm Based on Modified Partial Convolution and Veiling Effect Removal," in *Proc. ECCV*, 2020. doi
38. W. Yang, R. T. Tan, J. Feng, J. Liu, Z. Guo and S. Yan, "Deep Joint Rain Detection and Removal from a Single Image," in *Proc. IEEE CVPR*, 2017. arXiv
39. R. Qian, R. T. Tan, W. Yang, J. Su and J. Liu, "Attentive Generative Adversarial Network for Raindrop Removal from a Single Image," in *Proc. IEEE/CVF CVPR*, 2018. arXiv
40. X. Li, Y. Jin, X. Jin, Z. Wu, B. Li, Y. Wang et al., "NTIRE 2025 Challenge on Day and Night Raindrop Removal for Dual-Focused Images: Methods and Results," in *Proc. IEEE/CVF CVPR Workshops*, 2025. arXiv
41. C. Wei, W. Wang, W. Yang and J. Liu, "Deep Retinex Decomposition for Low-Light Enhancement," in *Proc. BMVC*, 2018. arXiv
42. Y. Cai, H. Bian, J. Lin, H. Wang, R. Timofte and Y. Zhang, "Retinexformer: One-stage Retinex-based Transformer for Low-light Image Enhancement," in *Proc. IEEE/CVF ICCV*, 2023. arXiv
43. H. Zhang, H. Xu, X. Tian, J. Jiang and J. Ma, "Image Fusion Meets Deep Learning: A Survey and Perspective," *Information Fusion*, vol. 76, pp. 323–336, 2021. doi
44. S. Nah, T. H. Kim and K. M. Lee, "Deep Multi-scale Convolutional Neural Network for Dynamic Scene Deblurring," in *Proc. IEEE CVPR*, 2017. doi ·arXiv
45. K. He, X. Chen, S. Xie, Y. Li, P. Dollár and R. Girshick, "Masked Autoencoders Are Scalable Vision Learners," in *Proc. IEEE/CVF CVPR*, 2022. arXiv
46. R. Suvorov, E. Logacheva, A. Mashikhin, A. Remizova, A. Ashukha, A. Silvestrov, N. Kong, H. Goka, K. Park and V. Lempitsky, "Resolution-robust Large Mask Inpainting with Fourier Convolutions," in *Proc. IEEE/CVF WACV*, 2022. arXiv
47. S. Sengupta, V. Jayaram, B. Curless, S. Seitz and I. Kemelmacher-Shlizerman, "Background Matting: The World is Your Green Screen," in *Proc. IEEE/CVF CVPR*, 2020. arXiv
48. O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Proc. MICCAI*, 2015. arXiv
49. H. Guo, J. Li, T. Dai, Z. Ouyang, X. Ren and S.-T. Xia, "MambaIR: A Simple Baseline for Image Restoration with State-Space Model," in *Proc. ECCV*, 2024. arXiv
50. H. Guo, Y. Guo, Y. Zha, Y. Zhang, W. Li, T. Dai, S.-T. Xia and Y. Li, "MambaIRv2: Attentive State Space Restoration," in *Proc. IEEE/CVF CVPR*, 2025. arXiv
51. B. Li, X. Liu, P. Hu, Z. Wu, J. Lv and X. Peng, "All-In-One Image Restoration for Unknown Corruption," in *Proc. IEEE/CVF CVPR*, 2022. code & paper
52. J. M. J. Valanarasu, R. Yasarla and V. M. Patel, "TransWeather: Transformer-based Restoration of Images Degraded by Adverse Weather Conditions," in *Proc. IEEE/CVF CVPR*, 2022. arXiv
53. V. Potlapalli, S. W. Zamir, S. Khan and F. S. Khan, "PromptIR: Prompting for All-in-One Image Restoration," in *Proc. NeurIPS*, 2023. arXiv
54. Y. Cui, S. W. Zamir, S. Khan, A. Knoll, M. Shah and F. S. Khan, "AdaIR: Adaptive All-in-One Image Restoration via Frequency Mining and Modulation," in *Proc. ICLR*, 2025. arXiv
55. Z. Luo, F. K. Gustafsson, Z. Zhao, J. Sjölund and T. B. Schön, "Controlling Vision-Language Models for Multi-Task Image Restoration," in *Proc. ICLR*, 2024. arXiv
56. M. V. Conde, G. Geigle and R. Timofte, "InstructIR: High-Quality Image Restoration Following Human Instructions," in *Proc. ECCV*, 2024. arXiv

57. H. Zhao, O. Gallo, I. Frosio and J. Kautz, "Loss Functions for Image Restoration with Neural Networks," *IEEE Transactions on Computational Imaging*, 2017 (arXiv title: "Loss Functions for Neural Networks for Image Processing"). arXiv
58. J. Johnson, A. Alahi and L. Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution," in *Proc. ECCV*, 2016. doi ·arXiv
59. L. Jiang, B. Dai, W. Wu and C. C. Loy, "Focal Frequency Loss for Image Reconstruction and Synthesis," in *Proc. IEEE/CVF ICCV*, 2021. arXiv
60. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
61. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," in *Proc. IEEE/CVF CVPR*, 2018. arXiv
62. A. Mittal, R. Soundararajan and A. C. Bovik, "Making a 'Completely Blind' Image Quality Analyzer," *IEEE Signal Processing Letters*, 2013. doi
63. Y. Blau, R. Mechrez, R. Timofte, T. Michaeli and L. Zelnik-Manor, "The 2018 PIRM Challenge on Perceptual Image Super-resolution," in *Proc. ECCV Workshops*, 2018. arXiv
64. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in *Proc. NeurIPS*, 2017. arXiv
65. Y. Blau and T. Michaeli, "The Perception-Distortion Tradeoff," in *Proc. IEEE/CVF CVPR*, 2018. arXiv
66. Z. Wang, J. Chen and S. C. H. Hoi, "Deep Learning for Image Super-Resolution: A Survey," *IEEE TPAMI*, 2021. arXiv
67. C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo and C.-W. Lin, "Deep Learning on Image Denoising: An Overview," *Neural Networks*, vol. 131, pp. 251–275, 2020. arXiv
68. M. Elad, B. Kawar and G. Vaksman, "Image Denoising: The Deep Learning Revolution and Beyond — A Survey Paper," *SIAM Journal on Imaging Sciences*, vol. 16, no. 3, pp. 1594–1654, 2023. doi ·arXiv
69. K. Zhang, W. Ren, W. Luo, W.-S. Lai, B. Stenger, M.-H. Yang and H. Li, "Deep Image Deblurring: A Survey," *International Journal of Computer Vision*, 2022. arXiv
70. J. Gui, X. Cong, Y. Cao, W. Ren, J. Zhang, J. Zhang, J. Cao and D. Tao, "A Comprehensive Survey and Taxonomy on Single Image Dehazing Based on Deep Learning," *ACM Computing Surveys*, 2023. doi ·arXiv
71. W. Yang, R. T. Tan, S. Wang, Y. Fang and J. Liu, "Single Image Deraining: From Model-Based to Data-Driven and Beyond," *IEEE TPAMI*, 2021. doi ·arXiv
72. J. Jiang, Z. Zuo, G. Wu, K. Jiang and X. Liu, "A Survey on All-in-One Image Restoration: Taxonomy, Evaluation and Future Trends," *IEEE TPAMI*, 2025. arXiv
73. X. Li, Y. Ren, X. Jin, C. Lan, X. Wang, W. Zeng, X. Wang and Z. Chen, "Diffusion Models for Image Restoration and Enhancement: A Comprehensive Survey," *International Journal of Computer Vision*, 2025. arXiv
74. A. Levin, Y. Weiss, F. Durand and W. T. Freeman, "Understanding and Evaluating Blind Deconvolution Algorithms," in *Proc. IEEE CVPR*, 2009, pp. 1964–1971. doi
75. R. Fergus, B. Singh, A. Hertzmann, S. T. Roweis and W. T. Freeman, "Removing Camera Shake from a Single Photograph," *ACM Transactions on Graphics* (Proc. SIGGRAPH), vol. 25, no. 3, pp. 787–794, 2006. doi

76. Q. Shan, J. Jia and A. Agarwala, "High-quality Motion Deblurring from a Single Image," *ACM Transactions on Graphics* (Proc. SIGGRAPH), vol. 27, no. 3, 2008. doi
77. S. Cho and S. Lee, "Fast Motion Deblurring," *ACM Transactions on Graphics* (Proc. SIGGRAPH Asia), vol. 28, no. 5, 2009. doi
78. L. Xu and J. Jia, "Two-Phase Kernel Estimation for Robust Motion Deblurring," in *Proc. ECCV*, 2010, pp. 157–170. doi
79. D. Krishnan, T. Tay and R. Fergus, "Blind Deconvolution Using a Normalized Sparsity Measure," in *Proc. IEEE CVPR*, 2011, pp. 233–240. doi
80. J. Pan, D. Sun, H. Pfister and M.-H. Yang, "Blind Image Deblurring Using Dark Channel Prior," in *Proc. IEEE CVPR*, 2016, pp. 1628–1636. doi
81. J. Sun, W. Cao, Z. Xu and J. Ponce, "Learning a Convolutional Neural Network for Non-uniform Motion Blur Removal," in *Proc. IEEE CVPR*, 2015. arXiv
82. C. J. Schuler, M. Hirsch, S. Harmeling and B. Schölkopf, "Learning to Deblur," *IEEE TPAMI*, vol. 38, no. 7, pp. 1439–1451, 2016. doi ·arXiv
83. A. Chakrabarti, "A Neural Approach to Blind Motion Deblurring," in *Proc. ECCV*, 2016. arXiv
84. D. Ulyanov, A. Vedaldi and V. Lempitsky, "Deep Image Prior," in *Proc. IEEE/CVF CVPR*, 2018, pp. 9446–9454. doi ·arXiv
85. D. Ren, K. Zhang, Q. Wang, Q. Hu and W. Zuo, "Neural Blind Deconvolution Using Deep Priors," in *Proc. IEEE/CVF CVPR*, 2020. arXiv
86. O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin and J. Matas, "DeblurGAN: Blind Motion Deblurring Using Conditional Adversarial Networks," in *Proc. IEEE/CVF CVPR*, 2018. arXiv
87. K. Zhang, W. Zuo and L. Zhang, "Learning a Single Convolutional Super-Resolution Network for Multiple Degradations," in *Proc. IEEE/CVF CVPR*, 2018. arXiv
88. J. Gu, H. Lu, W. Zuo and C. Dong, "Blind Super-Resolution With Iterative Kernel Correction," in *Proc. IEEE/CVF CVPR*, 2019. arXiv
89. S. Bell-Kligler, A. Shocher and M. Irani, "Blind Super-Resolution Kernel Estimation using an Internal-GAN," in *Proc. NeurIPS*, 2019. link ·arXiv
90. Z. Luo, Y. Huang, S. Li, L. Wang and T. Tan, "Unfolding the Alternating Optimization for Blind Super Resolution," in *Proc. NeurIPS*, 2020. arXiv
91. K. Zhang, L. Van Gool and R. Timofte, "Deep Unfolding Network for Image Super-Resolution," in *Proc. IEEE/CVF CVPR*, 2020. arXiv
92. K. Zhang, J. Liang, L. Van Gool and R. Timofte, "Designing a Practical Degradation Model for Deep Blind Image Super-Resolution," in *Proc. IEEE/CVF ICCV*, 2021. arXiv
93. A. Liu, Y. Liu, J. Gu, Y. Qiao and C. Dong, "Blind Image Super-Resolution: A Survey and Beyond," *IEEE TPAMI*, 2022 (early access). doi ·arXiv
94. A. Mittal, A. K. Moorthy and A. C. Bovik, "No-Reference Image Quality Assessment in the Spatial Domain," *IEEE Transactions on Image Processing*, vol. 21, no. 12, pp. 4695–4708, 2012. doi
95. C. Ma, C.-Y. Yang, X. Yang and M.-H. Yang, "Learning a No-Reference Quality Metric for Single-Image Super-Resolution," *Computer Vision and Image Understanding*, vol. 158, pp. 1–16, 2017. doi ·arXiv
96. N. Venkatanath, D. Praneeth, M. C. Bh, S. S. Channappayya and S. S. Medasani, "Blind Image Quality Evaluation Using Perception Based Features," in *Proc. Twenty First National Conference on Communications* (NCC), 2015, pp. 1–6. doi ·MATLAB documentation

97. H. Talebi and P. Milanfar, “NIMA: Neural Image Assessment,” *IEEE Transactions on Image Processing*, vol. 27, no. 8, pp. 3998–4011, 2018. doi ·arXiv
98. S. Su, Q. Yan, Y. Zhu, C. Zhang, X. Ge, J. Sun and Y. Zhang, “Blindly Assess Image Quality in the Wild Guided by a Self-Adaptive Hyper Network,” in *Proc. IEEE/CVF CVPR*, 2020, pp. 3664–3673. doi
99. J. Ke, Q. Wang, Y. Wang, P. Milanfar and F. Yang, “MUSIQ: Multi-scale Image Quality Transformer,” in *Proc. IEEE/CVF ICCV*, 2021, pp. 5128–5137. doi ·arXiv
100. S. Yang, T. Wu, S. Shi, S. Lao, Y. Gong, M. Cao, J. Wang and Y. Yang, “MANIQA: Multi-dimension Attention Network for No-Reference Image Quality Assessment,” in *Proc. IEEE/CVF CVPR Workshops*, 2022, pp. 1190–1199. doi ·arXiv
101. C. Chen, J. Mo, J. Hou, H. Wu, L. Liao, W. Sun, Q. Yan and W. Lin, “TOPIQ: A Top-Down Approach From Semantics to Distortions for Image Quality Assessment,” *IEEE Transactions on Image Processing*, vol. 33, pp. 2404–2418, 2024. doi ·arXiv
102. J. Wang, K. C. K. Chan and C. C. Loy, “Exploring CLIP for Assessing the Look and Feel of Images,” in *Proc. AAAI Conference on Artificial Intelligence*, 2023. arXiv
103. H. Wu, Z. Zhang, W. Zhang, C. Chen, L. Liao, C. Li, Y. Gao, A. Wang, E. Zhang, W. Sun, Q. Yan, X. Min, G. Zhai and W. Lin, “Q-Align: Teaching LMMs for Visual Scoring via Discrete Text-Defined Levels,” in *Proc. ICML*, PMLR 235, pp. 54015–54029, 2024. link ·arXiv
104. H. R. Sheikh, M. F. Sabir and A. C. Bovik, “A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms,” *IEEE Transactions on Image Processing*, vol. 15, no. 11, pp. 3440–3451, 2006 (LIVE Image Quality Assessment Database). doi ·database
105. N. Ponomarenko, L. Jin, O. Ieremeiev, V. Lukin, K. Egiazarian, J. Astola, B. Vozel, K. Chehdi, M. Carli, F. Battisti and C.-C. J. Kuo, “Image Database TID2013: Peculiarities, Results and Perspectives,” *Signal Processing: Image Communication*, vol. 30, pp. 57–77, 2015. doi ·database
106. H. Lin, V. Hosu and D. Saupe, “KADID-10k: A Large-scale Artificially Distorted IQA Database,” in *Proc. Eleventh International Conference on Quality of Multimedia Experience (QoMEX)*, 2019. doi ·database
107. V. Hosu, H. Lin, T. Sziranyi and D. Saupe, “KonIQ-10k: An Ecologically Valid Database for Deep Learning of Blind Image Quality Assessment,” *IEEE Transactions on Image Processing*, vol. 29, pp. 4041–4056, 2020. arXiv
108. Y. Fang, H. Zhu, Y. Zeng, K. Ma and Z. Wang, “Perceptual Quality Assessment of Smartphone Photography,” in *Proc. IEEE/CVF CVPR*, 2020, pp. 3674–3683. doi
109. J. Gu, H. Cai, H. Chen, X. Ye, J. Ren and C. Dong, “PIPAL: A Large-Scale Image Quality Assessment Dataset for Perceptual Image Restoration,” in *Proc. ECCV*, 2020. arXiv
110. C. Chen and J. Mo, “IQA-PyTorch: PyTorch Toolbox for Image Quality Assessment,” software (pip package `pyiqa`), 2022. GitHub
111. ITU-R, “Recommendation BT.500-15: Methodologies for the Subjective Assessment of the Quality of Television Images,” 2023. ITU
112. R. A. Bradley and M. E. Terry, “Rank Analysis of Incomplete Block Designs: The Method of Paired Comparisons,” *Biometrika*, vol. 39, no. 3–4, pp. 324–345, 1952. doi
113. D. Geman and C. Yang, “Nonlinear Image Recovery with Half-Quadratic Regularization,” *IEEE Transactions on Image Processing*, vol. 4, no. 7, pp. 932–946, 1995. doi
114. S. V. Venkatakrishnan, C. A. Bouman and B. Wohlberg, “Plug-and-Play Priors for Model Based Reconstruction,” in *Proc. IEEE Global Conference on Signal and Information Processing (GlobalSIP)*,

2013, pp. 945–948. doi
115. K. Zhang, W. Zuo, S. Gu and L. Zhang, “Learning Deep CNN Denoiser Prior for Image Restoration,” in *Proc. IEEE/CVF CVPR*, 2017, pp. 2808–2817. doi ·arXiv
116. A. Chambolle, “An Algorithm for Total Variation Minimization and Applications,” *Journal of Mathematical Imaging and Vision*, vol. 20, no. 1–2, pp. 89–97, 2004. doi
117. A. Buades, B. Coll and J.-M. Morel, “A Non-Local Algorithm for Image Denoising,” in *Proc. IEEE CVPR*, vol. 2, 2005, pp. 60–65. doi
118. D. Ulyanov, A. Vedaldi and V. Lempitsky, “Deep Image Prior,” *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1867–1888, 2020. doi ·arXiv
119. Y. Jiang, Z. Zhang, T. Xue and J. Gu, “AutoDIR: Automatic All-in-One Image Restoration with Latent Diffusion,” in *Proc. ECCV, 2024*. paper
120. K. Zhu, J. Gu, Z. You, Y. Qiao and C. Dong, “An Intelligent Agentic System for Complex Image Restoration Problems,” in *Proc. ICLR, 2025*. paper
121. Y. Zuo, Q. Zheng, M. Wu, X. Jiang, R. Li, J. Wang, Y. Zhang, G. Mai, L. V. Wang, J. Zou, X. Wang, M.-H. Yang and Z. Tu, “4KAgent: Agentic Any Image to 4K Super-Resolution,” in *Proc. NeurIPS, 2025*. paper
122. Y. Yu, Z. Zeng, H. Hua, J. Fu and J. Luo, “PromptFix: You Prompt and We Fix the Photo,” in *Proc. NeurIPS, 2024*. paper

DLCV 09A・非盲影像超解析度

從不適定問題、插值與線性基準，逐步理解非盲超解析度的六個架構決策、感知與擴散目標，以及保真與評估規則。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/

你將學到

- 說明即使退化已知，超解析度仍有多解。
- 區分 MSE 的條件平均與 L1 的逐座標中位數。
- 比較上採樣、殘差學習、正規化與注意力選擇。
- 解讀感知與生成目標的忠實度—真實感取捨。
- 註明色彩、邊界及退化協定進行評估，並檢查偽影。

先看全貌

超解析度要把小圖放大，也讓它更清楚。伸展像素容易，補足從未記錄的細節才難；模型從自然影像知識補上內容，同時帶來能力與風險。

> **用最簡單的話說**
>
> 朋友用電話描述一張照片：短棕髮、微笑、淺色牆前的人。你能畫出好看的圖，卻不知道她真正的雀斑位置。許多畫都符合描述；謹慎畫家在不確定處畫柔和輪廓，大膽畫家補上清晰雀斑。前者模糊，後者自然卻可能錯。SR 模型也有這種取捨，本章介紹它們怎麼設計與檢查。

手機數位變焦、遊戲與影片放大、衛星與顯微影像都會遇到 SR，許多殘差學習與生成先驗也先在此接受檢驗。

範圍。本章談退化已知的**非盲單圖 SR**，主要是固定 bicubic，適合先學架構。未知退化見 DLCV 09A.1；影片見 DLCV 09A.3；一般生成先驗見 DLCV 09B；損失與指標見 DLCV 09。先備是 IP 02 插值 和 IP 04 混疊。

貫穿範例。以 `skimage.data` 的 512 × 512 太空人，經已知模糊及區塊平均縮小四倍再放大。程式用 NumPy、SciPy、scikit-image、scikit-learn、OpenCV，可在 CPU 快速執行。

問題：一個輸入，許多答案

縮圖會失去資訊，多個 HR 可產生同一 LR；平方誤差以它們的平均為目標，可能模糊位置不一致的邊緣和紋理。

問題設定與符號

單圖超解析度（SISR）從 $y \in \mathbb{R}^{H\times W\times C}$ 估計 $x \in \mathbb{R}^{sH\times sW\times C}$，$s$ 是倍率，H, W 是 LR 尺寸，C 是色彩通道。DLCV 09 的前向模型為：

$$y = (x * k)\downarrow_s + n,$$

k 是光學、像素積分或抗混疊核，$*$ 是逐通道二維卷積，$\downarrow_s$ 每 $s \times s$ 留一個樣本，n 為雜訊。非盲推論已知核、倍率及雜訊，條件模型可接受變動值；本章多用訓練／測試相符的固定退化，通常每倍率另訓練。多張位移 LR 的舊式多幀問題 [1] 見影片 SR。

為什麼這是病態問題

HR 未知數是量測的 s^2 倍。$s = 4$ 時，整體維度每個量測對應 16 個未知，即使無雜訊且算子滿列秩，仍至少有 15 個自由維度。模糊會跨區塊混合，這不是每塊有 15 個獨立不可見像素。$\mathcal{D}(x) = (x * k)\downarrow_s$ 有大**零空間**：若 $\mathcal{D}(z) = 0$，加上 z 不改變觀測。細紋、精確邊緣和小字常落在這類未約束資訊。

所以 SR 需要一個**先驗**（prior）$p(x)$，也就是「哪些 HR 影像是合理的」的模型。依貝氏定理，每個 HR 候選都有一個後驗機率

$$p(x \mid y) \propto p(y \mid x)\, p(x),$$

其中 $p(y \mid x)$ 是概似（likelihood，在給定雜訊下 $\mathcal{D}(x)$ 能多好地解釋 y），$p(x)$ 是先驗。傳統方法手寫先驗；深度 SR 則從（LR, HR）配對中學習先驗。

後驗平均造成的模糊

大多數 SR 網路都是在配對上最小化像素損失來訓練的。用平方誤差時，最好的網路輸出的是**後驗平均**（posterior mean）

$$\hat{x}_{\mathrm{MMSE}}(y) = \mathbb{E}[x \mid y] = \int x\, p(x \mid y)\, dx,$$

這是最小均方誤差（MMSE）估計；L1 則以逐座標後驗中位數為最優，未必等於平均。若多個清楚候選的邊位置／紋理相位不同，平均會變平滑，甚至不是合理樣本。更多資料不能消除目標本身的平均效果。

一個一維的小例子可以把這件事講清楚。一個 64 個取樣的 HR 訊號是一個位置未知的階梯邊緣。我們把每 4 個取樣平均成一個，加上標準差 0.2 的雜訊，然後精確算出 32 個可能邊緣位置上的後驗、它的平均，以及從中隨機抽出的一個樣本。

```
import numpy as np

rng = np.random.default_rng(0)
n, s, sigma = 64, 4, 0.2                # HR length, scale factor, LR noise level
cand = np.arange(16, 48)                # possible edge positions t (the prior: uniform)
X = (np.arange(n)[None, :] >= cand[:, None]).astype(float)    # every HR step edge x_t
DX = X.reshape(len(cand), n // s, s).mean(axis=2)              # their LR versions D(x_t)

err_mmse, err_samp, jump_mmse, jump_samp = [], [], [], []
for _ in range(2000):
    t = rng.integers(len(cand))
    y = DX[t] + sigma * rng.standard_normal(n // s)              # observed LR signal
```

```
    logp = -((y - DX) ** 2).sum(axis=1) / (2 * sigma**2)          # log posterior over t
    post = np.exp(logp - logp.max()); post /= post.sum()
    x_mmse = post @ X                                             # posterior mean E[x | y]
    x_samp = X[rng.choice(len(cand), p=post)]                     # one posterior sample
    err_mmse.append(np.mean((x_mmse - X[t]) ** 2)); err_samp.append(np.mean((x_samp - X[t])
     ↪  ** 2))
    jump_mmse.append(np.diff(x_mmse).max()); jump_samp.append(np.diff(x_samp).max())

print(f"MSE   posterior mean {np.mean(err_mmse):.4f} | posterior sample
 ↪  {np.mean(err_samp):.4f} "
      f"| ratio {np.mean(err_samp) / np.mean(err_mmse):.2f}")
print(f"largest one-pixel jump (a real edge has 1.00): mean {np.mean(jump_mmse):.2f} | sample
 ↪  {np.mean(jump_samp):.2f}")
```

```
MSE   posterior mean 0.0072 | posterior sample 0.0143 | ratio 1.97
largest one-pixel jump (a real edge has 1.00): mean 0.48 | sample 1.00
```

後驗平均的誤差只有後驗樣本的一半，但它的邊緣是斜坡：平均而言，它最大的單像素跳躍只有 0.48，而每個真實邊緣（以及每個樣本）都跳 1.00。樣本看起來和真實邊緣一模一樣，只是有時位置不對。比值約為 2 並非巧合。Blau 與 Michaeli 證明，如果一個估計器的輸出分布與真實影像完全相同（例如後驗取樣器），它的 MSE 永遠不需要超過 MMSE 估計器的兩倍，也就是 PSNR 差 3 dB [2]。圖 1 顯示一個例子，以及兩個估計器之間的整個範圍。

僅線上版：為什麼後驗樣本的誤差是 *MMSE* 的兩倍 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/notes/為什麼後驗樣本的誤差是-mmse-的兩倍/*

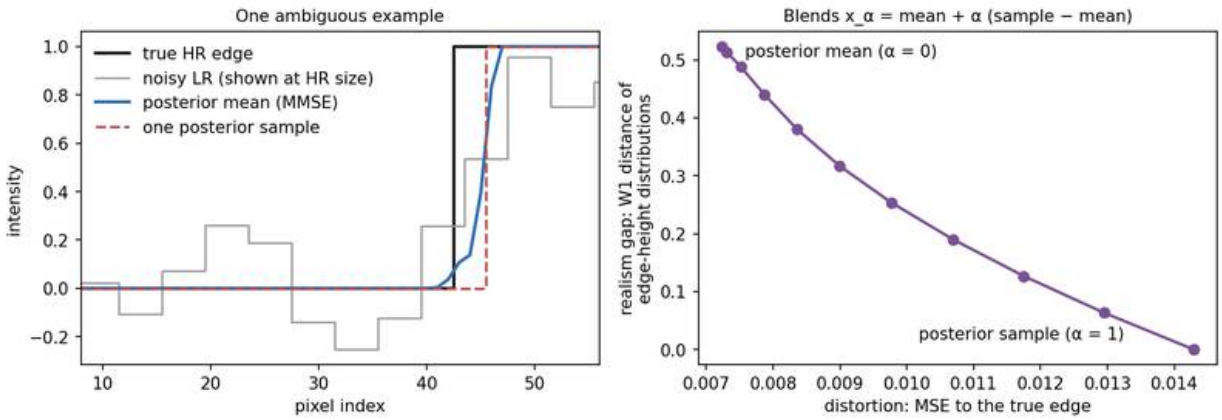


圖 *1* — 一維階梯邊緣小例子中的後驗平均模糊（×*4*，雜訊 $\sigma = 0.2$）。左：一個有歧義的例子。右：把後驗平均與後驗樣本混合，就是在失真（*MSE*）與真實感（輸出中邊緣高度的分布與真實邊緣之間的距離）之間取捨。沒有任何一種混合在兩個軸上都最好。

這個小例子串起全章：MSE 取平均，L1 取中位數，兩者給中心預測；感知／ GAN 偏好合理細節，隨機擴散能抽不同候選。評估須分開保真、自然外觀及不確定性。

從頻域看 LR 失去的資訊

白話說。下採樣可能造成混疊並削弱細節；SR 須恢復受損結構，也須推估輸入已無法決定的資訊。

LR Nyquist 上限為每 HR 像素 $1/(2s)$ 週期，$s = 4$ 時 **0.125**。更高頻可能折回低頻成混疊，或被前模糊移除，見 IP 04。上限以上無法由量測唯一確定，下方被模糊削弱的頻帶則須復原。

僅線上版：*LR 頻譜的混疊 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/notes/lr-頻譜的混疊/*

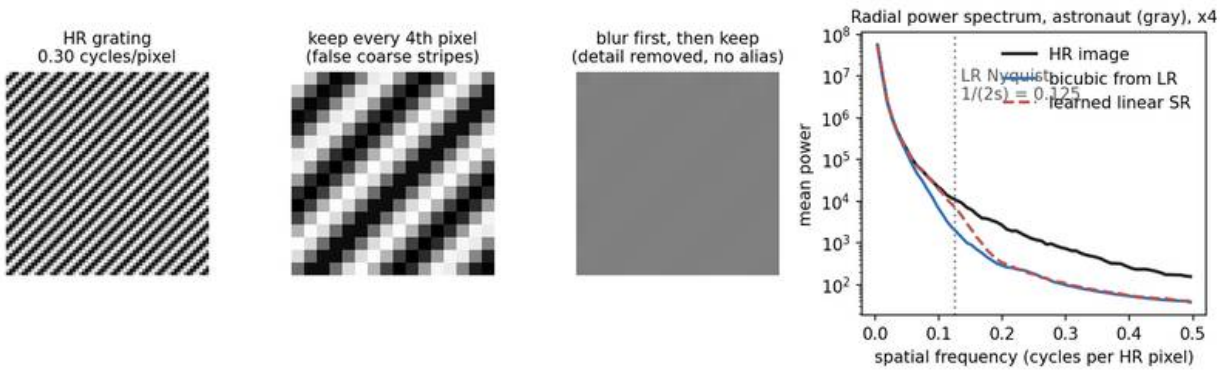


圖 *2 — 0.30* 週期條紋直接四倍抽取產生假粗紋，軸分量約 *0.21* 折為每 *LR* 像素約 *−0.15*。前濾波改為消除它。右圖插值不能決定缺失真實頻譜，非理想核也可能產生高頻偽影；線性模型主要改善上限以下頻帶。

基準設定：已知退化

用固定已知算子產生 LR—HR 配對，再對原圖計算重建誤差；常見規則是亮度通道 PSNR。

多數基準用 MATLAB `imresize` 預設 bicubic，縮小時預設抗混疊 [3]、[4]；沒有額外雜訊、相機模糊或壓縮。可重現使多年結果可比，但模型只學反解這個算子。

本章用另一個已知配方：HR 高斯寬 **1.2**、4 × 4 平均、Gaussian **0.01**、八位元量化。比較四種插值，按 Y 通道、裁 s 邊框計 PSNR。

```
import numpy as np, cv2
from scipy import ndimage as ndi
from skimage import data, img_as_float64

def degrade(hr, s=4, blur=1.2, noise=0.01, rng=None):
    """hr: (H, W, 3) in [0, 1], H and W divisible by s. Returns the LR image y."""
    rng = rng or np.random.default_rng(0)
    x = ndi.gaussian_filter(hr, sigma=(blur, blur, 0))          # 1) lens blur
    H, W, C = x.shape
    x = x.reshape(H // s, s, W // s, s, C).mean(axis=(1, 3))      # 2) s x s pixel
     ↪  integration + decimation
    x = x + noise * rng.standard_normal(x.shape)                  # 3) sensor noise
    return np.round(np.clip(x, 0, 1) * 255) / 255                 # 4) 8-bit quantization
```

```
def y_channel(rgb):
    """BT.601 luma on the [16, 235] scale (the MATLAB rgb2ycbcr convention)."""
    return 16 + rgb @ np.array([65.481, 128.553, 24.966])

def psnr(a, b, shave, peak):
    if shave:
        a, b = a[shave:-shave, shave:-shave], b[shave:-shave, shave:-shave]
    return 10 * np.log10(peak**2 / np.mean((a - b) ** 2))

hr = img_as_float64(data.astronaut())                              # (512, 512, 3)
lr = degrade(hr)                                                   # (128, 128, 3)
print("LR shape:", lr.shape)
for name, flag in [("nearest", cv2.INTER_NEAREST), ("bilinear", cv2.INTER_LINEAR),
                   ("bicubic", cv2.INTER_CUBIC), ("lanczos", cv2.INTER_LANCZOS4)]:
    sr = cv2.resize(lr, (512, 512), interpolation=flag)
    sr = np.clip(sr, 0, 1)
    y_hr, y_sr = y_channel(hr), y_channel(sr)
    print(f"{name:8s}  Y, shave 4: {psnr(y_hr, y_sr, 4, 255):.2f} dB | "
          f"Y, no shave: {psnr(y_hr, y_sr, 0, 255):.2f} dB | RGB, shave 4: {psnr(hr, sr, 4,
          ↪  1):.2f} dB")
```

```
LR shape: (128, 128, 3)
nearest   Y, shave 4: 24.62 dB | Y, no shave: 24.66 dB | RGB, shave 4: 23.06 dB
bilinear  Y, shave 4: 25.35 dB | Y, no shave: 25.40 dB | RGB, shave 4: 23.83 dB
bicubic   Y, shave 4: 26.36 dB | Y, no shave: 26.40 dB | RGB, shave 4: 24.81 dB
lanczos   Y, shave 4: 26.50 dB | Y, no shave: 26.54 dB | RGB, shave 4: 24.94 dB
```

此例各規則的排名相同，Y 與 RGB 卻差約 **1.5 dB**，因此只比相同計算規則。

圖 3 再加入 JPEG。非盲是假設整條鏈已知；未知或逐圖變化則轉入盲／真實 SR。



圖 *3 — ×4* 的模糊 → 降取樣 → 雜訊 →*JPEG*。非盲預先知道管線，*DLCV 09A.1* 處理未知情況。*LR* 只為展示而用最近鄰放大。

傳統基準方法

插值以固定濾波器分配既有樣本；學得線性濾波可補部分細節，但仍難描述自然影像的多樣性。

插值：固定線性濾波

內插把 LR 取樣放到 HR 網格上，再用鄰近取樣的加權和填補空隙。在一維中，

$$\hat{x}(t) = \sum_{u} y[u]\, \phi\!\left(\frac{t + \tfrac{1}{2}}{s} - \tfrac{1}{2} - u \right),$$

t 是 HR 索引，u 是 LR 索引，ϕ 為插值核，半像素項對齊 OpenCV 的像素中心慣例；二維沿列與行分離套用。IP 02 有常見核推導。

- **最近鄰／線性：**寬 1 方盒與寬 2 三角，分別偏塊狀和柔軟。
- **三次卷積：** Keys [5] 的寬 4 分段三次式，$a = -\tfrac{1}{2}$ 有該家族的最佳近似階；小負瓣略銳化。OpenCV `INTER_CUBIC` 用 $a = -0.75$。
- **Lanczos：** $\phi(t) = \operatorname{sinc}(t)\operatorname{sinc}(t/a)$，$|t| < a$，其餘零；$\operatorname{sinc}(t) = \sin(\pi t)/(\pi t)$，$a$ 為單側葉數 [6]。`INTER_LANCZOS4` 用 4。

較寬的核更接近理想 sinc，Lanczos 此例 **26.50 dB** 略高於 bicubic **26.36**。但固定插值對邊緣和平坦區使用同一線性規則，不能從 LR 唯一確定缺失真實頻率；有限核也會產生頻譜複本和振鈴，有高頻不代表恢復原細節。

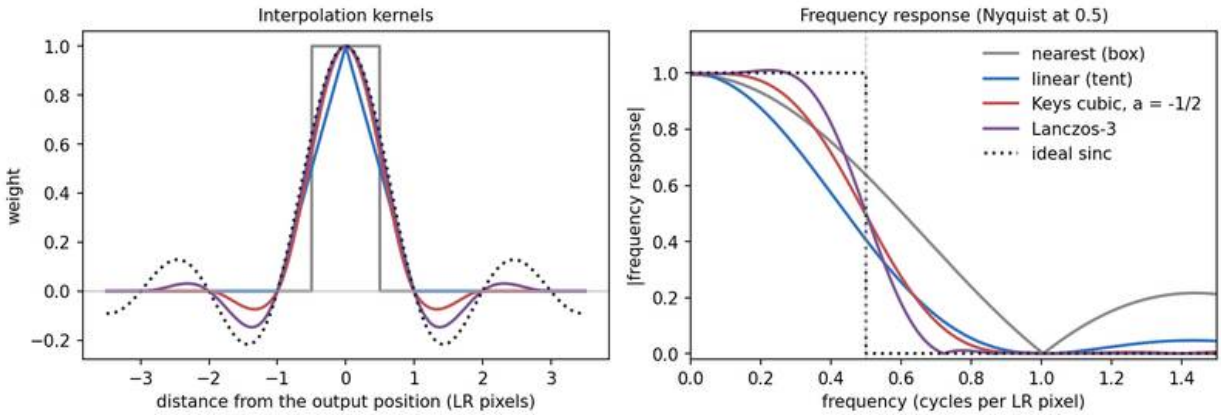


圖 *4* — 內插核（左）與其頻率響應的幅值（右）。好的內插器讓 *LR* 奈奎斯特極限（每個 *LR* 像素 *0.5* 週期）以下的頻率通過，並擋下極限以上的頻譜複本。



圖 *5* — 已知模糊 *1.2*、雜訊 *0.01* 的 ×*4* 插值。它重新分配現有資訊，不能唯一還原模糊和抽取移除的細節。

從配對區塊學線性濾波

白話說。從配對範例學習區塊映射，再用預先計算的線性修正有效率地處理相似區塊。

範例法從外部資料 [7] 或單圖跨尺度重複 [8] 找配對，稀疏編碼用雙字典 [9]；ANR ／ A+ 預算 ridge 迴歸 [10]、[11]，RAISR 依梯度方向、強度及一致性分桶 [12]。共同形式為：

$$\hat{r} = W_{c(p)}^{\top}\tilde{p}, \qquad W_c = \arg\min_{W} \sum_{i \in c} \|W^{\top}\tilde{p}_i - r_i\|_2^2 + \lambda \|W\|_F^2 = (P_c^{\top}P_c + \lambda I)^{-1} P_c^{\top} R_c,$$

其中 p 是 LR 區塊，$\tilde{p}$ 是它的特徵向量，$c(p)$ 是它所屬的類別（原子或雜湊桶），r 是要預測的 HR 細節，$(\tilde{p}_i, r_i)$ 是類別 c 的訓練配對，分別堆疊成 P_c 與 R_c 的列，λ 是嶺迴歸的權重，$\|\cdot\|_F$ 是 Frobenius 範數。

僅線上版：以範例為基礎的 *SR* 與稀疏編碼細節 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/notes/以範例為基礎的-sr-與稀疏編碼細節/*

此例灰階每個 LR 點取 7×7 鄰域、去平均加偏置，預測 4×4 HR **殘差**。每點輸出 s^2 再重排為區塊，即單層線性子像素卷積。用另 **11** 張 `skimage` 圖，分別以只有平均的銳利 LR、及先模糊 **1.2** 再平均的 LR 訓練，測太空人。

```
import numpy as np, cv2
from numpy.lib.stride_tricks import sliding_window_view
from scipy import ndimage as ndi
from skimage import data, img_as_float64
from skimage.color import rgb2gray

s, p = 4, 7                                   # scale factor, LR patch size

def degrade(x, blur):                         # blur (optional) -> s x s average
    x = ndi.gaussian_filter(x, blur) if blur else x
    H, W = x.shape[0] // s * s, x.shape[1] // s * s
    return x[:H, :W].reshape(H // s, s, W // s, s).mean(axis=(1, 3))

def bicubic(lr):
    return cv2.resize(lr, None, fx=s, fy=s, interpolation=cv2.INTER_CUBIC)

def features(lr):                             # one p x p patch per LR pixel, mean removed,
 ↪  plus a bias
    f = sliding_window_view(np.pad(lr, p // 2, mode="reflect"), (p, p)).reshape(lr.size, p *
     ↪  p)
    return np.c_[f - f.mean(axis=1, keepdims=True), np.ones(lr.size)]

def to_blocks(r):                             # HR residual -> one s*s vector per LR pixel
 ↪  (space-to-depth)
    h, w = r.shape[0] // s, r.shape[1] // s
    return r.reshape(h, s, w, s).transpose(0, 2, 1, 3).reshape(h * w, s * s)

def from_blocks(b, h, w):                     # inverse: depth-to-space (pixel shuffle)
    return b.reshape(h, w, s, s).transpose(0, 2, 1, 3).reshape(h * s, w * s)

def train(images, blur, lam=1e-3):
    A, B = [], []
```

```
    for im in images:
        im = im[: im.shape[0] // s * s, : im.shape[1] // s * s]
        lr = degrade(im, blur)
        A.append(features(lr)); B.append(to_blocks(im - bicubic(lr)))
    A, B = np.concatenate(A), np.concatenate(B)
    return np.linalg.solve(A.T @ A + lam * np.eye(A.shape[1]), A.T @ B)   # ridge: (p*p+1,
     ↪  s*s)

def super_resolve(lr, Wt):
    return bicubic(lr) + from_blocks(features(lr) @ Wt, *lr.shape)

def psnr(a, b):
    a, b = a[s:-s, s:-s], b[s:-s, s:-s]          # shave s border pixels
    return 10 * np.log10(1.0 / np.mean((a - np.clip(b, 0, 1)) ** 2))

train_imgs = [img_as_float64(im) for im in (data.camera(), data.moon(), data.coins(),
 ↪  data.page(),
              data.text(), data.brick(), data.grass(), data.gravel(),
              rgb2gray(data.coffee()), rgb2gray(data.chelsea()), rgb2gray(data.rocket()))]
models = {"trained sharp": train(train_imgs, 0.0), "trained blurry": train(train_imgs, 1.2)}
test = img_as_float64(rgb2gray(data.astronaut()))           # never seen in training
print("weights per model:", models["trained sharp"].shape)
for name, blur in [("sharp", 0.0), ("blurry", 1.2)]:
    lr = degrade(test, blur)
    row = "  ".join(f"{m} {psnr(test, super_resolve(lr, W)):.2f}" for m, W in models.items())
    print(f"test {name:6s} | bicubic {psnr(test, bicubic(lr)):.2f} | {row}  (dB)")
```

```
weights per model: (50, 16)
test sharp  | bicubic 25.61 | trained sharp 26.05  trained blurry 25.43  (dB)
test blurry | bicubic 24.98 | trained sharp 25.66  trained blurry 26.26  (dB)
```

每模型只有 $50 \times 16 = 800$ 權重，約一秒封閉求解。退化相符時比 bicubic 增 **0.44** ／ **1.28 dB**。模糊訓練模型遇銳利輸入會過銳化，**25.43** 低於 bicubic **25.61**；銳利訓練遇模糊修不足，**25.66** 低於匹配模型 **26.26**。正確退化假設很重要，Efrat 等人 [13] 也有相關經典結果。圖 6 的 16 濾波對應 16 子像素偏移。



雙三次



線性 *SR*（退化相符）

滑桿比較太空人臉部同一個裁切的模糊雙三次 ×4 上取樣，與學習式線性迴歸器較清晰的輸出。

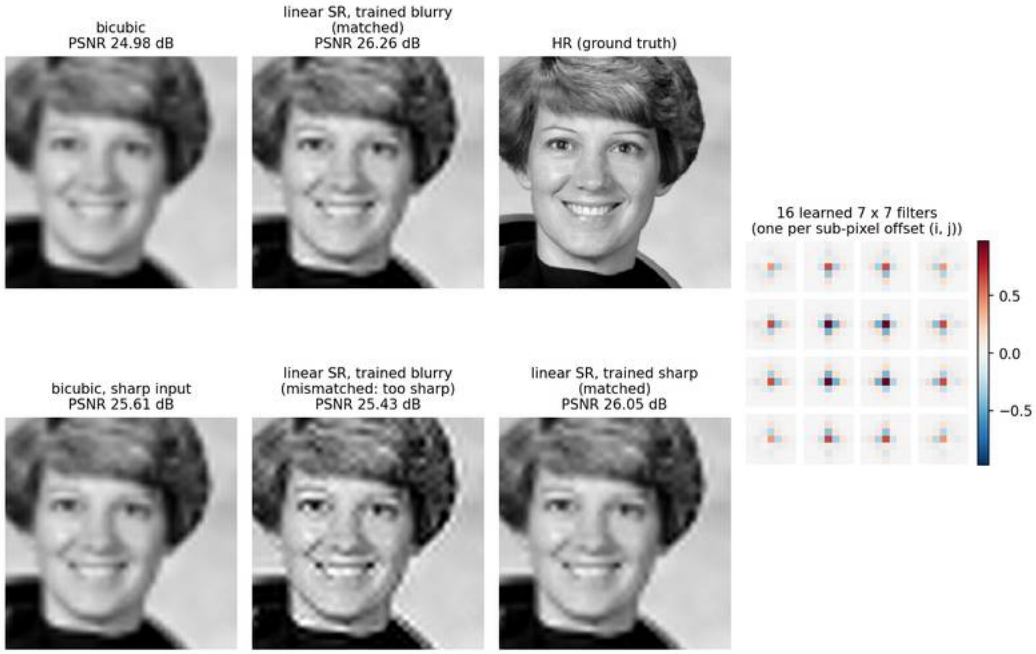


圖 *6* — 學習式線性迴歸器（每個子像素偏移一個 7×7 濾波器，預測相對於雙三次的殘差）。訓練與測試的退化相符時有幫助；不相符時（下列中間）會過度銳化，分數低於雙三次。*PSNR* 是在完整的灰階測試影像上、裁去 *4* 像素邊界後計算。

深度 SR：六個設計決策

深網路是前面線性迴歸的非線性擴充。從三層 CNN 到 Transformer，可沿六個解決具體問題、也帶來成本的決策理解。

表 1 是本節地圖，下文按同一順序解釋。

決策	代表模型	解決問題	代價
1. Bicubic 後端到端學習	SRCNN [14]	手工區塊管線	每層在 HR 工作
1. 改在末端放大	FSRCNN [15]、ESPCN [16]	HR 的 s^2 成本	每倍率放大尾端
2. 子像素卷積	ESPCN [16]	學得放大器成本	初始化棋盤紋
3. 深網路及殘差	VDSR [17]	深層訓練慢／不穩	計算與啟動記憶體
4. 去 BN、殘差縮放	EDSR [18]	記憶體及動態範圍	寬模型須穩定化
5. 通道注意力、殘差中殘差	RCAN [19]	通道等權、400 多層訓練	深度帶來延遲
6. 遠距內容注意力	SwinIR [20]、HAT [21]	固定局部卷積	注意力時間／記憶體

這些像素損失模型仍是中心估計：MSE 為條件平均，L1 為逐座標中位數，不代表完整分布。架構可改善預測，下一節則改變統計目標。

決策一：在哪裡放大

白話說。每軸放大 s 倍後有 s^2 倍像素。先放大使後續每層都在大圖工作，後放大則讓主體留在小圖。

SRCNN [14] 先 bicubic，再三卷積做區塊抽取、非線性映射與重建。前放大簡單，也可共用多倍率，如 VDSR [17]；但 $k \times k$、$C_{\text{in}} \to C_{\text{out}}$、$h \times w$ 卷積需要：

$$\text{MAC} = h\, w\, C_{\text{in}}\, C_{\text{out}}\, k^2$$

次乘加。成本正比 hw，相同層在 HR 精確多 s^2 倍。

```
def conv_macs(h, w, c_in, c_out, k=3):
    """Multiply-accumulates of one k x k convolution on an h x w feature map."""
    return h * w * c_in * c_out * k * k

s, H, W, C = 4, 128, 128, 64                    # x4 SR of a 128 x 128 input, 64 feature
 ↪  channels
body_pre = conv_macs(s * H, s * W, C, C)        # one body layer at HR size (pre-upsampling)
body_post = conv_macs(H, W, C, C)               # the same layer at LR size (post-upsampling)
tail = conv_macs(H, W, C, 3 * s * s)            # sub-pixel tail: 64 -> 3*s*s channels, then
 ↪  pixel shuffle
print(f"one body layer, pre-upsampling : {body_pre / 1e9:.2f} GMAC")
print(f"one body layer, post-upsampling: {body_post / 1e9:.2f} GMAC (ratio {body_pre /
 ↪  body_post:.0f})")
print(f"sub-pixel tail, 64 -> 48 channels at LR size: {tail / 1e9:.2f} GMAC")
```

```
one body layer, pre-upsampling : 9.66 GMAC
one body layer, post-upsampling: 0.60 GMAC (ratio 16)
sub-pixel tail, 64 -> 48 channels at LR size: 0.45 GMAC
```

128 × 128 LR、×4 的 64 通道層，HR 約 **9.66 GMAC**、LR **0.60**。FSRCNN [15] 用末端轉置卷積及薄映射，在 C++ Intel i7 比較中比 SRCNN-Ex 快逾 **40 倍**。代價是尾端依倍率；MDSR [18] 共享主體、×2 ／ ×3 ／ ×4 各尾端。後續多採末端放大。

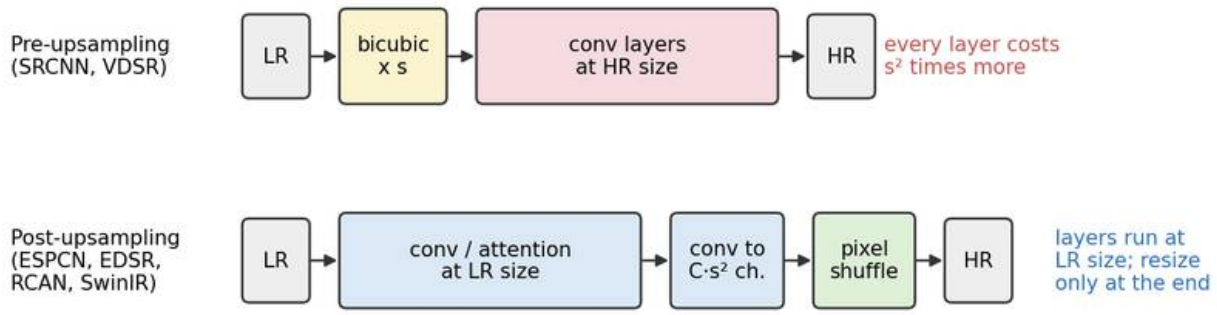


圖 7 — 先上取樣（*SRCNN*、*VDSR*）與使用子像素卷積的後上取樣（*ESPCN* 與大多數後來的網路）。*FSRCNN* 也在最後才上取樣，但用的是轉置卷積。

決策二：轉置或子像素卷積

白話說。每 LR 點需產生 s^2 個輸出。轉置卷積把每點以學得印章分散出去；子像素卷積在 LR 算成 s^2 通道，再重排為區塊。

stride s 的轉置卷積等價於樣本間插 $s-1$ 零再卷積。不同輸出偏移只使用不同核係數，總共 s^2 模式，因此可寫成 LR 的多個卷積再交錯輸出 [16]。ESPCN 末層直接輸出 Cs^2，無參數 pixel shuffle 變成 C 通道大圖；EDSR ／ RCAN ／ SwinIR ／ HAT 也使用 [18]、[19]、[20]、[21]。

精確的版本是一條索引規則。設 T 是形狀為 (Cs^2, H, W) 的 LR 特徵張量，$X = \mathrm{PS}(T)$ 是形狀為 (C, sH, sW) 的輸出，則

$$X[c,\ sh+i,\ sw+j] = T[c\,s^2 + s\,i + j,\ h,\ w], \qquad 0 \le c < C,\ \ 0 \le h < H,\ \ 0 \le w < W,\ \ 0 \le i,j < s,$$

c 是輸出通道，(h, w) 是 LR 位置，(i, j) 是區塊內子像素偏移；依行優先將 s^2 通道散到區塊。這符合 PyTorch `PixelShuffle` [22] 將通道拆成 (C, s, s) 再排列的慣例。反向 pixel unshuffle 把區塊收回通道，高效率模型可用它先降低空間大小。

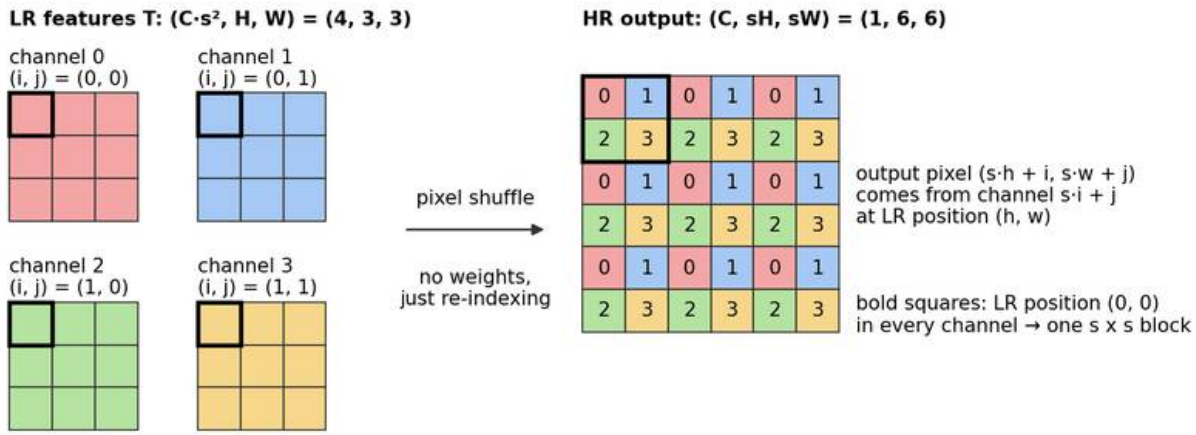


圖 *8* — $s = 2$、$C = 1$ 時的 *pixel shuffle*。同一個 *LR* 位置上的四個值（每個通道一個）變成一個 2×2 的輸出區塊。

NumPy 用 reshape ／ transpose ／ reshape 實作兩方向，逐元素檢查索引式及精確互逆。

```
import numpy as np

def pixel_shuffle(t, s):
    """Depth-to-space: (C*s*s, H, W) -> (C, H*s, W*s)."""
    Cs2, H, W = t.shape
    C = Cs2 // (s * s)
    return t.reshape(C, s, s, H, W).transpose(0, 3, 1, 4, 2).reshape(C, H * s, W * s)

def pixel_unshuffle(x, s):
    """Space-to-depth: (C, H*s, W*s) -> (C*s*s, H, W)."""
    C, Hs, Ws = x.shape
    H, W = Hs // s, Ws // s
    return x.reshape(C, H, s, W, s).transpose(0, 2, 4, 1, 3).reshape(C * s * s, H, W)

rng = np.random.default_rng(0)
s, C, H, W = 3, 2, 4, 5
t = rng.standard_normal((C * s * s, H, W))
x = pixel_shuffle(t, s)
print(x.shape)                                                  # (2, 12, 15)

# The index rule: output (c, s*h + i, s*w + j) comes from channel c*s*s + s*i + j at (h, w).
ok = all(x[c, s*h + i, s*w + j] == t[c*s*s + s*i + j, h, w]
         for c in range(C) for h in range(H) for w in range(W)
         for i in range(s) for j in range(s))
print("index rule holds:", ok)
print("unshuffle(shuffle(t)) == t:", np.array_equal(pixel_unshuffle(x, s), t))
print("shuffle(unshuffle(x)) == x:", np.array_equal(pixel_shuffle(pixel_unshuffle(x, s), s),
 ↪  x))
```

```
(2, 12, 15)
index rule holds: True
```

```
unshuffle(shuffle(t)) == t: True
shuffle(unshuffle(x)) == x: True
```

核尺寸為 s 倍數時，可對應兩種放大表示。Pixel shuffle 是排列、精確可逆、梯度為反排列，但各偏移獨立濾波，隨機初始化可能讓平坦輸入變棋盤紋。ICNR [23] 將偏移濾波初始化為副本，起始等價卷積後最近鄰放大。DLCV 01B 有程式分析；線性例子的 16 個核就是此偏移。

決策三：深度與殘差預測

白話說。深層網路捕捉較廣上下文，但較難最佳化；殘差學習只預測簡單放大輸入的修正量。

VDSR [17] 以 **20** 層 3×3 卷積和兩項設計使深層可訓練。先用全域殘差，讓 f_θ 只補插值修正：

$$\hat{x} = U(y) + f_\theta(U(y)), \qquad \mathcal{L}(\theta) = \tfrac{1}{2}\|x - U(y) - f_\theta(U(y))\|_2^2,$$

U 為 bicubic，f_θ 接收 HR 大小輸入，$\hat{x}$ 是輸出，$\mathcal{L}$ 為配對損失；再用高學習率搭配**可調梯度裁切** [17]。

僅線上版：可調梯度裁切 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/notes/可調梯度裁切/*

插值已含低頻，目標 $x - U(y)$ 較小、以零為中心；以下在貫穿範例測量。

```
import numpy as np, cv2
from scipy import ndimage as ndi
from skimage import data, img_as_float64
from skimage.color import rgb2gray

s = 4
hr = img_as_float64(rgb2gray(data.astronaut()))                          # (512, 512), values
 ↪  in [0, 1]
lr = ndi.gaussian_filter(hr, 1.2).reshape(128, s, 128, s).mean(axis=(1, 3))   # the running
 ↪  example
up = cv2.resize(lr, None, fx=s, fy=s, interpolation=cv2.INTER_CUBIC)   # bicubic upsampling
r = hr - up                                                              # what a residual
 ↪  network predicts
print(f"HR image: mean {hr.mean():.3f}  std {hr.std():.3f}")
print(f"residual: mean {r.mean():+.3f}  std {r.std():.3f}  |r| < 0.05 at {np.mean(np.abs(r) <
 ↪  0.05):.0%} of pixels")
```

```
HR image: mean 0.442  std 0.296
residual: mean -0.000  std 0.056  |r| < 0.05 at 82% of pixels
```

殘差標準差約為影像五分之一，**82%** 像素絕對值低於 **0.05**。零輸出已接近 bicubic，可集中容量處理邊紋。末端放大模型保留局部殘差及長跳接，見 DLCV 01B。加法便宜，不代表增加主體深度免費。

決策四：移除批次正規化

白話說。 BN 重新縮放特徵，分類常有用，但 SR 要保留精確強度與範圍。EDSR 移除它，將記憶體用於更寬網路。

訓練 BN 用批次統計，測試用移動統計。SRResNet [24] 保留 BN，EDSR [18] 為範圍彈性與記憶體移除，訓練相較約省 **40%**。完整模型 **32** 殘差塊、**256** 通道，以 **0.1** 殘差縮放穩定；改 L1，×3 ／ ×4 由 ×2 初始化，贏 NTIRE 2017 [25]。ESRGAN [26] 又指出統計不符時 BN 可生偽影。

去 BN 仍須小心初始化及縮放。啟動範圍也可能更大，使量化困難，見 DLCV 01C；一般正規化選擇見 DLCV 01B。

決策五：通道重新加權（RCAN）

白話說。不同通道承載低頻或細節，通道注意力依影像調高低，而非一律等權。

RCAN [19] 對各通道取平均，經瓶頸產生縮放。對 C 個 $H \times W$ 通道 $x_1, \dots, x_C$：

$$z_c = \frac{1}{HW} \sum_{i=1}^{H} \sum_{j=1}^{W} x_c(i,j), \qquad \mathbf{s} = f(W_U\, \delta(W_D\, \mathbf{z})), \qquad \hat{x}_c = s_c\, x_c,$$

$x_c(i,j)$ 是值，$\mathbf{z}$ 為通道平均，W_D 將 C 降為 C/r，δ 是 ReLU，W_U 還原，f 是 sigmoid，$s_c \in (0, 1)$ 縮放通道。$C = 64, r = 16$，瓶頸僅 **4** 通道。一般 SE 背景見 DLCV 01B。

殘差中殘差含 **10** 組、每組 **20** 通道注意力塊、短及長跳接，超過 **400** 層 [19]。低頻可走跳接，主體偏細節，但深度仍增加序列執行延遲。

決策六：遠距內容注意力（SwinIR、HAT）

白話說。卷積用固定權重，注意力依內容產生權重，可借相似結構資訊。整圖昂貴，因此使用交替位移小視窗。

每個 $M \times M$ LR token 視窗，注意力為：

$$\text{Attention}(Q, K, V) = \text{SoftMax}(QK^{\top}/\sqrt{d} + B)V,$$

Q, K, V 為線性投影、各 M^2 列，d 是 query ／ key 維度，B 為相對位置偏置 [21]。每輸出按 query—key 相似度平均 values，見 ML 05。$h \times w$ token、C 通道成本：

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C, \qquad \Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC,$$

MSA 是全域，W-MSA 是視窗，首項投影、次項注意力 [27]。固定視窗時對像素線性，整圖則平方。$C = 180, M = 8$、128 × 128 LR，每層約 **98.8 GMAC** 全域、**2.50 GMAC** 視窗，是本章算術例。

SwinIR [20] 淺層 3 × 3、殘差 Swin 主體及 pixel shuffle；標準設定 **6** 塊、各 **6** 層、視窗 **8**、寬 **180**、L1。HAT [21] 依 LAM 歸因發現利用範圍有限，加入平行通道注意力、視窗 **16**、重疊交叉注意力，及同 SR 任務的 ImageNet 預訓練，再微調 DIV2K ／ Flickr2K。DAT [28] 交替空間／通道，MambaIR [29] 用狀態掃描擴大上下文。

視窗、容量及大圖分塊增加時間與記憶體，也可能留接縫。L1 架構的改善來自更有效上下文，不是隨機抽樣。

共同配方。 DIV2K 或加 Flickr2K 小裁切，翻轉、90° 旋轉、L1、Adam；EDSR 用 48 × 48 LR。「+」如 EDSR+ 為八種幾何變換輸出還原平均 [18]，並非免費。一般損失見 DLCV 09。

感知式與對抗式 SR

改變損失可獎勵自然細節，代價可能是較低 PSNR 與虛構，而非只改網路架構。

感知損失與 GAN

Johnson 等人 [31] 用 VGG [30] 感知距離得到較清晰但較低 PSNR；SRGAN [24] 加判別器與 **16** 殘差塊 SRResNet，提升人類偏好。ESRGAN [26] 用無 BN 的 RRDB、相對判別器和啟動前 VGG 特徵，其變體贏 PIRM 2018 區域 3 [32]。常見生成損失為：

$$\mathcal{L} = \lambda_1 \lVert \hat{x} - x \rVert_1 + \lambda_p \lVert \phi(\hat{x}) - \phi(x) \rVert_2^2 + \lambda_a \mathcal{L}_{\text{adv}}(\hat{x}),$$

$\hat{x}$ 為輸出，x 為目標，ϕ 為固定特徵層，$\mathcal{L}_{\text{adv}}$ 為對抗項，$\lambda_1, \lambda_p, \lambda_a \geq 0$ 為權重；GAN 見 ML 07。

ESRGAN [26] 的**網路內插**以 $\theta = (1-\alpha)\theta_{\text{PSNR}} + \alpha\theta_{\text{GAN}}$、$0 \leq \alpha \leq 1$ 混合像素模型及 GAN 微調權重，不重訓探索平滑／紋理取捨。

僅線上版：*ESRGAN 設計詳解 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/notes/esrgan-設計詳解/*

感知–失真權衡

Blau 與 Michaeli [2] 以輸出和真值的期望差異定義**失真**，以輸出分布與真實分布的距離定義**感知品質**，研究最佳邊界。允許失真越小，最佳可達分布品質不會更好；歧義後驗的平均可能不自然，但無雜訊可逆問題未必有衝突。詳見 DLCV 09。本章從平均移向後驗樣本，MSE **0.0072→0.0143**，自然度差距縮小。非線性權重內插不等於輸出內插，也不保證位於理論最優邊界。

- 方法應該說明它瞄準權衡上的哪個位置。只用 PSNR，或只用一個感知指標，去比較 GAN 與 MSE 網路，無法說明哪一個比較好。
- 基準必須同時量測兩個軸。PIRM 2018 挑戰賽就是這樣做的：它依 RMSE 把感知–失真平面分成三個區域（RMSE 上限分別為 11.5、12.5 與 16），在每個區域內以一個感知指數為方法排名，這個指數由兩個無參考指標組成：NIQE [33] 與 Ma 等人的 SR 專用指標 [34], [32]。

已知退化的擴散 SR

條件擴散近似條件影像分布，可產生多個合理候選，代價是迭代推論。訓練好的取樣器不自動證明精確貝氏後驗或量測一致性。

SR3 [35] 以 bicubic LR 做非盲條件，U-Net [36] 輸入為吵 HR 與 bicubic 放大 LR 的通道串接，學多種雜訊程度去除。推論從純高斯開始，原文最多 **100** 步。

×8 人臉 16→128 的研究，受試者把 SR3 當真照片接近 **50%**，GAN 基準至多 **34%**。×4 ImageNet 同骨幹回歸 PSNR ／ SSIM 較高，SR3 樣本品質較好；它也檢查縮回 LR 的一致性 [35]。訓練程序見 DLCV 09B，擴散機制見 ML 08。

生成細節未對齊會降 PSNR，即使人類偏好，但不是所有模型必然較低。內容仍可能錯且多次評估昂貴；大型先驗、少步／一步真實照片方法見 DLCV 09A.1。

退化未知：盲與真實世界 SR

白話說。真圖帶未知模糊、雜訊及壓縮，固定 bicubic 模型會像前面錯配線性例子般誤判。方法可廣泛合成、先估退化或實拍配對，也須在無真值下評估；完整內容見 DLCV 09A.1：盲與真實世界超解析度。

任意倍率 SR

白話說。固定 ×4 模型只能產生特定大小；任意倍率模型把影像視為連續函數，能查詢 ×2.7 等非整數輸出。

Meta-SR [37] 依倍率及輸出相對位置預測放大濾波權重；LIIF [38] 學**局部隱式影像函數**：

$$\hat{I}(x_q) = f_\theta(z^*,\ x_q - v^*),$$

x_q 是連續查詢座標，z^* 是 LR 編碼器在最近網格 v^* 的特徵，f_θ 是輸出 RGB 的小 MLP。原文以 ×1–×4 訓練，展示至 **×30**。代價是每輸出像素一次 MLP；極端倍率依賴學得假設，不會增加量測資訊。

僅線上版：*LIIF 的三項改良 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/super-resolution/notes/liif-的三項改良/*

效率

白話版。許多 SR 用途（手機、視訊通話、遊戲算繪）需要在小晶片上以毫秒級的時間給出結果。高效率 SR 問的是：在維持品質門檻的前提下，網路可以做到多小、多快。

末端 pixel shuffle 先省空間成本，其他如 IMDN [39] 分出保留特徵、繼續處理部分，再以對比感知通道注意力融合。參數、FLOPs、時間不同；NTIRE 高效率 2024 ／ 2025 ×4 [40]、[41] 要求驗證 **26.90 dB**、測試 **26.99 dB** 以上再降低成本。壓縮見 DLCV 01C。

評估 SR

保真依固定 PSNR ／ SSIM 規則，外觀另加感知及人類評估，不能只用一個數字混排。一般指標見 DLCV 09，此處聚焦 SR 特有細節。

在 Y 通道上計算並裁去邊界的 PSNR 與 SSIM

經典 MATLAB 社群規則通常是：

1. 把 HR 影像與 SR 輸出都轉成 YCbCr，只保留亮度（luma）通道

$$Y = 16 + 65.481\,R + 128.553\,G + 24.966\,B,$$

$R, G, B \in [0, 1]$ 為 gamma 編碼，$Y \in [16, 235]$；係數依 MATLAB ／ scikit-image `rgb2ycbcr` 的 BT.601 [42]、[43]。2. 兩張圖裁掉邊框，避免 padding 邊界主導。3. 剩餘 Y 以峰值 **255** 算 PSNR 及 SSIM [44]，定義見 DLCV 09。

同篇也可能不同：EDSR [18] 基準 Y 裁 s，消融 DIV2K 則 RGB 裁 $6 + s$；NTIRE ×4 裁 **4** [45]、[46]。通道／色域曾在報告例子差達 **4 dB** [4]。若 $Y = 16 + 219\ell$，以 Y ／峰 255 比 ℓ ／峰 1 多 $20 \log_{10}(255/219) \approx 1.32$ dB。量化、裁切和縮放也有影響，公平比較應用同一評估腳本。

標準測試集

SR 論文在一小組公開測試集上回報結果，每個測試集各有特色 [4]：

測試集	影像數	內容	出處
Set5	5	嬰兒、鳥、蝴蝶、頭像、女性	[47]
Set14	14	人物、動物、花、漫畫、投影片	[48]
BSD100 (B100)	100	自然場景，即 Berkeley 分割資料集的測試影像	[49]
Urban100	100	具有規則結構的建築與城市景觀	[50]
Manga109	109	日本漫畫單行本（線稿與網點）	[51], [52]
DIV2K validation	100	多樣的 2K 解析度照片	[53]

Set5、Set14 很小，百分之幾 dB 意義有限；Urban100 重複結構多，大視窗 HAT 在此尤其受益 [21]。

為什麼 PSNR 與感知排名不一致

三種原因造成分歧：MSE 最優中心可能模糊；紋理統計對卻位移，像素誤差仍大；感知／無參考模型如 LPIPS [54]、NIQE [33] 評外觀，也可能被過銳化影響。NTIRE 2025 [46] 分 PSNR 復原及六指標合成感知賽道。DLCV 09 建議同報保真、感知，生成模型另做人類研究。

資料集與基準

白話說。幾個資料集與年度競賽形塑了非盲 SR 的常見基準。

- **DIV2K** [53] 有 **1,000** 張 2K 圖，訓練／驗證／測試 **800** ／ **100** ／ **100** [46]；Flickr2K 有 **2,650**，合稱 DF2K。HAT [21] 先 ImageNet 同任務預訓練，再微調。
- **NTIRE 2017** [25] 各有 bicubic、可由配對學習的未知退化，×2 ／ ×3 ／ ×4。
- **NTIRE 2024** ／ **2025** [45]、[46] 用 DIV2K ×4 bicubic，2025 加感知賽道。
- **高效率 SR** [40]、[41] 在固定品質門檻下比較時間、參數與 FLOPs。

真實資料與挑戰不只有乾淨 bicubic，見 DLCV 09A.1。

失效模式

白話版。 SR 模型失敗的方式，可能是太謹慎（模糊），也可能是太大膽（捏造細節）。大膽的失敗比較危險，因為結果看起來很有說服力。

- **過度平滑：**中心估計可能抑制歧義細節，MSE ／ L1 分別學平均／中位數，不保證每張都模糊，也不表示所有候選。
- **假文字／人臉：**清楚卻錯字或身分；Ren 等人 [55] 指出既有指標不足，提議多模態虛構評估。
- **紋理偽影：** GAN 重複噪狀細節、錯核暈邊 [26]。
- **偏色：**擴散仍有全域色彩漂移 [56]。
- **退化錯配：**修不足／過度，補救見 DLCV 09A.1。
- **接縫：**大圖非重疊分塊可留下邊界 [20]。

對任何 SR 輸出都很有用的一個合理性檢查，是**與輸入的一致性**：用假設的模型把輸出退化，再與 y 比較。差異很大，代表模型改動了 LR 影像確實量測到的內容。

現代觀點

綜述論文

Wang、Chen、Hoi [4] 分監督、無監督、領域專用，再以放大位置、算子、殘差／遞迴／密集／注意力、損失與訓練策略等獨立軸描述，提醒新模型通常只改其中幾項。它早於 Transformer ／擴散。Moser 等人 [56] 則按像素／潛在／小波、條件、引導、損傷空間及零樣本整理擴散 SR，指出成本、比較、可解釋性與偏色。盲及真實綜述見 DLCV 09A.1。

2023–2026 年的最新進展

1. **保真架構：** HAT、DAT、MambaIR [21]、[28]、[29] 搭配預訓練，在 NTIRE 復原賽道評估 [45]、[46]。
2. **評估跟上生成：**感知專用賽道及虛構評估 [46]、[55]，不再只依 PSNR。
3. **效率獨立比較：**在品質門檻下測成本 [40]、[41]。

尚未解決的問題

- **可信生成：**如何提供自然細節，同時辨識哪些已量測、哪些只是推估。

- **評估：**中心估計、感知分數迎合與新虛構指標的限制。
- **小基準飽和：**微小 dB 增益不代表真實相機用途。
- **成本：**大注意力／擴散與高效率網路差異很大。
- **極端倍率：**訓練範圍外、幾乎全靠先驗時的行為仍不明確。

重點整理

- 已知下採樣仍會遺失資訊，未確定細節由先驗決定。
- MSE 偏向條件平均，L1 偏向逐座標中位數；兩者都可能平滑不確定結構。
- 非盲設定的結果，取決於假設退化是否符合輸入。
- 後段上採樣、殘差路徑與特徵混合影響成本及細節恢復，應在相同配方下比較。
- 感知—失真前緣不禁止從較弱基準同時改善兩類指標；學得的生成器也沒有精確取樣保證。
- 註明色彩通道與邊界裁切，檢查低解析度一致性，以及文字、人臉、紋理和接縫。

練習

1. **OpenCV 用的是哪一種三次核？**做一個由 16 個 0 組成、正中間有一個 1 的一維訊號，用 `cv2.resize(..., interpolation=cv2.INTER_CUBIC)` 放大 4 倍，再把結果與 Keys 核在半像素對齊位置 $(t+\frac{1}{2})/4-\frac{1}{2}-8$ 上的值比較，分別取 $a=-0.5$ 與 $a=-0.75$。接著畫出兩個核的頻率響應，比較它們在 LR 奈奎斯特頻率處的值。

   提示

   對單位脈衝做縮放，輸出就是在輸出位置上取樣的核本身。在我們的執行中，$a=-0.75$ 的最大絕對差為 0.0，$a=-0.5$ 約為 0.037。a 越負，負瓣越深，響應在 1 附近維持得稍久一點，然後略微過衝：這是更銳利、但振鈴更明顯的內插器。用 `INTER_LANCZOS4` 與 $a=4$ 的 Lanczos 核重做一次；結果很接近但並不完全相同（我們量到的最大差約為 0.0016），因為函式庫會把權重正規化。

2. **評估算術。**（a）證明 Y ／峰 255 的 PSNR 等於 $\ell=(Y-16)/219$ ／峰 1 的 PSNR 加 $20\log_{10}(255/219)$。（b）把裁邊從 4 改 0 或 16，比較。（c）為什麼零 padding 網路尤其需要裁邊，`cv2.resize` 影響較小？

   提示

   (a) $Y_{\text{hr}}-Y_{\text{sr}}=219(\ell_{\text{hr}}-\ell_{\text{sr}})$，所以 MSE 乘上 219^2，PSNR 變成 $10\log_{10}(255^2/(219^2\,\text{MSE}_\ell))$，也就是 $\text{PSNR}_\ell+20\log_{10}(255/219)$，約為 $\text{PSNR}_\ell+1.32$ dB。(b) 在這裡裁邊只讓結果改變幾百分之一 dB（最近鄰為 24.66 對 24.62 dB），因為 OpenCV 對邊界的外插相當合理，影像的邊緣也很平滑。(c) 零填補讓卷積網路在影像之外看到黑色，所以靠近邊界的輸出系統性地較差；裁邊去掉的是一塊品質取決於填補方式、而不是取決於方法本身的區域。

3. **通道順序的錯誤。**假設某個函式庫把 Cs^2 個通道排成 $k=cs^2+si+j$（我們的公式），另一個排成 $k=(si+j)C+c$。你把用第一種順序訓練的權重載入第二種。$C=3$、$s=2$ 時，12 個通道中有哪些會落在正確的位置？輸出看起來會是什麼樣子？

提示

令 $q = 2i + j$，解 $4c + q = 3q + c$：$3c = 2q$，所以只有 $(c, q) = (0, 0)$ 與 $(2, 3)$ 重合，12 個通道中只有 2 個（我們逐一列舉全部 12 個確認過）。其他值被移到錯誤的色彩通道與錯誤的子像素偏移，所以即使粗略的影像仍認得出來，每個 2×2 區塊都會出現細碎的彩色馬賽克。$C = 1$ 時兩種順序一致，這就是為什麼這類錯誤常常要到從灰階模型換成彩色模型時才出現。

4. **讓線性模型更好**。從迴歸器的程式出發，(a) 把區塊大小 p 改成 3、5、9 與 11；(b) 依局部梯度的角度把 LR 像素分類（如 RAISR 的做法），每一類擬合一個迴歸器；(c) 在 LR 訓練影像上加 $\sigma = 0.01$ 的雜訊、測試影像不加，再反過來做一次。回報每項改動的 PSNR 並解釋。

提示

(a) 在模糊的測試輸入上，我們量到 $p = 3$ 為 25.94 dB、$p = 5$ 為 26.18 dB，$p = 7, 9, 11$ 都是 26.26 dB：超過 7×7 之後，線性模型就不再進步。(b) 四個角度類別（由稍微平滑過的 LR 影像梯度算出）得到 26.33 dB，八個類別得到 26.32 dB，提升很小：類別讓濾波器能順著邊緣方向，這正是 RAISR 與 A+ 勝過單一全域濾波器的原因。(c) 用有雜訊的輸入訓練，模型學會平滑，所以對乾淨輸入銳化不足；用乾淨輸入訓練，模型學會放大細緻的細節，所以會連雜訊也一起放大。這是圖 6 中模糊不匹配的雜訊版本。

5. **手算後驗平均**。兩個 HR 像素 $a, b \in \{0, 1\}$ 彼此獨立，為 0 或 1 的機率相等。LR 像素為 $y = (a + b)/2$，沒有雜訊。(a) 對 y 的每個可能值，寫出 (a, b) 的後驗平均與一個後驗樣本。(b) 計算兩個估計器每個像素的期望 MSE。(c) 哪一個估計器的輸出看起來像 HR 資料？

提示

(a) $y = 0$ 與 $y = 1$ 完全確定了 (a, b)。當 $y = 0.5$（機率 1/2）時，後驗是 $(0, 1)$ 或 $(1, 0)$，機率相等，所以平均為 $(0.5, 0.5)$，樣本則是兩者之一。(b) 只有 $y = 0.5$ 有貢獻。平均在那裡每個像素的平方誤差為 0.25，所以它的 MSE 是 $0.5 \times 0.25 = 0.125$。樣本有一半的時間完全正確，另一半的時間兩個像素都錯 1，所以它的 MSE 是 $0.5 \times 0.5 \times 1 = 0.25$：剛好兩倍。(c) 只有樣本：真實像素是 0 或 1，0.5 這個值從不出現在 HR 資料中。

6. **全域注意力的代價**。用決策六公式，計算 $C = 180, M = 8$、64×64 與 128×128 LR 的 Ω(MSA)、Ω(W-MSA)。邊長倍增時比值如何變？多大的視窗會與全域同成本？

提示

64×64 約 **6.6** ／ **0.63 GMAC**、比 **10.5**；128×128 約 **98.8** ／ **2.50**、比 **39.5**。像素四倍，投影 $4hwC^2$ 與視窗項四倍，全域 $2(hw)^2C$ 十六倍。$M^2 = hw$ 時一窗覆全圖，兩注意力項相等。

參考文獻

1. S. C. Park, M. K. Park and M. G. Kang, “Super-Resolution Image Reconstruction: A Technical Overview,” *IEEE Signal Processing Magazine*, vol. 20, no. 3, pp. 21–36, 2003. doi

2. Y. Blau and T. Michaeli, “The Perception-Distortion Tradeoff,” in *Proc. IEEE/CVF CVPR*, 2018, pp. 6228–6237. doi ·arXiv
3. MathWorks, “imresize: Resize image,” MATLAB documentation (default method bicubic; antialiasing on when shrinking). link
4. Z. Wang, J. Chen and S. C. H. Hoi, “Deep Learning for Image Super-Resolution: A Survey,” *IEEE TPAMI*, vol. 43, no. 10, pp. 3365–3387, 2021. doi ·arXiv
5. R. Keys, “Cubic Convolution Interpolation for Digital Image Processing,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981. doi
6. C. E. Duchon, “Lanczos Filtering in One and Two Dimensions,” *Journal of Applied Meteorology*, vol. 18, no. 8, pp. 1016–1022, 1979. doi
7. W. T. Freeman, T. R. Jones and E. C. Pasztor, “Example-Based Super-Resolution,” *IEEE Computer Graphics and Applications*, vol. 22, no. 2, pp. 56–65, 2002. doi
8. D. Glasner, S. Bagon and M. Irani, “Super-Resolution from a Single Image,” in *Proc. IEEE ICCV*, 2009, pp. 349–356. doi
9. J. Yang, J. Wright, T. S. Huang and Y. Ma, “Image Super-Resolution via Sparse Representation,” *IEEE Transactions on Image Processing*, vol. 19, no. 11, pp. 2861–2873, 2010. doi
10. R. Timofte, V. De Smet and L. Van Gool, “Anchored Neighborhood Regression for Fast Example-Based Super-Resolution,” in *Proc. IEEE ICCV*, 2013, pp. 1920–1927. doi ·CVF
11. R. Timofte, V. De Smet and L. Van Gool, “A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution,” in *Computer Vision – ACCV 2014*, Lecture Notes in Computer Science, Springer, 2015, pp. 111–126. doi
12. Y. Romano, J. Isidoro and P. Milanfar, “RAISR: Rapid and Accurate Image Super Resolution,” *IEEE Transactions on Computational Imaging*, vol. 3, no. 1, pp. 110–125, 2017. doi ·arXiv
13. N. Efrat, D. Glasner, A. Apartsin, B. Nadler and A. Levin, “Accurate Blur Models vs. Image Priors in Single Image Super-Resolution,” in *Proc. IEEE ICCV*, 2013, pp. 2832–2839. doi ·CVF
14. C. Dong, C. C. Loy, K. He and X. Tang, “Image Super-Resolution Using Deep Convolutional Networks,” *IEEE TPAMI*, vol. 38, no. 2, pp. 295–307, 2016. doi ·arXiv
15. C. Dong, C. C. Loy and X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network,” in *Proc. ECCV*, 2016, pp. 391–407. doi ·arXiv
16. W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert and Z. Wang, “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network,” in *Proc. IEEE CVPR*, 2016, pp. 1874–1883. doi ·arXiv
17. J. Kim, J. K. Lee and K. M. Lee, “Accurate Image Super-Resolution Using Very Deep Convolutional Networks,” in *Proc. IEEE CVPR*, 2016, pp. 1646–1654. doi ·arXiv
18. B. Lim, S. Son, H. Kim, S. Nah and K. M. Lee, “Enhanced Deep Residual Networks for Single Image Super-Resolution,” in *Proc. IEEE CVPR Workshops*, 2017, pp. 1132–1140. doi ·arXiv
19. Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong and Y. Fu, “Image Super-Resolution Using Very Deep Residual Channel Attention Networks,” in *Proc. ECCV*, 2018, pp. 294–310. doi ·arXiv
20. J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool and R. Timofte, “SwinIR: Image Restoration Using Swin Transformer,” in *Proc. IEEE/CVF ICCV Workshops*, 2021, pp. 1833–1844. doi ·arXiv
21. X. Chen, X. Wang, J. Zhou, Y. Qiao and C. Dong, “Activating More Pixels in Image Super-Resolution Transformer,” in *Proc. IEEE/CVF CVPR*, 2023, pp. 22367–22377. doi ·arXiv
22. PyTorch contributors, “PixelShuffle.cpp” (implementation of `torch.nn.PixelShuffle` and `torch.nn.PixelUnshuffle`), PyTorch source code. GitHub

23. A. Aitken, C. Ledig, L. Theis, J. Caballero, Z. Wang and W. Shi, “Checkerboard Artifact Free Sub-Pixel Convolution: A Note on Sub-Pixel Convolution, Resize Convolution and Convolution Resize,” arXiv:1707.02937, 2017. arXiv
24. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” in *Proc. IEEE CVPR*, 2017, pp. 105–114. doi ·arXiv
25. R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang et al., “NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results,” in *Proc. IEEE CVPR Workshops*, 2017, pp. 1110–1121. doi ·CVF
26. X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao and C. C. Loy, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” in *Proc. ECCV Workshops*, 2018 (Lecture Notes in Computer Science, 2019), pp. 63–79. doi ·arXiv
27. Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin and B. Guo, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows,” in *Proc. IEEE/CVF ICCV*, 2021, pp. 9992–10002. doi ·arXiv
28. Z. Chen, Y. Zhang, J. Gu, L. Kong, X. Yang and F. Yu, “Dual Aggregation Transformer for Image Super-Resolution,” in *Proc. IEEE/CVF ICCV*, 2023, pp. 12278–12287. doi ·arXiv
29. H. Guo, J. Li, T. Dai, Z. Ouyang, X. Ren and S.-T. Xia, “MambaIR: A Simple Baseline for Image Restoration with State-Space Model,” in *Proc. ECCV*, 2024, pp. 222–241. doi ·arXiv
30. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” in *Proc. ICLR*, 2015. arXiv
31. J. Johnson, A. Alahi and L. Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution,” in *Proc. ECCV*, 2016, pp. 694–711. doi ·arXiv
32. Y. Blau, R. Mechrez, R. Timofte, T. Michaeli and L. Zelnik-Manor, “The 2018 PIRM Challenge on Perceptual Image Super-Resolution,” in *Proc. ECCV Workshops*, 2018. arXiv
33. A. Mittal, R. Soundararajan and A. C. Bovik, “Making a “Completely Blind” Image Quality Analyzer,” *IEEE Signal Processing Letters*, vol. 20, no. 3, pp. 209–212, 2013. doi
34. C. Ma, C.-Y. Yang, X. Yang and M.-H. Yang, “Learning a No-Reference Quality Metric for Single-Image Super-Resolution,” *Computer Vision and Image Understanding*, vol. 158, pp. 1–16, 2017. doi ·arXiv
35. C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet and M. Norouzi, “Image Super-Resolution via Iterative Refinement,” *IEEE TPAMI*, 2022. doi ·arXiv
36. O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Proc. MICCAI*, 2015, pp. 234–241. doi ·arXiv
37. X. Hu, H. Mu, X. Zhang, Z. Wang, T. Tan and J. Sun, “Meta-SR: A Magnification-Arbitrary Network for Super-Resolution,” in *Proc. IEEE/CVF CVPR*, 2019, pp. 1575–1584. doi ·arXiv
38. Y. Chen, S. Liu and X. Wang, “Learning Continuous Image Representation with Local Implicit Image Function,” in *Proc. IEEE/CVF CVPR*, 2021, pp. 8624–8634. doi ·arXiv
39. Z. Hui, X. Gao, Y. Yang and X. Wang, “Lightweight Image Super-Resolution with Information Multi-distillation Network,” in *Proc. ACM International Conference on Multimedia*, 2019, pp. 2024–2032. doi ·arXiv
40. B. Ren, Y. Li, N. Mehta, R. Timofte et al., “The Ninth NTIRE 2024 Efficient Super-Resolution Challenge Report,” in *Proc. IEEE/CVF CVPR Workshops*, 2024. CVF ·arXiv
41. B. Ren, H. Guo, L. Sun, Z. Wu, R. Timofte, Y. Li et al., “The Tenth NTIRE 2025 Efficient Super-

Resolution Challenge Report," in *Proc. IEEE/CVF CVPR Workshops*, 2025. CVF ·arXiv
42. ITU-R, "Recommendation BT.601: Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide Screen 16:9 Aspect Ratios." ITU
43. MathWorks, "rgb2ycbcr: Convert RGB color values to YCbCr color space," Image Processing Toolbox documentation. link
44. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
45. Z. Chen, Z. Wu, E. Zamfir, K. Zhang, Y. Zhang, R. Timofte et al., "NTIRE 2024 Challenge on Image Super-Resolution (×4): Methods and Results," in *Proc. IEEE/CVF CVPR Workshops*, 2024, pp. 6108–6132. doi ·arXiv
46. Z. Chen, K. Liu, J. Gong, J. Wang, L. Sun, Z. Wu, R. Timofte, Y. Zhang et al., "NTIRE 2025 Challenge on Image Super-Resolution (×4): Methods and Results," in *Proc. IEEE/CVF CVPR Workshops*, 2025, pp. 1525–1535. doi ·arXiv
47. M. Bevilacqua, A. Roumy, C. Guillemot and M.-L. Alberi Morel, "Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding," in *Proc. BMVC*, 2012, pp. 135.1–135.10. doi
48. R. Zeyde, M. Elad and M. Protter, "On Single Image Scale-Up Using Sparse-Representations," in *Curves and Surfaces*, Lecture Notes in Computer Science, Springer, 2012, pp. 711–730. doi
49. D. Martin, C. Fowlkes, D. Tal and J. Malik, "A Database of Human Segmented Natural Images and Its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics," in *Proc. IEEE ICCV*, 2001, vol. 2, pp. 416–423. doi
50. J.-B. Huang, A. Singh and N. Ahuja, "Single Image Super-Resolution from Transformed Self-Exemplars," in *Proc. IEEE CVPR*, 2015, pp. 5197–5206. doi ·CVF
51. A. Fujimoto, T. Ogawa, K. Yamamoto, Y. Matsui, T. Yamasaki and K. Aizawa, "Manga109 Dataset and Creation of Metadata," in *Proc. 1st International Workshop on coMics ANalysis, Processing and Understanding* (*MANPU*), 2016, pp. 1–5. doi
52. Y. Matsui, K. Ito, Y. Aramaki, A. Fujimoto, T. Ogawa, T. Yamasaki and K. Aizawa, "Sketch-Based Manga Retrieval Using Manga109 Dataset," *Multimedia Tools and Applications*, vol. 76, no. 20, pp. 21811–21838, 2017. doi
53. E. Agustsson and R. Timofte, "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study," in *Proc. IEEE CVPR Workshops*, 2017, pp. 1122–1131. doi ·CVF
54. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," in *Proc. IEEE/CVF CVPR*, 2018, pp. 586–595. doi ·arXiv
55. W. Ren, R. Goyal, Z. Hu, T. T. Aumentado-Armstrong, I. Mohomed and A. Levinshtein, "Hallucination Score: Towards Mitigating Hallucinations in Generative Image Super-Resolution," arXiv:2507.14367, 2025. arXiv
56. B. B. Moser, A. S. Shanbhag, F. Raue, S. Frolov, S. Palacio and A. Dengel, "Diffusion Models, Image Super-Resolution, and Everything: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 7, pp. 11793–11813, 2025. doi ·arXiv
57. A. Shocher, N. Cohen and M. Irani, "'Zero-Shot' Super-Resolution Using Deep Internal Learning," in *Proc. IEEE/CVF CVPR*, 2018, pp. 3118–3126. doi ·arXiv

DLCV 09A.1・盲與真實世界影像超解析度

從估核、退化表示、合成資料與實拍配對，理解如何處理未知損傷，再比較生成先驗，以及沒有真值時的評估與保真限制。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/

你將學到

- 區分退化參數未知，以及真實世界退化家族不確定的情況。
- 比較顯式退化估計與隱式表示。
- 理解模糊核不匹配、影像內部統計與合成訓練管線。
- 追蹤生成先驗如何使用低解析度與語意條件。
- 在缺少乾淨參考時評估忠實度與感知品質。

先看全貌

DLCV 09A 假設已知退化，以固定縮圖方式訓練反向映射，再對照原 HR。真實照片沒有配方：鏡頭、失焦、手震、感光元件、光線，以及未知軟體的銳化、縮放與壓縮都會留下影響。盲與真實世界 SR 要在這些未知下仍然工作。

> **用最簡單的話說**
>
> 朋友傳來模糊小照片，要你放大變清楚。若知道它如何弄髒，就能針對那種損傷處理；但你不在現場。可以猜損傷再修、用許多自己弄壞的照片練習，或請懂照片的畫家補畫缺失。本章介紹這三種策略，也學習沒有乾淨副本時怎麼檢查成果。

手機變焦、老照片、網路圖與影片放大幾乎都面對未知退化。Bicubic 基準很強的模型，可能不適合真實 JPEG；真實 JPEG 上看起來很好的輸出，也可能虛構細節。家族分類幫助辨識假設、能力與失敗方式。

本章位置。 DLCV 09A 提供退化、感知／失真取捨及基準；DLCV 09B 介紹一般生成復原；DLCV 09 定義無參考指標。少步方法另見 DLCV 09A.2。主線解釋家族，論文頁補完整設計。程式只用 NumPy、SciPy、scikit-image、OpenCV 與 `skimage.data`，可在 CPU 快速執行。

問題：沒有人量過的損傷

白話說。退化未知時，必須由同一觀測推估清晰影像及其損傷過程。

簡述退化模型 DLCV 09A 的經典模型，將 LR 影像 y 寫為：

$$y = (x \circledast k)\downarrow_s + n,$$

x 是 HR，k 是鏡頭、失焦、運動、像素積分或抗混疊模糊核；$\circledast$ 是逐色彩通道二維卷積，$\downarrow_s$ 以倍率 s 降取樣，可取每個 $s \times s$ 區塊一點或 bicubic 縮放；n 是加性雜訊。真實管線還有品質 q 的

JPEG C_q，其位置可能改變，以及去馬賽克、去雜訊、銳化與色調處理。較完整但仍簡化的例子為 $y = C_q((x \circledast k)\downarrow_s + n)$。

盲與真實世界的意思　文獻用詞不完全一致，本章採以下區分：

- **非盲 SR**：測試時已知 k、s 及雜訊程度；固定 bicubic 是每張圖共用設定的特例。
- **盲 SR**：至少一項未知且隨影像改變；許多研究仍假設高斯類模糊與高斯雜訊，只是不知道參數。
- **真實世界 SR ／ Real-ISR**：退化不只未知，也可能超出模型家族，包括壓縮、相機處理與空間變化，通常沒有對應 HR 真值。

因此，在正確模型家族內估未知參數很準，不代表遇到錯誤退化家族仍能工作。

為什麼 bicubic 訓練會失效　**白話說**。只練習反解一種縮圖方式，就可能把另一種損傷當成同一件事來修。

DLCV 09A 的線性模型展示：銳利訓練、模糊測試會修不足，反之會過度銳化。Efrat 等人 [1] 說明正確模糊模型的重要性；無雜訊訓練也容易放大雜訊和 JPEG 區塊。SRMD [2] 指出：假設核比真核寬會振鈴，較窄則過度平滑，後面以小實驗驗證。

研究領域的地圖

白話說。先問退化是顯式表示或隱式學習，再問證據來自外部資料或輸入本身。

Liu 等人 [3] 依顯式／隱式與外部／單圖整理，指出當時單圖隱式格仍空缺；Chen 等人 [4] 分退化建模、實拍配對、領域轉換及自我學習。兩篇早於擴散時代，因此表中另補 2023 年起擴大的生成先驗家族 [5]。

家族	估計或學習什麼	代表方法	優點	常見失敗
顯式、給定核	測試直接提供核與雜訊	SRMD [2]、UDVD [6]、USRNet [7]	單模型處理多種已知退化，控制可解釋	需要可靠核，錯誤造成模糊或振鈴
顯式、外部資料估核	核代碼與 SR 交替估計	IKC [8]、DAN [9]、DCLS [10]	退化符合假設時準確	JPEG、相機處理與空間變化超出範圍
顯式、單圖統計	跨尺度重複區塊推估核	Michaeli–Irani [11]、KernelGAN [12]、ZSSR [13]	不需外部訓練，可逐圖調適	逐圖訓練慢，先驗弱且怕雜訊
退化表示	分辨退化的嵌入	DASR [14]、KDSR [15]	不需核標籤，快速前向	仍受訓練退化範圍限制
隱式、合成管線	廣泛隨機退化到清晰圖的映射	BSRGAN [16]、Real-ESRGAN [17]、Liang 等人的 DASR [18]	單模型處理多種照片	平滑、GAN 偽影與分布外退化
隱式、實拍配對	特定相機配對映射	RealSR [19]、DRealSR [20]、City100 [21]	真實資料	規模小、相機專屬、對齊誤差

家族	估計或學習什麼	代表方法	優點	常見失敗
隱式、未配對轉換	退化生成器或領域轉換器	CinCGAN [22]、Bulat 等人 [23]、FSSR [24]、Ji 等人 [25]	真 LR 不需 HR 配對	GAN 不穩、虛假紋理
生成先驗（2023–2026）	LR 控制的預訓練擴散先驗	StableSR [26]、DiffBIR [27]、PASD [28]、SeeSR [29]、SUPIR [30]、ResShift [31]	強退化下仍有自然紋理	虛構、低 PSNR、取樣慢與偏色

分類可重疊：DAN 顯式卻由外部資料學估計器；BSRGAN 稱隱式，訓練退化仍是手寫公式。另有兩篇 **DASR**：Wang 等人的對比退化表示 [14]，與 Liang 等人的高效率退化調適 [18]，比較時先確認指哪篇。

顯式退化建模

白話說。保留 $y = (x \circledast k)\downarrow_s + n$，把核和雜訊當旋鈕。可以由外部告知，也可以讓模型從影像估設定。

把核告訴網路　SRMD [2] 以 PCA 將 $p \times p$ 核壓成短向量，附加雜訊值，各數字展成 LR 大小常數圖，再與影像串接，學 $\hat{x} = F(y, k, \sigma)$；控制圖也可空間變化。UDVD [6] 依退化產生動態卷積；USRNet [7] 展開 MAP 求解器，交替使用已知核／倍率／雜訊的資料步與學得去雜訊先驗，服務多種核及倍率。其分裂背景見 DLCV 09。

這些仍是**非盲**模型：從反解固定退化，改為接收退化條件；前提是有人能提供可靠估計。

僅線上版：*SRMD：一個網路處理多種已知退化（CVPR 2018）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/srmd 一個網路處理多種已知退化 cvpr-2018/*

估計並修正核　**白話說**。錯核會留下特徵：太窄偏模糊，太寬偏振鈴。看輸出來辨識誤差，再修正核。

IKC [8] 先預測核代碼，以 SFTMD 做條件 SR，再讓 corrector 看目前 SR 與核代碼來修正。SFTMD 透過每個殘差區塊的空間特徵轉換注入條件 [32]。DAN [9] 將估計器與復原器展成共享權重的固定輪數交替；復原器以估計核而非真核訓練，學會容忍估計錯誤。DCLS [10] 則將估核及最小平方去模糊改在 LR 特徵空間進行。

經典模型下，交替最佳化寫成：

$$k^{(i+1)} = \arg\min_k \|y - (x^{(i)} \circledast k)\downarrow_s\|_1, \qquad x^{(i+1)} = \arg\min_x \|y - (x \circledast k^{(i+1)})\downarrow_s\|_1 + \phi(x),$$

i 是輪數，$x^{(i)}, k^{(i)}$ 是目前影像／核估計，ϕ 是影像先驗；DAN [9] 以學得模組取代兩個最小化步。

僅線上版：*IKC*：反覆修正模糊核（*CVPR 2019*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/ikc 反覆修正模糊核 cvpr-2019/*；*DAN*：展開交替最佳化（*NeurIPS 2020*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/dan 展開交替最佳化 neurips-2020/*

從影像本身估計核　白話說。內部模糊核估計利用跨尺度區塊重複性，尋找能保留相似區塊統計的下採樣核。

Michaeli–Irani [11] 據此做非參數核估計。KernelGAN [12] 只用測試圖訓練深層**線性**生成器，做 ×2 縮圖；判別器比較其 7×7 區塊與原圖，最後讀出核供 ZSSR ／ SRMD。ZSSR [13] 另以輸入產生 LR—更低解析度配對，逐圖訓練小 CNN。KernelGAN 原文約每圖一分鐘 GPU，ZSSR 還有額外訓練時間。

僅線上版：*KernelGAN*：以單圖 *GAN* 估計核（*NeurIPS 2019*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/kernelgan 以單圖-gan-估計核 neurips-2019/*

僅線上版：為什麼核難以識別 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/notes/為什麼核難以識別/*

核不匹配會怎麼樣　白話說。非盲求解器會認真反解你給的錯誤模糊。假設模糊不足留下柔軟影像，假設過強則把本來銳利的邊緣過度銳化成振鈴。

最簡單的非盲 ×2 實驗以共軛梯度求解：

$$\hat{x}_{\sigma_a} = \arg\min_x \| (x \circledast g_{\sigma_a}) \downarrow_2 - y \|_2^2 + \lambda \| \nabla x \|_2^2$$

g_{σ_a} 是假設寬度 σ_a 的高斯核，y 由真寬度 σ 產生，∇ 是梯度，以法方程式中的 Laplacian 實作；$\lambda = 2 \times 10^{-3}$ 控制平滑先驗。

```
import numpy as np
from scipy import ndimage as ndi
from scipy.sparse.linalg import LinearOperator, cg
from skimage import data, img_as_float64

s = 2                                                    # scale factor
x = img_as_float64(data.camera())[128:384, 128:384]      # HR test crop (256 x 256)
H, W = x.shape

def blur(z, sigma):                                      # circular Gaussian blur
 ↪  (self-adjoint)
    return ndi.gaussian_filter(z, sigma, mode="wrap")

def A(z, sigma):                                         # degradation: blur, then keep every
 ↪  s-th pixel
    return blur(z, sigma)[::s, ::s]

def At(r, sigma):                                        # its adjoint: zero-fill upsampling,
 ↪  then blur
    up = np.zeros((H, W)); up[::s, ::s] = r
```

```
    return blur(up, sigma)

def lap(z):                                              # discrete Laplacian (smoothness
↪  prior)
    return ndi.laplace(z, mode="wrap")

def nonblind_sr(y, sigma, lam=2e-3):
    """Solve min_x ||A_sigma x - y||^2 + lam ||grad x||^2 with conjugate gradients."""
    op = LinearOperator((H * W, H * W), dtype=float,
                        matvec=lambda v: (At(A(v.reshape(H, W), sigma), sigma) - lam *
                         ↪  lap(v.reshape(H, W))).ravel())
    sol, _ = cg(op, At(y, sigma).ravel(), maxiter=200)
    return sol.reshape(H, W)

def psnr(a, b):
    return 10 * np.log10(1.0 / np.mean((a - np.clip(b, 0, 1)) ** 2))

rng = np.random.default_rng(0)
assumed = [0.6, 1.0, 1.6, 2.2, 2.8]
print("true \\ assumed " + "".join(f"{a:7.1f}" for a in assumed))
for true in [1.0, 1.6, 2.2]:
    y = A(x, true) + 0.003 * rng.standard_normal((H // s, W // s))   # LR observation
    row = [psnr(x, nonblind_sr(y, a)) for a in assumed]
    print(f"sigma = {true:.1f}      " + "".join(f"{p:7.2f}" for p in row))
```

```
true \ assumed     0.6    1.0    1.6    2.2    2.8
sigma = 1.0        26.80  28.43  24.59  19.84  16.91
sigma = 1.6        24.22  25.30  27.23  24.13  19.83
sigma = 2.2        22.45  22.92  24.11  25.51  23.43
```

每列最高分在真核與假設核相符的對角線；損失並不對稱。真 $\sigma = 1.0$ 時，假設 2.8 損失約 **11.5 dB**，假設 0.6 損失 **1.6 dB**；過度去模糊放大雜訊與振鈴，修不足主要留模糊。圖 1 用真 $\sigma = 1.6$ 展示，正是 IKC [8] 想辨識的特徵。

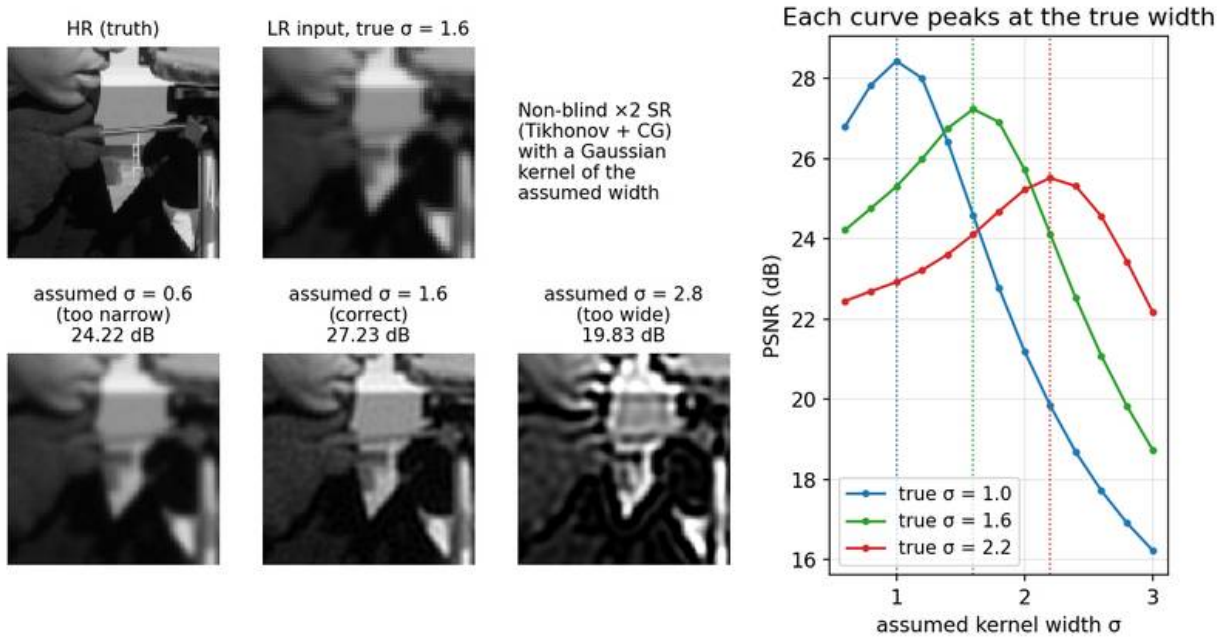


圖 *1* — 非盲 ×*2* 的核不匹配實驗。假設核較窄會留模糊，較寬則過度銳化與振鈴；*PSNR* 以完整 *256* × *256* 裁切計算。

退化表示學習

白話說。不一定要知道核的精確形狀，也可以學會分辨損傷種類，形成不需核標籤的緊湊指紋。

Wang 等人的 DASR [14] 假設同圖各處退化相同、跨圖不同：同圖兩區塊為正對，別圖為負對。編碼器搭配 MoCo 式負例佇列及 InfoNCE [33]，再以表示預測深度卷積權重與通道縮放。KDSR [15] 讓看得到 HR—LR 的教師教只看 LR 的學生，以涵蓋模糊、雜訊及 JPEG；Liang 等人的 DASR [18] 則估顯式參數，再混合多個卷積專家。

嵌入將「逐點估核」改為「辨認退化」，免核標籤，卻仍依賴訓練退化多樣性，不保證處理未見家族。

僅線上版：*DASR*：對比退化表示（*CVPR 2021*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/dasr 對比退化表示 cvpr-2021/*

用退化合成做隱式建模

白話說。不逐圖寫出損傷，而設計涵蓋廣泛情況的損傷生成器，製作大量配對，再訓練同一模型。

BSRGAN：打亂步驟　BSRGAN [16] 擴大模糊、降取樣與雜訊種類，也打亂順序：等向／非等向高斯、nearest ／ bilinear ／ bicubic ／先縮後放、Gaussian ／ JPEG ／反向再前向相機處理產

生的感光雜訊，最後再 JPEG。固定 bicubic 與經典退化成為特例。沿用 ESRGAN 式網路 [34]，先 L1 訓練 BSRNet，再加感知與對抗項成 BSRGAN。

Real-ESRGAN：重複步驟　Real-ESRGAN [17] 用**兩輪**退化模擬拍攝、編輯、重複壓縮縮放。每輪抽模糊、縮放、Gaussian ／ Poisson 雜訊與 JPEG；末端縮放和 sinc 視為一組，與 JPEG 隨機交換順序。有限 sinc 近似理想低通，可產生振鈴／過衝，教模型移除而非放大暈邊。另用 U-Net 式逐像素判別器 [35] 搭配頻譜正規化 [36] 穩定廣泛退化訓練。

僅線上版：逐步看高階退化管線 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/notes/逐步看高階退化管線/*；*Sinc* 濾波與振鈴 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/notes/sinc-濾波與振鈴/*

僅線上版：*BSRGAN*：實用的退化模型（*ICCV 2021*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/bsrgan 實用的退化模型 iccv-2021/*；*Real-ESRGAN*：純合成高階退化（*ICCVW 2021*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/real-esrgan 純合成高階退化 iccvw-2021/*

訓練配對如何產生　程式以兩輪模糊、隨機縮放、雜訊和 JPEG，再交換末端縮放／ sinc 與 JPEG，示範二階退化。部分範圍依原文 [17]，但縮放、光子雜訊及核家族有簡化；教學版有時略過整輪二階，官方只可能略過二階模糊。因此不是完整訓練配方重現。

```
import numpy as np, cv2
from scipy import ndimage as ndi
from scipy.special import j1
from skimage import data, img_as_float32

def gaussian_kernel(rng, size, smax):
    """Random isotropic or anisotropic Gaussian: k(c) ~ exp(-0.5 c^T S^-1 c), normalized."""
    s1, s2 = rng.uniform(0.2, smax, 2)
    if rng.random() < 0.5:
        s2 = s1                                                    # isotropic half of the
         ↪  time
    th = rng.uniform(0, np.pi)
    R = np.array([[np.cos(th), -np.sin(th)], [np.sin(th), np.cos(th)]])
    S = R @ np.diag([s1**2, s2**2]) @ R.T
    ax = np.arange(size) - size // 2
    C = np.stack(np.meshgrid(ax, ax, indexing="ij"), -1)
    k = np.exp(-0.5 * np.einsum("...i,ij,...j", C, np.linalg.inv(S), C))
    return k / k.sum(), f"gauss({s1:.1f},{s2:.1f})"

def sinc_kernel(size, wc):
    """2-D sinc low-pass: k(r) = wc J1(wc r) / (2 pi r), with k(0) = wc^2 / (4 pi)."""
    ax = np.arange(size) - size // 2
    r = np.hypot(*np.meshgrid(ax, ax, indexing="ij"))
    with np.errstate(invalid="ignore", divide="ignore"):
        k = np.where(r == 0, wc**2 / (4 * np.pi), wc * j1(wc * r) / (2 * np.pi * r))
    return k / k.sum()

def jpeg(img, q):
```

```
    ok, buf = cv2.imencode(".jpg", (np.clip(img, 0, 1) * 255).round().astype(np.uint8)[...,
    ↪  ::-1],
                           [cv2.IMWRITE_JPEG_QUALITY, int(q)])
    return cv2.imdecode(buf, cv2.IMREAD_COLOR)[..., ::-1].astype(np.float32) / 255

def one_round(img, rng, log, smax, nmax):
    """blur -> random resize -> noise -> JPEG: one 'classical' degradation."""
    size = int(rng.choice(np.arange(7, 22, 2)))
    if rng.random() < 0.1:
        k, name = sinc_kernel(size, rng.uniform(np.pi / 3, np.pi)), "sinc"
    else:
        k, name = gaussian_kernel(rng, size, smax)
    img = np.stack([ndi.convolve(img[..., c], k, mode="reflect") for c in range(3)], -1)
    f = rng.uniform(0.3, 1.2)                                          # resize factor
    ↪  (illustrative range)
    interp = rng.choice([cv2.INTER_AREA, cv2.INTER_LINEAR, cv2.INTER_CUBIC])
    img = cv2.resize(img, None, fx=f, fy=f, interpolation=interp)
    if rng.random() < 0.5:                                             # Gaussian noise, sigma
    ↪  in 1..nmax (8-bit units)
        sig = rng.uniform(1, nmax) / 255
        shape = img.shape[:2] + ((1,) if rng.random() < 0.4 else (3,))   # gray or color
        ↪  noise
        img = img + sig * rng.standard_normal(shape); noise = f"gauss{sig * 255:.0f}"
    else:                                                              # shot noise with a
    ↪  random photon count
        peak = 10 ** rng.uniform(1.5, 3)
        img = rng.poisson(np.clip(img, 0, 1) * peak) / peak; noise = f"poisson{peak:.0f}"
    q = rng.uniform(30, 95)
    img = jpeg(img, q)
    log.append(f"{name} x{f:.2f} {noise} q{q:.0f}")
    return img

def degrade(hr, rng, scale=4):
    """Second-order pipeline in the spirit of Real-ESRGAN (parameter ranges partly our
    ↪  own)."""
    log = []
    img = one_round(hr, rng, log, smax=3.0, nmax=30)
    if rng.random() < 0.8:
        img = one_round(img, rng, log, smax=1.5, nmax=25)          # second, milder round
    h, w = hr.shape[0] // scale, hr.shape[1] // scale
    def resize_sinc(z):
        z = cv2.resize(z, (w, h), interpolation=cv2.INTER_AREA)
        k = sinc_kernel(15, rng.uniform(np.pi / 3, np.pi))  # same kernel for all color
        ↪  channels
        return np.stack([ndi.convolve(z[..., c], k, mode="reflect") for c in range(3)], -1)
    final = [lambda z: jpeg(z, rng.uniform(30, 95)), resize_sinc]
    if rng.random() < 0.5:
        final = final[::-1]                                            # swap JPEG with the
        ↪  resize-and-sinc pair
    for op in final:
        img = op(img)
    return np.clip(img, 0, 1), log

hr = img_as_float32(data.astronaut())                                  # (512, 512, 3)
for seed in range(3):
```

```
    lr, log = degrade(hr, np.random.default_rng(seed))
    print(f"seed {seed}: LR {lr.shape} | " + " ; ".join(log))
```

```
seed 0: LR (128, 128, 3) | gauss(0.3,0.2) x0.85 poisson207 q64 ; gauss(0.9,0.9) x0.84 gauss5
 ↪  q90
seed 1: LR (128, 128, 3) | gauss(0.6,0.6) x1.04 gauss17 q71 ; gauss(1.4,0.6) x0.40 poisson42
 ↪  q40
seed 2: LR (128, 128, 3) | gauss(2.5,0.5) x0.47 gauss9 q67 ; gauss(0.3,1.1) x1.04 gauss19 q50
```

每行代表新 LR：`gauss(σ1,σ2)` 是兩軸寬度，另有縮放、雜訊與品質值。每次對同一 HR 重抽損傷，圖 2 的七張輸入遠比固定 bicubic 多樣。Real-ESRGAN [17] 在 GPU 合成，使用 **180** 組近期配對池避免同批共享縮放因素。



圖 *2* — 種子 *0–6* 從同一 *HR* 產生七組 *×4* 退化；單一模型必須處理所有情況。

後續改進　StableSR、SeeSR、SUPIR、ResShift 都用 Real-ESRGAN 類合成 [26]、[29]、[30]、[31]。但**範圍不一定比複雜度次要**：DiffBIR [27] 的前段復原器以範圍較廣的一階退化得到較佳無參考分數。LDL [37] 則從局部殘差統計建立偽影圖，專門懲罰 GAN 偽影，處理不同問題。

真實配對與未配對資料

白話說。可以實際拍出損傷：用不同焦段拍同場景，或在配對不可能時，從真實高低品質集合學習。

實拍配對 RealSR [19] 用不同焦段拍 DSLR，再對齊與校色；DRealSR [20] 擴至五種 DSLR；City100 [21] 用相機／手機改變焦段或距離研究視野取捨。DPED [38] 的同時手機／ DSLR 照片也常作無真值輸入。實拍可評估真實退化，卻仍規模小、相機專屬且有錯位，不能涵蓋所有網路照片。

未配對學習與領域轉換 CinCGAN [22] 先把真實 LR 轉乾淨 LR，再接預訓練 SR 並端到端調整。Bulat 等人 [23] 先學 HR→ 真實 LR 退化，再訓練復原；Lugmayr 等人 [39] 補回自然感光特性。DSGAN ／ FSSR [24] 將對抗項限制高頻，低頻另約束。Ji 等人 [25] 從目標領域估核與抽雜訊，贏得 NTIRE 2020 [40]。此類依賴資料分布，也可能產生嚴重 GAN 偽影 [3]。

用生成先驗處理真實世界 SR

白話說。大型文字擴散從廣泛影像學得先驗。2023 年起，許多復原方法保留大部分預訓練權重，再以小控制網路提供 LR，補出合理細節。

為什麼借用生成先驗 嚴重損傷時，LR 不再含所有 HR 細節。L2 迴歸學條件平均，可能平均不相容細節；生成模型則取一個合理答案。DLCV 09B 說明感知／失真，擴散概覽 與虛構問題 補背景。本章聚焦 SR 如何餵 LR、額外條件及保真控制。

本節使用 Real-ESRGAN 或類似合成。多數採 Stable Diffusion [41]，將 512 × 512 壓為 64 × 64 潛在網格再去雜訊；許多也調整 ControlNet [42]，用零初始化卷積把可訓練編碼分支連到凍結模型。

每種方法接收什麼條件

方法	先驗	LR 如何進入	額外條件	保真控制	原文步數
StableSR [26]	凍結 SD 2.1	時間編碼器、殘差區塊 SFT	空提示	解碼器特徵包覆權重 w	200 DDPM
DiffBIR [27]	凍結 SD 2.1	先復原，再餵 IRControlNet	負提示	區域自適應復原引導	50
PASD [28]	Stable Diffusion	去退化、像素感知交叉注意力	語意特徵與描述	雜訊排程	20
SeeSR [29]	凍結 SD 2	LR ControlNet、起始混入 LR	退化感知硬標籤及軟嵌入	—	50
SUPIR [30]	凍結 SDXL	強健編碼器、裁減 ControlNet 與 ZeroSFT	多模態描述、負品質提示	復原引導 τ_r	100 EDM
ResShift [31]	從頭訓練、VQGAN 潛在空間	起點靠近 LR、移動殘差	無	κ, p 排程	15

這些是原文預設，不代表等成本。SeeSR 補充資料 [29] 在同一 V100、128→512 比較：StableSR **18.70 秒**／ **200 步**，DiffBIR **5.85** ／ **50**，PASD **6.07** ／ **20**，SeeSR **7.24** ／ **50**，ResShift **1.12** ／ **15**，Real-ESRGAN **0.09 秒**。

三個反覆出現的設計：

- **條件強度隨雜訊變化**。StableSR 時間編碼器約在訊雜比 5×10^{-2} 最強，對應內容形成 [26]。
- **先移除損傷**。DiffBIR 先清理再生成，避免模型把雜訊／壓縮當內容；SUPIR 也調適編碼器增強退化強健性 [27]、[30]。
- **像素不足時加入語意**。PASD、SeeSR 從 LR 提取文字，SeeSR 讓乾淨／退化標籤一致；SUPIR 用多模態描述和使用者文字控制 [28]、[29]、[30]。

僅線上版：*StableSR*：以時間感知編碼器控制凍結先驗（*IJCV 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/stablesr 以時間感知編碼器控制凍結先驗 ijcv-2024/*；*DiffBIR*：先移除退化，再生成細節（*ECCV 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/diffbir 先移除退化再生成細節 eccv-2024/*；*SeeSR*：退化感知的語意提示（*CVPR 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/seesr 退化感知的語意提示 cvpr-2024/*；*SUPIR*：擴大先驗、資料與文字控制（*CVPR 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/blind-real-world-super-resolution/papers/supir 擴大先驗資料與文字控制 cvpr-2024/*

擴散 Transformer 與新先驗　白話說。生成先驗的去雜訊器從 U-Net 轉向 Transformer，SR 也跟著改變條件注入方式。

DreamClear [43] 使用 DiT、生成管線整理的一**百萬**高品質訓練圖及退化先驗專家混合。DiT4SR [44] 讓 LR 潛在串流持續參與注意力，再用跨串流卷積補局部細節。趨勢是更大先驗及更深入的條件交互。

保真、虛構與控制旋鈕　白話說。強先驗畫得可信，也可能多畫不存在的東西。因此需要「靠近輸入」與「自然外觀」的控制。

StableSR 的 $w \in [0, 1]$ 混合 LR 編碼特徵；DiffBIR 引導 s 在平坦區更強地拉向清理圖；SUPIR 的 τ_r 讓每步預測靠向 LR 潛在；ResShift 的 p 改殘差排程 [26]、[27]、[30]、[31]。指標未必單調，部分設定可同時改善多項。常見錯誤是把模糊船補成鳥等**語意錯誤**，以及天空／牆面的**虛構紋理**；小字小臉也是明列限制 [26]、[29]、[45]。

減少步數　白話說。每圖數十至數百次評估不利手機或影片。LR 已固定多數配置，若專門訓練，就可能從觀測附近以少步完成。

ResShift 的 15 步 [31] 是橋梁；SinSR [46] 蒸餾成一步，OSEDiff [45] 以 LoRA 和 VSD 學 LR→HR 潛在映射。詳細機制及計時見 DLCV 09A.2。

盲與真實世界的評估

白話說。無真值的照片無法直接算真實重建誤差。可測已知合成退化、預測人類評分或請人比較，每一種都只回答部分問題。

合成測試集的全參考指標　顯式方法常用 IKC 的 Gaussian8 [8]，或 KernelGAN 的 DIV2KRK [12]：DIV2K 驗證 **100** 張 [47]，每張隨機非等向受擾核。真實方法也測自家合成，例如 **3,000** 個

Real-ESRGAN 退化裁切 [26]、[29]、[45]，或 BSRGAN 的四種 DIV2K4D [16]。PSNR、SSIM [48] 測失真，LPIPS、DISTS [49]、[50] 測感知距離；同訓練管線的測試可能偏袒該模型。

RealSR ／ DRealSR 實拍也能算這些指標，但錯位會降低所有方法的 PSNR，並可能獎勵模糊；KernelGAN 也以此說明不採 NTIRE 2018 配對 [12]。

無參考指標　白話說。無參考（NR）指標不看原圖，預測外觀品質；它是人類意見的模型，仍可能被欺騙。

NIQE [51] 比較自然場景統計，越低越好；MUSIQ [52]、MANIQA [53] 是評分訓練模型，CLIP-IQA [54] 用 CLIP 比較好／壞照片提示，越高越好。定義見 DLCV 09。FID [55] 比較輸出與真實圖的特徵分布，不是逐圖正確性。

小實驗比較模糊、unsharp masking，以及模糊加細雜訊，以 PSNR 和刻意粗糙的 NR「清脆度」平均梯度評分。真正 NR 精細得多，但同樣評外觀而非真實內容。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, img_as_float64
from skimage.color import rgb2gray

hr = img_as_float64(rgb2gray(data.astronaut()))
rng = np.random.default_rng(0)

def psnr(a, b):
    return 10 * np.log10(1.0 / np.mean((a - np.clip(b, 0, 1)) ** 2))

def crispness(img):
    """A crude no-reference score: mean gradient magnitude (higher = 'sharper looking')."""
    return np.mean(np.hypot(ndi.sobel(img, 0), ndi.sobel(img, 1)))

smooth = ndi.gaussian_filter(hr, 1.5)                                  # a cautious, blurry
↪  restoration
candidates = {
    "smooth (posterior-mean-like)": smooth,
    "unsharp-masked x2": smooth + 2.0 * (smooth - ndi.gaussian_filter(smooth, 2)),
    "smooth + fake fine texture": smooth + 0.04 * rng.standard_normal(hr.shape),
}
for name, img in candidates.items():
    img = np.clip(img, 0, 1)
    print(f"{name:28s} PSNR {psnr(hr, img):6.2f} dB | crispness {crispness(img):.3f}")
print(f"{'ground truth':28s} PSNR    -     | crispness {crispness(hr):.3f}")
```

```
smooth (posterior-mean-like) PSNR  26.82 dB | crispness 0.170
unsharp-masked x2            PSNR  28.42 dB | crispness 0.259
smooth + fake fine texture   PSNR  24.48 dB | crispness 0.262
ground truth                 PSNR    -     | crispness 0.246
```

兩張「銳化」都比真值更清脆，虛假紋理最高、PSNR 最差。Unsharp 的 PSNR 改善是部分反解模糊；PSNR 也可能偏好稍模糊但位置正確，而非清楚卻偏移的紋理。

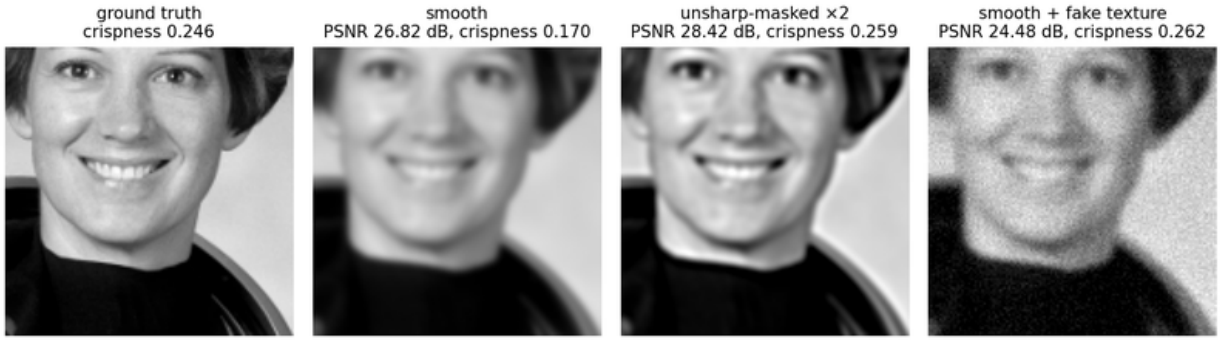


圖 *3* — 粗略無參考分數把銳化及假紋理評得比真值好，*PSNR* 則懲罰假紋理。真正指標較精密，公開實驗仍有類似張力。

為什麼排名不一致　原文表格也呈現相同張力：

- **無參考與外觀**。 BSRGAN 在 **20** 張 RealSRSet 的 NIQE **5.60**，比其他方法 **3.99–4.95** 差，作者卻認為視覺較好 [16]。
- **保真與寫實**。 StableSR 的 DIV2K 裁切 PSNR **23.26 dB**，低於部分 GAN 的 **24.62 dB**，FID ／ CLIPIQA ／ MUSIQ 卻較好 [26]。SUPIR 也呈現無參考強、PSNR ／ SSIM ／ LPIPS 弱的取捨 [30]。
- **指標彼此衝突**。 OSEDiff 表中多步 SeeSR ／ PASD 的 NIQE、MUSIQ、MANIQA 較強，OSEDiff 的 LPIPS、DISTS、FID、CLIPIQA 較強 [45]。
- **提示會推動分數**。負提示引導可提高 CLIPIQA、MUSIQ，過強又會銳化過度 [26]；可被取樣技巧提高的分數不是獨立真實性裁判。

至少在合成／實拍配對報一種全參考，真實圖報不同家族的兩種 NR，重要偏好主張另做人類研究。說明解析度及裁切；512 × 512 訓練模型在小裁切與大圖的表現可能不同 [26]。

人類研究與挑戰　人類研究測偏好，不判定虛構細節是否真有。StableSR 收集 **35** 人、**40** 圖、**1,400** 票 [26]；SeeSR 的 **20** 人研究中，誤認輸出為真值 **38.6%**，真實圖最佳偏好 **57.1%** [29]。方法匿名、隨機排序與足夠樣本等規則見 DLCV 09。

挑戰顯示領域從已知到未知退化的轉變：

- **NTIRE 2018：**一個 bicubic、三個未知模擬相機退化賽道，提供配對可學習 [56]。
- **NTIRE 2019：**實拍不同焦段配對，導入 RealSR [57]。
- **AIM 2019 ／ NTIRE 2020：**不提供配對，使用來源 LR 及部分賽道的未配對高品質圖；2020 依人類感知評比 [58]、[40]。
- **AIM 2020：** ×2 ／ ×3 ／ ×4 實拍，按 PSNR ／ SSIM [59]。
- **NTIRE 2025：** ×4 bicubic，並列感知與 PSNR 復原賽道 [60]。

失敗模式

白話說。一種錯誤是誤判損傷、修太多或太少；另一種是太相信先驗、補出不存在內容。

- **分布外退化：**合成資料未保證涵蓋未知損傷，Real-ESRGAN 也列出可能反而放大的限制 [17]。
- **錯核：**造成圖 1 的模糊／振鈴，單核假設也難處理空間變化。
- **GAN 偽影：**扭曲線條、氣泡紋理等，LDL 專門處理此問題 [17]、[16]、[37]。
- **語意虛構：**錯物件、平坦區假紋理，小字小臉尤其難 [29]、[26]。
- **偏色：** StableSR 以輸入通道平均／變異數或低頻小波帶校正 [26]。
- **分塊與解析度：**大圖可能有塊間不一致，原文聚合取樣用來減少它 [26]。
- **迎合指標：**銳化、顆粒與提示可能增分而不保真。

可用合理退化把 SR 再縮回 LR，比較輸入；文字和平坦區的大差異值得檢查。但算子本身未知，小殘差也不能證明正確，DLCV 09B 有更完整說明。

現代觀點

綜述論文　白話說。盲與真實 SR 的兩篇綜述建立用語，較新的擴散及廣義 SR 綜述再擴充它。

- **Liu 等人，Blind Image Super-Resolution: A Survey and Beyond，TPAMI** [3]：整理顯式外部資料、顯式單圖統計、隱式外部資料；回顧當時單圖隱式缺口、GAN 偽影與退化假設不一致的比較困難。
- **Chen 等人，Real-World Single Image Super-Resolution: A Brief Review，Information Fusion 2022** [4]：分退化建模、實拍配對、領域轉換及自學習，提醒合成基準可能高估真實照片能力。
- **Moser 等人，Diffusion Models, Image Super-Resolution, and Everything，TNNLS 2025** [5]：依輸入域、條件、引導與損傷空間整理擴散 SR，指出成本、偏色及可比較性問題。
- **Zhang 等人，Deep-Learning-Empowered Super Resolution: A Comprehensive Survey and Future Prospects，Proceedings of the IEEE 2025** [61]：涵蓋單圖、影片、立體、光場，按 CNN ／ Transformer ／ GAN ／擴散、資料及複雜度整理，適合作廣義索引。

2023–2026 年的研究方向

1. **生成先驗成為重要家族。** StableSR 至 SUPIR 以編碼器、ControlNet、SFT 控制凍結文字模型 [26]、[27]、[28]、[29]、[30]；DreamClear ／ DiT4SR 改用 Transformer [43]、[44]。
2. **語意作為條件：**退化標籤、多模態描述及文字控制 [29]、[30]。
3. **資料仍多為合成：** BSRGAN ／ Real-ESRGAN 類管線，SUPIR 報 **兩千萬**帶文字圖，DreamClear 一**百萬**整理的合成圖 [30]、[43]。
4. **速度成為工程目標：**從 200 步到 ResShift 的 15，再到 SinSR ／ OSEDiff 的一步 [31]、[46]、[45]，詳見 DLCV 09A.2。
5. **經典顯式盲 SR 持續發展：** DCLS、KDSR 改善估計與表示，仍以 Gaussian8 ／ DIV2KRK 評估 [10]、[15]。

開放問題

- **內容可信度：**指標難以穩定區分清楚正確和清楚虛構，已有多模態虛構評估提案 [62]。
- **管線外損傷：**空間變化、相機處理、反覆社群壓縮與影片編碼未完全涵蓋。
- **沒有真值的評估：** NR 彼此及與人類不一致，還可能被取樣技巧提高 [16]、[26]。
- **成本：**大型先驗、多步取樣與完整管線仍受裝置預算限制。
- **單圖隱式建模：**是綜述當時指出的缺口，不代表之後完全無人處理 [3]。

重點整理

- 註明未知的是退化參數、退化家族，或兩者皆是。
- 依顯式／隱式退化處理，以及外部／內部證據區分方法。
- 模糊核不匹配可能造成模糊或振鈴，應共同評估估計器與重建。
- 合成退化擴大訓練涵蓋範圍，卻不保證能處理未見損傷。
- 生成先驗能提供逼真紋理，也可能捏造缺乏支持的細節。
- 結合有參考測試、多類無參考指標與使用者研究，並預期不同排名。

練習

1. **核誤差的不對稱**。將實驗雜訊設零，再把 λ 增到 10^{-2}，各自重做表格。假設核太寬的懲罰如何改變？

   提示

   太寬會放大原本沒有被衰減的頻率。去雜訊後右側損失減少、仍有振鈴；加大平滑權重抑制放大，使兩側較平、最高分也可能下降。比較距對角線相同步數的差。

2. **打亂或重複**。改 `degrade` 支援 BSRGAN 式：等向／非等向模糊、×4 降取樣、雜訊、JPEG 用 `rng.permutation` 打亂，再最後 JPEG。對 **50** 組隨機 LR 做 bicubic 放大，與 HR 算平均 PSNR，哪一種更難？

   提示

   降取樣若先於模糊／雜訊，會在 LR 尺度模糊，雜訊也未經縮放平滑。應報平均及散布；散布有助理解退化空間涵蓋。

3. **閱讀無參考分數**。新增真值 JPEG 品質 **50** 及位移一像素兩候選，按 PSNR ／清脆度排序。哪個顯示 PSNR 懲罰了肉眼可能不在意的變化？

   提示

   一像素位移看起來近似，PSNR 可能大降；JPEG 降低部分清脆度，也可能因區塊邊提高它。實拍錯位因此可能偏袒較模糊方法。

4. **選擇家族**。為下列情境各選一家族並解釋：(a) 1990 年代掃描家庭照做相簿；(b) 顯微影像數細胞；(c) 手機 ×2 變焦須 **30 ms**；(d) 可拍校正配對的固定監視器。

提示

(a) 適度控制的生成先驗可處理混合損傷，但須檢查身分與細節。(b) 優先量測點擴散函數及配對復原，不能虛構計數。(c) 先評估小型回歸／GAN；短擴散仍須實測完整裝置成本。(d) 用該相機實拍配對，必要時結合量測核。

5. **KernelGAN 的 ×4 核。** $k_4 = k_2 \circledast k_2^{\uparrow}$，$k_2^{\uparrow}$ 在係數間插零。用一維高斯驗證兩次 ×2 等於一次此核 ×4，並求 k_2 寬度 σ 對應的 k_4 寬度。

提示

將第二次卷積移到第一次抽取前，核要膨脹兩倍：兩高斯寬度平方相加，得 $\sqrt{\sigma^2 + (2\sigma)^2} = \sqrt{5}\sigma$。可用長度 **256** 隨機訊號與 `np.convolve` 測試。

參考文獻

1. N. Efrat, D. Glasner, A. Apartsin, B. Nadler and A. Levin, "Accurate Blur Models vs. Image Priors in Single Image Super-Resolution," in *Proc. IEEE ICCV*, 2013, pp. 2832–2839. doi ·CVF
2. K. Zhang, W. Zuo and L. Zhang, "Learning a Single Convolutional Super-Resolution Network for Multiple Degradations," *CVPR*, 2018. paper
3. A. Liu, Y. Liu, J. Gu, Y. Qiao, C. Dong, "Blind Image Super-Resolution: A Survey and Beyond," *IEEE TPAMI* (*early access*), 2022. source
4. Honggang Chen, Xiaohai He, Linbo Qing, Yuanyuan Wu, Chao Ren, Ray E. Sheriff, Ce Zhu, "Real-world single image super-resolution: A brief review," *Information Fusion*, 2022. source
5. B. B. Moser, A. S. Shanbhag, F. Raue, S. Frolov, S. Palacio and A. Dengel, "Diffusion Models, Image Super-Resolution, and Everything: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 7, pp. 11793–11813, 2025. doi ·arXiv
6. Y. Xu, S. R. Tseng, Y. Tseng, H. Kuo, Y. Tsai, "Unified Dynamic Convolutional Network for Super-Resolution with Variational Degradations," *CVPR*, 2020. source
7. K. Zhang, L. Van Gool, R. Timofte, "Deep Unfolding Network for Image Super-Resolution," *CVPR*, 2020. source
8. J. Gu, H. Lu, W. Zuo and C. Dong, "Blind Super-Resolution With Iterative Kernel Correction," *CVPR*, 2019. paper
9. Z. Luo, Y. Huang, S. Li, L. Wang and T. Tan, "Unfolding the Alternating Optimization for Blind Super Resolution," *NeurIPS*, 2020. paper
10. Z. Luo, H. Huang, L. Yu, Y. Li, H. Fan and S. Liu, "Deep Constrained Least Squares for Blind Image Super-Resolution," *CVPR*, 2022. paper
11. T. Michaeli, M. Irani, "Nonparametric Blind Super-resolution," *Proceedings of the IEEE International Conference on Computer Vision*, 2013. source
12. S. Bell-Kligler, A. Shocher and M. Irani, "Blind Super-Resolution Kernel Estimation using an Internal-GAN," *NeurIPS*, 2019. paper
13. A. Shocher, N. Cohen and M. Irani, " 'Zero-Shot' Super-Resolution Using Deep Internal Learning," in *Proc. IEEE/CVF CVPR*, 2018, pp. 3118–3126. doi ·arXiv
14. L. Wang, Y. Wang, X. Dong, Q. Xu, J. Yang, W. An and Y. Guo, "Unsupervised Degradation Representation Learning for Blind Super-Resolution," *CVPR*, 2021. paper

15. B. Xia, Y. Zhang, Y. Wang, Y. Tian, W. Yang, R. Timofte and L. Van Gool, “Knowledge Distillation Based Degradation Estimation for Blind Super-Resolution,” *ICLR*, 2023. paper
16. K. Zhang, J. Liang, L. Van Gool, R. Timofte, “Designing a Practical Degradation Model for Deep Blind Image Super-Resolution,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021. source
17. X. Wang, L. Xie, C. Dong, Y. Shan, “Real-ESRGAN: Training Real-World Blind Super-Resolution With Pure Synthetic Data,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021. source
18. J. Liang, H. Zeng, L. Zhang, “Efficient and Degradation-Adaptive Network for Real-World Image Super-Resolution,” *ECCV*, 2022. source
19. J. Cai, H. Zeng, H. Yong, Z. Cao, L. Zhang, “Toward Real-World Single Image Super-Resolution: A New Benchmark and a New Model,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. source
20. P. Wei, Z. Xie, H. Lu, Z. Zhan, Q. Ye, W. Zuo, L. Lin, “Component Divide-and-Conquer for Real-World Image Super-Resolution,” *ECCV*, 2020. source
21. C. Chen, Z. Xiong, X. Tian, Z. Zha, F. Wu, “Camera Lens Super-Resolution,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019. source
22. Y. Yuan, S. Liu, J. Zhang, Y. Zhang, C. Dong, L. Lin, “Unsupervised Image Super-Resolution using Cycle-in-Cycle Generative Adversarial Networks,” *CVPR Workshops*, 2018. source
23. A. Bulat, J. Yang, G. Tzimiropoulos, “To learn image super-resolution, use a GAN to learn how to do image degradation first,” *ECCV*, 2018. source
24. M. Fritsche, S. Gu, R. Timofte, “Frequency Separation for Real-World Super-Resolution,” *ICCV Workshops*, 2019. source
25. X. Ji, Y. Cao, Y. Tai, C. Wang, J. Li, F. Huang, “Real-World Super-Resolution via Kernel Estimation and Noise Injection,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020. source
26. J. Wang, Z. Yue, S. Zhou, K. C. K. Chan, C. C. Loy, “Exploiting Diffusion Prior for Real-World Image Super-Resolution,” *IJCV*, 2024. source
27. X. Lin, J. He, Z. Chen, et al., “DiffBIR: Towards Blind Image Restoration with Generative Diffusion Prior,” *ECCV*, 2024. source
28. T. Yang, R. Wu, P. Ren, X. Xie, L. Zhang, “Pixel-Aware Stable Diffusion for Realistic Image Super-resolution and Personalized Stylization,” *ECCV*, 2024. source
29. R. Wu, T. Yang, L. Sun, Z. Zhang, S. Li, L. Zhang, “SeeSR: Towards Semantics-Aware Real-World Image Super-Resolution,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. source ·supplementary experiments
30. F. Yu, J. Gu, Z. Li, et al., “Scaling Up to Excellence: Practicing Model Scaling for Photo-Realistic Image Restoration In the Wild,” *CVPR*, 2024. source
31. Z. Yue, J. Wang, C. C. Loy, “ResShift: Efficient Diffusion Model for Image Super-resolution by Residual Shifting,” *NeurIPS*, 2023. source
32. X. Wang, K. Yu, C. Dong, C. C. Loy, “Recovering Realistic Texture in Image Super-Resolution by Deep Spatial Feature Transform,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018. source
33. K. He, H. Fan, Y. Wu, S. Xie, R. Girshick, “Momentum Contrast for Unsupervised Visual Representation Learning,” *CVPR*, 2020. source

34. X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao and C. C. Loy, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” in *Proc. ECCV Workshops*, 2018 (Lecture Notes in Computer Science, 2019), pp. 63–79. doi ·arXiv
35. O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Proc. MICCAI*, 2015, pp. 234–241. doi ·arXiv
36. T. Miyato, T. Kataoka, M. Koyama, Y. Yoshida, “Spectral Normalization for Generative Adversarial Networks,” *ICLR*, 2018. source
37. J. Liang, H. Zeng, L. Zhang, “Details or Artifacts: A Locally Discriminative Learning Approach to Realistic Image Super-Resolution,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. source
38. A. Ignatov, N. Kobyshev, R. Timofte, K. Vanhoey, L. Van Gool, “DSLR-Quality Photos on Mobile Devices with Deep Convolutional Networks,” *ICCV*, 2017. source
39. A. Lugmayr, M. Danelljan, R. Timofte, “Unsupervised Learning for Real-World Super-Resolution,” *arXiv preprint*, 2019. source
40. A. Lugmayr, M. Danelljan, R. Timofte, et al., “NTIRE 2020 Challenge on Real-World Image Super-Resolution: Methods and Results,” *CVPR Workshops*, 2020. source
41. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” *CVPR*, 2022. source
42. L. Zhang, A. Rao, M. Agrawala, “Adding Conditional Control to Text-to-Image Diffusion Models,” *ICCV*, 2023. source
43. Y. Ai, X. Zhou, H. Huang, X. Han, Z. Chen, Q. You, H. Yang, “DreamClear: High-Capacity Real-World Image Restoration with Privacy-Safe Dataset Curation,” *NeurIPS*, 2024. source
44. Z. Duan, J. Zhang, X. Jin, et al., “DiT4SR: Taming Diffusion Transformer for Real-World Image Super-Resolution,” *ICCV*, 2025. source
45. R. Wu, L. Sun, Z. Ma, L. Zhang, “One-Step Effective Diffusion Network for Real-World Image Super-Resolution,” *NeurIPS*, 2024. source
46. Y. Wang, W. Yang, X. Chen, et al., “SinSR: Diffusion-Based Image Super-Resolution in a Single Step,” *CVPR*, 2024. source
47. E. Agustsson and R. Timofte, “NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study,” in *Proc. IEEE CVPR Workshops*, 2017, pp. 1122–1131. doi ·CVF
48. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
49. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” in *Proc. IEEE/CVF CVPR*, 2018, pp. 586–595. doi ·arXiv
50. K. Ding, K. Ma, S. Wang, E. P. Simoncelli, “Image Quality Assessment: Unifying Structure and Texture Similarity,” *IEEE TPAMI*, 2022. source
51. A. Mittal, R. Soundararajan and A. C. Bovik, “Making a “Completely Blind” Image Quality Analyzer,” *IEEE Signal Processing Letters*, vol. 20, no. 3, pp. 209–212, 2013. doi
52. J. Ke, Q. Wang, Y. Wang, P. Milanfar, F. Yang, “MUSIQ: Multi-scale Image Quality Transformer,” *ICCV*, 2021. source
53. S. Yang, T. Wu, S. Shi, S. Lao, Y. Gong, M. Cao, J. Wang, Y. Yang, “MANIQA: Multi-dimension Attention Network for No-Reference Image Quality Assessment,” *CVPR Workshops*, 2022. source

54. J. Wang, K. C. K. Chan, C. C. Loy, “Exploring CLIP for Assessing the Look and Feel of Images,” *AAAI*, 2023. source
55. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium,” *NeurIPS*, 2017. source
56. R. Timofte, S. Gu, J. Wu, L. Van Gool, “NTIRE 2018 Challenge on Single Image Super-Resolution: Methods and Results,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018. source
57. J. Cai, S. Gu, R. Timofte, L. Zhang, “NTIRE 2019 Challenge on Real Image Super-Resolution: Methods and Results,” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019. source
58. A. Lugmayr, M. Danelljan, R. Timofte, et al., “AIM 2019 Challenge on Real-World Image Super-Resolution: Methods and Results,” *ICCV Workshops*, 2019. source
59. P. Wei, H. Lu, R. Timofte, et al., “AIM 2020 Challenge on Real Image Super-Resolution: Methods and Results,” *ECCV Workshops*, 2020. source
60. Z. Chen, K. Liu, J. Gong, J. Wang, L. Sun, Z. Wu, R. Timofte, Y. Zhang et al., “NTIRE 2025 Challenge on Image Super-Resolution (×4): Methods and Results,” in *Proc. IEEE/CVF CVPR Workshops*, 2025, pp. 1525–1535. doi ·arXiv
61. Le Zhang, Ao Li, Qibin Hou, Ce Zhu, Yonina C. Eldar, “Deep-Learning-Empowered Super Resolution: A Comprehensive Survey and Future Prospects,” *Proceedings of the IEEE*, 2025. source
62. W. Ren, R. Goyal, Z. Hu, T. T. Aumentado-Armstrong, I. Mohomed and A. Levinshtein, “Hallucination Score: Towards Mitigating Hallucinations in Generative Image Super-Resolution,” arXiv:2507.14367, 2025. arXiv
63. Real-ESRGAN contributors, “Real-ESRGAN model and ×4-plus training configuration,” official source, accessed 2026-11-02. model ·configuration
64. D. P. Kingma, J. Ba, “Adam: A Method for Stochastic Optimization,” *ICLR*, 2015. source
65. M. Bevilacqua, A. Roumy, C. Guillemot and M.-L. Alberi Morel, “Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding,” in *Proc. BMVC*, 2012, pp. 135.1–135.10. doi
66. R. Zeyde, M. Elad and M. Protter, “On Single Image Scale-Up Using Sparse-Representations,” in *Curves and Surfaces*, Lecture Notes in Computer Science, Springer, 2012, pp. 711–730. doi
67. D. Martin, C. Fowlkes, D. Tal and J. Malik, “A Database of Human Segmented Natural Images and Its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics,” in *Proc. IEEE ICCV*, 2001, vol. 2, pp. 416–423. doi
68. J.-B. Huang, A. Singh and N. Ahuja, “Single Image Super-Resolution from Transformed Self-Exemplars,” in *Proc. IEEE CVPR*, 2015, pp. 5197–5206. doi ·CVF
69. J. Kim, J. K. Lee and K. M. Lee, “Accurate Image Super-Resolution Using Very Deep Convolutional Networks,” in *Proc. IEEE CVPR*, 2016, pp. 1646–1654. doi ·arXiv
70. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” in *Proc. IEEE CVPR*, 2017, pp. 105–114. doi ·arXiv
71. R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang et al., “NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results,” in *Proc. IEEE CVPR Workshops*, 2017, pp. 1110–1121. doi ·CVF
72. Y. Matsui, K. Ito, Y. Aramaki, A. Fujimoto, T. Ogawa, T. Yamasaki and K. Aizawa, “Sketch-Based

Manga Retrieval Using Manga109 Dataset," *Multimedia Tools and Applications*, vol. 76, no. 20, pp. 21811–21838, 2017. doi
73. KernelGAN contributors, "Adversarial losses and kernel constraints," official source, accessed 2026-11-02. loss definitions ·alternating updates
74. X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, S. P. Smolley, "Least Squares Generative Adversarial Networks," *ICCV*, 2017. source
75. DASR contributors, "MoCo logits and contrastive cross-entropy implementation," official DASR source, commit `51524af8021c94a502ee002e135867a1b75dc7d0`, accessed 2026-11-02. encoder ·loss
76. Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong and Y. Fu, "Image Super-Resolution Using Very Deep Residual Channel Attention Networks," in *Proc. ECCV*, 2018, pp. 294–310. doi ·arXiv
77. K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proc. ICLR*, 2015. arXiv
78. Y. Zhang, X. Huang, J. Ma, et al., "Recognize Anything: A Strong Image Tagging Model," *CVPR Workshops*, 2024. source
79. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," *ICLR*, 2022. source
80. D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, R. Rombach, "SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis," *ICLR*, 2024. source
81. H. Liu, C. Li, Q. Wu, Y. J. Lee, "Visual Instruction Tuning," *NeurIPS*, 2023. source
82. T. Karras, M. Aittala, T. Aila, S. Laine, "Elucidating the Design Space of Diffusion-Based Generative Models," *NeurIPS*, 2022. source

DLCV 09A.2・單步擴散超解析度

理解 LR 潛在起點、殘差位移蒸餾、分數正規化與可調單步超解析度，同時檢查保真、速度與虛構細節的限制。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/

你將學到

- 區分依觀測條件化的復原與從雜訊開始的生成。
- 比較復原鏈蒸餾與預訓練先驗調適。
- 追蹤潛在起點、LoRA、提示與分數正規化的作用。
- 區分退化調適、反演與忠實度／細節控制。
- 評估缺失細節的多解性與完整部署成本。

先看全貌

> **用最簡單的話說**
>
> 從故事畫出整個場景，好比從零生成影像；放大小照片，則像在已經標出物件位置的紙上作畫。從這張紙開始可以省時間，但小圖沒有告訴你每片葉子或每個字，所以快速畫家仍可能補錯細節。畫得快，不會讓猜測變得更確定。

真實低解析度影像已含低頻結構、配置與部分語意。從其潛在表示開始，可避免花許多取樣步重建已有結構；模型仍須去除損傷、補足細節。方法可以是學得的單步映射、蒸餾的影像條件流，或以重建與分布目標調整預訓練先驗。

先讀 DLCV 09A：非盲影像超解析度 理解反問題，再讀 DLCV 09A.1：盲與真實世界超解析度 理解退化合成與評估。DLCV 12B：單步生成 說明端點蒸餾、一致性及分數分布匹配。本章加入**特定觀測**的限制：外觀合理之外，還須關心內容保真。

第一次先看家族表、量測歧義實驗及 OSEDiff 概覽。完整訓練與控制比較放在選讀論文頁。選擇取決於損傷、可接受的虛構細節及完整部署預算，沒有對每張圖都最佳的方法。

單步超解析度必須做到什麼

白話說。輸入提供起點，卻未揭露終點的每個細節。復原器要利用證據，也要承認未觀測細節有歧義。

觀測寫為 $y = \mathcal{D}(x) + n$：x 是未知高解析度影像，y 是低解析度觀測，$\mathcal{D}$ 是退化算子，n 是雜訊。真實退化通常未知且可能非線性。條件生成器輸出 $\hat{x} = G_\theta(y)$；若另有隨機輸入 z，則為 $G_\theta(y, z)$。

確定性模型可以每次輸出相同圖，仍然補出錯誤細節；隨機模型能產生多個合理答案，也不知道哪個才是真正拍下的原圖。兩者都不自動保真。觀測限制一部分內容，先驗則為剩餘部分提供偏好。

精確的歧義實驗 令 $A(x)$ 對每個 2×2 區塊取平均，形成 ×2 降取樣。值為 $+a, -a, -a, +a$ 的棋盤紋在區塊內總和為零，因此加上或減掉它都不改變 LR 觀測。這種紋理位於 A 的零空間。

以下合成平滑 LR 圖及正反兩種 HR 紋理，比較條件平均與合理模態。此處刻意已知量測模型；真實照片還須估計退化，檢查一致性更難。

```
import numpy as np

row, col = np.mgrid[:32, :32]
lr = 0.5 + 0.12*np.sin(2*np.pi*col/16) + 0.08*np.cos(2*np.pi*row/13)
base = lr.repeat(2, 0).repeat(2, 1)
i, j = np.indices(base.shape)
texture = 0.12*(-1.)**(i+j)  # zero mean in every 2x2 block
truth = base + texture
candidates = {'conditional mean': base, 'matching mode': base+texture, 'opposite mode':
 ↪  base-texture}
def measure(x):
    return x.reshape(32,2,32,2).mean((1,3))
for name, x in candidates.items():
    mse = np.mean((x-truth)**2)
    discrepancy = np.max(np.abs(measure(x)-lr))
    print(f'{name:16s} MSE {mse:.4f} | LR discrepancy {discrepancy:.1e}')
```

```
conditional mean MSE 0.0144 | LR discrepancy 0.0e+00
matching mode    MSE 0.0000 | LR discrepancy 5.6e-17
opposite mode    MSE 0.0576 | LR discrepancy 5.6e-17
```

實際結果顯示所有候選都在浮點誤差內重現 LR；但相對選定真值，反向清晰模態的平方誤差是條件平均的四倍。若隱藏正負紋理等機率，條件平均最小化期望平方誤差；抽取合理模態不保證抽中原圖。加速無法識別從未量測的位元。

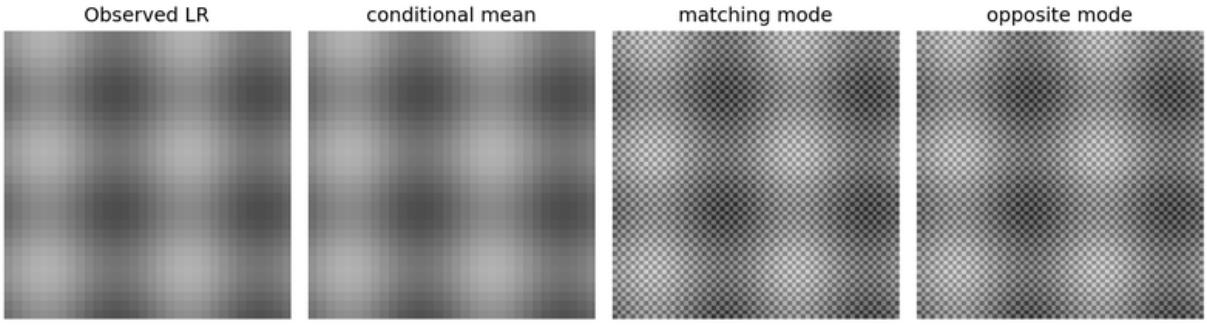


圖 *1* — 符合量測，比正確重建隱藏細節更弱。確定性單步網路和多步取樣器一樣，可能選到錯誤卻清晰的紋理。

哪些工作從推論移到訓練 多步復原反覆呼叫去雜訊網路；單步模型須學較大的影像條件轉換，可能使用調適、教師取樣、判別器或分數正規化。部署呼叫減少，仍可能換來昂貴的訓練機制，應兩邊一起報告。

預訓練文字生成去雜訊器，不會自動把 LR 潛在向量視為合法高雜訊輸入。OSEDiff [1] 明確調適這種影像到影像用法；擴散網路提供預訓練參數化，並不證明輸入符合原前向加雜訊分布。

短鏈與單步超解析度的地圖

白話說。有些縮短復原專用鏈，有些調整大型先驗，有些學起點或提供分離控制。先辨識改的是哪個元件，再比較方法名稱。

家族	起點	主要訓練訊號	方法例子	應檢查什麼
殘差位移鏈	以 LR 為中心的含雜訊狀態	影像條件擴散重建	ResShift [2]	排程、步數、訓練領域
條件端點蒸餾	LR 加初始擾動	教師端點、反向與真值循環	SinSR [3]	教師程序與學生實際映射
預訓練先驗調適	LR 潛在向量	配對重建與潛在分數正規化	OSEDiff [1]	編解碼調適、提示成本與分數追蹤
退化感知調適	LR 潛在向量加退化估計	重建、感知與對抗損失	S3Diff [4]	退化估計器與引導評估次數
對抗蒸餾	影像條件短取樣器	教師與判別器	AddSR [5]	訓練版本及推論預算
目標感知分數	LR 條件學生	配對目標分數與分布感知狀態	TSD-SR [6]	HR 目標如何進入訓練
學得起點／反演	預測的影像條件吵狀態	預訓練反向程序後的損失	InvSR [7]	起點預測器與總呼叫數
分離增強控制	LR 潛在向量	像素與語意 LoRA 目標	PiSA-SR [8]	調整哪些權重、保真是否單調改變

這是功能地圖，分類可以重疊；預訓練生成器、學得起點及對抗目標可同時出現。某方法支援多種預算時，四步成績仍須標四步。

盲超解析度及擴散超解析度綜述 [9]、[10] 提供更廣的分類。退化假設依然重要；快速推論不會讓某種合成退化訓練，自動適用所有相機、壓縮或歷史照片。

以觀測為中心的短鏈

白話說。讓高解析度影像逐漸移向低解析度觀測而非純雜訊，使反向程序從已觀測結構附近開始。

ResShift [2] 定義放大後 LR y 與 HR 目標 x_0 的殘差 $e = y - x_0$，前向邊際可寫成：

$$q(x_t \mid x_0, y) = \mathcal{N}\big(x_0 + \eta_t(y - x_0), \kappa^2 \eta_t I\big)\,.$$

η_t 從近零增加到一，κ 控制雜訊，I 是單位共變異數。y 已放大，或兩張圖已編碼至相容網格，所以維度一致。最後狀態近似以 LR 為中心的高斯雜訊，再反向補回細節。

網路從吵狀態、觀測及時間預測 HR，因此訓練的是復原條件程序。原潛在實作使用自動編碼器和短鏈；這不同於凍結文字生成先驗再加控制器。容量、資料與表示都不同，不能把跨方法分數差只歸因於鏈長。

SinSR [3] 把 ResShift 的反向程序改為確定性，再蒸餾端點為一次學生預測，並訓練反向映射與真值循環。它不是保持目標不變、只讓 ResShift 跑一次。

確定性教師為每組 LR 中心的隨機起點指定終點，使端點迴歸明確，隨機起點仍可選擇不同結果。真值一致性另提供超越模仿教師的配對訊號。同一網路兼做反向／復原是訓練設計，推論只需復原方向。

僅線上版：*ResShift*：復原條件的短鏈（*NeurIPS 2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/resshift 復原條件的短鏈 neurips-2023/*；*SinSR*：蒸餾確定性的影像條件映射（*CVPR 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/sinsr 蒸餾確定性的影像條件映射 cvpr-2024/*

從 LR 潛在向量開始：OSEDiff

白話說。 OSEDiff 用一次網路呼叫，把輸入表示轉成高解析度潛在表示；另以訓練正規化鼓勵自然輸出。

部署時的路徑 模型含調適影像編碼器 E_θ、調適去雜訊器 ϵ_θ、凍結解碼器 D 與提示擷取器 Y [1]。先把輸入放大至工作解析度，再寫為：

$$z_L = E_\theta(y), \quad c = Y(y), \quad \hat{z}_H = \frac{z_L - \beta_T \epsilon_\theta(z_L, T, c)}{\alpha_T}, \quad \hat{x}_H = D(\hat{z}_H).$$

z_L 是 LR 編碼的潛在向量，c 是文字條件，T 是固定擴散時間，α_T, β_T 沿用原文訊號／雜訊係數。公式**沒有**向 z_L 新增高斯雜訊；它是特定時間下學得的影像轉換，並不假設退化潛在向量真的來自原高雜訊前向分布。

LoRA [11] 對編碼器與去雜訊器加入低秩更新。凍結解碼器維持預訓練潛在空間的輸出意義。少量可訓練參數只代表調適大小，不是整個模型的參數或記憶體成本。

配對保真與學得的先驗 OSEDiff 結合像素 MSE、LPIPS 配對重建與變分分數蒸餾（VSD）[12]、[13]。前兩者對照 HR 訓練目標；分數項則用固定先驗去雜訊器和追蹤生成樣本的第二個去雜訊器，比較生成潛在分布。

這是 DLCV 12B 分布匹配的復原版本，沒有直接計算未知真實退化的似然。合成配對教模型保留內容，先驗教自然外觀，兩者都可能不適合某張分布外影像。

為什麼在潛在空間正規化 像素空間先驗梯度可能反覆解碼、再編碼生成影像。OSEDiff 直接作用於預測潛在向量，近似這段往返以節省成本；它仰賴自動編碼器表示足夠一致，不是任意潛在向量都成立的精確恆等式。

輔助正規化網路只在訓練使用。部署仍含影像編碼、提示擷取、一次去雜訊及解碼。計時要明示包含哪些步驟；改用大型描述模型，可能吃掉原本單次去雜訊省下的時間。

僅線上版：*OSEDiff*：*LR* 潛在起點與變分分數正規化（*NeurIPS 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/osediffl r-潛在起點與變分分數正規化 neurips-2024/*

依損傷調整模型

白話說。輕微模糊與強烈雜訊不適合完全相同的修正。S3Diff [4] 依退化估計調整低秩參數。

對 $W \in \mathbb{R}^{m\times n}$，一般 LoRA 為 $W' = W + AB$，其中 $A \in \mathbb{R}^{m\times r}$、$B \in \mathbb{R}^{r\times n}$，$r$ 是小秩。S3Diff 再加入退化條件中間矩陣：

$$W_i' = W_i + A_i C_i(d) B_i.$$

i 表示網路區塊，d 含估計的模糊／雜訊資訊，$C_i(d) \in \mathbb{R}^{r\times r}$ 調制更新。共用映射結合退化嵌入與區塊識別碼，產生區塊別修正，使不同影像使用不同有效權重。

退化向量只是估計，不是相機管線的完整知識；壓縮或空間變化未被描述時，修正也可能錯誤。輸入相依更新也不同於可一次合併的靜態 LoRA，仍需輸入相依計算。

S3Diff 使用通用正／負品質提示與線上負目標，不必逐圖產生語意描述。引導可結合不同條件預測；若某設定做兩次去雜訊，即使只有一次狀態更新，NFE 也應計兩次。

僅線上版：*S3Diff*：退化引導的低秩調適 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/s3diff* 退化引導的低秩調適/

改善短復原映射的其他方法

白話說。鏈縮短後，條件、目標與起點的誤差更顯著。近期方法改變不同元件，不只是繼續減少步數。

對抗復原蒸餾　AddSR [5]、[14] 將對抗擴散蒸餾用於影像條件復原：教師提供復原知識，判別器推動寫實輸出。它結合條件與蒸餾，並非把文字生成器直接套在放大輸入上。

對抗項能補出可信外觀，卻不保證細節正確。比較時保留教師、條件、版本與推論步數，不把最佳少步結果改標為單步。

目標感知分數與訓練狀態　TSD-SR [6] 指出，學生早期品質差時，預訓練教師可能給不可靠方向。它以配對 HR 潛在向量建構目標分數，並用分布感知抽樣產生更接近調適軌跡的訓練狀態。

配對真值只在訓練可用。比較教師對生成與真值潛在向量的預測，加入不同於學生／教師分數差的方向；這不證明已消除所有教師偏差。抽樣模組即使部署可移除，仍有訓練成本。

學習反向程序的起點　InvSR [7] 學影像條件的雜訊／起點預測器，再從該處執行預訓練反向程序。預測「雜訊」不必是獨立標準高斯，可以與觀測相關，補償以 LR 取代未知乾淨圖的誤差。

這不同於任意現成照片反演；起點模型專為復原訓練，損失作用於選定反向程序後。它可支援多種步數，但起點預測器本身也是一次網路執行，所以一個反向步不一定等於總共一次評估。

分開像素與語意增強　PiSA-SR [8] 先訓練像素重建的低秩更新，再固定它訓練語意／感知更新。預設把兩組結合、做一次預測；可調版本則分別計算像素與語意輸出，需兩次擴散評估，再調殘差權重。

可調版必須與預設單次版分開計數。控制只是改變模型行為，不是把每個像素物理拆成真實／虛構通道，也不保證指標單調。其分類器分數蒸餾不維護 OSEDiff 相同的生成分布分數模型，因此同時改變訓練機制與統計目標，不是完全等價的 VSD 捷徑。

僅線上版：*TSD-SR*：目標感知分數正規化（*CVPR 2025*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/tsd-sr 目標感知分數正規化 cvpr-2025/*；*InvSR*：學得的影像條件起點（*CVPR 2025*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/invsr 學得的影像條件起點 cvpr-2025/*；*PiSA-SR*：分開像素與語意調適（*CVPR 2025*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-diffusion-super-resolution/papers/pisa-sr 分開像素與語意調適 cvpr-2025/*

保真、寫實與控制

白話說。符合自然影像先驗，不等於符合觀測。保真項與寫實項可能往不同方向拉。

以二次問題示意：a 是已量測座標的誤差，b 是未量測紋理；先驗偏好 $(a,b)=(1,1)$，觀測限制 $a=0$。最小化 $a^2+\lambda[(a-1)^2+(b-1)^2]$，在 $\lambda>0$ 得 $a=\lambda/(1+\lambda)$、$b=1$。加強先驗也改動已受觀測限制的座標；強制 $a=0$ 可保護它，卻仍不知道紋理真值。

λ 只是示意權重，不是任何引用模型的實測超參數。神經復原器的座標無法如此乾淨分開，退化通常未知，損失也在資料集訓練，而非逐圖精確求解。這解釋了「更寫實」不等於「更保真」。

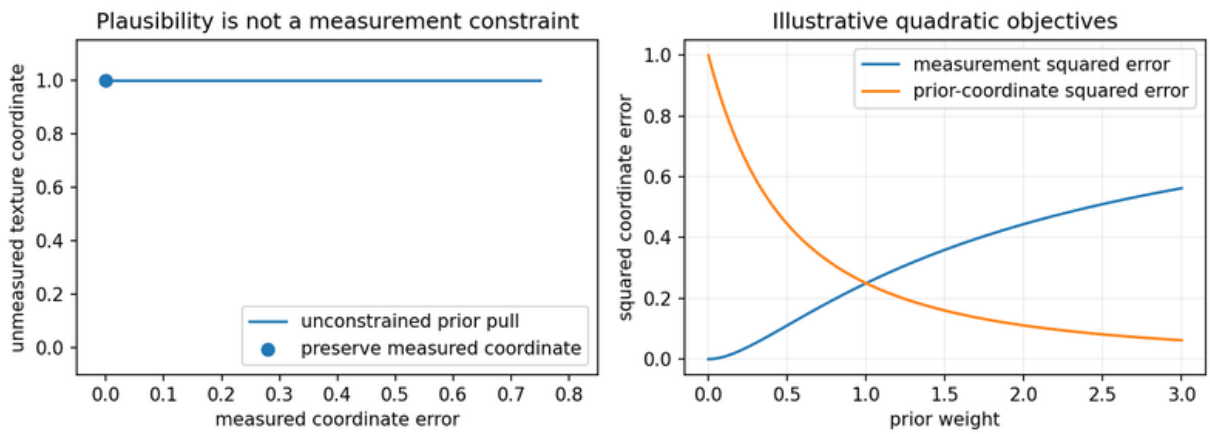


圖 2 — 二次示意目標區分量測一致與先驗偏好。這不是基準成績，也不是神經分數蒸餾的精確模型。

控制可能是訓練先驗權重、推論殘差強度、引導、退化估計或起始雜訊，彼此不可互換。改訓練權重通常要重訓；改推論係數只改同一模型；換模型還可能換資料與目標。應明示比較改動了什麼。

文字、人臉及重複結構尤其要檢查離散內容：字母、身分線索、磚塊數或邊界。可重複有助比較，卻不證明正確。保留原始輸入，將生成細節視為假設，不能當成新增量測。

如實衡量速度與品質

白話說。計算使用者真正等待的所有工作，把品質規則和計時列綁在一起。小調適檔或一次擴散更新，不是完整速度測量。

應計時的完整管線　常見流程含縮放、編碼、退化估計、提示擷取、生成／去雜訊、解碼及後處理，也可能分塊。記錄輸入**與輸出**解析度、裝置、精度、批次、暖機、同步及文字快取。

OSEDiff 表 2 在 A100、明示 **512 × 512** 輸入測試報告 **0.11 秒**，提示擷取研究另列成本 [1]。別假定與其他 128 放大至 512 的測試有相同計時邊界；若提示是否包含不明，應另報。大圖分塊編解碼可能占相當成本。

區分**可訓練參數、總參數、推論峰值記憶體及調適檔大小**。LoRA 縮小第一與最後兩者，不會等比例縮小凍結骨幹。動態更新、提示和反演各有工作；靜態合併也不會消除骨幹矩陣運算。

品質評估規則　配對合成／實拍資料使用全參考保真與感知指標；未配對真實圖則使用無參考評估及檢視 [15]、[16]、[17]。PSNR、SSIM 比較參考一致性；LPIPS、DISTS 比較感知表示 [18]、[12]、[19]。NIQE、MUSIQ、MANIQA、CLIPIQA [20]、[21]、[22]、[23] 不需 HR，卻不能驗證原場景。

固定通道、裁邊、縮放、對齊及種子。新增紋理可能提高無參考分數卻降低保真；實拍錯位影響像素指標，合成退化也可能偏袒相同訓練分布。詳細限制見 DLCV 09A.1。

隨機模型要報種子變異，不能只挑好看輸出；確定性模型要跨影像與損傷測試。小字、人臉、平坦區與規則網格應分別檢查，平均成績可能掩蓋不同錯誤。

現代觀點

白話說。短推論暴露了多種不同問題：選起點、保留觀測、估計可靠先驗方向，以及提供易懂控制。

Liu 等人的盲超解析度綜述 [9] 區分顯式退化、隱式資料分布及單圖資訊；Moser 等人的擴散超解析度綜述 [10] 依領域、條件與生成程序整理保真／成本取捨。Fan 等人的蒸餾綜述 [24] 則區分端點保真、軌跡與對抗監督。

2023–2025 年的方法不是單一替代鏈：ResShift 改程序，SinSR 蒸餾端點，OSEDiff 調適先驗，S3Diff 依退化調適，TSD-SR 改分數與狀態，InvSR 學起點，PiSA-SR 分開控制 [2]、[3]、[1]、[4]、[6]、[7]、[8]。瓶頸、骨幹與基準分布皆不同。

2026 年 TUDSR [25] 以兩次放大／擴散設計處理較大倍率，顯示短擴散元件如何放進完整系統。多階段仍須分別計數，不能宣稱整體一次操作；新倍率和架構也要重測保真與成本。

開放問題包括未知退化、文字與身分保留、劣質樣本上的分數、真實領域泛化、大圖分塊及易懂控制。介面應提供有意義的視覺選擇及原圖比較，不能暗示「增加細節」會找回未量測資訊。

重點整理

- 由觀測建立的起點，須配合預期影像轉換任務訓練才能有效省工。
- 單步推論描述計算預算，各方法仍可使用不同訓練目標。
- ResShift ／ SinSR 壓縮復原程序；OSEDiff 調適預訓練先驗。
- 潛在編碼、提示、解碼器選擇與輔助訓練各有不同作用及成本。
- 確定性與低解析度一致性都無法證明缺失細節正確。
- 量測完整流程，並比較忠實度、偏好、語意正確性與變異。

練習

1. **零空間歧義**。改變棋盤紋振幅，推導條件平均與相反紋理的平方誤差，並確認 LR 不變。

   提示

   振幅 a 下，平均的 MSE 為 a^2，反向模態為 $4a^2$；每個 2×2 紋理和為零。振幅須夠小，避免裁切改變量測方程式。

2. **錯誤的一致性檢查**。改以輕微模糊再縮放，候選 HR 不變。舊區塊平均相同，是否保證新 LR 相同？這對估計退化下的一致性有何啟示？

   提示

   零空間取決於算子。某降取樣消去的紋理，另一個未必消去。猜測算子下的一致，只是以該假設為條件的證據。

3. **訓練與推論網路**。畫出 OSEDiff，標記編碼器、去雜訊器、解碼器、提示、固定正規化器及調適正規化器。哪些接收生成器梯度，哪些部署會執行？

   提示

   生成器的調適編碼器／去雜訊器由重建與先驗更新；解碼器固定，調適正規化器有自己的去雜訊目標。兩個正規化器部署移除；除非另供提示或更改設定，提示擷取器仍保留。

4. **計算引導**。一次更新分別評估正／負提示，另執行退化估計與 VAE 編解碼。哪些能稱一步？時間報告須計什麼？

   提示

   狀態更新一次、去雜訊兩次。完整時間包含退化估計及編解碼，並明示合批慣例。快取文字嵌入能省文字編碼，不能省兩次去雜訊預測。

5. **有意義的控制**。以平坦天空、小字及重複窗戶設計「增加細節」比較，提出保真檢查、視覺品質檢查及平均無參考分數會漏掉的失敗。

   提示

   固定輸入及裁切，比較字母、窗緣與新增天空紋理。記錄品質分數，但不把它視為真值；彙總分數改善可能同時改錯離散字元。

參考文獻

1. R. Wu, L. Sun, Z. Ma, L. Zhang, “One-Step Effective Diffusion Network for Real-World Image Super-Resolution,” *NeurIPS*, 2024. source
2. Z. Yue, J. Wang, C. C. Loy, “ResShift: Efficient Diffusion Model for Image Super-resolution by Residual Shifting,” *NeurIPS*, 2023. source
3. Y. Wang, W. Yang, X. Chen, et al., “SinSR: Diffusion-Based Image Super-Resolution in a Single Step,” *CVPR*, 2024. source ·supplementary ablations
4. A. Zhang, Z. Yue, R. Pei, W. Ren, X. Cao, “Degradation-Guided One-Step Image Super-Resolution with Diffusion Priors,” *arXiv preprint*, 2024. paper
5. Y. Tai, R. Xie, C. Zhao, K. Zhang, Z. Zhang, J. Zhou, J. Yang, “AddSR: Accelerating diffusion-based blind super-resolution with adversarial diffusion distillation,” *Pattern Recognition*, 2026. published version ·manuscript supplying the discussed tables
6. L. Dong, Q. Fan, Y. Guo, Z. Wang, Q. Zhang, J. Chen, Y. Luo, C. Zou, “TSD-SR: One-Step Diffusion with Target Score Distillation for Real-World Image Super-Resolution,” *CVPR*, 2025. paper
7. Z. Yue, K. Liao, C. C. Loy, “Arbitrary-steps Image Super-resolution via Diffusion Inversion,” *CVPR*, 2025. paper
8. L. Sun, R. Wu, Z. Ma, S. Liu, Q. Yi, L. Zhang, “Pixel-level and Semantic-level Adjustable Super-resolution: A Dual-LoRA Approach,” *CVPR*, 2025. paper
9. A. Liu, Y. Liu, J. Gu, Y. Qiao, C. Dong, “Blind Image Super-Resolution: A Survey and Beyond,” *IEEE TPAMI* (*early access*), 2022. source
10. B. B. Moser, A. S. Shanbhag, F. Raue, S. Frolov, S. Palacio and A. Dengel, “Diffusion Models, Image Super-Resolution, and Everything: A Survey,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 7, pp. 11793–11813, 2025. doi ·arXiv
11. E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” *ICLR*, 2022. source
12. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” in *Proc. IEEE/CVF CVPR*, 2018, pp. 586–595. doi ·arXiv
13. Z. Wang, C. Lu, Y. Wang, F. Bao, C. Li, H. Su, J. Zhu, “ProlificDreamer: High-Fidelity and Diverse Text-to-3D Generation with Variational Score Distillation,” *NeurIPS*, 2023. paper
14. A. Sauer, D. Lorenz, A. Blattmann, R. Rombach, “Adversarial Diffusion Distillation,” *ECCV*, 2024. paper
15. J. Cai, H. Zeng, H. Yong, Z. Cao, L. Zhang, “Toward Real-World Single Image Super-Resolution: A New Benchmark and a New Model,” *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019. source
16. P. Wei, Z. Xie, H. Lu, Z. Zhan, Q. Ye, W. Zuo, L. Lin, “Component Divide-and-Conquer for Real-World Image Super-Resolution,” *ECCV*, 2020. source
17. E. Agustsson and R. Timofte, “NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study,” in *Proc. IEEE CVPR Workshops*, 2017, pp. 1122–1131. doi ·CVF
18. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
19. K. Ding, K. Ma, S. Wang, E. P. Simoncelli, “Image Quality Assessment: Unifying Structure and

Texture Similarity," *IEEE TPAMI*, 2022. source
20. A. Mittal, R. Soundararajan and A. C. Bovik, "Making a "Completely Blind" Image Quality Analyzer," *IEEE Signal Processing Letters*, vol. 20, no. 3, pp. 209–212, 2013. doi
21. J. Ke, Q. Wang, Y. Wang, P. Milanfar, F. Yang, "MUSIQ: Multi-scale Image Quality Transformer," *ICCV*, 2021. source
22. S. Yang, T. Wu, S. Shi, S. Lao, Y. Gong, M. Cao, J. Wang, Y. Yang, "MANIQA: Multi-dimension Attention Network for No-Reference Image Quality Assessment," *CVPR Workshops*, 2022. source
23. J. Wang, K. C. K. Chan, C. C. Loy, "Exploring CLIP for Assessing the Look and Feel of Images," *AAAI*, 2023. source
24. X. Fan, Z. Wu, H. Wu, L. Cao, "A Survey on Pre-Trained Diffusion Model Distillations," *arXiv preprint, version 2*, 2025. paper
25. Z. Wu, Y. Dong, X. Wei, "TUDSR: Twice Upsampling-Diffusion for Higher Super-Resolution," *CVPR*, 2026. paper
26. P. Esser, R. Rombach, B. Ommer, "Taming Transformers for High-Resolution Image Synthesis," *CVPR*, 2021. paper
27. X. Wang, L. Xie, C. Dong, Y. Shan, "Real-ESRGAN: Training Real-World Blind Super-Resolution With Pure Synthetic Data," *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021. source
28. J. Deng, W. Dong, R. Socher, L. Li, Kai Li, Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *CVPR*, 2009. doi
29. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," *CVPR*, 2022. source
30. R. Wu, T. Yang, L. Sun, Z. Zhang, S. Li, L. Zhang, "SeeSR: Towards Semantics-Aware Real-World Image Super-Resolution," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. source ·supplementary experiments
31. H. Liu, C. Li, Q. Wu, Y. J. Lee, "Visual Instruction Tuning," *NeurIPS*, 2023. source
32. Y. Li, K. Zhang, J. Liang, et al., "LSDIR: A Large Scale Dataset for Image Restoration," *CVPR Workshops*, 2023. paper
33. T. Karras, S. Laine, T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," *CVPR*, 2019. paper
34. I. Loshchilov, F. Hutter, "Decoupled Weight Decay Regularization," *ICLR*, 2019. paper
35. P. Esser, S. Kulal, A. Blattmann, et al., "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis," *ICML*, 2024. paper
36. J. Ho, T. Salimans, "Classifier-Free Diffusion Guidance," *arXiv preprint*, 2022. paper
37. R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang et al., "NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results," in *Proc. IEEE CVPR Workshops*, 2017, pp. 1110–1121. doi ·CVF

DLCV 09A.3・影片超解析度：用時間找回細節

影片超解析度如何利用多張彼此錯位的低解析度影格，找回單張影格容納不下的細節：影片退化模型、光流與扭曲、光流／可變形／注意力三種對齊方式、滑動視窗、循環與 Transformer 設計、真實世界與擴散式 VSR，以及結果如何評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/

你將學到

- 說明鄰近影格如何增加量測與時間限制。
- 建模跨影格的運動、模糊、下採樣、雜訊與壓縮。
- 比較光流扭曲、可變形對齊與注意力。
- 在延遲與記憶體限制下選擇視窗、循環或雙向處理。
- 同時評估空間忠實度、時間穩定性與生成偽影。

先看全貌

影片就是一疊間隔不到一秒拍下的照片。每張照片都很小、有點模糊，但相機或物體在兩次拍攝之間會稍微移動。影片超解析度把附近所有影格看到的東西合在一起，讓這樣的片段變得更清楚、更大。

> **用最簡單的話說**
>
> 想像你隔著紗窗看一幅畫。紗窗的細線擋住了一部分畫面，所以看一眼只能看到片段。現在把頭稍微移動一點再看一次：這次被擋住的是不同的部分。頭移動幾次之後，你就幾乎看遍整幅畫，前提是你記得每次移動了多遠。VSR 模型做的正是這件事：它算出每個東西在影格之間移動了多遠（也就是「記得移動了多遠」這一步），再用其他影格補上被擋住的部分。如果它忘了移動量，補上的部分就會落在錯的位置，畫面就糊成一團。

為什麼重要：人們看的像素大多來自影片。頻寬不足時串流會降到低解析度，老電影與家庭影片原本就是以低解析度錄製，手機、監視攝影機、衛星與顯微鏡拍下的也是小而有雜訊的影像序列。好的 VSR 模型能讓這些影片更清楚，又不會像逐影格處理的影像模型那樣出現畫面閃爍。VSR 也是許多現在影片模型常見想法的發源地：以光流對齊特徵、可變形對齊、長距離循環傳播與影片 Transformer。

本篇建立在 DLCV 09 之上；那一篇談退化模型、用合成配對訓練、損失函數與通用的影像指標，我們直接連結過去，不再重複。單張影像超解析度（輸入一個影格、輸出一個影格的設定）在 DLCV 09A 討論。這裡我們專注在時間多帶來了什麼。

為什麼影片不只是很多張影像

白話版。逐影格獨立放大會忽略鄰近量測，也未要求細節在時間上穩定。

子像素位移帶來新資訊　假設相機用一組粗糙的像素格取樣場景。在某一個影格中，落在取樣點之間的細節會遺失（更糟的是，會被折疊成錯誤的低頻圖樣，稱為**混疊**，aliasing）。如果下一個影格中場景位移了四分之一個像素，同一組像素格就會在不同的位置取樣場景。位移 $0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$ 個像素的四個影格，合起來對場景的取樣密度是其中任何一個的四倍（圖 1 是一維情形；在二維中，倍率 s 需要 s^2 種不同的位移）。這就是多影格超解析度能勝過單張影像超解析度的基本原因：多出來的影格是新的量測，而不是複本 [1]。

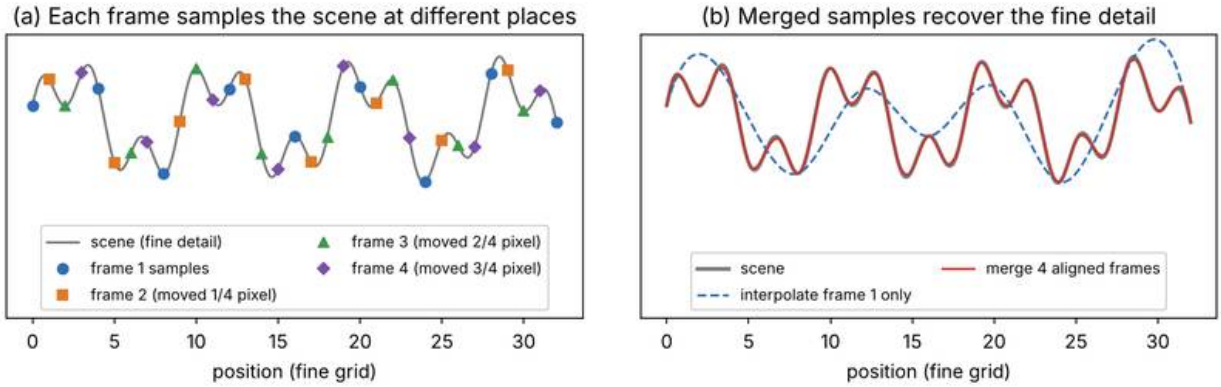


圖 *1* — 一維示意。(*a*) 四個影格在以四分之一像素為單位位移的位置上取樣同一個場景。(*b*) 只內插一個影格會漏掉細節；把四組對齊後的取樣合併就能還原。合成訊號。

必須同時滿足兩個條件。第一，運動必須是**子像素**的：位移整數個低解析度像素，會取到相同的位置，只多了雜訊平均的效果。第二，我們必須**知道運動**，而且準確到足以把每個取樣放到正確的位置。真實的運動不是整體平移：物體各自移動、旋轉、變形、出現與消失。估計並補償這些運動是 VSR 的核心問題，本篇大部分內容都圍繞著它。

時間上的冗餘　相鄰影格也高度冗餘：場景的大部分會被再看一次。獨立的雜訊在重複觀察中會被平均掉，模型也可以重複利用處理先前影格時做過的工作。影片編碼器利用的是同樣的冗餘：用運動向量由前一個影格預測每個影格（見 IP 08 的運動補償預測）。

時間一致性與閃爍　單張影像模型逐影格套用時，每個影格都會獨立做出許多小決定。凡是它捏造紋理的地方（草地、頭髮、水面、類似雜訊的細節），捏造出的紋理在每個影格都不同，影片就會**閃爍**（flicker），或像水沸騰般波動。即使每個影格單獨看都沒問題，這種時間瑕疵還是看得出來。VESPCN [2] 與 FRVSR [3] 都把時間一致性列為共同處理影格、而不是各自獨立處理的主要動機。所以 VSR 模型有兩個目標：每個影格要忠實而清晰，而且相鄰影格只應隨場景改變而改變。

影片退化模型

白話版。每個低解析度影格，都是那一刻的高解析度影格經過模糊、縮小、加雜訊，通常還經過壓縮的結果。影格之間的差異來自場景在移動。

每個影格一個方程式　把 DLCV 09 的超解析度模型延伸到由 T 個高解析度（HR）影格組成的片段 $\{x_t\}_{t=1}^{T}$，得到

$$y_t = C((x_t * k)\downarrow_s + n_t), \qquad t = 1, \dots, T,$$

其中 y_t 是時間 t 的低解析度（LR）影格，k 是模糊核（相機光學、感測器積分、抗混疊濾波器），$*$ 是二維卷積，$\downarrow_s$ 在每個方向上每隔 s 個像素保留一個（s 是放大倍率，基準測試通常是 4），n_t 是每個影格都全新產生的雜訊，C 是有失真的影片壓縮（在乾淨的基準測試中是恆等映射）。運動進入模型的方式是：每個 x_t 都是其鄰近影格移動後的版本。在影格 t 附近可以寫成

$$x_{t+j} \approx \mathcal{W}_{t \to t+j}(x_t),$$

其中 $\mathcal{W}_{t\to t+j}$ 是從影格 t 到影格 $t+j$ 的運動（扭曲），j 是小的影格偏移量，而 $\approx$ 藏起了運動解釋不了的部分：被遮擋的區域、進入畫面的物體、光照變化。把兩個式子合起來，

$$y_{t+j} \approx C((\mathcal{W}_{t\to t+j}(x_t) * k)\downarrow_s + n_{t+j}),$$

所以每個鄰近的 LR 影格，都是同一個未知量 x_t 的一次不同的量測。VSR 從 $y_{t-K}, \dots, y_{t+K}$（寬度 $2K+1$ 個影格的視窗）或整段片段來估計 x_t。

多影格重建是一個反問題　如果運動、模糊與倍率都已知，而且沒有壓縮，就能用最小平方法求出 x_t：

$$\hat{x}_t = \arg\min_x \frac{1}{2K+1} \sum_{j=-K}^{K} \| DB\,\mathcal{W}_{t\to t+j}(x) - y_{t+j} \|_2^2,$$

其中 B 是用 k 做的模糊，D 是降取樣 $\downarrow_s$，總和遍及視窗中的各個影格。只有一個影格時，這個問題嚴重欠定（$s=4$ 時，每個 LR 像素對應 $s^2 = 16$ 個未知的 HR 像素）。有許多位於不同子像素位置的影格時，它就變得好解得多。對這個目標函數做梯度下降，會反覆由目前的估計模擬出 LR 影格，與觀測到的影格比較，再把誤差分攤回 HR 格點；這就是 Irani 與 Peleg 提出的經典**迭代反向投影**（iterative back-projection）想法 [4], [1]，它也啟發了後面會談到的 RBPN 架構 [5]。

僅線上版：傳統多影格超解析度 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/notes/傳統多影格超解析度/*

玩具實驗：對齊帶來什麼　這段程式碼用一張 `skimage` 照片，加上隨機的子像素相機位移、高斯模糊、×4 子取樣與雜訊，造出一段 16 個影格的 LR 影片。接著比較第一個影格的四種重建：只用該影格的三次內插；不對齊就把所有上取樣影格平均；只用第一個影格的上述最小平方求解器；以及用全部 16 個影格、分別搭配真實位移與**由 LR 影格估計出的位移**（以子像素相位相關法 [6]）的同一個求解器。它在筆電 CPU 上大約十秒跑完。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, color
from skimage.registration import phase_cross_correlation

rng = np.random.default_rng(0)
```

```
s, T, blur, noise = 4, 16, 1.2, 0.01                        # scale, frames, blur sigma (HR
 ↪  px), noise std
hr = color.rgb2gray(data.astronaut())[64:320, 128:384]      # 256 x 256 ground-truth frame,
 ↪  values in [0, 1]
shifts = rng.uniform(0, s, size=(T, 2)); shifts[0] = 0      # sub-pixel camera motion of each
 ↪  frame (HR px)

def degrade(x, d):                                          # y_t = D B W_t x : shift, blur,
 ↪  keep every s-th pixel
    return ndi.gaussian_filter(ndi.shift(x, d, order=3, mode="reflect"), blur)[::s, ::s]
def degrade_T(r, d):                                        # adjoint: zero-fill, blur, shift
 ↪  back
    z = np.zeros((r.shape[0] * s, r.shape[1] * s)); z[::s, ::s] = r
    return ndi.shift(ndi.gaussian_filter(z, blur), -d, order=3, mode="reflect")
def upsample(y):                                            # cubic-spline interpolation on
 ↪  the HR grid
    u, v = np.mgrid[0:y.shape[0] * s, 0:y.shape[1] * s] / s
    return ndi.map_coordinates(y, [u, v], order=3, mode="nearest")
def psnr(x):
    c = (slice(8, -8), slice(8, -8))                        # skip a border, as benchmarks do
    return 10 * np.log10(1 / np.mean((np.clip(x, 0, 1)[c] - hr[c]) ** 2))

lr = np.stack([degrade(hr, d) + noise * rng.standard_normal((64, 64)) for d in shifts])
print("LR video:", lr.shape)

def reconstruct(frames, ds, iters=20):
    """Multi-frame least squares by gradient descent (iterative back-projection)."""
    x = np.mean([ndi.shift(upsample(f), -d, order=3, mode="reflect") for f, d in zip(frames,
     ↪  ds)], 0)
    for _ in range(iters):
        grad = np.mean([degrade_T(degrade(x, d) - f, d) for f, d in zip(frames, ds)], 0)
        x -= s * s * grad
    return x

# shifts estimated from the LR frames themselves (sub-pixel phase correlation), in HR pixels
est = np.array([-s * phase_cross_correlation(lr[0], f, upsample_factor=100)[0] for f in lr])
print(f"mean shift error: {np.abs(est - shifts).mean():.2f} HR px")

print(f"single frame, cubic interpolation {psnr(upsample(lr[0])):.2f} dB")
print(f"average of frames, no alignment   {psnr(np.mean([upsample(f) for f in lr], 0)):.2f}
 ↪  dB")
print(f"single frame, same solver         {psnr(reconstruct(lr[:1], shifts[:1])):.2f} dB")
print(f"{T} frames, true shifts            {psnr(reconstruct(lr, shifts)):.2f} dB")
print(f"{T} frames, estimated shifts       {psnr(reconstruct(lr, est)):.2f} dB")
# LR video: (16, 64, 64)
# mean shift error: 0.13 HR px
# single frame, cubic interpolation 25.50 dB
# average of frames, no alignment   20.99 dB
# single frame, same solver         25.81 dB
# 16 frames, true shifts            30.30 dB
# 16 frames, estimated shifts       29.33 dB
```

這次玩具實驗有三個教訓（所有數字都來自我們自己的執行，不是來自任何論文）。第一，**不對齊就平均，比只用一個影格更糟**（20.99 對 25.50 dB）：錯位的影格會把影像抹糊，正是比喻中「忘

了移動多遠」的失敗。第二，進步來自**影格，而不是求解器**：同一個最小平方求解器只用一個影格，勉強勝過內插（25.81 dB），用 16 個影格則有 30.30 dB。第三，運動可以直接**由 LR 影格本身估計**，平均誤差約為八分之一個 HR 像素，仍保住大部分進步（29.33 dB）。圖 2 顯示裁切結果，以及 PSNR 如何隨影格數增加。

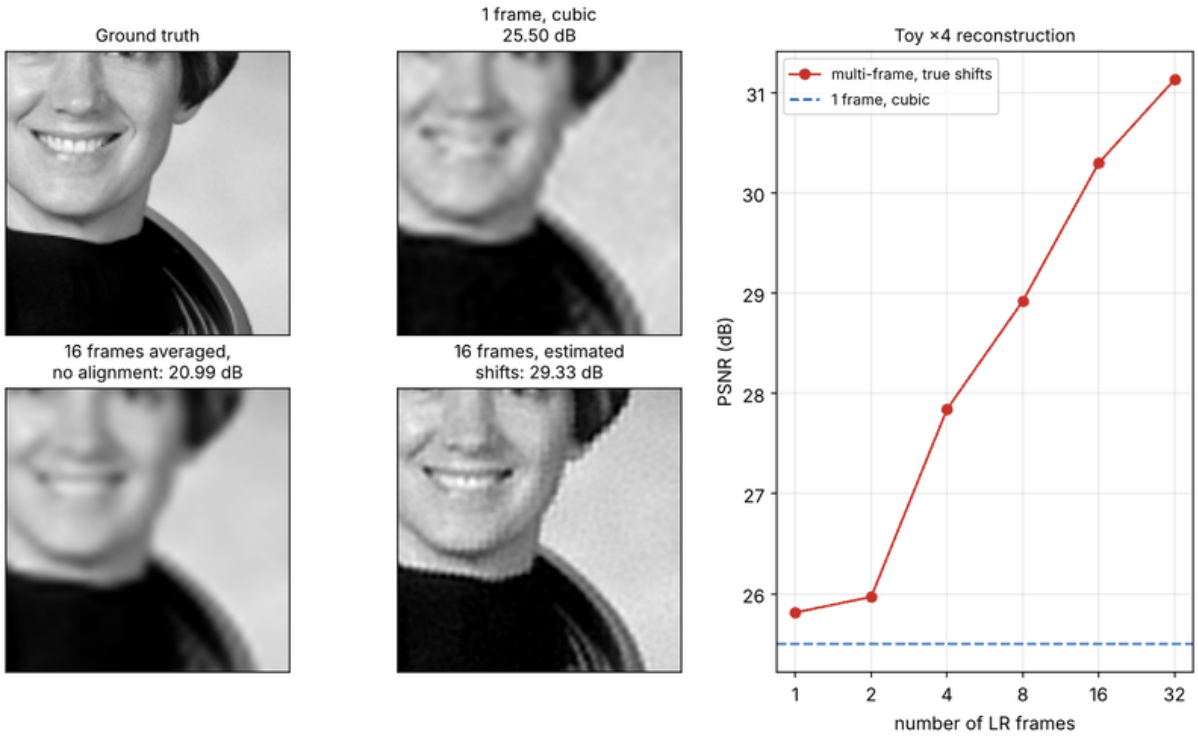


圖 *2* — 玩具多影格實驗。左：真實影像與三種重建的裁切圖（*PSNR* 計算於 *256* × *256* 影格，不含 *8* 像素的邊界）。右：使用真實位移的多影格求解器，*PSNR* 隨所用 *LR* 影格數增加的變化。數值來自我們執行圖形程式的結果。

拖動滑桿，在放大的裁切圖上比較單影格內插與 16 影格重建。



1 個影格，三次內插



16 個影格，估計位移

滑桿比較：左邊是單一影格得到的模糊放大裁切圖（太空人的笑容），右邊是十六個對齊影格得到的較清晰裁切圖，牙齒與髮際線的邊緣更清楚。

真實的影片會打破玩具實驗的假設（整體平移、已知的模糊、沒有遮擋或壓縮）。深度 VSR 保留了這個結構（估計運動、對齊、融合、重建），並從資料中學習每一個部分。圖 3 顯示幾乎所有方法都以某種形式遵循的流程；BasicVSR [7] 把它的四個元件稱為**傳播、對齊、聚合與上取樣**。

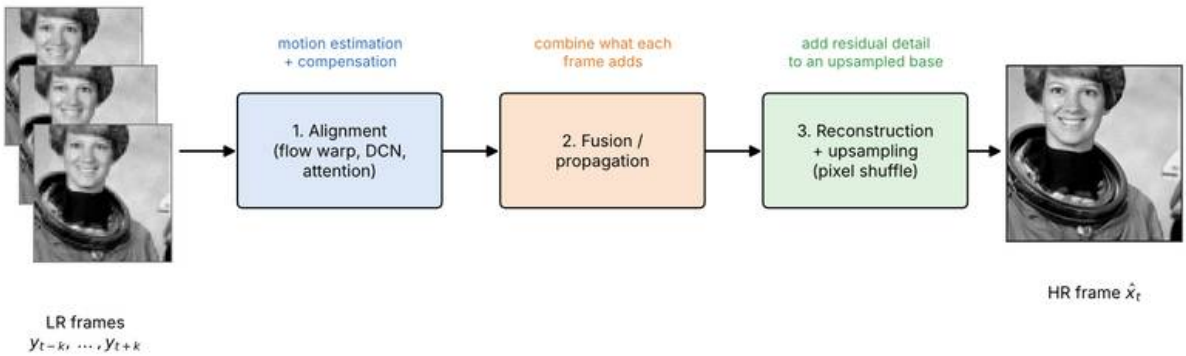


圖 *3* — 通用的 *VSR* 流程。相鄰的 *LR* 影格（或特徵）先對齊到目前影格，再於時間上融合或傳播，最後以較高的解析度重建，通常是加到上取樣基底影像上的殘差。

運動：光流與扭曲

白話版。光流估計影格間的像素對應，扭曲則依對應重新取樣，使內容對齊。

光流　經典的出發點是**亮度恆定**（brightness constancy）：一個點在移動時保持它的亮度，

$$I(u+\delta u, v+\delta v, t+1) = I(u,v,t),$$

其中 $I(u,v,t)$ 是第 u 欄、第 v 列、時間 t 的強度，$(\delta u, \delta v)$ 是該點在影格 t 與 $t+1$ 之間的位移。所有像素位移所構成的場，就是**光流**（optical flow）$F(p) = (\delta u(p), \delta v(p))$，對每個像素 p 各有一組。每個像素只有一個方程式，無法決定兩個未知數（這是「孔徑問題」：沿著一條直邊，只看得到垂直於邊的運動），所以每種方法都要加入「相鄰像素的運動相近」這個假設。Horn 與 Schunck [8] 加上整體平滑度的懲罰；OpenCV 內建的 Farnebäck 方法 [9] 則對每個鄰域擬合二次多項式，再由多項式如何移動來估計位移。

深度光流估計器從資料中學習這個映射。SPyNet [10] 以由粗到細的方式估計大的運動：在影像金字塔的每一層，先用目前的光流扭曲其中一張影像，再由一個小網路預測剩下的小幅更新；它很精簡，是 BasicVSR [7] 的光流模組。RAFT [11] 建立所有像素配對之間的相關體（correlation volume），並用循環單元反覆改進光流場。例如 BasicVSR 從預訓練的 SPyNet 出發，與模型其餘部分一起微調 [7]。

僅線上版：亮度恆定與傳統光流求解器 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/notes/亮度恆定與傳統光流求解器/*

以雙線性取樣做反向扭曲　要把影格 $t+1$ 對齊到影格 t，我們使用從 t 到 $t+1$ 的光流，並把值**拉**過來：對參考影格的每個像素 p，找出該點在另一個影格中的位置，並讀取那裡的值，

$$\hat{I}_{t+1\to t}(p) = I_{t+1}(p + F_{t\to t+1}(p)),$$

其中 $\hat{I}_{t+1\to t}$ 是被扭曲到影格 t 座標系的影格 $t+1$，$F_{t\to t+1}(p)$ 是像素 p 的位移。這就是**反向扭曲**：每個輸出像素都恰好得到一個值，沒有空洞。（前向扭曲是把每個來源像素推到它的新位置，會留下空洞與碰撞。）位置 $q = p + F(p)$ 通常不在像素格點上，所以要用周圍四個像素做**雙線性**內插：

$$I(q) = (1-a)(1-b)\,I_{00} + a(1-b)\,I_{10} + (1-a)\,b\,I_{01} + a\,b\,I_{11},$$

其中 $q = (u_0 + a, v_0 + b)$，(u_0, v_0) 為整數，$a, b \in [0, 1)$ 為小數部分，I_{ij} 是位置 $(u_0 + i, v_0 + j)$ 的值。雙線性取樣對影像值與光流都可微分，所以網路可以透過扭曲端到端地學習光流；這個「雙線性取樣器」是空間轉換網路（spatial transformer networks）[12] 的核心，也是每個以光流為基礎的 VSR 模型的核心。

下面的程式碼手寫了反向扭曲，並與 `cv2.remap` 比對；它造出兩個影格，其中背景移動 $(1.5, -2)$ 個像素，一個有紋理的方塊移動 $(6, 4)$ 個像素；然後分別用真實光流與 OpenCV 的 Farnebäck 光流，把第二個影格扭曲到第一個影格上。

```
import numpy as np, cv2
from scipy import ndimage as ndi
from skimage import data, transform

def bilinear_warp(img, flow):
    """Backward warping: out(p) = img(p + flow(p)), bilinear sampling. flow[..., 0] = dx,
     ↪  flow[..., 1] = dy."""
    H, W = img.shape
    yy, xx = np.mgrid[0:H, 0:W].astype(np.float64)
    x = np.clip(xx + flow[..., 0], 0, W - 1); y = np.clip(yy + flow[..., 1], 0, H - 1)
    x0 = np.minimum(np.floor(x).astype(int), W - 2); y0 = np.minimum(np.floor(y).astype(int),
     ↪  H - 2)
    ax, ay = x - x0, y - y0                                    # fractional parts = bilinear
     ↪  weights
    return ((1 - ay) * ((1 - ax) * img[y0, x0] + ax * img[y0, x0 + 1])
            + ay * ((1 - ax) * img[y0 + 1, x0] + ax * img[y0 + 1, x0 + 1]))

# two frames: the background moves by (dx, dy) = (1.5, -2.0) px, a textured square moves by
 ↪  (6, 4) px
bg = transform.resize(data.camera() / 255.0, (256, 256), anti_aliasing=True)
obj = transform.resize(data.coins() / 255.0, (60, 60), anti_aliasing=True)
def frame(t):
    f = ndi.shift(bg, (-2.0 * t, 1.5 * t), order=3, mode="reflect")   # ndi.shift takes
     ↪  (rows, cols)
    r, c = 90 + 4 * t, 100 + 6 * t
    f[r:r + 60, c:c + 60] = obj
    return f
```

```
I0, I1 = frame(0).astype(np.float32), frame(1).astype(np.float32)

# true backward flow from frame 0 to frame 1: where is pixel p of frame 0 found in frame 1?
flow = np.zeros((256, 256, 2), np.float32); flow[..., 0], flow[..., 1] = 1.5, -2.0
flow[90:150, 100:160] = (6.0, 4.0)
mine = bilinear_warp(I1, flow)
H, W = I0.shape; yy, xx = np.mgrid[0:H, 0:W].astype(np.float32)
ocv = cv2.remap(I1, xx + flow[..., 0], yy + flow[..., 1], cv2.INTER_LINEAR,
 ↪  borderMode=cv2.BORDER_REPLICATE)
print(f"max |ours - cv2.remap| = {np.abs(mine - ocv)[5:-5, 5:-5].max():.4f}")

u8 = lambda a: (np.clip(a, 0, 1) * 255).round().astype(np.uint8)        # Farneback expects
 ↪  8-bit images
est = cv2.calcOpticalFlowFarneback(u8(I0), u8(I1), None, pyr_scale=0.5, levels=4, winsize=21,
                                   iterations=5, poly_n=7, poly_sigma=1.5, flags=0)
m0 = np.zeros((H, W), np.float32); m0[90:150, 100:160] = 1          # square in frame 0
m1 = np.zeros((H, W), np.float32); m1[94:154, 106:166] = 1          # square in frame 1
occl = (bilinear_warp(m1, flow) > 0) & (m0 == 0)                  # background of frame 0
 ↪  hidden in frame 1
inner = (slice(8, -8), slice(8, -8))
for name, img in [("no alignment", I1), ("warped, true flow", mine), ("warped, Farneback",
 ↪  bilinear_warp(I1, est))]:
    e = np.abs(img - I0)
    print(f"{name:18s} mean |error| {e[inner].mean():.4f}   on occluded pixels
     ↪  {e[occl].mean():.4f}")
print("Farneback median flow (dx, dy): square", np.median(est[100:140, 110:150], axis=(0,
 ↪  1)).round(2),
      " background", np.median(est[200:240, 20:60], axis=(0, 1)).round(2))
# max |ours - cv2.remap| = 0.0000
# no alignment       mean |error| 0.0562   on occluded pixels 0.2367
# warped, true flow  mean |error| 0.0089   on occluded pixels 0.2617
# warped, Farneback  mean |error| 0.0133   on occluded pixels 0.1733
# Farneback median flow (dx, dy): square [6. 4.]  background [ 1.42 -1.88]
```

用真實光流扭曲，把兩個影格之間的平均差異降低到約六分之一，而 Farnebäck 的估計已經達到大部分效果（我們的執行結果）。但請看**被遮擋的像素**：就是在影格 $t+1$ 中被方塊蓋住的那一條背景。在那裡，即使是真實的光流也有很大的誤差，因為那些內容根本不在影格 $t+1$ 裡，它們被藏在方塊後面。任何光流都救不了這一點。VSR 模型必須學會在扭曲後的影格一致時相信它，不一致時就忽略它（圖 4）。

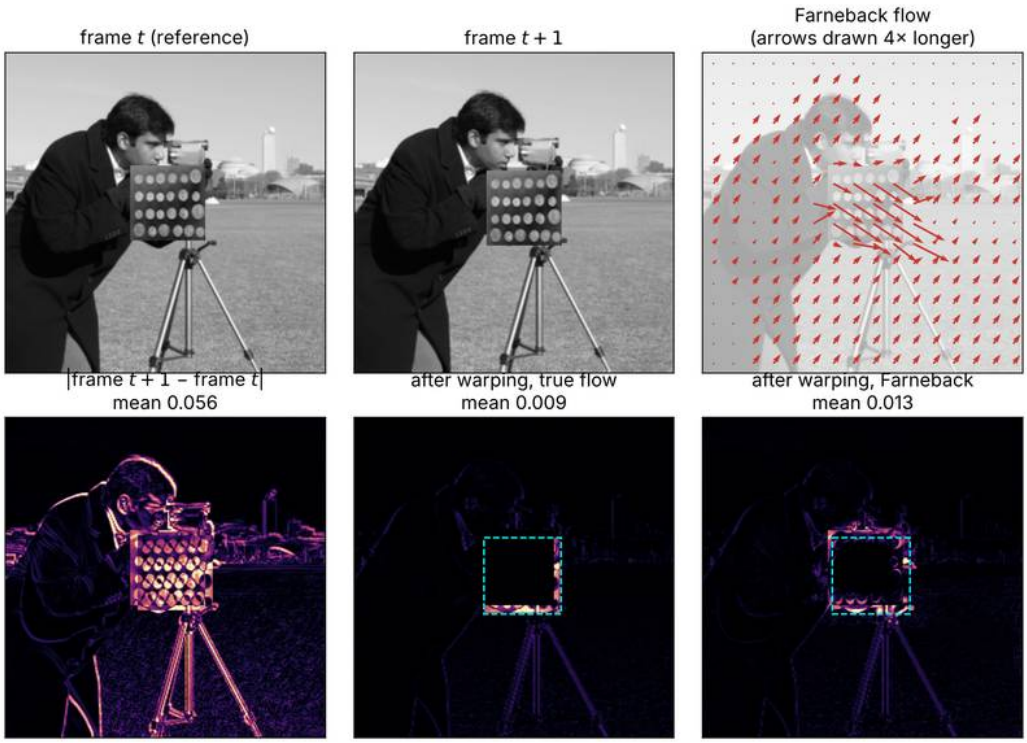


圖 *4* — 反向扭曲。把影格 *t+1* 扭曲到影格 *t* 上，消除了大部分差異，但被遮擋的帶狀區域（虛線框）除外：影格 *t* 中看得到的背景，在影格 *t+1* 中被藏起來了。在我們的執行中，*Farneback* 的誤差集中在運動邊界。平均絕對誤差來自我們的執行結果。

對齊策略

白話版。對齊可使用顯式光流、學得的可變形取樣位置，或對對應內容的注意力。

顯式對齊：以光流扭曲　最直接的做法是估計每個鄰近影格到參考影格的光流，再用它扭曲鄰近影格。

- **VESPCN** [2]（CVPR 2017）端到端訓練一個快速的多解析度空間轉換運動補償模組，搭配子像素卷積（pixel shuffle）網路 ESPCN [13] 的時空版本。
- **TOFlow** [14]（IJCV 2019）把光流網路與復原網路一起訓練，所以光流是**以任務為導向**的：它會移動任何能讓輸出最好的東西，即使這與實際的物理運動不同。這篇論文也提出了 Vimeo-90K。
- **影像扭曲與特徵扭曲**。早期的方法扭曲 LR 影像。不準確的光流會產生模糊或錯誤的扭曲影像，而損失的細節之後就無法挽回。後來的方法在影像上計算光流，卻扭曲特徵圖，讓後續各層能修正小幅的錯位；BasicVSR 的消融實驗發現，特徵扭曲優於影像扭曲 [7]。

以光流為基礎的對齊可解釋，在中等運動下效果很好。它的弱點是光流網路的成本、遮擋與運動邊界處的誤差，以及每個像素只有一個光流向量，只能指向一個位置。

隱式對齊：可變形卷積 一般的 3×3 卷積，是在每個像素周圍固定的格點上讀取輸入。**可變形卷積**（deformable convolution）[15] 則為每個取樣位置加上一個學得的偏移：

$$y(p) = \sum_{i=1}^{K} w_i \, x(p + p_i + \Delta p_i(p)),$$

其中 $y(p)$ 是像素 p 的輸出，K 是卷積核的取樣點數（3×3 時為 9），w_i 是第 i 個取樣點的權重，p_i 是它固定的格點位置（例如 $(-1,-1)$），$\Delta p_i(p)$ 是由一個小網路為每個像素與每個取樣點預測的實數偏移，$x(\cdot)$ 以雙線性取樣讀取。DCNv2 [16] 另外把每個取樣點乘上一個落在 $[0,1]$ 的學得調變權重，讓這一層能壓低不可靠位置的權重。

用於對齊時，偏移由參考影格與鄰近影格的特徵預測，可變形卷積則套用在鄰近影格的特徵上。**TDAN** [17] 以這種方式在特徵層級把每個支援影格對齊到參考影格，不需要計算光流。**EDVR** [18] 在**金字塔、串接與可變形（Pyramid, Cascading and Deformable, PCD）**模組中套用可變形對齊，在特徵金字塔上由粗到細處理大的運動，再用**時間與空間注意力（temporal and spatial attention, TSA）**融合對齊後的特徵；它拿下了 NTIRE 2019 影片復原挑戰賽的全部四個賽道。

Chan 等人 [19] 證明，可變形卷積可以拆解成數個空間扭曲（每個取樣點一個，各有自己的偏移場），再接一個卷積。所以可變形對齊是光流扭曲的推廣：用數個光流場取代一個；這種**偏移多樣性**（offset diversity）正是它對齊得更好的原因。同一項研究與 BasicVSR++ [20] 都回報，可變形對齊訓練起來不穩定：偏移可能變得非常大（「偏移溢位」，offset overflow）。因此 BasicVSR++ 把可變形偏移預測成**疊加在光流上的殘差**（光流引導的可變形對齊），結合光流的穩定性與可變形取樣的多樣性。

僅線上版：可變形卷積即「扭曲加卷積」— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/notes/可變形卷積即扭曲加卷積/*

不需顯式運動步驟的對齊 有些模型完全跳過運動估計，讓網路自行吸收運動。**DUF** [21] 為每個輸出像素，從輸入影格的時空鄰域預測一個動態上取樣濾波器與一個殘差，所以運動是由依內容而變的濾波器隱式處理的。**PFNL** [22] 逐步融合影格，並用非區域（non-local）運算（它把每個位置與其他每個位置比較）取代運動估計與補償。在（時間、高度、寬度）上做 3-D 卷積是第三種選擇。這些方法避開了光流的誤差，但有效搜尋範圍受限於感受域，所以大的運動較難處理。

注意力與 Transformer 注意力讓每個位置能與其他影格中許多候選位置比較，並對最佳匹配取加權平均，這是運動搜尋的軟式、學習版本。**VRT** [23] 把影片切成短的片段，並使用**時間互相自注意力（temporal mutual self-attention, TMSA）**：兩個影格之間的互相注意力執行聯合的運動估計、對齊與融合，自注意力則負責擷取特徵；片段的切分在相鄰層之間會位移，讓資訊能跨越片段邊界，另有一個**平行扭曲**（parallel warping）模組，額外加入鄰近特徵的光流扭曲。VRT 應用於影片 SR、去模糊、去雜訊、影格內插與時空 SR。**RVRT** [24] 使用**引導式可變形注意力**（guided deformable attention）：從光流出發，在前一個片段中預測數個相關位置，並以注意力權重聚合它們的特徵。

依時間設計分類的架構

白話版。一旦影格能夠對齊，問題就變成：每個輸出能看到哪些影格，以及資訊如何沿著影片傳遞。主要有四種答案，如圖 5 所示。

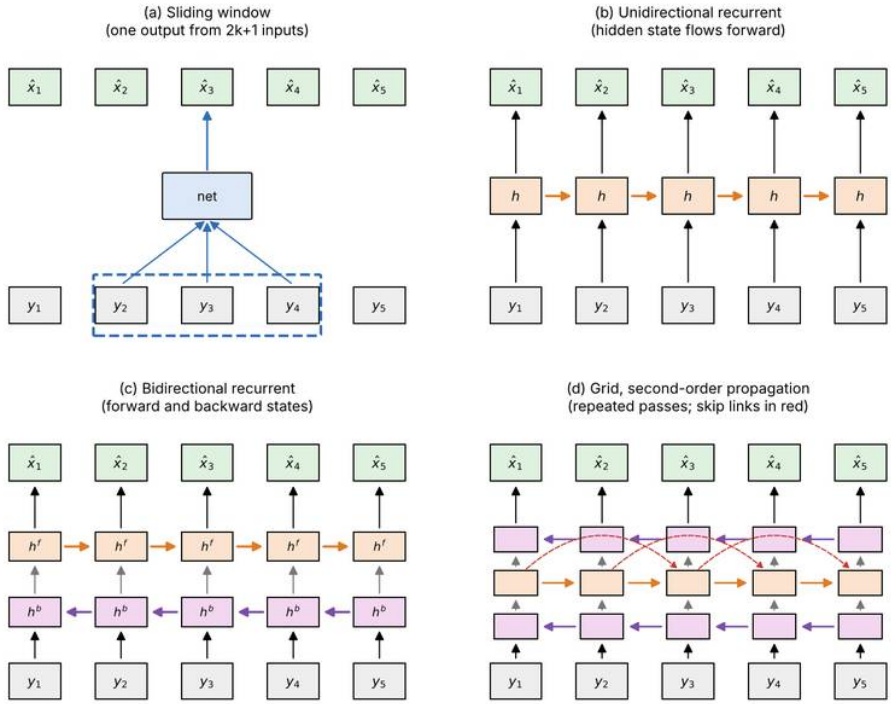


圖 5 — 時間設計。(*a*) 滑動視窗：每個輸出由固定的 $2K+1$ 個影格視窗計算。(*b*) 單向循環：隱藏狀態向前攜帶資訊。(*c*) 雙向循環：分開的前向與後向狀態，每個輸出在其時間步結合兩者。(*d*) 帶二階連接的網格傳播（*BasicVSR++*）：反覆的後向與前向傳遞，連結中也有跳過一個影格的。

滑動視窗　**滑動視窗**（sliding-window）模型取 $2K+1$ 個連續的 LR 影格，產生一個 HR 影格（中間那一個），然後把視窗移動一個影格。VESPCN、DUF、TDAN 與 EDVR 都是滑動視窗模型。它簡單又可平行，誤差也不會隨時間累積。FRVSR [3] 指出兩個弱點：每個輸入影格要被處理與扭曲 $2K+1$ 次，而且每個輸出都是獨立估計的，限制了時間一致性。固定的視窗也無法利用更遠處的資訊；BasicVSR 的作者測量到，把測試序列切成較短的獨立片段會降低 PSNR，而且在片段末端降得最多 [7]。

循環　**循環**（recurrent）模型保有一個概括過去的隱藏狀態 h_t，並逐影格更新它：

$$h_t = F(y_t,\ \mathcal{W}(h_{t-1}, F_{t\to t-1})), \qquad \hat{x}_t = U(h_t),$$

其中 F 是一個網路，把目前的 LR 影格 y_t 與被光流 $F_{t\to t-1}$ 扭曲到影格 t 的前一個狀態結合起來（扭曲 $\mathcal{W}$ 就是上一節的反向扭曲），而 U 是重建與上取樣的輸出頭。因為狀態在重複使用之前已

經對齊，資訊可以用每個影格固定的成本傳遞許多個影格。（關於一般的循環網路，請見 ML 04。）**FRVSR** [3] 把前一個 *HR* 輸出經光流扭曲並重新排列成 LR 大小後，餵給下一個影格的網路，這有助於產生時間一致的結果，而且每一步只需要一次扭曲。**RBPN** [5] 是混合式：在視窗內，它用受反向投影啟發的循環編碼器–解碼器，一次整合一個鄰近影格，並把估計出的運動當作輸入餵進去，而不是扭曲影格。

單向循環模型只看得到過去。早期的影格得到的資訊很少，而狀態中的誤差（來自不好的光流、遮擋或場景切換）會被帶進後面的影格：這就是**誤差累積**（error accumulation）。

雙向循環　**BasicVSR** [7] 執行兩條循環分支，一條沿時間向前，一條沿時間向後：

$$h_t^b = F_b(y_t, y_{t+1}, h_{t+1}^b), \qquad h_t^f = F_f(y_t, y_{t-1}, h_{t-1}^f), \qquad \hat{x}_t = U(h_t^f, h_t^b),$$

其中 h_t^b 與 h_t^f 是影格 t 的後向與前向特徵，F_b 與 F_f 是兩條分支（各自用 SPyNet 在它的兩個輸入影格之間估計光流，用它扭曲傳播而來的特徵，再以殘差區塊改進），U 則把兩種特徵串接起來，並用 pixel shuffle 上取樣。現在每個輸出都看得到整段片段，包括過去與未來。論文的消融實驗發現，單向版本在早期影格的 PSNR 明顯下降，而不做對齊則下降更多，這支持了它的配方：雙向傳播加上以光流為基礎的特徵對齊，就足以成為強大的基線。它的延伸版本 **IconVSR** 加入一個資訊補充（information-refill）模組，從稀疏的關鍵影格重新擷取特徵，以限制遮擋與邊界處的誤差累積；以及耦合傳播（coupled propagation），把後向分支的輸出餵給前向分支。

代價是延遲：後向分支需要未來的影格，所以在第一個輸出之前，必須先取得整段片段（或一大段）。這適合離線處理，以及有緩衝區的串流，但不適合即時影片。

僅線上版：*BasicVSR 與 IconVSR* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/papers/basicvsr-與-iconvsr/*

網格與二階傳播　**BasicVSR++** [20] 從兩方面讓傳播更豐富。**網格傳播**（grid propagation）以交替方式把特徵向後、向前傳遞數次，讓每個影格的特徵一再地用整段序列的資訊改進。**二階連接**（second-order connections）讓影格 t 的狀態同時使用影格 $t-1$ 與 $t-2$（二階馬可夫鏈，而不是一階），使資訊能繞過一個壞掉的影格，這對遮擋與細緻結構有幫助。搭配前面描述的光流引導可變形對齊，BasicVSR++ 在相近的參數量下，回報比 BasicVSR 高 0.82 dB 的 PSNR，並在 NTIRE 2021 影片超解析度與壓縮影片增強挑戰賽中拿下三個賽道的第一名、一個賽道的第二名 [20]。

Transformer 與循環–Transformer 混合　VRT [23] 以多個尺度的長距離時間注意力，**平行**處理所有影格，威力強大但很吃資源。RVRT [24] 直接描述了這個取捨：平行模型能把時間資訊融合得很好，但模型大、記憶體用量高；循環模型在各影格之間共用參數，所以精簡，但長距離建模較弱，也較難平行化。RVRT 在整體循環的框架內，把相鄰影格當成一個片段平行處理，用推論出的前一個片段來估計下一個，並以引導式可變形注意力做片段到片段的對齊。

下表整理了本篇的代表方法。

方法	發表處	時間設計	對齊
VESPCN [2]	CVPR 2017	滑動視窗	光流（空間轉換），影像層級

方法	發表處	時間設計	對齊
FRVSR [3]	CVPR 2018	循環（前一個 HR 輸出）	光流，影像層級
DUF [21]	CVPR 2018	滑動視窗	無（動態濾波器）
TOFlow [14]	IJCV 2019	滑動視窗	以任務為導向的光流
EDVR [18]	CVPRW 2019	滑動視窗	可變形（PCD）＋注意力融合
RBPN [5]	CVPR 2019	視窗內循環	運動當作輸入，反向投影
PFNL [22]	ICCV 2019	滑動視窗（漸進融合）	無（非區域）
TDAN [17]	CVPR 2020	滑動視窗	可變形，特徵層級
BasicVSR / IconVSR [7]	CVPR 2021	雙向循環	光流，特徵層級
BasicVSR++ [20]	CVPR 2022	網格，二階	光流引導的可變形
RVRT [24]	NeurIPS 2022	片段循環 Transformer	引導式可變形注意力
VRT [23]	TIP 2024	平行 Transformer	互相注意力＋平行扭曲

真實世界影片與壓縮

白話版。基準測試的 LR 影片是用一套乾淨、已知的配方做出來的。真實的影片有雜訊、以未知的方式模糊，最重要的是還被壓縮過。用乾淨配方訓練的模型，常常把壓縮瑕疵銳化了，而不是場景。

壓縮是真實世界最主要的退化　大部分影片都以 H.264/AVC [25] 這類以區塊為基礎的編碼器儲存與串流；它們用運動向量由先前的影格預測每個區塊，並把殘差量化。由此產生的瑕疵（區塊效應、振鈴、模糊的紋理、邊緣周圍的蚊式雜訊）在影格之間各不相同：不參考其他影格而編碼的影格（內部影格，intra frame）看起來與預測影格不同，品質也隨位元率改變。這些瑕疵不是獨立的雜訊，而且會隨內容移動，所以 VSR 模型可能把它們誤認為真實細節並傳播下去。**COMISR** [26] 為壓縮輸入設計 VSR，用雙向循環扭曲、保留細節的光流估計，以及拉普拉斯增強模組，把內部影格的位置等壓縮特性納入考量，並在一系列壓縮率下評估。

真實世界 VSR：RealBasicVSR　**RealBasicVSR** [27]（CVPR 2022）發現，對輕微退化有幫助的長期傳播，會**放大嚴重的瑕疵**：雜訊與壓縮瑕疵會與有用的資訊一起被傳播與放大。它的解法是在 BasicVSR 網路之前，對每個影格先套用一個影像**預清理**（pre-cleaning）模組。它也提出了 **VideoLQ**，這是一組沒有真實標準答案的真實低品質影片，以無參考指標與視覺比較來評估。

僅線上版：*RealBasicVSR* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/papers/realbasicvsr/*

這些退化流程是 DLCV 09 中盲退化流程的影片版：隨機模糊、縮放、雜訊與壓縮，另外還要選擇對片段的所有影格套用相同或不同的參數。

以擴散模型為基礎的 VSR　預訓練的文字生成影像擴散模型，能為真實世界影像 SR 產生非常逼真的紋理（見 DLCV 09）。在影片中，它們的隨機性是個問題：逐影格獨立取樣，會讓每個影格的紋理都不同，這是最嚴重的閃爍。因此近期的方法在預訓練的生成模型上加入時間機制：時間層加上光流引導、跨序列的潛在變數傳播（Upscale-A-Video [28]，CVPR 2024）、引導取樣路徑的運動引導損失（MGLD-VSR [29]，ECCV 2024）、本來就會建模運動的文字生成影片模

型（STAR [30]，ICCV 2025）、可處理任意長度與解析度影片的擴散 Transformer（SeedVR [31]，CVPR 2025），以及降低成本的一步蒸餾（DLoRAL [32]，NeurIPS 2025）。

僅線上版：以擴散模型為基礎的 *VSR* 模型 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-super-resolution/notes/以擴散模型為基礎的-vsr-模型/*

這些模型在真實影片上給出最令人信服的紋理，但它們承襲了影像層級的**捏造細節**（hallucinated detail，合理但從未存在於場景中的內容）風險，並多了影片層級的風險：每個影格看起來都合理，但影格之間不一致的細節。它們也比 CNN 或 Transformer 的 VSR 貴得多。

訓練 VSR 模型

白話版。訓練與影像復原相同，只是每個訓練樣本是一小段影片，而不是單一影像。

配對的做法是用上述模型把 HR 影片片段退化：基準測試用雙三次（BI）或模糊後降取樣（BD）配方（見「評估」），或用隨機化的真實世界流程。每一步取樣一小段片段，並在所有影格的同一位置隨機裁切；翻轉與旋轉對所有影格做相同處理，影格順序也可以反轉當作時間上的資料擴增（BasicVSR 就這樣反轉 Vimeo-90K 的 7 影格序列 [7]）。循環模型用較長的片段訓練（BasicVSR 在 REDS 上的設定使用 15 個影格的輸入），好讓傳播確實被練到；RealBasicVSR 發現較長的序列勝過較大的批次，指向同一個方向 [27]。在 BasicVSR 中，預訓練的光流網路與特徵擷取器在前 5,000 次迭代中是凍結的，之後光流網路以比其餘部分更小的學習率訓練 [7]。

損失函數通常是在每個輸出影格上的 L1 類型像素損失（BasicVSR 使用平滑版本 $\rho(z) = \sqrt{z^2 + \varepsilon^2}$，其中 ε 是一個小常數，z 是輸出與真實像素之間的差 [7]），真實世界模型再加上感知與對抗項；DLCV 09 討論了這些。有些方法加入**時間損失**，懲罰經運動補償後連續輸出之間的差異，例如 Lai 等人 [33] 的短期扭曲損失，它把每個輸出影格與經光流扭曲的前一個輸出比較，並在扭曲不可靠的地方遮罩掉。

評估

白話版。把每個影格都與真實標準答案比較評分，將分數平均，並明確說明 LR 影格是怎麼做出來的。然後觀看實際播放的影片，因為逐影格的分數看不到閃爍。

逐影格保真度：Y 通道 PSNR 與 SSIM　大多數 VSR 論文回報在**亮度（Y）通道**上的 PSNR 與 SSIM [34]（定義見 DLCV 09）；每個影格各自計算，先對片段內的影格平均，再對片段平均。Y 通道依循 ITU-R BT.601 [35]：由 $[0,1]$ 中經過 gamma 校正的 RGB 算出亮度 $E_Y = 0.299E_R + 0.587E_G + 0.114E_B$，並以 8 位元的「studio」刻度量化為 $Y = 219E_Y + 16$。這個慣例並不通用：在 BasicVSR 與 BasicVSR++ 的比較表中，除了 REDS4 是算在 RGB 上之外，所有結果都算在 Y 上 [7], [20]。混用不同的慣例會讓數字差好幾 dB，下面的程式碼就是例子。

```
import numpy as np
from skimage import data, color

def rgb_to_y(rgb):
    """BT.601 luma on the 8-bit studio scale: Y = 16 + 219 * (0.299 R + 0.587 G + 0.114 B),
    ↪  RGB in [0, 1]."""
```

```
    return 16 + 65.481 * rgb[..., 0] + 128.553 * rgb[..., 1] + 24.966 * rgb[..., 2]

def psnr(a, b, peak=255.0):
    return 10 * np.log10(peak ** 2 / np.mean((a.astype(np.float64) - b) ** 2))

rng = np.random.default_rng(0)
gt = np.stack([np.roll(data.astronaut(), 2 * t, axis=1) for t in range(5)]) / 255.0   # 5
 ↪  ground-truth frames
out = np.clip(gt + rng.normal(0, 0.02, gt.shape) * np.array([1.0, 0.3, 1.5]), 0, 1)  #
 ↪  unequal error per channel

y_scores = [psnr(rgb_to_y(o), rgb_to_y(g)) for o, g in zip(out, gt)]
rgb_scores = [psnr(255 * o, 255 * g) for o, g in zip(out, gt)]
print("matches skimage rgb2ycbcr:", np.allclose(rgb_to_y(gt[0]), color.rgb2ycbcr(gt[0])[...,
 ↪  0]))
print(f"Y-channel PSNR, mean of frames: {np.mean(y_scores):.2f} dB")
print(f"RGB PSNR,       mean of frames: {np.mean(rgb_scores):.2f} dB")
# matches skimage rgb2ycbcr: True
# Y-channel PSNR, mean of frames: 43.74 dB
# RGB PSNR,       mean of frames: 33.94 dB
```

同一個輸出在 Y 上的分數比在 RGB 上高約 10 dB（我們的執行結果），一部分是因為 Y 是三個通道的加權平均，獨立的通道誤差會部分抵消，另一部分是因為 Y 只涵蓋 255 個階層中的 219 個，而公式中的峰值仍是 255。務必說明所用的通道、邊界裁切，以及影格是否已四捨五入成 8 位元。

退化設定：BI 與 BD　兩種 LR 配方主導了乾淨的基準測試，而且在它們之下的結果不能直接比較：

- **BI（雙三次，bicubic）：**用 MATLAB 的 `imresize` 把每個 HR 影格縮小 ×4，它先用三次濾波器模糊，再做子取樣。Vimeo-90K SR 基準測試就是這樣建立的 [14]。
- **BD（模糊後降取樣，blur-downsampling）：**先用標準差 $\sigma = 1.6$ 的高斯函數模糊每個 HR 影格，再每隔四個像素保留一個，如 BasicVSR [7] 所規定。

BD 包含更強而且已知的模糊，所以模型除了上取樣，還必須去模糊。BasicVSR 的表格中，REDS4 只回報 BI，UDM10 只回報 BD，而 Vimeo-90K-T 與 Vid4 兩者都有 [7]。閱讀比較表時，先檢查每一欄的設定。

時間一致性　逐影格的 PSNR 看不到閃爍：每個影格隨機變動的誤差，可能和穩定的誤差得到相同的平均 PSNR。常用的指標有三類：

- **扭曲誤差**（warping error）[33]：用光流把輸出影格 $t+1$ 扭曲到影格 t，並量測加上遮罩的差異，

$$E_{\text{warp}}(V_t, V_{t+1}) = \frac{1}{\sum_i M_t^{(i)}} \sum_{i=1}^{N} M_t^{(i)} \lVert V_t^{(i)} - \hat{V}_{t+1}^{(i)} \rVert_2^2,$$

其中 $V_t^{(i)}$ 是輸出影格 t 的像素 i，$\hat{V}_{t+1}$ 是被扭曲到影格 t 的影格 $t+1$，N 是像素數，$M_t^{(i)} \in \{0,1\}$

是非遮擋遮罩；影片的分數是對所有連續配對取平均。它的弱點是：模糊、毫無細節的影片非常穩定，分數也很好。

- **tOF 與 tLP**（TecoGAN [36]）比較輸出與真實標準答案的時間行為，而不是要求完全不變：

$$\mathrm{tOF} = \left\| \mathrm{OF}(b_{t-1}, b_t) - \mathrm{OF}(g_{t-1}, g_t) \right\|_1, \qquad \mathrm{tLP} = \left| \mathrm{LP}(b_{t-1}, b_t) - \mathrm{LP}(g_{t-1}, g_t) \right|,$$

其中 b_t 與 g_t 是輸出與真實標準答案的影格，OF 是用 Farnebäck 方法估計的光流，LP 是 LPIPS 感知距離 [37]。TecoGAN 的作者提出它們，是因為單純的扭曲差異指標「很容易被欺騙」，例如被雙三次內插這類模糊的結果騙過。

- **時間剖面**（temporal profile）：取一列像素沿時間追蹤，堆疊成一張影像；斑點與條紋會顯露閃爍。VESPCN 用它做質化比較 [2]。

下面的程式碼造出一個運動已知的玩具場景和四種「輸出」：每個影格獨立捏造的紋理、沿運動方向遞迴平滑的同一批輸出、忽略運動的同樣平滑，以及一個模糊的輸出。閃爍的量測精神近似 tOF：輸出在運動補償後的時間變化，減去真實標準答案的時間變化。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, transform

rng = np.random.default_rng(0)
T, v = 12, (1, 2)                                       # 12 frames, the scene moves 1 px
 ↪  down and 2 px right per frame
base = transform.resize(data.camera() / 255.0, (256, 256), anti_aliasing=True)
gt = np.stack([np.roll(base, (v[0] * t, v[1] * t), axis=(0, 1)) for t in range(T)])
warp_prev = lambda f: np.roll(f, v, axis=(0, 1))          # exact motion compensation for
 ↪  this toy scene

def fake_detail():                                      # zero-mean "hallucinated texture",
 ↪  new every frame
    n = ndi.gaussian_filter(rng.standard_normal(base.shape), 0.8)
    return 0.03 * n / n.std()

per_frame = gt + np.stack([fake_detail() for _ in range(T)])  # each frame restored
 ↪  independently
smoothed, unaligned, blurry = per_frame.copy(), per_frame.copy(),
 ↪  np.stack([ndi.gaussian_filter(g, 1.5) for g in gt])
for t in range(1, T):
    smoothed[t] = 0.3 * per_frame[t] + 0.7 * warp_prev(smoothed[t - 1])   # recursive filter
     ↪  along the motion
    unaligned[t] = 0.3 * per_frame[t] + 0.7 * unaligned[t - 1]           # same filter,
     ↪  motion ignored

c = (slice(None), slice(16, -16), slice(16, -16))          # ignore borders (np.roll wraps
 ↪  around)
def psnr(out):
    return np.mean([10 * np.log10(1 / np.mean((o - g) ** 2)) for o, g in zip(out[c], gt[c])])
def flicker(out):
    """Mean |temporal change of output - temporal change of ground truth| after motion
     ↪  compensation."""
```

```
    d_out = out[1:] - np.stack([warp_prev(f) for f in out[:-1]])
    d_gt = gt[1:] - np.stack([warp_prev(f) for f in gt[:-1]])
    return np.abs(d_out - d_gt)[c].mean()

for name, out in [("per-frame", per_frame), ("aligned smoothing", smoothed),
                  ("unaligned smoothing", unaligned), ("blurry output", blurry)]:
    print(f"{name:20s} PSNR {psnr(out):5.2f} dB   flicker {flicker(out):.4f}")
# per-frame            PSNR 30.50 dB   flicker 0.0338
# aligned smoothing    PSNR 36.40 dB   flicker 0.0082
# unaligned smoothing  PSNR 21.74 dB   flicker 0.0223
# blurry output        PSNR 25.99 dB   flicker 0.0000
```

把這四行合起來讀（都來自我們的玩具實驗）。**沿著運動**傳播資訊，會降低閃爍，在這個玩具中 PSNR 也提高了，因為捏造的紋理是零平均的雜訊，會被平均掉；真實的捏造紋理不是這樣，所以不要期待 PSNR 有這麼大的進步。**不對齊**就傳播，會讓移動中的場景產生鬼影：PSNR 崩落，閃爍仍然很高。而**模糊的輸出**閃爍為零，但 PSNR 很差，這就是縮小版的 TecoGAN 警告：時間指標務必與保真度或感知指標一起回報。圖 6 以經過運動補償的時間剖面顯示同樣的效應。

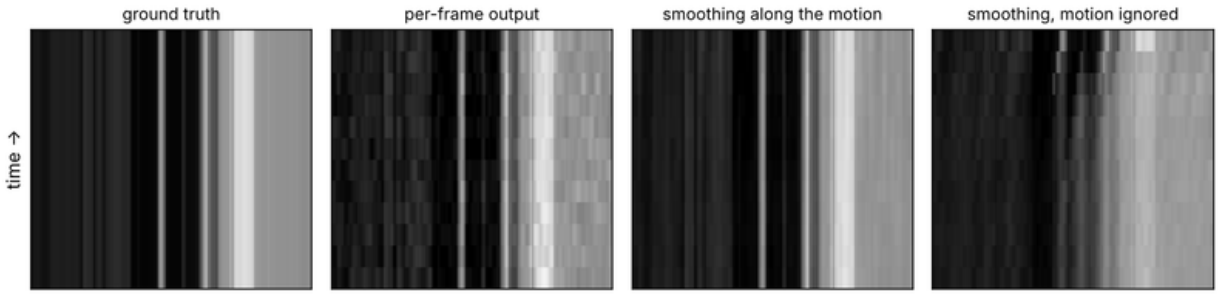


圖 *6* — 玩具場景經運動補償的時間剖面：一列場景像素在 *12* 個影格中的變化（時間向下）。閃爍表現為逐列的斑點；未對齊的平滑會抹糊結構。

資料集與測試集

資料集	內容	典型用途
Vimeo-90K [14]	取自 Vimeo 的 89,800 段影片（720p 或更高）；SR 基準測試使用 448 × 256 的 7 影格序列，×4 的 LR 輸入為 112 × 64，以 MATLAB `imresize` 製作	訓練；測試切分 **Vimeo-90K-T**
REDS [38]	300 段序列，每段 100 個影格，720 × 1280（240 段訓練、30 段驗證、30 段測試），為 NTIRE 2019 影片挑戰賽拍攝，運動大而複雜	訓練；**REDS4** 測試集
REDS4	四段 REDS 訓練片段（000、011、015、020），依循 EDVR [7] 保留作測試	BI 下的測試
Vid4 [39]	取自 Liu 與 Sun 的貝氏適應式 VSR 研究的四段短序列	BI 與 BD 下的測試
UDM10	與 PFNL [22] 一起釋出的十段測試序列	BD 下的測試

資料集	內容	典型用途
VideoLQ [27]	沒有真實標準答案的真實低品質影片	真實世界評估（無參考指標、視覺）

Liu 等人的綜述指出，隨著方法瞄準更複雜的運動與場景變化，測試影片從 Vid4 逐漸演變到 REDS [40]；BasicVSR 則把 REDS4 描述為包含大而複雜的運動 [7]。由於 Vimeo-90K-T 的序列只有七個影格，長距離傳播在那裡沒有太多可發揮的空間：BasicVSR 在 Vimeo-90K-T 上回報的分數略低於 EDVR 等滑動視窗方法，並把原因歸於序列太短 [7]。

挑戰賽 NTIRE（CVPR）與 AIM（ECCV）研討會的挑戰賽建立了資料集，並追蹤最新技術水準。**NTIRE 2019** 提出了 REDS [38]；它的影片 SR 挑戰賽 [41] 有一個雙三次賽道與一個含真實運動模糊的賽道，EDVR 當年拿下了全部四個影片復原賽道 [18]。**NTIRE 2021** 影片 SR [42] 有一個復原品質賽道與一個較低影格率賽道，後者把問題變成時空 SR（同時提高解析度與影格率）；BasicVSR++ 回報在那次挑戰賽與壓縮影片增強挑戰賽合計拿下三個第一名與一個第二名 [20]。**效率挑戰賽**評斷每單位運算量的品質：Mobile AI & AIM 2022 報告 [43] 要求在手機的神經處理單元上即時進行 ×4 VSR，以 REDS 訓練，並以執行時間與耗電評比；AIM 2024 的壓縮內容高效率 VSR 挑戰賽 [44] 則針對串流，要求把 540p 以 ×4 放大到 4K，以及把 360p 以 ×3 放大到 1080p。

效率與串流

白話版。影片模型必須對每個影格都執行，常常要即時，而且常常在未來影格還不存在時就得輸出。這些限制對架構的影響不亞於準確度。

- **每個影格的成本**。除非快取特徵，滑動視窗會把每個輸入影格處理 $2K+1$ 次；循環模型每個影格只處理一次並攜帶一個狀態，所以每個影格的成本不會隨影片長度增加 [3]。
- **延遲與因果性**。雙向與網格傳播需要未來的影格，所以第一個輸出得等到整段片段結束。即時影片（通話、遊戲串流）只容許因果設計或短的前瞻緩衝區，品質會有所損失（BasicVSR 的單向消融實驗 [7]）。長影片要分段處理，品質會在段落末端下降，如 BasicVSR 的分段實驗所示。
- **記憶體**。像 VRT 這樣的平行 Transformer，會在多個尺度上跨許多影格做注意力；它們的模型大小與記憶體用量是 RVRT 以片段層級的循環要解決的缺點 [24]。擴散式 VSR 更貴，這也是推動一步模型的動機 [32]。
- **部署**。效率挑戰賽為手機與串流設定了目標：少量參數與運算、即時影格率與低耗電，品質則與內插基線而不是最大的模型相比 [43], [44]。

失效模式

白話版。 VSR 會在它的主要假設不成立的地方失敗：這個假設是，相鄰影格顯示的是同樣的內容，只是稍微移動過。

- **遮擋與去遮擋**。在鄰近影格中被藏起來的內容（圖 4）沒有額外的量測；模型若在那裡相信扭曲後的特徵，就會產生鬼影或雙重邊緣。調變遮罩與注意力權重讓模型能壓低這些區域的權重 [20]。

- **大而複雜的運動**。快速的非剛體運動與動態模糊會讓對齊不準確；由粗到細的對齊（EDVR 的 PCD、SPyNet 的金字塔）有幫助，但不能消除問題。REDS 正是用來對這一點施加壓力 [18]。
- **誤差累積**。在循環模型中，錯誤的狀態會被往前帶，尤其是在遮擋或場景切換之後；IconVSR 的關鍵影格補充與 BasicVSR++ 的二階連結，都是直接的對策 [7], [20]。長距離傳播也會放大嚴重的輸入瑕疵，RealBasicVSR 以預清理來對付 [27]。
- **閃爍**。在單張影像模型或擴散取樣中，獨立的逐影格決策會造成時間上的不穩定，而逐影格指標看不到。
- **捏造的細節**。 GAN 與擴散式 VSR 可能創造出從未存在的合理紋理、文字或人臉，而在影片中，捏造的內容還可能漂移。對於監視、鑑識、醫療或科學影片，請優先選用以保真度為導向的模型，並說明輸出是由哪一種模型產生的。
- **領域差距**。用 BI 或 BD 退化訓練的模型，沒有考慮壓縮與真實雜訊 [26], [27]。

現代觀點

綜述論文

- **Liu 等人，「Video Super-Resolution Based on Deep Learning: A Comprehensive Survey」（Artificial Intelligence Review，2022）** [40]。回顧 37 個深度 VSR 方法，依**如何利用影格間資訊**分成七個子類別：有對齊（運動估計與補償、可變形卷積）或無對齊（2-D 與 3-D 卷積、循環 CNN、非區域運算、其他），並在基準測試上比較。重點：這個領域從對齊的短視窗，走向混合的設計與長序列，REDS 取代 Vid4 成為嚴格的測試集。
- **Tu 等人，「Optical Flow for Video Super-Resolution: A Survey」（Artificial Intelligence Review，2022）** [45]。涵蓋指標以及傳統與深度的光流式 VSR，並分析代表性的深度方法。重點：稠密的子像素光流是最常用的運動補償工具之一，所以理解光流，就是理解 VSR 對齊的一大部分。
- **Baniya 等人，「A Survey of Deep Learning Video Super-Resolution」（IEEE Transactions on Emerging Topics in Computational Intelligence，2024）** [46]。一套元件分類法（輸入、對齊、融合、改進與重建、上取樣），加上損失函數、品質與效率評估、應用，以及線上、真實世界、資源受限與時空 VSR 等趨勢。重點：設計選擇常由基準測試的進步所驅動，而不是被解釋清楚，實務情境值得更多關注。

2023–2026 年的最新進展

1. **傳播加對齊是以保真度為導向的 VSR 的核心**。帶光流引導可變形對齊的雙向或網格循環傳播（BasicVSR++），以及平行或片段循環的 Transformer（VRT、RVRT），是本篇的參考設計，在準確度與記憶體、延遲之間取捨。
2. **真實世界 VSR 轉向生成式先驗**。繼 RealBasicVSR 等以 CNN 為基礎的真實世界模型之後，出現了建立在影像擴散模型上的擴散式 VSR（Upscale-A-Video、MGLD-VSR），接著是建立在影片擴散模型與擴散 Transformer 上的（STAR、SeedVR），時間一致性是核心的技術問題。
3. **生成式 VSR 的降低成本**。一步擴散（DLoRAL）走的路，與一步影像 SR 相同。
4. **效率與壓縮成為一級目標**，挑戰賽中有針對行動硬體與壓縮串流內容的賽道 [43], [44]。

尚未解決的問題

- **忠實與合理**。在影片中量測並控制幻覺仍未解決，因為捏造的細節在時間上也可能不穩定。
- **時間評估**。扭曲誤差可以靠模糊鑽漏洞，tOF 與 tLP 依賴光流與特徵估計，沒有公認的時間品質感知指標，而使用者研究又很昂貴。
- **真實退化與編碼器**。合成流程仍然漏掉了真實的相機處理與編碼器行為；真實影片很少有標準答案。
- **長影片、線上處理與場景切換**。要把雙向與全域模型的品質，與因果、固定記憶體的處理結合起來，仍然困難。
- **遮擋與大的運動**。在快速的非剛體運動與去遮擋下，對齊仍是弱點。
- **生成式 VSR 的成本**，對於長的高解析度影片，仍遠高於 CNN 與 Transformer 的 VSR。

重點整理

- 次像素運動可能提供額外場景量測，逐影格獨立放大會忽略它們。
- 合併前須對齊對應內容，遮擋區域可能沒有有效匹配。
- 光流、可變形取樣與注意力提供不同對應機制及成本。
- 循環會傳遞資訊，也會傳遞錯誤；雙向處理還需要未來影格。
- 壓縮偽影與生成紋理可能隨時間累積或閃爍。
- 同時報告退化、色彩、邊界協定及空間、時間指標；單靠模糊也能顯得穩定。

練習

1. **整數運動與子像素運動**。在多影格程式碼中，把隨機位移換成整數位移（`rng.integers(0, s, size=(T, 2)).astype(float)`，保留 `shifts[0] = 0`），重新執行。再試試所有位移都等於零。用圖 1 的取樣論證，解釋你在每種情況得到的 PSNR。

   提示

   以 HR 像素為單位的整數位移，仍會讓 LR 取樣落在不同的 HR 相位上（HR 位移 1 等於四分之一個 LR 像素），所以它們仍然有幫助；重要的是位移對 s 個 HR 像素取餘數之後的值。所有位移都為零時，每個影格取樣的是相同的 HR 位置，所以只有雜訊被平均，結果會停留在接近單影格求解器的程度。試試 4 個 HR 像素的倍數位移，就能直接看到「沒有新資訊」的情況。

2. **配準誤差預算**。在呼叫 `reconstruct` 之前，對真實位移加上標準差為 0.25、0.5 與 1.0 個 HR 像素的高斯雜訊，並畫出 PSNR 對位移誤差的圖。與相位相關法 0.13 個像素的誤差比較。這對超解析度所需的對齊準確度，相較於去雜訊，說明了什麼？

   提示

   一旦位移誤差接近要還原的細節大小（一個 HR 像素的幾分之一），PSNR 應該會快速下降，因為取樣落在錯的位置，重建會變模糊或出現振鈴。靠平均來去雜訊能容忍較大的誤差，因為它只需要相同的內容，而不需要內容確切的子像素位置。這就是為什麼 VSR 需要子像素準確的光流，而許多影片去雜訊器用較粗的對齊就能運作。

3. **前向扭曲與反向扭曲**。寫一個前向扭曲函式，把影格 $t+1$ 的每個像素推到 $p - F(p)$（四捨五入到最近的像素），並在兩影格範例上與 `bilinear_warp` 的輸出比較。數出空洞（沒有收到任何值的像素）與碰撞（收到多個值的像素）各有多少。

   提示

   空洞出現在場景擴張或去遮擋的地方（移動方塊的後方），碰撞則出現在場景收縮或被遮擋的地方。反向扭曲兩者都沒有，因為每個輸出像素都恰好讀取一個（內插後的）值，這就是網路使用它的原因；代價是它需要在參考影格上定義的光流。

4. **滑動視窗與循環的成本**。一個視窗為 7 個影格的滑動視窗模型，對視窗中每個輸入影格各執行一次特徵擷取器，並對每個鄰近影格各執行一個光流網路。一個雙向循環模型每個影格執行一次特徵擷取器，以及兩次光流估計（每個方向一次）。數出每種設計在每個輸出影格上的特徵擷取次數與光流估計次數，分別計算有快取與沒有快取視窗之間的特徵的情況。對於一段 100 個影格的片段，哪一種設計的第一個輸出延遲較低？

   提示

   沒有快取時，視窗模型每個輸出要做 7 次特徵擷取與 6 次光流估計；有快取時，每個輸出只要 1 次新的擷取，但仍是 6 次光流（每個鄰近影格都必須對齊到新的中心影格）。雙向模型每個影格做 1 次擷取與 2 次光流。但它的第一個輸出需要對全部 100 個影格做後向傳遞，視窗模型則只需要影格 1 到 4。每個影格的成本有利於循環；延遲則有利於視窗。

5. **閃爍指標的陷阱**。在閃爍程式碼中，把零平均的捏造細節換成一個隨場景移動的固定紋理圖樣（只產生一次，並隨場景一起滾動）。重新執行，並解釋為什麼閃爍指標改變了，PSNR 卻沒有多大變化。這說明了 tOF/tLP 類型的指標能偵測什麼、不能偵測什麼？

   提示

   與場景一致移動的紋理，時間變化與真實標準答案相同，所以經運動補償的閃爍會降到約為零，而 PSNR 仍和之前差不多低，因為紋理仍然是錯的。時間指標偵測的是不一致，而不是不正確：一致地捏造出來的紋理能通過這些指標。保真度（或感知）指標與時間指標回答的是不同的問題。

參考文獻

1. S. Farsiu, M. D. Robinson, M. Elad and P. Milanfar, “Fast and Robust Multiframe Super Resolution,” *IEEE Transactions on Image Processing*, vol. 13, no. 10, pp. 1327–1344, 2004. doi
2. J. Caballero, C. Ledig, A. Aitken, A. Acosta, J. Totz, Z. Wang and W. Shi, “Real-Time Video Super-Resolution with Spatio-Temporal Networks and Motion Compensation,” in *Proc. IEEE CVPR*, 2017. CVF ·arXiv
3. M. S. M. Sajjadi, R. Vemulapalli and M. Brown, “Frame-Recurrent Video Super-Resolution,” in *Proc. IEEE/CVF CVPR*, 2018. CVF ·arXiv
4. M. Irani and S. Peleg, “Improving Resolution by Image Registration,” *CVGIP: Graphical Models and Image Processing*, vol. 53, no. 3, pp. 231–239, 1991. doi

5. M. Haris, G. Shakhnarovich and N. Ukita, “Recurrent Back-Projection Network for Video Super-Resolution,” in *Proc. IEEE/CVF CVPR*, 2019. CVF ·arXiv
6. M. Guizar-Sicairos, S. T. Thurman and J. R. Fienup, “Efficient Subpixel Image Registration Algorithms,” *Optics Letters*, vol. 33, no. 2, p. 156, 2008. doi
7. K. C. K. Chan, X. Wang, K. Yu, C. Dong and C. C. Loy, “BasicVSR: The Search for Essential Components in Video Super-Resolution and Beyond,” in *Proc. IEEE/CVF CVPR*, 2021. CVF · arXiv
8. B. K. P. Horn and B. G. Schunck, “Determining Optical Flow,” *Artificial Intelligence*, vol. 17, no. 1–3, pp. 185–203, 1981. doi
9. G. Farnebäck, “Two-Frame Motion Estimation Based on Polynomial Expansion,” in *Image Analysis* (Lecture Notes in Computer Science), pp. 363–370, Springer, 2003. doi
10. A. Ranjan and M. J. Black, “Optical Flow Estimation Using a Spatial Pyramid Network,” in *Proc. IEEE CVPR*, 2017. CVF ·arXiv
11. Z. Teed and J. Deng, “RAFT: Recurrent All-Pairs Field Transforms for Optical Flow,” in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 402–419, 2020. doi ·arXiv
12. M. Jaderberg, K. Simonyan, A. Zisserman and K. Kavukcuoglu, “Spatial Transformer Networks,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2015. proceedings · arXiv
13. W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert and Z. Wang, “Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network,” in *Proc. IEEE CVPR*, 2016. arXiv
14. T. Xue, B. Chen, J. Wu, D. Wei and W. T. Freeman, “Video Enhancement with Task-Oriented Flow,” *International Journal of Computer Vision*, vol. 127, no. 8, pp. 1106–1125, 2019. doi ·arXiv
15. J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu and Y. Wei, “Deformable Convolutional Networks,” in *Proc. IEEE ICCV*, 2017. CVF ·arXiv
16. X. Zhu, H. Hu, S. Lin and J. Dai, “Deformable ConvNets V2: More Deformable, Better Results,” in *Proc. IEEE/CVF CVPR*, 2019. CVF ·arXiv
17. Y. Tian, Y. Zhang, Y. Fu and C. Xu, “TDAN: Temporally-Deformable Alignment Network for Video Super-Resolution,” in *Proc. IEEE/CVF CVPR*, 2020. CVF ·arXiv
18. X. Wang, K. C. K. Chan, K. Yu, C. Dong and C. C. Loy, “EDVR: Video Restoration with Enhanced Deformable Convolutional Networks,” in *Proc. IEEE/CVF CVPR Workshops*, 2019. CVF ·arXiv
19. K. C. K. Chan, X. Wang, K. Yu, C. Dong and C. C. Loy, “Understanding Deformable Alignment in Video Super-Resolution,” in *Proc. AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, pp. 973–981, 2021. doi ·arXiv
20. K. C. K. Chan, S. Zhou, X. Xu and C. C. Loy, “BasicVSR++: Improving Video Super-Resolution with Enhanced Propagation and Alignment,” in *Proc. IEEE/CVF CVPR*, 2022. CVF ·arXiv
21. Y. Jo, S. W. Oh, J. Kang and S. J. Kim, “Deep Video Super-Resolution Network Using Dynamic Upsampling Filters Without Explicit Motion Compensation,” in *Proc. IEEE/CVF CVPR*, 2018. CVF
22. P. Yi, Z. Wang, K. Jiang, J. Jiang and J. Ma, “Progressive Fusion Video Super-Resolution Network via Exploiting Non-Local Spatio-Temporal Correlations,” in *Proc. IEEE/CVF ICCV*, 2019. CVF ·code and test data (Vid4, UDM10)
23. J. Liang, J. Cao, Y. Fan, K. Zhang, R. Ranjan, Y. Li, R. Timofte and L. Van Gool, “VRT: A Video Restoration Transformer,” *IEEE Transactions on Image Processing*, vol. 33, pp. 2171–2182, 2024.

doi ·arXiv
24. J. Liang, Y. Fan, X. Xiang, R. Ranjan, E. Ilg, S. Green, J. Cao, K. Zhang, R. Timofte and L. Van Gool, “Recurrent Video Restoration Transformer with Guided Deformable Attention,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 35, pp. 378–393, 2022. arXiv
25. T. Wiegand, G. J. Sullivan, G. Bjøntegaard and A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003. doi
26. Y. Li, P. Jin, F. Yang, C. Liu, M.-H. Yang and P. Milanfar, “COMISR: Compression-Informed Video Super-Resolution,” in *Proc. IEEE/CVF ICCV*, 2021. CVF ·arXiv
27. K. C. K. Chan, S. Zhou, X. Xu and C. C. Loy, “Investigating Tradeoffs in Real-World Video Super-Resolution,” in *Proc. IEEE/CVF CVPR*, 2022. CVF ·arXiv
28. S. Zhou, P. Yang, J. Wang, Y. Luo and C. C. Loy, “Upscale-A-Video: Temporal-Consistent Diffusion Model for Real-World Video Super-Resolution,” in *Proc. IEEE/CVF CVPR*, pp. 2535–2545, 2024. CVF ·arXiv
29. X. Yang, C. He, J. Ma and L. Zhang, “Motion-Guided Latent Diffusion for Temporally Consistent Real-World Video Super-Resolution,” in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 224–242, 2024. doi ·arXiv
30. R. Xie, Y. Liu, P. Zhou, C. Zhao, J. Zhou, K. Zhang, Z. Zhang, J. Yang, Z. Yang and Y. Tai, “STAR: Spatial-Temporal Augmentation with Text-to-Video Models for Real-World Video Super-Resolution,” in *Proc. IEEE/CVF ICCV*, 2025. CVF ·arXiv
31. J. Wang, Z. Lin, M. Wei, Y. Zhao, C. Yang, C. C. Loy and L. Jiang, “SeedVR: Seeding Infinity in Diffusion Transformer Towards Generic Video Restoration,” in *Proc. IEEE/CVF CVPR*, 2025. CVF ·arXiv
32. Y. Sun, L. Sun, S. Liu, R. Wu, Z. Zhang and L. Zhang, “One-Step Diffusion for Detail-Rich and Temporally Consistent Video Super-Resolution,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 38, 2025. arXiv
33. W.-S. Lai, J.-B. Huang, O. Wang, E. Shechtman, E. Yumer and M.-H. Yang, “Learning Blind Video Temporal Consistency,” in *Proc. ECCV*, 2018. CVF ·arXiv
34. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
35. ITU-R, “Recommendation ITU-R BT.601-7: Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios,” International Telecommunication Union, 2011. ITU
36. M. Chu, Y. Xie, J. Mayer, L. Leal-Taixé and N. Thuerey, “Learning Temporal Coherence via Self-Supervision for GAN-based Video Generation,” *ACM Transactions on Graphics*, vol. 39, no. 4, article 75, 2020. doi ·arXiv
37. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” in *Proc. IEEE/CVF CVPR*, 2018. CVF ·arXiv
38. S. Nah, S. Baik, S. Hong, G. Moon, S. Son, R. Timofte and K. M. Lee, “NTIRE 2019 Challenge on Video Deblurring and Super-Resolution: Dataset and Study,” in *Proc. IEEE/CVF CVPR Workshops*, 2019. CVF
39. C. Liu and D. Sun, “On Bayesian Adaptive Video Super Resolution,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 2, pp. 346–360, 2014. doi

40. H. Liu, Z. Ruan, P. Zhao, C. Dong, F. Shang, Y. Liu, L. Yang and R. Timofte, "Video Super-Resolution Based on Deep Learning: A Comprehensive Survey," *Artificial Intelligence Review*, vol. 55, no. 8, pp. 5981–6035, 2022. doi ·arXiv
41. S. Nah, R. Timofte, S. Gu, S. Baik, S. Hong, G. Moon, S. Son and K. M. Lee, "NTIRE 2019 Challenge on Video Super-Resolution: Methods and Results," in *Proc. IEEE/CVF CVPR Workshops*, 2019. CVF
42. S. Son, S. Lee, S. Nah, R. Timofte and K. M. Lee, "NTIRE 2021 Challenge on Video Super-Resolution," in *Proc. IEEE/CVF CVPR Workshops*, 2021. CVF ·arXiv
43. A. Ignatov, R. Timofte et al., "Power Efficient Video Super-Resolution on Mobile NPUs with Deep Learning, Mobile AI & AIM 2022 Challenge: Report," arXiv:2211.05256, 2022. arXiv
44. M. V. Conde, Z. Lei, W. Li, C. Bampis, I. Katsavounidis, R. Timofte et al., "AIM 2024 Challenge on Efficient Video Super-Resolution for AV1 Compressed Content," in *Proc. ECCV Workshops* (Lecture Notes in Computer Science), pp. 304–325, 2025. doi ·arXiv
45. Z. Tu, H. Li, W. Xie, Y. Liu, S. Zhang, B. Li and J. Yuan, "Optical Flow for Video Super-Resolution: A Survey," *Artificial Intelligence Review*, vol. 55, no. 8, pp. 6505–6546, 2022. doi · arXiv
46. A. A. Baniya, T.-K. Lee, P. W. Eklund and S. Aryal, "A Survey of Deep Learning Video Super-Resolution," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 4, pp. 2655–2676, 2024. doi ·arXiv
47. B. Wronski, I. Garcia-Dorado, M. Ernst, D. Kelly, M. Krainin, C.-K. Liang, M. Levoy and P. Milanfar, "Handheld Multi-Frame Super-Resolution," *ACM Transactions on Graphics*, vol. 38, no. 4, 2019. doi ·arXiv

DLCV 09B・生成式影像復原：取樣出一個合理的答案

迴歸訓練的復原模型為什麼會模糊、改成從後驗分布取樣是什麼意思，以及 GAN 先驗、條件式與零樣本擴散、潛在擴散和流式復原模型如何做到，並說明隨之而來的評估、成本與幻覺風險。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/

你將學到

- 說明平方誤差復原為何平均不確定細節。
- 區分合理後驗樣本與忠實還原。
- 比較 GAN 先驗、條件擴散與預訓練先驗的反問題求解器。
- 用可執行的簡例追蹤先驗引導與量測一致性。
- 評估隨機性、幻覺與完整推論成本。

先看全貌

DLCV 09 訓練的復原網路，目標是平均而言接近乾淨影像。損傷輕微時這樣做很有效。但損傷若抹掉了大量資訊（寬度只有 16 像素的人臉、壓縮得很重的照片、一大塊缺洞），「平均而言接近」就會給出平滑、像蠟像一樣的猜測。生成式復原換了一種立場：既然有許多乾淨影像都與輸入相符，那就從中挑一張，並確保它看起來像真實的照片。

> **用最簡單的話說**
>
> 朋友給你看一張又小又模糊的車牌照片，問你上面寫什麼。一種回答方式，是想像所有可能長成這團模糊的車牌，再把它們全部混在一起：你會得到一團灰色的污跡，它不算錯，卻完全讀不出來。另一種方式，是畫出一張確實可能長成這團模糊的車牌：字母清晰、一目了然。這幅畫看起來很棒，但它只是其中一種可能，字母也可能不是真正的那幾個。迴歸式的復原模型畫的是污跡；生成式的復原模型畫的是清晰的車牌。本篇要談的，就是如何把清晰的車牌畫好，以及永遠別忘了它可能是錯的。

為什麼重要：近年最讓人信服的真實世界超解析度，以及人臉與老照片復原，都來自生成模型，先是 GAN，約從 2021 年起則是擴散模型。同樣的技術也被用在醫學與科學的逆問題上，在那裡，捏造出來的細節可能造成實際的傷害。所以你需要同時具備兩種能力：知道這些方法怎麼運作，也知道怎麼衡量它們捏造了什麼。

本篇的位置。本篇建立在 DLCV 09 的退化模型、損失函數與評估指標，以及 ML 07 的生成模型之上；擴散與流匹配在 ML 08 推導，這裡只拿來使用。超解析度有自己的篇章（DLCV 09A 談影像、DLCV 09A.3 談影片），在這裡只當作範例。所有程式碼都用 NumPy 與 SciPy，在 CPU 上幾秒內就能跑完，用的是答案已知的玩具問題，所以每項說法都可以驗證。

為什麼迴歸會模糊：後驗平均

白話版。平方誤差訓練偏向條件平均；當多種清晰結構都能解釋相同輸入，其平均可能模糊。

一個輸入，許多答案

像 DLCV 09 那樣，把量測寫成 $y = \mathcal{D}(x) + n$。對於超解析度、去模糊、影像修補或重度壓縮，$\mathcal{D}$ 都會丟掉資訊，所以有一整組乾淨影像 x 都幾乎同樣好地解釋同一個 y。用貝氏的語言來說，看到 y 之後仍然剩下的不確定性，就是**後驗分布**（posterior distribution）

$$p(x \mid y) = \frac{p(y \mid x)\, p(x)}{p(y)},$$

其中 $p(x)$ 是**先驗**（prior；在看到 y 之前一張影像有多可能：自然影像很可能，雜訊不可能），$p(y \mid x)$ 是**概似**（likelihood；x 對量測的解釋有多好，若是標準差為 σ 的 Gaussian 雜訊，它正比於 $\exp(-\|y - \mathcal{D}(x)\|^2 / 2\sigma^2)$），而 $p(y)$ 是不依賴 x 的正規化常數。損傷輕微時，後驗很窄，每個合理的方法都會得到相同的結果。損傷嚴重時，後驗很寬，常常有好幾個分離的眾數（mode），例如「邊緣在這裡」或「邊緣再往右兩個像素」。

數學：平方誤差的最小值就是後驗平均

復原網路 f_θ 用成對資料 (y, x) 以平方誤差訓練，是在資料分布上最小化期望損失 $\mathbb{E}\,\|f_\theta(y) - x\|^2$。對每個固定的 y，最佳輸出 $\hat{x}$ 要最小化

$$\mathbb{E}[\|\hat{x} - x\|^2 \mid y] = \|\hat{x} - \mathbb{E}[x \mid y]\|^2 + \mathbb{E}[\|x - \mathbb{E}[x \mid y]\|^2 \mid y],$$

其中期望是對後驗 $p(x \mid y)$ 取的，$\mathbb{E}[x \mid y]$ 是後驗平均。第二項不依賴 $\hat{x}$（它是無法消除的不確定性），所以最小值出現在

$$\hat{x}_{\mathrm{MMSE}}(y) = \mathbb{E}[x \mid y] = \int x\, p(x \mid y)\, dx,$$

這就是**最小均方誤差**（minimum mean squared error, MMSE）估計量。夠大的網路訓練得夠久，就會逼近它。（L1 損失得到的則是逐像素的後驗中位數：有時比較銳利，但仍是一個像素一個像素地決定，而不是一張連貫的影像；DLCV 09 比較了各種損失。）如果用 MSE 或 PSNR 來打分，後驗平均就是正確答案；如果你要的是看起來真實的影像，它就是錯誤答案：把邊緣位置各不相同的影像平均起來，得到的影像根本沒有銳利的邊緣。

程式：平均很模糊、每個樣本都很銳利的玩具問題

我們需要一個能精確算出後驗的問題。這裡的乾淨「影像」是 64 個像素的一維訊號，帶有一個階梯邊緣（左邊是 0，右邊是 1），邊緣位置在第 8 到第 55 個像素之間、未知，且每個位置的機率相同。退化是：用標準差 3 個像素的 Gaussian 模糊，每 8 個像素只保留 1 個，再加上標準差 0.2 的 Gaussian 雜訊。乾淨訊號只有 48 種可能，所以後驗就是一份精確的 48 個機率的清單。

```
import numpy as np
from scipy.ndimage import gaussian_filter1d
from scipy.stats import wasserstein_distance

rng = np.random.default_rng(0)
n, factor, sigma = 64, 8, 0.2
```

```
positions = np.arange(8, 56)                                          # where the edge may be
X = (np.arange(n)[None, :] >= positions[:, None]).astype(float)  # every possible clean
 ↪  signal
A = lambda x: gaussian_filter1d(x, 3.0, axis=-1, mode="nearest")[..., ::factor]  # blur, keep
 ↪  1 of 8
AX = A(X)                                                             # every candidate, degraded

def posterior(y):
    """Exact p(edge position | y): uniform prior, Gaussian noise of std sigma."""
    logp = -np.sum((y - AX) ** 2, axis=1) / (2 * sigma**2)
    p = np.exp(logp - logp.max())
    return p / p.sum()

truth, mean, post, prior = [], [], [], []
for _ in range(2000):
    x = X[rng.integers(len(positions))]
    y = A(x) + sigma * rng.standard_normal(n // factor)
    p = posterior(y)
    truth.append(x)
    mean.append(p @ X)                                                # posterior mean (MMSE)
    post.append(X[rng.choice(len(positions), p=p)])                   # a sample from p(x | y)
    prior.append(X[rng.integers(len(positions))])                     # a sample that ignores y
truth = np.array(truth)

for name, est in [("posterior mean", mean), ("posterior sample", post), ("prior sample",
 ↪  prior)]:
    est = np.array(est)
    mse = np.mean((est - truth) ** 2)
    w1 = wasserstein_distance(est.ravel(), truth.ravel())             # pixel values like real
     ↪  ones?
    jump = np.abs(np.diff(est, axis=1)).max(axis=1).mean()            # sharpest step (real: 1.0)
    print(f"{name:16s} MSE {mse:.4f} | W1 {w1:.4f} | largest step {jump:.2f}")
# posterior mean   MSE 0.0169 | W1 0.0243 | largest step 0.22
# posterior sample MSE 0.0333 | W1 0.0010 | largest step 1.00
# prior sample     MSE 0.2485 | W1 0.0076 | largest step 1.00
```

我們在 2,000 個隨機測試訊號上比較三種估計量。各欄分別是：與乾淨訊號的 MSE；一個粗略的「逼真度」分數，也就是輸出像素值的分布與真實像素值的分布之間的一維 Wasserstein 距離 W1（真實訊號只含 0 與 1，所以介於中間的灰階值是不真實的）；以及輸出中最銳利階梯的高度（每個真實訊號都是 1.0）。本篇程式碼區塊裡的所有數字，都來自我們自己的執行結果。

- **後驗平均**的 MSE 最低，但它最銳利的階梯平均只有 0.22：它是一條斜坡，不是邊緣，而且它的灰階值使它在統計上不像任何真實訊號。
- **後驗樣本**是一個真正的邊緣（階梯 1.00，W1 接近零），放在資料允許的位置上。它的 MSE 是 0.0333，幾乎恰好是後驗平均 0.0169 的**兩倍**。這個兩倍不是巧合（見下一節）。
- **先驗樣本**不理會 y。它完全逼真，卻毫無用處：它的 MSE 差了大約 15 倍。光有逼真度不等於復原。（它的 W1 是 0.0076 而不是 0，來自 2,000 個訊號的測試所含的取樣雜訊。）

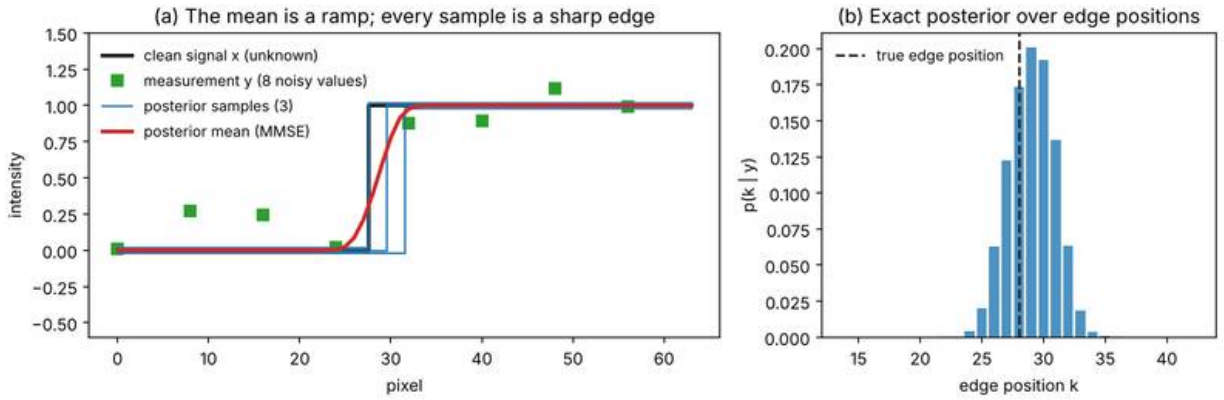


圖 *1* — 單一輸入的階梯邊緣玩具問題。(*a*) 後驗平均（紅色）是一條平滑的斜坡；每個後驗樣本（藍色）都是一個銳利的邊緣，但它們對邊緣在哪裡看法不一。(*b*) 邊緣位置的精確後驗：量測把位置縮小到幾個像素之內，卻無法精確釘死它。

「生成式」復原是什麼意思

白話版。生成式復原器選擇一個合理答案；理想後驗取樣器在反覆取樣下，應呈現給定輸入的條件不確定性。

精確版。生成式（或稱隨機性、後驗取樣）復原模型是一個隨機映射：給定 y 和一個隨機種子 z，它傳回 $\hat{x} = G(y, z)$，且在 y 固定時，其分布應該逼近 $p(x \mid y)$。本篇的每個方法都有三個要素：

1. **先驗** $p(x)$，也就是自然影像的分布，由生成模型（GAN、擴散模型、流、碼本）學得。
2. **概似或資料一致性項**，把輸出與輸入綁在一起，可以是明確的（已知 $\mathcal{D}$ 時用 $\|y - \mathcal{D}(\hat{x})\|^2$），也可以是隱含的（給網路看 y，再用成對資料訓練）。
3. **產生樣本的方法**：生成器的一次前向傳遞，或取樣器的許多個步驟。

有一個方便的恆等式把這三者連起來。對貝氏定理取對數、再對 x 取梯度，常數 $p(y)$ 就消失了：

$$\nabla_x \log p(x \mid y) = \nabla_x \log p(x) + \nabla_x \log p(y \mid x),$$

其中 $\nabla_x \log p(x)$ 是先驗的**分數**（score；一個指向更可能的影像的向量場），$\nabla_x \log p(y \mid x)$ 是概似的梯度（指向更能解釋 y 的影像）。所以每一個「沿著上坡方向走，再加上雜訊」的取樣器，都可以拆成先驗的推力與資料的推力。這個拆法是下面零樣本擴散復原的骨幹。

建構這類復原模型有兩大類做法：

- **直接學習後驗**（監督式）：像條件式 GAN 與條件式擴散模型那樣，用成對資料 (y, x) 訓練條件式生成器。不必把 $\mathcal{D}$ 寫出來，但遇到新的退化就需要重新訓練。
- **把先驗與已知的概似結合**（零樣本）：只用乾淨影像訓練一次無條件的生成模型，測試時再用 $\mathcal{D}$ 和 y 來引導它的取樣器，GAN 反演與 DPS 這類方法就是這樣做的。一個模型可以服務許多任務，但必須知道 $\mathcal{D}$，而且取樣通常比較慢。

感知–失真權衡

白話版。逼真樣本會對不確定細節作出選擇。在理想精確後驗假設下，取樣的期望 MSE 是最佳平均估計器的兩倍；學得的生成器沒有此保證。

精確版。 Blau 與 Michaeli [1] 對任何由 Y 輸出 $\hat{X}$ 的復原方法，定義它的**失真**（distortion）為 $\mathbb{E}[\Delta(X,\hat{X})]$，其中 Δ 是任何逐影像的失真度量，例如平方誤差；並把它的**感知品質**（perceptual quality）定義為真實影像分布 p_X 與復原影像分布 $p_{\hat{X}}$ 之間的某個散度 $d(p_X, p_{\hat{X}})$（為零表示輸出在統計上與真實影像無法區分，例如連完美的判別器也分不出來）。**感知–失真函數**為

$$P(D) = \min_{p_{\hat{X}|Y}} d(p_X, p_{\hat{X}}) \quad \text{subject to} \quad \mathbb{E}[\Delta(X,\hat{X})] \le D,$$

其中最小值是對所有（可能帶有隨機性的）復原方法 $p_{\hat{X}|Y}$ 取的，D 是失真預算。他們證明 $P(D)$ 是不遞增的，而且當 d 對第二個引數為凸函數時，它也是凸的 [1]。因此，在（失真、感知）平面的原點附近有一塊**達不到的區域**：把失真壓得超過某個點，就會迫使輸出偏離真實影像的分布。他們也證明這對任何失真度量都成立，不只是 MSE。

曲線上有兩個點最重要。令 $D_{\min}$ 是最低可能的失真（對平方誤差來說由 MMSE 估計量達成），$D_{\max}$ 是在感知品質完美（$p_{\hat{X}} = p_X$）時能達到的最低失真。對平方誤差，論文證明了（在其 arXiv 延伸版 [1] 中）

$$D_{\max} \le 2\, D_{\min},$$

而且這個界可以由**後驗取樣** $p_{\hat{X}|Y} = p_{X|Y}$ 達到。以 PSNR 來說，完美的逼真度所需付出的代價，絕不會超過大約 3 dB，因為 $10\log_{10} 2 \approx 3.01$。這正是我們玩具問題中的兩倍（0.0333 對 0.0169）。論文也指出這個界不一定是緊的：有些問題讓完美的逼真度所花的代價低於 MMSE 的兩倍。

僅線上版：為什麼後驗取樣恰好讓 *MSE* 變成兩倍 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/notes/為什麼後驗取樣恰好讓-mse-變成兩倍/*

圖 2 把玩具問題中的各個估計量放在這個平面上。在後驗平均與後驗樣本之間，我們也畫出簡單的混合 $(1-\lambda)\,\hat{x}_{\text{MMSE}} + \lambda\,\hat{x}_{\text{sample}}$，這是在兩者之間做取捨的一個簡單方法。它們的期望 MSE 是 $(1+\lambda^2)D_{\min}$，因為後驗樣本與平均的偏差，和真值與平均的偏差互不相關，所以 $\lambda = 1$ 得到兩倍。這些混合只是一條可以達到的路徑；真正的界 $P(D)$ 可能在它下方。

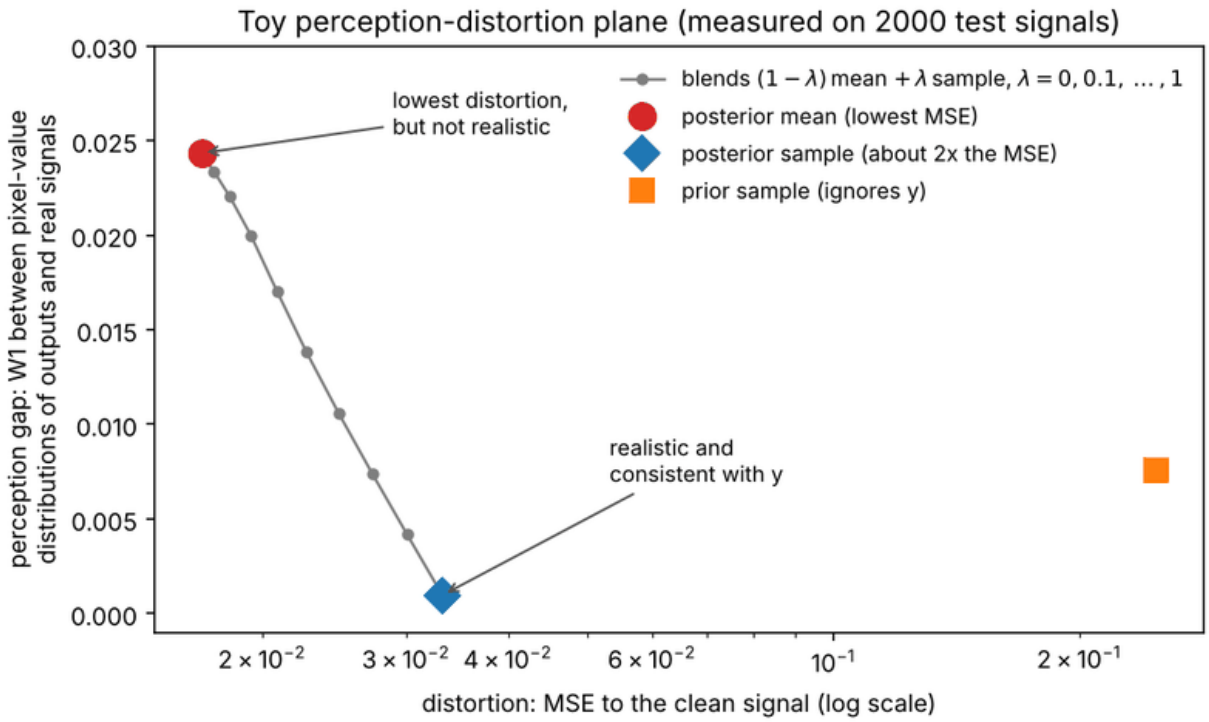


圖 2 — 階梯邊緣玩具問題的感知–失真平面，以 *2,000* 個測試訊號量測。從後驗平均換成後驗樣本，感知差距降到接近零，而 *MSE* 加倍。先驗樣本也很逼真，但它不理會輸入，所以遠遠落在右邊。

這在實務上的意義是：

- **方法是一個點，不是一個分數**。要同時報告一個失真指標（PSNR、SSIM）以及一個感知指標，並說明你的方法瞄準哪一個點。只用 PSNR 來比較 GAN 復原模型和 MSE 復原模型，是在比較兩個不同的目標。
- **「PSNR 更好、而且更逼真」只可能出現在離界線很遠的方法上**。一旦方法接近曲線，就只能沿著曲線移動。
- **許多生成式復原模型都提供一個旋鈕**，可以沿著曲線滑動：GAN 損失的權重、CodeFormer [10] 的保真度權重、StableSR [24] 的特徵包裝純量、DiffBIR [25] 的引導強度。選擇旋鈕，就是選擇權衡曲線上的一個點。

僅線上版：*Blau 與 Michaeli：The Perception-Distortion Tradeoff（CVPR 2018）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/papers/blau-與-michaelithe-perception-distortion-tradeoffcvpr-2018/*

第一族：以 GAN 為基礎的復原與 GAN 先驗

白話版。 GAN 加進了一位看過大量真實照片的評論者，只要復原影像看起來是假的，它就會抱怨。復原模型學著產生評論者能接受的輸出，這就把它從模糊的平均推向逼真的紋理。第二個想

法是重複利用一個本來就是為了生成人臉或場景而訓練好的 GAN：復原就變成「找出這個生成器能畫出、又與受損輸入相符的那張影像」。

對抗損失與感知損失

第一條路線維持監督式訓練，只改變損失（公式見 DLCV 09）。SRGAN [2] 把對抗損失和預訓練 VGG 特徵上的損失結合起來，沿用 Johnson 等人 [3] 的感知損失想法；ESRGAN [4] 改進了生成器與判別器；Real-ESRGAN [5] 用豐富的合成退化流程為真實照片訓練這類模型（DLCV 09A 有詳細介紹）。用本篇的說法，總損失

$$\mathcal{L} = \mathcal{L}_{\text{pixel}} + \lambda_p \, \mathcal{L}_{\text{feat}} + \lambda_a \, \mathcal{L}_{\text{adv}}$$

有一個像素項 $\mathcal{L}_{\text{pixel}}$（可分離的 L1 以逐座標後驗中位數為目標；平方 L2 以後驗平均為目標）、一個特徵項 $\mathcal{L}_{\text{feat}}$（比對深層特徵，能容許小幅位移）、以及一個對抗項 $\mathcal{L}_{\text{adv}}$（把輸出分布拉向真實影像），權重為 $\lambda_p, \lambda_a \geq 0$。Blau 與 Michaeli 指出，對抗項近似的是一個散度 $d(p_X, p_{\hat{X}})$，所以最小化這個損失，就是感知–失真問題的拉格朗日（Lagrangian）形式，而改變 λ_a 就是沿著曲線移動 [1]。

有兩點要注意。第一，大多數這類網路是**決定性**的，所以它們給的是一個合理的答案，而不是機率正確的樣本。第二，GAN 的訓練可能發生**模式崩塌**（ML 07），而且對抗項會獎勵判別器喜歡的任何紋理，包括場景中從未出現過的紋理。

預訓練 GAN 當作影像先驗

第二條路線把「影像長什麼樣子」和「這個輸入說了什麼」分開。在乾淨影像上訓練好的 GAN 生成器 G（例如在人臉上訓練的 StyleGAN [6]），把潛在碼 w 對應到影像 $G(w)$。它的輸出構成一個學得的**影像流形**（image manifold）：復原就可以在這個流形上搜尋與 y 相符的影像。

以最佳化進行 GAN 反演。求解

$$\hat{w} = \arg\min_{w} \| \mathcal{D}(G(w)) - y \|^2 + \lambda \, R(w), \qquad \hat{x} = G(\hat{w}),$$

其中 R 讓 w 留在生成器訓練時所見的區域內，λ 是它的權重。不必為每個任務訓練任何東西，而且 $\mathcal{D}$ 可以是任何可微分的退化。PULSE [7] 用這個做法做人臉超解析度，在 StyleGAN 的潛在空間裡搜尋縮小後會等於輸入的人臉；不同的隨機起點會得到不同的人臉，而且全都吻合，這直接呈現了「許多個答案」。

把生成器放進前向網路裡。為了避免每張影像都要做一次最佳化，GLEAN [8] 與 GFP-GAN [9] 把一個預訓練並凍結的 GAN 放在編碼器–解碼器之中：編碼器讀取受損影像，生成器提供逼真的特徵，一次前向傳遞就得到結果。CodeFormer [10] 則改用學得的人臉部件離散碼本，取代 GAN 的潛在空間，並且有一個在測試時以保真度換品質的旋鈕。

僅線上版：以 *GAN 為先驗的復原方法詳解* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/notes/以-gan-為先驗的復原方法詳解/*

長處與限制。GAN 先驗在生成器很好的窄領域（例如對齊過的人臉）中，一次前向傳遞就能給出銳利而逼真的結果。但它們只能產生生成器所能產生的東西：和訓練用人臉不像的輸入，會被

拉向「典型」的人臉，這是保真度與公平性的問題，PULSE 的作者也明確討論了這類偏差 [7]。一篇擴散復原的綜述 [44] 指出，訓練不穩定與紋理看起來假，是以 GAN 為基礎的復原最主要的兩個弱點，這也是接下來擴散先驗的動機之一。Xia 等人的綜述 [47] 回顧了 GAN 反演及其在復原與編輯上的應用。

第二族：以擴散模型為先驗

白話版。擴散模型是一個在每個雜訊程度上都訓練過的去雜訊器。從純雜訊出發，一次去掉一點雜訊，就能把雜訊變成逼真的影像。用在復原上，我們有兩種做法：要嘛教去雜訊器在去雜訊的同時看著受損的輸入（條件式擴散），要嘛拿一個只認得乾淨影像的去雜訊器，在每一小步去雜訊之後，輕推目前的猜測，讓它與受損輸入相符（零樣本後驗取樣）。

一頁看懂擴散

完整的推導在 ML 08；我們只需要四個事實，採用變異數保持（variance-preserving）的 DDPM 慣例 [11]。

前向加噪。對乾淨影像 x_0 與步數 $t = 1, \dots, T$，

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I),$$

其中 $\bar{\alpha}_t \in (0, 1)$ 沿著選定的排程，從約 1（幾乎乾淨）遞減到約 0（純雜訊），ϵ 是標準 Gaussian 雜訊。

去雜訊器就是分數估計器。一個訓練來由 x_t 預測 ϵ 的網路，會給出雜訊分布 p_t 的分數：$\nabla_{x_t} \log p_t(x_t) \approx -\epsilon_\theta(x_t, t)/\sqrt{1 - \bar{\alpha}_t}$（ML 08）[12]。

Tweedie 公式 [13] 把分數轉成乾淨影像的估計，

$$\hat{x}_0(x_t) = \mathbb{E}[x_0 \mid x_t] = \frac{x_t + (1 - \bar{\alpha}_t)\, \nabla_{x_t} \log p_t(x_t)}{\sqrt{\bar{\alpha}_t}},$$

其中 $\hat{x}_0$ 是給定雜訊影像時乾淨影像的後驗平均，所以當 t 很大時它是模糊的，正如第一節所說。

取樣。從 $x_T \sim \mathcal{N}(0, I)$ 出發，反覆從一個中心位於 x_t 與 $\hat{x}_0(x_t)$ 之間、變異數小而已知的 Gaussian 抽出 x_{t-1}（ML 08）。每一步花一次網路呼叫；經典的取樣器要用到數百步。用分數網路的退火 Langevin 動力學 [48]，以及連續時間的 SDE 觀點 [12]，則是用別的說法描述同一個想法。

程式：用精確分數做擴散。影像尺度的擴散需要訓練好的網路。在二維玩具問題上，我們可以略過網路：如果「影像」分布是 Gaussian 的混合，每個加噪後的邊際分布 p_t 仍是 Gaussian 的混合，它的分數有封閉形式。下面的先驗有四個團塊，權重為 0.1、0.2、0.3、0.4，標準差為 0.3。

```
import numpy as np

rng = np.random.default_rng(1)
mu = np.array([[-2.0, -2.0], [-2.0, 2.0], [2.0, -2.0], [2.0, 2.0]])  # a 2-D "image" prior:
w = np.array([0.1, 0.2, 0.3, 0.4])                                   # four Gaussian blobs
s2 = 0.3**2                                                          # each with variance s2
 ↪  * I
```

```
T = 200
beta = np.linspace(1e-4, 0.05, T)
alpha = 1 - beta
abar = np.cumprod(alpha)

def score(x, ab):
    """Exact grad log p_t(x) of the noised mixture, plus responsibilities r."""
    m, v = np.sqrt(ab) * mu, ab * s2 + (1 - ab)            # component k at time t: N(m_k, v
     ↪  I)
    d2 = ((x[:, None, :] - m[None]) ** 2).sum(-1)         # (N, K) squared distances
    logr = np.log(w) - d2 / (2 * v)
    r = np.exp(logr - logr.max(1, keepdims=True)); r /= r.sum(1, keepdims=True)
    return (r[:, :, None] * (m[None] - x[:, None, :])).sum(1) / v, r

def x0_hat(x, ab):                                          # Tweedie: E[x_0 | x_t]
    return (x + (1 - ab) * score(x, ab)[0]) / np.sqrt(ab)

def ddpm_step(x, t, x0):
    """Sample x_{t-1} from the Gaussian q(x_{t-1} | x_t, x_0 = x0)."""
    ab_prev = abar[t - 1] if t > 0 else 1.0
    mean = (np.sqrt(ab_prev) * beta[t] * x0 + np.sqrt(alpha[t]) * (1 - ab_prev) * x) / (1 -
     ↪  abar[t])
    var = beta[t] * (1 - ab_prev) / (1 - abar[t])
    return mean + np.sqrt(var) * rng.standard_normal(x.shape)

x = rng.standard_normal((4000, 2))                          # start from pure noise
for t in reversed(range(T)):
    x = ddpm_step(x, t, x0_hat(x, abar[t]))
_, r = score(x, 1.0)
print("signal left at t=T:", round(abar[-1], 5))
print("fraction per blob :", np.bincount(r.argmax(1), minlength=4) / len(x), "(target", w,
 ↪  ")")
print("spread inside blob:", round(float(np.sqrt(((x - mu[r.argmax(1)]) ** 2).mean())), 3),
 ↪  "(target 0.3)")
# signal left at t=T: 0.00612
# fraction per blob : [0.09675 0.2005  0.3075  0.39525] (target [0.1 0.2 0.3 0.4] )
# spread inside blob: 0.283 (target 0.3)
```

取樣器重現的團塊權重，誤差在取樣誤差之內。每個團塊內部的散布是 0.283，而不是 0.3：只有 200 個粗略的步驟時，反向過程會稍微太窄（用 1,000 個較小的步驟時，我們量到 0.296）。這是關於**步數**的第一個教訓：步數少比較便宜，但比較不精確。圖 3 顯示粒子雲從雜訊收縮成四個團塊的過程。

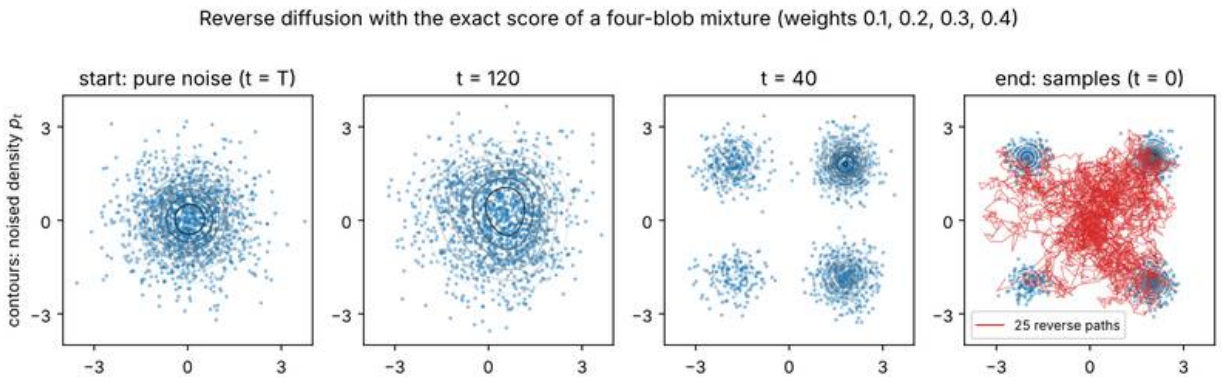


圖 *3* — 用四團塊先驗的精確分數做反向擴散。灰色等高線是各時刻雜訊後的密度；藍點是 *1,500* 個樣本；紅線（右）是 *25* 條個別的反向路徑。路徑之所以抖動，是因為每一步都注入新的雜訊。

路線 A：訓練條件式擴散復原模型

最直接的做法，是讓去雜訊器看到輸入：用成對資料訓練 $\epsilon_\theta(x_t, t, y)$，於是取樣器抽出的樣本逼近 $p(x \mid y)$。**SR3** [14] 把這個做法用在超解析度：把放大後的低解析度影像和雜訊影像一起交給 U-Net，取樣時從純雜訊一步一步精修。在論文針對 8× 人臉超解析度所做的人類研究中，受測者被 SR3 騙過的比例接近 50%，GAN 基準方法則至多是 34% [14]。**Palette** [15] 把同一套做法用在上色、影像修補、擴圖（uncropping）與 JPEG 復原上，並顯示單一多任務模型可以追上各任務專用的模型。

條件式模型需要成對資料，而且每一類退化都要重新訓練，不過它們能處理雜訊，也能處理難以寫成算子的退化（相機處理流程、壓縮）。從純雜訊開始，會浪費步數去重新製造 y 本來就有的東西；**ResShift** [16] 則建立一條在高解析度與低解析度影像之間、靠位移其殘差來移動的鏈，據報告只用 15 個取樣步驟就得到有競爭力的結果。

路線 B：用預訓練模型做零樣本後驗取樣

白話版。只用乾淨影像訓練一個無條件的擴散模型。要復原時，照常執行它的取樣器，但每走一步，就拿目前的乾淨影像猜測和輸入核對，並稍微修正一下。模型貢獻的是「看起來真實」，修正貢獻的是「與我們量測到的相符」。

精確版。把上一節的分數恆等式用在雜訊變數 x_t 上：

$$\nabla_{x_t} \log p_t(x_t \mid y) = \nabla_{x_t} \log p_t(x_t) + \nabla_{x_t} \log p_t(y \mid x_t),$$

其中第一項是預訓練模型的分數，第二項是**雜訊概似**（noisy likelihood）：雜訊狀態 x_t 對 y 的解釋有多好。第一項我們已經有了。第二項是難處，因為 y 取決於乾淨影像，而

$$p_t(y \mid x_t) = \int p(y \mid x_0)\, p(x_0 \mid x_t)\, dx_0$$

是把概似對所有可能產生 x_t 的乾淨影像取平均，對影像尺度的模型來說這是算不出來的。各種方法的差別主要就在於如何近似這一項，而它們最後都落到同一個迴圈（圖 4）：**先驗步驟**（用預訓

練模型做一步反向擴散），以及**資料一致性步驟**（朝 $\mathcal{D}(\hat{x}_0) \approx y$ 的方向做梯度步驟、投影或封閉形式的修正）。

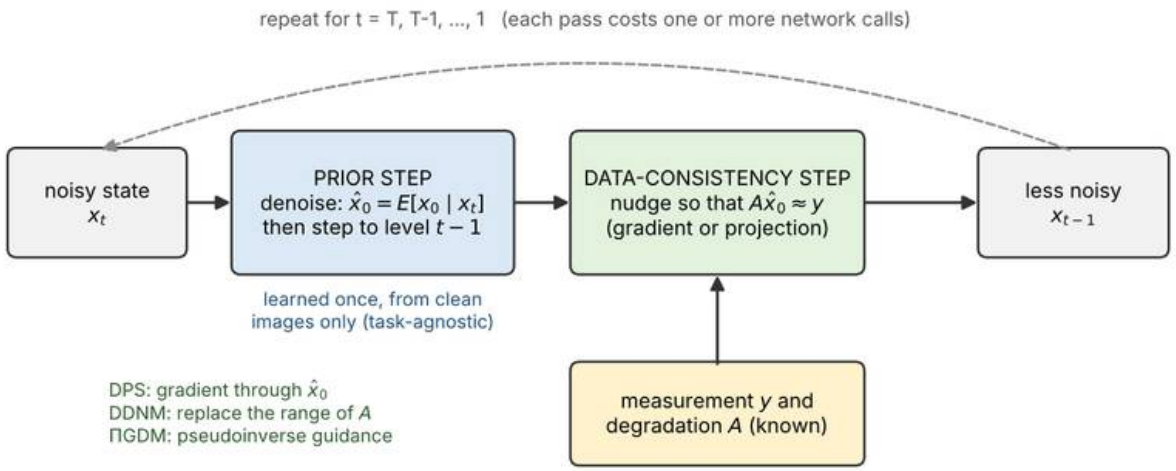


圖 4 — 零樣本擴散復原背後的迴圈。先驗步驟使用只在乾淨影像上訓練的模型；資料步驟使用已知的退化。各方法的差別，就在資料修正要怎麼做、在步驟中的哪個位置做。

DPS（diffusion posterior sampling）[17] 是最容易說明的。它把對 $p(x_0 \mid x_t)$ 的平均換成它的平均值，即 $p_t(y \mid x_t) \approx p(y \mid \hat{x}_0(x_t))$。在 Gaussian 雜訊下，這給出資料步驟

$$x_{t-1} \leftarrow x'_{t-1} - \zeta_t \nabla_{x_t} \|y - \mathcal{D}(\hat{x}_0(x_t))\|^2,$$

其中 x'_{t-1} 是一般先驗步驟的結果，$\zeta_t > 0$ 是步長；論文使用 $\zeta_t = \zeta'/\|y - \mathcal{D}(\hat{x}_0)\|$，$\zeta'$ 為常數。梯度要穿過 $\hat{x}_0$，所以每一步都需要**透過去雜訊器做反向傳播**。由於它從不精確地投影到量測上，DPS 也適用於含雜訊與非線性的退化（論文包含 Poisson 雜訊、Fourier 相位還原與非均勻去模糊）。

其他成員主要在資料步驟上有所不同：

- **DDRM**（denoising diffusion restoration models）[18] 在退化算子奇異值分解的座標下處理線性退化，對有量測到與沒量測到的座標採取不同的處理；不需反向傳播，步數也少。
- **DDNM**（denoising diffusion null-space model）[19] 把 $\hat{x}_0$ 中由量測決定的部分直接從資料複製過來，即 A^+y（A^+ 是線性算子 A 的偽逆），只在量測看不到的部分保留模型的猜測。
- **ΠGDM**（pseudoinverse-guided diffusion models）[20] 用退化算子的偽逆，加上對 $\hat{x}_0$ 的明確不確定性來建構引導；它會透過去雜訊器反向傳播，但不透過退化算子反向傳播，所以也涵蓋像 JPEG 這樣不可微分的量測。
- **DiffPIR** [21] 把即插即用的分裂法（輪流做資料步驟與去雜訊步驟，如 DLCV 09 所述）放進擴散取樣器裡，據報告最多用 100 次網路評估就有不錯的結果。

僅線上版：零樣本擴散取樣器有何不同 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/notes/零樣本擴散取樣器有何不同/*

程式：在後驗已知的問題上做 DPS 式取樣器。我們沿用四團塊先驗，並觀測二維「影像」的一

個線性投影 $y = Ax + n$，其中 $A = [1,1]/\sqrt{2}$，雜訊標準差 $\sigma_y = 0.2$。位於 $(-2,2)$ 與 $(2,-2)$ 的團塊都有 $Ax \approx 0$，所以接近 0 的量測確實是模稜兩可的。先驗和概似在每個團塊內都是 Gaussian，因此精確後驗仍是四團塊的混合，我們可以拿它來檢驗取樣器。$\hat{x}_0$ 的 Jacobian 以有限差分計算，這在二維很便宜；在影像尺度上，這一步就是反向傳播。

```
# Continues the previous snippet (same session): mu, w, s2, T, abar, x0_hat, ddpm_step.
rng = np.random.default_rng(3)
A = np.array([[1.0, 1.0]]) / np.sqrt(2)          # we only observe one projection of x
sig_y = 0.2
x_true = np.array([1.8, -2.1])
y = A @ x_true + sig_y * rng.standard_normal(1)

# Reference: the exact posterior of a Gaussian mixture under a linear-Gaussian measurement.
S = (A @ A.T)[0, 0] * s2 + sig_y**2              # variance of y within one blob
logpost = np.log(w) - (y - mu @ A.T)[:, 0] ** 2 / (2 * S)
post_w = np.exp(logpost - logpost.max()); post_w /= post_w.sum()
gain = s2 * A.T / S                              # Kalman gain, shape (2, 1)
post_mu = mu + (y - mu @ A.T) @ gain.T           # posterior mean inside each blob
post_cov = s2 * (np.eye(2) - gain @ A)
k = rng.choice(4, size=2000, p=post_w)
exact = post_mu[k] + rng.multivariate_normal(np.zeros(2), post_cov, size=2000)
mmse = post_w @ post_mu                          # posterior mean
least_sq = (np.linalg.pinv(A) @ y).ravel()       # minimum-norm solution of A x = y

def data_grad(x, ab, h=1e-4):
    """grad_x ||y - A x0_hat(x)||^2 through Tweedie's estimate (finite-difference
    ↪ Jacobian)."""
    J = np.stack([(x0_hat(x + h * e, ab) - x0_hat(x - h * e, ab)) / (2 * h) for e in
    ↪ np.eye(2)], -1)
    resid = y - x0_hat(x, ab) @ A.T               # (N, 1)
    return -2 * np.einsum("nij,ik,nk->nj", J, A.T, resid), np.abs(resid)

x = rng.standard_normal((2000, 2))
for t in reversed(range(T)):
    x_prior = ddpm_step(x, t, x0_hat(x, abar[t]))  # 1) prior step: one reverse-diffusion
    ↪ step
    g, res = data_grad(x, abar[t])
    x = x_prior - 0.1 / (res + 1e-8) * g          # 2) data-consistency step, DPS step size

blob = lambda pts: np.bincount(np.argmin(((pts[:, None] - mu) ** 2).sum(-1), 1), minlength=4)
↪ / len(pts)
print("least squares", least_sq.round(2), "| posterior mean", mmse.round(2), "| truth",
↪ x_true)
for name, pts in [("exact posterior", exact), ("DPS", x)]:
    print(f"{name:15s} blob shares {blob(pts)} | median |y - Ax| {np.median(np.abs(y - pts @
    ↪ A.T)):.3f}"
          f" | mean sq. error {np.mean(((pts - x_true) ** 2).sum(1)):.2f}")
for name, est in [("least squares", least_sq), ("posterior mean", mmse)]:
    print(f"{name:15s} sq. error {np.sum((est - x_true) ** 2):.2f}")
# least squares [0.14 0.14] | posterior mean [ 0.5 -0.3] | truth [ 1.8 -2.1]
# exact posterior blob shares [0.    0.409 0.591 0.   ] | median |y - Ax| 0.122 | mean sq.
↪ error 12.97
# DPS             blob shares [0.     0.4055 0.5945 0.    ] | median |y - Ax| 0.100 | mean
↪ sq. error 13.09
```

```
# least squares   sq. error 7.77
# posterior mean  sq. error 4.93
```

請對照圖 5 來讀輸出：

- **最小平方**解 $A^{+}y$ 與**後驗平均**都落在兩個合理團塊的中間，那裡的先驗幾乎是空的：這就是這個玩具問題裡「模糊」的答案。後驗平均的平方誤差最低（4.93），因為它採取了避險的做法。
- **DPS 式樣本**落在右邊的兩個團塊，占比 0.41 與 0.59，與精確後驗吻合。它們的均方誤差（13.09）高得多，因為約 40% 的樣本落在另一個團塊：一個很有把握、卻是錯的答案。這是後驗取樣照預期運作的結果，也是幻覺細節的二維版本。
- DPS 對量測的擬合比真正的後驗稍微更緊（殘差中位數 0.100 對 0.122）：它是一種近似，它的殘差取決於步長 ζ'（練習 3）。在高維度中，這類偏差要難發現得多。

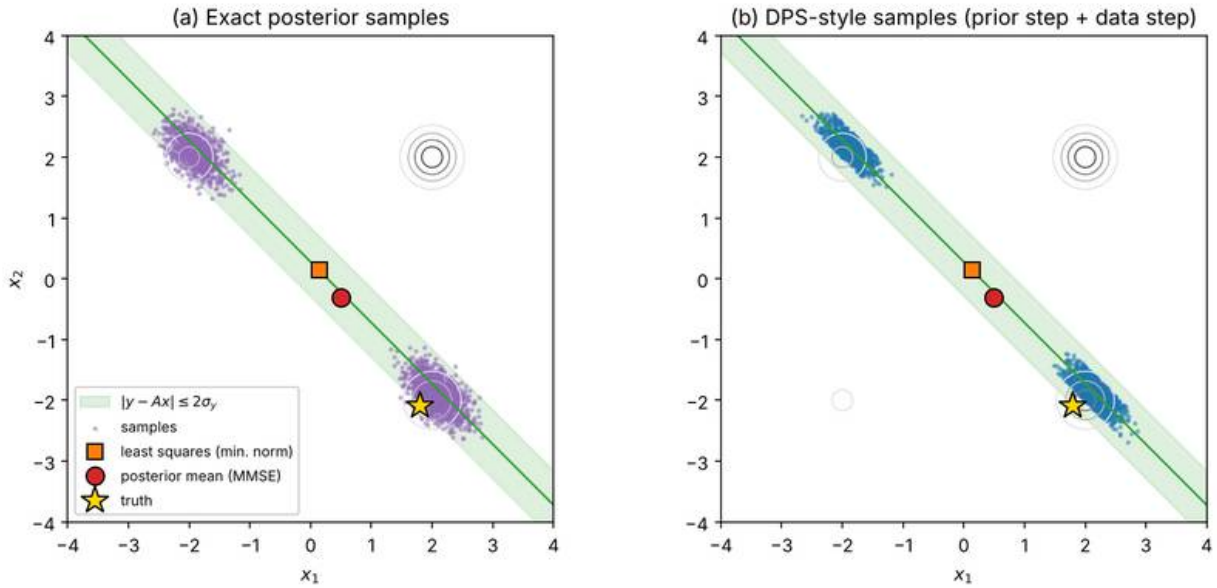


圖 5 — 後驗已知的線性逆問題。(*a*) 精確後驗樣本；(*b*) 來自圖 4 迴圈的 *DPS* 式樣本。兩者都落在量測所允許的兩個團塊，而最小平方與後驗平均則落在兩者之間的空白處。

僅線上版：*Chung* 等人：*Diffusion Posterior Sampling for General Noisy Inverse Problems*（*ICLR 2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/papers/chung-等 人 diffusion-posterior-sampling-for-general-noisy-inverse-problemsiclr-2023/*

條件式與零樣本如何選擇

問題	條件式（SR3、Palette、ResShift）	零樣本（DPS、DDRM、DDNM、ΠGDM、DiffPIR）
訓練	每一類退化一個模型，用成對資料	在乾淨影像上訓練一個無條件模型

問題	條件式（SR3、Palette、ResShift）	零樣本（DPS、DDRM、DDNM、ΠGDM、DiffPIR）
測試時的退化	隱含；只要訓練時出現過，可以是未知的	必須已知（或估計出來），且通常要可微分
雜訊與真實相機流程	訓練時有涵蓋就能處理	只能處理到概似模型所允許的程度
每張影像的成本	數十到數百步，不需反向傳播	通常數百步；DPS 類方法還要反向傳播
適合的情境	一個任務，且有大量訓練配對	物理已知、新的算子、不想重新訓練

一個粗略的準則：如果你能針對你的退化收集或合成配對，就訓練條件式模型（或非生成式的模型）。如果算子是已知的物理（模糊核、取樣遮罩、斷層掃描投影器），而且每換一種設定就重新訓練並不實際，零樣本取樣就很有吸引力。

第三族：潛在擴散與基礎模型復原器

白話版。最強的影像生成器，是用龐大影像集訓練出來的大型文字生圖擴散模型。它們對草地、布料和磚牆長什麼樣子的了解，遠超過任何復原資料集所能教的。近年的真實世界復原模型，會把這樣的模型凍結（或幾乎凍結），加上一個小網路讓它看到受損影像，再讓它把缺少的細節畫進去。結果可以非常驚人，也可能憑空捏造東西。

建立在預訓練的文字生圖模型上

潛在擴散（latent diffusion）[22] 不在像素上，而是在自編碼器壓縮後的潛在空間中執行擴散，這使大型文字生圖模型變得負擔得起（ML 08）。ControlNet [23] 示範了怎麼在這樣的模型上加入新的影像條件，而不破壞它（其作者描述這個模型是用數十億張影像預訓練的）：預訓練模型被鎖住，它編碼層的一個可訓練副本接收條件，兩者用零初始化的卷積連接，所以訓練是從未改變的模型開始。復原方法兩個想法都借用了。**StableSR** [24] 固定生成器，透過一個時間感知的編碼器注入低品質影像，並提供一個使用者設定的純量，在品質與保真度之間取捨。**DiffBIR** [25] 先用一般的復原網路移除退化，再用 ControlNet 式的轉接器重新生成遺失的細節，同樣有可調的保真度引導。**SUPIR** [26] 把先驗與資料的規模都放大（論文描述了 2,000 萬張附文字標註的高解析度影像），並讓文字提示來引導復原。

更少的步數：從數百步到一步

多步的潛在復原模型很慢，而且它們隨機的起始雜訊讓每次執行的輸出都不同。蒸餾與重新設計的模型把這兩者都降低了：**SinSR** [27]、**OSEDiff** [28] 與 **TSD-SR** [30] 一步就完成復原，**InvSR** [29] 則是一到五步，做法主要是從低品質影像而不是純雜訊出發，並蒸餾一個多步的教師模型。一般的蒸餾家族（漸進式、一致性、對抗式）在 ML 08 中有說明。

僅線上版：潛在擴散與一步復原方法逐一介紹 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/notes/潛在擴散與一步復原方法逐一介紹/*

兩個後果。第一，從輸入出發的一步復原模型幾乎是決定性的：它像 GAN 一樣給出一個合理的答案，而不是散布正確的樣本。第二，步數不是成本的全部：一步模型的網路可能更大，而且潛

在方法還要在完整解析度上負擔自編碼器的編碼器與解碼器。

幻覺問題

白話版。強大的先驗用典型的東西填補空缺，而不是用原本在那裡的東西。輸入說得很少的地方，輸出大多是先驗：它看起來很真實，影像裡也沒有任何線索告訴你它是捏造的。它會出現在這些地方：

- **文字與符號**。招牌、車牌和文件上的字母非常具體：一個小錯誤就會改變意思。生成式復原模型可以產生乾淨、清晰可讀、卻是錯的字元。
- **人臉與身分**。人臉先驗產生的是訓練分布中的一張臉；對特定的人來說，鼻子或眼睛稍有不同，就是身分改變。
- **紋理**。磚牆、布料和樹葉可能被另一種不同、但同樣合理的圖案取代。
- **潛在解碼器**。潛在復原模型的自編碼器本身就有重建誤差；把乾淨影像編碼再解碼，就能看出光是它就改變了什麼，正如 ML 08 所建議的。
- **醫學與科學影像**。一個並不存在的合理結構，或被抹平的真實病灶，都是診斷錯誤。Bhadra 等人 [35] 在斷層掃描重建中分析了這件事：他們把一個估計拆成由量測決定的部分，以及由先驗補上的**零空間**（null-space）部分，並據此定義了幻覺圖。

零空間的觀點用一行就能說明機制，它用的是 DDNM 背後的分解：對線性的 A，任何估計 $\hat{x}$ 都滿足 $\hat{x} = A^{+}A\hat{x} + (I - A^{+}A)\hat{x}$。資料一致性只限制第一個部分。第二個部分的一切都來自先驗，而當 A 丟掉很多資訊時，這個部分就很大，所以**一個完全符合資料一致性的輸出，仍然可能大部分都是捏造的**。與 y 一致是必要條件，永遠不是充分條件。

幻覺的量測本身就是一個研究主題。最近一項針對生成式超解析度的研究主張，幻覺的細節無論用保真度指標或無參考品質模型，都很難捕捉到，並提出了一種「幻覺分數」（Hallucination Score），做法是對多模態大型語言模型下提示，其作者發現這個分數與人類判斷高度吻合 [36]。

第四族：流匹配與整流流復原器

白話版。流匹配學習一個平滑的「速度場」，沿著幾乎筆直的路徑把雜訊送到影像，所以只需要很少的步數。用在復原上，可以把一個模糊的最佳猜測送到逼真的影像，也可以把預訓練的流接到和擴散相同的「先驗 + 資料」迴圈裡。

流匹配 [31] 與整流流 [32] 訓練網路預測一條路徑的速度，路徑連接一個簡單分布與資料分布；樣本來自對常微分方程的積分，而筆直的路徑只需要很少的積分步數（ML 08）。有兩個經過驗證的復原用法：

- **PMRF**（posterior-mean rectified flow）[33] 直接瞄準感知–失真一節的 $D_{\max}$ 這個點：在輸出分布與真實影像相同的估計量中，MSE 最低的那一個。依據論文引用的一項理論結果，它先預測**後驗平均**（一個一般的 MMSE 網路），再用一個逼近最佳傳輸映射的整流流模型，把這個模糊的估計**傳輸**到逼真的影像。這是理解整篇內容的一種乾淨的方式：先得到最佳的平均，再盡可能少地移動它，讓它看起來真實。
- **PnP-Flow** [34] 由預訓練的流匹配模型定義出隨時間變化的去雜訊器，並交替進行三件事：對資料保真項做梯度步驟、重新投影回學得的流路徑，以及去雜訊：這是圖 4 的迴圈，只是把擴

散模型換成流，並避免了透過 ODE 做反向傳播。

評估生成式復原

白話版。分開問四個問題：輸出與真值接近嗎（有真值時）？它看起來真實嗎？它與輸入相符嗎？再執行一次，它會改變多少？一個數字無法回答這四個問題，而一張漂亮的圖一個也回答不了。

指標的定義在 DLCV 09。對生成式復原模型，請這樣使用它們。

1. **有參考影像時的保真度**。 PSNR 與 SSIM [39]（低失真）、LPIPS [38]（與參考影像的感知距離）。在相同資料模型下，給定觀測後與真值條件獨立的精確後驗樣本，其期望平方誤差是 MMSE 估計量的兩倍 [1]。只有用這兩個期望 MSE 計算 PSNR 時，才得到約 3 dB 的差距；逐張 PSNR 再取平均不一定如此。學得的生成器與訓練出的迴歸基準不一定符合這些理想假設。應測量實際權衡，不能只靠較低的 PSNR 證明更逼真。
2. **逼真度**。 NIQE [41]、MUSIQ [42] 與 CLIP-IQA [43] 這類無參考指標替單張影像打分；FID [40] 則比較輸出與真實影像集合之特徵分布的高斯近似。它是實用的分布比較代理量，不是完整影像分布 p_X 與 $p_{\hat{X}}$ 之間的精確距離，而且需要很多影像。無參考指標可以靠加進紋理來作弊，所以千萬不要只用它們。
3. **與輸入的一致性**。重新套用退化再比較：$\|y - \mathcal{D}(\hat{x})\|$ 應該約等於雜訊水準。這不需要參考影像，並且能抓出忽略輸入的輸出。但它抓不到零空間的幻覺（見前面的小節）。
4. **隨機性**。取樣器給的是輸出的分布。請把它報告出來。

報告多個樣本。這段程式碼延續階梯邊緣玩具問題：一個測試輸入、每次執行八個復原結果、三次使用不同隨機種子的執行，就像同一方法的三位使用者會看到的情況。

```
# Recreate the step-edge toy: the intervening DPS example uses a different A.
import numpy as np
from scipy.ndimage import gaussian_filter1d
n, factor, sigma = 64, 8, 0.2
positions = np.arange(8, 56)
X = (np.arange(n)[None, :] >= positions[:, None]).astype(float)
A = lambda x: gaussian_filter1d(x, 3.0, axis=-1, mode="nearest")[..., ::factor]
AX = A(X)

def posterior(y):
    logp = -np.sum((y - AX)**2, axis=1) / (2 * sigma**2)
    p = np.exp(logp - logp.max())
    return p / p.sum()

rng = np.random.default_rng(5)
x = X[20]                                                    # a test signal: edge at pixel 28
y = A(x) + sigma * rng.standard_normal(n // factor)
p = posterior(y)
for seed in range(3):                                        # three "users" with different
↪   seeds
    r = np.random.default_rng(seed)
    S = X[r.choice(len(positions), size=8, p=p)]             # 8 restorations of the same
     ↪  input
    err = ((S - x) ** 2).mean(axis=1)
```

```
    print(f"seed {seed}: MSE mean {err.mean():.4f} ± {err.std():.4f} | best of 8
     ↪ {err.min():.4f}"
          f" | edges at {sorted(set(np.argmax(S, axis=1).tolist()))}")
S = X[rng.choice(len(positions), size=200, p=p)]
std = S.std(axis=0)                                          # per-pixel spread across samples
print("pixels with std > 0.1:", np.flatnonzero(std > 0.1), "| true edge at", np.argmax(x))
# seed 0: MSE mean 0.0273 ± 0.0268 | best of 8 0.0000 | edges at [23, 24, 26, 27, 28, 29]
# seed 1: MSE mean 0.0273 ± 0.0103 | best of 8 0.0156 | edges at [25, 26, 27, 29, 30]
# seed 2: MSE mean 0.0293 ± 0.0213 | best of 8 0.0000 | edges at [24, 25, 26, 27, 28]
# pixels with std > 0.1: [22 23 24 25 26 27 28 29 30] | true edge at 28
```

三個教訓，各附一條報告規則：

- **散布和平均一樣大**。在單一輸入上，每個樣本的 MSE 變動的幅度，和它的平均值差不多。要報告樣本之間（以及不同種子之間）的平均與散布，而不是只報告一個樣本。
- **「K 個中取最佳」是神諭（oracle）**。挑出最接近真值的樣本，在三個種子中有兩個得到零 MSE，這個數字沒有任何使用者能得到，因為使用者沒有真值。如果論文報告 K 取最佳，就必須明說，也不該拿來和單一樣本的結果比較。（例如 DPS 在相位還原上報告四個樣本中的最佳者，這是該問題的通行做法，並且有明說 [17]。）
- **樣本散布是診斷線索，不是經過校準的誤差圖**。在這個精確後驗玩具問題中，200 個樣本的逐像素標準差在邊緣附近很大（第 22 到 30 個像素）。學得的取樣器卻可能反覆產生相同的錯誤細節，讓低散布與高誤差同時出現；額外取樣也有運算成本。Angelopoulos 等人 [37] 用保留資料校準影像值預測集合：以指定的校準信心水準，限制來自相同分布的新影像中，未被區間涵蓋像素比例的期望值。這不保證每張個別影像的每個像素都涵蓋真值，也不保證任意分布偏移下仍然成立。

運算成本。要報告網路評估次數（NFE，包含引導與任何額外的傳遞）、解析度、硬體與實際耗時，對潛在方法還要報告自編碼器的成本。舉一個具體的參照：DPS 論文報告，在一張 RTX 2080 Ti GPU 上、每次 1,000 次網路評估，每張 256 × 256 的 FFHQ 影像約 95 秒，每張 ImageNet 影像約 600 秒 [17]。蒸餾後的一步復原模型把步數降到一步，但要比較的是完整流程，而不是步數。

一份最起碼誠實的報告：在有參考影像時報告 PSNR/SSIM 與 LPIPS；兩三個來自不同家族的無參考指標，再加上足夠大的集合上的 FID；重新退化後的殘差；樣本與種子之間的平均與散布；NFE 與耗時；重大主張要有使用者研究；以及一個包含失敗案例、而不是只有最好看結果的圖庫。

失效模式，以及何時不該使用生成式復原

白話版。當目標是一張令人愉快、合理的圖片時，生成式復原是對的工具。當這張圖片要被當作證據時，它就是錯的工具。

典型的失效模式。

- **捏造的具體內容**：文字、數字、人臉、小物件、醫學發現（見幻覺一節）。
- **忽略輸入**：資料一致性太弱，尤其是在強烈朝向先驗或提示引導時，會給出一個略有不同的場景的逼真影像；色偏是擴散超解析度已知的問題 [46]。

- **假設的退化與真實的退化不符**：零樣本方法需要正確的 $\mathcal{D}$。錯誤的模糊核或雜訊水準會被資料步驟傳遞下去，而先驗再讓錯誤的答案看起來很有說服力。
- **分布外的輸入**：在人臉或自然照片上訓練的先驗，會把顯微鏡、衛星或醫學影像拉向錯誤的「典型」外觀，也會把不像訓練資料的人臉拉向典型的人臉（PULSE 討論過的偏差 [7]）。
- **不同次執行與不同尺寸之間的不穩定性**：不同種子給出不同的細節，而在固定尺寸上訓練的模型，處理較大的影像需要特殊做法（StableSR 的漸進聚合取樣就是一種解法 [24]）。
- **成本**：每張影像要數百次網路呼叫，或是一個非常大的一步網路。

何時不該使用它，或只在明確標示並附上不確定性時才使用：

- **證據與量測**：鑑識、法律、新聞與監視用途、車牌、要被閱讀的文件，以及任何有人會依據某個細節採取行動的情況。偏好 MMSE 復原模型（平滑，但不會對具體細節下定論），或是顯示一組樣本並附上不確定性圖。
- **醫學與科學**：診斷、定量與任何下游量測。使用針對該任務驗證過的方法，並做像 [35] 這樣的幻覺分析，同時讓量測到的資料保持可見。
- **輕微的退化**：當後驗很窄（輕微雜訊、乾淨影像的 2× 超解析度）時，迴歸網路本來就能給出銳利的答案，而且更快、風險更小。
- **餵給其他演算法的流程**（偵測、OCR、量測）：復原應該用下游任務來評判；捏造的細節可能造成誤偵測。

一條簡短的決策規則：如果一個錯誤但逼真的細節，比一塊模糊的區域更糟，那就不要在沒有不確定性報告的情況下使用生成式復原模型。

現代觀點

綜述論文。

- **Li 等人，“Diffusion Models for Image Restoration and Enhancement: A Comprehensive Survey”（IJCV 2025）** [44]。把擴散式復原分成**監督式**方法（依取樣器的條件化方式分組）與**零樣本**方法（投影、分解與後驗估計），另外涵蓋盲復原與真實世界復原，並在數個任務上比較各方法。重點：感知品質最好，而取樣成本、模型大小、真實世界退化和評估是主要的障礙。
- **Daras 等人，“A Survey on Diffusion Models for Inverse Problems”（arXiv 2024）** [45]。涵蓋不再額外訓練、直接使用預訓練擴散模型的方法，依它們處理量測項的方式分成四個家族：對 $\nabla \log p_t(y \mid x_t)$ 的**明確近似**（DPS、ΠGDM 與 DDRM 一系屬於此類）、**變分推論**、**CSGM 型**的潛在最佳化，以及**漸近精確**的 Monte Carlo 方法；第二套分類則依一致性如何被強制（梯度、投影、取樣、最佳化）來歸類。重點：許多看似不同的方法，其實都是計算一個量測誤差，再以不同方式把它「提升」回影像空間，而這個領域缺乏一個衡量品質對運算量的標準基準。
- **Moser 等人，“Diffusion Models, Image Super-Resolution, and Everything: A Survey”（IEEE TNNLS 2025）** [46]。專注於超解析度：基礎、輸入域（像素或潛在空間）、條件化、引導、退化空間與零樣本方法。重點：擴散縮小了與人類偏好之間的大部分差距，而運算成本、論文之間的可比性、可解釋性與色偏仍是未解決的問題。
- **Xia 等人，“GAN Inversion: A Survey”（IEEE TPAMI 2023）** [47]。回顧影像如何被映射回預訓練 GAN 的潛在空間，以及這如何支援復原與編輯。重點：第一族的 GAN 先驗方法都是反演的特例，它們的限制在於生成器的涵蓋範圍與反演的成本。

僅線上版：*Li* 等人：*Diffusion Models for Image Restoration and Enhancement: A Comprehensive Survey*（*IJCV 2025*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/papers/li-等 人 diffusion-models-for-image-restoration-and-enhancement-a-comprehensive-surveyijcv-2025/*

2023–2026 年的最新進展。

1. **預訓練的文字生圖模型成為真實世界復原的預設先驗**，透過轉接器連接（StableSR、DiffBIR、SUPIR），並提供明確的保真度旋鈕，因為不受控制的生成會偏離輸入。
2. **步數從數百步降到一步**，靠的是殘差位移鏈、蒸餾與反演（ResShift、SinSR、OSEDiff、InvSR、TSD-SR）。一步模型用後驗的散布換取速度與可重複性。
3. **零樣本後驗取樣成熟為一個家族**，有共通的結構（DPS、DDRM、DDNM、ΠGDM、DiffPIR），並延伸到流（PnP-Flow），也有統整這些近似的綜述 [45]。
4. **理論重新進入實務**：感知–失真的界現在直接指引方法設計，例如 PMRF 的「先取後驗平均，再做傳輸」[33]。
5. **幻覺開始被量測**，而不只是被提及，例如使用專門的分數 [36]。

尚未解決的問題。

- **量測忠實度**：目前沒有被接受的指標能說「既銳利又正確」。保真度指標會懲罰所有細節；無參考指標則獎勵捏造的細節。
- **影像尺度上真正的後驗取樣**：零樣本方法是近似的（我們的玩具問題即使在二維也顯示出小小的偏差），而少步模型的多樣性很低。對高解析度影像給出經過校準的不確定性，大致上仍是開放問題。
- **未知的退化**：零樣本方法需要 $\mathcal{D}$；監督式方法需要正確的訓練分布。
- **成本**：用大型先驗做高解析度與影片復原，仍然很昂貴。
- **負責任的使用**：領域偏移、人臉先驗的族群偏差，以及生成內容的標示，在實務上都還沒有解決。

僅線上版：*DDNM*：保留量測成分，生成零空間內容 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/papers/ddnm 保留量測成分生成零空間內容/*；*Chung* 等人：*Diffusion Posterior Sampling for General Noisy Inverse Problems*（*ICLR 2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/papers/chung-等 人 diffusion-posterior-sampling-for-general-noisy-inverse-problemsiclr-2023/*；*BlindDPS*：共同推論影像與退化 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/generative-restoration/papers/blinddps 共同推論影像與退化/*

重點整理

- 最佳平方誤差估計是條件平均，平均多解結構可能造成模糊。
- 合理的生成細節仍可能與原始場景不同。
- 感知—失真界限針對理想分布與估計器 [1]，學得的生成器不會自動滿足條件。
- 區分依任務訓練的條件模型，以及搭配量測限制的預訓練先驗。
- 輸入一致性無法偵測未觀測成分中的虛構；減少取樣步數也不會消除多解性。
- 報告忠實度、真實感、一致性、隨機種子與完整成本，不隱藏多次取樣後挑最佳結果的做法。

- 當逼真但錯誤的細節會破壞用途時，選擇保守的復原方式。

練習

1. **用手算出那個兩倍**。在階梯邊緣玩具問題中，假設對某個輸入，後驗以機率 0.5 把邊緣放在第 30 個像素，以機率 0.5 放在第 32 個像素，而真值在第 30 個像素。計算後驗平均的 MSE（每像素，64 個像素），以及後驗樣本的期望 MSE。比值是 2 嗎？解釋為什麼定理中的兩倍是對各輸入取的平均，而不是對每個輸入都成立的保證。

   提示

   兩個候選只在第 30 與第 31 個像素不同。平均在那裡是 0.5，所以它的 MSE 是 $2 \times 0.25/64 \approx 0.0078$；樣本有 0.5 的機率是對的，否則在兩個像素上各錯 1，所以它的期望 MSE 是 $0.5 \times 2/64 \approx 0.0156$：比值為二。如果後驗改成把 0.9 放在錯誤的位置，這個真值的比值會是 $1.8/1.62 \approx 1.1$。定理是對 x 與 y 聯合取平均，在那裡，真值本身就是依後驗分布的。

2. **走一遍權衡**。用第一段程式碼，計算下列各項的 MSE 與 W1：(a) 後驗平均加上標準差 0.1 的 Gaussian 雜訊，(b) 後驗中位數的邊緣位置（選擇後驗累積機率第一次超過 0.5 的位置），以及 (c) 後驗眾數。把它們放在圖 2 的平面上。哪些靠近後驗樣本那個點？

   提示

   加雜訊並不會產生邊緣：在我們的執行中，最大的階梯只有 0.40，W1 增加到約 0.095（雜訊造成許多不真實的中間值），MSE 增加到 0.0274，所以 (a) 是遠離兩個目標。中位數與眾數都是單一的真實邊緣，所以它們的 W1 和樣本一樣接近零（我們的執行中分別是 0.0009 與 0.0011），而它們的 MSE（0.0245 與 0.0259）低於隨機樣本的（0.0338），因為它們從不挑選不太可能的位置。不過它們是決定性的：它們不會重現後驗的散布，而集合層級的指標可以偵測到這一點（相似輸入的所有輸出都一致）。

3. **DPS 的步長**。在 DPS 式的程式碼中，把資料步驟裡的常數 0.1 換成 0.02、0.5 與 2.0。對每一個，報告團塊占比、殘差 $|y - Ax|$ 的中位數與均方誤差，並與精確後驗比較。接著把正規化步驟 $0.1/(\text{res} + 10^{-8})\, g$ 換成未正規化的步驟 $0.2\, g$。你對 DPS 的步長與雜訊水準 σ_y 之間的關係有什麼結論？

   提示

   使用正規化的步驟時，不論殘差多大，每個資料步驟的長度大致相同，所以最後殘差中位數會落在那個常數附近（在我們的執行中，0.02、0.1、0.5 與 2.0 分別得到 0.021、0.100、0.496 與 1.994）。太大的值會讓殘差遠高於雜訊水準，並扭曲團塊占比（2.0 時是 0.24 與 0.76，精確值是 0.41 與 0.59）；0.02 時約有 1% 的樣本最後落入量測所排除的團塊，因為早期的修正太弱。未正規化的步驟 $0.2\, g$ 得到的殘差中位數是 0.007，比雜訊水準所允許的緊得多。兩種選擇都沒有像精確後驗那樣與 σ_y 綁在一起：步長是調出來的超參數，這是 DPS 只是一種近似的原因之一。

4. **步數與精確度**。在擴散的程式碼中，分別用 $T = 50$（並用 `beta = np.linspace(1e-4, 0.2, T)`）和 $T = 1000$（並用 `np.linspace(1e-4, 0.01, T)`）執行取樣器。記錄每個團塊內部的散布。把結果與成本的討論連起來，並說明為什麼一步的蒸餾復原模型是訓練出來的，而不是單純用更少的步數得到的。

提示

在我們的執行中，50 步的散布約為 0.263，200 步為 0.283，1,000 步為 0.296（目標 0.3）：即使用精確的分數，粗略的步驟也會讓反向過程太窄。一步模型不能靠對同一個取樣器跨一大步來達到精確度；它必須經過訓練（蒸餾，或用不同的方式建構），才能跳得正確。

5. **零空間的幻覺**。取一個長度 16 的一維訊號，並令 A 為相鄰兩項取平均（一個 8×16 的矩陣）。計算 $P = A^{+}A$，並驗證 P 是投影（$P^2 = P$）。構造兩個不同的訊號，量測 Ax 完全相同，一個平滑，另一個帶有銳利的棋盤格圖案。自然訊號的生成式先驗會挑哪一個？各個訊號的哪個部分是「捏造的」？

   提示

   兩兩平均的零空間，包含每一對之內符號交替的所有圖案 $(+a, -a)$（每對的振幅 a 任意）；把它加到平滑訊號上，Ax 不變。每一對之內的差，也就是 $(I - P)x$，資料完全看不到，全部由先驗提供：平滑訊號的先驗會挑小的，紋理先驗則可能加進一些。這就是 [35] 的幻覺圖想法。

6. **設計一個評估**。你必須決定，一步的潛在擴散復原模型能否取代以 MSE 訓練的網路，用來復原掃描的檔案文件。寫出評估方案：資料、指標、樣本與種子的數量、檢查幻覺字元的方法、運算量測，以及決策規則。

   提示

   盡可能使用有參考影像的真實掃描（用高解析度重新掃描），再加上乾淨文件的合成退化。報告 PSNR/SSIM/LPIPS、一個無參考指標，以及最關鍵的：兩種方法、以及多個種子相對於真實文字的 OCR 字元錯誤率；也報告兩個種子在某個字元上意見不一的頻率（不確定性訊號）。包含 NFE、耗時與記憶體，並計入自編碼器。依任務來決定：如果生成模型提高了字元錯誤率，或在不同種子之間改變了字元，就保留 MSE 網路來閱讀，只把生成式的用於顯示，並清楚標示。

參考文獻

1. Y. Blau and T. Michaeli, “The Perception-Distortion Tradeoff,” in *Proc. IEEE/CVF CVPR*, 2018, pp. 6228–6237 (extended version on arXiv). doi ·arXiv
2. C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” in *Proc. IEEE CVPR*, 2017, pp. 105–114. doi ·arXiv
3. J. Johnson, A. Alahi and L. Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution,” in *Proc. ECCV*, 2016, pp. 694–711. doi ·arXiv
4. X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao and C. C. Loy, “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks,” in *Proc. ECCV Workshops*, 2018 (LNCS, 2019), pp. 63–79. doi ·arXiv
5. X. Wang, L. Xie, C. Dong and Y. Shan, “Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data,” in *Proc. IEEE/CVF ICCV Workshops*, 2021, pp. 1905–1914. doi ·arXiv

6. T. Karras, S. Laine and T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks,” in *Proc. IEEE/CVF CVPR*, 2019, pp. 4396–4405. doi ·arXiv
7. S. Menon, A. Damian, S. Hu, N. Ravi and C. Rudin, “PULSE: Self-Supervised Photo Upsampling via Latent Space Exploration of Generative Models,” in *Proc. IEEE/CVF CVPR*, 2020, pp. 2434–2442. doi ·arXiv
8. K. C. K. Chan, X. Wang, X. Xu, J. Gu and C. C. Loy, “GLEAN: Generative Latent Bank for Large-Factor Image Super-Resolution,” in *Proc. IEEE/CVF CVPR*, 2021, pp. 14240–14249. doi · arXiv
9. X. Wang, Y. Li, H. Zhang and Y. Shan, “Towards Real-World Blind Face Restoration with Generative Facial Prior,” in *Proc. IEEE/CVF CVPR*, 2021, pp. 9164–9174. doi ·arXiv
10. S. Zhou, K. C. K. Chan, C. Li and C. C. Loy, “Towards Robust Blind Face Restoration with Codebook Lookup Transformer,” in *Proc. NeurIPS*, vol. 35, 2022, pp. 30599–30611. doi ·arXiv
11. J. Ho, A. Jain and P. Abbeel, “Denoising Diffusion Probabilistic Models,” in *Proc. NeurIPS*, 2020. proceedings ·arXiv
12. Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon and B. Poole, “Score-Based Generative Modeling through Stochastic Differential Equations,” in *Proc. ICLR*, 2021. arXiv
13. B. Efron, “Tweedie’s Formula and Selection Bias,” *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1602–1614, 2011. doi
14. C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet and M. Norouzi, “Image Super-Resolution via Iterative Refinement,” *IEEE TPAMI*, 2022 (early access). doi ·arXiv
15. C. Saharia, W. Chan, H. Chang, C. Lee, J. Ho, T. Salimans, D. Fleet and M. Norouzi, “Palette: Image-to-Image Diffusion Models,” in *Proc. ACM SIGGRAPH Conference*, 2022, pp. 1–10. doi · arXiv
16. Z. Yue, J. Wang and C. C. Loy, “ResShift: Efficient Diffusion Model for Image Super-resolution by Residual Shifting,” in *Proc. NeurIPS*, vol. 36, 2023, pp. 13294–13307. doi ·arXiv
17. H. Chung, J. Kim, M. T. McCann, M. L. Klasky and J. C. Ye, “Diffusion Posterior Sampling for General Noisy Inverse Problems,” in *Proc. ICLR*, 2023. arXiv
18. B. Kawar, M. Elad, S. Ermon and J. Song, “Denoising Diffusion Restoration Models,” in *Proc. NeurIPS*, vol. 35, 2022, pp. 23593–23606. doi ·arXiv
19. Y. Wang, J. Yu and J. Zhang, “Zero-Shot Image Restoration Using Denoising Diffusion Null-Space Model,” in *Proc. ICLR*, 2023. arXiv
20. J. Song, A. Vahdat, M. Mardani and J. Kautz, “Pseudoinverse-Guided Diffusion Models for Inverse Problems,” in *Proc. ICLR*, 2023. ICLR page
21. Y. Zhu, K. Zhang, J. Liang, J. Cao, B. Wen, R. Timofte and L. Van Gool, “Denoising Diffusion Models for Plug-and-Play Image Restoration,” in *Proc. IEEE/CVF CVPR Workshops*, 2023, pp. 1219–1229. doi ·arXiv
22. R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” in *Proc. IEEE/CVF CVPR*, 2022, pp. 10674–10685. doi · arXiv
23. L. Zhang, A. Rao and M. Agrawala, “Adding Conditional Control to Text-to-Image Diffusion Models,” in *Proc. IEEE/CVF ICCV*, 2023, pp. 3813–3824. doi ·arXiv
24. J. Wang, Z. Yue, S. Zhou, K. C. K. Chan and C. C. Loy, “Exploiting Diffusion Prior for Real-World Image Super-Resolution,” *International Journal of Computer Vision*, vol. 132, no. 12, pp. 5929–5949, 2024. doi ·arXiv

25. X. Lin, J. He, Z. Chen, Z. Lyu, B. Dai, F. Yu, Y. Qiao, W. Ouyang and C. Dong, "DiffBIR: Toward Blind Image Restoration with Generative Diffusion Prior," in *Proc. ECCV*, 2024 (LNCS), pp. 430–448. doi ·arXiv
26. F. Yu, J. Gu, Z. Li, J. Hu, X. Kong, X. Wang, J. He, Y. Qiao and C. Dong, "Scaling Up to Excellence: Practicing Model Scaling for Photo-Realistic Image Restoration In the Wild," in *Proc. IEEE/CVF CVPR*, 2024, pp. 25669–25680. doi ·arXiv
27. Y. Wang, W. Yang, X. Chen, Y. Wang, L. Guo, L.-P. Chau, Z. Liu, Y. Qiao, A. C. Kot and B. Wen, "SinSR: Diffusion-Based Image Super-Resolution in a Single Step," in *Proc. IEEE/CVF CVPR*, 2024, pp. 25796–25805. doi ·arXiv
28. R. Wu, L. Sun, Z. Ma and L. Zhang, "One-Step Effective Diffusion Network for Real-World Image Super-Resolution," in *Proc. NeurIPS*, vol. 37, 2024, pp. 92529–92553. doi ·arXiv
29. Z. Yue, K. Liao and C. C. Loy, "Arbitrary-steps Image Super-resolution via Diffusion Inversion," in *Proc. IEEE/CVF CVPR*, 2025, pp. 23153–23163. doi ·arXiv
30. L. Dong, Q. Fan, Y. Guo, Z. Wang, Q. Zhang, J. Chen, Y. Luo and C. Zou, "TSD-SR: One-Step Diffusion with Target Score Distillation for Real-World Image Super-Resolution," in *Proc. IEEE/CVF CVPR*, 2025, pp. 23174–23184. doi ·arXiv
31. Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel and M. Le, "Flow Matching for Generative Modeling," in *Proc. ICLR*, 2023. arXiv
32. X. Liu, C. Gong and Q. Liu, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow," in *Proc. ICLR*, 2023. arXiv
33. G. Ohayon, T. Michaeli and M. Elad, "Posterior-Mean Rectified Flow: Towards Minimum MSE Photo-Realistic Image Restoration," in *Proc. ICLR*, 2025. arXiv
34. S. Martin, A. Gagneux, P. Hagemann and G. Steidl, "PnP-Flow: Plug-and-Play Image Restoration with Flow Matching," in *Proc. ICLR*, 2025. arXiv
35. S. Bhadra, V. A. Kelkar, F. J. Brooks and M. A. Anastasio, "On Hallucinations in Tomographic Image Reconstruction," *IEEE Transactions on Medical Imaging*, vol. 40, no. 11, pp. 3249–3260, 2021. doi ·arXiv
36. W. Ren, R. Goyal, Z. Hu, T. T. Aumentado-Armstrong, I. Mohomed and A. Levinshtein, "Hallucination Score: Towards Mitigating Hallucinations in Generative Image Super-Resolution," arXiv:2507.14367, 2025. arXiv
37. A. N. Angelopoulos, A. P. Kohli, S. Bates, M. I. Jordan, J. Malik, T. Alshaabi, S. Upadhyayula and Y. Romano, "Image-to-Image Regression with Distribution-Free Uncertainty Quantification and Applications in Imaging," in *Proc. ICML*, PMLR 162, 2022. proceedings ·arXiv
38. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," in *Proc. IEEE/CVF CVPR*, 2018, pp. 586–595. doi ·arXiv
39. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
40. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in *Proc. NeurIPS (NIPS)*, 2017. arXiv
41. A. Mittal, R. Soundararajan and A. C. Bovik, "Making a 'Completely Blind' Image Quality Analyzer," *IEEE Signal Processing Letters*, vol. 20, no. 3, pp. 209–212, 2013. doi
42. J. Ke, Q. Wang, Y. Wang, P. Milanfar and F. Yang, "MUSIQ: Multi-scale Image Quality Trans-

former," in *Proc. IEEE/CVF ICCV*, 2021, pp. 5128–5137. doi ·arXiv
43. J. Wang, K. C. K. Chan and C. C. Loy, "Exploring CLIP for Assessing the Look and Feel of Images," in *Proc. AAAI*, vol. 37, no. 2, 2023, pp. 2555–2563. doi ·arXiv
44. X. Li, Y. Ren, X. Jin, C. Lan, X. Wang, W. Zeng, X. Wang and Z. Chen, "Diffusion Models for Image Restoration and Enhancement: A Comprehensive Survey," *International Journal of Computer Vision*, vol. 133, no. 11, pp. 8078–8108, 2025. doi ·arXiv
45. G. Daras, H. Chung, C.-H. Lai, Y. Mitsufuji, J. C. Ye, P. Milanfar, A. G. Dimakis and M. Delbracio, "A Survey on Diffusion Models for Inverse Problems," arXiv:2410.00083, 2024. arXiv
46. B. B. Moser, A. S. Shanbhag, F. Raue, S. Frolov, S. Palacio and A. Dengel, "Diffusion Models, Image Super-Resolution, and Everything: A Survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 7, pp. 11793–11813, 2025. doi ·arXiv
47. W. Xia, Y. Zhang, Y. Yang, J.-H. Xue, B. Zhou and M.-H. Yang, "GAN Inversion: A Survey," *IEEE TPAMI*, vol. 45, no. 3, pp. 3121–3138, 2023. doi ·arXiv
48. Y. Song and S. Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution," in *Proc. NeurIPS*, 2019. arXiv
49. H. Chung, J. Kim, S. Kim and J. C. Ye, "Parallel Diffusion Models of Operator and Image for Blind Inverse Problems," in *Proc. IEEE/CVF CVPR*, 2023, pp. 6059–6069. paper

DLCV 09C・低光影像增強

從經典調光、Retinex 與曲線，到高效率網路、生成先驗、雙邊網格和色彩表，同時理解品質、成本與低光評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/

你將學到

- 區分色調校正，以及雜訊、細節與色彩復原。
- 比較直方圖、gamma、Retinex 與攝影變換方法。
- 分別依監督訊號與架構分類學習方法。
- 選擇能揭露曝光、色彩與雜訊失敗的評估協定。
- 量測全解析度成本，並評估下游任務適用性。

先看全貌

在暗處拍到的照片，並不只是「把亮度調低的正常照片」。感測器收到的光子非常少，所以影像又暗、又有雜訊，還常常帶著色偏。低光影像增強的目標，是產生「如果當時光線充足，你會拍到的那張照片」：曝光正確、顏色自然、細節乾淨，而且不引入新的假影。

> **用最簡單的話說**
>
> 想像你在幾乎關燈的房間裡畫畫。畫出來又淡又髒，你也搞不清楚哪一筆是哪個顏色。事後開燈、把這張畫「影印得亮一點」是沒用的：髒污也跟著變亮，畫錯的顏色還是錯的。好的低光增強器，就像一位朋友看著你那張淡淡的畫，把它重新畫一遍，畫成白天看起來的樣子：同一張圖、正確的顏色，而且沒有髒污。

為什麼重要：手機每天都在暗處拍照，車輛與機器人必須在夜間看得見，監視系統、科學影像與醫療內視鏡也都在跟同樣的物理限制奮戰。低光也會傷害下游模型：用白天照片訓練的偵測器，到了晚上準確率會下降。因此 LLIE 既是畫質問題，也是電腦視覺的強健性問題。

> **關於本篇**
>
> 本篇由先前的 LLIE 文章 [1] 擴寫而成（原文在此），依照原文的大綱，補上數學、程式碼、圖表與文獻綜述。

相關章節。 DLCV 09 提供一般復原背景；DLCV 10 合併多張影像的互補證據。本章將品質與效率放在每個家族旁一起討論。

目的與難點

白話版。低光增強結合色調調整，以及含雜訊細節與色彩的復原；單靠變亮無法兼顧兩者。

精確版。 LLIE 和影像轉換（image-to-image translation）不同：後者會改變影像的屬性（夏天變冬天、照片變畫作），LLIE 則應該完整保留場景與內容，只把光線不足造成的破壞還原。原文 [1] 把這些破壞分成兩個難點：

1. **色調強化**。在全域與局部提高曝光，同時不壓垮亮部、也不把對比壓平。燈具這類亮區必須維持明亮，暗區要被打開，而且區域之間的明暗順序應該保留。
2. **細節還原**。找回埋在雜訊與量化誤差裡的紋理與顏色，同時抑制雜訊而不是放大它。

在深度學習裡，只要網路被訓練成輸出乾淨的影像，細節還原本來就是它在做的事，因此多數方法的設計重點放在色調（要把哪些影像性質或物理模型放進網路）與學習機制（用什麼監督方式與損失函數），而不是另外設計一個專門的細節模組。

為什麼不是單純調亮

線性增益 $y = c\,x$（$c > 1$）會在四個地方失敗，全都可以在圖 1 看到：

- **雜訊**。光子到達是隨機的。光子很少時，訊噪比很低，而增益會把訊號和雜訊一起放大。影像變亮了，卻沒有變乾淨。
- **色偏**。低光環境的光常帶有顏色（鎢絲燈、鈉燈路燈），相機的白平衡與三個色彩通道的雜訊程度也不同。三通道等比例放大會保留色偏；分別放大又可能產生奇怪的顏色。
- **飽和與過曝**。已經接近上限的像素（燈、窗戶、螢幕）會在 1 被截斷。大到足以照亮暗部的全域增益，會毀掉亮部。
- **量化**。一張暗的 8 位元影像可能只用到 20 到 40 個灰階。把它們拉開，就會出現明顯的色階斷層（banding）。

低光拍攝的模型

明確寫出退化模型很有幫助，一方面便於理解，另一方面可用來**合成訓練資料**。請在線性光（數值正比於光子數）上操作，而不是在經過 gamma 編碼的像素上（gamma 請見 IP 03）。常用的是 Foi 等人的 Poisson–Gaussian 模型 [2]：

$$z = \frac{1}{\eta}\,\mathcal{P}(\eta\, k\, x) + n, \qquad n \sim \mathcal{N}(0, \sigma_r^2),$$

其中 $x \in [0,1]$ 是乾淨的線性影像，$k \in (0,1]$ 是到達感測器的光量比例，η 是把強度換算成光子數的尺度（η 越大代表光子越多、相對雜訊越小），$\mathcal{P}(\cdot)$ 是模擬**散粒雜訊**（shot noise）的 Poisson 隨機變數，n 是標準差 σ_r 的高斯**讀出雜訊**（read noise）。之後相機再套上色調曲線並量化。z 的變異數約為 $kx/\eta + \sigma_r^2$，會隨訊號增加，也就是說雜訊是與訊號相關的；這就是為什麼「加高斯雜訊」很難代表真實的低光。

```
import numpy as np
from skimage import data, img_as_float

rng = np.random.default_rng(0)
clean = img_as_float(data.coffee())            # H x W x 3, sRGB-encoded, in [0, 1]
```

```
def synth_low_light(x, k=0.05, peak=2000.0, read_std=8e-4, gamma=2.2):
    lin = k * x ** gamma                            # 1) less light, in linear units
    lin = rng.poisson(lin * peak) / peak            # 2) shot noise (Poisson)
    lin = lin + rng.normal(0, read_std, lin.shape) # 3) read noise (Gaussian)
    out = np.clip(lin, 0, 1) ** (1 / gamma)         # 4) re-encode for display
    return np.round(out * 255) / 255                # 5) 8-bit quantization

dark = synth_low_light(clean)
print(clean.mean(), dark.mean())                    # about 0.39 -> 0.10
```

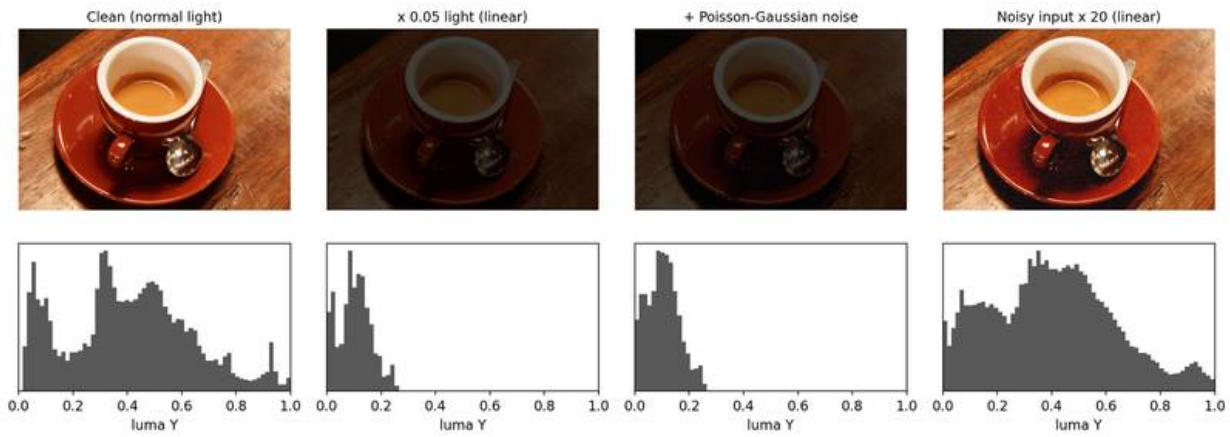


圖 *1* — 合成的低光影像。拿掉 *95%* 的光，直方圖被擠進最下方四分之一。在暗影像裡很難看到雜訊，但用完美的增益把曝光還原後（最右），雜訊就現形了：調亮與去噪是兩個獨立的問題。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

最右邊那張是本篇最重要的一張圖：即使是理想的亮度校正，雜訊依然存在。這就是為什麼 LLIE 是一個復原問題（請見 DLCV 09・影像復原），而不只是色調對應問題。

非深度學習的 LLIE 與低光影像的特性

白話版。直方圖方法重新分配強度，Retinex 方法估計並調整照明；各自的統計假設決定處理行為。

低光影像在色彩空間上的特性

許多色彩空間會把「有多亮」的通道和「是什麼顏色」的通道分開。不同空間稱它為明度（value）、強度（intensity）、亮度（luma）或明亮度（lightness），定義也各不相同。設 $R, G, B \in [0, 1]$（色彩空間本身請見 IP 06）：

$$\begin{aligned}
\text{HSV:} \quad & V = \max(R, G, B), \\
\text{HSI:} \quad & I = \tfrac{1}{3}(R + G + B), \\
\text{YCbCr (BT.601):} \quad & Y' = 0.299\,R' + 0.587\,G' + 0.114\,B', \\
\text{CIELAB:} \quad & L^* = 116\,f(Y_{\text{lin}}/Y_n) - 16.
\end{aligned}$$

其中 R', G', B' 是經 gamma 編碼的值；Y_{lin} 是線性光的亮度，Y_n 是參考白的亮度；$f(t)$ 在一個小門檻以上為 $t^{1/3}$，以下為線性。幾點說明：

- V **就是「亮通道」（bright channel）**。它是逐像素的最大值，與去霧中使用的暗通道（dark channel，逐區塊的最小值）[3] 相對。在暗影像中，V 是被黑暗破壞最少的通道，這也是 LIME [4] 用它來初始化光照的原因。
- **亮度權重**。 ITU-R BT.601 [5] 定義亮度為 $Y' = 0.299R' + 0.587G' + 0.114B'$，這個全範圍（full range）形式就是 JPEG/JFIF 使用的版本。8 位元的有限範圍（limited range，又稱 studio range）影片中，BT.601 把黑色放在 16、白色放在 235：$Y = 16 + 219\,Y'$。若 $R', G', B' \in [0, 255]$，就變成 $Y = 16 + 0.257R' + 0.504G' + 0.098B'$，因為 $219/255 \times (0.299, 0.587, 0.114) \approx (0.257, 0.504, 0.098)$。兩個公式描述的是同一個亮度，只差在範圍與偏移量。
- L^* 被設計成感知上均勻，相同的數值差在人眼看來差不多大，因此很適合拿來做對比增強。

實務上的結論是：既然亮度可以和顏色拆開，就可以**只增強亮度通道**並保留色度。圖 2 顯示，選哪個通道影響很大。

```
R, G, B = dark[..., 0], dark[..., 1], dark[..., 2]
V = dark.max(axis=2)                          # HSV value
I = dark.mean(axis=2)                         # HSI intensity
Y = 0.299 * R + 0.587 * G + 0.114 * B         # BT.601 luma, full range (JPEG/JFIF)
Y_limited = 16 + 219 * Y                      # 8-bit limited ("studio") range, 16..235
print(np.allclose(Y_limited, 16 + (65.481 * R + 128.553 * G + 24.966 * B)))  # True
```



圖 2 — 相同 *gamma* 在不同色彩空間效果不同。V ／ I 以共同倍率縮放 *RGB*，裁切前保色相及 *HSV* 飽和度，但感知色彩豐富度仍可能改變。固定色度只增 Y ／ L^* 會顯得較淡。

共同縮放 RGB 保留通道比例及色相，裁切前 HSV 飽和度不變，感知色彩卻可能隨亮度增加。固定 Cb ／ Cr 或 a ／ b 只增加 Y' ／ L^*，可能顯得褪色。HVI-CIDNet [6] 以學得色彩表示處理這種色調耦合。

直方圖等化與 CLAHE

白話版。直方圖等化把強度重新分配到可用範圍，展開暗影像集中的數值，也可能放大雜訊。

精確版。對 L 個灰階、正規化直方圖 $p_r(r_k) = n_k/(MN)$ 的影像，HE 以縮放後的累積分布函數（CDF）作為對應：

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), \qquad k = 0, \dots, L-1,$$

其中 n_k 是灰階 r_k 的像素數，MN 是影像大小。T 的斜率正比於直方圖，所以像素很多的灰階會被拉開。推導請見 IP 03。對彩色影像，常見做法是先轉到有亮度通道的色彩空間，只等化該通道，再轉回來，例如 RGB → Lab → 等化 L^* → RGB。

全域 HE 在低光下有兩個問題：它會過度拉伸大片平坦區域（連同其中的雜訊），而且單一對應無法同時照顧暗角與明亮的燈。**自適應 HE**（adaptive HE）為每個區塊計算對應；**CLAHE**（contrast-limited AHE）在計算 CDF 前先把每個區塊的直方圖在某個上限截斷，藉此限制 T 的斜率，也就限制了雜訊增益，最後在相鄰區塊間內插以避免接縫 [7], [8]。

僅線上版：*CLAHE：區域等化但不放大雜訊 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/clahe 區域等化但不放大雜訊/*

Gamma 校正

冪次律 $s = r^\gamma$ 在 $\gamma < 1$ 時對暗部的提升大於亮部；它的斜率 $\gamma r^{\gamma-1}$ 在 $r = 0$ 附近很大。它便宜又單調，但屬於全域操作：整張圖只有一個 γ、不會去噪，而且依套用的空間不同，同樣面臨上述「褪色或過飽和」的選擇。

```
from skimage import color, exposure

gamma_out = dark ** 0.4                                    # power-law curve on every
 ↪  channel
lab = color.rgb2lab(dark)
L = lab[..., 0] / 100                                      # L* scaled to [0, 1]
he = lab.copy();    he[..., 0] = 100 * exposure.equalize_hist(L)
clahe = lab.copy(); clahe[..., 0] = 100 * exposure.equalize_adapthist(L, clip_limit=0.01)
he_out, clahe_out = color.lab2rgb(he), color.lab2rgb(clahe)
for name, im in [("gamma", gamma_out), ("HE", he_out), ("CLAHE", clahe_out)]:
    print(f"{name:6s} mean {im.mean():.3f}   std {im.std():.3f}")
```



圖 3 — 傳統增強方法作用在圖 1 的有雜訊輸入上。全部都變亮了，但沒有一個去除雜訊；在輸入最暗的地方，*Retinex* 的輸出出現強烈的彩色雜訊。

Retinex 理論

白話版。 Retinex 認為，眼睛（或相機）接收到的影像是「物體本身的顏色」乘上「照在它上面的光」。如果能估計出光，就能把它除掉，看到物體真正的樣子，再換上一個更好的光。

精確版。 Land 的 Retinex 理論 [9] 把觀察到的影像 S 表示為

$$S(\mathbf{x}) = R(\mathbf{x}) \circ L(\mathbf{x}),$$

$\mathbf{x}$ 是位置，$R \in [0,1]$ 是簡化模型中跨配對光照一致的**反射率**，L 是除陰影邊界外通常平滑的**照明**，$\circ$ 是逐元素乘。照明不是色彩空間的輝度；模型分量也不保證唯一物理可識別。

從一個乘積解出兩個未知數是病態問題（ill-posed），因此每個 Retinex 方法都會加上先驗。最經典的是「照度是平滑的」。

單尺度 Retinex（SSR） [10] 用高斯模糊 G_σ 估計 L，並在對數域運算，讓乘法變成加法：

$$\log R_c(\mathbf{x}) = \log S_c(\mathbf{x}) - \log\left(G_\sigma * S_c\right)(\mathbf{x}),$$

對每個色彩通道 c 計算，$*$ 為卷積，σ 為周圍（surround）尺度。σ 小，局部對比強但邊緣出現光暈（halo）；σ 大，色調較好但細節較弱。**多尺度 Retinex（MSR）** [11] 以權重 w_n 平均多個尺度（通常三個）的 SSR 輸出：

$$\log R_c^{\mathrm{MSR}} = \sum_{n=1}^{N} w_n \left[\log S_c - \log(G_{\sigma_n} * S_c)\right].$$

由於 SSR 與 MSR 逐通道運算，顏色容易變灰。**MSRCR**（帶色彩還原的 MSR）[11] 將結果乘上一個色彩還原因子，依據是各通道佔 $R+G+B$ 的比例；其他變體則只對強度通道做 MSR，並保

留輸入的色度。

```
from scipy import ndimage as ndi

def ssr(img, sigma=40.0, eps=1e-3):
    log_s = np.log(img + eps)
    log_l = np.log(ndi.gaussian_filter(img, sigma=(sigma, sigma, 0)) + eps)  # L ~ Gaussian
     ↪  blur
    r = log_s - log_l                                          # log R = log S - log L
    lo, hi = np.percentile(r, [1, 99])                         # robust stretch for display
    return np.clip((r - lo) / (hi - lo), 0, 1)

msr = np.mean([ssr(dark, s) for s in (15, 80, 250)], axis=0)   # MSR, equal weights
 ↪  (simplified)
```

做增強時，我們通常不只要 R，因為單獨的 R 看起來平板又不自然。我們要的是把 L 調亮再組合回去：$\hat{S} = R \circ \hat{L}$，例如 $\hat{L} = L^{\gamma}$、$\gamma < 1$。這個「分解、調整 L、重組」的模式，就是大多數 Retinex 深度網路的範本。



圖 4 — *LIME* 風格的 *Retinex* 分解。照度以 $max(R, G, B)$ 初始化後再平滑；反射率為 S / L。把 L 以 *gamma 0.35* 調亮再乘回去，得到自然的結果；單獨的 R 則顯得平板而過度飽和。

LIME：把 Retinex 寫成最佳化問題

LIME [4] 以亮通道 $\hat{T}(\mathbf{x}) = \max_c S_c(\mathbf{x})$ 初始化，再結構感知平滑，保持主要邊緣。增強為 S/T^{γ}；若 $R = S/T$，等價 $RT^{1-\gamma}$，和前面重組指數慣例須區分。除法放大雜訊，因此可另去雜訊。它展示物理模型、手工先驗及最佳化的經典流程。

另一個模型：對反轉影像去霧

Retinex 並不是唯一的成像模型。去霧使用的是**大氣散射模型**（atmospheric scattering model），其寫法見 DLCV 09：有霧的觀測影像 I 是無霧的場景輻射 J 經透射率 t 衰減後，再加上全域大氣光 A。這個模型不能直接套用在暗影像上，因為暗影像看起來並不像有霧的影像。

Dong 等人觀察到，**反轉**後的低光影像 $1 - S$ 確實像有霧：暗區變成明亮、低對比的區域，就像霧一樣。因此他們先反轉影像，套用去霧演算法（精神上與暗通道先驗 [3] 相同），再把結果反轉回來 [12]。令 I_{haze} 為有霧影像、J_{haze} 為無霧結果，兩次反轉為

$$I_{\text{haze}} = 1 - S_{\text{Retinex}}, \qquad J_{\text{haze}} = 1 - \hat{S}_{\text{enhanced}}.$$

從「哪些資訊可靠」的角度看：去霧假設暗通道代表無霧的內容；但在低光影像裡，暗通道大多是被破壞的像素，所以重點轉向亮通道，也就是受黑暗影響最少的區域。LIME 論文 [4] 分析了這種反轉做法，並說明它可以改寫成 Retinex 的形式，所以兩個模型本質上是同一個假設，只是 Retinex 的表述較簡單。

統計方法

許多傳統方法在這些模型之上再加入統計量或先驗：保持場景明暗順序的自然度約束（NPE [13]）、分層或二維直方圖模型（DICM 資料集背後的方法 [14]），以及融合多張合成曝光。它們都是手工設計、不需要訓練資料，泛化行為可預期；弱點則是雜訊，通常被忽略或交給另一個去噪器處理。

深度學習 LLIE

兩個軸：學習訊號與設計家族

架構與監督是不同選擇。依 Li 等人 [15]：**SL** 用配對，**RL** 用調整動作獎勵，**UL** 用未配對暗／正常圖，**ZSL** 在該綜述指只用低光、甚至逐測試圖最佳化，**SSL** 指半監督、結合配對與未配對。此處 SSL 不是自監督；其他論文用詞不同，要看實際資料需求。

設計軸問預測什麼：曲線、照明、影像殘差、生成去雜訊軌跡或精簡修圖轉換。Retinex 可搭多種訓練，擴散是模型家族，不是監督類別。

方法	設計與學習訊號	品質機制	原文成本與範圍
Retinex-Net [16]	配對分解／照明，無固有影像標籤	反射一致、結構平滑	多階段加可選去雜訊，要計完整流程
EnlightenGAN [17]	未配對影像生成	全域／局部對抗、自我特徵	判別器只在訓練
Zero-DCE [18]	無參考、空間曲線	曝光、色彩、對比、參數平滑	**79,416** 參數，1200 × 900 **84.99 G**，見 [19] 表 4
Zero-DCE++ [19]	低解析共用曲線	類似先驗、較小表達	**10,561** 參數，同尺寸 **0.115 G**
RUAS [20]	搜尋展開照明／去雜訊、無參考	分開亮度與雜訊	表 2 四捨五入 **0.001 M** ／ **0.003 M**
SCI [21]	無監督校準串接	訓練校準、推論單塊	表 4 四捨五入 **0.0003 M**，校準器不部署
Retinexformer [22]	配對照明 Transformer	照明引導復原	**1.61 M**、**256 × 256** 時 **15.57 G**
LLFlow ／擴散 [23]、[24]、[25]、[26]、[27]	條件生成，監督各異	條件分布／先驗	須計所有網路評估與編解碼
HDRNet [28]	配對修圖、局部仿射網格	局部／全域預測轉換	縮圖網路加全解析度 guide ／ slicing ／套用
自適應 3D LUT [29]	配對／未配對修圖	每圖全域色彩表	Titan RTX、3840 × 2160 **1.66 ms**，表 7

各列解析度、計數與硬體不同，用來解釋設計，不能直接當速度榜。LUT 修圖時間也不代表暗手機圖的去雜訊效果。初學先跑 gamma、光子實驗及曲線玩具，再讀架構。

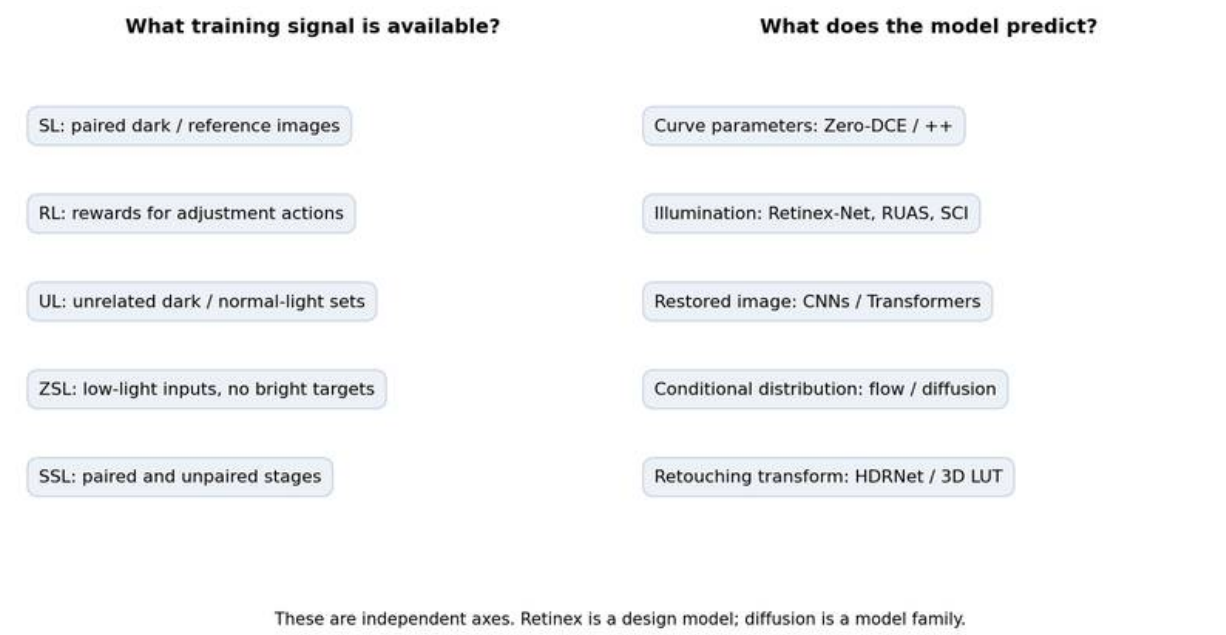


圖 5 — 分開監督與架構。同一設計可用不同資料訊號；擴散是家族，*Retinex* 是建模選擇。

監督式方法（成對資料）

以低光與正常光影像對 (x_i, y_i) 訓練，沿用 DLCV 09 的一般做法：用重建損失擬合 f_θ，通常是 $\mathcal{L} = \|f_\theta(x) - y\|_1$ 再加上感知或結構項（這些損失請見 DLCV 09）。成對資料取得成本高（同一靜態場景拍兩種曝光，或用合成的），而且網路會學到參考影像特定的亮度與色彩風格，這一點在後面談評估時很重要。

代表性的例子：**LLNet** [30]（2017）是最早的深度 LLIE 模型之一，以去噪自編碼器同時學習調亮與去噪；Tao 等人 [31]（2017）先用 CNN 去噪，再以亮通道模型增強對比；**MIRNet** [32]（2020）是一個多解析度的通用復原骨幹；**SNR-aware** [33]（2022）估計逐像素的訊噪比圖，在局部鄰域雜訊太大、不可信的地方改用長距離注意力；**HVI-CIDNet** [6]（2025）則學習一個把色彩與強度解耦的色彩空間，處理圖 2 的問題。

僅線上版：更多深度 *LLIE* 方法 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/更多深度-llie-方法/*

高效率端到端設計

直接網路可同時學復原和色調，成本隨空間特徵增長。IAT [34] 結合局部乘加圖與全域色彩矩陣／ gamma；薄卷積、depthwise 及小全域分支省計算，仍受未見雜訊限制。

結構重參數化 [35] 訓練多線性分支、部署合併成單卷積；權重共享則省儲存參數。兩者不保證每台裝置都快，應測完整匯出圖。

僅線上版：高效率網路元件 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/高效率網路元件/*；結構重參數化逐步說明 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/結構重參數化逐步說明/*

以 Retinex 為基礎的網路

Retinex-Net [16] 讓低光與正常圖各有 R, L，先分解、調整 $L \to \hat{L}$、再重組，也提出 LOL。配對訓練限制重建、反射一致與結構平滑；沒有量測的固有分量標籤，因此不是唯一物理解。另有照明網路調色調、反射去雜訊處理放大雜訊。

僅線上版：*Retinex-Net*：分解、調光、去雜訊 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/retinex-net 分解調光去雜訊/*

這個模式很快就流行起來。**KinD** [36]（2019）與 **KinD++** [37]（2021）加入反射率還原步驟，因為暗影像的反射率帶有雜訊與色偏；**Retinexformer** [22]（2023）在 Retinex 模型中加入代表雜訊與假影的擾動項，採單階段訓練，並用照度估計引導自注意力，後來成為常見的比較基準；**RetinexMamba** [38]（2024 年預印本）把注意力換成 Mamba 狀態空間模組。「Retinex 分解＋新骨幹」多年來一直是很穩定的設計模式。另一條較少見的路線則以大氣散射模型為基礎，建構輕量網路（CPGA-Net [39]、CPGA-Net+ [40]）。

僅線上版：更多深度 *LLIE* 方法 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/更多深度-llie-方法/*

在同一家族內比較品質與成本

先在縮圖預測平滑轉換、再套原像素，可保留原解析細節，但不會恢復裁切或雜訊消失的資訊。固定逐點轉換不足以完整去雜訊／模糊；若參數由鄰域網路預測，整個曲線系統仍可使用空間上下文。

小照明值除法同時放大細節與雜訊；反射復原另需成本。以下比較放大照明圖和放大已增強影像，參考是同一玩具算子全解析執行，不是實拍真值。

以 LIME [4] 初始亮通道加普通 Gaussian 平滑估照明，不做完整結構最佳化。分別在全解析及每軸小八倍的 **1/64** 像素上估計，再比較兩種放大。

```
import numpy as np
from scipy.ndimage import gaussian_filter
from skimage import data, img_as_float
from skimage.metrics import peak_signal_noise_ratio as psnr
from skimage.transform import resize

dark = (0.15 * img_as_float(data.coffee()) ** 2.2) ** (1 / 2.2)   # 400 x 600, 15% of the
 ↪  light

def illumination(x, sigma):
    """A simple illumination estimate: the bright channel (max of R, G, B), smoothed."""
    return np.clip(gaussian_filter(x.max(axis=-1), sigma), 0.02, 1)

def relight(x, L, gamma=0.4):
```

```
    """Retinex relighting: O = R * L**gamma = I * L**(gamma - 1)."""
    return np.clip(x * L[..., None] ** (gamma - 1), 0, 1)

reference = relight(dark, illumination(dark, sigma=15))          # estimate L at full
 ↪  resolution
small = resize(dark, (50, 75), anti_aliasing=True)               # 8x smaller in each
 ↪  direction
L_small = illumination(small, sigma=15 / 8)                      # estimate L on 1/64 of the
 ↪  pixels
results = {
    "L at 1/8 size, resized up": relight(dark, resize(L_small, dark.shape[:2], order=1)),
    "relight at 1/8 size, resize output": resize(relight(small, L_small), dark.shape,
     ↪  order=1),
}
print(f"mean brightness {dark.mean():.2f} -> {reference.mean():.2f}")
for name, out in results.items():
    print(f"{name:36s} PSNR vs full-resolution estimate {psnr(reference, out,
     ↪  data_range=1):5.2f} dB")
```

```
mean brightness 0.16 -> 0.35
L at 1/8 size, resized up            PSNR vs full-resolution estimate 54.57 dB
relight at 1/8 size, resize output   PSNR vs full-resolution estimate 22.87 dB
```

放大平滑**照明圖**與全解析估計相近，玩具超過 **54 dB**；放大**結果圖**少約 **31.7 dB**，因細節已丟失。有銳利陰影的照明圖更難，可能需保邊上取樣。

Retinex 啟發的展開式網路

展開（unrolling，或 unfolding）是把 Retinex 問題的迭代最佳化演算法，每一次迭代變成一個帶有可學習參數的網路階段。網路架構繼承了演算法的結構與可解釋性，而訓練取代了手工調整的先驗。

- **RUAS** [20]（2021）展開照度估計與去噪的最佳化模型，搜尋先驗模組的架構，並以協同式的無參考（reference-free）策略訓練，所以它同時也屬於非監督那一欄。它非常小也非常快。
- **URetinex-Net** [41]（2022）把 Retinex 分解的最佳化展開成網路，包含初始化、R 與 L 的展開更新，以及照度調整等學習模組。

非配對訓練：EnlightenGAN

EnlightenGAN [17]（2021）使用生成對抗網路（GAN），只需要一組暗影像和另一組不相關的明亮影像。生成器是 U-Net，並以輸入照度推得的自正則化注意力圖引導，產生增強影像。**全域**判別器評估整張影像，**局部**判別器評估隨機小區塊，讓輸出在整體與細節上都像光線充足的照片。由於沒有真值，**自特徵保持**（self-feature-preserving）損失比較輸入與輸出的 VGG 特徵，讓內容維持不變。

零參考方法與曲線估計

白話版。 Zero-DCE 不直接輸出新影像，而是替每個像素輸出一條調亮曲線，套用它，再用不需要任何參考照片的損失來評分：「曝光好不好？顏色平不平衡？曲線平不平滑？局部對比有沒有保留？」

LE 曲線。 Zero-DCE [18]（2020）定義了一條二次的**光增強曲線**（light-enhancement curve）

$$\mathrm{LE}(x;\alpha) = x + \alpha\, x\,(1-x), \qquad \alpha \in [-1,1],$$

$x \in [0,1]$ 是像素。$\mathrm{LE}(0)=0$、$\mathrm{LE}(1)=1$ 保範圍；$|\alpha| \le 1$ 下導數 $1+\alpha(1-2x) \ge 0$，**固定**曲線保序，但不同位置不同參數不保證像素間保序。它可微、可反傳，反覆套用增加表達：

$$\mathrm{LE}_n(\mathbf{x}) = \mathrm{LE}_{n-1}(\mathbf{x}) + \mathcal{A}_n(\mathbf{x})\,\mathrm{LE}_{n-1}(\mathbf{x})\big(1-\mathrm{LE}_{n-1}(\mathbf{x})\big),$$

其中 LE_0 是輸入，$n = 1,\dots,8$，而 $\mathcal{A}_n$ 是**逐像素、逐通道**的參數圖，而非單一的 α。一個七層的 CNN（DCE-Net，每層 32 個通道、約 79k 個參數、沒有降取樣）以 tanh 輸出 $8 \times 3 = 24$ 張參數圖。每個像素因此都有自己的高階曲線，局部調整就是這樣從看似全域的公式中產生的。

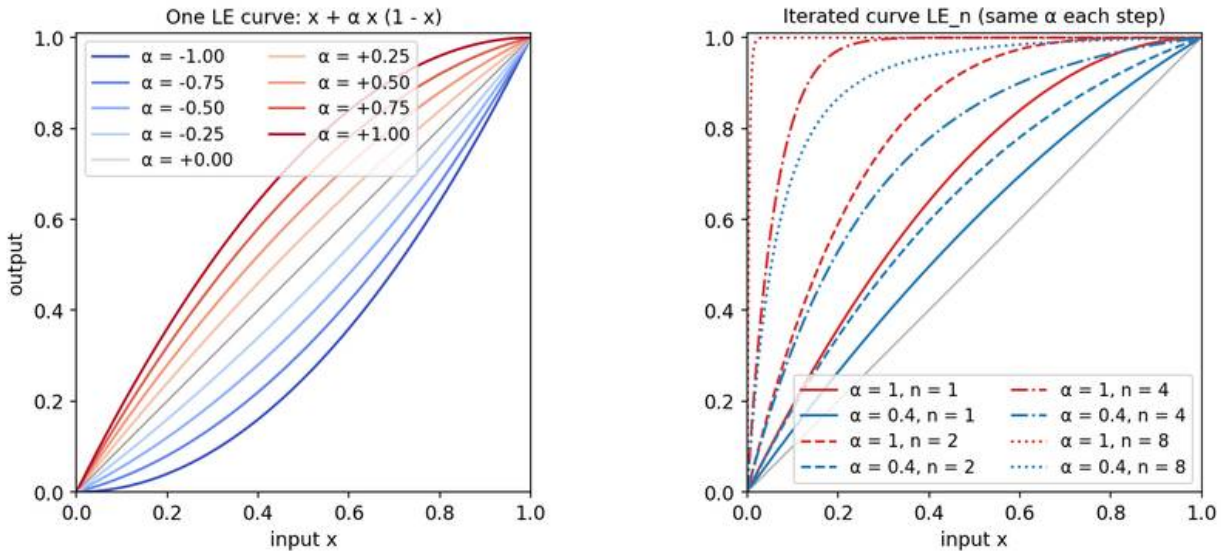


圖 *6 — LE* 曲線。左：α 從 -1 到 1 的單次迭代。右：迭代讓曲線變陡；$\alpha = 1$、$n = 8$ 時，0.02 這個值已經被對應到將近 1，所以暗部裡微小的雜訊值會被極度放大。

```
def le_iterate(x, alphas):
    """Zero-DCE curve: LE_n = LE_{n-1} + a_n * LE_{n-1} * (1 - LE_{n-1})."""
    y = x.copy()
    for a in alphas:              # a can be a scalar or a per-pixel map
        y = y + a * y * (1 - y)
    return y

x = np.array([0.0, 0.02, 0.1, 0.3, 0.6, 1.0])
print(le_iterate(x, [1.0] * 8).round(3))   # [0. 0.994 1. 1. 1. 1.]
print(le_iterate(x, [0.4] * 8).round(3))   # [0. 0.245 0.695 0.93 0.985 1.]
```

第一行輸出顯示了危險之處：對很小的 x，每一步大約乘上 $1+\alpha$，所以 $\alpha = 1$ 迭代八次的增益接近 $2^8 = 256$。暗部裡的雜訊也會被放大同樣的倍數。

無參考損失。四項先驗維持局部對比、推動區塊平均曝光、減少全域 RGB 不平衡及平滑參數。它們都不是真實正常曝光標籤，灰世界可能破壞彩色場景，平滑曲線也會放大雜訊。原文曝光目標 **0.6**、**8** 次迭代 [18]。

僅線上版：*Zero-DCE ／ Zero-DCE++：沒有參考影像的曲線學習 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/zero-dcezero-dce 沒有參考影像的曲線學習/*

後續方法改良了這個想法：

- **Zero-DCE++** [19]（2022）改用深度可分離卷積（depthwise separable convolution），只預測 **3** 張曲線參數圖並在所有迭代中重複使用，而且在降取樣後的輸入上估計，再縮放回原尺寸。模型因此縮小到約 10k 個參數，即使在 CPU 上也跑得相當快。
- **SCI** [21]（2022）以權重共享的多階段串接學習照度，並用**自校正**（self-calibrated）模組讓各階段的輸出在訓練時收斂到相同狀態，因此測試時只需要一**個**基本模組。它以非監督損失訓練。
- **CLIP-LIT** [42]（2023）針對逆光影像。它使用視覺-語言模型 CLIP，學習一對文字提示（prompt），一個代表逆光、一個代表光線良好，再迭代地精修提示與增強網路，讓輸出與「光線良好」提示的 CLIP 相似度成為訓練訊號。

用 PyTorch 寫一個零參考小玩具

為了看看這些損失函數如何運作，下面的程式碼在 CPU 上訓練一個迷你的 Zero-DCE 風格網路約半分鐘。它用四張 `skimage` 影像的隨機裁切訓練，每塊都在線性光中以隨機倍率變暗；測試則用訓練時從沒看過的咖啡杯影像。全程不使用任何真值。（損失權重與論文不同，因為各項的正規化方式不同：曝光誤差用平方而非絕對值，梯度用平均而非加總。）

```
import numpy as np
import torch
torch.set_num_threads(1)
import torch.nn as nn
import torch.nn.functional as F
from skimage import data, img_as_float
from skimage.transform import resize

torch.manual_seed(0); rng = np.random.default_rng(0)
darken = lambda x, k: (k * x ** 2.2) ** (1 / 2.2)          # less light, in linear units
train = [resize(img_as_float(f()), (256, 256), anti_aliasing=True)
         for f in (data.astronaut, data.chelsea, data.rocket, data.immunohistochemistry)]

def batch(n=4, size=64):                                    # random crops, random darkness
    xs = []
    for _ in range(n):
        img = train[rng.integers(len(train))]
        i, j = rng.integers(0, 256 - size, 2)
        xs.append(darken(img[i:i + size, j:j + size], rng.uniform(0.02, 0.3)))
    return torch.tensor(np.stack(xs), dtype=torch.float32).permute(0, 3, 1, 2)
```

```
class TinyDCE(nn.Module):
    def __init__(self, n_iter=8, w=16):
        super().__init__()
        self.n_iter = n_iter
        self.body = nn.Sequential(
            nn.Conv2d(3, w, 3, padding=1), nn.ReLU(), nn.Conv2d(w, w, 3, padding=1),
             ↪  nn.ReLU(),
            nn.Conv2d(w, w, 3, padding=1), nn.ReLU(), nn.Conv2d(w, 3 * n_iter, 3, padding=1),
             ↪  nn.Tanh())
    def forward(self, x):
        alphas = self.body(x).chunk(self.n_iter, dim=1)    # 8 maps of shape B x 3 x H x W
        y = x
        for a in alphas:
            y = y + a * y * (1 - y)                         # the LE curve
        return y, torch.cat(alphas, 1)

def L_exp(y, E=0.6, p=16):   # mean brightness of 16x16 patches should be near E
    return (F.avg_pool2d(y.mean(1, keepdim=True), p) - E).pow(2).mean()
def L_col(y):                # gray-world: channel means should agree
    r, g, b = y.mean((2, 3)).unbind(1)
    return ((r - g) ** 2 + (r - b) ** 2 + (g - b) ** 2).mean()
def L_tv(a):                 # curve parameters should vary smoothly in space
    return (a[..., 1:, :] - a[..., :-1, :]).pow(2).mean() + (a[..., :, 1:] - a[..., :,
     ↪  :-1]).pow(2).mean()
def L_spa(y, x, p=4):        # keep the contrast between neighbouring 4x4 regions
    Y, X = (F.avg_pool2d(t.mean(1, keepdim=True), p) for t in (y, x))
    return sum(((torch.diff(Y, dim=d) - torch.diff(X, dim=d)) ** 2).mean() for d in (2, 3))

net = TinyDCE()
opt = torch.optim.Adam(net.parameters(), lr=1e-3)
for step in range(600):                                    # small CPU training experiment
    x = batch()
    y, a = net(x)
    loss = L_spa(y, x) + 10 * L_exp(y) + 5 * L_col(y) + 200 * L_tv(a)
    opt.zero_grad(); loss.backward(); opt.step()

test = darken(img_as_float(data.coffee()), 0.05)           # an image never used in training
with torch.no_grad():
    out, _ = net(torch.tensor(test, dtype=torch.float32).permute(2, 0, 1)[None])
print(f"mean brightness: {test.mean():.3f} -> {out.mean():.3f}")   # about 0.10 -> 0.50
```

拖動滑桿，比較暗的輸入與小玩具的輸出：



暗的輸入



零參考小玩具

暗的咖啡杯影像與小網路的輸出：輸出亮得多，木紋清晰可見，顏色比原照片淺、略偏綠且飽和度較低。

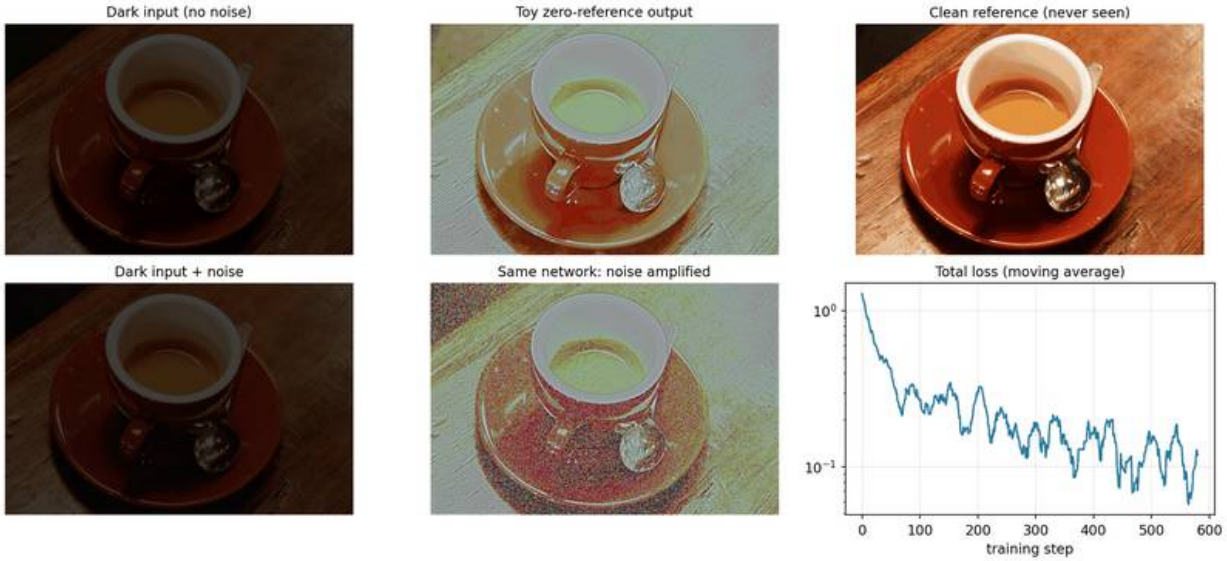


圖 7 — 零參考損失做得到與做不到的事。曝光損失給出好的亮度，但灰色世界的色彩損失把紅褐色場景往灰色拉（上排中間對比上排右邊）；遇到真實雜訊時，逐像素曲線會把雜訊放大（下排中間）。

這個小玩具在一張圖裡同時展示了零參考想法的優點與極限。沒有任何一張參考影像，網路就學到了合理的調亮方式，而且能遷移到沒看過的照片。但每一個損失都是一種先驗，而先驗可能是錯的：場景本來就偏紅或偏褐時，灰色世界假設就失效；固定的曝光目標 E 會把刻意保留的暗區拉平；而損失中沒有任何一項要求去噪。真正的零參考方法用多得多的影像訓練，有些還加入了明確的雜訊處理，但這些失敗模式在已發表的結果中同樣看得到。

正規化流與擴散模型

LLFlow [23]（2022）觀察到：同一張暗影像有許多種可接受的明亮版本，而逐像素損失會把它們平均成一個模糊的折衷。它學習一個條件式正規化流（normalizing flow）：一個可逆網路，在以暗輸入特徵為條件下，把正常光影像對應到高斯分布。這些特徵由呼應傳統方法的先驗引導：輸入的直方圖等化版本、色彩圖（接近 Retinex 的反射率），以及雜訊圖。

2023 年起，**擴散模型**（diffusion model）大量進入 LLIE。它們以暗影像為條件，從隨機雜訊逐步去噪來生成輸出，因此能合成回歸式網路會模糊掉的合理細節。Diff-Retinex [24] 用擴散模型生成 Retinex 的反射率與照度，LightenDiffusion [26] 在潛在空間中分解並以非配對資料訓練，Reti-Diff [27] 只用潛在擴散產生精簡的先驗來引導一個復原 Transformer，GSAD [25] 則以影像結構正則化擴散軌跡。代價是運算量與保真度：多步去噪讓推論變慢，生成的細節可能看起來對，卻不一定真實。

僅線上版：更多深度 *LLIE* 方法 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-*

enhancement/notes/更多深度-llie-方法/

即時攝影修圖轉換

HDRNet [28] 與自適應 LUT [29] 起源於修圖，從縮圖預測精簡轉換再套原像素。能高解析調色調，不等於完整低光雜訊復原。

雙邊網格儲存空間仿射，guide 為第三座標，讓邊緣兩側選不同轉換。內插八節點，對 RGB 加常數套 3×4。HDRNet 學預測；選讀實驗用加權最小平方擬合，不是原網路重現。

3D LUT 以 RGB 索引而非位置，在 $M \times M \times M$ 晶格各存三輸出，三線性混合八角。自適應模型預測係數 a_n，合成基底 Φ_n：

$$\Phi = \sum_{n=1}^{N} a_n \Phi_n.$$

N 是基底數，Φ 為此圖色表。內插對表值線性，因此先合表後查一次，等於逐表查再合。全域 LUT 在同圖把相同 RGB 映成相同輸出，不能獨立照亮其中一區。

> **互動示範（網站版）**：請開啟本篇教學的網站版本，操作此示範。開啟示範

預設三個 **33** 節點基底 [29]，值數 $3M^3$。細晶格以記憶體換暗部陡曲線精度；非均勻及空間／上下文變體改變取捨 [43]、[44]、[45]、[46]、[47]。

僅線上版：雙邊網格與色表數學 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/雙邊網格與色表數學/*；雙邊網格實驗 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/雙邊網格實驗/*；實作並檢查三線性 *LUT* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/實作並檢查三線性-lut/*

Raw 域的低光

以上方法都作用在處理過的 sRGB 影像上，而相機的處理流程（去馬賽克、白平衡、去噪、色調曲線、壓縮）早已丟掉部分資訊。**Learning to See in the Dark** [48]（2018）改把 **raw** 感測器資料送進 U-Net，用它取代整條處理流程。它的 SID 資料集有 5,094 張短曝光 raw 影像，對應 424 張不同的長曝光參考影像，來自 Sony α7S II（Bayer 感測器）與 Fujifilm X-T2（X-Trans 感測器），曝光比為 ×100、×250 與 ×300。Raw 資料保留光子數與數值之間的線性關係，位元數也更多，因此在 sRGB 中毫無希望的極暗場景也能被救回來。代價是模型綁定在單一感測器上。

All-in-one 復原

低光增強也是 all-in-one 復原模型中的一項任務，例如 **InstructIR** [49]（2024），它依自然語言指令（例如「把這張照片調亮」）復原影像。all-in-one 這一整類方法請見 DLCV 09。

面向下游任務的低光處理

如果目標是偵測而非好看的照片，常見的基準做法是「先增強、再偵測」（請見 DLCV 03・物件偵測）。但為人眼調校的增強，不一定是偵測器需要的。**MAET** [50]（2021）以物理方式模擬低光退化，並訓練偵測器同時預測退化參數，讓特徵能感知退化。**FeatEnHancer** [51]（2023）則在網路內部增強階層式特徵，直接用下游任務的損失訓練，而不是增強像素。



圖 *8* — 本篇提到的方法時間軸（依發表年份）。顏色標示以 *Retinex* 為基礎或展開式的設計、非配對或零參考訓練、*flow* 與擴散模型，以及挑戰賽。

誠實地量測效率

白話版。效率的主張很容易提出，卻很難比較。公平的比較輕量設計，會說明確切計算了什麼、在哪個解析度、用哪種硬體，並且涵蓋從輸入影格到輸出影格的每一個步驟。

Dehghani 等人指出，參數量、FLOPs、吞吐量與延遲可能彼此矛盾，只回報其中一項會給出誤導的印象 [52]。針對增強：

- **在部署的解析度下計數**，並且回報每百萬像素的 MAC，讓讀者可以換算。對全解析度網路來說，256 × 256 的 FLOP 數完全說不出 4K 的情況；對以縮圖為基礎的設計則相反。
- **要包含全解析度的步驟**。切片、LUT 內插、曲線迭代、導引濾波器上取樣與縮放，常是自訂運算子，自動的 FLOP 計數器會略過它們。它們正是「預測再套用」設計中隨解析度成長的部分。
- **說明「參數」包含什麼**。一張 33 節點的 3D LUT 約有 10.8 萬個值，比整個 Zero-DCE++ 約 1 萬個參數 [19] 或 IAT 約 9 萬個 [34] 都多。
- **在目標裝置上為整條路徑計時**：固定並回報裝置、執行環境、數值精度、批次大小 1、解析度與執行緒數，先暖機、回報百分位數、停表前先同步 GPU，並把色彩轉換與記憶體傳輸都算進去（協定見 ML 11）。在真實手機上計時的挑戰賽讓這件事更具體：例如 Mobile AI 2022 的學習式 ISP 挑戰賽，在 Snapdragon 8 Gen 1 GPU 上用 TensorFlow Lite 量測執行時間 [53]。論文中的 GPU 數字（例如上面的 Titan RTX 或 V100 計時）只告訴你在那張 GPU 上的相對速度，而不是你的手機上的速度。
- **在全解析度下評斷品質**，使用 DLCV 09 的指標與 DLCV 09C 中低光特有的評估方式，因為粗網格或粗表造成的光暈與色帶，在輸出為了評分而縮小之後可能消失。

僅線上版：解析度、運算與記憶體計數 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/解析度運算與記憶體計數/*

失效模式

症狀	原因	檢查
亮顆粒／假紋理	暗部增益放大感光雜訊	平坦區與雜訊感知復原
色階斷層	少量暗階被大幅拉開	位元深度及近黑漸層
暈邊	粗照明／網格跨強邊界	全解析邊緣、網格與 guide
不自然顏色	灰世界、裁切、稀疏 LUT 訓練	膚色、霓虹和彩色光
影片閃爍	逐幀參數變動	時間參數及移動場景
高分卻難辨識	指標與任務不同	明示任務規則及視覺檢查

全域 LUT 不能依位置區分相同 RGB，固定逐點轉換也不會平均空間獨立雜訊。但參數預測網路可用鄰居，因此要檢查完整系統，而不是假設所有曲線模型限制完全一樣。

僅線上版：色階、暈邊與時間穩定性 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/色階暈邊與時間穩定性/*

評估

白話版。通用的評估工具箱（全參考與無參考指標、學習式與視覺語言評估器、使用者研究，以及隨之而來的陷阱與回報規範）只在 DLCV 09 完整說明一次，本節不再重複，而是談暗影像有什麼不同：為什麼無參考分數在這裡特別重要、常見指標在曝光與色彩上如何誤導、專為增強設計的量測、GT-mean 陷阱，以及以任務為導向的評估。

全參考指標

在成對資料集上，LLIE 論文報告的 PSNR、SSIM [54] 與 LPIPS [55]，定義與 DLCV 09 完全相同。唯一與低光有關的提醒是亮度：與參考影像之間一點點整體曝光差異，就會讓 PSNR 掉很多，這正是下面 GT-mean 陷阱的主題。

無參考指標：為什麼低光評估離不開它

白話版。無參考（no-reference, NR）指標只看一張影像，就猜它的品質好不好，沒有「標準答案」可以比對。低光論文需要這類指標，因為最貼近真實的測試照片根本沒有標準答案；就算有，那也只是眾多可接受亮度中的其中一種。

精確版。有兩個事實把 LLIE 推向無參考評估：

1. **真實測試集是非配對的**。 LIME [4]、MEF [56]、NPE [13]、DICM [14] 與 VV [57] 都是小規模的真實照片集，常有不均勻的光照，沒有真值。LOL [16] 這類成對資料集雖然存在，但它們是在受控條件下拍攝的（靜態場景、兩種曝光），模型可能在上面表現很好卻無法泛化（即

[15] 所報告的泛化落差）。因此幾乎每篇 LLIE 論文都會展示非配對測試集上的結果，而能替這些結果打分數的，只有無參考指標與人的判斷。

2. **「正確」的亮度本身就有歧義**。即使有參考影像，正常光照的照片也只是某位攝影師對曝光與色彩的一種選擇。稍微亮一點或暖一點的輸出可能一樣好看，PSNR 卻會掉好幾 dB（這正是下面 GT-mean 協定出現的原因）。無參考指標至少不會懲罰一個不同但合理的曝光。

代價是：無參考指標本身是一個人類偏好的模型，而每個模型都有盲點。無參考影像品質評估（image quality assessment, IQA）的一般理論，包括 NIQE、BRISQUE、MUSIQ 與 CLIP-IQA 的建構方式，以及 NR 指標如何以人類意見驗證，請見 DLCV 09。以下各小節不再重複，而是聚焦在每一類指標面對增強後的低光影像時的表現。

經典的自然場景統計指標：NIQE、BRISQUE 與 PI

NIQE [58]、BRISQUE [59] 與感知指標（Perceptual Index, PI）[60], [61] 都是自然場景統計（NSS）指標：它們量測影像的統計量（MSCN 係數）離乾淨照片的統計量有多遠，三者都是越低越好。MSCN 係數與 BRISQUE、NIQE 的建構方式在 DLCV 09 中說明，並附有可執行的程式碼，展示模糊、雜訊與過度銳化如何移動這些特徵；PI 的定義則在 DLCV 09 補充資料中。低光論文大量使用這些指標；例如 Zero-DCE [18] 就在非配對測試集上回報 PI，並搭配使用者研究。

它們獎勵什麼，以及在曝光問題上如何誤導。這三個指標都是為結構性失真（模糊、雜訊、壓縮、振鈴）設計的，不是為曝光或色彩：

- MSCN 正規化在定義上就把局部平均與對比除掉了，所以整體亮度的改變幾乎不會影響特徵。增強不足的影像與曝光良好的影像可能拿到相近的分數。
- 灰階特徵忽略色偏與飽和度，而這正是圖 2 裡的錯誤。
- 雜訊與過度銳化會增加高頻能量，對這些統計量來說，一張雜訊被放大、紋理「豐富」的輸出，可能比一張乾淨但略柔的輸出更「自然」。強烈局部對比的 HE 與 CLAHE 輸出就是典型例子。
- 用來定義「高品質」的影像集與訓練用的失真裡，很少有暗景、夜景或逆光場景，所以「自然」其實等於「像白天」。

無參考評估需要互補檢查

LOE [13] 看亮度順序變動，不檢查夠不夠亮；色彩豐富度 [62] 可能因色雜訊上升；雜訊估計 [63] 須區分真紋理。學得品質模型也受訓練域限制，可能少見極暗圖。

結合不同指標家族、全解析裁切，明示前處理和實作。人類比較要隨機順序、說明環境與問題；喜好和任務用途不同。測試時雖有來源圖，來源亮度並非目標亮度。

僅線上版：理解無參考低光指標 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/notes/理解無參考低光指標/*

GT-mean 陷阱

白話版。在成對基準上，整體亮度的一點點差異就會大幅改變 PSNR。有些論文在評分前，會把每張輸出重新縮放，讓它的平均亮度與真值一致。但實際使用時，你永遠拿不到這個資訊。

精確版。「GT-mean」評估把輸出乘上 $\lambda = \mathbb{E}[y]/\mathbb{E}[f(x)]$，再對 $\lambda f(x)$ 計算指標，$\mathbb{E}[\cdot]$ 為影像平均。由於 LLIE 的「正確」亮度本來就有歧義，這樣做可以大幅提高 PSNR。Retinexformer 的作者在程式碼中提供了這個模式，以便與 LLFlow、KinD 及近期擴散模型等方法比較，但明確表示不建議使用，因為它用真值來修正輸出 [22]。Liao 等人 [64]（2025）研究輸出與參考之間的亮度不匹配，提出用於訓練的 GT-mean 損失函數，並同時回報標準與 GT-mean 指標。讀表格時：先確認使用哪一種設定，絕不混用兩者，並優先選擇在標準協定下表現好的方法。

以任務為導向的評估

由於 LLIE 常是前處理步驟，另一種檢驗方式是在增強後的影像上跑偵測或分割模型（例如在 ExDark [65] 上）並回報準確率。Liu 與 Fan 2025 年的綜述 [66] 發現兩者脫節：監督式方法常產生感知品質高的影像，對視覺任務的提升卻有限；零樣本（zero-shot）方法雖然畫質分數較低，在各種任務上的提升反而更一致。Liu 等人的基準 [67] 也評估了低光下的人臉偵測。完整的評估應該回報保真度、無參考品質、任務表現與執行時間。

基準資料集

白話版。資料集的差別在於：有沒有成對、暗是真實的還是合成的、是 raw 還是 sRGB，以及有沒有偵測用的標註。

資料集	類型	內容與用途
LOL (v1) [16]	成對、真實、sRGB	改變曝光拍攝的 500 組低光/正常光影像（485 組訓練／15 組測試），400×600。最常用的 LLIE 基準；測試集很小。
LOL-v2 [68]	成對，真實＋合成子集	較大的 *Real* 子集（實拍成對）與合成成對資料的 *Synthetic* 子集。兩個子集要分開回報。
LSRW [69]	成對、真實	5,650 組，來自 Nikon D7500（3,170 組）與 Huawei P40 Pro（2,480 組）；5,600 組訓練、50 組測試。測試跨裝置泛化。
SDSD [70]	成對影片	以機電系統對齊、拍攝兩次的動態室內與室外場景；常拆成影格做影像測試。
SID [48]	成對、raw	5,094 張短曝光 raw 影像，搭配 424 張長曝光參考（Sony 與 Fuji）。極低光、raw 到 sRGB。
MIT-Adobe FiveK [71]	成對、修圖	5,000 張 raw 照片，每張由五位專家修圖。用於色調與色彩增強（照片修圖）。
SICE [72]	多重曝光序列	每個場景從曝光不足到過曝的多張拍攝與一張參考；用來訓練曝光校正（例如 Zero-DCE）。
VE-LOL [67]	成對＋偵測	VE-LOL-L 用於增強，VE-LOL-H 附有人臉標註，用於高階任務評估。

資料集	類型	內容與用途
ExDark [65]	非配對、有標註	7,363 張低光影像、12 個物件類別、10 種光照條件。以任務為導向的評估（偵測）。
LIME [4]、MEF [56]、NPE [13]、DICM [14]、VV [57]	非配對測試集	小型的真實低光或光照不均照片集，用於視覺比較與無參考指標。

挑戰賽：NTIRE 2024 與 2025

CVPR 的 NTIRE workshop（這個系列的整體介紹見 DLCV 09）連續兩年舉辦 LLIE 挑戰賽。**NTIRE 2024** 挑戰賽 [73] 涵蓋超高解析度（4K 以上）、不均勻照明、逆光、極暗與夜景等情境，共有 428 人註冊、22 隊有效提交。**NTIRE 2025** 挑戰賽 [74] 有 762 人註冊、28 隊有效提交。這些報告整理了勝出隊伍的設計，是「在固定且公平的協定下什麼方法有效」的一個好切面。

現代觀點

Li 等人 [15] 的共同規則整理訓練策略、泛化、成本及真實手機資料，提醒配對基準不必然轉移到手機。Zheng [75] 補混合曝光／影片，VE-LOL [67] 與任務研究 [66] 區分外觀和用途；擴散綜述 [76] 按條件、表示與程序分六家族。綜述是有日期的地圖，新方法仍看原文。

2026 高效率增強。 NTIRE [77] 以模型大小約束及多品質排名聚合，檔案小不等於手機延遲低。報告呈現精簡編解碼、色彩／照明先驗及重參數化；HDRNet ／ LUT 的修圖結果則需另驗證低光用途。

目前的最先進技術（2023–2026）。主要有三個方向。第一，**Retinex 引導的骨幹網路**：保留「光與反射率」結構的 Transformer（Retinexformer [22]）與狀態空間模型（RetinexMamba [38]）。第二，**生成模型**：擴散模型（Diff-Retinex [24]、GSAD [25]、LightenDiffusion [26]、Reti-Diff [27]）與 flow（LLFlow [23]）。第三，**更好的表示法與評估協定**：學習出來的色彩空間（HVI-CIDNet [6]）、考慮亮度不匹配的訓練與評估（GT-Mean loss [64]）、語言引導與 all-in-one 復原（CLIP-LIT [42]、InstructIR [49]），以及任務感知的增強（MAET [50]、FeatEnHancer [51]）。

尚未解決的問題。

- **真實雜訊：**合成 sRGB 與感光處理後不一定相符，需改善模型 [2]、RAW [48] 或聯合復原。
- **時間一致性：**逐幀閃爍，SDSD 配對 [70] 稀少，生成模型尤其難。
- **評估：**目標亮度不唯一、GT-mean、NR 與任務落差都影響比較。
- **裝置效率：** Zero-DCE++ [19]、SCI [21]、小先驗網路 [39] 展示可行方向；擴散須計所有步驟。
- **泛化：**跨相機、過曝／欠曝、背光與彩色光仍困難 [15]。

僅線上版：*KinD*：修復反射率，另外控制照明 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/kind 修復反射率另外控制照明/*

僅線上版：*IAT*：局部仿射圖與全域相機式轉換 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/iat 局部仿射圖與全域相機式轉換/*；*Multinex*：以

解析先驗融合出 *Retinex* 殘差 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/multinex 以解析先驗融合出-retinex-殘差/*

僅線上版：*CPGA-Net*：通道先驗、*gamma* 估計與學習式融合 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/cpga-net 通道先驗 gamma-估計與學習式融合/*；*CPGA-Net+*：注意力、*BDSF* 與較強版本 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/low-light-enhancement/papers/cpga-net 注意力 bdsf-與較強版本/*

重點整理

- 變亮只改變色調，復原還須處理雜訊、受損細節與色彩。
- 亮度通道選擇會影響色彩與飽和度，應註明編碼及數值範圍。
- Retinex 分解依賴照明／反射假設，不是唯一可觀測的物理分離。
- 把監督方式與設計分開：配對、非配對或零參考訓練可搭配不同架構。
- 空間變化的增強可能改變像素排序；曝光與平滑先驗在雜訊或鮮豔場景中可能失效。
- 分開報告 GT-mean 與標準協定，並評估下游任務，不假設越亮越好。
- 搭配互補品質指標與使用者研究，單一分數無法涵蓋曝光、色彩、雜訊與忠實度。

練習

1. **單調曲線**。證明 $\mathrm{LE}(x;\alpha) = x + \alpha x(1-x)$ 在 $[0,1]$ 上單調不減，若且唯若 $|\alpha| \le 1$。$\alpha = 1.5$ 時會出什麼問題？

   提示

   導數為 $1+\alpha(1-2x)$，對 x 是線性的，所以在 $[0,1]$ 上的最小值出現在端點：$x=0$ 時為 $1+\alpha$，$x=1$ 時為 $1-\alpha$。兩者都 ≥ 0 恰好等價於 $|\alpha| \le 1$。$\alpha = 1.5$ 時，$x=1$ 處的斜率為 -0.5，接近白色的像素會隨輸入變亮而變暗，明暗順序反轉，在亮部造成假影。

2. **迭代曲線的增益**。對很小的輸入 x，證明以固定 α 迭代八次大約會把 x 乘上 $(1+\alpha)^8$。用這個結果解釋為什麼零參考曲線方法會放大暗部雜訊，並提出一個能減輕問題的損失函數或架構修改。

   提示

   x 很小時 $x(1-x) \approx x$，所以每一步都是 $x \mapsto (1+\alpha)x$。$\alpha = 1$ 時就是 $2^8 = 256$，標準差為 σ 的雜訊變成 256σ。可能的修正：加入去噪分支或考慮雜訊的損失、在估計訊噪比低的地方限制 α（類似 SNR-aware 的精神），或懲罰輸出中、平滑後的輸入所沒有的高頻能量。

3. **兩個亮度公式**。一個 8 位元像素為 $(R', G', B') = (200, 100, 50)$。計算它的 BT.601 全範圍亮度與有限範圍亮度，並說明兩者編碼的是同一個量。

   提示

   全範圍：$0.299\cdot 200 + 0.587\cdot 100 + 0.114\cdot 50 = 124.2$。有限範圍：$16 + 0.257\cdot 200 + 0.504\cdot 100 + 0.098\cdot 50 \approx 122.7$。換算回去：$(122.7 - 16)\cdot 255/219 \approx 124.2$。唯一的差別是比例 $219/255$ 與偏移量 16。

4. **沒有真值的 Retinex 損失**。在 Retinex-Net 中，為什麼需要 $\mathcal{L}_{ir} = \|R_{\text{low}} - R_{\text{normal}}\|_1$？描述一個能讓 $\mathcal{L}_{\text{recon}}$ 的自我重建項與 $\mathcal{L}_{is}$ 都變成零、卻毫無用處的分解，並說明 $\mathcal{L}_{ir}$ 如何排除它。

   提示

   令 $I_i \equiv 1$（完全平滑）且 $R_i = S_i$。每張影像都能被完美重建，平滑損失也為零，但根本沒有做任何分解。由於 $S_{\text{low}} \neq S_{\text{normal}}$，這會讓 $R_{\text{low}} \neq R_{\text{normal}}$，所以 $\mathcal{L}_{ir}$ 很大。要求兩者反射率相等，就迫使兩張影像之間的亮度差異被放進照度裡。

5. **手算 GT-mean**。你的模型輸出平均為 0.40，真值平均為 0.50，除了這個整體倍率之外輸出完美無缺。GT-mean 的縮放因子是多少？如果影像是常數，縮放前後的 PSNR 各是多少？為什麼縮放後的數字無法公平地與正常評估的方法比較？

   提示

   $\lambda = 0.50/0.40 = 1.25$。縮放前：MSE $= 0.1^2 = 0.01$，PSNR $= 20$ dB。縮放後輸出等於參考，誤差為 0，PSNR 趨於無窮大。實際使用時方法永遠拿不到真值的平均，所以 GT-mean 數字獎勵的是正確的結構，卻忽略了任務中最難的部分：決定亮度。

6. **更換色彩先驗**。在 PyTorch 小玩具中，把灰色世界損失換成保留輸入色度的損失，例如 $\|y/\sum_c y_c - x/\sum_c x_c\|_1$。重新訓練，並把咖啡杯的結果與圖 7 比較。兩種先驗各在什麼情況下較好？

   提示

   分母要加上一個小 epsilon。保留輸入色度能讓紅褐色的杯子維持紅色，但也會保留光源的色偏（例如鈉燈）。灰色世界能去除色偏，卻會讓真正色彩鮮豔的場景褪色。實際的方法常結合較弱的色彩先驗，與從資料學到的色彩校正。

參考文獻

1. Shyandram, “簡介 Deep Learning Low-Light Image Enhancement (LLIE)” (Introduction to deep-learning low-light image enhancement, in Chinese), Medium post, 2024; updated 2026. link
2. A. Foi, M. Trimeche, V. Katkovnik and K. Egiazarian, “Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data,” *IEEE Transactions on Image Processing*, vol. 17, no. 10, pp. 1737–1754, 2008. doi
3. K. He, J. Sun and X. Tang, “Single image haze removal using dark channel prior,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 12, pp. 2341–2353, 2011. doi
4. X. Guo, Y. Li and H. Ling, “LIME: Low-light image enhancement via illumination map estimation,” *IEEE Transactions on Image Processing*, vol. 26, no. 2, pp. 982–993, 2017. doi
5. ITU-R, “Recommendation BT.601-7: Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios,” 2011. ITU
6. Q. Yan, Y. Feng, C. Zhang, et al., “HVI: A new color space for low-light image enhancement,” in *Proc. IEEE/CVF CVPR*, 2025. arXiv

7. S. M. Pizer, E. P. Amburn, J. D. Austin, et al., “Adaptive histogram equalization and its variations,” *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, pp. 355–368, 1987. doi
8. K. Zuiderveld, “Contrast limited adaptive histogram equalization,” in *Graphics Gems IV*, Academic Press, 1994, pp. 474–485. doi
9. E. H. Land, “The retinex theory of color vision,” *Scientific American*, vol. 237, no. 6, pp. 108–128, 1977. doi
10. D. J. Jobson, Z. Rahman and G. A. Woodell, “Properties and performance of a center/surround retinex,” *IEEE Transactions on Image Processing*, vol. 6, pp. 451–462, 1997. doi
11. D. J. Jobson, Z. Rahman and G. A. Woodell, “A multiscale retinex for bridging the gap between color images and the human observation of scenes,” *IEEE Transactions on Image Processing*, vol. 6, no. 7, pp. 965–976, 1997. doi
12. X. Dong, Y. Pang and J. Wen, “Fast efficient algorithm for enhancement of low lighting video,” *ACM SIGGRAPH 2010 Posters*, 2010. SIGGRAPH history doi
13. S. Wang, J. Zheng, H.-M. Hu and B. Li, “Naturalness preserved enhancement algorithm for non-uniform illumination images,” *IEEE Transactions on Image Processing*, vol. 22, no. 9, pp. 3538–3548, 2013. doi
14. C. Lee, C. Lee and C.-S. Kim, “Contrast enhancement based on layered difference representation of 2D histograms,” *IEEE Transactions on Image Processing*, vol. 22, no. 12, pp. 5372–5384, 2013. doi
15. C. Li, C. Guo, L. Han, J. Jiang, M.-M. Cheng, J. Gu and C. C. Loy, “Low-light image and video enhancement using deep learning: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 12, pp. 9396–9416, 2022. arXiv
16. C. Wei, W. Wang, W. Yang and J. Liu, “Deep retinex decomposition for low-light enhancement,” in *Proc. British Machine Vision Conference (BMVC)*, 2018. arXiv project
17. Y. Jiang, X. Gong, D. Liu, et al., “EnlightenGAN: Deep light enhancement without paired supervision,” *IEEE Transactions on Image Processing*, vol. 30, pp. 2340–2349, 2021. arXiv
18. C. Guo, C. Li, J. Guo, et al., “Zero-reference deep curve estimation for low-light image enhancement,” in *Proc. IEEE/CVF CVPR*, 2020, pp. 1777–1786. arXiv
19. C. Li, C. Guo and C. C. Loy, “Learning to enhance low-light image via zero-reference deep curve estimation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, 2022. arXiv
20. R. Liu, L. Ma, J. Zhang, X. Fan and Z. Luo, “Retinex-inspired unrolling with cooperative prior architecture search for low-light image enhancement,” in *Proc. IEEE/CVF CVPR*, 2021, pp. 10556–10565. arXiv
21. L. Ma, T. Ma, R. Liu, X. Fan and Z. Luo, “Toward fast, flexible, and robust low-light image enhancement,” in *Proc. IEEE/CVF CVPR*, 2022, pp. 5627–5636. arXiv
22. Y. Cai, H. Bian, J. Lin, H. Wang, R. Timofte and Y. Zhang, “Retinexformer: One-stage Retinex-based transformer for low-light image enhancement,” in *Proc. IEEE/CVF ICCV*, 2023. arXiv code
23. Y. Wang, R. Wan, W. Yang, H. Li, L.-P. Chau and A. C. Kot, “Low-light image enhancement with normalizing flow,” in *Proc. AAAI Conference on Artificial Intelligence*, vol. 36, no. 3, pp. 2604–2612, 2022. arXiv
24. X. Yi, H. Xu, H. Zhang, L. Tang and J. Ma, “Diff-Retinex: Rethinking low-light image enhancement with a generative diffusion model,” in *Proc. IEEE/CVF ICCV*, 2023. arXiv

25. J. Hou, Z. Zhu, J. Hou, H. Liu, H. Zeng and H. Yuan, "Global structure-aware diffusion process for low-light image enhancement," in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2023. arXiv
26. H. Jiang, A. Luo, X. Liu, S. Han and S. Liu, "LightenDiffusion: Unsupervised low-light image enhancement with latent-Retinex diffusion models," in *Proc. European Conference on Computer Vision* (*ECCV*), 2024. arXiv
27. C. He, C. Fang, Y. Zhang, et al., "Reti-Diff: Illumination degradation image restoration with Retinex-based latent diffusion model," in *Proc. International Conference on Learning Representations* (*ICLR*), 2025. proceedings
28. M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff and F. Durand, "Deep bilateral learning for real-time image enhancement," *ACM Transactions on Graphics* (Proc. SIGGRAPH), vol. 36, no. 4, art. 118, 2017. doi arXiv
29. H. Zeng, J. Cai, L. Li, Z. Cao and L. Zhang, "Learning image-adaptive 3D lookup tables for high performance photo enhancement in real-time," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, 2022. doi arXiv
30. K. G. Lore, A. Akintayo and S. Sarkar, "LLNet: A deep autoencoder approach to natural low-light image enhancement," *Pattern Recognition*, vol. 61, pp. 650–662, 2017. doi
31. L. Tao, C. Zhu, J. Song, T. Lu, H. Jia and X. Xie, "Low-light image enhancement using CNN and bright channel prior," in *Proc. IEEE International Conference on Image Processing* (*ICIP*), 2017, pp. 3215–3219. doi
32. S. W. Zamir, A. Arora, S. Khan, et al., "Learning enriched features for real image restoration and enhancement," in *Proc. European Conference on Computer Vision* (*ECCV*), 2020, pp. 492–511. arXiv
33. X. Xu, R. Wang, C.-W. Fu and J. Jia, "SNR-aware low-light image enhancement," in *Proc. IEEE/CVF CVPR*, 2022, pp. 17693–17703. doi
34. Z. Cui, K. Li, L. Gu, S. Su, P. Gao, Z. Jiang, Y. Qiao and T. Harada, "You only need 90K parameters to adapt light: A light weight transformer for image enhancement and exposure correction," in *Proc. British Machine Vision Conference* (*BMVC*), 2022. paper arXiv
35. X. Ding, X. Zhang, N. Ma, J. Han, G. Ding and J. Sun, "RepVGG: Making VGG-style ConvNets great again," in *Proc. IEEE/CVF CVPR*, 2021, pp. 13728–13737. doi arXiv
36. Y. Zhang, J. Zhang and X. Guo, "Kindling the darkness: A practical low-light image enhancer," in *Proc. ACM International Conference on Multimedia* (*ACM MM*), 2019, pp. 1632–1640. doi
37. Y. Zhang, X. Guo, J. Ma, W. Liu and J. Zhang, "Beyond brightening low-light images," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 1013–1037, 2021. doi
38. J. Bai, Y. Yin, Q. He, Y. Li and X. Zhang, "RetinexMamba: Retinex-based Mamba for low-light image enhancement," arXiv:2405.03349, 2024. arXiv
39. S.-E. Weng, S.-G. Miaou and R. Christanto, "A lightweight low-light image enhancement network via channel prior and gamma correction," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 39, no. 12, art. 2554013, 2025. doi code manuscript
40. S.-E. Weng, C.-Y. Hsiao, L.-W. Lu, et al., "Rethinking theoretical illumination for efficient low-light image enhancement" (CPGA-Net+), arXiv:2409.05274, 2024. arXiv
41. W. Wu, J. Weng, P. Zhang, X. Wang, W. Yang and J. Jiang, "URetinex-Net: Retinex-based deep unfolding network for low-light image enhancement," in *Proc. IEEE/CVF CVPR*, 2022, pp. 5891–5900. doi

42. Z. Liang, C. Li, S. Zhou, R. Feng and C. C. Loy, “Iterative prompt learning for unsupervised backlit image enhancement,” in *Proc. IEEE/CVF ICCV*, 2023. arXiv
43. C. Yang, M. Jin, X. Jia, Y. Xu and Y. Chen, “AdaInt: Learning adaptive intervals for 3D lookup tables on real-time image enhancement,” in *Proc. IEEE/CVF CVPR*, 2022, pp. 17522–17531. CVF arXiv
44. C. Yang, M. Jin, Y. Xu, R. Zhang, Y. Chen and H. Liu, “SepLUT: Separable image-adaptive lookup tables for real-time image enhancement,” in *Proc. European Conference on Computer Vision (ECCV)*, 2022, pp. 201–217. doi arXiv
45. T. Wang, Y. Li, J. Peng, Y. Ma, X. Wang, F. Song and Y. Yan, “Real-time image enhancer via learnable spatial-aware 3D lookup tables,” in *Proc. IEEE/CVF ICCV*, 2021, pp. 2471–2480. CVF arXiv
46. C. Liu, H. Yang, J. Fu and X. Qian, “4D LUT: Learnable context-aware 4D lookup table for image enhancement,” *IEEE Transactions on Image Processing*, vol. 32, pp. 4742–4756, 2023. doi arXiv
47. W. Kim and N. I. Cho, “Image-adaptive 3D lookup tables for real-time image enhancement with bilateral grids,” in *Proc. European Conference on Computer Vision (ECCV)*, 2024, pp. 91–108. doi ECCV
48. C. Chen, Q. Chen, J. Xu and V. Koltun, “Learning to see in the dark,” in *Proc. IEEE/CVF CVPR*, 2018, pp. 3291–3300. arXiv
49. M. V. Conde, G. Geigle and R. Timofte, “InstructIR: High-quality image restoration following human instructions,” in *Proc. European Conference on Computer Vision (ECCV)*, 2024. arXiv
50. Z. Cui, G.-J. Qi, L. Gu, S. You, Z. Zhang and T. Harada, “Multitask AET with orthogonal tangent regularity for dark object detection,” in *Proc. IEEE/CVF ICCV*, 2021. arXiv
51. K. A. Hashmi, G. Kallempudi, D. Stricker and M. Z. Afzal, “FeatEnHancer: Enhancing hierarchical features for object detection and beyond under low-light vision,” in *Proc. IEEE/CVF ICCV*, 2023. arXiv
52. M. Dehghani, A. Arnab, L. Beyer, A. Vaswani and Y. Tay, “The efficiency misnomer,” in *Proc. International Conference on Learning Representations (ICLR)*, 2022. ICLR arXiv
53. A. Ignatov, R. Timofte, S. Liu, et al., “Learned smartphone ISP on mobile GPUs with deep learning, Mobile AI & AIM 2022 challenge: Report,” in *Proc. European Conference on Computer Vision (ECCV) Workshops*, 2023, pp. 44–70. doi arXiv
54. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
55. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *Proc. IEEE/CVF CVPR*, 2018. arXiv
56. K. Ma, K. Zeng and Z. Wang, “Perceptual quality assessment for multi-exposure image fusion,” *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3345–3356, 2015. doi
57. V. Vonikakis, “Datasets” (source of the VV test images), personal web page. link
58. A. Mittal, R. Soundararajan and A. C. Bovik, “Making a ‘completely blind’ image quality analyzer,” *IEEE Signal Processing Letters*, vol. 20, no. 3, pp. 209–212, 2013. doi
59. A. Mittal, A. K. Moorthy and A. C. Bovik, “No-reference image quality assessment in the spatial domain,” *IEEE Transactions on Image Processing*, vol. 21, no. 12, pp. 4695–4708, 2012. doi
60. Y. Blau, R. Mechrez, R. Timofte, T. Michaeli and L. Zelnik-Manor, “The 2018 PIRM challenge on perceptual image super-resolution,” in *Proc. European Conference on Computer Vision (ECCV)*

Workshops, 2018. arXiv
61. C. Ma, C.-Y. Yang, X. Yang and M.-H. Yang, "Learning a no-reference quality metric for single-image super-resolution," *Computer Vision and Image Understanding*, vol. 158, pp. 1–16, 2017. doi arXiv
62. D. Hasler and S. E. Süsstrunk, "Measuring colorfulness in natural images," in *Proc. SPIE 5007, Human Vision and Electronic Imaging VIII*, 2003. doi
63. J. Immerkær, "Fast noise variance estimation," *Computer Vision and Image Understanding*, vol. 64, no. 2, pp. 300–302, 1996. doi
64. J. Liao, S. Hao, R. Hong and M. Wang, "GT-Mean loss: A simple yet effective solution for brightness mismatch in low-light image enhancement," in *Proc. IEEE/CVF ICCV*, 2025. arXiv
65. Y. P. Loh and C. S. Chan, "Getting to know low-light images with the Exclusively Dark dataset," *Computer Vision and Image Understanding*, vol. 178, pp. 30–42, 2019. arXiv
66. F. Liu and L. Fan, "A review of advancements in low-light image enhancement using deep learning," *Neurocomputing*, vol. 652, art. 131052, 2025. arXiv
67. J. Liu, D. Xu, W. Yang, M. Fan and H. Huang, "Benchmarking low-light image enhancement and beyond," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 1153–1184, 2021. doi project
68. W. Yang, W. Wang, H. Huang, S. Wang and J. Liu, "Sparse gradient regularized deep Retinex network for robust low-light image enhancement," *IEEE Transactions on Image Processing*, vol. 30, pp. 2072–2086, 2021. doi
69. J. Hai, Z. Xuan, R. Yang, et al., "R2RNet: Low-light image enhancement via Real-low to Real-normal Network," *Journal of Visual Communication and Image Representation*, vol. 90, art. 103712, 2023. arXiv
70. R. Wang, X. Xu, C.-W. Fu, et al., "Seeing dynamic scene in the dark: A high-quality video dataset with mechatronic alignment," in *Proc. IEEE/CVF ICCV*, 2021, pp. 9680–9689. doi code
71. V. Bychkovsky, S. Paris, E. Chan and F. Durand, "Learning photographic global tonal adjustment with a database of input/output image pairs," in *Proc. IEEE CVPR*, 2011, pp. 97–104. dataset
72. J. Cai, S. Gu and L. Zhang, "Learning a deep single image contrast enhancer from multi-exposure images," *IEEE Transactions on Image Processing*, vol. 27, no. 4, pp. 2049–2062, 2018. doi
73. X. Liu, Z. Wu, A. Li, et al., "NTIRE 2024 challenge on low light image enhancement: Methods and results," in *Proc. IEEE/CVF CVPR Workshops*, 2024. arXiv
74. X. Liu, Z. Wu, F.-A. Vasluianu, et al., "NTIRE 2025 challenge on low light image enhancement: Methods and results," in *Proc. IEEE/CVF CVPR Workshops*, 2025. arXiv
75. S. Zheng, Y. Ma, J. Pan, C. Lu and G. Gupta, "Low-light image and video enhancement: A comprehensive survey and beyond," arXiv:2212.10772, 2022. arXiv
76. E. Adhikarla, Y. Liu and B. D. Davison, "Diffusion models for low-light image enhancement: A multi-perspective taxonomy and performance analysis," arXiv:2510.05976, 2025. arXiv
77. J. Yan, C. Tu, W. Zhang, Z. Wang, P. Cao, Q. Lin, Y. Fang, X. Liu, Z. Wu, Z. Zhou and R. Timofte, "NTIRE 2026 challenge on efficient low light image enhancement: Methods and results," NTIRE 2026 workshop report, arXiv:2605.02212, 2026. arXiv
78. O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. Medical Image Computing and Computer-Assisted Intervention (MICCAI)*,

2015, pp. 234–241. doi arXiv
79. J. Chen, A. Adams, N. Wadhwa and S. W. Hasinoff, "Bilateral guided upsampling," *ACM Transactions on Graphics* (Proc. SIGGRAPH Asia), vol. 35, no. 6, 2016. doi project
80. K. He, J. Sun and X. Tang, "Guided image filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013. doi
81. C. Chen and J. Mo, "IQA-PyTorch: PyTorch toolbox for image quality assessment," software (pip package `pyiqa`). GitHub
82. J. Ke, Q. Wang, Y. Wang, P. Milanfar and F. Yang, "MUSIQ: Multi-scale image quality transformer," in *Proc. IEEE/CVF ICCV*, 2021. arXiv
83. J. Wang, K. C. K. Chan and C. C. Loy, "Exploring CLIP for assessing the look and feel of images," in *Proc. AAAI Conference on Artificial Intelligence*, 2023. arXiv
84. V. Kannan, S. Malik, N. C. Babu and R. Soundararajan, "Quality assessment of low-light restored images: A subjective study and an unsupervised model," *IEEE Access*, vol. 11, pp. 68216–68230, 2023. doi arXiv
85. ITU-R, "Recommendation BT.500-15: Methodologies for the subjective assessment of the quality of television images," 2023. ITU
86. R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: The method of paired comparisons," *Biometrika*, vol. 39, no. 3–4, pp. 324–345, 1952. doi
87. H. Talebi and P. Milanfar, "NIMA: Neural image assessment," *IEEE Transactions on Image Processing*, 2018. arXiv
88. S. Yang, T. Wu, S. Shi, et al., "MANIQA: Multi-dimension attention network for no-reference image quality assessment," in *Proc. IEEE/CVF CVPR Workshops*, 2022, pp. 1190–1199. doi arXiv
89. H. Wu, Z. Zhang, W. Zhang, C. Chen, et al., "Q-Align: Teaching LMMs for visual scoring via discrete text-defined levels," in *Proc. International Conference on Machine Learning (ICML)*, PMLR 235, 2024. arXiv
90. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861, 2017. arXiv
91. M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF CVPR*, 2018, pp. 4510–4520. doi arXiv
92. W.-S. Lai, J.-B. Huang, O. Wang, E. Shechtman, E. Yumer and M.-H. Yang, "Learning blind video temporal consistency," in *Proc. European Conference on Computer Vision (ECCV)*, 2018, pp. 179–195. doi arXiv
93. S. W. Hasinoff, D. Sharlet, R. Geiss, A. Adams, J. T. Barron, F. Kainz, J. Chen and M. Levoy, "Burst photography for high dynamic range and low-light imaging on mobile cameras," *ACM Transactions on Graphics* (Proc. SIGGRAPH Asia), vol. 35, no. 6, 2016. doi
94. J. Chen, S. Paris and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Transactions on Graphics* (Proc. SIGGRAPH), vol. 26, no. 3, art. 103, 2007. doi
95. C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 1998, pp. 839–846. doi
96. S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *International Journal of Computer Vision*, vol. 81, no. 1, pp. 24–52, 2009. doi
97. H. Wu, S. Zheng, J. Zhang and K. Huang, "Fast end-to-end trainable guided filter," in *Proc. IEEE/CVF CVPR*, 2018, pp. 1838–1847. doi arXiv

98. R. Wang, Q. Zhang, C.-W. Fu, X. Shen, W.-S. Zheng and J. Jia, “Underexposed photo enhancement using deep illumination estimation,” in *Proc. IEEE/CVF CVPR*, 2019, pp. 6842–6850. doi CVF
99. Y. Lin, Z. Fu, K. Wen, T. Ye, S. Chen, G. Meng, Y. Wang, et al., “DPLUT: Unsupervised low-light image enhancement with lookup tables and diffusion priors,” in *Proc. AAAI Conference on Artificial Intelligence*, vol. 39, no. 5, pp. 5316–5324, 2025. doi arXiv
100. K. Dabov, A. Foi, V. Katkovnik and K. Egiazarian, “Image denoising by sparse 3-D transform-domain collaborative filtering,” *IEEE Transactions on Image Processing*, 2007. doi
101. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *ICLR*, 2015. arXiv
102. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” *CVPR*, 2009. doi
103. A. Brateanu, T. Mu, C. O. Ancuti and C. Ancuti, “Multinex: Lightweight Low-light Image Enhancement via Multi-prior Retinex,” in *Proc. IEEE/CVF CVPR*, 2026. paper supplement

DLCV 10・影像融合：互補來源與有用輸出

從多焦點與多曝光實驗，走到跨模態、學習式與任務感知融合，理解來源保留、配準、品質指標與實際用途。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/

你將學到

- 定義紅外線—可見光、多焦點或多曝光融合需要保留的資訊。
- 比較平均、決策圖、品質加權與多尺度混合。
- 理解缺少目標影像時，損失如何定義學得的融合結果。
- 評估配準假設與下游任務目標。
- 解讀互相衝突的融合指標，並檢查典型偽影。

先看全貌

許多相機看同一個場景的方式各不相同。熱像儀在黑暗中看得到人，卻看不到他身後牆面的紋理。對焦在前景的鏡頭會把背景拍糊。短曝光保住天空，卻丟了陰影。**影像融合**（image fusion）取兩張或更多這樣的影像，合成一張保留各自優點的影像。

> **用最簡單的話說**
>
> 想像三個朋友各自拍了同一個遊樂場。一張只有鞦韆是清楚的，一張只有溜滑梯是清楚的，還有一張是戴著特殊眼鏡拍的，看得出哪些小朋友身上是暖的。你想要一張可以掛在牆上的照片。你可以把每張照片最好的部分剪下來拼貼在一起，但要很小心：剪得不好，別人會看到接縫；如果幾張照片不是站在同一個位置拍的，剪下來的部分也對不上。影像融合就是這種剪貼，只是做得很仔細，而且是一個像素一個像素地做。

為什麼重要：

- **夜間駕駛、監視與搜救**會同時使用可見光與熱像相機。駕駛或操作員只看一個螢幕，所以兩種畫面必須合成一張，同時顯示溫暖的行人與路面標線。
- **攝影也會用到融合**。一組包圍曝光可以合成一張同時有明亮窗戶與昏暗房間的照片；焦點堆疊讓微距與顯微鏡照片從前到後都清楚。
- **融合在影像任務中很特別：沒有正確答案可以照抄**。沒有人擁有一張同時是熱像又是可見光的影像。光是這一點，就決定了整個領域：方法怎麼設計、網路怎麼訓練，以及評估為什麼這麼難。

本篇建立在前面兩篇之上。DLCV 09 用一段文字把影像融合放進復原的分類中；這裡才完整講它。IP 07 建立了小波融合會用到的小波轉換；本篇補上 IP 07 沒有涵蓋的拉普拉斯金字塔。至於一般的無參考影像品質評估，請見 DLCV 09；這裡只談融合特有的指標。

什麼是影像融合

白話版。融合把同一場景的互補觀測整合成一張影像，須先定義輸出應保留各來源的哪些資訊。

精確版。給定 K 張尺寸為 $H \times W$ 且已對位的來源影像 $I_1, \dots, I_K$，融合方法是一個映射

$$F = \Phi(I_1, \dots, I_K), \qquad F \in \mathbb{R}^{H \times W \times C},$$

F 是融合影像，C 是通道數（灰階為 1、彩色為 3），Φ 是人工設計或學得的融合規則。「已配準」表示各來源的同一像素位置 (i, j) 對應場景中的同一點。許多傳統方法會對像素或轉換係數做**加權組合**；在像素域中，例如：

$$F(i,j) = \sum_{k=1}^{K} w_k(i,j)\, I_k(i,j), \qquad \sum_{k=1}^{K} w_k(i,j) = 1, \quad w_k(i,j) \geq 0,$$

$w_k(i,j)$ 是來源 k 在像素 (i,j) 的權重。這個凸組合模型適用於許多像素混合規則，但一般學習式解碼器不必受限於來源的逐像素值域。遙測融合也可能輸出多波段資料立方體，形狀與輸入不同。

本篇談的是輸出影像的**像素層級**融合 [1]。另一條路是把兩個來源都餵給網路，只在決策層級合併，這樣可以把某一項任務做得很好，但就拿不到人可以直接看、其他任務也可重複使用的融合影像了 [2]。深度融合網路介於兩者之間：它在網路內部合併特徵，但最後仍輸出像素。

> **從一個小實驗開始**
>
> 第一次閱讀時，先看任務比較與平均範例，再跑**多焦點實驗與互動實驗室**。接著學金字塔混合與多曝光融合。挑選深度模型前，先回來看評估：外觀更銳利的結果，仍可能丟失重要證據。各篇論文導讀適合第二輪選讀。

三項任務

本篇的三項任務，差別在於每個來源各提供什麼、對位有多難，以及是否可能有真值（圖 1）。

	紅外線—可見光	多焦點	多曝光
來源	兩種感測器的熱影像（長波紅外線）與可見光影像	同一相機在不同對焦距離拍攝兩張以上影像	同一相機以不同曝光時間拍攝兩張以上影像
各來源的貢獻	紅外線：適當條件下的熱對比；可見光：反射光的紋理、顏色與細節	各張照片中清晰的區域	長曝光的暗部與短曝光的亮部
常見規則	保留紅外線的顯著熱目標與可見光紋理	逐區域**選擇**最清晰的來源	以偏好曝光適當且有細節的權重**混合**
對齊	困難：鏡頭、解析度與外觀不同	相機與場景靜止時較容易	靜態場景較容易；移動物體會產生鬼影
真值？	沒有	僅合成資料有，例如把清晰影像模糊化	沒有；色調映射後的 HDR 只是可能參考，並非唯一答案

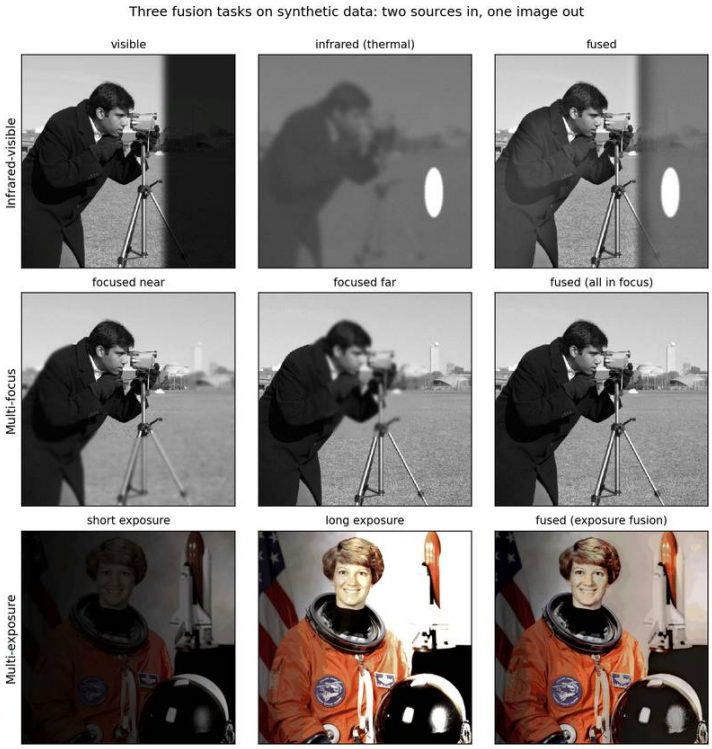


圖 *1* — 以 *scikit-image* 測試影像建立的合成資料上的三項融合任務：紅外線與可見光（金字塔融合）、多焦點（決策圖），以及多曝光（兩張照片的曝光融合）。三者都由本篇的程式碼產生。

紅外線—可見光融合。熱相機量測紅外線輻射，受溫度、放射率與反射輻射影響。熱對比足夠的目標可在黑暗中突出，但可見程度仍取決於溫差、大氣與感測器。可見光相機量測反射光，能呈現符合人眼習慣的紋理細節，但暗處訊號很少 [3]、[2]。融合希望同時突顯熱目標與保留背景細節。然而「各取所長」不是唯一規則：同一像素無法同時像熱目標那麼亮，又像可見光陰影那麼暗。

多焦點融合。鏡頭景深有限，某距離清晰，更近與更遠的物體則模糊。把不同對焦設定的照片融合，可形成**全對焦影像**，也稱焦點堆疊或延伸景深。若至少一個來源含有所需的清晰證據，來源選擇可近似這個理想。但在深度邊界，真實散焦可能同時污染兩個來源，兩者都未必正確。因此多焦點方法常採取選擇，而非直接平均。

多曝光融合（multi-exposure fusion, MEF）。相機感測器一次曝光只能記錄有限的亮度範圍，一組包圍曝光則涵蓋較寬的範圍。傳統做法是**高動態範圍**（high-dynamic-range, HDR）成像：從這組照片還原相機響應與場景輻射量 [4]，再用色調對應（tone mapping）運算子壓縮成可顯示的影像。多曝光融合跳過輻射量這一步，直接把 8 位元的曝光混合成一張曝光良好的 8 位元影像 [5]。它不需要相機校正，速度也快；代價是輸出只是一張好看的照片，不是物理量測。

醫學與遙測融合的問題不同

上述三種攝影任務並未涵蓋整個領域；輸出的通道數與物理意義也可能改變。

情境	互補證據	期望輸出	主要檢查
醫學多模態融合	CT 呈現衰減相關結構；MRI 提供不同組織對比；PET/SPECT 呈現示蹤劑分布	結合解剖與功能資訊的顯示影像，或特定任務的特徵	解剖對齊與模態特有證據的保留
全色銳化	高空間解析度的寬波段全色影像，加上較粗的多光譜波段	更細網格上的多光譜資料	空間細節**與**光譜保真
高光譜—多光譜融合	粗解析度的多個窄波段，加上細解析度的少數寬波段	更高空間解析度的高光譜立方體	同時符合兩個感測器的空間與光譜量測

U2Fusion 的 PET–MRI 評估 [32] 與 DDFM 的 CT/MRI、功能影像實驗 [41] 都涉及醫學情境。不能只把它視為「複製較亮的像素」，因為各模態的亮度意義不同。顯示融合也不會完整保留兩來源的定量值；量測時須保留原始資料，診斷效益則須透過合適的任務研究檢驗。

遙測融合常估計多波段影像，而非一張悅目的灰階圖 [54]。簡化的觀測模型可說明差異：

$$Y_H = \mathcal{D}(\mathcal{B}(X)) + N_H, \qquad Y_M = \mathcal{R}(X) + N_M.$$

$X \in \mathbb{R}^{H\times W\times L}$ 是未知的高解析度立方體，含 L 個窄光譜波段。$\mathcal{B}$ 對各波段施加空間模糊，$\mathcal{D}$ 降採樣，Y_H 為粗解析度高光譜觀測。$\mathcal{R}$ 依光譜響應把窄波段混為較少的多光譜波段，產生 Y_M；N_H, N_M 是量測誤差。真實模型還須處理校準、配準與可能隨波段改變的模糊。

全色銳化的高解析度觀測只有一個寬波段。注入細節時須考慮其光譜響應，不能直接假設它等於多光譜通道的平均。全解析度下通常沒有目標立方體真值；可透過降低現有資料解析度建立參考實驗，但結論取決於合成退化是否符合真實感測器。

這些任務與攝影融合共享特徵擷取、組合的概念，卻需要不同損失與評估。紅外線—可見光測試並不等於醫學或光譜重建驗證。「通用融合」須說清楚：通用的是架構、分別訓練的模型，還是跨任務共用的一個檢查點。

影像對位：一切方法的前提

白話版。融合是逐像素比較來源影像的，所以同一個像素必須在每個來源中都對應同一個點。如果有一張影像偏移了幾個像素，每條邊緣都會出現兩次。

配準估計來源間的幾何轉換，將它們扭曲到共同網格；動態場景則涉及稠密運動估計，見 DLCV 09A.3。腳架可減少誤差，但焦點變化、鏡頭畸變與移動物體仍可能需要配準。紅外線—可見光的視角、像素分布與解析度不同，精準配對不易取得，而多數融合方法仍需要逐像素對齊的來源 [3]；現有基準也多採對齊影像 [2]。深度學習段落末尾會再談放寬此假設的方法。

最簡單的規則：平均，以及它為什麼讓對比變淡

白話版。把來源影像平均，每樣東西都保留了一點，但只出現在其中一個來源的東西，會以一半的強度出現。融合的全部難處，就是避免這件事。

精確版。取兩張共用背景 b 的來源影像 I_A 與 I_B，並假設某個振幅為 a 的特徵（邊緣、紋理、溫暖的目標）只出現在 I_A 中：$I_A = b + a\,s$，$I_B = b$，其中 s 是特徵的形狀。平均是

$$F_{\text{avg}} = \tfrac{1}{2}(I_A + I_B) = b + \tfrac{a}{2}\,s,$$

所以特徵的對比減半。若有 K 個來源，而特徵只出現在其中一個，它會降到 a/K。在多焦點融合中情況更糟：模糊的來源會多加上一份模糊的邊緣副本，所以平均後看到的是一條銳利的邊緣外圍帶著柔軟的暈。在紅外線與可見光融合中，在紅外線中很亮、在可見光中很暗的目標，最後會變成灰色（圖 6，最後一欄）。在多曝光融合中，把曝光不足與曝光過度的兩張平均，得到的是一張平淡、低對比的影像。

解法是讓權重 $w_k(i,j)$ 取決於每個來源在哪裡有資訊：清晰的來源、溫暖的目標、曝光良好的像素，都給高權重。傳統方法這幾節要談的，就是怎麼衡量「有資訊」，以及怎麼改變權重而不產生接縫。

多焦點融合：對焦測度與決策圖

白話版。清晰的區域有強烈的細微細節，模糊的區域沒有。在每張照片的每個像素周圍衡量細節量，挑細節最多的那張，複製它的像素。

精確版。**對焦測度**（focus measure）$M_k(i,j) \ge 0$ 為來源 k 在像素 (i,j) 周圍有多少高頻細節打分數。一個簡單的選擇，屬於傳統多焦點融合常用的一類 [5]，是拉普拉斯的局部能量，

$$M_k = G_\sigma * (\nabla^2 I_k)^2,$$

其中 $\nabla^2 I_k = \partial^2 I_k/\partial x^2 + \partial^2 I_k/\partial y^2$ 是來源的拉普拉斯（一種二階導數濾波器，見 IP 03），平方使它成為能量，$G_\sigma *$ 表示用標準差為 σ 的高斯平滑，把能量在鄰域內取平均。模糊會去掉高頻，所以對焦來源的 M_k 較大。**決策圖**（decision map）記錄勝出者，

$$D(i,j) = \arg\max_k M_k(i,j), \qquad F(i,j) = I_{D(i,j)}(i,j),$$

其中 $D(i,j) \in \{1,\dots,K\}$ 是像素 (i,j) 所選來源的索引。

原始的決策圖是有雜訊的：在平坦區域（天空、素色牆面），沒有任何來源有細節，勝負由雜訊決定。有兩個標準的後處理步驟。**清理**（cleaning）移除小島，例如用多數決（中值濾波器）或形態學運算，假設對焦區域大而且相連。** 柔化 **（softening）在二值圖的邊界附近稍微模糊一下，讓從一個來源切換到另一個來源時不留下明顯的接縫。

下面的程式碼用 camera 測試影像建立一組合成影像對：圓盤內 A 張是清晰的，圓盤外 B 張是清晰的，散焦以高斯模糊建模。因為模糊是我們自己做的，所以我們有全對焦的真值，可以量 PSNR；真實的多焦點影像對是辦不到的。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, img_as_float
from skimage.metrics import peak_signal_noise_ratio as psnr

rng = np.random.default_rng(0)
x = img_as_float(data.camera())                       # 512 x 512: the all-in-focus "truth"
yy, xx = np.mgrid[:512, :512]
near = (yy - 200) ** 2 + (xx - 250) ** 2 < 150 ** 2   # the part that is in focus in shot A
blurred = ndi.gaussian_filter(x, 3.0)                 # defocus modeled as a Gaussian blur
```

```
A = np.where(near, x, blurred) + 0.01 * rng.standard_normal(x.shape)  # sharp inside the disk
B = np.where(near, blurred, x) + 0.01 * rng.standard_normal(x.shape)  # sharp outside it

def focus_measure(img, s=2.0):
    """Local energy of the Laplacian: large where the image has sharp detail."""
    return ndi.gaussian_filter(ndi.laplace(img) ** 2, s)

raw = focus_measure(A) > focus_measure(B)                # decision map: True = take A
clean = ndi.median_filter(raw.astype(float), 31) > 0.5   # majority vote removes small
 ↪  islands
w = ndi.gaussian_filter(clean.astype(float), 2.0)        # soften the seam a little

results = {"average": (A + B) / 2,
           "raw decision map": np.where(raw, A, B),
           "cleaned, soft map": w * A + (1 - w) * B}
for name, F in results.items():
    print(f"{name:18s} PSNR = {psnr(x, np.clip(F, 0, 1), data_range=1):.2f} dB")
textured = ndi.gaussian_filter(np.abs(ndi.laplace(x)), 2) > 0.01   # where the choice matters
for name, m in [("raw", raw), ("cleaned", clean)]:
    print(f"{name:7s} map agrees with the true focus map on {np.mean(m == near):.1%} of all
     ↪  pixels, "
          f"{np.mean((m == near)[textured]):.1%} of textured pixels")
```

```
average            PSNR = 29.97 dB
raw decision map   PSNR = 39.22 dB
cleaned, soft map  PSNR = 39.81 dB
raw     map agrees with the true focus map on 80.9% of all pixels, 91.3% of textured pixels
cleaned map agrees with the true focus map on 91.2% of all pixels, 98.4% of textured pixels
```

決策圖比平均高出約 10 dB，因為它複製的是清晰的像素，而不是把它們和模糊的像素混在一起（圖 2）。清理讓決策圖忠實許多（在有紋理的像素上，一致率從 91.3% 提高到 98.4%），但 PSNR 只增加一點點，0.6 dB，因為原始決策圖的大部分錯誤出現在平坦區域，在那裡 A 與 B 本來就長得一樣。這些數字來自我們的玩具範例，不是來自任何論文。

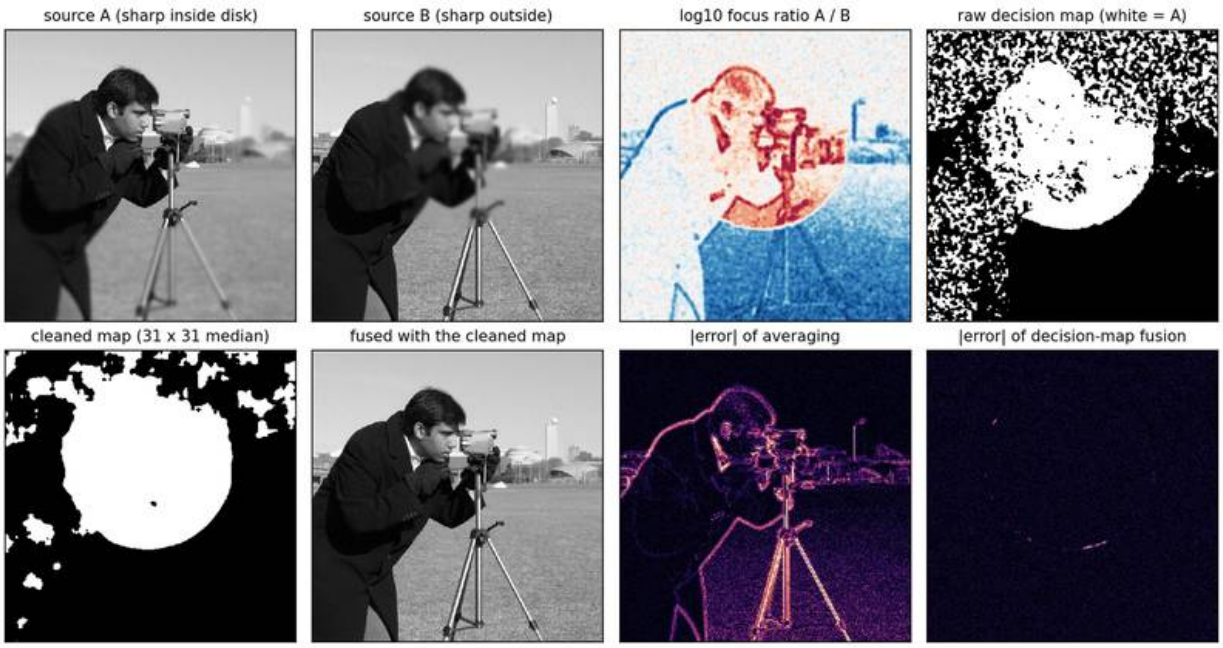


圖 2 — 用對焦測度做多焦點融合。平坦天空中的對焦比值是由雜訊決定的，中值濾波器大致清掉了它；那裡兩個來源一樣好，所以殘留的小塊無傷大雅。誤差圖（色階相同）顯示了為什麼選擇比平均好：平均在每條邊緣都留下誤差。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

玩具範例隱藏了什麼。真實的散焦不是把高斯模糊貼在遮罩上。在深度邊緣附近，對焦在遠處的那張照片裡，模糊的前景會擴散到背景上，所以那裡兩張都不清晰。這種**散焦擴散效應**（defocus spread effect）正是為什麼在模擬資料或常用的 Lytro 影像對上表現很好的方法，常在散焦強烈的真實照片上失敗 [6]、[7]；圖 6 在我們玩具範例的分層版本上展示了這一點。

僅線上版：散焦擴散效應 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/notes/散焦擴散效應/*

多尺度融合：拉普拉斯金字塔與小波

白話版。在多個尺度融合：選取有用細節，並平滑混合粗尺度亮度，以減少可見過渡。

拉普拉斯金字塔

精確版。 Burt 與 Adelson 的**拉普拉斯金字塔**（Laplacian pyramid）[8] 把影像表示成一疊帶通影像，加上一張小的低通殘差。令 $G_0 = I$，並且

$$G_{\ell+1} = \mathrm{REDUCE}(G_\ell), \qquad L_\ell = G_\ell - \mathrm{EXPAND}(G_{\ell+1}), \quad \ell = 0, \dots, N-1,$$

其中 REDUCE 用一個小的低通核做模糊，再每隔一列、每隔一行保留一個（OpenCV 的 `pyrDown`），EXPAND 以 2 倍上取樣，並用同一個核內插（`pyrUp`），G_ℓ 是**高斯金字塔**（Gaussian pyramid）第 ℓ 層（影像模糊後縮小的副本），L_ℓ 是**拉普拉斯金字塔**第 ℓ 層，也就是 G_ℓ 有、而更粗的一層沒有的細節。金字塔的頂端是殘差 $L_N = G_N$。第 ℓ 層的寬與高是影像的 $2^{-\ell}$ 倍。影像可以透過**還原**（collapse）精確地恢復：

$$G_N = L_N, \qquad G_\ell = L_\ell + \mathrm{EXPAND}(G_{\ell+1}), \quad \ell = N-1, \dots, 0,$$

而 $I = G_0$。對任何 EXPAND 都能精確還原，因為每個 L_ℓ 儲存的，正是 EXPAND 預測不到的部分。這個名稱來自作者的觀察：這種編碼等同於用許多尺度的拉普拉斯運算子對影像取樣 [8]。

僅線上版：建構與還原拉普拉斯金字塔 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/notes/建構與還原拉普拉斯金字塔/*

融合兩個金字塔

金字塔融合把每個來源分解，逐層合併金字塔，再還原結果。經典的金字塔融合選取來源中最顯著的拉普拉斯係數 [9]、[5]；一個簡單的版本，規則分成兩部分：

$$L_\ell^F(i,j) = \begin{cases} L_\ell^A(i,j) & \text{if } |L_\ell^A(i,j)| \geq |L_\ell^B(i,j)|, \\ L_\ell^B(i,j) & \text{otherwise,} \end{cases} \qquad L_N^F = \tfrac{1}{2}(L_N^A + L_N^B),$$

其中 L_ℓ^A、L_ℓ^B 與 L_ℓ^F 分別是來源 A、來源 B 與融合影像的金字塔第 ℓ 層。**取絕對值最大**（max-abs）規則在每個位置與每個尺度，保留絕對值較大的係數，也就是較強的局部特徵；粗略的殘差取平均，因為它承載整體亮度，沒有哪個來源應該直接勝出。在每個尺度逐係數選擇，表示來源之間的切換會分散到金字塔濾波器的支撐範圍內，越粗的層範圍越寬；這正是 Burt 與 Adelson 為了把影像鑲嵌拼接而不留明顯接縫所提出的多解析度混合想法 [10]。Burt 與 Kolczynski 後來改良了金字塔融合，讓它對小幅位移與雜訊更穩健，也能處理顯著度相近但對比相反的特徵 [9]。

這段程式碼接續多焦點的範例（它用到 `A`、`B`、`x` 與 `psnr`）。

```
import cv2
import pywt

def laplacian_pyramid(img, levels=5):
    G = [img.astype(np.float32)]
    for _ in range(levels):
        G.append(cv2.pyrDown(G[-1]))                      # blur, then keep every 2nd pixel
    L = [G[i] - cv2.pyrUp(G[i + 1], dstsize=G[i].shape[::-1]) for i in range(levels)]
    return L + [G[-1]]                                    # band-pass details + coarse base

def collapse(L):
    img = L[-1]
    for detail in reversed(L[:-1]):
        img = cv2.pyrUp(img, dstsize=detail.shape[::-1]) + detail
    return img

def pick_stronger(p, q):
    return np.where(np.abs(p) >= np.abs(q), p, q)         # "max-abs" selection rule
```

```
def pyramid_fuse(a, b, levels=5):
    La, Lb = laplacian_pyramid(a, levels), laplacian_pyramid(b, levels)
    fused = [pick_stronger(p, q) for p, q in zip(La[:-1], Lb[:-1])]
    fused.append((La[-1] + Lb[-1]) / 2)                  # base: average the coarse level
    return collapse(fused).astype(float)

def wavelet_fuse(a, b, wavelet="db2", levels=4):
    ca, cb = pywt.wavedec2(a, wavelet, level=levels), pywt.wavedec2(b, wavelet, level=levels)
    fused = [(ca[0] + cb[0]) / 2]                        # approximation band: average
    for da, db in zip(ca[1:], cb[1:]):                   # (H, V, D) detail bands per level
        fused.append(tuple(pick_stronger(p, q) for p, q in zip(da, db)))
    return pywt.waverec2(fused, wavelet)

print(f"pyramid reconstruction error: {np.abs(collapse(laplacian_pyramid(A)) -
 ↪  A).max():.1e}")
for name, F in [("Laplacian pyramid", pyramid_fuse(A, B)), ("wavelet (db2)", wavelet_fuse(A,
 ↪  B))]:
    print(f"{name:18s} PSNR = {psnr(x, np.clip(F, 0, 1), data_range=1):.2f} dB")
```

```
pyramid reconstruction error: 5.9e-08
Laplacian pyramid  PSNR = 37.60 dB
wavelet (db2)      PSNR = 36.43 dB
```

重建誤差約為 10^{-7}，是單精度的捨入誤差：金字塔本身是無損的。金字塔融合比平均高出 7 dB 以上，但仍比決策圖低約 2 dB。原因從圖 3 可以看出：逐係數的選擇有雜訊，尤其在細層，模糊來源的雜訊有時會在平坦區域勝出。改成比較平滑後的係數能量，而不是單一係數，也就是 Li 等人 [11] 的「區域式」（area-based）選擇，可以補上大部分差距（練習 3）。一般性的教訓是：當真正的答案是一種選擇（多焦點）時，空間上連貫的決策勝過逐係數的決策。

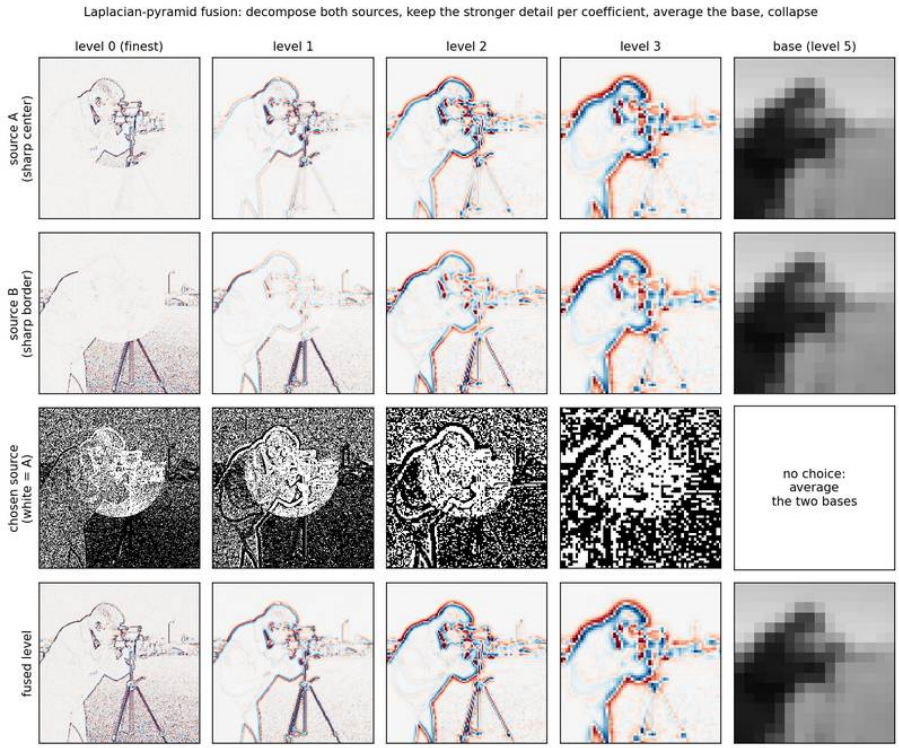


圖 3 — 多焦點影像對的拉普拉斯金字塔融合。每個來源被分成帶通層（紅與藍分別代表正與負的係數）；取絕對值最大規則逐位置挑選較強的係數（第 3 列，白色＝來源 A），基底取平均，還原融合後的金字塔就得到結果。

小波融合

小波融合遵循相同的計畫，只是用 IP 07 的離散小波轉換取代金字塔：分解兩個來源，在水平、垂直與對角的每個頻帶保留較強的細節係數，近似頻帶取平均，然後反轉換。Li、Manjunath 與 Mitra [11] 提出這個做法，使用**區域式**取最大規則（在小視窗內比較活動量，而不是單一係數），以及一**致性驗證**（consistency verification）步驟，讓係數跟隨其鄰居的選擇；他們主張小波轉換的緊緻性、正交性與方向選擇性，使它比拉普拉斯金字塔更適合。在我們的玩具範例中，簡單的逐係數規則得到 36.43 dB，比金字塔低約 1 dB。一個已知的弱點是，抽樣式小波轉換不具位移不變性：把影像移動一個像素，係數就變了，所以逐係數的決策也可能跟著改變。更好的位移不變性，也是 Burt 與 Kolczynski 改良金字塔融合的目標之一 [9]。

曝光融合：不經 HDR 也能得到曝光良好的影像

白話版。曝光融合依對比、飽和度與良好曝光程度加權，再透過影像金字塔混合以減少接縫。

精確版。 Mertens、Kautz 與 Van Reeth [5] 對第 k 張曝光在像素 (i, j) 計算三種**品質測度**：

- **對比** $C_{ij,k}$：對灰階曝光套用拉普拉斯濾波器的絕對響應（在邊緣與紋理處較高）；

- **飽和度** $S_{ij,k}$：像素 R、G、B 值的標準差（鮮豔的顏色散布大，褪色的顏色散布小）；
- **曝光良好度** $E_{ij,k} = \prod_{c\in\{R,G,B\}} \exp\!\left(-\frac{(I^c_{ij,k}-0.5)^2}{2\sigma^2}\right)$：圍繞中灰的高斯鐘形曲線，分別套用在每個色彩通道 $I^c_{ij,k} \in [0,1]$ 上再相乘，他們的實作中 $\sigma = 0.2$。

它們以帶指數的乘積組合，並對 N 張曝光做正規化：

$$W_{ij,k} = (C_{ij,k})^{\omega_C}\,(S_{ij,k})^{\omega_S}\,(E_{ij,k})^{\omega_E}, \qquad \hat{W}_{ij,k} = \frac{W_{ij,k}}{\sum_{k'=1}^{N} W_{ij,k'}},$$

其中 $\omega_C, \omega_S, \omega_E \geq 0$ 決定各測度的影響力（指數為 0 就是關掉該測度），$\hat{W}_{ij,k}$ 是正規化後的權重，在每個像素上對各張曝光的總和為 1。乘積的作用像「且」：像素只有在有細節、而且色彩豐富、而且曝光良好時，才會得到高權重 [5]。

為什麼要用金字塔。直接混合像素，$R_{ij} = \sum_k \hat{W}_{ij,k} I_{ij,k}$，會在權重快速變化的地方產生明顯的接縫，因為各張曝光的絕對亮度差異很大 [5]。用高斯平滑權重可以消除接縫，卻會在邊緣周圍產生光暈。曝光融合改成**逐金字塔層**混合：每張曝光的每個拉普拉斯層，乘上其權重圖的高斯金字塔中對應的那一層，

$$L\{R\}_\ell = \sum_{k=1}^{N} G\{\hat{W}_k\}_\ell \, L\{I_k\}_\ell,$$

$L\{\cdot\}_\ell$ 為拉普拉斯金字塔第 ℓ 層，$G\{\cdot\}_\ell$ 為高斯金字塔第 ℓ 層；收合 $L\{R\}$ 得到結果 R [5]。細節使用較尖銳的權重，粗尺度亮度則使用平滑權重；在配準與金字塔設定適當時，可減少接縫與光暈。

下面的程式碼建立一個具有 8 檔光線衰減（左暗右亮）的合成場景，用三種曝光時間、透過簡單的相機模型（gamma、截斷、8 位元量化）拍攝，並融合這組包圍曝光。它也拿結果和 OpenCV 的實作 `cv2.createMergeMertens` 對照。

```
import numpy as np
import cv2
from skimage import data, img_as_float

# A synthetic high-dynamic-range scene: linear radiance times a light falloff of 8 stops.
rgb = img_as_float(data.astronaut()) ** 2.2                       # undo display gamma ->
 ↪  linear
light = 2.0 ** np.linspace(-6, 2, rgb.shape[1])[None, :, None]    # dark left, bright right
radiance = rgb * light

def camera(E, t):
    """Expose for time t, apply a 1/2.2 display gamma, clip and quantize to 8 bits."""
    return np.round(np.clip((t * E) ** (1 / 2.2), 0, 1) * 255) / 255

stack = [camera(radiance, t) for t in (1 / 8, 1.0, 8.0)]          # under, middle, over

def mertens_weights(imgs, sigma=0.2, eps=1e-12):
    W = []
    for I in imgs:
        gray = I.mean(axis=2).astype(np.float32)
        C = np.abs(cv2.Laplacian(gray, cv2.CV_32F))                # contrast
        S = I.std(axis=2)                                          # saturation
```

```
        E = np.prod(np.exp(-(I - 0.5) ** 2 / (2 * sigma ** 2)), axis=2)  # well-exposedness
        W.append(C * S * E + eps)
    W = np.array(W)
    return W / W.sum(axis=0)                                  # weights sum to 1 per
     ↪  pixel

def gauss_pyr(img, n):
    G = [img.astype(np.float32)]
    for _ in range(n):
        G.append(cv2.pyrDown(G[-1]))
    return G

def lap_pyr(img, n):
    G = gauss_pyr(img, n)
    return [G[i] - cv2.pyrUp(G[i + 1], dstsize=G[i].shape[1::-1]) for i in range(n)] +
     ↪  [G[-1]]

def exposure_fusion(imgs):
    n = int(np.log2(min(imgs[0].shape[:2])))                  # full depth: 9 levels for
     ↪  512 px
    fused = None
    for I, Wk in zip(imgs, mertens_weights(imgs)):
        level = [g[..., None] * l for g, l in zip(gauss_pyr(Wk, n), lap_pyr(I, n))]
        fused = level if fused is None else [f + l for f, l in zip(fused, level)]
    out = fused[-1]
    for detail in reversed(fused[:-1]):
        out = cv2.pyrUp(out, dstsize=detail.shape[1::-1]) + detail
    return np.clip(out, 0, 1)

R = exposure_fusion(stack)
naive = np.clip(np.sum(mertens_weights(stack)[..., None] * np.array(stack), axis=0), 0, 1)
ref = cv2.createMergeMertens(1.0, 1.0, 1.0).process(
    [(255 * I[..., ::-1]).round().astype(np.uint8) for I in stack])[..., ::-1]   # OpenCV
     ↪  uses BGR
for name, I in [("under", stack[0]), ("middle", stack[1]), ("over", stack[2]),
                ("per-pixel blend", naive), ("pyramid blend", R)]:
    ok = np.mean(np.all((I > 0.05) & (I < 0.95), axis=2))        # no channel near black or
     ↪  clipped
    print(f"{name:15s} well-exposed pixels: {100 * ok:4.1f}%")
print(f"mean |ours - OpenCV MergeMertens| = {np.abs(R - np.clip(ref, 0, 1)).mean():.4f}")
```

```
under           well-exposed pixels: 45.8%
middle          well-exposed pixels: 58.7%
over            well-exposed pixels: 41.2%
per-pixel blend well-exposed pixels: 72.3%
pyramid blend   well-exposed pixels: 73.8%
mean |ours - OpenCV MergeMertens| = 0.0027
```

我們的實作與 OpenCV 的差異在 $[0,1]$ 範圍內不到 0.003；這個小差異大概來自實作細節，例如金字塔深度、邊界處理與輸入的捨入。粗略的「曝光良好像素」比例，從任何單張曝光至多 59%，在融合後升到 74%。注意這個計數漏掉了什麼：逐像素混合的得分幾乎一樣好（72%），看起來卻明顯壞掉，在權重跳動的地方有接縫與斑點（圖 4，左下）。單一數字看不到瑕疵，這個主題會在

評估那一節再度出現。

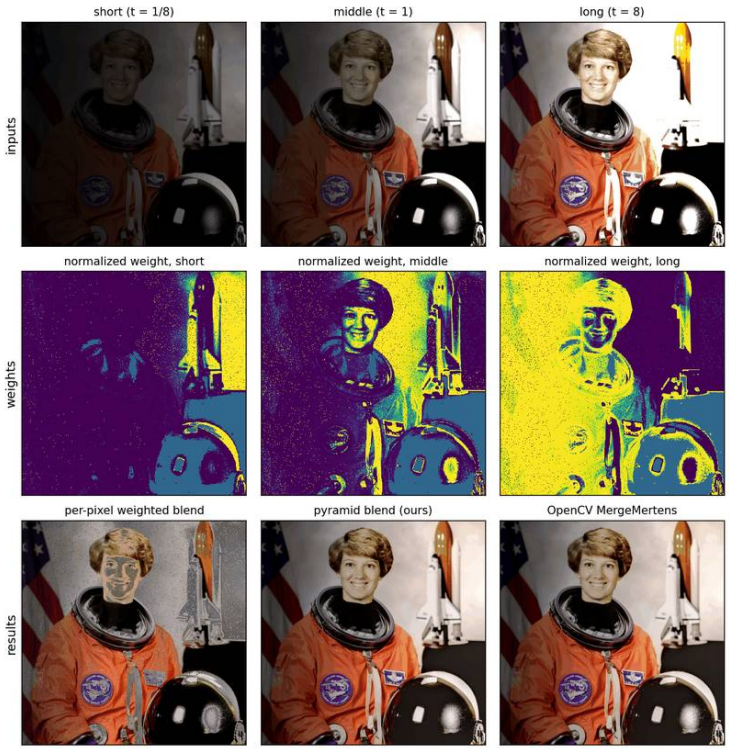


圖 *4* — 合成包圍曝光的曝光融合。權重挑出每張曝光中曝光良好且有細節的部分（在平坦的黑色區域，所有權重都很小，正規化後每張曝光各得三分之一）。直接用這些權重混合像素會失敗；混合金字塔層才行，而且與 *OpenCV* 的實作相符。

曝光融合不是 HDR。輸出不保留輻射量的數值，所以不能重新打光或重新曝光；而且在亮度變化劇烈時，多解析度混合可能產生虛假的低頻亮度變化，Mertens 等人指出，較高的金字塔能部分解決這個問題 [5]。曝光融合也假設各張曝光完全對齊，或許需先經過對位 [5]；有相機或物體移動時，結果會出現鬼影（圖 6）。最後，Mertens 等人把自己的混合與較早那種選擇最顯著係數的金字塔融合做比較，發現選擇的結果在曝光堆疊上太暗，還有色偏 [5]：對曝光而言，混合勝過選擇，與多焦點融合正好相反。

僅線上版：曝光融合（*Mertens*、*Kautz* 與 *Van Reeth*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/曝光融合 mertenskautz-與-van-reeth/*

兩種較晚的傳統工具：導引濾波與稀疏編碼

白話版。後來有兩個想法改進了傳統規則：用保邊濾波器讓權重圖跟著物體邊緣走；以及用學出來的字典描述影像區塊，讓「有多少細節」能在更豐富的表示上衡量。

導引濾波融合（guided-filtering fusion, GFF）。 Li、Kang 與 Hu [12] 把每個來源分成**基底層**（大尺度的強度變化）與**細節層**（小尺度的細節），各層用加權平均融合，而權重圖用導引濾波器 [13] 精修，讓它們符合空間一致性，也就是強度相近的鄰近像素得到相近的權重。導引濾波器在

IP 03 有說明，它的學習式與快速變體則在 DLCV 09C。作者報告，用同一個方法處理多光譜、多焦點、多模態與多曝光影像，都達到當時最先進的結果 [12]。這個設計展現了傳統時代的通用配方：二尺度或多尺度分解、活動量測度，以及一個把活動量變成乾淨權重的平滑步驟。

稀疏表示（sparse-representation, SR）融合。 Yang 與 Li [14] 用過完備字典上的稀疏係數表示來源，以取最大值規則組合係數，再由組合後的係數與字典重建融合影像；只要改變稀疏編碼的近似準則，同一套架構也能在融合的同時還原有雜訊的來源。後來的工作改成從來源影像的區塊本身學習字典 [15]。SR 方法曾是最強的傳統多焦點方法之一，但它們很慢：在一項基準中，融合一對影像花了七分鐘以上 [7]。

僅線上版：稀疏表示融合 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/notes/稀疏表示融合/*

沒有真值時如何評估融合

白話版。沒有正確的融合影像可以比對，融合只能用經驗法則打分數：結果細節多不多、對比高不高、從來源保留的資訊多不多？每條規則只抓到一個面向，所以它們互相矛盾，而且方法可以被調整到在選定的那幾項上表現漂亮。

這一節刻意放在深度學習之前：深度融合網路的訓練損失，和這些指標由同樣的成分構成，所以理解指標就能理解損失。

為什麼融合沒有唯一的正確答案

對合成資料上的多焦點融合，全對焦影像是已知的，拿它算 PSNR 或 SSIM [16] 是有意義的，就像我們的第一段程式碼那樣。對紅外線與可見光融合以及多曝光融合，不存在這樣的影像，而基準研究一致認為，沒有任何單一指標優於其他所有指標 [17]、[18]、[7]。所以基準會回報許多指標：VIFB 實作了 13 項 [17]，多曝光基準 MEFB 與多焦點基準 MFIFB 各有 20 項 [18]、[7]。VIFB 把它們分成四個家族：以資訊理論為基礎、以影像特徵為基礎、以結構相似度為基礎，以及以人類感知為基礎 [17]。它的結果顯示了後果：某個方法贏得資訊理論指標，另一個贏得特徵類指標，第三個贏得結構類指標，沒有任何方法全贏 [17]。

僅線上版：*VIFB*：可見光與紅外線影像融合基準 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/vifb 可見光與紅外線影像融合基準/*

只看融合影像的指標

令 F 為融合影像，尺寸為 $M \times N$，值在 $[0, 1]$ 內。

熵（entropy, EN）衡量直方圖散布得多開：

$$\mathrm{EN} = -\sum_{l=0}^{L-1} p_l \log_2 p_l,$$

其中 L 是灰階數（直方圖的區間數），p_l 是落在區間 l 的像素比例。EN 的單位是位元；值越大，表示灰階用得越平均。它很容易受雜訊影響，所以基準研究只把它當輔助指標 [18]。

標準差（SD）量測整體對比，$\mathrm{SD} = \sqrt{\frac{1}{MN}\sum_{i,j}(F(i,j)-\mu)^2}$，$\mu$ 是 F 的平均。有些實作省略 $\frac{1}{MN}$，會改變尺度；只有比較影像尺寸相同時，排名才不受這個省略影響。

空間頻率（spatial frequency, SF）衡量相鄰差值的整體活動量 [19]、[18]：

$$\mathrm{SF} = \sqrt{\mathrm{RF}^2 + \mathrm{CF}^2}, \qquad \mathrm{RF}^2 = \frac{1}{MN}\sum_{i,j}\big(F(i,j)-F(i,j-1)\big)^2, \quad \mathrm{CF}^2 = \frac{1}{MN}\sum_{i,j}\big(F(i,j)-F(i-1,j)\big)^2,$$

其中 RF（列頻率）使用水平差值，CF（行頻率）使用垂直差值；不同實作的正規化方式不同。SF 大，表示有很多強邊緣與很多紋理。

這三者從不看來源影像。它們獎勵「更多」：更多灰階、更多對比、更多邊緣。雜訊可以免費供應這三樣東西。

與來源影像比較的指標

互資訊（mutual information, MI）衡量融合影像透露了多少關於每個來源的資訊 [20]：

$$\mathrm{MI} = \mathrm{MI}_{A,F} + \mathrm{MI}_{B,F}, \qquad \mathrm{MI}_{X,F} = \sum_{x,f} p_{X,F}(x,f)\log_2\frac{p_{X,F}(x,f)}{p_X(x)\,p_F(f)},$$

其中 $p_{X,F}$ 是來源 $X \in \{A,B\}$ 與 F 的聯合直方圖（正規化到總和為 1），p_X、p_F 是它的邊際分布。當 F 的灰階能夠準確預測每個來源的灰階時，MI 就大（定義見 IT 01）。Hossny、Nahavandi 與 Creighton 從解析上證明，這個經典的 MI 總和本身並不是有效的融合效能指標 [21]，而基準工具包實作的是歸功於他們的正規化版本，把每一項除以兩個熵的總和 [18]。

僅線上版：互資訊作為融合指標 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/notes/互資訊作為融合指標/*

以梯度為基礎的融合效能 $Q^{AB/F}$（Xydeas 與 Petrović [22]）衡量來源的**邊緣資訊**有多少保留在 F 中。在每個像素上，它把每個來源 X 的邊緣強度 g 與方向 α（由影像梯度得到）和 F 的做比較，把比較結果轉成保留值 $Q^{XF}(i,j) = Q_g^{XF}(i,j)\,Q_\alpha^{XF}(i,j)$，也就是邊緣強度保留值與方向保留值的乘積，再以代表各來源重要性的權重 $w^X(i,j)$ 加權平均 [18]：

$$Q^{AB/F} = \frac{\sum_{i,j}\big(Q^{AF}(i,j)\,w^A(i,j) + Q^{BF}(i,j)\,w^B(i,j)\big)}{\sum_{i,j}\big(w^A(i,j)+w^B(i,j)\big)}.$$

值接近 1，表示兩個來源的邊緣都保留了強度與方向。下面的程式碼實作一個簡化版本，我們稱之為 Q_G：強度保留是較小與較大梯度大小的比值，方向保留隨邊緣夾角線性下降，權重是來源的梯度大小。回報 $Q^{AB/F}$ 時，請使用參考實作。

以結構與感知為基礎的指標。 Piella 與 Heijmans [23] 根據 Wang 與 Bovik 的影像品質指數，提出三種融合品質指標的變體；它們不需要真值影像，因為它們把 F 和來源比較，作者發現它們與主觀評估一致。VIFF [24] 把視覺資訊保真度準則改造用於融合，並在多個解析度上計算。對多曝光融合，**MEF-SSIM** [25] 是依據一項主觀研究設計的：作者發現一般的融合指標對多曝光融合的感知品質預測得很差，於是圍繞區塊結構一致性測度建了一個 SSIM 風格的指數，與人類判斷的相關性好得多。Liu 等人 [26] 針對夜視應用，在六種多解析度融合演算法上比較了十二種融合指標。

VIF 是資訊傳遞模型，不是原始熵。它估計來源的模型化資訊經過候選影像與視覺通道後，還保留多少 [55]。融合版本的來源彙整方式不同，分數大於一或跨論文尺度不同時，須檢查實作；來源本身仍不是正確融合真值。

僅線上版：*VIF 與來源參考的融合分數 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/notes/vif-與來源參考的融合分數/*

一個會騙過指標的玩具

下面的程式碼用 camera 影像建立一組紅外線與可見光影像對：可見光影像右側有一塊很深的陰影，其中的人看不見；熱像平滑、幾乎沒有紋理，而人在其中發亮。它為四個候選打分數：完全不融合（可見光影像本身）、平均、金字塔融合（細節取絕對值最大，粗略基底取兩者中較亮的，使溫暖的目標保持它的亮度），以及加了雜訊的平均。作為對使用者真正在乎的事的獨立檢查，它還量測**人物對比**：*F* 在人身上的平均值，減去人周圍一圈環帶上的平均值。

```
import numpy as np
import cv2
from scipy import ndimage as ndi
from skimage import data, img_as_float

rng = np.random.default_rng(1)
scene = img_as_float(data.camera())
yy, xx = np.mgrid[:512, :512]
person = ((yy - 330) / 60) ** 2 + ((xx - 420) / 22) ** 2 < 1        # a warm "person", 120 px
 ↪  tall
shadow = ndi.gaussian_filter((xx > 340).astype(float), 8)          # a dark area on the right
vis = scene * (1 - 0.85 * shadow)                                  # visible: person hidden
 ↪  in shadow
ir = 0.25 + 0.3 * ndi.gaussian_filter(scene, 6)                    # thermal: smooth, little
 ↪  texture
ir = np.clip(ir + 0.6 * ndi.gaussian_filter(person.astype(float), 2)  # the warm body glows
             + 0.01 * rng.standard_normal(ir.shape), 0, 1)

def lap_pyr(img, n=5):
    G = [img.astype(np.float32)]
    for _ in range(n):
        G.append(cv2.pyrDown(G[-1]))
    return [G[i] - cv2.pyrUp(G[i + 1], dstsize=G[i].shape[::-1]) for i in range(n)] + [G[-1]]

def collapse(L):
    img = L[-1]
    for d in reversed(L[:-1]):
        img = cv2.pyrUp(img, dstsize=d.shape[::-1]) + d
    return img

def pyramid_fuse(a, b):                    # max-abs details, and the brighter coarse base
    La, Lb = lap_pyr(a), lap_pyr(b)
    out = [np.where(np.abs(p) >= np.abs(q), p, q) for p, q in zip(La[:-1], Lb[:-1])]
    out.append(np.maximum(La[-1], Lb[-1]))
    return np.clip(collapse(out), 0, 1).astype(float)

def entropy(F, bins=256):
```

```
    p = np.histogram(F, bins, (0, 1))[0] / F.size
    p = p[p > 0]
    return -np.sum(p * np.log2(p))

def spatial_frequency(F):
    rf = np.mean(np.diff(F, axis=1) ** 2)           # row frequency: horizontal differences
    cf = np.mean(np.diff(F, axis=0) ** 2)           # column frequency: vertical differences
    return np.sqrt(rf + cf)

def mutual_info(X, F, bins=64):
    pxy = np.histogram2d(X.ravel(), F.ravel(), bins, [[0, 1], [0, 1]])[0] / X.size
    px, pf = pxy.sum(1, keepdims=True), pxy.sum(0, keepdims=True)
    nz = pxy > 0
    return np.sum(pxy[nz] * np.log2(pxy[nz] / (px @ pf)[nz]))

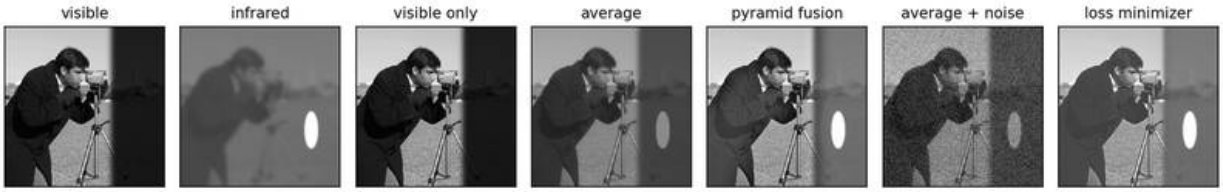
def q_gradient(A, B, F):
    """A simplified Q^{AB/F}: how well F keeps the edge strength and direction of A and B."""
    def grad(I):
        gx, gy = ndi.sobel(I, 1), ndi.sobel(I, 0)
        return np.hypot(gx, gy), np.arctan2(gy, gx)
    gF, aF = grad(F)
    num = den = 0.0
    for X in (A, B):
        gX, aX = grad(X)
        strength = np.minimum(gX, gF) / (np.maximum(gX, gF) + 1e-12)
        d = np.abs(aX - aF) % np.pi                 # edge direction is defined modulo pi
        direction = 1 - np.minimum(d, np.pi - d) / (np.pi / 2)
        num += np.sum(strength * direction * gX)    # weight each pixel by the source edge
         ↪  strength
        den += np.sum(gX)
    return num / den

ring = ndi.binary_dilation(person, iterations=15) & ~person
candidates = {"visible only": vis,
              "average": (vis + ir) / 2,
              "pyramid fusion": pyramid_fuse(vis, ir),
              "average + noise": np.clip((vis + ir) / 2 + 0.08 *
               ↪  rng.standard_normal(vis.shape), 0, 1)}
print(f"{'':16s}  EN    SF     SD     MI    Q_G   person contrast")
for name, F in candidates.items():
    mi = mutual_info(vis, F) + mutual_info(ir, F)
    tc = F[person].mean() - F[ring].mean()          # task check: does the person stand out?
    print(f"{name:16s} {entropy(F):4.2f}  {spatial_frequency(F):.3f}  {F.std():.3f}
     ↪  {mi:4.2f}  "
          f"{q_gradient(vis, ir, F):.3f}  {tc:6.3f}")
```

```
                  EN    SF     SD     MI    Q_G   person contrast
visible only     6.68  0.065  0.293  5.72  0.784  -0.000
average          6.69  0.034  0.172  4.33  0.455   0.257
pyramid fusion   7.48  0.067  0.221  3.75  0.779   0.513
average + noise  7.49  0.163  0.189  1.41  0.296   0.259
```

逐欄讀這張表（所有數字都來自這個玩具範例）：

- **雜訊贏了 EN 與 SF**。在平均上加雜訊，得到最高的熵，以及幾乎是平均五倍的空間頻率，但影像其實變差了。
- **完全不融合贏了 SD、MI 與 Q_G**。單獨的可見光影像對比最高，互資訊最高（複製一個來源，會使 $\mathrm{MI}_{A,F}$ 等於該來源的熵），連邊緣分數也最高，因為強烈的可見光邊緣在權重 w^X 中占大宗。然而人卻看不見：它的對比是零（印出來是 -0.000）。
- **金字塔融合只贏了任務檢查**。相對於平均，它把人的對比加倍，同時保留可見光的紋理（Q_G 幾乎和可見光影像一樣高），但五項標準指標中沒有一項它是最好的。



Higher is 'better' for every column; the shaded cell is each column's winner

candidate	EN (bits)	SF	SD	MI (bits)	Q_G	person contrast
visible only	6.68	0.065	0.293	5.72	0.784	0.000
average	6.69	0.034	0.172	4.33	0.455	0.257
pyramid fusion	7.48	0.067	0.221	3.75	0.779	0.513
average + noise	7.49	0.163	0.189	1.41	0.296	0.259
loss minimizer	6.99	0.065	0.200	4.24	0.718	0.513

圖 *5* — 合成的紅外線與可見光影像對的五個候選融合結果及其分數（我們的玩具範例；損失最小化結果在深度學習那一節計算）。每項指標都是「越高越好」，但優勝者卻是一張有雜訊的影像，以及一張忽略紅外線來源的影像。只有任務檢查找出了有用的結果。

回報什麼

這個玩具範例刻意走極端，但每個效應在實務上都會出現：雜訊與過度銳化會提高 EN、SF 與 SD；看起來像某個來源的結果在 MI 上得分很高；而權重取決於來源的指標，會偏袒邊緣較強的來源。基準的作者從真實方法得到同樣的結論：研究者傾向挑選支持自己方法的指標 [17]、[7]，而且定量與定性的比較可能不一致 [7]。實務建議：

1. 回報來自**不同家族**的多個指標，其中至少包含一個與來源比較的指標（$Q^{AB/F}$、以 SSIM 為基礎的分數或 VIFF），不要只報 EN、SD 或 SF。
2. 在可能有參考影像的地方使用**有參考**的指標：合成的多焦點影像對，或有挑選過參考影像的多曝光資料集（但要注意，參考影像只是一種可接受的答案）。
3. 當融合會餵給下游任務時，加入**下游任務**，例如在融合影像上的偵測或分割準確率，近年的紅外線與可見光基準就是這麼做的 [27]、[28]。
4. 展示困難區域的**裁切圖**（深度邊緣、亮部、陰影中的溫暖目標），而且當主張與視覺品質有關時，做使用者研究。

深度影像融合

白話版。深度網路從資料學習權重、特徵與組合規則。麻煩的是，沒有任何正確輸出的範例，所以網路是被訓練去滿足和上面那些指標同一類的規則。

缺少的真值

復原網路（DLCV 09）從成對資料（退化輸入，乾淨目標）學習。融合網路有輸入，卻沒有目標。這個領域用四種方式繞過這個問題：

1. **合成一個有真值的任務**。對多焦點融合，把清晰影像的部分區域模糊化：清晰影像就是目標，模糊遮罩就是真正的決策圖。大多數深度多焦點方法都是這樣訓練的 [7]；IFCNN 用一個由 RGB-D 資料集建成的合成多焦點資料集來訓練通用融合網路，使真值融合影像得以存在 [29]。
2. **先訓練別的，再用規則融合特徵**。訓練一個自編碼器去重建普通影像，測試時把兩個來源都編碼，用手工規則合併特徵，再解碼 [30]。
3. **非監督式損失**。定義一個損失，衡量 F 保留了每個來源的強度、梯度或結構到什麼程度，並在沒有標註的來源影像對上訓練網路使它最小化 [31]、[32]。
4. **讓另一項任務來監督**。把 F 餵給偵測器或分割器，並加上它的損失，使融合以其實用性來評斷 [33]、[27]。

非監督式損失

一篇近期綜述把非監督式紅外線與可見光融合的損失分成像素層級的項（L1 或 MSE）、由評估指標構成的項（最主要是 SSIM），以及針對資料特性（例如影像梯度）的項 [2]。一種典型的組合是

$$\mathcal{L}(F) = \lambda_{\text{int}} \lVert F - T \rVert_1 + \lambda_{\text{grad}} \left\lVert \lvert \nabla F \rvert - \lvert \nabla T_g \rvert \right\rVert_1 + \lambda_{\text{ssim}} \sum_{X} \omega_X (1 - \text{SSIM}(F, I_X)),$$

其中 T 是由來源建立的強度目標（例如加權平均，或逐像素取最大值以保留較亮的熱目標），$\lvert \nabla T_g \rvert$ 是梯度大小目標（例如每個像素上兩個來源梯度大小中較大者），$\text{SSIM}(F, I_X)$ 是 F 與來源 X 之間的結構相似度 [16]，ω_X 是各來源的權重，λ 則是各項的權重。方法的差別主要就在這些選擇：DeepFuse [31] 只用 MEF-SSIM [25] 訓練，SwinFusion [34] 用 SSIM、紋理與強度項，U2Fusion [32] 則用對每個來源含有多少資訊的估計，來加權一個相似度損失。

僅線上版：代表性融合網路的訓練訊號 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/notes/代表性融合網路的訓練訊號/*

損失就是答案的定義

白話版。如果沒有人給網路目標，損失就是目標。網路能做到的最好，就是針對每一對輸入，輸出使損失最小的那張影像。對簡單的二次損失，我們可以不用任何網路，直接算出那張影像並看看它。

這段程式碼接續評估那一節的玩具範例（它用到 `vis`、`ir`、`person`、`ring` 與各個指標函式）。它最小化上述損失的平方誤差版本：一個朝向來源逐像素最大值的強度項，加上一個朝向較強來源梯度的梯度項，

$$F^\star = \arg\min_F \|F - \max(I_A, I_B)\|_2^2 + \lambda \|\nabla F - \mathbf{g}\|_2^2,$$

其中 ∇F 把 F 的水平與垂直前向差分疊在一起，$\mathbf{g}$ 在每個位置與每個方向上，取兩個來源中差分絕對值較大者的差分，$\lambda > 0$ 在強度與梯度之間取捨。把目標函數的梯度設為零，得到一個稀疏線性系統 $(\mathbf{I} + \lambda \mathbf{D}^\top \mathbf{D}) F = \max(I_A, I_B) + \lambda \mathbf{D}^\top \mathbf{g}$，其中 $\mathbf{D}$ 是差分運算子；共軛梯度法幾秒鐘就能解出來。

```
import scipy.sparse as sp
from scipy.sparse.linalg import cg

def diff_ops(H, W):
    d = lambda n: sp.diags([-np.ones(n - 1), np.ones(n - 1)], [0, 1], shape=(n - 1, n))
    return sp.kron(sp.eye(H), d(W)).tocsr(), sp.kron(d(H), sp.eye(W)).tocsr()  # Dx, Dy

def fuse_by_loss(a, b, lam=10.0):
    """Minimize ||F - max(a, b)||^2 + lam * ||grad F - g||^2, where g is the stronger source
     ↪  gradient."""
    H, W = a.shape
    Dx, Dy = diff_ops(H, W)
    gx = [Dx @ s.ravel() for s in (a, b)]
    gy = [Dy @ s.ravel() for s in (a, b)]
    pick_x = np.abs(gx[0]) >= np.abs(gx[1])
    pick_y = np.abs(gy[0]) >= np.abs(gy[1])
    tx, ty = np.where(pick_x, gx[0], gx[1]), np.where(pick_y, gy[0], gy[1])
    target = np.maximum(a, b).ravel()                # intensity term: the brighter source
    M = sp.eye(H * W) + lam * (Dx.T @ Dx + Dy.T @ Dy)  # normal equations of the quadratic
     ↪  loss
    rhs = target + lam * (Dx.T @ tx + Dy.T @ ty)
    f, info = cg(M, rhs, x0=target, rtol=1e-6, maxiter=2000)
    if info != 0: raise RuntimeError("Conjugate gradient did not converge")
    return np.clip(f.reshape(H, W), 0, 1)

for lam in (1.0, 10.0):
    F = fuse_by_loss(vis, ir, lam)
    mi = mutual_info(vis, F) + mutual_info(ir, F)
    print(f"loss minimizer, lam={lam:4.1f}: EN {entropy(F):.2f}  SF
     ↪  {spatial_frequency(F):.3f}  "
          f"SD {F.std():.3f}  MI {mi:.2f}  Q_G {q_gradient(vis, ir, F):.3f}  "
          f"person contrast {F[person].mean() - F[ring].mean():.3f}")
```

```
loss minimizer, lam= 1.0: EN 6.91  SF 0.060  SD 0.195  MI 4.68  Q_G 0.659  person contrast
 ↪  0.513
loss minimizer, lam=10.0: EN 6.99  SF 0.065  SD 0.200  MI 4.24  Q_G 0.718  person contrast
 ↪  0.513
```

損失最小化的結果裡，人和金字塔融合一樣亮（對比 0.513），同時保留可見光的紋理（圖 5，右側）。把 λ 從 1 提高到 10，會使 Q_G 與 SF 上升、MI 下降：損失各項之間的權重會讓結果在各指

標上移動，而選擇權重，就是選擇什麼才叫「正確」的融合。這是對非監督式深度融合誠實的總結：網路帶來了速度、從大量訓練影像對泛化的能力，以及比原始梯度更豐富的特徵，但好結果的定義仍是人寫的，就寫在損失裡。

深度融合網路的家族

綜述依架構與訓練策略整理這個領域 [35]、[2]。下表列出每個家族的代表成員；論文導讀中的紅外線與可見光融合綜述導讀 (https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/liu-等人的紅外線與可見光融合綜述 tpami-2025/) 對分類法有更深入的說明。

家族	想法	代表方法
學習式決策圖或權重	網路預測哪個來源對焦，或預測逐像素權重；輸出仍是來源像素的選擇或混合	多焦點的 CNN 對焦圖 [36]；MEF-Net 在低解析度預測權重，並用導引濾波器上取樣 [37]
自編碼器＋融合規則	訓練編碼器–解碼器重建影像；用手工規則融合編碼後的特徵	DenseFuse [30]
端到端、非監督式	網路把來源映射成 F，以強度／梯度／SSIM 類型的損失訓練	DeepFuse [31]、U2Fusion [32]、SwinFusion [34]、CDDFuse [38]
在合成資料上監督式訓練	在有已知融合影像的合成影像對上訓練	IFCNN [29]
對抗式（GAN）	判別器促使 F 看起來像某一個來源的紋理，內容損失則保留另一個來源的強度	FusionGAN [39]、TarDAL [27]
任務導向	對 F 的偵測或分割損失引導融合網路	SeAFusion [33]、TarDAL [27]、SegMiF [28]、TDFusion [40]
生成式先驗與語言	預訓練的擴散模型或文字編碼器引導融合	DDFM [41]、Text-IF [42]

有兩個家族依賴在其他篇章說明過的生成模型：GAN 見 ML 07，擴散模型見 ML 08。在所有家族中，架構對結果特性的影響，不如訓練訊號來得大，這正是表中間那一欄真正區分的東西。

各類學習式方法究竟學到什麼

架構表是地圖，不是訓練配方。沿著監督訊號的來源，才能理解結果。

自編碼器加規則。 DenseFuse 從一般影像學特徵與重建 [30]，推論時再用固定規則合併特徵。訓練不需融合真值，但重建本身不會教它哪個模態應優先。同一組網路權重，使用相加或活動度加權，也會產生不同偏好。

監督式卷積融合。 IFCNN 以有真值的合成配對訓練，再逐元素合併來源特徵 [29]。它有直接監督，但也承接合成資料的假設；若模糊模型忽略前景擴散或感測器差異，真實影像上的選擇就可能錯誤。

直接無監督融合。 U2Fusion 用自適應資訊權重，讓輸出接近來源 [32]；必須同時理解其結構與強度項。VGG 權重估計與持續學習懲罰用於訓練，最終產圖網路推論時不需要 VGG 或 EWC。

對抗式融合。 FusionGAN 在來源保留目標外加入判別器，以保留紅外線強度並鼓勵可見光般的紋理 [39]。逼真外觀只是分布偏好，不能證明某條邊緣有來源支持。TarDAL 則將對抗角色分為熱目標與可見光細節，並加入下游監督 [27]。

注意力融合。 SwinFusion 以視窗與移位視窗注意力，學習來源內與來源間的互動 [34]。查詢、鍵與值讓特徵引用不同位置或模態的證據；視窗限制成本，連續移位則跨越邊界。它仍需要融合目標與對齊證據，也可能傳遞雜訊。

擴散融合。 DDFM 反覆結合預訓練去雜訊先驗與來源修正 [41]，省去融合專用的去雜訊器訓練，但要付出迭代推論成本。每個擴散步內做一次 EM，不代表整個融合只需一步；概似模型與影像先驗都會影響結果。

統一與任務感知系統。「統一」可能指損失或架構，而非共用權重，須查檢查點及訓練設定。U2Fusion 明確追求跨任務共用參數 [32]。任務感知模型則透過偵測或分割來評判融合 [27]、[33]，需要對應標註，也可能專精於該任務。兩種名稱都不保證一個模型可處理任意模態或未配準輸入。

僅線上版：*DenseFuse*：學習重建，在特徵空間選擇融合規則 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/densefuse 學習重建在特徵空間選擇融合規則/*；*U2Fusion*：跨融合任務共用一個無監督模型 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/u2fusion 跨融合任務共用一個無監督模型/*；*TarDAL*：同時為外觀與物件偵測融合 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/tardal 同時為外觀與物件偵測融合/*；*DDFM*：以概似修正擴散先驗 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/ddfm 以概似修正擴散先驗/*

為下游任務而融合

白話版。如果融合影像要餵給偵測器，最好的融合就是讓偵測器好用的那一個，而不是熵最高的那一個。

SeAFusion [33] 把即時融合網路和分割網路串接起來，加上語意損失，並在迴圈中訓練兩者 [2]。** TarDAL** [27] 把融合與偵測寫成同一個雙層最佳化問題，將它展開成一個融合網路，包含一個生成器與兩個判別器（一個負責紅外線目標，一個負責可見光紋理），後面接著偵測器，並引入以同步的紅外線與可見光系統拍攝的 M3FD 基準。** SegMiF** [28] 透過階層式注意力，在融合與分割兩個子網路之間交換中間特徵，並收集了一個全天候的基準，含有 15 個像素層級標註的類別。** TDFusion** [40] 更進一步，學習融合損失本身：用元學習（meta-learning）訓練一個損失生成模組，使它所監督的融合網路最小化下游任務損失。任務導向的訓練以可量測的目標回答了「什麼才是正確」的問題，但它是針對一項任務與一個網路最佳化的，而且任務表現並不能完全反映視覺品質 [2]。

融合未對齊的影像

白話版。真實的紅外線與可見光相機很少完全對齊。對位容錯的方法會估計錯位，並在融合網路內部修正它。

對錯位的影像對逐像素融合，會在物體周圍產生鬼影與雙重邊緣 [43]、[44]。近期工作有兩種策略 [2]。**風格轉換路線**先把一種模態轉換成另一種，使跨模態對位變成單一模態對位：UMF-

CMGR [43] 由可見光影像生成偽紅外線影像，用多層級對位網路把真實的紅外線影像對位到它上面，然後融合。** 共享特徵路線 ** 把兩種模態映射到共同的特徵空間，並在其中預測形變場。SuperFusion [44] 把對位（雙向形變場）、融合，以及來自預訓練分割模型的語意約束，結合在一個框架中。多曝光融合也有類似的問題，來自相機晃動與移動的物體（鬼影）。

資料集與基準

白話版。紅外線與可見光融合有幾個公開的對齊影像對集合，有些附有物體標註；多焦點與多曝光融合則有較小的測試集與基準工具包。真實的紅外線與可見光資料都沒有真值融合影像。

資料集	任務	內容（依作者的描述）
TNO [45]	紅外線與可見光	軍事與監視場景的增強視覺、近紅外線與長波紅外線夜間影像
RoadScene [32]	紅外線與可見光	從 FLIR 影片中挑選的 221 對對齊道路場景影像，隨 U2Fusion 發布
MSRS [46]	紅外線與可見光	1,444 對對齊的道路場景影像，由 MFNet 資料集 [47] 去除 125 對錯位影像並增強紅外線影像而成
LLVIP [48]	紅外線與可見光	15,488 對嚴格對齊的影像，多數在非常暗的場景拍攝，附行人標註
M3FD [27]	紅外線與可見光	4,200 對已對位且附偵測標註的影像，另有 300 對來自獨立場景、供融合使用；影像多為 1024 × 768
FMB [28]	紅外線與可見光	全天候基準，含 15 個像素層級標註的類別，供融合與分割使用
VIFB [17]	紅外線與可見光	21 對的測試集、含 20 種融合演算法的程式庫，以及 13 項指標
Lytro [15]	多焦點	20 對多焦點影像，長期是最廣泛使用的測試集 [7]
MFFW [6]	多焦點	從網路收集的 19 對真實影像，散焦擴散效應很強
MFIFB [7]	多焦點	含 105 對測試影像、30 種演算法與 20 項指標的基準
SICE [49]	多曝光	589 個多曝光序列（4,413 張影像），每個序列的參考影像，是透過主觀比較 13 種融合與 HDR 演算法而選出的
MEFB [18]	多曝光	含 100 對測試影像、16 種演算法與 20 項指標的基準

讀這張表要留意三點。**前處理也是資料的一部分：**RoadScene 的紅外線影像處理過背景熱雜訊，MSRS 的紅外線影像用基於暗通道先驗的方法增強過對比，所以「紅外線來源」其實已經被處理過。** 場景很窄：**大多數紅外線與可見光資料是都市道路與校園，而 2025 年的綜述把更多樣的場景與對位真值，列為缺少的基準 [2]**。舊的測試集已經飽和：** 在 20 對 Lytro 影像上，常用方法表現非常相近，因為那裡的散焦擴散效應很弱 [6]、[7]。SICE 也在 DLCV 09C 中用於曝光校正。

失效模式

白話版。融合在它的假設不成立的地方失敗：來源沒有對齊、來源不是單純的「清晰或模糊」、權重變化太快或太慢，以及把該選擇的東西拿去平均的規則。

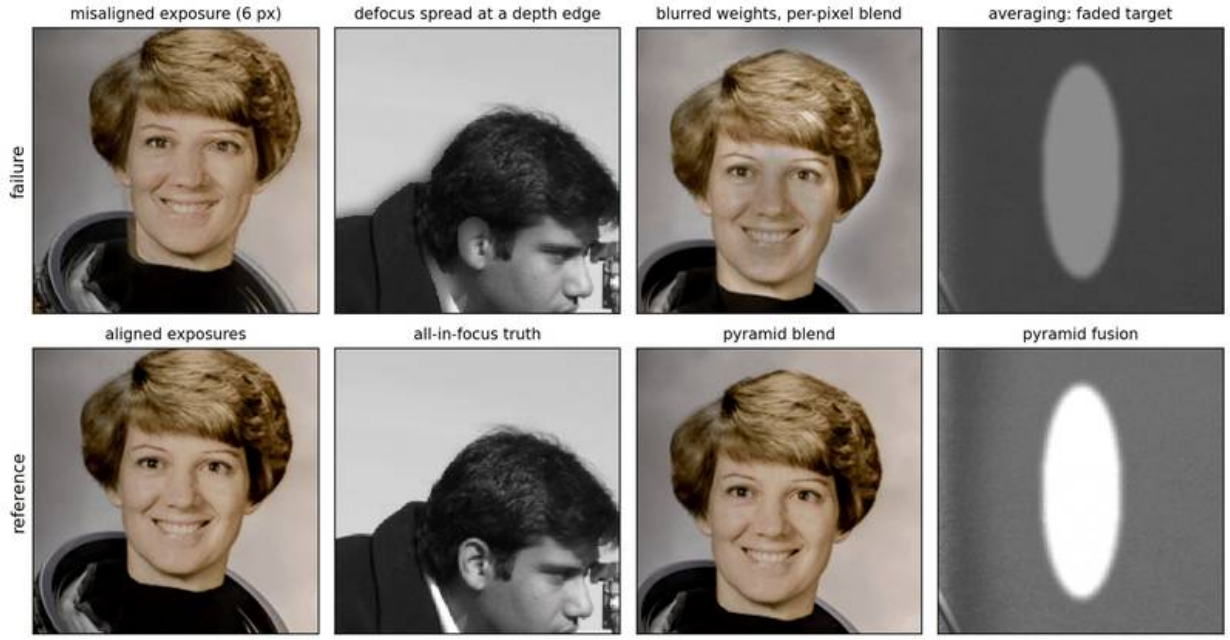


圖 *6* — 合成資料上的四種失效模式（上）與各自的參考（下）：對位錯誤造成的鬼影、深度邊緣處的散焦擴散效應（分層散焦模型，模糊的前景溢出到背景上）、平滑權重造成的光暈，以及褪色的熱目標。

- **配準鬼影**。來源差幾個像素，就會出現雙重邊緣（圖 6 第一欄）；多曝光的移動物體也一樣。先檢查對齊。
- **決策圖邊界偽影**。深度邊緣附近，前景散焦會擴散到背景，使兩來源都不清晰，二元選擇可能留下軟環或方塊邊緣（第二欄）。錯誤的硬切換也會產生接縫。
- **光暈**。權重過度平滑，或基底層在太細的尺度融合，會在強邊緣附近形成明暗帶（第三欄）[5]。多尺度可改善，但金字塔太淺仍會產生接縫與低頻誤差（練習 4）。
- **熱目標被沖淡**。平均或過度偏重可見光的損失，會削弱熱目標對比（第四欄）；反過來則會讓背景像缺乏紋理的熱影像。
- **色偏**。常見作法只融合亮度，例如 YCbCr 的 Y，再複製或混合色度，DeepFuse 的多曝光方法即如此 [31]。若融合亮度與色度來源差異很大，例如暗背景上的亮熱目標，顏色就可能褪色或偏色。Mertens 等人的比較也顯示，選擇而非混合金字塔係數會造成色偏 [5]。
- **放大雜訊**。最大絕對值選擇與梯度損失偏好高頻，暗處的雜訊可能主導高頻響應，使結果更銳利卻更吵。EN、SF 也會獎勵這種結果。
- **迎合指標**。專門最佳化 EN、SF、SD 的方法，可能產生以上偽影卻得到更好分數（圖 5）。

現代觀點

綜述。六篇綜述合起來涵蓋這個領域：

- **Ma、Ma 與 Li（Information Fusion 2019）[3]** 回顧傳統與早期深度紅外線與可見光融合，引入對位作為前提，整理應用與評估指標，並用九項指標比較十八種方法。重點：深度學習接手之前，傳統方法及其評估習慣的參考點。
- **Liu 等人（Information Fusion 2018）[1]** 寫了一篇早期（2018）的回顧，談跨任務的像素層級融合的深度學習。他們列出傳統融合的困難（手工設計的活動量測度與融合規則），主張學習到的特徵可以取代它們，並把早期方法（CNN、卷積稀疏表示、堆疊自編碼器）歸納成通用框架。
- **Zhang 與 Demiris（TPAMI 2023）[35]** 依架構（以 CNN、自編碼器、GAN 與 Transformer 為基礎）回顧深度可見光與紅外線融合，並附資料集與評估方法。重點：深度方法已成為這項任務的主流做法。
- **Liu 等人（TPAMI 2025）[2]** 沿著三個維度整理超過 180 種學習式紅外線與可見光方法：資料相容性（對位）、融合（架構與損失）與任務適應（偵測、分割），並在對位、融合與下游任務上比較它們。
- **Zhang（TPAMI 2022）[50]** 綜述深度多焦點融合（方法、資料集、指標），並以實驗比較深度與傳統方法。
- **Xu 等人（Remote Sensing 2022）[51]** 回顧針對靜態與動態場景的多曝光融合，並用九項指標比較 18 種方法。

僅線上版：*Liu* 等人的紅外線與可見光融合綜述（*TPAMI 2025*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-fusion/papers/liu-等人的紅外線與可見光融合綜述 tpami-2025/*

最新進展（2023–2026）。有四個趨勢很突出。第一，**融合越來越以下游任務來評斷**：方法以偵測或分割損失訓練，M3FD 與 FMB 等基準附有標註，融合損失本身也可以從任務損失中學習 [27]、[28]、[40]。第二，**生成式先驗與語言**已經進場：不需要特定任務訓練的擴散式融合 [41]、由成像先驗做的自監督訓練 [52]，以及也能處理退化來源的文字引導融合 [42]。第三，**對位正在移進模型內部**，出現了針對未對齊紅外線與可見光影像對的聯合對位與融合網路 [43]、[44]。第四，**影片融合**正在起步：一項 CVPR 2026 的工作引入了 M3SVD，包含 220 段同步且已對位的紅外線與可見光影片、共 153,797 個影格，以及一個以雙時間注意力減輕逐影格影像融合時間不一致性的模型 [53]。對多焦點融合，一項 2020 年的基準研究發現，用模擬資料訓練的深度方法無法很好地泛化到具挑戰性的真實影像對，而傳統方法仍具競爭力 [7]。對於聲稱大幅進步的說法要小心，並檢查它們依據的是哪些指標。

尚未解決的問題。

- **評估**。仍然沒有任何一個指標或一組指標，既與人類判斷一致又與任務表現一致；2025 年的綜述把更好的評估指標列為必要的方向 [2]。
- **真實的錯位與退化**。大多數方法假設來源已對位而且乾淨。具有對位真值的基準，以及對任一來源中的雜訊、模糊、低光與惡劣天氣都穩健的融合，仍待解決 [2]、[42]。
- **真實的散焦**。要讓多焦點融合能泛化，需要能重現散焦擴散效應的訓練資料，以及能處理深度邊緣的方法 [6]、[7]。
- **效率**。許多近期的網路很大；在無人機、汽車與手持裝置上做融合，需要輕量、對硬體友善的設計 [2]（通用技術見 DLCV 09C）。

- **影片**。研究一直在融合影像而不是影片，主要是因為缺少大型的同步多感測器影片資料集 [53]；以低成本達到時間一致性仍是待解決的問題。

重點整理

- 選擇融合規則前，先決定各來源應提供什麼。
- 平均可能削弱單一來源特徵，權重應反映有用資訊。
- 多尺度混合把細節選擇與粗尺度亮度混合分開，降低接縫風險。
- 沒有唯一融合目標時，訓練損失表達所需答案及其取捨。
- 配準錯誤會產生重影；須檢查邊界、光暈、色彩、雜訊與來源特定目標。
- 搭配多類指標、局部裁切與任務結果；高分也可能獎勵雜訊或單一來源複製。

練習

1. **平均下的對比**。溫暖的目標在紅外線影像中的強度是 $b + a$，在可見光影像中是 b，兩者的背景強度都是 b。(a) 證明平均後的目標對比是 $a/2$，而任何權重為 $w_{\mathrm{IR}}, w_{\mathrm{VIS}}$ 的逐像素加權平均都會給出 $w_{\mathrm{IR}}\, a$。(b) 在我們的玩具範例中，平均的人物對比是 0.257，金字塔融合是 0.513。金字塔融合在目標上的行為，像是紅外線來源取多少權重？為什麼它能做到這一點，又不會讓其他地方的可見光紋理變暗？

   提示

   (a) 利用 $w_{\mathrm{IR}} + w_{\mathrm{VIS}} = 1$，$F = w_{\mathrm{IR}}(b + a) + w_{\mathrm{VIS}}\, b = b + w_{\mathrm{IR}}\, a$。(b) 0.513 大約是 0.257 的兩倍，所以在目標上，金字塔融合的行為像是 $w_{\mathrm{IR}} \approx 1$。它之所以做得到，是因為權重不是全域的：粗略基底取較亮的來源（紅外線目標），細節層則取任何一個更強的細節，也就是目標以外的可見光紋理。全域權重則必須在兩者之間取捨。

2. **對焦視窗與清理**。在多焦點的程式碼中，改變對焦測度的平滑 $\sigma \in \{1, 2, 4\}$ 與中值視窗 $k \in \{1, 15, 31, 61\}$（$k = 1$ 表示不清理），最後一律接標準差為 2 的高斯柔化混合。回報 PSNR。哪種設定贏？為什麼在真實照片上答案可能不同？

   提示

   在我們的執行中，所有設定都落在 39.43 到 40.05 dB 之間；最好的是 $\sigma = 2$ 且不做中值清理（40.05 dB），而最大的視窗 $k = 61$ 對每個 σ 都最差（約 39.4–39.5 dB）。不清理時，柔化混合會在布滿斑點的平坦區域混合 A 與 B，使它們各自獨立的雜訊被平均掉，那裡的 PSNR 略微上升；很大的中值視窗則會讓圓盤邊界附近的決策圖錯位。在真實照片上，沒清理的斑點會在來源不同的地方（雜訊、小幅錯位、散焦擴散效應）顯現為明顯的斑塊，所以對外觀而言，清理通常比在玩具範例的 PSNR 上顯得更重要。

3. **區域式選擇**。把金字塔程式碼中的 `pick_stronger` 換成比較局部能量的規則：用標準差為 s 的高斯平滑 $(L_\ell^A)^2$ 與 $(L_\ell^B)^2$，並選擇能量較大者的係數。試 $s \in \{1, 1.5, 3\}$，並與逐係數規則比較。

提示

使用 `ndi.gaussian_filter(p**2, s)`。在我們的執行中，PSNR 從 37.60 dB（逐係數）上升到 $s = 1$, 1.5, 3 時的 39.14、39.42 與 39.62 dB，接近決策圖。平滑活動量使相鄰的係數做出相同的決策，這正是區域式規則與 Li 等人的一致性驗證所要達成的：在多焦點融合中，對焦是區域的性質，不是單一係數的性質。

4. **曝光融合中的金字塔深度與品質測度**。(a) 用 $n = 0, 1, 3, 5, 7, 9$ 層的金字塔重跑曝光融合（$n = 0$ 就是逐像素混合），回報曝光良好比例；並觀察影像。(b) 一次只用一種品質測度（只用對比、只用飽和度，或只用曝光良好度），在完整深度下做融合。你觀察到什麼？

提示

(a) 在我們的執行中，$n = 0$ 時比例是 72.3%，$n = 1, 3, 5, 7$ 時依序是 72.2%、70.9%、69.6% 與 68.5%，$n = 9$ 時是 73.8%；$n = 7$ 時與 9 層結果的平均絕對差仍有 0.068。淺的金字塔保留接縫與低頻亮度誤差，因為缺少了權重混合得最平滑的最粗幾層；在 512 像素的影像上，即使從 7 層到 9 層，結果也變化很大。(b) 只用曝光良好度，得到最高的比例（77.5%），只用對比是 69.1%，只用飽和度是 71.5%。Mertens 等人也發現，單靠曝光良好度就能給出相當不錯的影像；對比與飽和度對細節與色彩有貢獻，而這個計數並不量測它們。

5. **雜訊如何灌高空間頻率**。令 $F' = F + n$，其中 n 是每個像素上獨立、均值為零、變異數為 σ^2 的雜訊。(a) 證明 SF^2 的期望值大約增加 $4\sigma^2$。(b) 在玩具範例上驗證：對平均影像加上 $\sigma = 0.02$, 0.04, 0.08 的雜訊，把量到的 SF 與 $\sqrt{\mathrm{SF}^2 + 4\sigma^2}$ 比較。

提示

(a) F' 的水平差值，等於 F 的差值加上 $n(i,j) - n(i,j-1)$，後者的變異數是 $2\sigma^2$，且與 F 不相關；所以 $\mathbb{E}[\mathrm{RF}'^2] \approx \mathrm{RF}^2 + 2\sigma^2$，CF 同理，於是 $\mathbb{E}[\mathrm{SF}'^2] \approx \mathrm{SF}^2 + 4\sigma^2$。(b) 在我們的執行中（隨機種子與程式碼中的不同），$\sigma = 0.02$ 時量到與預測的 SF 都是 0.0528，0.04 時都是 0.0871，0.08 時分別是 0.1631 與 0.1637；0.08 時的小差距來自截斷到 $[0, 1]$。EN 從 6.69 上升到 6.99、7.20 與 7.49 位元。

6. **互資訊獎勵複製**。(a) 證明若 $F = I_A$，則 $\mathrm{MI}_{A,F} + \mathrm{MI}_{B,F} = H(I_A) + \mathrm{MI}(I_A, I_B)$，其中 H 是分箱後影像的熵。(b) 用 64 個區間在玩具範例上驗證。(c) 用 16、64 與 256 個區間，計算平均影像與可見光影像的 MI 分數。排名會隨區間數改變嗎？數值呢？

提示

(a) $\mathrm{MI}(X, X) = H(X)$，因為知道 X 就消除了關於 X 的所有不確定性。(b) 在我們的執行中，$H(\mathrm{vis}) = 4.728$ 位元，$\mathrm{MI}(\mathrm{ir}, \mathrm{vis}) = 0.995$ 位元，兩者相加是 5.723，也就是程式碼中「僅可見光」的 MI 分數。(c) 平均影像得到 2.64、4.33 與 5.00 位元，可見光影像得到 3.42、5.72 與 7.82 位元：排名不變，但可見光影像的數值從 16 個區間到 256 個區間增加了一倍以上，所以使用不同區間數或不同實作的論文，其 MI 數值不能直接比較。

參考文獻

1. Y. Liu, X. Chen, Z. Wang, Z. J. Wang, R. K. Ward and X. Wang, “Deep learning for pixel-level image fusion: Recent advances and future prospects,” *Information Fusion*, vol. 42, pp. 158–173, 2018. doi
2. J. Liu, G. Wu, Z. Liu, D. Wang, Z. Jiang, L. Ma, W. Zhong, X. Fan and R. Liu, “Infrared and visible image fusion: From data compatibility to task adaption,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, no. 4, pp. 2349–2369, 2025. doi arXiv
3. J. Ma, Y. Ma and C. Li, “Infrared and visible image fusion methods and applications: A survey,” *Information Fusion*, vol. 45, pp. 153–178, 2019. doi
4. P. E. Debevec and J. Malik, “Recovering high dynamic range radiance maps from photographs,” in *Proc. SIGGRAPH*, 1997, pp. 369–378. doi
5. T. Mertens, J. Kautz and F. Van Reeth, “Exposure fusion: A simple and practical alternative to high dynamic range photography,” *Computer Graphics Forum*, vol. 28, no. 1, pp. 161–171, 2009. doi
6. S. Xu, X. Wei, C. Zhang, J. Liu and J. Zhang, “MFFW: A new dataset for multi-focus image fusion,” arXiv:2002.04780, 2020. arXiv
7. X. Zhang, “Multi-focus image fusion: A benchmark,” arXiv:2005.01116, 2020. arXiv
8. P. J. Burt and E. H. Adelson, “The Laplacian pyramid as a compact image code,” *IEEE Transactions on Communications*, vol. 31, no. 4, pp. 532–540, 1983. doi
9. P. J. Burt and R. J. Kolczynski, “Enhanced image capture through fusion,” in *Proc. 4th International Conference on Computer Vision (ICCV)*, 1993, pp. 173–182. doi
10. P. J. Burt and E. H. Adelson, “A multiresolution spline with application to image mosaics,” *ACM Transactions on Graphics*, vol. 2, no. 4, pp. 217–236, 1983. doi
11. H. Li, B. S. Manjunath and S. K. Mitra, “Multisensor image fusion using the wavelet transform,” *Graphical Models and Image Processing*, vol. 57, no. 3, pp. 235–245, 1995. doi
12. S. Li, X. Kang and J. Hu, “Image fusion with guided filtering,” *IEEE Transactions on Image Processing*, vol. 22, no. 7, pp. 2864–2875, 2013. doi
13. K. He, J. Sun and X. Tang, “Guided image filtering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013. doi
14. B. Yang and S. Li, “Multifocus image fusion and restoration with sparse representation,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 4, pp. 884–892, 2010. doi
15. M. Nejati, S. Samavi and S. Shirani, “Multi-focus image fusion using dictionary-based sparse representation,” *Information Fusion*, vol. 25, pp. 72–84, 2015. doi
16. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
17. X. Zhang, P. Ye and G. Xiao, “VIFB: A visible and infrared image fusion benchmark,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2020. CVF arXiv
18. X. Zhang, “Benchmarking and comparing multi-exposure image fusion algorithms,” *Information Fusion*, vol. 74, pp. 111–131, 2021. doi arXiv
19. A. M. Eskicioglu and P. S. Fisher, “Image quality measures and their performance,” *IEEE Transactions on Communications*, vol. 43, no. 12, pp. 2959–2965, 1995. doi

20. G. Qu, D. Zhang and P. Yan, “Information measure for performance of image fusion,” *Electronics Letters*, vol. 38, no. 7, pp. 313–315, 2002. doi
21. M. Hossny, S. Nahavandi and D. Creighton, “Comments on ‘Information measure for performance of image fusion’,” *Electronics Letters*, vol. 44, no. 18, pp. 1066–1067, 2008. doi
22. C. S. Xydeas and V. Petrović, “Objective image fusion performance measure,” *Electronics Letters*, vol. 36, no. 4, pp. 308–309, 2000. doi
23. G. Piella and H. Heijmans, “A new quality metric for image fusion,” in *Proc. IEEE International Conference on Image Processing (ICIP)*, 2003, vol. 3, pp. III-173–176. doi
24. Y. Han, Y. Cai, Y. Cao and X. Xu, “A new image fusion performance metric based on visual information fidelity,” *Information Fusion*, vol. 14, no. 2, pp. 127–135, 2013. doi
25. K. Ma, K. Zeng and Z. Wang, “Perceptual quality assessment for multi-exposure image fusion,” *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3345–3356, 2015. doi
26. Z. Liu, E. Blasch, Z. Xue, J. Zhao, R. Laganiere and W. Wu, “Objective assessment of multiresolution image fusion algorithms for context enhancement in night vision: A comparative study,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 1, pp. 94–109, 2012. doi
27. J. Liu, X. Fan, Z. Huang, G. Wu, R. Liu, W. Zhong and Z. Luo, “Target-aware dual adversarial learning and a multi-scenario multi-modality benchmark to fuse infrared and visible for object detection,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 5802–5811. CVF arXiv M3FD
28. J. Liu, Z. Liu, G. Wu, L. Ma, R. Liu, W. Zhong, Z. Luo and X. Fan, “Multi-interactive feature learning and a full-time multi-modality benchmark for image fusion and segmentation,” in *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 8115–8124. CVF arXiv
29. Y. Zhang, Y. Liu, P. Sun, H. Yan, X. Zhao and L. Zhang, “IFCNN: A general image fusion framework based on convolutional neural network,” *Information Fusion*, vol. 54, pp. 99–118, 2020. doi
30. H. Li and X.-J. Wu, “DenseFuse: A fusion approach to infrared and visible images,” *IEEE Transactions on Image Processing*, vol. 28, no. 5, pp. 2614–2623, 2019. doi arXiv
31. K. R. Prabhakar, V. S. Srikar and R. V. Babu, “DeepFuse: A deep unsupervised approach for exposure fusion with extreme exposure image pairs,” in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 4724–4732. doi CVF
32. H. Xu, J. Ma, J. Jiang, X. Guo and H. Ling, “U2Fusion: A unified unsupervised image fusion network,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, pp. 502–518, 2022. doi RoadScene
33. L. Tang, J. Yuan and J. Ma, “Image fusion in the loop of high-level vision tasks: A semantic-aware real-time infrared and visible image fusion network,” *Information Fusion*, vol. 82, pp. 28–42, 2022. doi
34. J. Ma, L. Tang, F. Fan, J. Huang, X. Mei and Y. Ma, “SwinFusion: Cross-domain long-range learning for general image fusion via Swin Transformer,” *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 7, pp. 1200–1217, 2022. doi
35. X. Zhang and Y. Demiris, “Visible and infrared image fusion using deep learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 8, pp. 10535–10554, 2023. doi
36. Y. Liu, X. Chen, H. Peng and Z. Wang, “Multi-focus image fusion with a deep convolutional neural network,” *Information Fusion*, vol. 36, pp. 191–207, 2017. doi
37. K. Ma, Z. Duanmu, H. Zhu, Y. Fang and Z. Wang, “Deep guided learning for fast multi-

exposure image fusion," *IEEE Transactions on Image Processing*, vol. 29, pp. 2808–2819, 2020. doi

38. Z. Zhao, H. Bai, J. Zhang, Y. Zhang, S. Xu, Z. Lin, R. Timofte and L. Van Gool, "CDDFuse: Correlation-driven dual-branch feature decomposition for multi-modality image fusion," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 5906–5916. CVF arXiv
39. J. Ma, W. Yu, P. Liang, C. Li and J. Jiang, "FusionGAN: A generative adversarial network for infrared and visible image fusion," *Information Fusion*, vol. 48, pp. 11–26, 2019. doi ·public article transcription
40. H. Bai, J. Zhang, Z. Zhao, Y. Wu, L. Deng, Y. Cui, T. Feng and S. Xu, "Task-driven image fusion with learnable fusion loss," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025, pp. 7457–7468. CVF arXiv
41. Z. Zhao, H. Bai, Y. Zhu, J. Zhang, S. Xu, Y. Zhang, K. Zhang, D. Meng, R. Timofte and L. Van Gool, "DDFM: Denoising diffusion model for multi-modality image fusion," in *Proc. IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 8082–8093. CVF arXiv
42. X. Yi, H. Xu, H. Zhang, L. Tang and J. Ma, "Text-IF: Leveraging semantic text guidance for degradation-aware and interactive image fusion," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 27026–27035. CVF arXiv
43. D. Wang, J. Liu, X. Fan and R. Liu, "Unsupervised misaligned infrared and visible image fusion via cross-modality image generation and registration," in *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 2022, pp. 3508–3515. doi arXiv
44. L. Tang, Y. Deng, Y. Ma, J. Huang and J. Ma, "SuperFusion: A versatile image registration and fusion network with semantic awareness," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 12, pp. 2121–2137, 2022. doi
45. A. Toet, "The TNO multiband image data collection," *Data in Brief*, vol. 15, pp. 249–251, 2017. doi
46. L. Tang, J. Yuan, H. Zhang, X. Jiang and J. Ma, "PIAFusion: A progressive infrared and visible image fusion network based on illumination aware," *Information Fusion*, vol. 83–84, pp. 79–92, 2022. doi MSRS
47. Q. Ha, K. Watanabe, T. Karasawa, Y. Ushiku and T. Harada, "MFNet: Towards real-time semantic segmentation for autonomous vehicles with multi-spectral scenes," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 5108–5115. doi
48. X. Jia, C. Zhu, M. Li, W. Tang and W. Zhou, "LLVIP: A visible-infrared paired dataset for low-light vision," in *Proc. IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2021, pp. 3496–3504. CVF arXiv
49. J. Cai, S. Gu and L. Zhang, "Learning a deep single image contrast enhancer from multi-exposure images," *IEEE Transactions on Image Processing*, vol. 27, no. 4, pp. 2049–2062, 2018. doi
50. X. Zhang, "Deep learning-based multi-focus image fusion: A survey and a comparative study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, 2022. doi
51. F. Xu, J. Liu, Y. Song, H. Sun and X. Wang, "Multi-exposure image fusion techniques: A comprehensive review," *Remote Sensing*, vol. 14, no. 3, art. 771, 2022. doi
52. Z. Zhao, H. Bai, J. Zhang, Y. Zhang, K. Zhang, S. Xu, D. Chen, R. Timofte and L. Van Gool, "Equivariant multi-modality image fusion," in *Proc. IEEE/CVF Conference on Computer Vision*

and Pattern Recognition (CVPR), 2024, pp. 25912–25921. CVF arXiv
53. L. Tang, Y. Wang, M. Gong, Z. Li, Y. Deng, X. Yi, C. Li, H. Xu, H. Zhang and J. Ma, "VideoFusion: A spatio-temporal collaborative network for multi-modal video fusion," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026, pp. 19559–19569. CVF
54. N. Yokoya, C. Grohnfeldt and J. Chanussot, "Hyperspectral and multispectral data fusion: A comparative review of the recent literature," *IEEE Geoscience and Remote Sensing Magazine*, 2017. doi
55. H. R. Sheikh and A. C. Bovik, "Image information and visual quality," *IEEE Transactions on Image Processing*, 2006. doi
56. H. Xu et al., "U2Fusion," official implementation accompanying the TPAMI paper, 2022. code
57. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015. arXiv
58. J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, 2017. doi
59. Y. Zhang et al., "IFCNN," official implementation and errata accompanying the Information Fusion paper, 2020. code
60. K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," *CVPR*, 2016. CVF
61. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," *CVPR*, 2009. doi
62. N. Silberman, D. Hoiem, P. Kohli and R. Fergus, "Indoor segmentation and support inference from RGBD images," *ECCV*, 2012. project
63. P. Dhariwal and A. Nichol, "Diffusion models beat GANs on image synthesis," *NeurIPS*, 2021. arXiv
64. J. Liu et al., "TarDAL," official implementation accompanying the CVPR paper, 2022. code
65. G. Jocher et al., "YOLOv5," Ultralytics object-detection software, 2020 onward. code
66. J. Ma et al., "FusionGAN," official implementation accompanying the Information Fusion paper, 2019. code
67. L. Tang, J. Yuan and J. Ma, "SeAFusion," official implementation accompanying the Information Fusion paper, 2022. code
68. T. Mertens, J. Kautz and F. Van Reeth, "Exposure Fusion," in *Proc. Pacific Graphics*, 2007. paper ·author project

DLCV 11・三維重建：從多視角幾何到 NeRF 與 Gaussian Splatting

一組照片如何變成三維模型：相機幾何、運動恢復結構與多視角立體視覺、神經輻射場與有號距離場、3D Gaussian Splatting、網格萃取、前饋式重建模型，以及如何評估重建結果，並附上經過測試的 NumPy 與 OpenCV 程式碼。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/

你將學到

- 為算繪、幾何或編輯選擇場景表示。
- 使用相機投影、對極幾何與三角測量。
- 追蹤運動恢復結構與多視角立體重建。
- 比較神經場、高斯潑濺與前饋重建。
- 分開評估保留視角與幾何準確度。

先看全貌

你繞著一尊雕像走一圈，用手機拍了三十張照片。三維重建把這些照片變成一個可以旋轉、量測、3D 列印，或放進遊戲的三維模型。這裡有兩個問題必須一起回答：每張照片的相機在哪裡，以及雕像表面的每一小塊在哪裡。兩者都沒有現成答案，而且彼此互相幫忙。

> **用最簡單的話說**
>
> 閉上一隻眼睛、舉起一根手指，換另一隻眼睛看，手指會「跳」一下。近的東西跳得很多，遠的東西幾乎不動。大腦就是靠這個跳動量感覺到深度。電腦對從不同位置拍的照片做同一件事：在兩張照片裡找到同一個點，量它「跳」了多少，就能算出它有多遠。對幾百萬個點都這樣做，就得到形狀。比較新的方法多了一招：先猜一個三維場景，再把每張照片應該長什麼樣子畫出來，畫得和真照片不一樣的地方，就修正原本的猜測。

為什麼重要。地圖、電影布景、文化遺產典藏、機器人環境、擴增實境錨點，以及工廠的數位孿生（digital twin），都靠重建建立。它也清楚說明了深度學習如何改變電腦視覺：傳統流程是純粹的幾何與最佳化；神經輻射場（neural radiance field, NeRF）[5] 讓場景本身變成一個訓練出來的網路；3D Gaussian Splatting（3DGS）[6] 把網路換成數百萬個明確的小色塊，可以即時算繪；最新的模型則一次前向傳遞就預測出相機與幾何。

圖 1 是本篇的路線圖。左邊的傳統階段至今仍餵給幾乎所有現代方法：NeRF 與 3DGS 都是用 COLMAP [2]、[4] 這類運動恢復結構軟體算出的相機位姿來訓練的。

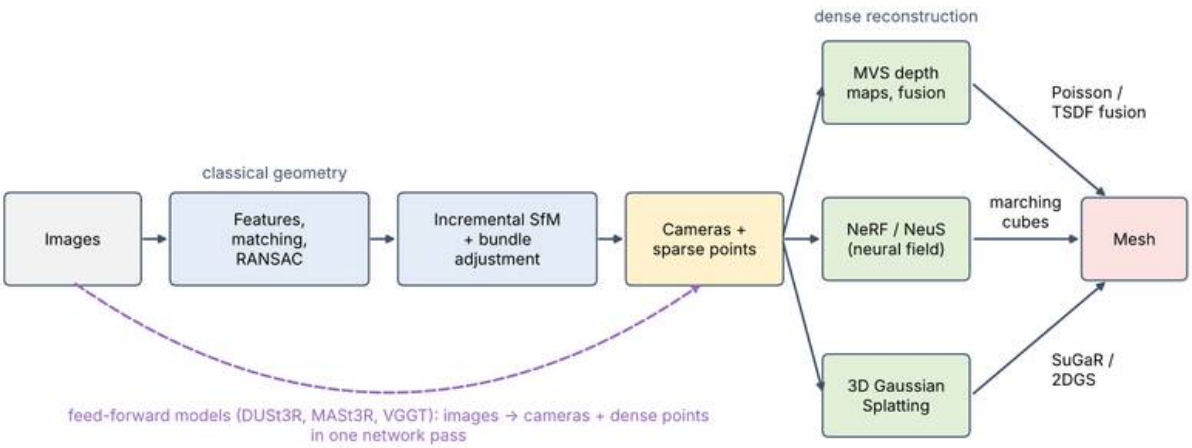


圖 *1* — 本篇涵蓋的重建流程。傳統幾何（左）給出相機與稀疏點；稠密方法（右）給出完整模型；前饋式網路（虛線）跳過傳統階段。

本篇假設你已經知道 IP 12 的關鍵點與描述子配對，也懂 DLCV 01 的深度網路基礎。幾何方面的標準教科書是 Hartley 與 Zisserman [1]。

相關篇章。 DLCV 11A 深入介紹重建背後的密集對應問題——立體匹配、單目深度與光流；DLCV 03B 則利用三維幾何來偵測物體。

什麼是三維重建

白話版。從靜態場景照片估計相機姿態，以及形狀與外觀的三維表示。

精確版。給定靜態場景的 N 張影像 $I_1, \dots, I_N$。我們要求出每張影像的相機參數 $(K_i, R_i, \mathbf{t}_i)$（下一節定義），以及一個場景模型 $\mathcal{S}$，使得用第 i 號相機算繪 $\mathcal{S}$ 時能重現 I_i。有兩種變體很重要。**幾何重建**要求精確的曲面，並以雷射或結構光掃描作為標準來評判。**新視角合成**（novel-view synthesis）要求從沒有拍過的相機位置產生逼真的影像，並以保留不用的照片來評判。NeRF 與 3DGS 是為第二個目標設計的，所以要從它們取得好的曲面，需要額外的功夫。

這個問題有個內建的限制：光靠照片，整個場景放大或縮小都不會改變任何一張影像。近距離拍的玩具屋和遠距離拍的真房子，可以產生一模一樣的照片。因此除非有其他資訊（例如已知的距離，或校正過的立體相機組）來固定，否則從影像重建只能決定到**差一個尺度**（up to scale；也包括世界座標系的選擇）[1]。

儲存三維場景的七種方式

白話版。依算繪、量測、編輯或學習需求選表示；圖 2 用同一場景比較常見形式。

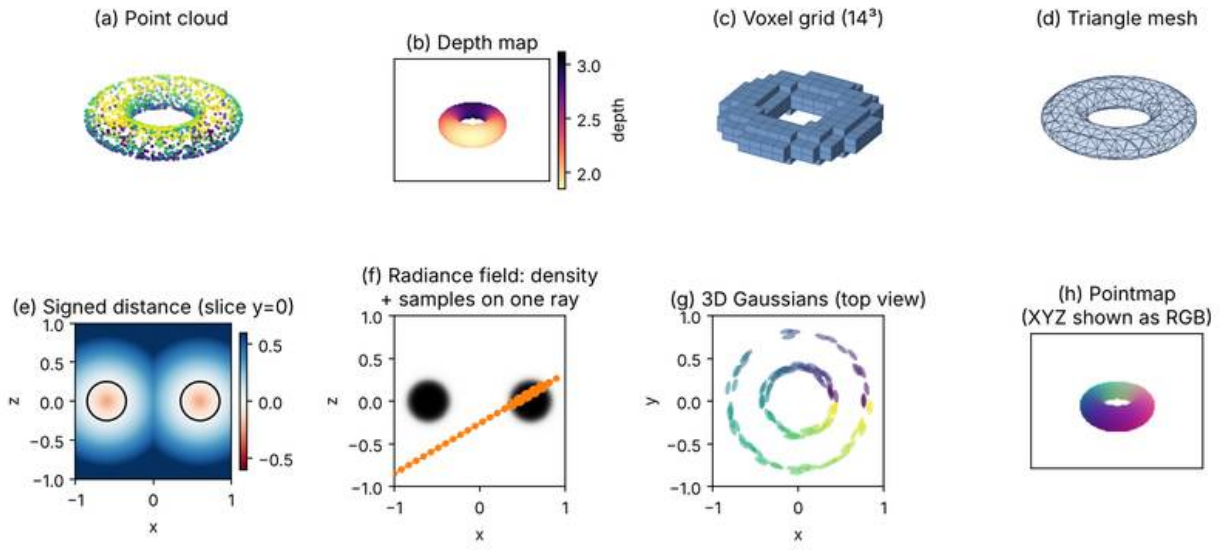


圖 2 — 一個形狀，八種編碼：(*a*) 點雲、(*b*) 深度圖、(*c*) 體素格、(*d*) 三角形網格、(*e*) 有號距離函數、(*f*) 輻射場（密度切面與沿一條光線的取樣點）、(*g*) *3D* 高斯、(*h*) 點圖，也就是前饋式模型預測的逐像素三維點影像。

表示法	儲存什麼	優點	缺點
點雲（point cloud）	三維點的清單，常附顏色與法向量	簡單；三角測量與深度感測器的直接輸出	沒有曲面與連接關係；有破洞；很難算繪得漂亮
深度圖（depth map）	一台相機每個像素的一個距離	適合影像網路；容易逐視角估計	只涵蓋一個視角；必須融合
體素格（voxel grid）	三維格網每個格子的一個值（佔據、顏色）	規則，容易用三維卷積處理	記憶體隨解析度的三次方成長
網格（mesh）	頂點與三角形	精簡、可編輯，是圖學引擎與 3D 印表機使用的格式	拓樸固定；很難直接從影像最佳化
有號距離函數（SDF）	到最近表面的距離，內部為負	以零等值面表示不漏水的表面；梯度就是法向量	需要封閉曲面；算繪前必須轉換
輻射場（radiance field）	每個三維點的密度與隨視角變化的顏色	逼真的新視角，包含細薄與毛茸茸的結構	光線步進算繪很慢；曲面是隱式的
3D 高斯	許多帶有顏色與不透明度的小橢球	即時算繪；明確且可編輯	基元數量多（吃記憶體）；曲面定義不明確

SDF 可以是儲存下來的格網，也可以是網路算出的函數：DeepSDF [20] 學習一個輸出有號距離的網路，Occupancy Networks [21] 則學習某點在物體內部的機率。兩者都稱為**神經隱式**（neural implicit）表示法，因為曲面是以一個學出來的函數的等值面隱含地給出。

相機幾何基礎

白話版。針孔模型透過相機變換、投影與透視除法，把三維點映射到影像座標。

針孔相機：內參與外參

相機對一個三維點做兩件事。它先描述這個點相對於相機自己的位置（在相機的左邊多遠、上方多高、前方多遠），再除以它的距離，把它壓平到感測器上。用符號寫：世界座標中的點 $\mathbf{X} \in \mathbb{R}^3$ 先被移到相機座標系，再投影：

$$\mathbf{X}_c = R\,\mathbf{X} + \mathbf{t}, \qquad \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\mathbf{X}_c, \qquad K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix},$$

其中 R（3×3 旋轉）與 $\mathbf{t}$（平移）是**外參**（extrinsics）：它們說明相機在哪裡、朝哪個方向；$\mathbf{X}_c$ 是相機座標中的點，z 軸沿著視線方向；K 存放**內參**（intrinsics）：以像素為單位的焦距 f_x, f_y、光軸與影像相交的主點 (c_x, c_y)，以及偏斜量 s（真實感測器中為零）；(u, v) 是像素；λ 等於該點的深度 z_c。除以深度，就是遠的東西看起來比較小的原因。相機中心在世界座標中是 $C = -R^\top\mathbf{t}$。把兩個步驟合併，得到 3×4 的**投影矩陣**（projection matrix）$P = K[R \mid \mathbf{t}]$。

真實的鏡頭還會讓直線彎曲（徑向畸變）；校正工具會估計額外的畸變係數並先把影像去畸變，所以本篇其餘部分假設是理想的針孔相機。這裡使用的齊次座標（結尾帶一個 1）與 IP 02 單應性筆記中的相同（單應性：投影轉換）。

兩個視角：對極幾何

白話版。如果你在一張照片裡看到一個點，並不知道它有多遠，所以在三維中它可能落在一條射線上的任何位置。在第二張照片裡，這條射線會呈現為一條直線。真正的對應點一定落在這條線上。這把二維搜尋變成一維搜尋，反過來，也讓我們只憑配對點就能還原兩台相機的相對擺放。

圖 3 (a) 畫出這個幾何。兩個相機中心與三維點 $\mathbf{X}$ 張成一個平面，稱為**對極平面**（epipolar plane）。它與每張影像相交成一條**對極線**（epipolar line）。相機 1 同一條射線上的任何點 $\mathbf{X}'$，在相機 1 中都有相同的像 $\mathbf{x}$，而它在相機 2 中的像會沿著那條線移動。

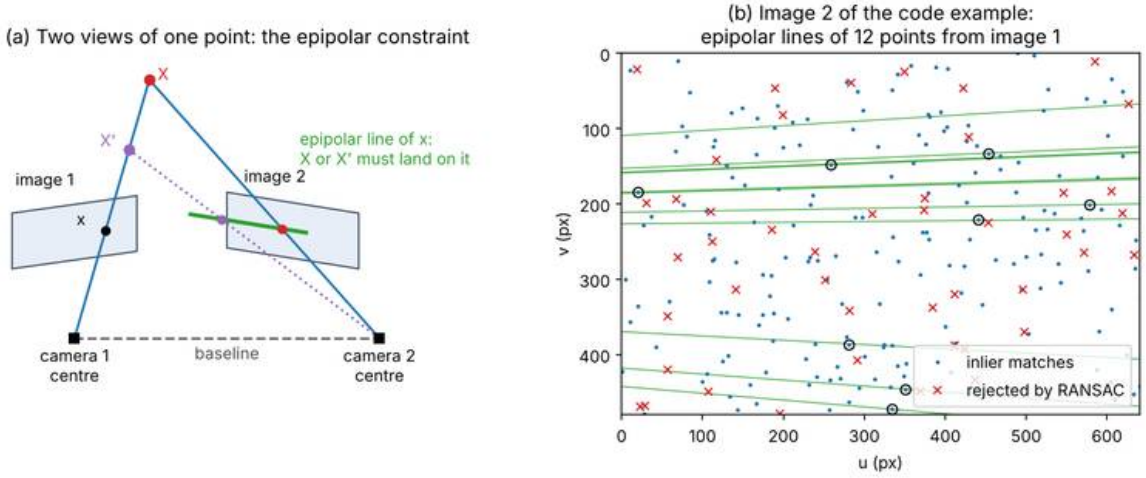


圖 3 — (*a*) 對極約束：通過 x 的射線上所有的點，都投影到影像 2 的同一條線上。(*b*) 下方程式範例的影像 2：由估計的本質矩陣算出的對極線，通過相配的點（圈起處）。

用代數來說，令 $\hat{\mathbf{x}}_1 = K_1^{-1}(u_1, v_1, 1)^\top$ 與 $\hat{\mathbf{x}}_2 = K_2^{-1}(u_2, v_2, 1)^\top$ 為一組配對點的正規化座標（把內參去掉的像素）。若相機 2 相對於相機 1 的旋轉為 R、平移為 $\mathbf{t}$，則每個正確的配對都滿足

$$\hat{\mathbf{x}}_2^\top E\,\hat{\mathbf{x}}_1 = 0, \qquad E = [\mathbf{t}]_\times R, \qquad [\mathbf{t}]_\times = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix},$$

其中 E 是 3×3 的**本質矩陣**（essential matrix）[7]，$[\mathbf{t}]_\times$ 是與 $\mathbf{t}$ 做外積的矩陣。這個式子說的是：兩條視線與基線落在同一個平面上。在像素座標中，同一個約束寫成 $\mathbf{x}_2^\top F \mathbf{x}_1 = 0$，其中**基礎矩陣**（fundamental matrix）$F = K_2^{-\top} E K_1^{-1}$，$\mathbf{x}_1, \mathbf{x}_2$ 是齊次像素座標。$F\mathbf{x}_1$ 正是圖 3 (b) 中畫在影像 2 上的對極線。

用白話說：**E 編碼了兩台已校正相機的相對位姿，F 則在內參未知時做同樣的事**。E 有五個自由度：旋轉三個，平移的方向兩個，平移的長度則因尺度不確定性而遺失。所以五組配對就夠了，這是 Nistér 的五點演算法 [9] 所解的最小情況。有八組或更多配對時，約束對 E 或 F 的九個元素是線性的：八點演算法 [7]、[8] 每組配對疊一個方程式，再用奇異值分解取零向量。

僅線上版：正規化八點演算法 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/正規化八點演算法/*

知道 E 之後，可以把它分解成 R 和 $\mathbf{t}$ 的方向。分解會得到四個候選位姿；只有一個會讓這些點位在兩台相機的前方，這個檢查稱為**手性檢查**（cheirality check，或 chirality check）[1]。

三角測量

白話版。兩台相機都已知時，一組配對點給出兩條射線。它們（幾乎）相交的地方就是三維點。有雜訊時，兩條線永遠不會剛好相交，所以我們挑一個最能同時解釋兩個觀測的點。

最簡單的方法是線性的。每張影像對未知點給出兩個線性方程式（把未知深度消去的針孔方程式），四個方程式就像八點系統一樣，用 SVD 取一個小矩陣的零向量來解 [1]。三角測量出的點好不好，用它的**重投影誤差**（reprojection error）來判斷：把它投影回每張影像，量它與觀測點之間的像素距離。

僅線上版：線性三角測量（DLT）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/線性三角測量 dlt/*

準確度與兩條射線的夾角關係很大。對焦距為 f、基線為 B 的兩台平行相機，深度為 Z 的點在兩張影像之間會位移**視差**（disparity）$d = fB/Z$ 像素，所以視差的小誤差 δd 會造成深度誤差 $\delta Z \approx Z^2 \delta d/(fB)$。深度誤差隨距離的平方成長，並隨基線變寬而縮小；練習 1 會用數字說明。

程式碼：相對位姿與三角測量

這段程式碼建立一個合成場景，把它投影到兩台帶有 0.5 像素雜訊的相機，將 15% 的配對換成隨機的錯誤配對，再用 OpenCV 還原相對位姿。`cv2.findEssentialMat` 在五點求解器外面包了一層 RANSAC（其文件指明用的是 Nistér 的演算法 [9]）；`cv2.recoverPose` 執行手性檢查。RANSAC 的答案來自最小樣本，所以我們再用所有內點以線性方式重新擬合 E，也就是 IP 12 的 RANSAC 筆記（RANSAC：在有離群值時擬合模型）所說的「在共識集合上重新擬合」那一步。

```
import numpy as np
import cv2

rng = np.random.default_rng(0)
K = np.array([[800.0, 0, 320], [0, 800.0, 240], [0, 0, 1]])   # focal length 800 px,
 ↪  principal point (320, 240)

def look_at(C, target):
    """Rotation of a camera at centre C that looks at target (rows: camera x, y, z axes)."""
    z = (target - C) / np.linalg.norm(target - C)
    x = np.cross([0.0, 1.0, 0.0], z); x /= np.linalg.norm(x)
    return np.stack([x, np.cross(z, x), z])

def project(K, R, t, X):
    """Pinhole projection of world points X (N, 3) to pixels (N, 2)."""
    x = (X @ R.T + t) @ K.T           # world -> camera -> homogeneous pixels
    return x[:, :2] / x[:, 2:]        # divide by depth

def linear_E(x1, x2, K):
    """Least-squares E from many matches: each match gives one linear equation in E's 9
     ↪  entries."""
    Ki = np.linalg.inv(K)
    a = np.c_[x1, np.ones(len(x1))] @ Ki.T               # normalized camera coordinates
    b = np.c_[x2, np.ones(len(x2))] @ Ki.T
    A = np.einsum('ni,nj->nij', b, a).reshape(-1, 9)      # row n encodes b_n^T E a_n = 0
    E = np.linalg.svd(A)[2][-1].reshape(3, 3)             # best null vector of A
    U, _, Vt = np.linalg.svd(E)
    return U @ np.diag([1.0, 1.0, 0.0]) @ Vt              # nearest valid essential matrix

X = rng.uniform([-3, -2, 4], [3, 2, 8], size=(300, 3))           # 300 scene points
C2 = np.array([2.0, 0.0, 0.0])                                    # camera 2 centre: 2 units
 ↪  to the right
R2 = look_at(C2, np.array([0.0, 0.0, 6.0])); t2 = -R2 @ C2        # camera 1 is the world
 ↪  frame
x1 = project(K, np.eye(3), np.zeros(3), X) + rng.normal(0, 0.5, (300, 2))   # 0.5 px noise
x2 = project(K, R2, t2, X) + rng.normal(0, 0.5, (300, 2))
x2[:45] = rng.uniform([0, 0], [640, 480], (45, 2))                # 45 wrong matches (15 %)

def pose_errors(R, t):
    rot = np.degrees(np.arccos(np.clip((np.trace(R.T @ R2) - 1) / 2, -1, 1)))
    return rot, np.degrees(np.arccos(t.ravel() @ t2 / np.linalg.norm(t2)))

E, mask = cv2.findEssentialMat(x1, x2, K, method=cv2.RANSAC, prob=0.999, threshold=1.5)
_, R, t, mask = cv2.recoverPose(E, x1, x2, K, mask=mask)
inl = mask.ravel() > 0
print(f"inliers kept: {inl.sum()} of 300 | wrong matches rejected: {(~inl[:45]).sum()} of
 ↪  45")
print("RANSAC estimate      : rotation error %.2f deg, translation direction error %.2f deg"
 ↪  % pose_errors(R, t))
_, R, t, _ = cv2.recoverPose(linear_E(x1[inl], x2[inl], K), x1[inl], x2[inl], K)
print("refit on all inliers : rotation error %.2f deg, translation direction error %.2f deg"
 ↪  % pose_errors(R, t))
print(f"length of estimated t: {np.linalg.norm(t):.2f} (true baseline
 ↪  {np.linalg.norm(C2):.2f})")
```

```
P1 = K @ np.hstack([np.eye(3), np.zeros((3, 1))])     # P1 = K [I | 0]
P2 = K @ np.hstack([R, t])                            # P2 = K [R | t]
Xh = cv2.triangulatePoints(P1, P2, x1[inl].T, x2[inl].T)
Xe = (Xh[:3] / Xh[3]).T                               # homogeneous -> Euclidean
res = np.r_[np.linalg.norm(project(K, np.eye(3), np.zeros(3), Xe) - x1[inl], axis=1),
            np.linalg.norm(project(K, R, t.ravel(), Xe) - x2[inl], axis=1)]
print(f"reprojection error: RMS {np.sqrt(np.mean(res ** 2)):.2f} px")
rel = np.linalg.norm(2.0 * Xe - X[inl], axis=1) / np.linalg.norm(X[inl], axis=1)   # scale
 ↪  fixed by the true baseline
print(f"3D error after fixing the scale: median {100 * np.median(rel):.2f} % of the
 ↪  distance")
```

```
inliers kept: 250 of 300 | wrong matches rejected: 44 of 45
RANSAC estimate      : rotation error 0.30 deg, translation direction error 0.42 deg
refit on all inliers : rotation error 0.07 deg, translation direction error 0.46 deg
length of estimated t: 1.00 (true baseline 2.00)
reprojection error: RMS 0.40 px
3D error after fixing the scale: median 0.31 % of the distance
```

這些數字只來自這次合成實驗。RANSAC 剔除了 45 個錯誤配對中的 44 個。用所有內點重新擬合，把旋轉誤差從 0.30° 降到 0.07°；平移方向則無論哪種都在半度以內。估計出的平移長度剛好是 1：OpenCV 只回傳它的方向，因為影像無法告訴我們真正的基線是 2。一旦從外部提供尺度，三角測量出的點與真實距離相差約 0.3% 以內。0.40 像素的重投影誤差低於 0.5 像素的雜訊水準，這在意料之中，因為這些點是對自己的觀測做擬合。

運動恢復結構

白話版。運動恢復結構（structure from motion, SfM）把兩視角的技巧一次用到幾百張照片上：它在所有照片之間找到相配的點，一台一台地放置相機，三角測量出各個點，並持續修整所有東西，讓每個三維點都投影到它真正被看到的像素上。

特徵、配對與幾何驗證

首先，SfM 必須知道不同照片中的哪些像素拍到同一個實際的點。它從局部特徵開始。每張影像產生帶有描述子的關鍵點，例如 SIFT [11]（IP 12），再在影像對之間配對描述子（IP 12，配對）。許多配對是錯的，所以每個影像對都要做**幾何驗證**：用 RANSAC [10] 擬合基礎矩陣（或本質矩陣、單應性），只有足夠多配對互相吻合時才保留該影像對 [2]。通過驗證的影像對組成**場景圖**（scene graph）：影像是節點，一條邊表示「這兩張影像看到相同的點」。

增量式重建

COLMAP 的論文描述了標準的增量式流程 [2]，它源自早期處理網路照片集的系統，例如 Photo Tourism [12]：

1. **初始化**：選一對共享許多配對、基線又夠寬的影像；估計它們的相對位姿並三角測量共享的點。
2. **註冊**新影像：解透視 n 點（Perspective-n-Point, PnP）問題，也就是在 RANSAC 之中，由它的關鍵點與已三角測量的三維點之間的配對求出相機位姿。
3. **三角測量**新影像與至少一張已註冊影像都看到的新點。
4. **光束法平差**（見下）同時修整所有相機與點，然後**過濾**重投影誤差大的點。
5. 回到步驟 2 重複，直到再也沒有影像可以註冊。

COLMAP 論文改進了其中每一步，包括如何挑選下一張影像，以及如何三角測量點。它最核心的教訓在步驟 4：註冊與三角測量必須與反覆的光束法平差、重新三角測量與離群值過濾交錯進行，因為沒有這些修整，小誤差會隨著影像一張張加入而累積，重建就會漂移 [2]。

光束法平差

白話版。光束法平差（bundle adjustment, BA）是最後的修整：把每台相機和每個三維點都稍微移動一點，直到所有「光束」都和點實際被看到的像素對齊。

BA 解的是下面這個非線性最小平方問題

$$\min_{\{P_c\},\{\mathbf{X}_k\}} \sum_{(c,k)\in\mathcal{O}} \rho\Big(\|\pi(P_c,\mathbf{X}_k)-\mathbf{x}_{ck}\|^2\Big),$$

其中 P_c 是相機 c 的參數（位姿，內參未知時也包含內參），$\mathbf{X}_k$ 是三維點，$\mathcal{O}$ 是觀測的集合（點 k 出現在影像 c 中），$\mathbf{x}_{ck}$ 是觀測到的像素，π 是針孔投影，ρ 是降低離群值權重的穩健損失 [2]、[13]。標準的求解器是 Levenberg–Marquardt，一種帶阻尼的 Gauss–Newton 法。它之所以快，是因為每個殘差只依賴一台相機和一個點，所以 Jacobian幾乎全是零 [2]、[13]。BA 無法固定整體的尺度與座標系，而且只會找到局部最小值，所以需要增量式步驟提供的好起點。

僅線上版：程式碼中的光束法平差 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/程式碼中的光束法平差/*

實務上的 COLMAP

COLMAP 是隨 [2] 與 [3] 釋出的開源 SfM 與 MVS 流程，有圖形介面與命令列介面 [4]。它輸出相機內參與位姿，加上稀疏點雲，這正是下面稠密方法的標準輸入：NeRF 論文中真實場景的位姿取自 COLMAP [5]，3DGS 則以 SfM 的點來初始化高斯 [6]。後面每個階段都繼承了這裡犯的誤差（見失效模式中的位姿誤差）。

多視角立體視覺

白話版。 SfM 給出相機與一組稀疏的點。多視角立體視覺（multi-view stereo, MVS）補上其餘部分：對參考影像的每個像素，沿著其他影像中的對極線搜尋，找出所有視角對表面長相意見一致的那個深度。

光度一致性與平面掃描

核心的檢驗是**光度一致性**（photo-consistency）：如果某像素的表面點深度為 Z，那麼它周圍的小區塊，在每一張看得到該點的影像裡看起來都應該一樣。一個深度假設的分數，由區塊之間的匹配代價來打分，例如差的平方和（sum of squared differences, SSD）或正規化互相關。

平面掃描（plane sweep）演算法 [14] 能有效率地檢驗許多深度。讓一個與參考影像平行的平面掃過一段深度範圍 $Z_1, \dots, Z_D$。對每個深度，把每張來源影像扭曲（warp）到參考視角，就像整個場景都落在那個平面上一樣；單一平面的扭曲是一個單應性。逐像素計算代價，把代價疊成大小為 $D \times H \times W$ 的**代價體**（cost volume），再為每個像素選出代價最低的深度：

$$\hat{Z}(u,v) = \arg\min_{k} \sum_{(u',v') \in \mathcal{W}(u,v)} \left(I_{\text{ref}}(u',v') - \tilde{I}^{(k)}_{\text{src}}(u',v') \right)^2,$$

其中 $\tilde{I}^{(k)}_{\text{src}}$ 是被深度 Z_k 處的平面扭曲後的來源影像，$\mathcal{W}(u,v)$ 是像素周圍的小視窗。對兩台平行的相機（一對校正過的相機），平面單應性簡化成依視差 fB/Z_k 做水平位移，程式碼也就能寫得很短。

僅線上版：平面誘導的單應性 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/平面誘導的單應性/*

```
import numpy as np
from scipy import ndimage as ndi

rng = np.random.default_rng(0)
H, W, f, B = 120, 160, 150.0, 0.5                    # image size, focal length (px),
 ↪  baseline
tex = ndi.gaussian_filter(rng.random((400, 400)), 1.5)   # random texture painted on the
 ↪  world
tex = (tex - tex.min()) / np.ptp(tex)

def render(cx):
    """Ray-cast a camera at (cx, 0, 0) looking down +Z: a square at Z=5 in front of a wall at
     ↪  Z=8."""
    v, u = np.mgrid[0:H, 0:W].astype(float)
    dx, dy = (u - W / 2) / f, (v - H / 2) / f          # ray direction (dx, dy, 1)
    Xs, Ys = cx + 5 * dx, 5 * dy                        # hit point on the plane Z = 5
    front = (np.abs(Xs) < 1) & (np.abs(Ys) < 1)         # inside the square?
    Z = np.where(front, 5.0, 8.0)
    X, Y = cx + Z * dx, Z * dy
    img = ndi.map_coordinates(tex, [200 + 40 * Y + 100 * front, 200 + 40 * X], order=1)
    img[~front & (X > 1.5) & (Y < -0.5)] = 0.5          # a textureless patch on the wall
    return img, Z

ref, Z_true = render(0.0)                               # reference camera
src, _ = render(B)                                      # source camera, B to the right

depths = 1 / np.linspace(1 / 10, 1 / 3.5, 80)           # depth hypotheses, uniform in 1/Z
cost = np.empty((len(depths), H, W))
for k, Z in enumerate(depths):
    d = f * B / Z                                       # the plane Z induces a horizontal
     ↪  shift d
```

```
    warped = ndi.shift(src, (0, d), order=1, mode='nearest')   # src pixel u - d lands on u
    cost[k] = ndi.uniform_filter((ref - warped) ** 2, 7)       # windowed SSD
     ↪  photo-consistency
Z_est = depths[cost.argmin(0)]                          # winner-take-all depth per pixel

ok = np.abs(Z_est - Z_true) / Z_true < 0.05
flat = (Z_true == 8) & (ref == 0.5)
inner = np.zeros_like(ok); inner[8:-8, 20:-8] = True    # ignore borders the source camera
 ↪  cannot see
print(f"pixels within 5% of the true depth: {100 * ok[inner & ~flat].mean():.1f}% (textured),
 ↪  "
      f"{100 * ok[inner & flat].mean():.1f}% (textureless patch)")
```

```
pixels within 5% of the true depth: 95.3% (textured), 14.8% (textureless patch)
```

在有紋理的表面上，這次合成實驗中 95.3% 被評估的像素，其深度落在真值的 5% 以內。在無紋理的區塊裡只有 14.8%，因為對一塊平坦的灰色區塊來說，每個深度假設的匹配程度都一樣好。圖 4 (d) 直接顯示了這點：有紋理像素的代價曲線在真實深度處有一個尖銳的最小值；無紋理像素的曲線則是平的。有紋理表面上剩下的誤差出現在左邊界（只有一台相機看得到），以及方塊旁邊，那裡有一部分背景被第二台相機擋住（遮擋）。

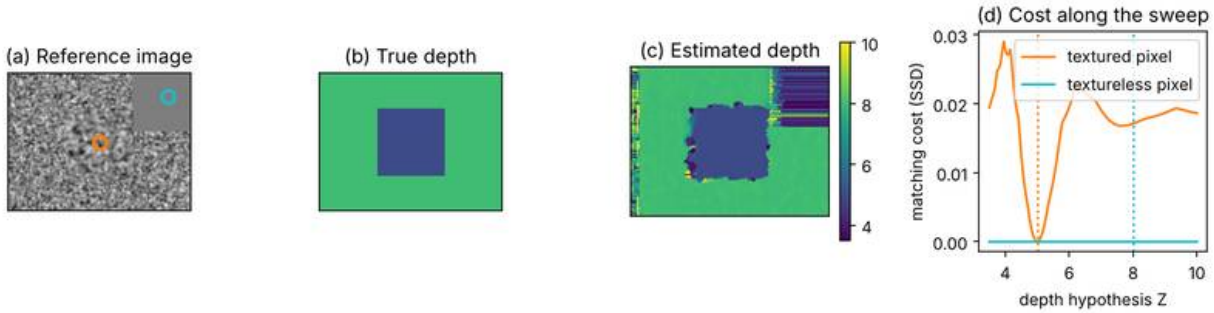


圖 *4* — 合成校正影像對上的平面掃描立體視覺。(*a*) 含無紋理區塊（右上）的參考影像。(*b*) 真實深度。(*c*) 以 7×7 的 *SSD* 視窗得到的贏者全拿（*winner-take-all*）深度。(*d*) 兩個圈起像素的代價曲線；點線標示真實深度。

從深度圖到單一模型

每張影像一張深度圖，還不是一個模型。**深度圖融合**（depth-map fusion）把它們合併起來。Curless 與 Levoy 的體積法 [15] 在體素格裡存放每張深度圖所量到的、到表面的有號距離的滾動加權平均：

$$D_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x}) D_i(\mathbf{x}) + w_{i+1}(\mathbf{x})\, d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})}, \qquad W_{i+1}(\mathbf{x}) = W_i(\mathbf{x}) + w_{i+1}(\mathbf{x}),$$

其中 $d_{i+1}(\mathbf{x})$ 是體素 $\mathbf{x}$ 沿視線方向到深度圖 $i+1$ 所見表面的有號距離，w_{i+1} 是它的信心權重，D_i、W_i 是累積了 i 張圖之後的距離與權重。表面是 D 的零等值面，由 marching cubes（見下）

萃取。實務上，距離在表面附近會被截斷，所以這個方法叫做 **TSDF**（truncated signed distance function，截斷有號距離函數）融合。COLMAP 的 MVS [3] 則是逐像素估計深度與法向量，使用逐像素的視角選擇與多視角幾何一致性檢查，再把一致的深度融合成稠密點雲。

學習式 MVS

白話版。 MVS 中手工設計的部分，也就是區塊代價，以及清理代價體的規則，都可以從資料中學出來。

MVSNet [16] 保留平面掃描的結構，但讓每一步都可微分。共用的 CNN 從所有影像擷取特徵，每個來源視角的特徵被扭曲到參考相機的正面平行平面上（可微分的單應性扭曲）。**以變異數為基礎的代價**把任意數量的視角合併成一個代價體：變異數低代表各視角意見一致。接著用三維 CNN 對代價體做正則化，沿深度軸做 softmax 得到每個深度的機率，在這個分布下的期望深度（「soft argmin」）是連續的輸出，可以用迴歸損失來訓練 [16]。MVSNet 在 DTU [37] 上訓練，並在不微調的情況下於 Tanks and Temples [38] 上評估 [16]。作者提出學習的動機，是低紋理、鏡面與反光的區域，手工設計的代價在那裡會失效 [16]。

神經輻射場（NeRF）

白話版。 NeRF 把場景存成一個小小的神經網路，這個網路對空間中任何一點、任何觀看方向回答同一個問題：「這裡有多少東西，它朝我這個方向送出什麼顏色？」要產生一張影像，就對每個像素射出一條光線，在光線上許多點詢問網路，再把答案混合起來。訓練就是調整網路，直到算繪出來的影像和照片相符。

以函數表示場景

NeRF [5] 是一個多層感知器（multilayer perceptron）$F_\Theta : (\mathbf{x}, \mathbf{d}) \mapsto (\sigma, \mathbf{c})$，其中 $\mathbf{x} \in \mathbb{R}^3$ 是位置，$\mathbf{d}$ 是觀看方向，$\sigma \geq 0$ 是 $\mathbf{x}$ 處的**體積密度**（volume density），$\mathbf{c}$ 是朝 $\mathbf{d}$ 方向發出的 RGB 顏色。網路的設計讓密度只依賴位置，顏色則同時依賴方向，這讓 NeRF 能重現隨視角改變的亮面反光 [5]。一個網路只表示一個場景；沒有包含許多場景的訓練集。

僅線上版：*NeRF*（*Mildenhall* 等人，*2020*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/papers/nerfmildenhall-等人 2020/*

沿光線的體積算繪

白話版。想像光線穿過一團濃淡不一的霧。每一小段霧都會用自己的顏色發光，同時擋住它後方的光。像素的顏色就是所有發光加起來，每一段發光都被它前方的霧減弱。

相機光線為 $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$，其中原點 $\mathbf{o}$ 是相機中心，方向是 $\mathbf{d}$，距離 t 介於近界 t_n 與遠界 t_f 之間。依照經典的體積算繪 [17]，期望顏色為

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\,\sigma(\mathbf{r}(t))\,\mathbf{c}(\mathbf{r}(t), \mathbf{d})\,dt, \qquad T(t) = \exp\!\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\,ds\right),$$

其中 $T(t)$ 是**透射率**（transmittance），即光線從 t_n 走到 t 沒有被擋住的機率 [5]。NeRF 用光線上的 N 個取樣 $t_1 < \ldots < t_N$ 來計算它：

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} w_i\,\mathbf{c}_i, \qquad w_i = T_i\,\alpha_i, \qquad \alpha_i = 1 - e^{-\sigma_i \delta_i}, \qquad T_i = \prod_{j=1}^{i-1}(1-\alpha_j),$$

其中 $\sigma_i, \mathbf{c}_i$ 是網路在第 i 個取樣處的輸出，$\delta_i = t_{i+1} - t_i$ 是間距，α_i 是第 i 段的**不透明度**（opacity），T_i 是通過它前面各段之後還剩下的光，w_i 是第 i 個取樣在像素中的權重。這正是半透明圖層由前到後的 **alpha 合成**（alpha compositing）[5]。權重的總和 $\sum_i w_i$ 是像素的總不透明度，$\sum_i w_i t_i$ 則是深度的估計。每一步都可微分，所以照片誤差可以一路傳回網路的權重。

下面的程式碼用精確解來檢查這個數值積分。光線先穿過藍色的霧，再穿過一層紅色表面，最後抵達白色背景；對分段常數的密度，這些積分有封閉解。

```
import numpy as np

# One ray from t = 0 to t = 6: blue fog on [1, 2] (density 0.5), a red surface on [3, 3.5]
 ↪  (density 4),
# white background behind everything.
slabs = [(1.0, 2.0, 0.5, np.array([0.2, 0.4, 1.0])),
         (3.0, 3.5, 4.0, np.array([1.0, 0.1, 0.1]))]
bg = np.ones(3)

def field(t):
    """Density sigma(t) and colour c(t) at sample positions t."""
    sigma, c = np.zeros_like(t), np.zeros(t.shape + (3,))
    for a, b, s, col in slabs:
        inside = (t >= a) & (t < b)
        sigma[inside], c[inside] = s, col
    return sigma, c

def render(n):
    """NeRF quadrature: alpha_i = 1 - exp(-sigma_i delta_i), T_i = prod_{j<i} (1 - alpha_j),
     ↪  w_i = T_i alpha_i."""
    edges = np.linspace(0, 6, n + 1)
    t, delta = (edges[:-1] + edges[1:]) / 2, np.diff(edges)
    sigma, c = field(t)
    alpha = 1 - np.exp(-sigma * delta)
    T = np.concatenate([[1.0], np.cumprod(1 - alpha)[:-1]])
    w = T * alpha
    T_end = T[-1] * (1 - alpha[-1])                    # light that reaches the background
    return w @ c + T_end * bg, w.sum(), w @ t

# Analytic answer for piecewise-constant density (exact integrals of the volume rendering
 ↪  equation)
C, acc, depth, T0 = np.zeros(3), 0.0, 0.0, 1.0
for a, b, s, col in slabs:
    L = b - a
    absorbed = T0 * (1 - np.exp(-s * L))
    C, acc = C + absorbed * col, acc + absorbed
    depth += T0 * (a * (1 - np.exp(-s * L)) + (1 - np.exp(-s * L)) / s - L * np.exp(-s * L))
    T0 *= np.exp(-s * L)
C += T0 * bg
```

```
print(f"analytic : colour {np.round(C, 4)}, opacity {acc:.4f}, depth {depth:.4f}")
for n in [16, 64, 256, 1024]:
    Cq, accq, dq = render(n)
    print(f"N = {n:4d} : colour {np.round(Cq, 4)}, opacity {accq:.4f}, depth {dq:.4f}")
```

```
analytic : colour [0.6852 0.2919 0.528 ], opacity 0.9179, depth 2.2373
N =   16 : colour [0.7498 0.3318 0.5195], opacity 0.8466, depth 2.1655
N =   64 : colour [0.7006 0.2986 0.5232], opacity 0.9040, depth 2.2263
N =  256 : colour [0.689  0.2934 0.5267], opacity 0.9146, depth 2.2348
N = 1024 : colour [0.6862 0.2923 0.5277], opacity 0.9171, depth 2.2367
```

隨著取樣數增加，數值積分會收斂到解析解的顏色、不透明度與深度。取 16 個取樣時差距明顯，因為薄薄的表面層落進了一兩個太粗的區間。這就是 NeRF 不均勻取樣的原因：先由一個每條光線 64 個取樣的粗網路找出權重大的位置，再由細網路在那裡加上集中的 128 個取樣（階層式取樣）[5]。再看看深度：2.24（圖 5 用 256 個取樣時是 2.23），落在霧與表面之間，那裡其實沒有任何表面。半透明的「霧」出現在錯誤的位置，就是失效模式中所討論的漂浮物瑕疵的來源。圖 5 畫出這條光線的密度、透射率與權重。

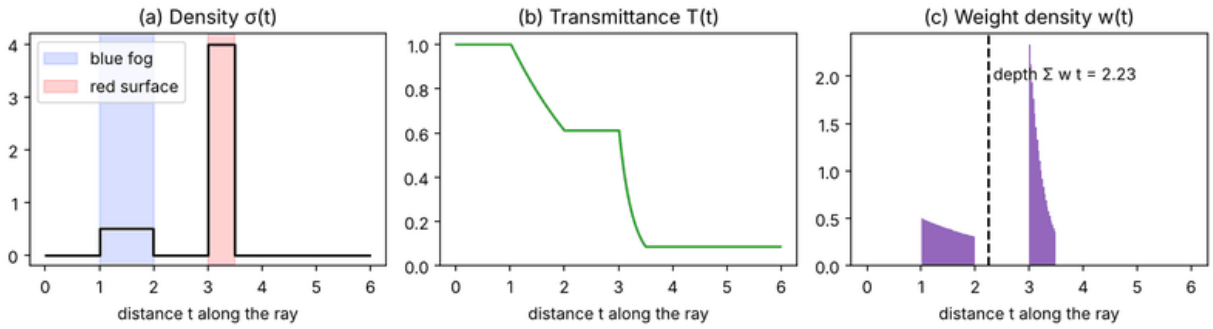


圖 5 — 程式碼中那條光線的體積算繪：(*a*) 密度、(*b*) 透射率、(*c*) 每單位長度的權重。期望深度（虛線）落在霧與表面之間，那裡什麼都沒有。

位置編碼

白話版。一個直接輸入原始 (x, y, z) 的普通 MLP，只能學到平滑而模糊的函數。改為輸入座標在許多頻率下的正弦與餘弦，它才能表示銳利的邊緣與細緻的紋理。

NeRF 把每個座標 p（正規化到 $[-1, 1]$）映射成

$$\gamma(p) = \big(\sin(2^0\pi p), \cos(2^0\pi p), \dots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p)\big),$$

其中 L 是頻帶數量：位置用 $L = 10$，方向用 $L = 4$ [5]。沒有這個編碼，論文的結果會失去細節 [5]。Tancik 等人 [18] 解釋了原因：座標 MLP 偏向低頻（頻譜偏差，spectral bias），而對輸入做傅立葉特徵映射就能消除這個偏差。這些正弦函數類似 Transformer 的位置資訊（ML 05），但這裡編碼的是連續座標，而不是 token 的位置。

訓練，以及為什麼慢

訓練只需要有位姿的影像，其他都不用。每一步從訓練影像中取樣一批像素（光線），用粗網路與細網路把它們算繪出來，並最小化顏色的平方誤差 [5]。問題出在成本。每個像素需要幾百次網路評估，一個場景需要幾十萬步。論文報告每個場景 100–300k 次迭代，在一張 NVIDIA V100 GPU 上約 1–2 天 [5]。算繪慢也是同樣的原因：論文每條光線使用 256 次網路查詢，每張影像介於 1.5 億到 2 億次之間，並報告在 V100 上每個畫面約 30 秒 [5]。

加速：雜湊網格

白話版。不要叫一個大網路記住所有東西，而是把學到的特徵存在多個解析度的查找表裡，再用一個很小的網路來解碼。查表比跑許多層網路便宜得多。

Instant-NGP [19] 以**多解析度雜湊編碼**（multiresolution hash encoding）取代頻率編碼。空間由 L 層網格覆蓋，從粗到細。每一層網格在它的角點存放可訓練的特徵向量，但是放在大小固定為 T 的表裡；在細的網格上，角點遠多於表中的項目，就由一個雜湊函數決定角點使用哪一項，所以好幾個角點共用同一項。要編碼一個點時，在每一層內插它周圍角點的特徵，串接起來，再交給一個小型 MLP。論文並不明確地解決共用項目的問題；它仰賴粗的層級和 MLP 自己把它們分開 [19]。作者報告在「幾秒內」就能訓練出高品質的結果，並在一張 GPU 上以 1920×1080 解析度做到幾十毫秒的算繪 [19]。這類以網格為基礎的編碼，現在在快速的 NeRF 變體中很常見。

僅線上版：多解析度雜湊編碼 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/多解析度雜湊編碼/*

神經隱式曲面：NeuS

白話版。NeRF 的密度不一定會形成乾淨的表面：它可以是一團雲。如果你想要曲面，就讓網路改為輸出有號距離，並用一種讓算繪出的權重剛好在表面上達到峰值的方式，把距離轉成不透明度。

NeuS [22] 把場景表示成一個 SDF 網路 $f(\mathbf{x})$，它的零等值面就是曲面。它透過 sigmoid $\Phi_s(x) = (1 + e^{-sx})^{-1}$ 把 SDF 轉成不透明度，其中 s 是會在訓練中增大的、學出來的銳利度。作者指出，最直覺的做法是直接把 logistic 密度 $\phi_s(f) = \Phi_s'(f)$ 當成體積密度，這樣得到的權重會在表面之前達到峰值（偏差），他們於是推導出一個一階無偏、又能處理遮擋的不透明度 [22]。離散形式為

$$\alpha_i = \max\left(\frac{\Phi_s(f(\mathbf{p}_i)) - \Phi_s(f(\mathbf{p}_{i+1}))}{\Phi_s(f(\mathbf{p}_i))},\ 0 \right),$$

其中 $\mathbf{p}_i$ 是光線上第 i 個取樣；權重之後使用與 NeRF 相同的合成公式。**Eikonal** 項 $(\|\nabla f\| - 1)^2$ 讓 f 保持為合法的距離函數 [22]。因為曲面是等值面，用 marching cubes 就能直接得到網格。NeuS 不需要前景遮罩就能重建物體，並且在 DTU [37] 等資料上做過評估 [22]。

僅線上版：為什麼 *NeuS* 的權重會在表面上達到峰值 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/為什麼-neus-的權重會在表面上達到峰值/*

3D Gaussian Splatting

白話版。 3DGS 不用你得沿著每條光線去查詢的網路，而是用數百萬個小小的、有顏色的、半透明的橢球來表示場景，像一團柔軟的顏料滴。要產生影像時，把每一滴投影到螢幕上成為一個柔軟的橢圓，再由前到後疊起來。沒有光線步進，也沒有網路：只有排序與混合，GPU 做起來非常快。

高斯基元

每個色塊都是一個三維的鐘形曲線：中心最濃，向外平滑淡出，而且在某些方向被拉得比其他方向更長。精確地說，每個基元 [6] 有中心 $\boldsymbol{\mu} \in \mathbb{R}^3$、決定大小與方向的共變異數 Σ、不透明度 $o \in [0,1]$，以及以球諧函數（spherical harmonics）係數儲存的顏色（球面上一組平滑的函數，讓顏色可以隨觀看方向改變）。它在空間中的涵蓋範圍為

$$G(\mathbf{x}) = \exp\!\Big(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{\top}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big), \qquad \Sigma = R\,S\,S^{\top}R^{\top},$$

其中 $S = \operatorname{diag}(s_1, s_2, s_3)$ 存放三個軸長，R 是以單位四元數儲存的旋轉 [6]。這個分解保證 Σ 在梯度下降期間一直是合法（半正定）的共變異數，直接最佳化 Σ 的六個元素則做不到這一點 [6]。

投影到影像

白話版。透過相機看到的三維橢球，看起來是二維的橢圓。3DGS 用相機投影的局部線性近似來算出這個橢圓，對小的高斯來說很準確。

依照 EWA splatting [23]，投影後高斯的二維共變異數為

$$\Sigma' = J\,W\,\Sigma\,W^{\top}J^{\top},$$

其中 W 是世界到相機轉換的旋轉部分，J 是在高斯中心處計算的透視投影 Jacobian，只取 Σ' 左上角 2×2 的區塊，就得到影像空間中的橢圓 [6]、[23]。投影後的中心就是 $\boldsymbol{\mu}$ 的針孔投影。

僅線上版：投影 *3D* 高斯：局部仿射近似 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/notes/投影-3d-高斯局部仿射近似/*

Alpha 合成與即時算繪

像素顏色使用和 NeRF 相同的合成規則，只是對象換成覆蓋該像素、並依深度排序的高斯：

$$C = \sum_i \mathbf{c}_i\,\alpha_i \prod_{j=1}^{i-1}(1-\alpha_j), \qquad \alpha_i = o_i \exp\!\Big(-\tfrac{1}{2}(\mathbf{u}-\boldsymbol{\mu}'_i)^{\top}\Sigma'^{-1}_i(\mathbf{u}-\boldsymbol{\mu}'_i)\Big),$$

其中 $\mathbf{u}$ 是像素位置，$\boldsymbol{\mu}'_i$ 與 Σ'_i 是高斯 i 投影後的中心與共變異數，$\mathbf{c}_i$ 是它隨視角變化的顏色。為什麼快？算繪器 [6] 把螢幕切成 16×16 像素的圖塊（tile），把每個可見的高斯指派給它碰到的圖塊，用 GPU 基數排序（radix sort）把所有（圖塊, 深度）鍵值一次排好，然後平行地由前到後混合每個圖塊的清單，像素幾乎不透明時就停止。沒有網路要評估，也沒有空白空間要取樣。論文

報告在 1080p 達到即時（至少每秒 30 個畫面）的新視角合成，品質可與當時最好的 NeRF 變體相比 [6]。

僅線上版：*3D Gaussian Splatting*（*Kerbl* 等人，*2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/papers/3d-gaussian-splattingkerbl-等人 2023/*

最佳化與自適應密度控制

白話版。訓練從稀疏的 SfM 點開始，每個點一個高斯，然後讓這組高斯長出來並修剪：缺少細節的地方就增加高斯，一個大高斯涵蓋太多的地方就把它分裂，看不見的高斯則移除。

所有參數都用梯度下降來最佳化，損失為算繪影像與真實影像之間的 $(1-\lambda)\mathcal{L}_1 + \lambda\mathcal{L}_{\text{D-SSIM}}$，其中 $\lambda = 0.2$ [6]。每 100 次迭代，這個方法會對視空間（view-space）位置梯度大的高斯進行加密，梯度大表示該區域還沒有被充分解釋 [6]：

- 在重建不足區域中的小高斯會被**複製**（clone；複製並沿梯度方向移動）；
- 在高變異區域中的大高斯會被**分裂**（split）成兩個較小的高斯，尺度除以 1.6；
- 不透明度低於門檻的高斯會被**修剪**（prune）；此外，每 3000 次迭代會把所有不透明度重設到接近零，這限制了靠近輸入相機的漂浮物所造成的高斯增長 [6]。

程式碼：二維 splatting 的小實驗

這段程式碼用 Adam 最佳化器 [49]，把 40 個二維高斯擬合到一張 48×48 的影像，使用的正是上面的合成規則（深度順序固定，也沒有加密）。反向傳遞用鏈鎖律（chain rule）手寫出來（3DGS 論文也是明確推導梯度，而不是依賴自動微分 [6]）；梯度在撰寫本篇時已與有限差分核對過。

```
import numpy as np
from skimage.draw import disk, polygon, ellipse

# Target: a small synthetic RGB image (48 x 48) on a black background
H = W = 48
target = np.zeros((H, W, 3))
target[disk((16, 16), 9)] = [0.9, 0.2, 0.2]
target[ellipse(30, 30, 6, 14, rotation=np.deg2rad(30))] = [0.2, 0.4, 0.9]
target[polygon([40, 44, 34], [6, 22, 14])] = [0.2, 0.8, 0.3]
ys, xs = np.mgrid[0:H, 0:W].astype(float)

def sig(z):
    return 1 / (1 + np.exp(-z))

def render(p, target=None):
    """Front-to-back alpha compositing of K 2D Gaussians (index order = depth order).
    p: mu (K,2), log_s (K,2), theta (K,), op (K,) opacity logits, col (K,3) colour logits."""
    c, s, th = sig(p['col']), np.exp(p['log_s']), p['theta']
    dx = xs[None] - p['mu'][:, 0, None, None]; dy = ys[None] - p['mu'][:, 1, None, None]
    ct, st = np.cos(th)[:, None, None], np.sin(th)[:, None, None]
    u1, u2 = ct * dx + st * dy, -st * dx + ct * dy                    # offsets in the
     ↪  Gaussian's own axes
    sx2, sy2 = (s[:, 0] ** 2)[:, None, None], (s[:, 1] ** 2)[:, None, None]
```

```
    g = np.exp(-0.5 * (u1 ** 2 / sx2 + u2 ** 2 / sy2))              # Gaussian footprint
    o = sig(p['op'])[:, None, None]
    a = np.minimum(o * g, 0.99)                                      # alpha of each
     ↪  Gaussian at each pixel
    T = np.cumprod(np.concatenate([np.ones((1, H, W)), 1 - a[:-1]]), axis=0)   #
     ↪  transmittance in front
    w = a * T                                                        # compositing weights
    img = np.einsum('khw,kc->hwc', w, c)                            # black background adds
     ↪  nothing
    if target is None:
        return img
    # backward pass for the loss L = mean squared error (the chain rule written out by hand)
    G = 2 * (img - target) / target.size                             # dL/dimg, shape (H,
     ↪  W, 3)
    contrib = w[..., None] * c[:, None, None, :]                    # (K, H, W, 3)
    behind = np.cumsum(contrib[::-1], axis=0)[::-1] - contrib        # colour from
     ↪  Gaussians behind k
    dC_da = T[..., None] * c[:, None, None, :] - behind / (1 - a[..., None])
    da = np.einsum('khwc,hwc->khw', dC_da, G) * (o * g < 0.99)
    dq = da * o * g * -0.5                                           # through g = exp(-q /
     ↪  2)
    grads = {
        'col': np.einsum('khw,hwc->kc', w, G) * c * (1 - c),
        'op': (da * g).sum((1, 2)) * sig(p['op']) * (1 - sig(p['op'])),
        'log_s': np.stack([(dq * -2 * u1 ** 2 / sx2).sum((1, 2)), (dq * -2 * u2 ** 2 /
         ↪  sy2).sum((1, 2))], 1),
        'theta': (dq * 2 * u1 * u2 * (1 / sx2 - 1 / sy2)).sum((1, 2)),
        'mu': -np.stack([(dq * (2 * u1 * ct / sx2 - 2 * u2 * st / sy2)).sum((1, 2)),
                         (dq * (2 * u1 * st / sx2 + 2 * u2 * ct / sy2)).sum((1, 2))], 1),
    }
    return img, grads

rng = np.random.default_rng(0)
K = 40
p = {'mu': rng.uniform(4, 44, (K, 2)), 'log_s': np.full((K, 2), np.log(3.0)), 'theta':
 ↪  rng.uniform(0, np.pi, K),
     'op': np.zeros(K), 'col': np.zeros((K, 3))}
lr = {'mu': 0.5, 'log_s': 0.05, 'theta': 0.05, 'op': 0.1, 'col': 0.1}
m = {k: np.zeros_like(v) for k, v in p.items()}; v2 = {k: np.zeros_like(v) for k, v in
 ↪  p.items()}

def psnr(img):
    return 10 * np.log10(1 / np.mean((img - target) ** 2))

for it in range(1, 501):
    img, gr = render(p, target)
    for k in p:                                                      # Adam update
        m[k] = 0.9 * m[k] + 0.1 * gr[k]; v2[k] = 0.999 * v2[k] + 0.001 * gr[k] ** 2
        p[k] -= lr[k] * (m[k] / (1 - 0.9 ** it)) / (np.sqrt(v2[k] / (1 - 0.999 ** it)) +
         ↪  1e-8)
    if it in (1, 50, 100, 200, 500):
        print(f"iteration {it:3d}: PSNR {psnr(img):.2f} dB")
```

```
iteration   1: PSNR 11.32 dB
```

```
iteration  50: PSNR 20.88 dB
iteration 100: PSNR 21.85 dB
iteration 200: PSNR 23.12 dB
iteration 500: PSNR 23.57 dB
```

PSNR（定義見 DLCV 09）上升很快，然後在這次實驗中於 23.6 dB 附近持平。這個高原正是這個實驗的重點：只有固定數量的柔軟色塊時，尖銳的角落和三角形的細尖端都擬合不出來，而一開始就放在錯誤位置的色塊也會留在原地。這正是自適應密度控制要修正的問題，練習 4 請你把它加進去。圖 6 顯示擬合結果與最終的橢圓。

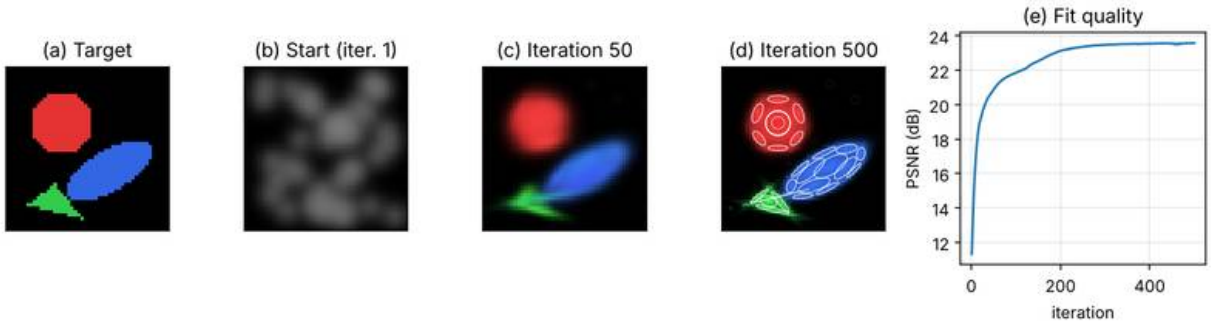


圖 *6* — 用 *40* 個高斯與 *alpha* 合成做的 *splatting* 小實驗：(*a*) 目標、(*b*) 起點、(*c*) *50* 次與 (*d*) *500* 次迭代後，並畫出每個高斯一個標準差的橢圓；(*e*) *PSNR* 對迭代次數。數字只來自這次小實驗。

取得網格

白話版。圖學引擎、物理模擬器與 3D 印表機要的是三角形。上面每一種表示法都需要一個轉換步驟才能得到三角形，而轉換本身也會帶來誤差。

Marching cubes

Marching cubes [24] 把取樣過的純量場（SDF、佔據或融合後的 TSDF）轉成某個等值面的三角形網格。它逐一走訪由八個相鄰格點取樣組成的每個立方體，看哪些角點在內部、哪些在外部，在符號改變的立方體邊上放置頂點（用場值的線性內插），再用查找表中的三角形把它們連起來。scikit-image 提供了實作 [46]。這段程式碼在幾種格網解析度下，萃取精確環面 SDF 的零等值面，並用下面評估一節定義的幾何指標來量測結果。

```
import numpy as np
from scipy.spatial import cKDTree
from skimage.measure import marching_cubes

R, r = 0.6, 0.25                                    # torus: ring radius R, tube radius r

def sdf_torus(x, y, z):
    """Exact signed distance to the torus (negative inside)."""
```

```
    return np.hypot(np.hypot(x, y) - R, z) - r

def sample_mesh(verts, faces, n, rng):
    """n points spread uniformly over the area of a triangle mesh."""
    tri = verts[faces]
    area = 0.5 * np.linalg.norm(np.cross(tri[:, 1] - tri[:, 0], tri[:, 2] - tri[:, 0]),
     ↪  axis=1)
    k = rng.choice(len(faces), n, p=area / area.sum())
    a, b = rng.random((2, n, 1))
    flip = a + b > 1; a[flip], b[flip] = 1 - a[flip], 1 - b[flip]
    return tri[k, 0] + a * (tri[k, 1] - tri[k, 0]) + b * (tri[k, 2] - tri[k, 0])

def sample_torus(n, rng):
    """n points uniform over the true torus surface (rejection sampling on the area
     ↪  element)."""
    u, v = rng.uniform(0, 2 * np.pi, (2, 3 * n))
    ok = rng.random(3 * n) * (R + r) < R + r * np.cos(v)   # keep with probability
     ↪  proportional to R + r cos v
    u, v = u[ok][:n], v[ok][:n]
    return np.c_[(R + r * np.cos(v)) * np.cos(u), (R + r * np.cos(v)) * np.sin(u), r *
     ↪  np.sin(v)]

def geometry_metrics(P, G, tau):
    """Accuracy, completeness, Chamfer (their mean) and F-score at threshold tau."""
    d_pg = cKDTree(G).query(P)[0]                      # each reconstructed point -> nearest
     ↪  true point
    d_gp = cKDTree(P).query(G)[0]                      # each true point -> nearest
     ↪  reconstructed point
    prec, rec = (d_pg < tau).mean(), (d_gp < tau).mean()
    return d_pg.mean(), d_gp.mean(), (d_pg.mean() + d_gp.mean()) / 2, 2 * prec * rec / (prec
     ↪  + rec)

rng = np.random.default_rng(0)
G = sample_torus(100_000, rng)
acc, comp, cd, fs = geometry_metrics(sample_torus(100_000, rng), G, tau=0.01)
print(f"true surface vs itself: Chamfer {cd:.4f} | F-score@0.01 {fs:.3f}   (the floor set by
 ↪  sampling)")
for n in [12, 24, 48, 96]:
    ax = np.linspace(-1, 1, n)
    vol = sdf_torus(*np.meshgrid(ax, ax, ax, indexing='ij'))
    verts, faces, _, _ = marching_cubes(vol, level=0.0, spacing=(ax[1] - ax[0],) * 3)
    verts -= 1.0                                       # grid index space -> world
     ↪  coordinates [-1, 1]
    P = sample_mesh(verts, faces, 100_000, rng)
    acc, comp, cd, fs = geometry_metrics(P, G, tau=0.01)
    print(f"grid {n:2d}^3: {len(faces):6d} triangles | acc {acc:.4f} comp {comp:.4f} "
          f"Chamfer {cd:.4f} | F-score@0.01 {fs:.3f} | max |SDF| on vertices
           ↪  {np.abs(sdf_torus(*verts.T)).max():.4f}")
```

```
true surface vs itself: Chamfer 0.0039 | F-score@0.01 0.995   (the floor set by sampling)
grid 12^3:    448 triangles | acc 0.0154 comp 0.0154 Chamfer 0.0154 | F-score@0.01 0.299 |
 ↪  max |SDF| on vertices 0.0084
grid 24^3:   2272 triangles | acc 0.0052 comp 0.0052 Chamfer 0.0052 | F-score@0.01 0.988 |
 ↪  max |SDF| on vertices 0.0032
```

```
grid 48^3:   9568 triangles | acc 0.0040 comp 0.0039 Chamfer 0.0040 | F-score@0.01 0.995 |
 ↪  max |SDF| on vertices 0.0008
grid 96^3:  38832 triangles | acc 0.0039 comp 0.0039 Chamfer 0.0039 | F-score@0.01 0.995 |
 ↪  max |SDF| on vertices 0.0002
```

這次實驗帶來兩個教訓。第一，即使在 12^3 的格網上，頂點也幾乎剛好落在曲面上（最大的 |SDF| 是 0.0084），但 Chamfer 距離卻是 0.0154：頂點之間的平坦三角形橫切過彎曲的管面。第二，指標在約 0.0039 就不再改善，因為把同一個真實曲面的兩次獨立隨機取樣互相比較，本來就是這個值。量到的誤差只有相對於這個底線才有意義，而這個底線取決於取樣了多少個點。圖 7 顯示網格與曲線。

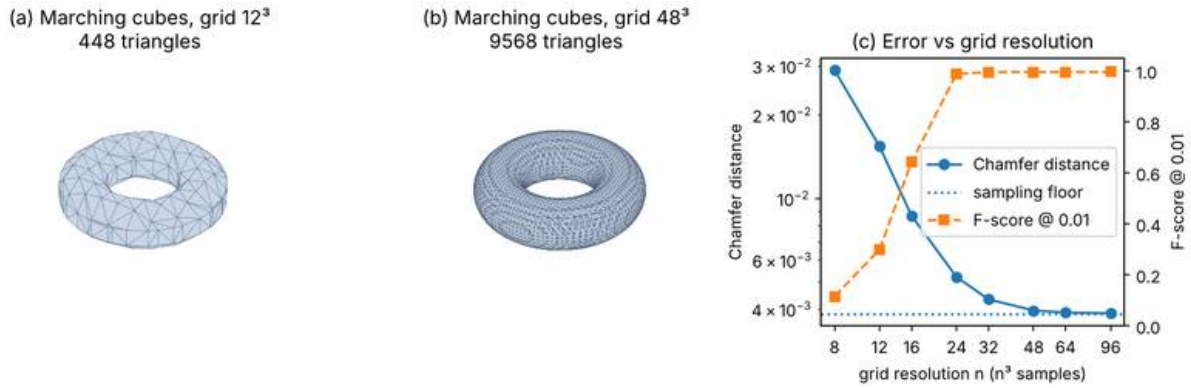


圖 *7* — 在精確環面 *SDF* 上的 *marching cubes*：(*a*) 12^3 與 (*b*) 48^3 格網；(*c*) *Chamfer* 距離與門檻 $\tau = 0.01$ 的 *F-score* 對格網解析度，並附上比較真實曲面兩次取樣所得的底線。

Poisson 表面重建

當輸入是帶有法向量的點雲（例如來自 MVS）時，Poisson 表面重建 [25] 是標準工具。它尋找一個指示函數（indicator function）χ（內部為 1、外部為 0），使它的梯度與法向量相符，這就導出 Poisson 方程式 $\Delta\chi = \nabla \cdot \mathbf{V}$，其中 Δ 是 Laplacian，$\mathbf{V}$ 是由法向量建立的向量場；曲面是 χ 的一個等值面 [25]。因為它一次用上所有的點，所以對雜訊很穩健 [25]；屏蔽（screened）變體 [26] 還會把曲面拉向這些點。結果是不漏水的曲面，這很方便，但也表示資料中的破洞會被捏造出來的曲面補起來。

從高斯得到網格

3D 高斯並不定義曲面：最佳化之後，它們雜亂、互相重疊，而且常常和真實幾何對不齊。有兩條研究路線處理這個問題。**SuGaR** [27] 加入一個正則化項，鼓勵高斯平貼在曲面上，然後從高斯密度等值面上取樣的點，用 Poisson 重建萃取出網格，也可以選擇性地把新的高斯綁定到網格上，並一起精修兩者 [27]。**2D Gaussian Splatting** [28] 則改變基元：每個高斯是一個平坦、有方向的二維圓盤（曲面元素），用精確的光線–圓盤相交來算繪，再加上深度失真與法向量一致性損失。網格接著用 TSDF 融合把算繪出的深度圖融合得到 [28]，與上面 Curless 和 Levoy 的方法是同一個想法。

前饋式重建

白話版。到目前為止的方法，每次都只針對一個場景做最佳化。最新的模型只訓練一次，使用大量三維資料，之後給它新的照片，它就在一次傳遞中輸出相機以及每個像素對應的稠密三維點，不需要配對、RANSAC 或光束法平差。

DUSt3R [29] 輸入兩張內參與位姿都未知的影像，迴歸出兩張**點圖**（pointmap）：對每張影像的每個像素，給出它所看到的三維點，兩者都表示在第一台相機的座標系中 [29]。點圖是一張有三個通道 (X, Y, Z) 的影像（圖 2 (h)）。由它可以推得深度、焦距、相對位姿與像素對應。這個網路是 Transformer [48] 的編碼器–解碼器：兩張影像通過同一個 Vision Transformer 編碼器 [47]，兩個解碼器透過交叉注意力交換資訊，再由迴歸頭輸出點圖與每個像素的信心值 [29]。輸出只定義到差一個未知的尺度 [29]。超過兩張影像時，DUSt3R 預測成對的點圖，再執行一次全域對齊，解出一組一致的點圖、位姿與每對影像的尺度，這是光束法平差在學習式幾何上的對應做法 [29]。

MASt3R [30] 加入一個輸出稠密局部特徵的頭（以匹配損失訓練），以及快速的互惠配對機制，讓點圖網路也能產生準確的像素對應 [30]。**VGGT** [31] 更進一步：一個大型 Transformer 以交替的逐畫面與全域注意力，接收一張、幾張或數百張影像，一次預測所有影像的相機參數、深度圖、點圖與點軌跡；作者報告這只需不到一秒，而且不需進一步處理就常常勝過以最佳化為基礎的替代方法 [31]。

這些模型並沒有讓幾何過時。它們的預測仍然可以精修：VGGT 的作者報告它的相機與軌跡是光束法平差很好的初始值，經過 BA 後準確度還能進一步提升 [31]。它們的準確度也取決於場景和訓練資料有多接近。不過，當影像很少、重疊很少或沒有校正時，它們是目前最好的工具。

動態場景（簡介）

白話版。如果場景在動，單一個三維模型就不夠了：模型還必須依賴時間。

常見的做法是一個標準（靜態）場景，加上隨時間變化的形變。D-NeRF [32] 把時間當成輸入，並一起學習兩個網路：一個把場景編碼在標準空間中，另一個把標準場景對應到給定時間的形變狀態；它只需要一台繞著場景移動的相機 [32]。4D Gaussian Splatting [33] 保留同一組 3D 高斯，並用一個輕量 MLP 預測它們在每個時間點的形變，這個 MLP 的輸入是分解過的時空特徵網格；論文報告了即時算繪，例如在 RTX 3090 GPU 上以 800×800 解析度達到每秒 82 個畫面 [33]。動態重建比靜態重建的限制少得多，因為每個瞬間被看到的視角更少。

評估重建結果

白話版。有兩件不同的事可以量測：從新視角算繪出來的畫面看起來像不像真的照片，以及形狀在幾何上對不對。一個方法可能其中一項做得很好，另一項卻很差。

新視角品質

留出一些照片，用它們的相機位姿算繪重建結果，再做比較。標準指標是 PSNR、SSIM [34] 與 LPIPS [35]，全都定義在 DLCV 09。PSNR 與 SSIM 量測像素與結構的一致性；LPIPS 在深度特徵空間中量測感知相似度。常見的協定（3DGS 沿用自 Mip-NeRF 360）是把每次拍攝的每第 8 張照片留作測試集 [6]。新視角指標取決於留出的位姿是否正確，所以它們也會獎勵好的 SfM。

幾何準確度

令 $\mathcal{P}$ 為從重建結果取樣的點，$\mathcal{G}$ 為從真值掃描取樣的點。定義兩個有方向的距離

$$\text{acc} = \frac{1}{|\mathcal{P}|}\sum_{\mathbf{p}\in\mathcal{P}} \min_{\mathbf{g}\in\mathcal{G}} \|\mathbf{p}-\mathbf{g}\|, \qquad \text{comp} = \frac{1}{|\mathcal{G}|}\sum_{\mathbf{g}\in\mathcal{G}} \min_{\mathbf{p}\in\mathcal{P}} \|\mathbf{g}-\mathbf{p}\|,$$

其中 $|\cdot|$ 計算點的數目。**準確度**（accuracy）問的是「我的點離真值近嗎？」；**完整度**（completeness）問的是「真值是否全部被涵蓋？」。它們的和或平均是 **Chamfer 距離**，自 Fan 等人 [36] 以來被廣泛用作點集合的損失與指標，他們用的是平方距離的定義；各篇論文在是否平方、取總和還是平均上並不一致，所以比較數字前請先檢查定義。在 DTU 上，MVSNet 報告準確度與完整度的平均作為「整體」分數 [16]。

Tanks and Temples [38] 用門檻 τ 而不是平均。令 $e_{\mathbf{p}}$ 為重建點到真值的距離，$e_{\mathbf{g}}$ 為真值點到重建結果的距離，

$$P(\tau) = \frac{100}{|\mathcal{P}|}\sum_{\mathbf{p}\in\mathcal{P}} [\, e_{\mathbf{p}} < \tau \,], \qquad R(\tau) = \frac{100}{|\mathcal{G}|}\sum_{\mathbf{g}\in\mathcal{G}} [\, e_{\mathbf{g}} < \tau \,], \qquad F(\tau) = \frac{2P(\tau)R(\tau)}{P(\tau)+R(\tau)},$$

其中 $[\cdot]$ 在條件成立時為 1，否則為 0，P 是精確率（以百分比表示），R 是召回率，F 是它們的調和平均，即 **F-score** [38]。因為是調和平均，只有重建結果同時準確又完整時 F 才會高；一組稀疏但完美的點，或是填滿整個空間的點雲，兩者分數都很低 [38]。F-score 對少數遠處的離群值也很穩健，而這些離群值可能主導 Chamfer 的平均（練習 5）。

資料集與基準

- **DTU** [37]：包含 80 個桌面場景的多視角立體視覺資料集，每個場景有 49 或 64 個精確已知的相機位置（來自六軸工業機器人），參考幾何來自結構光掃描器 [37]。是 MVS 與神經曲面方法的標準資料集（NeuS 使用其中 15 個場景 [22]）。
- **Tanks and Temples** [38]：在實驗室外拍攝的大型室內外場景的高解析度影片，真值來自工業雷射掃描器，分成中等與進階兩組，並在線上伺服器上用上面的 F-score 評估 [38]。
- **Mip-NeRF 360** [39]：9 個無邊界場景（5 個室外、4 個室內），每個場景有一個中心物體與細節豐富的背景，為新視角合成而拍攝 [39]。3DGS 在它以及 Tanks and Temples 場景等資料上做評估 [6]。

失效模式

白話版。上述方法依賴的證據與假設不同。傳統配對受益於穩定外觀、紋理與重疊視角；SfM 會估計相機姿態，不要求全部預先已知 [2]。NeRF 明確建模隨視角改變的顏色，但原始形式使用已知姿態 [5]；DUSt3R 則能在沒有先驗相機校正與姿態的情況下推論幾何 [29]。靜態場景方法需要擴充才能處理動態。下列失效模式指出缺少哪些資訊、哪些假設受到挑戰，並非所有方法都有相同限制。

- **無紋理表面**（白牆、天空）。光度一致性在每個深度都相同，如同平面掃描那次實驗；重建結果會有破洞或雜訊。學習式 MVS 與前饋式模型用學到的先驗把它們補起來，而那些先驗可能是錯的。
- **反光與透明表面**。鏡子或有光澤的汽車會顯示一個反射的場景，而它移動的方式和表面不同，所以配對與密度都會出錯。NeRF 傾向用藏在半透明表面底下、位於物體內部的光源來偽造亮面反光；Ref-NeRF [40] 則改為建模繞表面法向量反射的輻射，改善了鏡面表面 [40]。
- **視角太少**。只有兩三張影像時，許多場景都能同樣好地解釋這些照片，逐場景最佳化會過度擬合（訓練視角很好，新視角卻壞掉）。像 DUSt3R [29] 這類前饋式模型是常見的解決辦法。
- **漂浮物**。自由空間中半透明的密度團塊，常常靠近相機，從訓練視角看起來沒問題，但從新視角看就像霧。Mip-NeRF 360 為它們命名，並加入以失真為基礎的正則化項來抑制它們 [39]；3DGS 為了同樣的原因會定期重設不透明度 [6]。
- **尺度模糊**。單眼影像集合只能把幾何固定到差一個尺度；要得到公制大小，需要已知距離、校正過的相機組，或其他感測器 [1]。
- **位姿誤差**。 NeRF 與 3DGS 假設位姿準確；小誤差就會讓細節變模糊。BARF [41] 同時最佳化位姿與輻射場，並對位置編碼使用由粗到細的排程，因為高頻否則會妨礙對位 [41]。
- **移動的物體與光照變化**，例如拍攝過程中有人走過或太陽移動，破壞的是靜態場景這個假設本身。

現代觀點

綜述論文怎麼說

有四篇回顧為本篇定下架構。Seitz 等人 [42] 為多視角立體視覺建立了傳統的分類法，至今仍能描述這裡的每一種方法：場景表示法、光度一致性量測、可見性模型、形狀先驗、重建演算法，以及對初始化的需求。Furukawa 與 Hernández [43] 把 MVS 呈現為由兩個要素組成的影像–幾何一致性最佳化：穩健的光度一致性量測與有效率的最佳化，並回顧成功的演算法與尚存的挑戰。Tewari 等人 [44] 回顧神經算繪，從場景表示法與可微分的影像形成，到新視角合成、跨場景泛化、動態內容、編輯與重新打光等應用，並列出尚未解決的問題，例如與傳統圖學工具的整合，以及可解釋性。Chen 與 Wang [45] 回顧 3D Gaussian Splatting：它的原理與最佳化、延伸它的七個方向、應用，以及效能比較；他們指出研究重心正從逐場景最佳化，轉向前饋式的高斯預測。

僅線上版：「現代觀點」綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/3d-reconstruction/papers/現代觀點綜述導讀/*

2023–2026 的最新進展

- **明確、可光柵化的表示法**贏得了速度競賽。自 3DGS [6] 以來，大量的視角合成工作建立在高斯基元之上。3DGS 綜述 [45] 把後續工作分成稀疏輸入、省記憶體、逼真的變體、更好的最佳化、帶有額外屬性的高斯、混合表示法，以及新的算繪演算法；幾何（SuGaR [27]、2DGS [28]）與動態（4DGS [33]）本身就是活躍的方向。
- NeuS 這一系的**神經隱式曲面** [22]，在不漏水又準確的曲面比算繪速度重要時，仍是首選。
- **前饋式幾何**是最大的改變。DUSt3R [29]、MASt3R [30] 與 VGGT [31] 直接從影像估計相機與稠密幾何；它們越來越常被用來初始化或取代傳統 SfM 流程的部分環節，3DGS 綜述 [45] 也描述了高斯上的同樣趨勢。
- **傳統流程仍是基準**。 COLMAP [2]、[3] 仍是訓練與評估時相機位姿的常用來源，光束法平差也仍用來精修學習式的預測。

尚未解決的問題

1. **幾何與外觀兼顧**。為新視角調校的方法產生的曲面很差，而曲面準確的方法算繪得比較不逼真。能快速兼顧兩者的單一表示法仍未出現。
2. **反射、透明與光照**。把表面、材質與照明分開（逆向算繪，inverse rendering），對真實拍攝的資料仍未解決。
3. **前饋式模型的泛化**：遠離訓練分布時的準確度、公制尺度，以及非常大的場景。
4. **動態與野外拍攝**：有移動的人、變化的光線與很少的視角。
5. **評估**。「真值」中的位姿誤差、與取樣有關的 Chamfer 數字，以及不一致的指標定義，讓跨論文比較很脆弱。

重點整理

- 場景重建需要相機幾何及適合輸出的表示；僅靠影像重建可能留下尺度歧義。
- 投影、對應與三角測量連結影像量測及三維結構。
- 運動恢復結構估計相機與稀疏結構，密集重建則加入更多對應假設。
- 神經場最佳化算繪觀測，良好新視角合成不單獨證明表面準確。
- 高斯與前饋表示改變算繪或推論成本，仍須檢查幾何。
- 報告視角指標及幾何準確度／完整度，並註明對齊、取樣與門檻。

練習

1. **手算深度誤差**。一對校正過的立體影像，$f = 800$ 像素，基線 $B = 0.1$ m。用 $\delta Z \approx Z^2 \delta d/(fB)$，如果視差差了 0.5 像素，$Z = 5$ m 的點深度誤差有多大？B 要多大才能把誤差壓到 2 公分以下？

   提示

   $\delta Z \approx 25 \cdot 0.5/(800 \cdot 0.1) = 0.156$ m。解 $25 \cdot 0.5/(800B) < 0.02$ 得 $B > 0.78$ m。基線變寬有幫助，但也會讓兩個視角看起來差異更大，使配對更困難。

2. **看見尺度模糊**。在兩視角的程式碼中，投影之前把所有場景點 `X` 與相機中心 `C2` 都乘以 3。證明像素座標不變，且估計出的位姿相同。列出三種可以固定尺度的外部資訊。

   提示

   投影要除以深度，所以把 $\mathbf{X}$ 和 $\mathbf{t}$ 一起縮放會抵消：$K(R(3\mathbf{X}) + 3\mathbf{t}) = 3K(R\mathbf{X} + \mathbf{t})$。外部資訊：場景中量到的距離（尺或已知大小的標記）、立體相機組的已知基線，或其他感測器，例如慣性測量單元或 GPS。

3. **讀懂權重**。在體積算繪的程式碼中，證明 $\sum_i w_i = 1 - \prod_i (1 - \alpha_i)$。然後移除霧的那一層，重新計算深度。為什麼穿過霧的光線的「深度」不是任何一個表面的深度？

   提示

   用歸納法：$T_{i+1} = T_i(1 - \alpha_i)$，所以 $w_i = T_i - T_{i+1}$，總和可以逐項消去，得到 $T_1 - T_{N+1} = 1 - T_{N+1}$。沒有霧時，所有權重都落在紅色層上，深度接近 3。有霧時，$\sum_i w_i t_i$ 是兩個不同位置的平均，所以可能落在空白空間；漂浮物就是這樣破壞深度圖的。

4. **加入自適應密度控制**。擴充二維 splatting 的程式碼：每 100 次迭代，找出累積位置梯度最大的高斯；複製小的、分裂大的（尺度除以 1.6），並移除不透明度低於 0.02 的高斯。把最後的 PSNR 與固定 40 個高斯的版本比較，也與單純從 80 個高斯開始的版本比較。

   提示

   陣列變大時，Adam 的動差陣列 `m` 與 `v2` 也必須跟著變大（新增的列初始化為零）。供參考：單純從 80 個高斯而不是 40 個開始、其他都不變，在我們的實驗中第 500 次迭代達到 27.00 dB（40 個時是 23.57 dB）。加密應該把新的高斯放在誤差所在之處，例如三角形的角落。

5. **Chamfer 與 F-score**。在 marching cubes 的程式碼中（格網 48），先在 $(0, 0, 0.6)$ 附近加入 1,000 個「漂浮物」點，然後改成刪掉 $x < 0$ 的那一半網格取樣。重新計算準確度、完整度、Chamfer 距離與 F-score。哪個指標對哪種缺陷有反應？

   提示

   在我們的實驗中，漂浮物（占 1% 的點）讓準確度從 0.0040 升到 0.0097，Chamfer 距離從 0.0040 升到 0.0068，而門檻 0.01 的 F-score 幾乎沒動（0.995 到 0.990）。刪掉一半曲面後，準確度維持在 0.0040，但完整度升到 0.2226，Chamfer 距離升到 0.1133，F-score 降到 0.665。平均值對少數遠處的離群值很敏感；設門檻的分數則不會，但它們透過召回率清楚反映出缺少的部分。

6. **平面掃描的視窗大小**。用 3、7 與 15 像素的 SSD 視窗執行平面掃描的程式碼。有紋理與無紋理的分數如何變化？方塊的邊緣又發生什麼事？

   提示

   在我們的實驗中，有紋理的分數是 95.2%、95.3% 與 96.6%，無紋理的分數是 4.9%、14.8% 與 27.7%。大視窗會碰到平坦區塊外面的紋理，所以區塊內的像素從周圍借用深度；代價是深度邊緣變模糊，方塊看起來被「撐胖」進背景。學習式 MVS 用三維 CNN 取得這種脈絡，卻不產生模糊。

本篇建立的場景模型，也是 DLCV 13 世界模型的建構元件，在那裡，代理人必須預測三維環境如何變化。

參考文獻

1. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, 2004. doi
2. J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proc. IEEE CVPR*, 2016. CVF
3. J. L. Schönberger, E. Zheng, J.-M. Frahm and M. Pollefeys, "Pixelwise view selection for unstructured multi-view stereo," in *Proc. ECCV*, 2016. doi
4. J. L. Schönberger et al., "COLMAP," official documentation (general-purpose structure-from-motion and multi-view stereo pipeline). link
5. B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. ECCV*, 2020; also *Communications of the ACM*, vol. 65, no. 1, pp. 99–106, 2022. doi ·CACM doi ·arXiv
6. B. Kerbl, G. Kopanas, T. Leimkühler and G. Drettakis, "3D Gaussian Splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, 2023. doi ·arXiv
7. H. C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, pp. 133–135, 1981. doi
8. R. I. Hartley, "In defense of the eight-point algorithm," *IEEE TPAMI*, vol. 19, no. 6, pp. 580–593, 1997. doi
9. D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE TPAMI*, vol. 26, no. 6, pp. 756–770, 2004. doi
10. M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981. doi
11. D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. doi
12. N. Snavely, S. M. Seitz and R. Szeliski, "Photo tourism: Exploring photo collections in 3D," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 835–846, 2006. doi
13. B. Triggs, P. F. McLauchlan, R. I. Hartley and A. W. Fitzgibbon, "Bundle adjustment — A modern synthesis," in *Vision Algorithms: Theory and Practice*, LNCS 1883, pp. 298–372, Springer, 2000. doi
14. R. T. Collins, "A space-sweep approach to true multi-image matching," in *Proc. IEEE CVPR*, pp. 358–363, 1996. doi
15. B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. SIGGRAPH*, pp. 303–312, 1996. doi
16. Y. Yao, Z. Luo, S. Li, T. Fang and L. Quan, "MVSNet: Depth inference for unstructured multi-view stereo," in *Proc. ECCV*, 2018. CVF ·doi
17. N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, 1995. doi
18. M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," in *Proc. NeurIPS*, 2020. proceedings
19. T. Müller, A. Evans, C. Schied and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, 2022. doi ·arXiv

20. J. J. Park, P. Florence, J. Straub, R. Newcombe and S. Lovegrove, "DeepSDF: Learning continuous signed distance functions for shape representation," in *Proc. IEEE/CVF CVPR*, 2019. CVF
21. L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *Proc. IEEE/CVF CVPR*, 2019. CVF
22. P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura and W. Wang, "NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction," in *Proc. NeurIPS*, 2021. proceedings ·arXiv
23. M. Zwicker, H. Pfister, J. van Baar and M. Gross, "EWA volume splatting," in *Proc. IEEE Visualization*, 2001. doi
24. W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," in *Proc. SIGGRAPH*, pp. 163–169, 1987. doi
25. M. Kazhdan, M. Bolitho and H. Hoppe, "Poisson surface reconstruction," in *Proc. Eurographics Symposium on Geometry Processing*, 2006. doi
26. M. Kazhdan and H. Hoppe, "Screened Poisson surface reconstruction," *ACM Transactions on Graphics*, vol. 32, no. 3, 2013. doi
27. A. Guédon and V. Lepetit, "SuGaR: Surface-aligned Gaussian splatting for efficient 3D mesh reconstruction and high-quality mesh rendering," in *Proc. IEEE/CVF CVPR*, 2024. CVF ·arXiv
28. B. Huang, Z. Yu, A. Chen, A. Geiger and S. Gao, "2D Gaussian splatting for geometrically accurate radiance fields," in *Proc. SIGGRAPH (Conference Papers)*, 2024. doi ·arXiv
29. S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii and J. Revaud, "DUSt3R: Geometric 3D vision made easy," in *Proc. IEEE/CVF CVPR*, 2024. CVF ·arXiv
30. V. Leroy, Y. Cabon and J. Revaud, "Grounding image matching in 3D with MASt3R," in *Proc. ECCV*, 2024. doi ·arXiv
31. J. Wang, M. Chen, N. Karaev, A. Vedaldi, C. Rupprecht and D. Novotny, "VGGT: Visual geometry grounded transformer," in *Proc. IEEE/CVF CVPR*, 2025. CVF ·arXiv
32. A. Pumarola, E. Corona, G. Pons-Moll and F. Moreno-Noguer, "D-NeRF: Neural radiance fields for dynamic scenes," in *Proc. IEEE/CVF CVPR*, 2021. CVF
33. G. Wu, T. Yi, J. Fang, L. Xie, X. Zhang, W. Wei, W. Liu, Q. Tian and X. Wang, "4D Gaussian splatting for real-time dynamic scene rendering," in *Proc. IEEE/CVF CVPR*, 2024. CVF ·arXiv
34. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. doi
35. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proc. IEEE/CVF CVPR*, 2018. CVF
36. H. Fan, H. Su and L. J. Guibas, "A point set generation network for 3D object reconstruction from a single image," in *Proc. IEEE CVPR*, 2017. CVF
37. R. Jensen, A. Dahl, G. Vogiatzis, E. Tola and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *Proc. IEEE CVPR*, pp. 406–413, 2014. doi
38. A. Knapitsch, J. Park, Q.-Y. Zhou and V. Koltun, "Tanks and Temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017. doi ·benchmark
39. J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan and P. Hedman, "Mip-NeRF 360: Unbounded anti-aliased neural radiance fields," in *Proc. IEEE/CVF CVPR*, 2022. CVF ·arXiv
40. D. Verbin, P. Hedman, B. Mildenhall, T. Zickler, J. T. Barron and P. P. Srinivasan, "Ref-NeRF: Structured view-dependent appearance for neural radiance fields," in *Proc. IEEE/CVF CVPR*,

2022. CVF
41. C.-H. Lin, W.-C. Ma, A. Torralba and S. Lucey, "BARF: Bundle-adjusting neural radiance fields," in *Proc. IEEE/CVF ICCV*, 2021. CVF
42. S. M. Seitz, B. Curless, J. Diebel, D. Scharstein and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *Proc. IEEE CVPR*, vol. 1, pp. 519–528, 2006. doi
43. Y. Furukawa and C. Hernández, "Multi-view stereo: A tutorial," *Foundations and Trends in Computer Graphics and Vision*, vol. 9, no. 1–2, pp. 1–148, 2015. doi
44. A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann et al., "Advances in neural rendering," *Computer Graphics Forum*, vol. 41, no. 2, pp. 703–735, 2022. doi ·arXiv
45. G. Chen and W. Wang, "A survey on 3D Gaussian splatting," *ACM Computing Surveys*, vol. 58, no. 12, 2026. doi ·arXiv
46. scikit-image contributors, "skimage.measure.marching_cubes," *scikit-image documentation* (*API reference*). docs
47. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. ICLR*, 2021. arXiv
48. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is all you need," in *Proc. NeurIPS*, 2017. proceedings
49. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR*, 2015. arXiv

DLCV 11A・深度估計與光流

深度與運動的密集對應：校正過的立體對與視差、匹配代價、半全域匹配與學習式代價體、使用尺度與平移不變損失的單目相對與公制深度、從影片做自監督深度、從 FlowNet 到 RAFT 的光流、場景流，以及這三項任務的指標、基準與失效模式，並附經過測試的 NumPy 與 OpenCV 程式碼。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/

你將學到

- 區分立體對應、時間光流與單目深度推論。
- 連結視差、校準相機幾何與深度誤差。
- 比較匹配成本、成本體積與迭代細化。
- 區分相對深度、公尺尺度深度與視角合成監督。
- 評估時註明對齊、可見性遮罩、單位及資料切分。

先看全貌

深度圖告訴你每個像素上的表面有多遠。光流場告訴你每個像素上的那個點，在下一張影格裡移到哪裡。機器人、汽車、手機與影片剪輯軟體兩者都需要：用深度知道東西在哪裡，用光流知道東西怎麼動。兩者回答的是同一個問題：另一張影像中，哪個像素拍到的是同一個點？

> **用最簡單的話說**
>
> 把一支鉛筆舉在臉前，先閉左眼，再閉右眼。鉛筆會往旁邊跳一下，而遠處的窗戶幾乎沒動。跳得多遠，就告訴你它有多近。立體相機用兩顆鏡頭做同樣的事，電腦再量出每個像素跳了多遠。如果只有一隻眼睛，你仍然可以猜距離，因為你知道一輛車或一扇門通常有多大；但近距離看一輛玩具車，你就可能被騙。光流是把這個「跳來跳去」的遊戲放到時間上玩：把這張影格和下一張比較，量出每一小塊東西移動了多遠。

為什麼重要。立體深度讓機器人和汽車不靠雷射掃描器也能量距離；單張影像的深度讓手機把人像的背景模糊掉，也能把物件放進影片裡；光流則驅動影片超解析度、影格內插與影像穩定。

本篇與其他篇章的關係。 DLCV 11 介紹相機幾何、三角測量與多視角立體視覺；DLCV 09A.3 介紹亮度恆定與扭曲；長距離的點追蹤在 DLCV 07，稀疏關鍵點配對在 IP 12。我們連結過去，不重複它們。本篇的重點是密集的情況：每個像素都有一個答案。

圖 1 畫出本篇的四種模型家族：三種是比對兩張影像，第四種則靠它學到的知識。

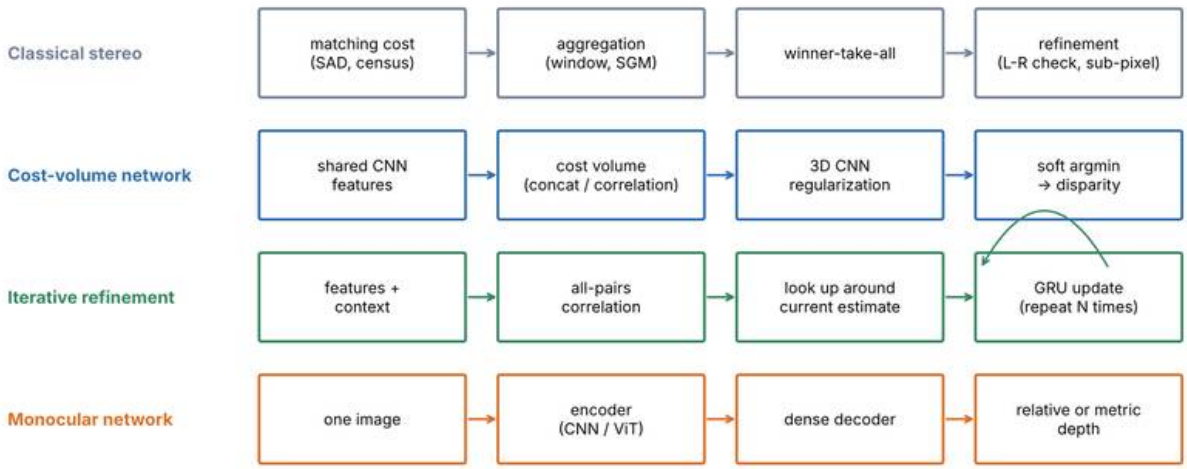


圖 *1* — 密集對應與深度模型的四個家族。傳統立體匹配（灰）與代價體網路（藍）遵循同樣的四個階段；迭代模型（綠）重複「查表再更新」的步驟；單目網路（橘）從單張影像預測深度。

三個密集對應問題

白話版。校正後立體匹配沿同一列找像素對應；光流跨影格匹配，單目深度則沒有第二張影像可用。

精確版。令 I_1, I_2 為兩張 $H \times W$ 像素的影像，$\mathbf{p} = (u, v)$ 為一個像素，u 是欄、v 是列。

- **光流**是一個向量場 $\mathbf{F}(\mathbf{p}) = (\Delta u, \Delta v)$，使得在 I_1 的 $\mathbf{p}$ 看到的點，在 I_2 的 $\mathbf{p} + \mathbf{F}(\mathbf{p})$ 被看到。每個像素有兩個未知數；相機與場景都可能在動。
- **立體視差**是特殊情況：兩台相機在同一瞬間看著場景，並且已經校正（見下一節），所以每個配對都落在同一列上：$\mathbf{F}(\mathbf{p}) = (-d(\mathbf{p}), 0)$。每個像素只有一個未知數，即視差 $d \geq 0$，它可以換算成深度。
- **單目深度**要求只從 I_1 得到深度圖 $Z(\mathbf{p})$。沒有對應可用，所以答案來自學到的規律，而且除非網路學會了公制線索，否則只定義到一個未知的尺度。
- **場景流**要求每個像素所見之點的三維運動；它結合兩個時刻的深度與光流（見下方的場景流（簡介））。

兩張影像的問題共享同一個結構：一個為候選配對打分的**代價**、一次在候選之間的搜尋，以及一種處理代價有歧義的像素的方法。

立體幾何：校正與視差

白話版。校準並校正的影像對中，近點視差較大；焦距與相機基線把視差換成深度。

校正

對一般的相機對，一個像素的配對會落在另一張影像的**對極線**上（DLCV 11，對極幾何）。校正（rectification）對兩張影像各施加一個單應性，讓所有對極線變成水平，且對應的線有相同的列編號 [1]。校正之後，兩台虛擬相機有相同的內參、平行的光軸，以及純水平的基線。立體相機組只需校準一次，每個影格在匹配之前都先校正；下面所有立體基準提供的都是校正過的影像對。本篇其餘部分，影像對一律是校正過的。

視差與深度

把左相機中心放在原點，右相機放在 x 軸上距離 B 處（圖 2a）。深度為 Z、水平位置為 X 的點，在左影像投影到欄 $x_L = fX/Z$，在右影像投影到欄 $x_R = f(X-B)/Z$，兩者都從主點起算。相減得到

$$d = x_L - x_R = \frac{fB}{Z}, \qquad Z = \frac{fB}{d},$$

其中 f 是以像素為單位的焦距，B 是**基線**（baseline，兩個相機中心之間的距離，單位為公尺），Z 是沿光軸的深度，d 是以像素為單位的**視差**（disparity）。深度與視差成反比：視差是一種逆深度，只差一個係數 fB。大視差屬於近的物體，視差為零表示無窮遠的點。

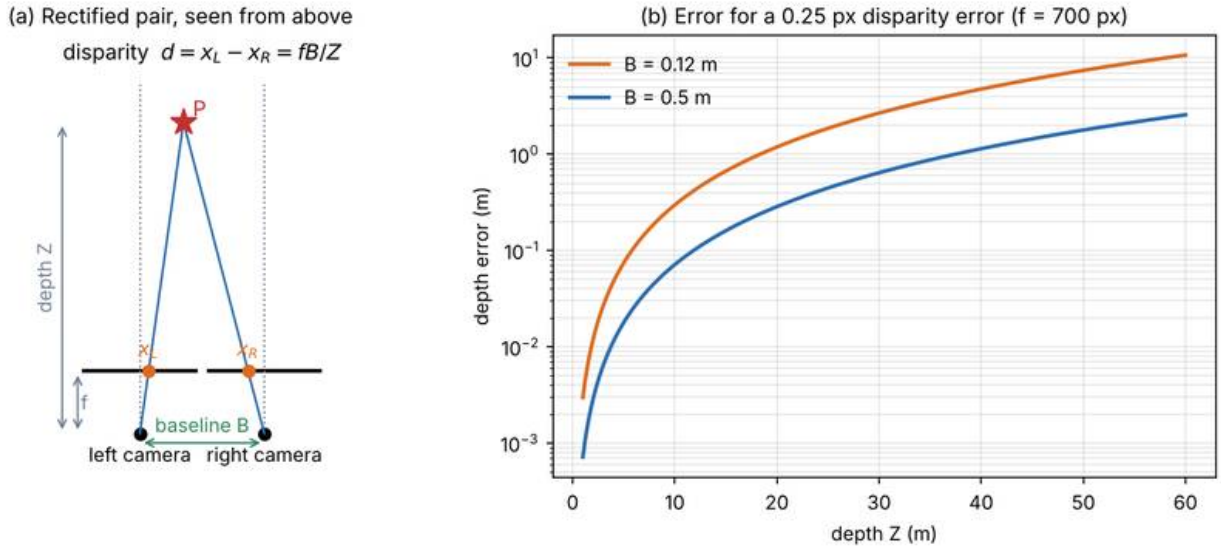


圖 2 — (*a*) 從上方看一對校正過的立體影像：同一點在兩張影像之間位移了視差 d。(*b*) 視差誤差為 *0.25* 像素、焦距為 *700* 像素時的深度誤差；誤差隨距離的平方成長，並隨基線變寬而縮小。

因為 $Z = fB/d$，視差的小誤差 δd 會變成約 $\delta Z \approx Z^2 \delta d/(fB)$ 的深度誤差，這與 DLCV 11，三角測量推導的關係相同。下面的程式碼用一台假想的 $f = 700$ 像素的相機，把它換成數字。

```
import numpy as np
```

```
f = 700.0                                   # focal length in pixels (a hypothetical rig)
for B in [0.12, 0.5]:                       # baselines in metres
    print(f"baseline {B} m: depth at a disparity of 1 px = {f * B:.0f} m")
    for Z in [2, 10, 30, 60]:
        d = f * B / Z                       # disparity in pixels
        dZ = Z ** 2 * 0.25 / (f * B)        # depth error for a 0.25 px disparity error
        print(f"   Z = {Z:2d} m  d = {d:6.2f} px  depth error for 0.25 px: {dZ:6.3f} m ({100
         ↪  * dZ / Z:4.1f}%)")
```

```
baseline 0.12 m: depth at a disparity of 1 px = 84 m
   Z =  2 m  d =  42.00 px  depth error for 0.25 px:  0.012 m ( 0.6%)
   Z = 10 m  d =   8.40 px  depth error for 0.25 px:  0.298 m ( 3.0%)
   Z = 30 m  d =   2.80 px  depth error for 0.25 px:  2.679 m ( 8.9%)
   Z = 60 m  d =   1.40 px  depth error for 0.25 px: 10.714 m (17.9%)
baseline 0.5 m: depth at a disparity of 1 px = 350 m
   Z =  2 m  d = 175.00 px  depth error for 0.25 px:  0.003 m ( 0.1%)
   Z = 10 m  d =  35.00 px  depth error for 0.25 px:  0.071 m ( 0.7%)
   Z = 30 m  d =  11.67 px  depth error for 0.25 px:  0.643 m ( 2.1%)
   Z = 60 m  d =   5.83 px  depth error for 0.25 px:  2.571 m ( 4.3%)
```

在 10 m 處，視差差四分之一像素，在基線 12 cm 時會造成 30 cm 的誤差，在基線 50 cm 時是 7 cm。在 60 m 處，短基線只看到 1.4 像素的視差，深度不確定性達 18%。這就是為什麼行車用的立體相機要用寬基線與高解析度，也是為什麼子像素精度很重要：最後那一小部分像素，決定了遠距離的表現。

傳統立體匹配

白話版。傳統區塊匹配搜尋候選視差、比較局部區塊，再細化不可靠匹配。

Scharstein 與 Szeliski 把密集的雙影像立體匹配演算法整理成四個步驟，大多數方法都以某種形式執行這些步驟 [2]：

1. **匹配代價計算**：對每個像素、每個視差，算出一個相異度。
2. **代價（支撐）聚合**：在鄰域內結合代價以降低雜訊。
3. **視差計算或最佳化**：為每個像素選出一個視差，可以是局部的選擇，也可以是對整張影像最佳化一個能量函數。
4. **視差精修**：子像素估計、一致性檢查、填補與平滑。

隨那篇論文而設的 Middlebury 立體匹配網站，至今仍提供線上評估 [2], [3]。DLCV 11 的平面掃描，正是使用視窗化 SSD 的步驟 1–3；這裡我們把每個步驟看得更仔細，並加上精修。

匹配代價與代價體

最簡單的代價是比較小視窗內的強度。像素 (u, v) 在視差 d 時的**絕對差總和**（sum of absolute differences, SAD）為

$$C(u,v,d) = \sum_{(i,j)\in\mathcal{W}} |I_L(u+i,v+j) - I_R(u+i-d,v+j)|,$$

其中 I_L, I_R 是左右影像，$\mathcal{W}$ 是以該像素為中心、$w \times w$ 視窗的位移集合，d 遍歷整數假設 $0, 1, \ldots, D-1$。改用平方差就是 SSD。對每個像素、每個 d 都算出 C，就填滿一個大小為 $D \times H \times W$ 的**代價體**（cost volume）。**贏者全拿**（winner-take-all, WTA）規則選出 $\hat{d}(u,v) = \arg\min_d C(u,v,d)$。

SAD 假設兩台相機對同一個點記錄到相同的亮度。真實的相機在增益與曝光上並不相同，所以實用的系統會用**零均值正規化互相關**（zero-mean normalized cross-correlation, ZNCC）或 **census 轉換**，後者把「哪些鄰居比該像素暗」編碼成一個位元字串，再比較位元字串 [4]。

僅線上版：穩健的匹配代價：*ZNCC* 與 *census* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/notes/穩健的匹配代價 zncc-與-census/*

圖 3a 畫出代價體的一列：深色的谷底描出各個表面。圖 3b 畫出四種典型的代價曲線。有紋理的像素有一個尖銳的最小值；無紋理的像素是平坦的曲線；重複的圖案每個週期都有一個同樣深的最小值；被遮擋的像素在右影像中被藏起來，根本沒有正確的最小值。

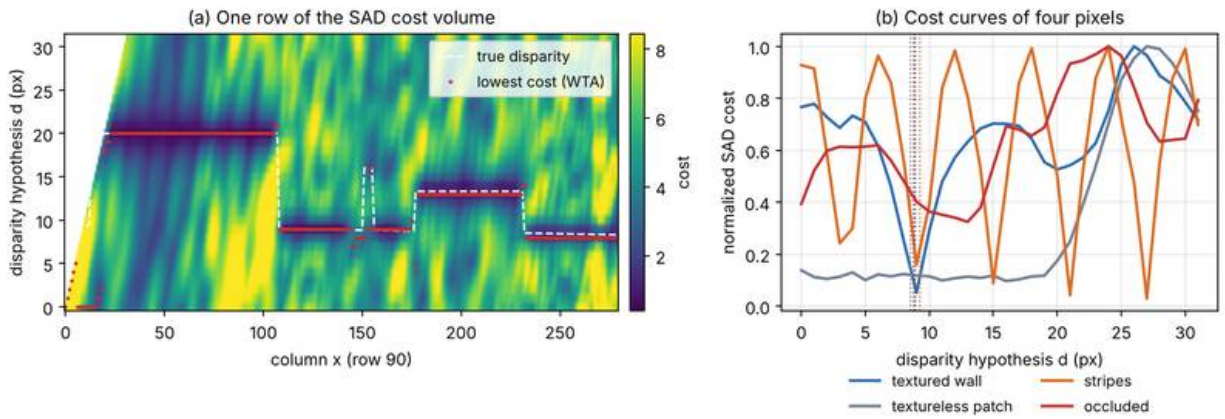


圖 *3* — (*a*) 下面合成場景的 *SAD* 代價體的一列；低代價（深色）沿著真實視差（虛線）。(*b*) 四個像素的代價曲線；點線標出它們的真實視差。

程式碼：附左右一致性檢查的區塊匹配

這段程式碼對一個已知深度的合成場景做光線投射，產生一對校正過的影像：一面傾斜、有紋理的牆，牆前有一個箱子、一個圓盤與一根細柱，牆上還有一塊無紋理的區塊與一塊細條紋。它建立 SAD 代價體，取 WTA 視差，並執行**左右一致性檢查**：改用右影像當參考再比對一次，兩個答案相差超過一個像素的像素就剔除。被遮擋的像素通常過不了這項檢查，因為它們在右影像中的「配對」屬於另一個表面，而那個表面自己的配對指向別處。為了比較，它也執行 OpenCV 的半全域匹配器（下一小節）。誤差以 **bad-1** 回報，也就是視差誤差超過 1 像素的像素百分比，並依各種區域分別計算。

```
import numpy as np
import cv2
from scipy import ndimage as ndi
```

```
rng = np.random.default_rng(0)
H, W, f, B = 180, 280, 320.0, 0.25                    # image size, focal length (px),
 ↪  baseline
tex = ndi.gaussian_filter(rng.random((700, 700)), 1.2)
tex = (tex - tex.min()) / np.ptp(tex)                 # random texture painted on every
 ↪  surface

def render(cx):
    """Ray-cast a rectified camera at (cx, 0, 0) looking down +Z: image, depth Z, region
     ↪  labels."""
    v, u = np.mgrid[0:H, 0:W].astype(float)
    dx, dy = (u - W / 2) / f, (v - H / 2) / f            # ray direction (dx, dy, 1)
    Z = (9 + 0.15 * cx) / (1 - 0.15 * dx)                # slanted wall Z = 9 + 0.15 X
    lab = np.zeros((H, W), int)                          # 0 wall, 1 box, 2 disc, 3 thin pole
    for k, z, inside in [(2, 6.0, lambda X, Y: (X - 1.2) ** 2 + (Y - 0.3) ** 2 < 0.36),
                         (3, 5.0, lambda X, Y: np.abs(X - 0.2) < 0.04),
                         (1, 4.0, lambda X, Y: (np.abs(X + 1.0) < 0.6) & (np.abs(Y) < 0.6))]:
        hit = inside(cx + z * dx, z * dy) & (z < Z)
        Z, lab = np.where(hit, z, Z), np.where(hit, k, lab)
    X, Y = cx + Z * dx, Z * dy
    img = ndi.map_coordinates(tex, [350 + 40 * Y + 60 * lab, 350 + 40 * X], order=1)
    flat = (lab == 0) & (X > 2.2) & (X < 3.6) & (Y > -2.0) & (Y < -0.9)
    rep = (lab == 0) & (X > -3.6) & (X < -2.2) & (Y > 0.9) & (Y < 2.0)
    img[flat] = 0.5                                       # textureless patch:
     ↪  label 4
    img[rep] = 0.5 + 0.4 * np.sin(2 * np.pi * X[rep] / 0.15)        # repetitive stripes:
     ↪  label 5
    lab[flat], lab[rep] = 4, 5
    return img + rng.normal(0, 0.01, img.shape), Z, lab

left, Z_left, lab = render(0.0)
right, Z_right, _ = render(B)
d_true = f * B / Z_left                                  # ground-truth disparity of the left
 ↪  view
rows, cols = np.mgrid[0:H, 0:W]
xr = np.round(cols - d_true).astype(int)                 # where each left pixel lands in the
 ↪  right view
occluded = (xr < 0) | (np.abs(Z_right[rows, np.clip(xr, 0, W - 1)] - Z_left) > 0.05)

def sad_volume(ref, other, D, win, sign):
    """cost[d](y, x) = sum over a win x win window of |ref(y, x) - other(y, x - sign*d)|."""
    cost = np.full((D, H, W), np.inf)
    for d in range(D):
        c = ndi.uniform_filter(np.abs(ref - np.roll(other, sign * d, axis=1)), win) * win **
         ↪  2
        if sign > 0: c[:, :d] = np.inf                   # no partner pixel exists there
        else:        c[:, W - d:] = np.inf
        cost[d] = c
    return cost

D, win = 32, 7
d_left = sad_volume(left, right, D, win, +1).argmin(0)     # winner-take-all, left image as
 ↪  reference
d_right = sad_volume(right, left, D, win, -1).argmin(0)    # winner-take-all, right image as
 ↪  reference
```

```
lr_ok = np.abs(d_left - d_right[rows, np.clip(cols - d_left, 0, W - 1)]) <= 1   # left-right
 ↪  check

sgbm = cv2.StereoSGBM_create(minDisparity=0, numDisparities=D, blockSize=5,
                             P1=8 * 25, P2=32 * 25, uniquenessRatio=0, disp12MaxDiff=-1)
u8 = lambda a: np.clip(a * 255, 0, 255).astype(np.uint8)
d_sgbm = sgbm.compute(u8(left), u8(right)).astype(float) / 16    # OpenCV stores 16 x
 ↪  disparity

valid = ~occluded & (cols >= D)                        # visible in both views, away from the
 ↪  left strip
regions = {"textured": valid & np.isin(lab, [0, 1, 2]), "textureless": valid & (lab == 4),
           "stripes": valid & (lab == 5), "thin pole": valid & (lab == 3)}
bad = lambda d, m: 100 * np.mean(np.abs(d - d_true)[m] > 1)      # % of pixels off by > 1 px
print(f"disparity {d_true.min():.1f}-{d_true.max():.1f} px; occluded: {100 *
 ↪  occluded.mean():.1f}% of pixels")
print("bad-1 (%)          " + "".join(f"{k:>12s}" for k in regions))
for name, d in [("SAD + WTA", d_left), ("OpenCV SGBM", d_sgbm)]:
    print(f"{name:19s}" + "".join(f"{bad(d, m):12.1f}" for m in regions.values()))
print(f"left-right check: keeps {100 * lr_ok[valid].mean():.1f}% of visible pixels, "
      f"rejects {100 * (~lr_ok[occluded]).mean():.1f}% of occluded ones")
kept = valid & lr_ok
print(f"SAD + WTA bad-1 on all visible pixels: {bad(d_left, valid):.1f}% -> after the check
 ↪  {bad(d_left, kept):.1f}%")
```

```
disparity 8.3-20.0 px; occluded: 7.7% of pixels
bad-1 (%)              textured textureless     stripes   thin pole
SAD + WTA                   1.8        67.7        89.6        17.4
OpenCV SGBM                 1.7        18.2        53.3        67.2
left-right check: keeps 96.6% of visible pixels, rejects 93.0% of occluded ones
SAD + WTA bad-1 on all visible pixels: 6.1% -> after the check 3.6%
```

這些數字只來自這個合成場景。在有紋理的表面上，SAD 區塊匹配有 98% 的像素是對的；在無紋理的區塊上，三分之二是錯的；在條紋上，十個有九個是錯的，正如代價曲線所預測的。左右一致性檢查剔除了 93% 被遮擋的像素，同時保留了 96.6% 看得見的像素，保留的像素上的 bad-1 從 6.1% 降到 3.6%。被剔除的像素是洞，不是錯誤：後面的階段知道不要相信它們，可以從鄰居填補。

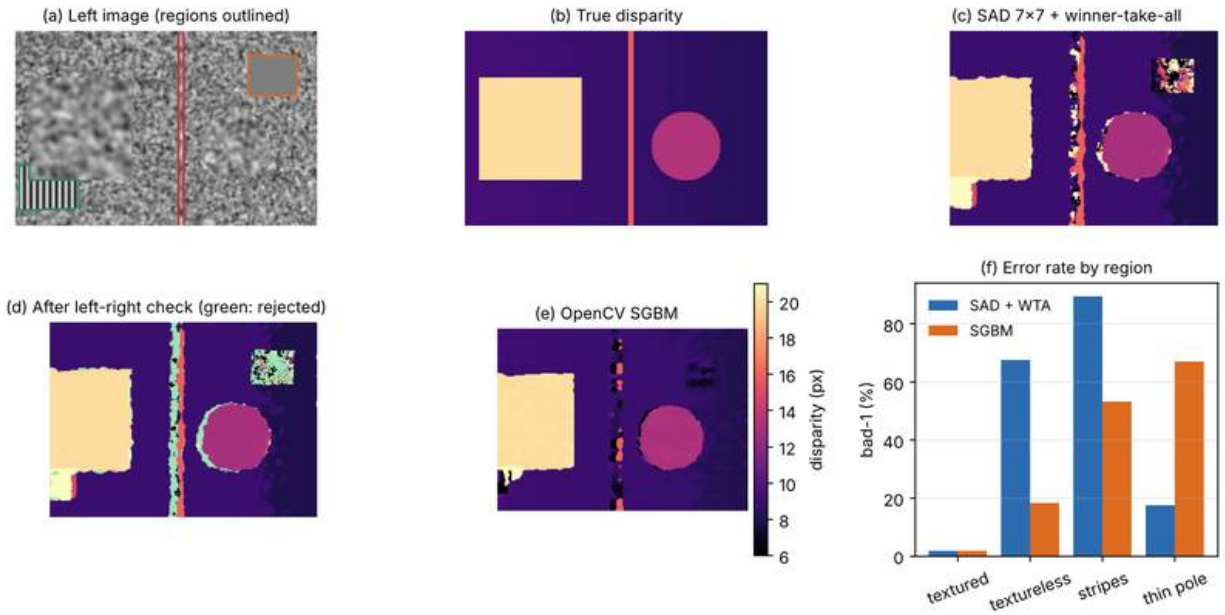


圖 4 — 在合成校正影像對上的區塊匹配。(*c*) 7 × 7 SAD 加贏者全拿；(*d*) 左右一致性檢查之後（被剔除的像素為綠色）；(*e*) OpenCV 的半全域匹配器；(*f*) 程式碼算出的各區域 *bad-1*。最左邊的窄帶沒有對應的像素，不納入評估。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

代價聚合與半全域匹配

白話版。視窗匹配讓每個像素自己做決定。比較好的規則是讓相鄰的像素「投票」：空白區域中的像素，應該採用它有紋理的鄰居的視差，除非有充分的證據顯示這裡有深度跳變。

傳統的說法是對整張視差圖 D 寫一個能量函數，

$$E(D) = \sum_{\mathbf{p}} C(\mathbf{p}, D(\mathbf{p})) + \sum_{(\mathbf{p},\mathbf{q})} P_1 \,[\,|D(\mathbf{p}) - D(\mathbf{q})| = 1\,] + P_2 \,[\,|D(\mathbf{p}) - D(\mathbf{q})| > 1\,],$$

其中第二個總和遍歷相鄰的像素對 $\mathbf{p}, \mathbf{q}$，$[\cdot]$ 在條件成立時為 1、否則為 0，P_1 是視差改變一級（傾斜表面）時的小懲罰，$P_2 > P_1$ 是視差跳變（深度邊緣）時較大的懲罰 [5], [6]。要在二維影像上精確地最小化這個能量，一般而言是不可行的。

Hirschmüller 的**半全域匹配**（semi-global matching, SGM）[5] 近似地做這件事：沿著穿過每個像素的許多條一維直線路徑最小化能量，每條路徑都能用動態規劃精確求解，再把結果加總。沿著方向為 $\mathbf{r}$ 的路徑，聚合後的代價為

$$L_{\mathbf{r}}(\mathbf{p}, d) = C(\mathbf{p}, d) + \min\Big(L_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, d),\; L_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, d \pm 1) + P_1,\; \min_k L_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, k) + P_2\Big) - \min_k L_{\mathbf{r}}(\mathbf{p}-\mathbf{r}, k),$$

其中 $\mathbf{p} - \mathbf{r}$ 是路徑上的前一個像素，最後一項只是讓數字維持有界 [6]。所有方向的代價加總後，再用 WTA 選出視差，因此每個像素都從多個方向的遠處接收到證據，而計算成本與代價體的大

小成線性關係。OpenCV 的 `StereoSGBM` 是修改過的版本：預設用 5 個方向，而不是完整演算法的 8 個；用小區塊，而不是單一像素；用 Birchfield–Tomasi 代價，而不是互資訊 [7]。

這段程式碼展示了 SGM 的取捨。在無紋理的區塊上，錯誤像素從 67.7% 降到 18.2%，因為平滑項把牆的視差帶進了區塊內。但寬度約五個像素的細柱大部分被抹掉了（17.4% → 67.2%）：它要付兩次 P_2 的跳變，卻只換到很少的證據（圖 4e）。兩種方法都解不開條紋，這是資料本身真正的歧義。平滑先驗是在賭世界由大片表面組成，而這個賭注在細長結構上會輸。

僅線上版：半全域匹配一步一步來 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/notes/半全域匹配一步一步來/*

精修：子像素視差與後處理

WTA 回傳整數視差，但圖 2b 已顯示，不到一個像素的小數部分也很重要。常見的修正是用拋物線穿過最佳整數視差及其兩個鄰居的代價，取拋物線的最小值。其他後處理步驟包括左右一致性檢查、中值濾波器，以及從鄰居填補被剔除的像素；例如 MC-CNN 的流程就結合了交叉式聚合、SGM、左右一致性檢查、子像素強化、中值濾波器與雙邊濾波器 [6]。

僅線上版：子像素視差與 *soft argmin* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/notes/子像素視差與-soft-argmin/*

學習式立體匹配

白話版。傳統流程的每一步都是手工規則。深度立體匹配一次一個地把它們換成學習出來的規則：先是代價，接著是聚合，最後是在大型合成資料集上端到端訓練的整條流程。

Laga 等人 [8] 與 Tosi 等人 [9] 的綜述，順序都和這一節相同：先是流程中被學習的部分，再是端到端網路，然後是 2020 年代的架構。

學習匹配代價

Žbontar 與 LeCun 的 MC-CNN [6] 訓練一個卷積網路，判斷兩個小區塊是否拍到同一個點，用由真實視差建出的區塊對作為二元分類資料集。網路的輸出取代 SAD 成為匹配代價，傳統流程的其餘部分（交叉式聚合、SGM、左右一致性檢查、子像素強化、濾波）接著執行。教訓是：學習出來的相似度，比原始強度更有辨別力得多，但手工設計的聚合仍然留著。

端到端的代價體網路

端到端訓練需要許多影像的密集真值，而真實掃描很少提供。Mayer 等人 [10] 算繪了三個資料集（FlyingThings3D、Monkaa、Driving）；光是 FlyingThings3D 就有約 25,000 個立體影格，內容是沿著隨機三維路徑飛行的隨機物體，他們用它訓練了 DispNet，一個迴歸視差的編碼器–解碼器 [10]。這些「Scene Flow」資料集合起來，成為立體網路常用的預訓練資料。

Kendall 等人的 **GC-Net** [11] 讓代價體變得明確且可學習。一個共用的 2D CNN 從兩張影像擷取特徵。對每個視差級別，左特徵會與位移了該級別的右特徵**串接**，得到大小為高 × 寬 ×（最大視

差 + 1）× 特徵維度的 4D 體 [11]。對這個體做 3D 卷積，學習聚合，同時利用空間與視差上的情境。最後，**soft argmin** 把正則化後的代價 c_d 變成子像素視差：

$$\hat{d} = \sum_{d=0}^{D_{\max}} d \cdot \sigma(-c_d), \qquad \sigma(-c_d) = \frac{e^{-c_d}}{\sum_{d'} e^{-c_{d'}}},$$

其中 σ 是沿視差維度的 softmax，$D_{\max}$ 是最大視差 [11]。它和 argmin 不同，是可微分的，所以網路可以對 $\hat{d}$ 用迴歸損失做端到端訓練。**PSMNet** [12] 在代價體之前加入一個空間金字塔池化模組，在多個尺度上收集情境，並加入堆疊的沙漏式 3D CNN 與中間監督來正則化代價體，用 smooth L1 損失訓練 [12]。兩者針對的都是病態區域，例如遮擋、重複圖案、無紋理與反光的表面，在這些地方局部區塊給不出答案 [12]。

3D 卷積的代價是記憶體與計算量都正比於 $D \times H \times W$，所以這些網路只能在降低的解析度與固定的最大視差下運作。

迭代精修與零樣本立體匹配

RAFT-Stereo [13] 把 RAFT 光流網路（在下方的光流一節說明）改造成適用於校正過的立體影像：它把 RAFT 的 4D 全對體換成只比較同一列像素的 3D 體，並用循環單元逐步精修視差場，這些單元在目前的估計附近查詢該體，還加入多層級 GRU，更有效率地把資訊傳遞到整張影像 [13]。沒有 3D 卷積，也沒有需要正則化的固定視差範圍。Tosi 等人稱這個設計是「game changer」，指出它對領域偏移的穩健性大幅提高，而且大多數近期的框架都跟隨它 [9]。IGEV-Stereo [14] 結合了這兩種想法：一個編碼幾何與情境的幾何編碼體，為循環更新提供良好的起點，並在每次迭代中被索引 [14]。

這些綜述強調的未解問題是**泛化**：在一個基準上微調的網路，常在另一個基準上失效 [9], [15]。FoundationStereo [16] 以零樣本使用為目標：它在一個約一百萬對立體影像的大型合成資料集上訓練，並自動移除有歧義的樣本，還透過 side-tuning 骨幹網路，改編單目視覺基礎模型的特徵，以縮小合成影像與真實影像之間的差距 [16]。

單目深度估計

白話版。只有一張影像時，沒有東西可以比對。網路必須像人閉上一隻眼睛那樣辨認深度：靠透視、物體的已知大小、遮擋、陰影、霧氣，以及越遠越細緻的紋理。這些線索能告訴你哪些東西比較近，卻分不出玩具車和真車。

為什麼單張影像不夠：尺度歧義

把整個場景縮放 s 倍，影像不會改變，因為投影會除以深度（圖 5a；另見 DLCV 11，以及三維偵測的 DLCV 03B）。Eigen 等人把全域尺度稱為「深度預測中的根本歧義」，並指出他們的大部分誤差單單來自平均深度 [17]。

網路仍然可以學到公制線索，例如物體的典型大小，或行車資料集的相機高度，但相機或場景類型一變，這些線索就失效。因此會用到兩種輸出：

- **相對（仿射不變）深度**。預測只對到一個未知的尺度與平移。許多這類模型，包括 MiDaS 與 Depth Anything，預測的是逆深度（類似視差的值）[18], [19]，所以輸出 $\hat{y}$ 滿足 $1/Z \approx s\hat{y} + t$，其中 $s > 0$ 與 t 未知。搭配少數幾個公制測量時，這對排序、分割、編輯與許多機器人任務就夠用了。
- **公制深度**。預測以公尺為單位。這需要已知的相機，或一個能從影像估計相機的模型。

忽略尺度與平移的損失函數

要在深度只知道到某個尺度、或單位不同的資料集上訓練，損失函數必須忽略那些未知的因子。Eigen 等人提出了**尺度不變對數誤差**（scale-invariant log error）[17]：

$$D(y, y^*) = \frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log y_i^* + \alpha(y, y^*))^2, \qquad \alpha(y, y^*) = \frac{1}{n}\sum_{i=1}^{n}(\log y_i^* - \log y_i),$$

其中 y_i 與 y_i^* 是像素 i 的預測與真實深度，n 是有真值的像素數，α 是讓兩張圖最佳對齊的對數尺度偏移；把 y 乘上任何常數，D 都不變 [17]。訓練時，他們用這個誤差與一般對數空間 L2 誤差的混合，權重 $\lambda = 0.5$，他們發現這能給出良好的絕對尺度預測 [17]。

Ranftl 等人的 **MiDaS** [18] 更進一步：它預測逆深度（視差）到尺度與平移，並在計算損失之前，先把每個預測對齊到真值。令 $\mathbf{d} \in \mathbb{R}^M$ 為 M 個有效像素上的預測視差，$\mathbf{d}^*$ 為真值，對齊要解

$$(s, t) = \arg\min_{s,t}\sum_{i=1}^{M}(s d_i + t - d_i^*)^2, \qquad \hat{d}_i = s d_i + t,$$

這是一個直線的普通最小平方擬合，有封閉解 [18]。**尺度與平移不變損失**（scale-and-shift-invariant loss）則是 $\mathcal{L}_{\text{ssi}} = \frac{1}{2M}\sum_i \rho(\hat{d}_i - d_i^*)$，其中 ρ 是逐像素的損失，這裡取平方誤差 [18]。MiDaS 還有穩健版本（以中位數為基礎的對齊、修剪的絕對值損失）、一個讓深度邊緣保持銳利的梯度匹配項，並混合了五個訓練資料集，其中一個取自 3D 電影 [18]。它以**零樣本跨資料集轉移**來評估，也就是在訓練時從未見過的資料集上測試 [18]。

僅線上版：*MiDaS（Ranftl 等人，2022）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/papers/midasranftl-等人 2022/*

這段程式碼建立一張玩具深度圖（地面、遠處的牆與一個箱子），模擬一個輸出為逆深度、帶有未知尺度與平移的相對模型，並比較三種解讀它的方式。

```
import numpy as np

rng = np.random.default_rng(1)
H, W, f, cam_h = 120, 160, 150.0, 1.5
v, u = np.mgrid[0:H, 0:W].astype(float)
Z = np.full((H, W), 30.0)                                   # far wall
below = v > H / 2 + 1
Z[below] = np.minimum(30.0, f * cam_h / (v[below] - H / 2))    # ground plane seen by a
 ↪  camera 1.5 m high
box = (np.abs(u - 50) < 15) & (v > 45) & (v < 80)
Z[box] = 5.0                                                # an object 5 m away
```

```
# A "relative" network outputs inverse depth up to an unknown scale and shift, plus some
 ↪  noise.
pred = 0.37 / Z + 0.05 + rng.normal(0, 0.001, Z.shape)

def metrics(Zp, Zt):
    absrel = np.mean(np.abs(Zp - Zt) / Zt)
    rmse = np.sqrt(np.mean((Zp - Zt) ** 2))
    delta1 = np.mean(np.maximum(Zp / Zt, Zt / Zp) < 1.25)
    return f"AbsRel {absrel:6.3f}   RMSE {rmse:6.2f} m   delta<1.25 {100 * delta1:5.1f}%"

Z_raw = 1 / pred                                              # read the output as if it
 ↪  were metric
Z_med = Z_raw * np.median(Z) / np.median(Z_raw)               # median scaling (scale only)
A = np.c_[pred.ravel(), np.ones(pred.size)]                   # least squares for (s, t) in
 ↪  inverse depth
s, t = np.linalg.lstsq(A, (1 / Z).ravel(), rcond=None)[0]
Z_ssi = 1 / (s * pred + t)
print("raw output        ", metrics(Z_raw, Z))
print("median scaling    ", metrics(Z_med, Z))
print("scale+shift (LSQ) ", metrics(Z_ssi, Z), f"  s = {s:.3f}, t = {t:.4f}")
```

```
raw output         AbsRel  0.460   RMSE  10.46 m   delta<1.25  10.2%
median scaling     AbsRel  0.785   RMSE   7.24 m   delta<1.25  54.8%
scale+shift (LSQ)  AbsRel  0.043   RMSE   1.84 m   delta<1.25  99.5%   s = 2.699, t = -0.1348
```

這些是玩具數字。把輸出當成公制深度來讀，它毫無用處（只有 10% 的像素 $\delta < 1.25$）。**中位數縮放**（median scaling）把深度乘上一個係數；它改善了 δ，卻讓 AbsRel 變得更糟，因為沒有任何單一係數能抵消逆深度中的平移。最小平方擬合求得 $s = 2.699$、$t = -0.1348$，幾乎正好就是模擬時設定的 $1/Z = 2.703\,\hat{y} - 0.135$，深度圖也變得幾近完美。同一個預測，視對齊方式而定，可以看起來很差，也可以看起來很棒，所以讀論文時一定要確認它用的是哪一種。

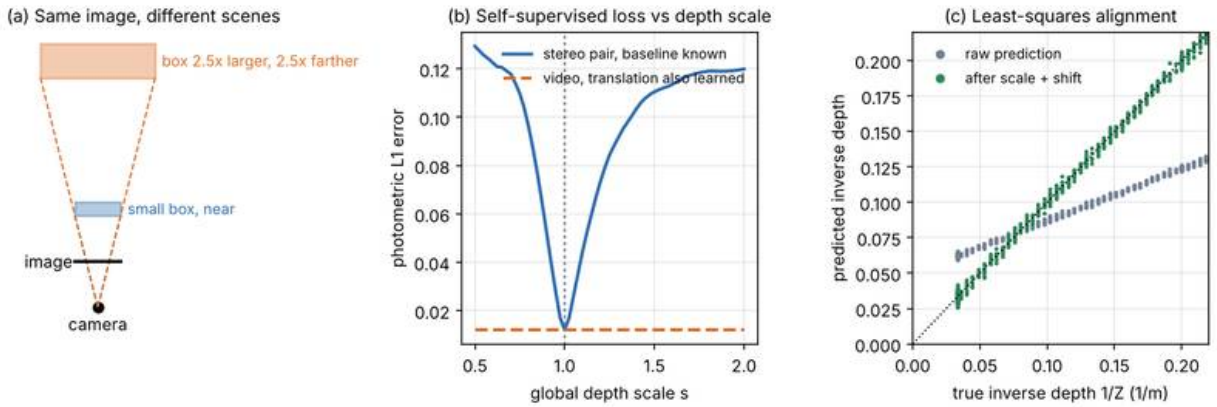


圖 *5* — 尺度歧義。(*a*) 大 *2.5* 倍、也遠 *2.5* 倍的箱子，會拍出同樣的影像。(*b*) 自監督的光度損失，只有在基線已知時才能釘住尺度；如果相機平移也是學出來的，每個尺度都同樣吻合。(*c*) 仿射不變的預測，在逆深度上做最小平方尺度與平移之後會變得準確（資料來自上面的程式碼）。

從相對深度到公制深度的基礎模型

現代的單目模型使用標準的密集預測網路；進步來自訓練資料，以及處理尺度的方式。在**相對深度**這一邊，DPT [20] 把 Vision Transformer 編碼器帶進密集預測，Depth Anything [19], [21] 用數千萬張偽標記影像擴大訓練，Marigold [22] 則把影像擴散模型改造成深度估計器。在**公制深度**這一邊，ZoeDepth [23] 在相對模型上加入公制預測頭，而 Metric3D [24], [25]、UniDepth [26] 與 Depth Pro [27] 則明確處理相機，做法是把影像映射到標準焦距，或從影像估計相機。要記住的一個想法是：相對深度現在已經能穩健地用在許多種影像上，而一旦模型知道、或能推斷出相機，公制深度就變得可用。DUSt3R 與 VGGT 這類同時預測幾何與相機的密集點圖模型，在 DLCV 11，前饋式重建中介紹。

僅線上版：單目深度基礎模型：資料與相機 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/notes/單目深度基礎模型資料與相機/*

從立體對與影片做自監督深度

白話版。不用真值深度，改用第二個視角當老師。如果網路的深度是對的，你就能把第二個視角扭曲到第一個視角上，得到同一幅畫面。扭曲後的影像與真實影像之間的差異，就是訓練訊號。

以視角合成作為監督

Garg 等人 [28] 用相機位移已知且很小的立體影像對，訓練單目網路：左影像的預測深度，加上已知的基線，把右影像扭曲到左視角，這個重建的光度誤差就是損失，完全不需要深度標籤 [28]。Godard 等人 [29] 加入**左右一致性**（left–right consistency）損失，讓為兩個視角預測的視差彼此一致，改善了品質 [29]。Zhou 等人 [30] 拿掉了第二台相機：他們用無標記的單目影片，同時訓練一個深度網路和一個**位姿網路**（pose network），唯一的監督是相鄰影格之間的視角合成 [30]。

對目標影格 I_t 與來源影格 $I_{t'}$，每個目標像素 $\mathbf{p}$（以齊次座標表示）會被映射到來源影格：

$$\mathbf{p}' \sim K\, T_{t\to t'}\, D_t(\mathbf{p})\, K^{-1}\mathbf{p},$$

其中 K 是相機內參矩陣，$D_t(\mathbf{p})$ 是該像素的預測深度，$T_{t\to t'}$ 是預測的相對相機位姿（旋轉與平移），$\sim$ 表示在齊次尺度之下相等。用雙線性取樣器（DLCV 09A.3）在 $\mathbf{p}'$ 處對 $I_{t'}$ 取樣，就得到合成影像 $I_{t'\to t}$；梯度會穿過它，流進兩個網路。

Monodepth2

Godard 等人的 Monodepth2 [31] 指出，一個謹慎做出設計選擇的簡單模型，勝過更複雜的模型。它的兩張影像之間的光度誤差，結合了 SSIM（DLCV 09，全參考指標）與 L1：

$$\mathrm{pe}(I_a, I_b) = \frac{\alpha}{2}\big(1 - \mathrm{SSIM}(I_a, I_b)\big) + (1-\alpha)\|I_a - I_b\|_1, \qquad \alpha = 0.85,$$

其中 I_a, I_b 是目標影像與一張合成影像，逐像素比較 [31], [32]。三個想法讓它很穩健 [31]：

1. **逐像素最小重投影**。有多個來源影格（前一張與後一張）時，取各來源的誤差最小值，而不是平均，即 $\mathcal{L}_p = \min_{t'} \mathrm{pe}(I_t, I_{t'\to t})$，因為一個在某個來源被遮擋的像素，通常在另一個來源看得見。
2. **自動遮罩**（auto-masking）。忽略「不扭曲時的誤差已經比扭曲後更低」的像素。這能排除靜止的相機，以及隨相機一起移動的物體，否則網路會從前方行駛的汽車學到「無限遠的深度」。
3. **全解析度多尺度損失**。在計算誤差之前，先把較低解析度的深度預測上取樣到輸入尺寸，以減少瑕疵。

一個考慮邊緣的平滑項，權重為 $\lambda = 0.001$，用來抑制影像本身平滑處的深度變化 [31]。

再談尺度

從單目影片訓練時，深度網路從未看到任何公制量：深度與平移可以一起縮放，而不改變 $\mathbf{p}'$。圖 5b 在合成立體場景上顯示了這一點：基線固定時，光度損失在真實尺度處有尖銳的最小值；如果平移跟著深度一起縮放，損失就是平的。這就是為什麼 Monodepth2 對單目訓練的結果，是在經過逐張影像的**真值中位數縮放**後才回報（沿用 SfMLearner 的做法），但對立體訓練的結果則不這麼做，因為「尺度可由已知的相機基線推得」[31]。其他限制則是匹配本身的限制：無紋理與反光的表面、移動的物體與遮擋。

把光流當作學習問題

白話版。光流就是沒有規則手冊的立體匹配：配對可能往任何方向移動，可能是不到一個像素，也可能是一百個像素，而且物體各自獨立運動。傳統方法為每一對影像解一個能量函數；深度方法則從合成影片學習如何匹配與精修。

從能量函數到網路

傳統方法從亮度恆定 $I_1(\mathbf{p}) = I_2(\mathbf{p} + \mathbf{F}(\mathbf{p}))$ 出發，它對每個像素的兩個未知數只給一條方程式，所以要再加一個假設才能解開：流場的全域平滑性（Horn 與 Schunck [33]）、小視窗內流為常數並以最小平方求解（Lucas 與 Kanade [34]），或影像的局部多項式模型（Farnebäck [35]）。DLCV 04B 介紹了這些想法與孔徑問題（aperture problem）。

選讀：亮度恆定與傳統光流求解器（見 *DLCV 04B* 的「補充資料」頁）。線性化的光流方程式、*Horn–Schunck* 能量函數，以及 *Farnebäck* 方法背後的多項式展開。

Hur 與 Roth [36] 把轉向深度學習的過程描述成兩個家族：把 CNN 當作特徵擷取器，放在傳統能量公式之中；以及直接迴歸光流的端到端網路。第二個家族勝出了。有四個里程碑通往 RAFT：

- **FlowNet** [37] 用 CNN 端到端地迴歸光流。由於缺乏真實的訓練資料，作者建立了 **Flying Chairs**，這是 22,872 對影像，內容是算繪的 3D 椅子模型，以隨機仿射轉換移動在照片背景之上，他們發現用它訓練的網路仍然能泛化到 Sintel 與 KITTI [37]。它的 FlowNetC 變體加入了**相關層**（correlation layer），在一系列位移下比較兩張影像的特徵。
- **FlowNet 2.0** [38] 堆疊了數個 FlowNet，每個都用目前的估計扭曲第二張影像，並顯示訓練資料集的順序影響很大 [38]。

- **SPyNet** [39] 使用傳統的影像金字塔：在每一層，其中一張影像被目前的光流扭曲，再由一個小網路預測不到一個像素的更新 [39]。
- **PWC-Net** [40] 在學習出來的特徵上結合**金字塔**（pyramid）、**扭曲**（warping）與**代價體**（cost volume），而且比 FlowNet2 小 17 倍 [40]。

RAFT：全對相關與迭代更新

白話版。 RAFT 把第一張影像的每個像素和第二張影像的每個像素比較，只做一次。然後它從零運動出發，像一個人在修改草圖那樣，反覆查看目前猜測附近的匹配分數，再把猜測微調一點；每次都是同一個小更新，一再重複。

Teed 與 Deng 的 RAFT [41] 有三個部分 [41]：

1. **特徵**。 CNN 編碼器把兩張影像映射成 1/8 解析度、$D = 256$ 個通道的特徵；另有一個獨立的情境網路（context network），只編碼第一張影像。
2. **全對相關**。相關體裝的是每一對特徵的內積，

$$C_{ijkl} = \sum_{h} g(I_1)_{ijh}\, g(I_2)_{klh},$$

其中 $g(I)_{ijh}$ 是位於第 i 列、第 j 欄的特徵的第 h 個通道，$C \in \mathbb{R}^{H\times W\times H\times W}$ 是在特徵解析度下的大小。對最後兩個維度用核大小 1、2、4、8 做平均池化，就得到四層的**相關金字塔**（correlation pyramid），它對第一張影像保持完整解析度，並用較粗的層涵蓋大位移。3. **迭代更新**。從 $\mathbf{f}_0 = \mathbf{0}$ 開始，一個卷積 GRU（閘控循環單元）反覆讀取每個像素目前所指位置周圍小鄰域（半徑 $r = 4$）內、所有金字塔層級的相關值，再加上情境特徵與目前的光流，輸出一個更新：$\mathbf{f}_{k+1} = \mathbf{f}_k + \Delta\mathbf{f}$。各次迭代共用權重。學習出來的凸組合上取樣（convex upsampling）把 1/8 解析度的光流變成完整解析度。

損失對每一次迭代都做監督，權重隨迭代次數指數增加，

$$\mathcal{L} = \sum_{i=1}^{N} \gamma^{N-i}\, \|\mathbf{f}_{\text{gt}} - \mathbf{f}_i\|_1, \qquad \gamma = 0.8,$$

其中 $\mathbf{f}_i$ 是第 i 次更新後上取樣的光流，N 是更新次數，$\mathbf{f}_{\text{gt}}$ 是真值 [41]。RAFT 在訓練時展開 12 次更新，測試時可以跑更多次；論文在 Sintel 上用 32 次更新、在 KITTI 上用 24 次更新評估 [41]。訓練遵循標準排程：先 Flying Chairs，再 FlyingThings3D，然後在目標基準上微調 [41]。

這個設計為什麼贏？全對體避開了由粗到細方案的弱點，也就是小而快的物體可能在粗層級就永遠消失 [41]；共用的更新表現得像一個學習出來的最佳化器；而這套做法也轉移到了立體匹配 [13] 與場景流 [42]。

僅線上版：*RAFT（Teed 與 Deng，2020）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/papers/raftteed-與-deng2020/*

RAFT 之後

後來的模型各自改變這套做法的一個部分。GMFlow [43] 與 Unimatch [44] 用 Transformer 特徵的**全域匹配**取代迭代查表：每個像素都和另一張影像的所有像素比較，光流是 softmax 加權的平均位置；Unimatch 用同一個模型做光流、立體匹配與有位姿影像的深度 [44]。FlowFormer [45] 用 Transformer 層編碼 RAFT 的代價體。SEA-RAFT [46] 保留 RAFT，但加入 mixture-of-Laplace 損失、直接迴歸的初始光流與剛體運動預訓練，以求更快的速度與更好的泛化 [46]。

程式碼：量測光流誤差

這段程式碼建立一對已知光流的影格：背景（camera 測試影像）繞中心旋轉 2° 並放大 3%，一塊有紋理的區塊疊在上面，移動 (14, 6) 像素。它用一層與四層金字塔執行 OpenCV 的 Farnebäck 光流，並對不同種類的像素回報**端點誤差**（end-point error, EPE，誤差向量的長度）與 KITTI 使用的 **Fl** 離群值比例（定義見評估）。

```
import numpy as np
import cv2
from scipy import ndimage as ndi
from skimage import color, data, transform

I0 = transform.resize(data.camera() / 255.0, (240, 320), anti_aliasing=True)
H, W = I0.shape
rows, cols = np.mgrid[0:H, 0:W].astype(float)
cy, cx = H / 2, W / 2

# Ground-truth motion from frame 0 to frame 1: the background rotates by 2 degrees and zooms
 ↪  by 3 %
# about the image centre; a 50 x 50 textured patch moves by (dx, dy) = (14, 6) pixels.
a, s = np.deg2rad(2.0), 1.03
bx = cx + s * (np.cos(a) * (cols - cx) - np.sin(a) * (rows - cy))     # where each background
 ↪  pixel goes
by = cy + s * (np.sin(a) * (cols - cx) + np.cos(a) * (rows - cy))
flow = np.stack([bx - cols, by - rows], -1)
patch = transform.resize(color.rgb2gray(data.astronaut())[30:230, 150:350], (50, 50),
 ↪  anti_aliasing=True)
obj0 = (rows >= 90) & (rows < 140) & (cols >= 120) & (cols < 170)
flow[obj0] = (14.0, 6.0)

# Render frame 1 by backward mapping (inverse rotation and zoom), then paste the moved patch.
xs = cx + (np.cos(a) * (cols - cx) + np.sin(a) * (rows - cy)) / s
ys = cy + (-np.sin(a) * (cols - cx) + np.cos(a) * (rows - cy)) / s
I1 = ndi.map_coordinates(I0, [ys, xs], order=3, mode="reflect")
I0 = I0.copy(); I0[90:140, 120:170] = patch
I1[96:146, 134:184] = patch
# occluded: background points of frame 0 that land under the patch in frame 1
occ = ~obj0 & (by >= 96) & (by < 146) & (bx >= 134) & (bx < 184)

def evaluate(est, mask):
    epe = np.linalg.norm(est - flow, axis=-1)[mask]
    mag = np.linalg.norm(flow, axis=-1)[mask]
```

```
    fl = np.mean((epe > 3) & (epe > 0.05 * mag))       # KITTI outlier: > 3 px and > 5 % of
     ↪  the true motion
    return f"EPE {epe.mean():5.2f} px  Fl {100 * fl:5.1f}%"

u8 = lambda im: (np.clip(im, 0, 1) * 255).round().astype(np.uint8)
inner = np.zeros((H, W), bool); inner[10:-10, 10:-10] = True
grad = ndi.gaussian_filter(np.hypot(ndi.sobel(I0, 0), ndi.sobel(I0, 1)), 4)
bg = inner & ~obj0 & ~occ
flat = grad < 0.05                                     # almost no texture (mostly the sky)
for levels in [1, 4]:
    est = cv2.calcOpticalFlowFarneback(u8(I0), u8(I1), None, pyr_scale=0.5, levels=levels,
     ↪  winsize=15,
                                       iterations=5, poly_n=7, poly_sigma=1.5, flags=0)
    print(f"Farneback, {levels} pyramid level(s):")
    for name, m in [("background, textured", bg & ~flat), ("background, flat", bg & flat),
                    ("moving patch", obj0), ("occluded", occ), ("all (Fl-all)", inner)]:
        print(f"   {name:21s} {evaluate(est, m)}")
print(f"true motion: background up to {np.linalg.norm(flow[~obj0], axis=-1).max():.1f} px,
 ↪  patch "
      f"{np.hypot(14, 6):.1f} px; occluded pixels: {occ.sum()}")
```

```
Farneback, 1 pyramid level(s):
   background, textured  EPE  0.87 px  Fl   9.3%
   background, flat      EPE  5.51 px  Fl  90.3%
   moving patch          EPE 18.24 px  Fl 100.0%
   occluded              EPE  5.07 px  Fl  68.0%
   all (Fl-all)          EPE  2.86 px  Fl  35.6%
Farneback, 4 pyramid level(s):
   background, textured  EPE  1.18 px  Fl  11.9%
   background, flat      EPE  5.52 px  Fl  90.5%
   moving patch          EPE  2.30 px  Fl  30.4%
   occluded              EPE 12.20 px  Fl  99.8%
   all (Fl-all)          EPE  2.55 px  Fl  35.2%
true motion: background up to 9.3 px, patch 15.2 px; occluded pixels: 871
```

這些數字只來自這對玩具影像。只用一層時，區塊 15 px 的運動超出能力範圍，區塊上每個像素都是離群值；用四層時找到了區塊（EPE 2.30 px），這是 SPyNet 與 PWC-Net 由粗到細想法的傳統版本。仍有兩種失敗。在平坦區域（主要是天空），90% 的像素是離群值：根本沒有東西可以匹配。在**被遮擋**的像素上，也就是區塊在第 1 張影格中蓋住的那一條帶，金字塔反而讓情況更糟（EPE 12.2 px），因為粗層級把它們跟著區塊一起拖走（圖 6）。學習式方法被訓練成能從資料中處理這兩種情況，但兩者都不會消失。

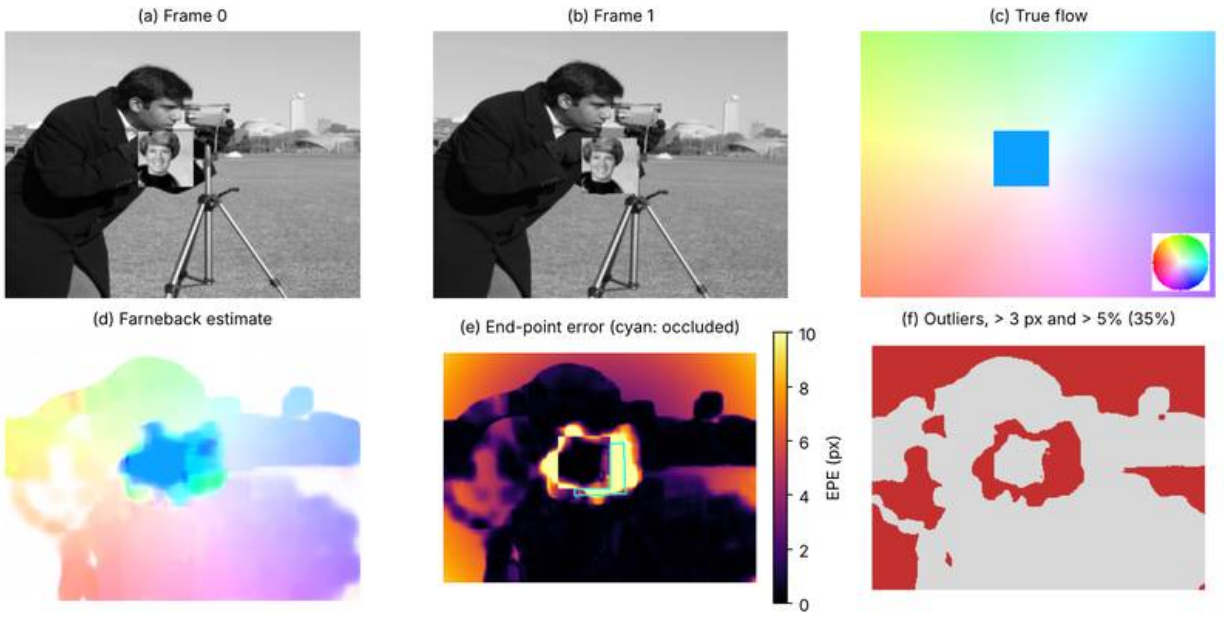


圖 *6* — 玩具影像對上的 *Farnebäck* 光流（*4* 層金字塔）。光流以色相表示方向、以飽和度表示長度。誤差集中在無紋理的區域、移動區塊的周圍，以及被遮擋的條帶（青色）。*10 px* 的邊界不納入評估。

場景流（簡介）

白話版。光流說的是一個點在影像中怎麼動。場景流說的是它在三維中怎麼動：變近、變遠、往左或往右，以公尺計。

場景流（scene flow）是場景的密集三維運動場，這個詞可追溯到 Vedula 等人 [47]；光流是它在影像上的投影。RAFT-3D 把任務表述為：給定一對立體或 RGB-D 影片影格，估計每個像素的三維運動 [42]。用立體相機時，可以由兩個時刻的視差，加上兩者之間的光流來估計。Menze 與 Geiger 的 KITTI 2015 基準 [48] 評估的正是這幾個部分：D1（第一影格的視差）、D2（第二影格的視差）、Fl（光流），以及 SF，只要這三者中任何一個錯誤，該像素就算錯 [49]。Menze 與 Geiger 把場景建模為少數幾個剛性移動的物體，每個超像素是一個指派給其中一個物體的 3D 平面 [48]；RAFT-3D [42] 則改用 RAFT 式的迭代，估計剛性三維運動的密集場，並以學習出來的嵌入把像素軟分組成剛性物體 [42]。動態重建（DLCV 11）與點追蹤（DLCV 07）是它的長距離親戚。

評估

白話版。三項任務都是逐像素對照真值打分，用平均誤差與「錯誤像素的百分比」。細節，尤其是單目預測如何對齊、哪些像素要計算，對數字的影響比你想像的更大。

深度指標

標準的一組指標可追溯到 Eigen 等人 [17]。令 y_i 為預測深度、y_i^* 為真實深度，共 n 個被評估的像素，

$$\text{AbsRel} = \frac{1}{n}\sum_i \frac{|y_i - y_i^*|}{y_i^*}, \qquad \text{RMSE} = \sqrt{\frac{1}{n}\sum_i (y_i - y_i^*)^2},$$

$$\text{SqRel} = \frac{1}{n}\sum_i \frac{(y_i - y_i^*)^2}{y_i^*}, \qquad \delta_k = \frac{1}{n}\left|\left\{ i : \max\!\left(\tfrac{y_i}{y_i^*}, \tfrac{y_i^*}{y_i}\right) < 1.25^k \right\}\right|,$$

另外還有 $\log y$ 的 RMSE [17]。AbsRel、SqRel 與 RMSE 是誤差（越低越好）；$k = 1, 2, 3$ 的 $\delta_1, \delta_2, \delta_3$ 是準確率（越高越好）。AbsRel 以相對誤差衡量近處與遠處的像素；RMSE 由遠處的像素主導；δ_1 計算落在真值 25% 以內的像素。

每一個結果都要說明三個慣例：**對齊**（公制模型不對齊，自監督單目模型做逐張影像的中位數縮放 [31]，仿射不變模型做最小平方尺度與平移 [18]）；**深度上限**（KITTI 上標準是 80 m [31]）；以及**切分**（KITTI 上常用 Eigen 等人的切分 [17], [31]）。對齊的程式碼已顯示光是第一項就能改變多少。

立體匹配指標

- **EPE**（平均絕對視差誤差，單位為像素）。
- **bad-τ**：視差誤差大於 τ 像素的像素百分比。Middlebury 的第 3 版評估預設回報 bad 2.0，分成全部像素或只算未被遮擋的像素 [3]；KITTI 2012 回報誤差像素的百分比，門檻從 2 到 5 像素，分成未被遮擋區域（Out-Noc）與全部（Out-All）[50]。
- **D1**（KITTI 2015）：一個像素的視差誤差低於 3 px 或低於真值的 5%，就算正確；D1 是離群值的百分比，可對背景、前景或全部像素回報（D1-bg、D1-fg、D1-all）[49]。後面那個相對條件，避免了視差很大的近處物體被評得太嚴。

光流指標

- **EPE**：估計與真實光流向量之間的平均歐氏距離，$\frac{1}{n}\sum_i \|\mathbf{F}_i - \mathbf{F}_i^*\|_2$，其中 $\mathbf{F}_i$ 與 $\mathbf{F}_i^*$ 是像素 i 的估計與真實光流；FlowNet 稱它是光流的標準誤差量測，也用它來訓練 [37]。Sintel 的結果以平均 EPE（常寫成 AEPE）回報。
- **Fl-all**（KITTI 2015）：端點誤差至少 3 px 且至少為真實光流長度 5% 的像素百分比，在所有有真值的像素上計算 [49]。程式碼中的 `evaluate` 函式算的正是這個。

資料集與基準

資料集	任務	內容	真值
KITTI 2012 [51], [50]	立體匹配、光流	194 對訓練與 195 對測試的真實道路場景影像	稀疏，來自雷射掃描器

資料集	任務	內容	真值
KITTI 2015 [48], [49]	立體匹配、光流、場景流	200 個訓練與 200 個測試場景，含移動的汽車	稀疏、半自動，並在移動車輛上擬合 3D CAD 模型
Middlebury 2014 [52]	立體匹配	33 個高解析度室內場景；第 3 版評估使用其中 20 個（10 個訓練、10 個測試）	密集，子像素精度
ETH3D [53]	立體匹配、多視角立體視覺	室內與室外場景，含 DSLR 影像與低解析度立體影片；包含一個低解析度雙視角基準	高精度雷射掃描
MPI Sintel [54]	光流（也含立體匹配、深度）	一部開源動畫電影的影格；有「clean」與「final」兩種通道，final 含動態模糊與大氣效果	密集，算繪產生
Flying Chairs [37]	光流（訓練）	22,872 對合成影像	密集，算繪產生
FlyingThings3D [10]	立體匹配、光流、場景流（訓練）	約 25,000 個 960 × 540 的合成立體影格	密集，算繪產生
NYU Depth v2 [55]	單目深度（室內）	464 個室內場景的 Kinect 影片；1,449 對有密集標記的 RGB-D 影像	密集，來自 Kinect 深度相機，缺值已填補

大小與真值類型取自所引用的頁面與論文；Sintel 兩種通道的描述來自 [37]，KITTI 2015 的半自動真值來自 [49], [48]。合成資料集用於預訓練，真實資料集用於微調與測試。

失效模式

白話版。本篇的每個方法都依賴兩樣東西之一：有東西可以比對，或是從訓練資料學到的先驗。下面每一種失敗，都破壞了其中之一。

- **無紋理區域**（白牆、天空、馬路）：匹配代價是平的（圖 3b），要靠平滑性或學習的情境來填補，而兩者都可能出錯。
- **重複圖案**（柵欄、磁磚）：好幾個視差都同樣吻合，正如條紋所示。
- **反光與透明的表面**：被匹配的內容是倒影或後方的場景。Tosi 等人把透明與反光物體，列為深度立體匹配四大未解挑戰之一，其餘是領域偏移、過度平滑與非對稱立體對 [9]。
- **遮擋**：不存在正確的配對；需要偵測（左右一致性檢查）與填補，而在光流中，它們會被拖著跟鄰近的運動走。
- **細長結構與深度邊緣**：視窗會讓前景物體變胖，平滑性會抹掉柱子和電線，而對雙峰代價取的 soft argmin 會落在兩個表面之間。
- **小物體的大運動**：由粗到細的方案可能把它們弄丟；RAFT 的全對體就是針對這點 [41]。
- **領域差距**：在合成資料上預訓練、在某個基準上微調的模型，面對新的相機、天氣與場景會退化 [9], [36]。
- **尺度與相機**（單目）：相對深度沒有尺度，而在焦距未知或影像被意外裁切時，公制深度會失效。
- **自監督中的移動物體**會破壞靜止場景的假設；自動遮罩只能處理最簡單的情況 [31]。Rajapaksha 等人還把細小、微小與遠處的物體，列為單目深度尚未填補的缺口 [56]。

現代觀點

綜述怎麼說

有五篇綜述涵蓋本篇的範圍。**Laga 等人** [8] 綜述了到約 2019 年為止的深度立體匹配（雙視角與多視角），分成學習傳統流程中某些部分的方法，以及端到端流程，並涵蓋資料集與監督方式；單目深度不在他們的範圍內。**Poggi 等人** [15] 強調學習與立體匹配互相幫助：學習改善立體匹配，立體匹配又監督單目網路；他們也涵蓋信心估計與領域偏移。**Tosi 等人** [9] 回顧 2020 年代：把架構分成代價體聚合、神經架構搜尋、迭代最佳化、Vision Transformer 與馬可夫隨機場，並列出四個未解挑戰（領域偏移、過度平滑、透明與反光物體、非對稱立體對）；他們在 2024 年指出，立體匹配的基礎模型當時仍然缺席。**Rajapaksha 等人** [56] 依輸入與輸出、架構與監督程度整理單目深度。**Hur 與 Roth** [36] 追溯光流從能量最小化到 CNN 的歷程，並指出領域過擬合與複雜的訓練排程是主要限制。

僅線上版：「現代觀點」綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/depth-optical-flow/papers/現代觀點綜述導讀/*

2023–2026 年的最新進展

- **單目深度的基礎模型**。大規模資料（Depth Anything [19], [21]）、生成式先驗（Marigold [22]）與明確的相機處理（Metric3D [24], [25]、UniDepth [26]、Depth Pro [27]），讓對任意照片做零樣本深度變得實用，先是相對深度，現在越來越多是公制深度。
- **立體匹配與光流的迭代匹配**。相關體加循環更新的 RAFT 做法，主導了這兩項任務 [9]；近期的變體加入幾何體（IGEV-Stereo [14]）、帶注意力的全域匹配（GMFlow、Unimatch [43], [44]），或更簡單的訓練（SEA-RAFT [46]）。
- **零樣本立體匹配**。 FoundationStereo [16] 結合一百萬對合成影像與單目先驗，回應了 Tosi 等人所說缺少基礎模型的問題。
- **與三維重建的匯流**。預測點圖與相機的前饋式模型（DLCV 11），模糊了深度估計、立體匹配與運動恢復結構之間的界線。

尚未解決的問題

1. **野外的公制深度**：未知、被裁切或不尋常的相機，以及每個預測都有可靠的不確定性。
2. **非朗伯場景**：玻璃、鏡子與水，對立體匹配與光流都是問題 [9]。
3. **銳利的邊界與細長結構**，且不能過度平滑，也不能在表面之間出現「飛點」（flying pixels）。
4. **時間一致性**：逐影格的深度在影片中會閃爍；長序列上的光流與深度應該彼此一致。
5. **評估**：稀疏或半自動的真值、不一致的對齊慣例，以及基準飽和，讓很小的差異難以解讀。

重點整理

- 校正後立體匹配限制在影像列上，光流跨時間尋找對應，單目深度則依賴學得或幾何假設。
- 視差越小，深度越敏感；校準、基線與次像素精度限制可用範圍。
- 成本聚合與正則化補足不確定區域，也可能抹去細窄結構。

- 區分相對與公尺尺度深度，並註明評估使用的尺度或平移對齊。
- 光度監督依賴可見性與場景假設，移動或遮擋內容需要另行處理。
- 報告誤差單位、評估像素、深度上限與切分；模型名稱相近不代表協定可比。

練習

1. **選擇基線**。一台 $f = 1000$ 像素的立體相機，必須在 40 m 處把深度誤差壓在 1 m 以下，假設視差誤差為 0.3 像素。最小的基線是多少？此時 40 m 處的點視差是多少？在 80 m 會發生什麼事？

   提示

   由 $\delta Z \approx Z^2 \delta d / (fB) < 1$：$B > 1600 \cdot 0.3/1000 = 0.48$ m。40 m 處的視差為 $fB/Z = 12$ px。在 80 m 處，誤差變成四倍（約 4 m），視差為 6 px。

2. **視窗大小與變胖**。在區塊匹配的程式碼中，把 `win` 改成 3、7 與 15，比較四個區域的 bad-1。為什麼大視窗在一個地方有幫助，在另一個地方卻有害？

   提示

   在我們的執行中，無紋理區域的 bad-1 從 84.4%（3×3）降到 67.7%（7×7）與 39.9%（15×15），而細柱則從 15.3% 升到 17.4% 與 74.6%。大視窗會從平坦區塊之外收集紋理，但在深度邊緣附近，它會混入兩個表面，把前景的視差擴散到背景（「前景變胖」），這就吞掉了細柱。

3. **條紋與週期**。牆上條紋的週期是 0.15 個世界單位，那裡的牆約在 8.6 單位遠處。估計它們以像素為單位的週期（取 $f = 320$ px），然後解釋為什麼區塊匹配在那裡會給出錯誤的視差。場景要怎麼改，條紋才能被匹配？演算法要怎麼改，可能有幫助？

   提示

   週期約為 $320 \cdot 0.15/8.6 \approx 5.6$ px，比那裡的真實視差（約 9.3 px）還小，所以相差一個週期的位移同樣吻合。擴大搜尋範圍沒有幫助。非週期性的紋理有用；偏好周圍牆面視差的平滑項能減少錯誤（在我們的執行中，SGBM 在條紋上的 bad-1 是 53.3%，區塊匹配是 89.6%），但無法消除。

4. **手算尺度不變損失**。證明當 y 乘上常數 $c > 0$ 時，Eigen 的誤差 $D(y, y^*)$ 不變。接著證明 MiDaS 的最小平方 (s, t) 使 $\mathcal{L}_{\text{ssi}}$ 在 $d \mapsto ad + b$（任何 $a \neq 0$ 與 b）下維持不變。

   提示

   $\log(cy_i) = \log c + \log y_i$，而 α 吸收了 $-\log c$，所以每個殘差都不變。對 MiDaS：如果 (s, t) 對 d 是最佳的，那麼 $(s/a, t - sb/a)$ 對 $ad + b$ 會給出同樣的對齊值。

5. **金字塔與大運動**。在光流的程式碼中，把區塊移動 (24, 10) 而不是 (14, 6)，分別用 1、4 與 6 層金字塔執行；然後只移動 (2, 1)。第 1 張影格中貼上的位置與遮擋遮罩也要一併更新。金字塔何時有幫助？

提示

在我們的執行中，(24, 10) 時，每種設定都弄丟了區塊（EPE 約 26 px），而 6 層的結果和 4 層完全相同，所以多出來的層在這裡沒有任何貢獻。(2, 1) 時，三種設定的區塊 EPE 都是 0.27 px。金字塔只有在運動能被它的粗層級帶到局部模型能處理的範圍內時才有幫助；超出這個範圍，就需要不同的搜尋方式，例如 RAFT 的全對相關。

參考文獻

1. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, 2004. doi
2. D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 7–42, 2002. doi
3. D. Scharstein, R. Szeliski and H. Hirschmüller, "Middlebury Stereo Evaluation — Version 3," official benchmark page (bad 2.0 metric, non-occluded and all masks). link ·stereo home page
4. R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Computer Vision — ECCV '94* (Lecture Notes in Computer Science), pp. 151–158, Springer, 1994. doi
5. H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008. doi
6. J. Žbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, no. 65, pp. 1–32, 2016. JMLR ·arXiv
7. OpenCV contributors, "cv::StereoSGBM class documentation," OpenCV source, `modules/calib3d/include/opencv2/calib3d.hpp` (4.x branch). source
8. H. Laga, L. V. Jospin, F. Boussaid and M. Bennamoun, "A survey on deep learning techniques for stereo-based depth estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 4, pp. 1738–1764, 2022. doi ·arXiv
9. F. Tosi, L. Bartolomei and M. Poggi, "A survey on deep stereo matching in the twenties," *International Journal of Computer Vision*, vol. 133, no. 7, pp. 4245–4276, 2025. doi ·arXiv
10. N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *Proc. IEEE CVPR*, pp. 4040–4048, 2016. CVF ·doi
11. A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," in *Proc. IEEE ICCV*, 2017. CVF
12. J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proc. IEEE/CVF CVPR*, 2018. CVF
13. L. Lipson, Z. Teed and J. Deng, "RAFT-Stereo: Multilevel recurrent field transforms for stereo matching," in *Proc. International Conference on 3D Vision (3DV)*, pp. 218–227, 2021. doi ·arXiv
14. G. Xu, X. Wang, X. Ding and X. Yang, "Iterative geometry encoding volume for stereo matching," in *Proc. IEEE/CVF CVPR*, 2023. CVF

15. M. Poggi, F. Tosi, K. Batsos, P. Mordohai and S. Mattoccia, "On the synergies between machine learning and binocular stereo for depth estimation from images: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. doi ·arXiv
16. B. Wen, M. Trepte, J. Aribido, J. Kautz, O. Gallo and S. Birchfield, "FoundationStereo: Zero-shot stereo matching," in *Proc. IEEE/CVF CVPR*, 2025. CVF ·arXiv
17. D. Eigen, C. Puhrsch and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proc. NeurIPS*, 2014. proceedings ·arXiv
18. R. Ranftl, K. Lasinger, D. Hafner, K. Schindler and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 3, pp. 1623–1637, 2022. doi ·arXiv
19. L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng and H. Zhao, "Depth Anything: Unleashing the power of large-scale unlabeled data," in *Proc. IEEE/CVF CVPR*, 2024. CVF ·arXiv
20. R. Ranftl, A. Bochkovskiy and V. Koltun, "Vision transformers for dense prediction," in *Proc. IEEE/CVF ICCV*, 2021. CVF
21. L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng and H. Zhao, "Depth Anything V2," in *Proc. NeurIPS*, 2024. proceedings ·arXiv
22. B. Ke, A. Obukhov, S. Huang, N. Metzger, R. C. Daudt and K. Schindler, "Repurposing diffusion-based image generators for monocular depth estimation," in *Proc. IEEE/CVF CVPR*, 2024. CVF ·arXiv
23. S. F. Bhat, R. Birkl, D. Wofk, P. Wonka and M. Müller, "ZoeDepth: Zero-shot transfer by combining relative and metric depth," arXiv:2302.12288, 2023. arXiv
24. W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen and C. Shen, "Metric3D: Towards zero-shot metric 3D prediction from a single image," in *Proc. IEEE/CVF ICCV*, 2023. CVF · arXiv
25. M. Hu, W. Yin, C. Zhang, Z. Cai, X. Long, K. Wang, H. Chen, G. Yu, C. Shen and S. Shen, "Metric3D v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 10579–10596, 2024. doi ·arXiv
26. L. Piccinelli, Y.-H. Yang, C. Sakaridis, M. Segu, S. Li, L. Van Gool and F. Yu, "UniDepth: Universal monocular metric depth estimation," in *Proc. IEEE/CVF CVPR*, 2024. CVF ·arXiv
27. A. Bochkovskii, A. Delaunoy, H. Germain, M. Santos, Y. Zhou, S. R. Richter and V. Koltun, "Depth Pro: Sharp monocular metric depth in less than a second," in *Proc. ICLR*, 2025. OpenReview ·arXiv
28. R. Garg, V. Kumar B.G., G. Carneiro and I. Reid, "Unsupervised CNN for single view depth estimation: Geometry to the rescue," in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 740–756, 2016. doi ·arXiv
29. C. Godard, O. Mac Aodha and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proc. IEEE CVPR*, 2017. CVF
30. T. Zhou, M. Brown, N. Snavely and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *Proc. IEEE CVPR*, 2017. CVF
31. C. Godard, O. Mac Aodha, M. Firman and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proc. IEEE/CVF ICCV*, 2019. CVF ·arXiv
32. Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–

612, 2004. doi
33. B. K. P. Horn and B. G. Schunck, “Determining optical flow,” *Artificial Intelligence*, vol. 17, no. 1–3, pp. 185–203, 1981. doi
34. B. D. Lucas and T. Kanade, “An iterative image registration technique with an application to stereo vision,” in *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, 1981. pdf
35. G. Farnebäck, “Two-frame motion estimation based on polynomial expansion,” in *Image Analysis* (Lecture Notes in Computer Science), pp. 363–370, Springer, 2003. doi
36. J. Hur and S. Roth, “Optical flow estimation in the deep learning age,” in *Modelling Human Motion*, N. Noceti, A. Sciutti and F. Rea, Eds., pp. 119–140, Springer, 2020. doi ·arXiv
37. A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazırbaş, V. Golkov, P. van der Smagt, D. Cremers and T. Brox, “FlowNet: Learning optical flow with convolutional networks,” in *Proc. IEEE ICCV*, 2015. CVF ·arXiv
38. E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy and T. Brox, “FlowNet 2.0: Evolution of optical flow estimation with deep networks,” in *Proc. IEEE CVPR*, 2017. CVF
39. A. Ranjan and M. J. Black, “Optical flow estimation using a spatial pyramid network,” in *Proc. IEEE CVPR*, 2017. CVF
40. D. Sun, X. Yang, M.-Y. Liu and J. Kautz, “PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume,” in *Proc. IEEE/CVF CVPR*, 2018. CVF
41. Z. Teed and J. Deng, “RAFT: Recurrent all-pairs field transforms for optical flow,” in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 402–419, 2020. doi ·arXiv
42. Z. Teed and J. Deng, “RAFT-3D: Scene flow using rigid-motion embeddings,” in *Proc. IEEE/CVF CVPR*, 2021. CVF ·arXiv
43. H. Xu, J. Zhang, J. Cai, H. Rezatofighi and D. Tao, “GMFlow: Learning optical flow via global matching,” in *Proc. IEEE/CVF CVPR*, 2022. CVF
44. H. Xu, J. Zhang, J. Cai, H. Rezatofighi, F. Yu, D. Tao and A. Geiger, “Unifying flow, stereo and depth estimation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 13941–13958, 2023. doi ·arXiv
45. Z. Huang, X. Shi, C. Zhang, Q. Wang, K. C. Cheung, H. Qin, J. Dai and H. Li, “FlowFormer: A transformer architecture for optical flow,” in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 668–685, 2022. doi ·arXiv
46. Y. Wang, L. Lipson and J. Deng, “SEA-RAFT: Simple, efficient, accurate RAFT for optical flow,” in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 36–54, 2024. doi ·arXiv
47. S. Vedula, S. Baker, P. Rander, R. Collins and T. Kanade, “Three-dimensional scene flow,” in *Proc. IEEE ICCV*, vol. 2, pp. 722–729, 1999. doi
48. M. Menze and A. Geiger, “Object scene flow for autonomous vehicles,” in *Proc. IEEE CVPR*, pp. 3061–3070, 2015. CVF ·doi
49. A. Geiger, P. Lenz, C. Stiller and R. Urtasun, “Scene flow evaluation 2015 (stereo 2015, flow 2015, scene flow 2015),” The KITTI Vision Benchmark Suite, official benchmark page (dataset size, D1, D2, Fl and SF outlier definitions). link
50. A. Geiger, P. Lenz, C. Stiller and R. Urtasun, “Stereo evaluation 2012,” The KITTI Vision Benchmark Suite, official benchmark page (dataset size, Out-Noc and Out-All at 2–5 px). link
51. A. Geiger, P. Lenz and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *Proc. IEEE CVPR*, pp. 3354–3361, 2012. doi
52. D. Scharstein, H. Hirschmüller, Y. Kitajima, G. Krathwohl, N. Nešić, X. Wang and P. Westling,

"High-resolution stereo datasets with subpixel-accurate ground truth," in *Pattern Recognition (Proc. GCPR)* (Lecture Notes in Computer Science), pp. 31–42, Springer, 2014. doi ·datasets

53. T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys and A. Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *Proc. IEEE CVPR*, pp. 2538–2547, 2017. CVF ·datasets
54. D. J. Butler, J. Wulff, G. B. Stanley and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 611–625, 2012. doi ·project page
55. N. Silberman, D. Hoiem, P. Kohli and R. Fergus, "Indoor segmentation and support inference from RGBD images," in *Proc. ECCV* (Lecture Notes in Computer Science), pp. 746–760, 2012. doi ·dataset page
56. U. Rajapaksha, F. Sohel, H. Laga, D. Diepeveen and M. Bennamoun, "Deep learning-based depth estimation methods from monocular image and videos: A comprehensive survey," *ACM Computing Surveys*, vol. 56, no. 12, 2024. doi ·arXiv

DLCV 12・影像生成：從 GAN 到文字生成影像系統

把影像生成器當成視覺系統來建構與評估：任務與「好」的意義、各模型家族的一張地圖、文字生成影像模型中的文字編碼器、交叉注意力與引導，以及控制、個人化與編輯、自迴歸與統一多模態生成器、評估、資料、風險與失效模式。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/

你將學到

- 依條件與所需輸出定義生成及編輯任務。
- 區分生成器家族，以及像素、潛在與 token 表示。
- 追蹤文字編碼、條件化、取樣與解碼。
- 比較空間控制、個人化與編輯機制。
- 評估品質、多樣性、對齊、成本與資料相關失敗。

先看全貌

影像生成器把一個需求變成一張新的圖片。這個需求可以什麼都沒有（「隨便一張臉」），可以是一個類別標籤（「一隻金翅雀」），可以是一句話（「草地上的一顆紅球」），也可以是另一張影像加上一道指令（「把它變成冬天」）。結果應該看起來真實、每次嘗試都有變化，而且確實畫出了被要求的內容。

> **用最簡單的話說**
>
> 想像一位替警方工作的肖像畫家。你描述一張臉，畫家先畫出一團粗略的模糊輪廓，然後不斷修飾，每畫一筆就回頭看一眼你說的話（「大眼睛、短頭髮」）。生成器做的就是同一件事，而且每秒做很多次。「話」由一個閱讀模型轉成數字，「回頭看」是一個叫交叉注意力的步驟，「從模糊輪廓修成圖片」的修飾則由生成器本身負責。有些畫家的做法不同，是一片一片地貼上馬賽克磚，但他們畫的時候都在聽那段描述。

為什麼重要。影像生成不到十年，就從又小又模糊的臉進步到逼真、能聽懂提示的圖片，現在已經是設計工具、照片編輯器、資料管線與復原系統的一部分（DLCV 09B 就用同樣的模型來修復受損的照片）。作為電腦視覺**系統**，現代生成器結合了語言模型、影像壓縮器、生成網路、取樣器與評估流程。每個部分都可能單獨出錯，也都有各自的文獻。

本篇的位置。理論放在 ML 學習路線，本篇不重複。ML 07 以程式推導 VAE、GAN、正規化流與經典指標（Inception Score、FID、精確率與召回率）。ML 08 推導擴散、DDIM、無分類器引導、潛在擴散、擴散 Transformer、流匹配與少步取樣器。ML 09 談像素與 token 上的自迴歸模型，ML 10 談 CLIP 與參數高效微調。這裡採取的是**系統觀點**：這些元件怎麼組裝成文字生成影像的產品、怎麼操控、怎麼評判，以及可能出什麼問題。兩段程式範例只用 NumPy 與 scikit-image，在 CPU 上幾秒內就能跑完，也不需要網路上的權重。

下一步。 DLCV 12A 把這些系統從影像延伸到影片。

任務：要生成什麼，以及「好」是什麼意思

白話版。生成在文字、標籤或輸入影像等條件下產生輸出；應分別評估真實感、多樣性與條件遵循。

六種生成任務

下面所有任務，都是從條件分布 $p(x \mid c)$ 取樣，其中 x 是影像，c 是條件；無條件生成使用空條件 $c = \emptyset$。

任務	條件 c	輸出	本篇中的範例方法
無條件生成	無	訓練領域內的一張影像	人臉上的 StyleGAN [1]
類別條件生成	一個類別標籤	該類別的一張影像	BigGAN [2]、DiT [3]
文字生成影像	一段文字提示	符合提示的一張影像	Imagen [4]、潛在擴散 [5]
影像轉影像	一張影像（常加上文字）	保留其結構的新影像	SDEdit [6]、ControlNet [7]
編輯	一張影像加上編輯需求	編輯後的影像	Prompt-to-Prompt [8]、InstructPix2Pix [9]
影像修補／外擴	一張影像加上遮罩	遮罩區域被填補的影像	RePaint [10]

因為形式相同，一個訓練好的文字生成影像模型就能涵蓋表中的大部分列：編輯、遮罩與控制影像，都只是額外的條件，或是取樣器不同的起點。

三個問題：保真度、多樣性、對齊

- **保真度（品質）**。每個樣本看起來像該領域的真實影像嗎？有沒有瑕疵？
- **多樣性（涵蓋度）**。許多樣本合起來，涵蓋了真實影像的變化嗎？還是模型一直重複幾個偏愛的結果（模式崩塌，mode collapse）？
- **對齊**。樣本畫出了條件所要求的內容嗎：正確的物體、數量、顏色、位置與文字？

再加上第四個實務問題：**成本**，也就是網路的評估次數，以及每張影像要花幾秒。前三個問題彼此拉扯。多數生成器都有一個旋鈕，用多樣性換取保真度與對齊：BigGAN 的**截斷技巧**（truncation trick）縮小輸入雜訊的範圍 [2]，擴散模型中的**無分類器引導**則把樣本推向條件 [11]。把這種旋鈕調高，影像會更乾淨、更符合提示，但也更相似。每次讀到單一的「品質」數字，都要記得這個取捨。分別量測保真度與多樣性的指標，推導在 ML 07；本篇在後面的評估一節補上對齊這一面。

各模型家族的一張地圖

白話版。 GAN、擴散、流與自迴歸模型學習不同生成程序；另須選擇以像素、連續潛在表示或 token 表達影像。

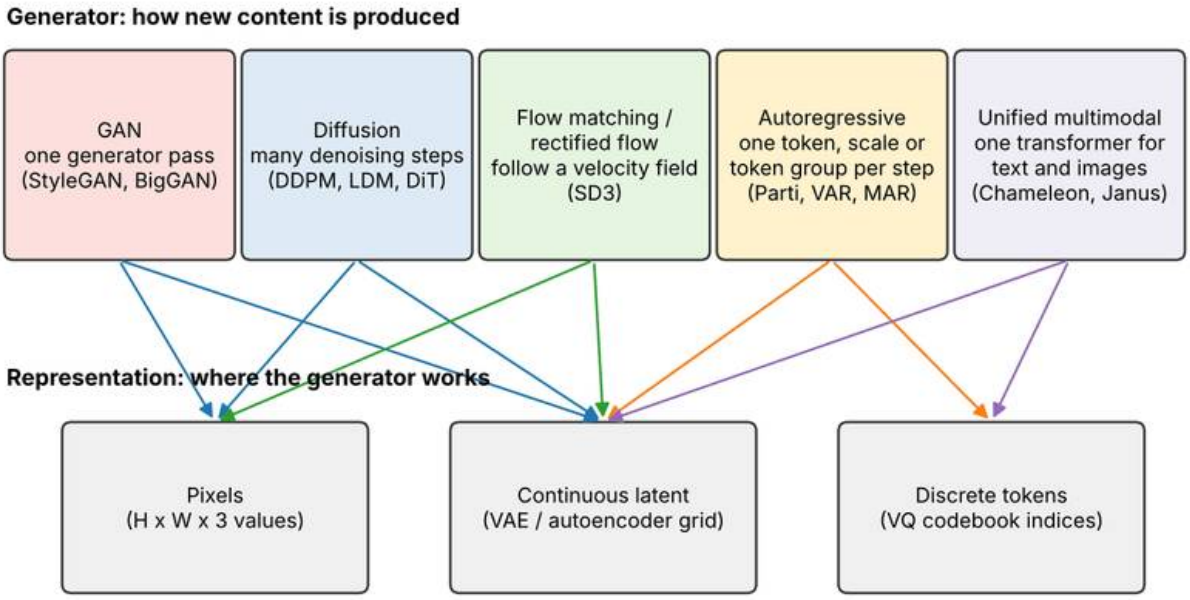


圖 *1* — 一張地圖上的各個家族。上排是新內容怎麼產生；下排是生成器在哪裡運作。箭頭表示常見的搭配；表示方式與生成器是各自獨立的設計選擇。

GAN：通過生成器一次。生成器把隨機雜訊對應成影像，同時訓練的判別器則試著分辨生成影像與真實影像；生成器靠騙過它來進步 [12]。取樣只要一次前向傳遞，所以 GAN 很快。StyleGAN [1] 把雜訊對應成一個中間的「風格」向量，並把它注入生成器的每個解析度，這把粗略的屬性（姿勢、身分）和細緻的屬性（雀斑、頭髮）分開，也讓各尺度可以分別控制。BigGAN [2] 在 ImageNet 上擴大類別條件 GAN 的規模，並提出上面提到的截斷技巧。GAN 的訓練不穩定，也可能漏掉某些模式，這讓它們很難擴展到開放式的提示。理論與程式：ML 07，GAN。

VAE 與 tokenizer：先壓縮。變分自編碼器（VAE）學習一個編碼器把影像編成緊湊的潛在編碼，一個解碼器把它還原成像素，以及編碼上的一個簡單先驗 [13]。在今天的系統裡，自編碼器的主要工作是**為另一個生成器做前置壓縮**：潛在擴散在比影像小好幾倍的連續潛在網格上執行去雜訊網路 [5]；以 token 為基礎的模型則使用**向量量化**（vector-quantized）自編碼器（VQ-VAE [14]、VQGAN [15]），把影像變成由學到的碼本（codebook）中整數索引組成的網格，就像字典裡的字。解碼器設下了上限：解碼器重建不出來的細節，沒有任何生成器能產生。細節見 ML 07，VAE 與 ML 09，視覺 token。

擴散：許多小步的去雜訊。訓練一個網路，從帶雜訊的影像中去掉一點 Gaussian 雜訊；要生成時，從純雜訊出發，反覆套用許多次 [16]。訓練是穩定的迴歸問題，加上更好的架構與分類器引導後，擴散模型在 ImageNet 上超越了 GAN [17]。把同樣的過程放在自編碼器的潛在網格上執行，也就是**潛在擴散**，讓高解析度的文字生成影像變得負擔得起 [5]。理論與程式：ML 08，以及其中的潛在擴散一節。

流匹配與整流流：沿著速度場前進。取一個雜訊樣本和一個資料樣本，用直線連起來，訓練網路預測沿著這類直線的行進方向；要生成時，從雜訊出發，沿著預測的方向前進 [18], [19]。它與擴散關係密切（只是雜訊與資料之間走的路徑不同），現在已用在大規模系統：Stable Diffusion 3 以整流流（rectified flow）目標訓練一個大型潛在 Transformer [20]。理論：ML 08，流匹配。

自迴歸與統一模型：一片一片地寫出圖片。把影像變成 token 序列，在已生成的所有內容的條件下，預測下一個 token（或下一組 token，或整張 token 圖下一個更清晰的版本）。這就是把語言模型的做法用在影像上，最新的形式是用單一個 Transformer 同時讀寫文字與影像。後面有專屬的一節。

家族	如何取樣	循序的網路呼叫次數	主要優點	主要注意事項
GAN	通過生成器一次	一次	速度快、平滑可編輯的潛在空間	訓練穩定性、模式涵蓋
擴散	從雜訊反覆去雜訊	數十到數百次；蒸餾後更少	品質、涵蓋度、訓練穩定	取樣成本
流匹配	對學到的速度做積分	數十次；蒸餾後更少	目標簡單、路徑更直	沒蒸餾還是需要好幾步
自迴歸	依某種順序預測 token	每個 token、每組或每個尺度一次	重用語言模型的機制、token 有精確概似	tokenizer 的上限、誤差累積

地圖上有兩點容易忘記。生成器與表示方式是各自獨立的選擇：擴散模型可以在像素上運作（Imagen [4]），也可以在潛在上運作（潛在擴散 [5]），自迴歸模型可以使用離散或連續的 token [21]。而且實際的系統會混合各家族：VQGAN 的 tokenizer 是用對抗損失訓練的 [15]，對抗式蒸餾則把擴散模型變成只要一到四步的生成器 [22]。

文字生成影像系統

白話版。潛在文字生成影像流程先編碼提示、產生條件化潛在表示，再解碼像素；條件介面連接語言與影像特徵。

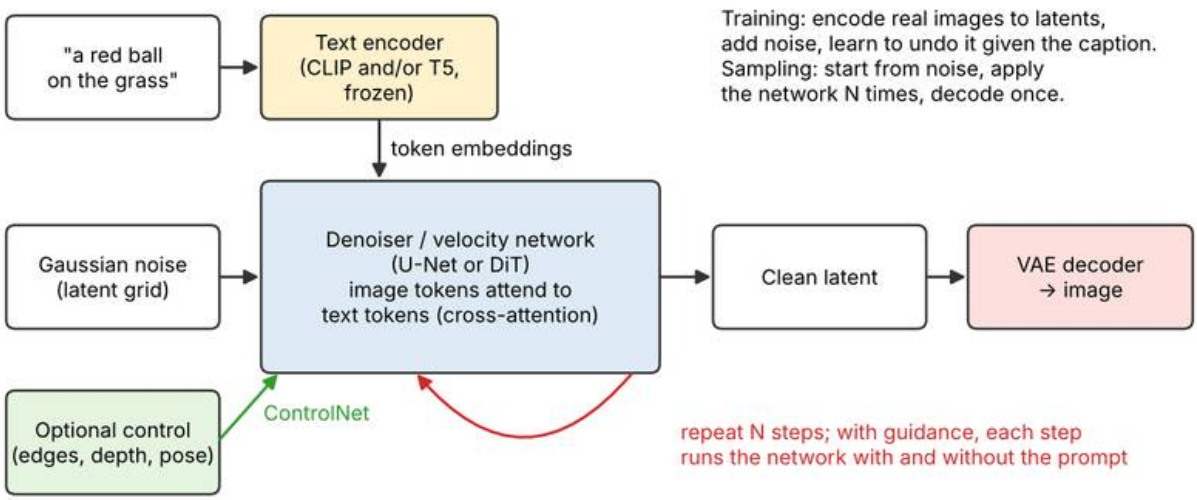


圖 2 — 潛在文字生成影像流程。文字編碼器與自編碼器通常分開訓練並凍結；只有去雜訊網路（或速度網路）在影像與標題的配對上訓練。取樣時呼叫去雜訊網路 N 次、解碼器一次。

訓練時，影像與標題配對中的每張影像會被編碼成潛在表徵、加上隨機強度的雜訊，去雜訊網路學習在**給定標題嵌入**的條件下還原雜訊。取樣時，去雜訊網路從純雜訊出發，執行 N 步，再由解碼器產生像素。這兩句話背後的擴散與取樣公式，在 ML 08；下面我們看與文字有關的部分。

文字編碼器：CLIP 或語言模型

白話版。文字編碼器決定生成器能「理解」什麼。一個訓練來配對影像與標題的模型，懂得視覺概念；只用文字訓練的大型語言模型，則更懂文法、詞序，以及和拼字有關的結構。許多系統兩種都用。

主要有兩類編碼器：

- **CLIP 文字編碼器**以對比方式訓練，讓影像與它的標題在共享的嵌入空間中彼此靠近 [23]（見 ML 10）。它們的嵌入有視覺上的依據。
- **語言模型編碼器**，例如 T5 [24]，只用文字訓練。Imagen 的關鍵發現是：一個大型**凍結**的 T5 編碼器出乎意料地好用，而把文字編碼器加大，對保真度與影像–文字對齊的改善，遠大於把影像擴散模型加大 [4]。

近期的系統會把兩者結合。SDXL 在先前 Stable Diffusion 模型的設計上加了第二個文字編碼器 [25]。Stable Diffusion 3 使用三個凍結的編碼器：兩個 CLIP 模型和 T5-XXL；作者報告指出，推論時拿掉 T5，對大多數提示損失不大，但對含有長篇文字或許多細節的提示明顯有害 [20]。有一個相關的發現與拼字有關：文字編碼器若看到的是整個詞片段而不是字元（「對字元視而不見」，character-blind），就很難一個字母一個字母地畫出一個字，而能辨識字元的編碼器，繪製視覺文字的準確度高得多 [26]。DALL·E 2 走的是另一條路：先由一個「先驗」（prior）網路從標題生成 CLIP **影像**嵌入，再由擴散解碼器根據這個嵌入生成影像 [27]。

交叉注意力：文字如何抵達像素

白話版。在每一層，影像的每個位置都先問一個問題（「這裡該放什麼？」），再掃過提示的所有字，找出能回答的那些字。它收集到的答案，是這些字的加權混合，匹配的字權重大。

潛在擴散為此在 U-Net 去雜訊網路中加入了交叉注意力層 [5]。令 $\Phi \in \mathbb{R}^{N\times d_\phi}$ 為某一層的影像特徵（N 個空間位置，每個是寬度 d_ϕ 的向量），$E \in \mathbb{R}^{L\times d_e}$ 為文字編碼器的輸出（L 個寬度 d_e 的 token）。學到的矩陣 $W_Q \in \mathbb{R}^{d_\phi\times d}$、$W_K, W_V \in \mathbb{R}^{d_e\times d}$ 把它們投影到共同的寬度 d：

$$Q = \Phi W_Q, \qquad K = E W_K, \qquad V = E W_V, \qquad A = \mathrm{softmax}\Big(\frac{QK^\top}{\sqrt{d}}\Big), \qquad \mathrm{out} = AV.$$

這裡 $Q \in \mathbb{R}^{N\times d}$ 為每個影像位置存放一個**查詢**（query），$K, V \in \mathbb{R}^{L\times d}$ 為每個文字 token 存放一個**鍵**（key）與一個**值**（value），softmax 沿著每一列計算，而 $A \in \mathbb{R}^{N\times L}$ 是**注意力圖**：第 i 列是 L 個 token 上的機率分布，說明位置 i 對每個字聽多少。輸出 $\mathrm{out} \in \mathbb{R}^{N\times d}$ 給每個影像位置一份字的值的混合，並加回影像特徵。查詢來自影像，鍵與值來自文字，這正是它稱為交叉注意力而不是自注意力的原因（比較 NLP 07）。把 A 的第 j 行當成影像來看，就得到**第 j 個字被畫在哪裡**的熱度圖，Prompt-to-Prompt 就把這個事實變成一種編輯方法 [8]。

Transformer 骨幹提供了另一種做法。Stable Diffusion 3 的 MM-DiT 把影像 token 與文字 token 串成一個序列，讓它們雙向互相注意，兩種模態各有各自的權重 [20]。於是文字也會「讀」影像，而不只是影像讀文字。

下面的玩具範例建立一個 16×16 的影像特徵網格，其中一塊圓盤區域像是「red」和「ball」這兩個字，背景像是「grass」，然後執行一次交叉注意力。為了易讀，投影矩陣用單位矩陣；在訓練好的模型中它們是學出來的。

```
import numpy as np

rng = np.random.default_rng(0)
d = 32                                          # feature width
words = ["a", "red", "ball", "on", "grass"]  # the prompt, one text token per word
E = rng.normal(size=(len(words), d))         # text-encoder outputs, one row per word

# A 16 x 16 grid of image tokens: a disk ("the ball") on a background ("the grass").
yy, xx = np.mgrid[:16, :16]
ball = ((yy - 6) ** 2 + (xx - 9) ** 2 <= 9).ravel()   # 29 of the 256 positions
X = 1.0 * rng.normal(size=(256, d))                   # noisy image features
X[ball] += 0.5 * (E[1] + E[2])                        # ball region looks like "red ball"
X[~ball] += 0.5 * E[4]                                # background looks like "grass"

# Cross-attention: queries from the image, keys and values from the text.
# Real models learn W_q, W_k, W_v; identity matrices keep the toy readable.
Q, K, V = X, E, E
scores = Q @ K.T / np.sqrt(d)                         # 256 image tokens x 5 words
A = np.exp(scores - scores.max(axis=1, keepdims=True))
A /= A.sum(axis=1, keepdims=True)                     # softmax over the words
out = A @ V                                           # each image token gets a mix of word
 ↪  values

print("each row sums to 1:", np.allclose(A.sum(axis=1), 1), "| output shape:", out.shape)
for j, w in enumerate(words):
    print(f"{w:6s} mean weight inside the ball={A[ball, j].mean():.2f}  "
          f"outside={A[~ball, j].mean():.2f}")
```

輸出如下：

```
each row sums to 1: True | output shape: (256, 32)
a      mean weight inside the ball=0.02  outside=0.06
red    mean weight inside the ball=0.43  outside=0.07
ball   mean weight inside the ball=0.52  outside=0.04
on     mean weight inside the ball=0.02  outside=0.08
grass  mean weight inside the ball=0.01  outside=0.74
```

（這些數字來自我們的玩具實驗，不是任何訓練好的模型。）在圓盤內，幾乎所有注意力都給了「red」和「ball」；在圓盤外，四分之三給了「grass」。虛詞「a」和「on」在任何地方權重都很小。圖 3 畫出這五張圖。請注意這個機制**不**保證的事：「red」和「ball」位在同一區，是因為特徵這樣設定；注意力公式裡沒有任何東西，會強迫顏色字附著到正確的物體上。當畫面上有兩個物體時，顏色可能注意到錯的那一個，這是後面要談的屬性綁定失敗的根源之一 [28]。

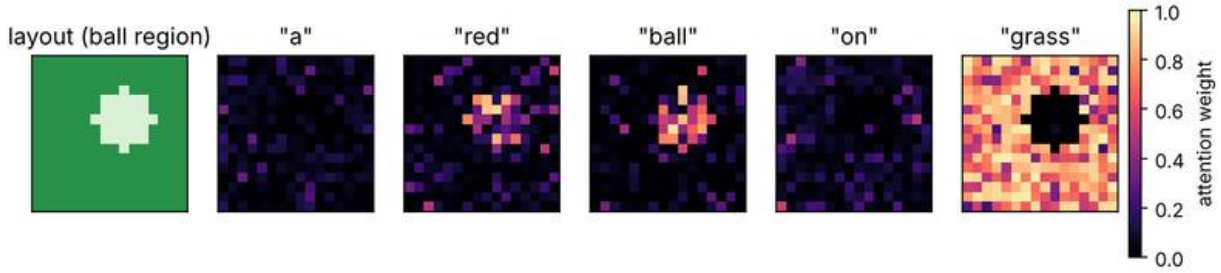


圖 3 — 玩具範例中的交叉注意力圖。每張圖都是注意力矩陣 A 的一行，重新排成影像網格：每個字被「畫」在哪裡。*Prompt-to-Prompt* 就是重複使用這類圖來編輯影像 [8]。

實務上的引導

無分類器引導（classifier-free guidance，CFG）的推導與測試在 ML 08；這裡只做提醒。訓練時，對隨機一部分的樣本把標題丟掉（換成空提示），讓同一個網路同時學會條件預測與無條件預測 [11]。取樣時把兩者結合：$\hat{\epsilon} = \epsilon_\emptyset + w(\epsilon_c - \epsilon_\emptyset)$，其中 ϵ_c 與 $\epsilon_\emptyset$ 是有提示與沒有提示的預測，w 是引導尺度；$w > 1$ 會越過條件預測，朝提示的方向再移動。

對系統建構者而言，有四件實務上的事很重要：

- **它是旋鈕，不是量測值。**較高的 w 通常讓影像更忠於提示、對比更高，但變化更少；w 非常高時，影像會過飽和而扭曲。要報告它的數值與慣例（有些程式碼寫成 $1 + w$）。
- **除非把兩個預測合成一個批次，否則每一步的工作量會加倍，**所以 30 步的引導取樣器大約要做 60 次網路評估。
- **它也適用於 token 模型。**自迴歸與遮罩 token 生成器，會對下一個 token 的 logits 套用同樣的公式（ML 09）。
- **蒸餾模型可能把它內建進去。**有些蒸餾後的檢查點，被訓練成重現有引導的輸出，所以引導尺度變成輸入或乾脆消失；比較設定之前，要先讀發布文件（ML 08）。

骨幹與幾個代表性系統

去雜訊網路最早是 **U-Net** [29]，一種有跳接的卷積編碼器–解碼器，並在數個解析度上插入交叉注意力層 [5]。**擴散 Transformer（DiT）**則用作用在潛在小塊上的 Vision Transformer 取代它，並且隨著 Transformer 的運算量增加，表現穩定進步 [3]（細節見 ML 08）。近期的大型系統多半使用 Transformer。下表列出幾個設計已有論文記載的系統；它是設計選擇的地圖，不是排名。

系統	在哪裡生成	生成器與骨幹	文字條件化
潛在擴散／Stable Diffusion [5]	自編碼器潛在空間	擴散，U-Net	對文字編碼器做交叉注意力
DALL·E 2 [27]	像素	以 CLIP 影像嵌入為條件的擴散解碼器	由先驗把標題對應成 CLIP 影像嵌入
Imagen [4]	像素，64 → 256 → 1024 級聯	擴散	凍結的 T5-XXL

系統	在哪裡生成	生成器與骨幹	文字條件化
Parti [30]	離散 token（ViT-VQGAN）	自迴歸編碼器–解碼器 Transformer	文字是編碼器的輸入
Muse [31]	離散 token	遮罩平行解碼，Transformer	來自預訓練語言模型的嵌入
SDXL [25]	自編碼器潛在空間	擴散，更大的 U-Net，可選的精修器	兩個文字編碼器
Stable Diffusion 3 [20]	自編碼器潛在空間	整流流，MM-DiT Transformer	兩個 CLIP 編碼器與 T5-XXL

Imagen 級聯的解析度，來自其參考文獻所列的專案頁面。

僅線上版：*Esser* 等人：*Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*（*ICML 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/papers/esser-等人 scaling-rectified-flow-transformers-for-high-resolution-image-synthesisicml-2024/*

控制、個人化與編輯

白話版。提示是很弱的方向盤：文字無法釘死確切的姿勢、某個人的狗，或「只改帽子」。有三類方法提供更強的操控：額外的**控制影像**（邊緣、深度、姿勢）、教模型新概念的小型**個人化**更新，以及改動給定影像、同時保留其餘部分的**編輯**流程。

空間控制：ControlNet

ControlNet 把空間條件（邊緣圖、深度圖、分割圖、人體姿勢）加到大型預訓練的文字生成影像模型上，又不損壞它 [7]。它把原始網路**鎖住**，為它的編碼層做一份接收條件的**可訓練副本**，再透過**零卷積**（zero convolution）把副本接回原始網路：零卷積是權重與偏差都從零開始的 1×1 卷積。在第一個訓練步，新加的分支輸出剛好是零，所以模型的行為與原始模型完全相同，學習從一個能運作的生成器出發。論文報告，不論資料集小（少於 5 萬）或大（超過 100 萬），訓練都很穩健 [7]。

僅線上版：*ControlNet* 的零卷積：為什麼訓練從原始模型開始 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/notes/controlnet-的零卷積為什麼訓練從原始模型開始/*

個人化：教模型一個新概念

三種方法在儲存量、速度與保真度之間各有取捨：

- **Textual inversion** 讓整個模型保持凍結，只從某個概念的 3–5 張影像學習一個新的詞嵌入（一個新的「偽詞」），做法是在這些影像上最小化一般的去雜訊損失 [32]。儲存下來的結果只是一個向量，但它的表達能力受限於凍結的模型本來就畫得出來的東西。
- **DreamBooth** 用某個主體的幾張影像微調整個模型，影像的標題包含一個罕見的識別字加上類別名稱，例如 “a [V] dog”，並加上一項**類別專屬的先驗保留損失**（prior-preservation loss），讓模型仍然畫得出一般的狗；沒有它，模型會漂移而失去多樣性 [33]。

- **LoRA**（低秩適配，low-rank adaptation）凍結每個權重矩陣 $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$，只學習加性更新 $\Delta W = BA$，其中 $B \in \mathbb{R}^{d_{\text{out}} \times r}$、$A \in \mathbb{R}^{r \times d_{\text{in}}}$，且秩 r 很小 [34]。它原本是為語言模型提出的；同樣的更新可以接在文字生成影像去雜訊網路的注意力層上，讓個人化的風格或主體變成一個可以開關的小檔案（ML 10，PEFT）。

僅線上版：個人化目標函數並排比較 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/notes/個人化目標函數並排比較/*

編輯與影像修補

- **SDEdit** 對輸入影像（例如一張粗略的彩色草圖）加雜訊，加到某個中間的雜訊強度，再執行一般的反向過程 [6]。起始雜訊強度低，會保留輸入的版面與顏色（忠實）；強度高，則給模型更多自由（真實）。一個數字就決定了這個平衡。
- **Prompt-to-Prompt** 透過更改提示（“a cat on a bike” → “a dog on a bike”）來編輯生成的影像，同時在取樣時重複使用原本的交叉注意力圖，所以版面維持不變，只有被改動的字的內容會變 [8]。要用這種方式編輯真實照片，必須先找到一個能重現它的雜訊，例如用確定性的 DDIM 反轉（ML 08）。
- **InstructPix2Pix** 依照書面指令（“make it look like winter”）行動。它的訓練資料是自動產生的：結合語言模型（GPT-3）與文字生成影像模型（Stable Diffusion），做出大量編輯前後的範例；用這些配對訓練的條件擴散模型，只要一次取樣就能完成編輯，不需要對每張影像微調或反轉 [9]。
- **RePaint** 用無條件擴散模型做影像修補：在每個反向步驟，它把已知區域換成輸入影像加上正確強度雜訊的副本，讓模型只生成被遮住的區域 [10]。

Huang 等人的一篇綜述，依學習策略、輸入條件與任務，整理了許多以擴散為基礎的編輯方法 [35]。

快速取樣簡述

文字生成影像模型是互動使用的，所以網路呼叫的次數很重要。各條路線在 ML 08，少步生成 都談過：DDIM 這類確定性取樣器，用同一個網路就需要較少的步數 [36]；漸進式蒸餾（progressive distillation）反覆把步數減半 [37]；一致性模型（consistency models）學習從軌跡上任何一點直接跳到終點 [38]；對抗式擴散蒸餾（adversarial diffusion distillation，ADD）把擴散教師與對抗損失結合，讓大型模型只用 1–4 步就能取樣 [22]。比較系統時，要算每一次網路評估（包含兩次引導傳遞），再加上解碼器。

自迴歸與統一多模態生成器

白話版。自迴歸生成器寫影像的方式，就像語言模型寫句子：一次一個 token，或一次一組 token，每次都以前面寫的為條件。最新的一步，是用**一個** Transformer 同時處理文字與影像，讓同一個模型既能描述圖片、回答關於圖片的問題，也能畫新的圖。

一段話看自迴歸這條路線

理論在 ML 09；這裡只做摘要。tokenizer（VQ-VAE 或 VQGAN，ML 09）把影像變成整數網格，Transformer 用鏈鎖法則為它們建模，以 teacher forcing 的下一個 token 交叉熵訓練（ML 09）。Parti [30] 與 LlamaGen [39] 這類逐行掃描（raster-order）模型，每步預測一個 token；MaskGIT [40] 與 Muse [31] 這類遮罩模型一次平行預測許多 token，並在幾輪中逐步修正（ML 09）；VAR 每步預測整張更清晰的 token 圖，由粗到細 [41]（ML 09）；MAR 則放棄離散碼本，用一個小型擴散頭為每個連續 token 建模 [21]（ML 09）。取樣成本、KV 快取以及與擴散的比較，在 ML 09。

實務上的自迴歸文字生成影像

從系統面來看，有四件事值得知道。

- **文字是以上下文的形式進來，而不是透過對另一個去雜訊網路的交叉注意力。** Parti 把文字生成影像當成翻譯：Transformer 編碼器讀提示，解碼器寫出影像 token；論文把這個編碼器–解碼器模型擴大到 200 億個參數，並提出 PartiPrompts 基準，包含超過 1,600 個英文提示 [30]。Muse 則把預訓練大型語言模型的文字嵌入餵給遮罩 token Transformer，不需微調就支援影像修補、外擴與不用遮罩的編輯 [31]。
- **語言模型的基礎設施可以直接轉用。** LlamaGen 使用單純的 Llama 式解碼器；除了 111M 到 3.1B 參數的類別條件模型，它還訓練了一個 775M 的文字條件模型，並報告 LLM 服務框架讓影像生成加速 326%–414% [39]。
- **tokenizer 具有決定性。** MAGVIT-v2 主張 tokenizer 是關鍵元件：配上它新的「免查表」（lookup-free）量化器，詞彙量可以變得很大，在相同訓練資料、相近模型大小與相近訓練預算下，語言模型在 ImageNet 上勝過擴散模型 [42]。文字和細線這類小細節，是粗糙 tokenizer 的典型犧牲品（ML 09）。
- **引導與取樣的旋鈕可以沿用。** 在 logits 上做 CFG、temperature 與 top-k 取樣，決定保真度與多樣性的平衡，而且每個有引導的步驟要做兩次前向傳遞。

統一多模態模型

白話版。與其用一個文字模型把提示交給另一個獨立的影像模型，統一模型把文字與影像片段放進**同一個序列**，用一個 Transformer 訓練。設計上的問題是：影像片段是離散 token 還是連續向量、影像部分用下一個 token 預測還是擴散生成，以及理解與生成是否共用同一個影像編碼器？

下面這些模型都有論文記載，並說明了主要的設計選擇：

- **Chameleon** 從一開始就把離散的影像 token 與文字 token 放在同一個序列（「早期融合」，early fusion），用一個模型處理圖說生成、問答，以及文字、影像與混合文件的生成 [43]。
- **Emu3** 從頭訓練一個 Transformer，**只**用下一個 token 預測，處理影像、文字與影片的離散 token [44]。
- **Transfusion** 讓影像保持連續，用一個 Transformer 同時訓練文字上的下一個 token 損失與影像小塊上的擴散損失；作者報告這樣的擴展性比把影像量化成 token 更好 [45]。
- **Show-o** 把文字的自迴歸建模，與影像 token 上的離散（遮罩）擴散結合起來 [46]。

- **Janus** 共用一個 Transformer，但為理解與生成使用**不同的**視覺編碼器，因為這兩個任務需要不同程度的影像細節 [47]。

模型	影像表示	影像目標	視覺編碼器
Chameleon [43]	離散 token	下一個 token	一個，共用
Emu3 [44]	離散 token	下一個 token	一個，共用
Transfusion [45]	連續小塊	擴散	一個，共用
Show-o [46]	離散 token	遮罩（離散）擴散	一個，共用
Janus [47]	生成用離散 token	下一個 token	理解與生成各自獨立

僅線上版：統一多模態模型逐一介紹 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/notes/統一多模態模型逐一介紹/*

目前仍未解決的是：一個模型能否在相同大小下，同時在理解與生成上都達到最先進；共用的視覺 tokenizer 讓每個任務付出多少代價（這是 Janus 的動機）；以及評估該如何比較統一模型與專門的擴散系統。這些論文中關於擴展的主張，都是各篇論文在自己資料上的量測，還不是定論。

評估文字生成影像模型

白話版。你需要四種證據：影像看起來真實又多樣嗎（分布指標）、整體上符合提示嗎（對齊分數）、細節對嗎（組合式檢查），以及人喜歡嗎（人類偏好）。每一種都能抓到其他幾種漏掉的失敗。

分布指標及其陷阱

FID 比較生成影像與真實影像的特徵統計；精確率與召回率把保真度與涵蓋度分開。兩者的定義與實作都在 ML 07，ML 08 則談較新的特徵空間。對文字生成影像模型，常見的協定是**零樣本 COCO FID**：用沒有在 COCO 上訓練的模型，根據 COCO 標題 [48] 生成影像，再與真實的 COCO 影像比較，Imagen 就是這麼做的 [4]。實務上有五個陷阱：

- **有限樣本偏差**。 FID 在有限的樣本集上有偏差，且偏差取決於被評估的模型，所以即使樣本數相同，比較也不完全公平；Chong 與 Forsyth 提議外插到無限樣本數 [49]。
- **前處理**。特徵擷取之前所用的縮放濾波器與 JPEG 壓縮，會大幅改變 FID；Parmar 等人展示了這點，並提供參考實作 [50]。
- **特徵看到什麼**。 Inception 特徵與 ImageNet 類別非常接近，因此讓生成影像的類別直方圖吻合，就可以降低 FID，卻不代表影像變好 [51]。
- **與人的一致性**。在一項大型人類研究中，沒有任何常見指標與人類判斷高度相關，且 FID 低估了擴散模型；作者建議使用 DINOv2 這類更豐富的特徵擷取器 [52]。CMMD 則以 CLIP 嵌入和核距離，取代 Inception 特徵與 Gaussian 假設 [53]。
- **沒有對齊**。 FID 完全不看提示：一個無視所有提示、卻畫出漂亮的類 COCO 影像的模型，也可能得到好分數。

提示對齊：CLIPScore

CLIPScore 衡量影像與文字在 CLIP 嵌入空間中有多一致 [54]。令 $\mathbf{c}$ 為文字的 CLIP 嵌入，$\mathbf{v}$ 為影像的 CLIP 嵌入，

$$\text{CLIP-S}(\mathbf{c}, \mathbf{v}) = w \cdot \max\left(\cos(\mathbf{c}, \mathbf{v}), 0\right),$$

其中 cos 是餘弦相似度，w 是只用來重新縮放範圍的常數（原論文中 $w = 2.5$，該論文是為影像圖說提出這個分數的）。用於文字生成影像的評估時，要對提示–影像配對取平均。它便宜，也不需要參考影像，但它繼承了 CLIP 的盲點：一個概括整張影像的嵌入，不擅長處理數量、空間關係，以及哪個屬性屬於哪個物體。GenEval 的研究發現，在這類組合式任務上，它的物件層級檢查與人類判斷的一致程度遠高於 CLIPScore [55]。

組合式基準

白話版。與其用一個整體相似度，不如對影像提出具體的是非題：有狗嗎、剛好三隻嗎、狗在貓的左邊嗎、車是紅的嗎？由偵測器或視覺–語言模型來回答。

GenEval 用模板為六種任務（單一物體、兩個物體、計數、顏色、位置與屬性綁定）產生提示，用實例分割模型偵測物體，用 CLIP 分類物體顏色，再把每張影像評為正確或不正確 [55]。**T2I-CompBench** 提供 6,000 個組合式提示，分三類（屬性綁定、物體關係與複雜組合），並為每類設計專屬的指標 [56]。Imagen 的 **DrawBench** [4] 與 Parti 的 **PartiPrompts** [30] 是供人類並排比較的提示集。物件偵測本身是 DLCV 03 的主題；開放詞彙偵測器（DLCV 03）可以檢查固定類別清單以外的物體。

僅線上版：*Ghosh* 等人：*GenEval: An Object-Focused Framework for Evaluating Text-to-Image Alignment*（*NeurIPS 2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/papers/ghosh-等人 geneval-an-object-focused-framework-for-evaluating-text-to-image-alignmentneurips-2023/*

下面的玩具範例說明為什麼物件層級的檢查很重要。它為提示 “two red circles and a blue square” 畫出由彩色圓盤與方塊組成的合成「生成影像」，用連通元件偵測形狀，並比較兩種分數：一個是**出現分數**（presence score），只問提示裡的哪些字出現在影像的任何地方（刻意設計成整體相似度分數的簡單替身，不是 CLIP 本身）；另一個是**精確檢查**（exact check），要求每個（顏色、形狀）配對都有正確的數量，像 GenEval 那樣。

```
import numpy as np
from skimage.draw import disk, rectangle
from skimage.measure import label, regionprops

COLORS = {"red": (0.85, 0.1, 0.1), "blue": (0.1, 0.2, 0.85)}

def render(objects, size=96):
    """Draw (color, shape) objects at fixed slots on a gray canvas: a stand-in for a
     ↪  generated image."""
    img = np.full((size, size, 3), 0.8)
    for k, (col, shape) in enumerate(objects):
        cy, cx = 24 + 48 * (k // 2), 24 + 48 * (k % 2)
        if shape == "circle":
```

```
            rr, cc = disk((cy, cx), 14)
        else:
            rr, cc = rectangle((cy - 12, cx - 12), extent=(24, 24))
        img[rr, cc] = COLORS[col]
    return img

def detect(img):
    """Toy 'detector': connected colored regions -> (color, shape) pairs."""
    found = []
    for region in regionprops(label(np.abs(img - 0.8).sum(-1) > 0.3)):
        r, g, b = img[tuple(region.coords.T)].mean(0)
        col = "red" if r > b else "blue"
        shape = "square" if region.extent > 0.9 else "circle"   # filled bbox fraction
        found.append((col, shape))
    return found

def presence_score(prompt, img):
    """Bag-of-words check: which prompt words appear anywhere (ignores counts and
     ↪  binding)."""
    words = {w for obj in prompt for w in obj}
    seen = {w for obj in detect(img) for w in obj}
    return len(words & seen) / len(words)

def exact_check(prompt, img):
    """GenEval-style check: every (color, shape) pair with the right count."""
    return sorted(detect(img)) == sorted(prompt)

prompt = [("red", "circle"), ("red", "circle"), ("blue", "square")]   # "two red circles and
 ↪  a blue square"
candidates = {
    "correct":          prompt,
    "wrong count":      prompt + [("red", "circle")],
    "colors swapped":   [("blue", "circle"), ("blue", "circle"), ("red", "square")],
    "object missing":   prompt[:2],
}
for name, objs in candidates.items():
    img = render(objs)
    print(f"{name:15s} presence score={presence_score(prompt, img):.2f}  "
          f"exact check={'pass' if exact_check(prompt, img) else 'fail'}")
```

輸出如下：

```
correct         presence score=1.00  exact check=pass
wrong count     presence score=1.00  exact check=fail
colors swapped  presence score=1.00  exact check=fail
object missing  presence score=0.50  exact check=fail
```

出現分數給數量錯誤和顏色互換的影像滿分，因為提示裡的每個字都出現在某個地方；只有精確檢查抓得到它們。真實的基準會用訓練好的偵測器與顏色分類器，取代我們的連通元件「偵測器」，並承襲它們的錯誤：檢查的好壞只取決於偵測器，這也是 GenEval 要報告它與人類標註者一致程度的原因 [55]。圖 4 畫出這四張影像。

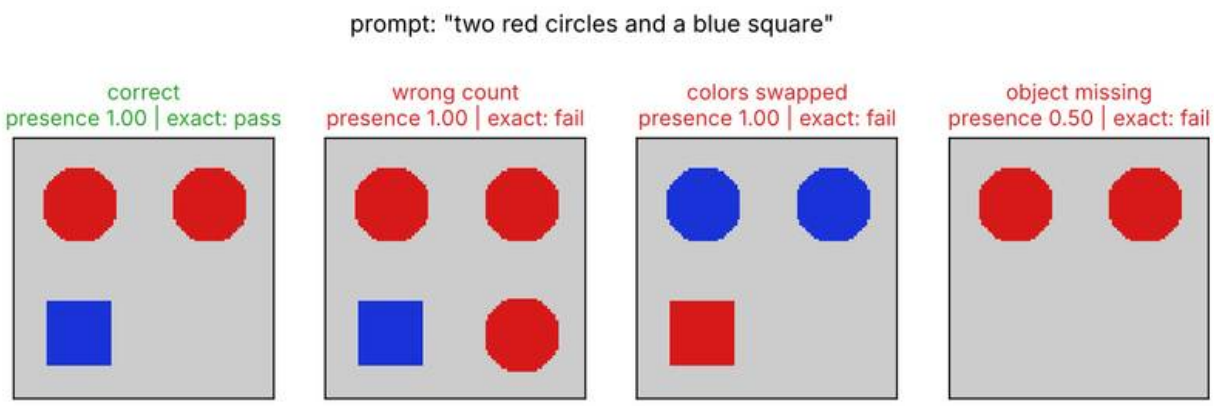


圖 *4* — 字詞出現分數看不到計數與綁定錯誤；物件層級的檢查則看得到。這裡用合成的形狀代替生成影像。

人類偏好與整體評估

人仍然是最終的參考標準。因為人類研究很慢，有幾個團隊訓練了**偏好模型**，預測人們針對某個提示會偏好兩張影像中的哪一張：ImageReward 以 13.7 萬筆專家比較訓練 [57]，PickScore 則以 Pick-a-Pic 訓練，這是透過網頁應用程式收集真實使用者選擇的開放資料集 [58]。**HEIM** 沿著 12 個面向（包括對齊、品質、美感、原創性、推理、知識、偏見、毒性、公平性、穩健性、多語言與效率）評估模型，涵蓋 62 個情境與 26 個模型，並發現沒有單一模型在所有面向都最好 [59]。學習出來的偏好模型本身也是帶有偏見的模型；如果把生成器調整成最大化它，分數可能比真實品質上升得更快。

報告檢查清單。對文字生成影像模型的任何主張，都要說明：提示（以及如何取樣）、每個提示的影像數與隨機種子、取樣器、步數與引導尺度、解析度、指標的實作與特徵擷取器，並展示未經挑選的樣本。把分布、對齊、組合式與偏好的結果並列報告，並說明哪些數字是自己跑的，哪些是從論文抄來的。

資料

白話版。生成器只能學到訓練影像和標題所呈現的東西，所以資料集決定了它的知識、風格與許多偏見。大型文字生成影像資料集是從網路上爬取的，這帶來品質、著作權與同意權的問題。

- **類別條件基準**使用 ImageNet [60]；文字生成影像論文常在 COCO 標題上做零樣本評估 [48]；人臉生成器則使用 FFHQ 這類資料集，也就是隨 StyleGAN 一起提出的高品質人臉資料集 [1]。
- **LAION-5B** 是公開釋出的資料集，包含 58.5 億個經 CLIP 過濾的影像–文字配對，其中 23.2 億個的文字是英文 [61]。它發布的是 URL 與替代文字，而不是影像本身；它的資料集說明（datasheet）寫明，LAION 以 CC-BY-4.0 釋出中介資料，但不擁有影像或文字的著作權，影像中出現的人可能並未同意上傳，且資料集目前的形式應該用於學術研究；它也說明了申請移除連結的方法 [61]。

- **內容稽核**。 Birhane 等人檢視了較早的 LAION-400M 資料集，發現其中有露骨且有問題的影像–文字配對，包括色情內容、惡意刻板印象與辱罵字眼 [62]。
- **標題也是資料**。 Stable Diffusion 3 以各半的原始標題與視覺–語言模型寫的合成標題混合訓練，並過濾資料中的色情內容、低美感分數與近似重複（以減少訓練影像被原樣吐出）[20]。

風險與負責任的使用

白話版。能畫出任何東西的模型，也能抄襲、強化刻板印象與欺騙。實務上的對策是量測這些行為，在資料與模型中減少它們，並為生成的影像加上標示，讓觀看的人可以查驗它們從哪裡來。

記憶與抄襲

Carlini 等人顯示，擴散模型會記住個別的訓練影像，並且可能把它們吐出來：他們用「生成後過濾」的流程，取出超過一千個訓練樣本，從個人的照片到有商標的標誌都有，並且在實驗中發現擴散模型的隱私性比 GAN 差 [63]。Somepalli 等人建立影像檢索工具，把生成影像與訓練影像做比較，發現包含 Stable Diffusion 在內的模型，有些案例複製了訓練資料的內容 [64]。對訓練資料去重 [20]，以及把生成影像與訓練集比對，是標準的第一道防線。

偏見

Bianchi 等人發現，普通的提示（特質、職業、物件）會產生並放大人口統計上的刻板印象，而且在他們的測試中，反刻板印象的提示與系統的防護措施都無法消除它們 [65]。Luccioni 等人提出一種量測生成人物之社會偏見的方法：比較性別與族群字詞造成的變化，與職業造成的變化，並發現在他們測試的三個系統中，邊緣化身分的代表性不足 [66]。偏見評估需要它自己的提示與分析；整體的品質指標看不出來。另見 ML 10，資料來源、偏誤與隱私。

深偽、出處與浮水印

標示生成影像有兩種互補的工具：

- **簽章式出處證明（C2PA）**。 C2PA 規格定義了出處聲明的「清單」（manifest，誰、如何製作了這份素材），它經過密碼學簽章，並綁定到素材上，綁定方式是密碼學雜湊（硬綁定），或指紋、隱形浮水印（軟綁定）[67]。有效的簽章只能證明聲明沒有被竄改；並不證明內容為真，而且存在中介資料裡的清單可以直接被移除。
- **隱形浮水印**。 Tree-Ring 浮水印把一個圖樣藏在擴散取樣器**初始雜訊**的傅立葉轉換中，並透過反轉取樣器來偵測 [68]。Stable Signature 微調潛在解碼器，讓每張生成影像都帶有一組二元簽章，由預訓練的擷取器讀出 [69]。傳統浮水印見 IP 08。

兩者都不是完整的答案。Zhao 等人證明並展示，像素層級的隱形浮水印可以被「再生攻擊」（regeneration attack，先加雜訊，再用生成模型重建）移除 [70]；Saberi 等人則指出，低擾動浮水印在「偵測不到有浮水印的影像」與「誤把真實影像標為有浮水印」之間存在根本的取捨，並展

示了偽造攻擊，能讓真實影像看起來像有浮水印 [71]。要把出處證明與浮水印當成需要權衡的證據，而不是證明。

僅線上版：硬綁定、軟綁定與浮水印 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/notes/硬綁定軟綁定與浮水印/*

失效模式

白話版。今天的生成器很擅長 “a photo of X”，但在需求中精確的部分仍然不可靠：有幾個、哪個是什麼顏色、東西在哪裡、文字寫了什麼。知道典型的失敗，就知道該測什麼。

- **計數**。 “Four apples” 常常得到三個或五個。GenEval 的計數任務（2、3 或 4 個物體）直接量測這點 [55]。
- **空間關係**。 “A cup to the left of a laptop” 常被畫成錯誤的排列；在 GenEval 原始的研究中，最好的模型只在 15% 的影像中畫出正確的位置 [55]。
- **屬性綁定與洩漏**。在 “a red car and a blue bench” 中，顏色可能在物體之間互換，或洩漏到背景；當時 GenEval 中最好的模型，只在 35% 的影像中把屬性綁定對 [55]。Attend-and-Excite 也記載了**災難性忽略**（catastrophic neglect），也就是提示中的某個主體完全沒被畫出來，並在取樣時調整交叉注意力圖，讓每個主體 token 都得到注意力 [28]。
- **文字渲染**。拼錯的招牌與標籤很常見；對字元視而不見的文字編碼器是已有記載的原因之一 [26]，而 Stable Diffusion 3 的消融實驗顯示，T5 編碼器對含有書面文字的提示最重要 [20]。
- **手**。生成的手常有錯誤的手指數量或不規則的形狀；HandRefiner 事後用以重建的手部網格為條件的影像修補來修復 [72]。請注意，物件層級的基準檢查不了這點：GenEval 的作者指出，他們的偵測器數得出人，卻數不出手指 [55]。
- **內容與風格的複製**。點出著名影像、產品或藝術家的提示，可能引出近乎複製的訓練內容 [64], [63]。

怎麼測試。為每種失敗類型建立一小組提示，每個提示用固定的隨機種子生成數張影像，盡量用物件層級的工具檢查，做不到的就用眼睛看，並追蹤各模型版本之間的失敗率。

現代觀點

值得一讀的綜述。

- **Zhang 等人 [73]**（2023）是對文字生成影像擴散模型的完整回顧：簡短介紹擴散、文字條件合成的背景、開創性系統及其改進、影像以外的應用（影片、其他模態、文字引導的編輯），以及開放的挑戰。
- **Croitoru 等人 [74]**（IEEE TPAMI 2023）綜述視覺中的擴散模型，依公式（DDPM、分數網路、SDE）與應用（生成、超解析度、影像修補、編輯、轉換、分割）分組。
- **Huang 等人 [35]**（IEEE TPAMI 2025）依學習策略、使用者輸入條件與編輯任務，為以擴散為基礎的影像編輯分類，特別關注影像修補與外擴，並提出一個基準（EditEval），搭配由大型多模態模型評分的指標。
- **Xiong 等人 [75]**（TMLR 2025）綜述視覺中的自迴歸模型，依表示方式分成以像素、以 token 與以尺度為基礎的模型，並討論它們與其他生成家族的關聯，以及在影像、影片、3-D 與多模

態生成上的用途。

2023–2026 年的最新進展。 (1) **採用流目標與多個文字編碼器的潛在 Transformer**，成為大規模的標準做法，如 Stable Diffusion 3 [20]，建立在 DiT [3] 之上。(2) **自迴歸模型回歸**，靠的是更好的 tokenizer [42]、next-scale 預測 [41] 與連續 token [21]。(3) 以一個 Transformer 同時理解與生成的**統一多模態模型**接連出現 [43], [45], [46], [47], [44]。(4) **少步生成**讓互動使用變得實際可行 [22], [38]。(5) **評估超越了 FID**，走向組合式檢查 [55], [56]、學習式偏好模型 [57], [58] 與整體性基準 [59]。近期的研究也重新檢視像素空間中的生成，以及在表徵空間中的生成；見 ML 08，像素空間、潛在空間與表徵自編碼器。

尚待解決的問題。可靠的計數、空間推理與屬性綁定；任何語言中都清晰可讀的文字；在單張影像層級上與人一致、又不能被取巧操弄的評估指標；在不損失品質的前提下防止記憶；量測並減少偏見；能撐過編輯的出處證明；訓練資料的同意與授權；以及高解析度與影片生成的成本。

僅線上版：*Latent Diffusion*：先壓縮感知資訊，再學習影像分布 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/papers/latent-diffusion 先壓縮感知資訊再學習影像分布/*；*Esser* 等人：*Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*（*ICML 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/image-generation/papers/esser-等人 scaling-rectified-flow-transformers-for-high-resolution-image-synthesisicml-2024/*

重點整理

- 選擇生成器前，先定義條件與輸出限制。
- 生成目標與影像表示是不同選擇，系統也能結合多個家族。
- 檢查文字或空間條件如何進入模型，以及引導在推論時改變什麼。
- 注意力圖呈現內部互動，不保證物件與屬性正確綁定。
- 控制、個人化與編輯需要不同資料及計算資源。
- 結合分布、對齊、組合與人類偏好評估，並報告取樣協定與成本。
- 記憶化、偏差與來源追溯須與視覺品質分開檢查。

練習

1. **對應任務。**對六種生成任務表中的每個任務，寫出對潛在文字生成影像模型而言，條件 c 是什麼樣的張量（形狀與型別），以及它由圖 2 流程中的哪個部分接收。

   提示

   提示會變成 L 個 token 嵌入，經由交叉注意力進入。類別標籤可以寫成文字提示，或是加到時間嵌入上的嵌入。對於影像轉影像與編輯，輸入影像會被編碼成潛在表徵，然後加雜訊到中間強度（SDEdit），或串接到去雜訊網路的輸入通道（InstructPix2Pix）；控制影像經由 ControlNet 的可訓練副本進入；影像修補用的遮罩，決定哪些潛在位置要重新生成。

2. **玩具範例中的忽略。**在交叉注意力程式中，加入第六個字 “dog”，它的嵌入是一個新的隨機向量，沒有任何影像區域與它相似。它的注意力圖長什麼樣子？它對其他字的權重有什麼影響？接著讓球區只像 “ball”（不像 “red”）。哪一張圖會改變？

提示

“dog” 那一行在各處都得到小而無結構的權重，因為沒有任何查詢與它的鍵匹配：沒有任何東西告訴生成器狗該放在哪裡，這正是 Attend-and-Excite 修正的情況，它在取樣時加強被忽略 token 的注意力。softmax 在每一列仍然加總為 1，所以其他字的權重會少一點。把 “red” 從球的特徵中拿掉，會讓 “red” 的圖變暗：顏色字只會注意特徵已經暗示它的地方。

3. **打破指標**。擴充組合式檢查的程式，加入一個候選影像：物體正確，但藍色方塊與其中一個紅圓重疊。玩具偵測器會回報什麼？真實的偵測器式基準需要什麼才能處理它？

提示

重疊的形狀會合併成一個連通元件，所以玩具偵測器看到的物體變少，形狀也不對；即使人可能接受這張影像，精確檢查仍然會失敗。真實的基準使用實例分割，可以把重疊的物體分開，GenEval 在判斷位置時也加入了處理重疊邊界框的規則。每一種自動檢查都有這樣的盲點，這就是基準要報告與人類標註者一致程度的原因。

4. **設計評估**。你必須判斷一個新模型，在「包裝上有簡短品牌名稱的產品照片」上，是否比目前的模型更好。寫出評估計畫：提示、影像數、自動指標、人類研究，以及什麼樣的結果會改變你的決定。

提示

使用固定的提示集，涵蓋不同的產品類型、文字長度與背景；每個提示用多個隨機種子；對渲染出來的文字與要求的字串做 OCR 檢查；對產品類型做物件層級檢查；用 FID 或 CMMD 對照真實產品照片來衡量真實感（固定實作與樣本數）；並做盲測的並排人類比較，評估對齊與品質。事先決定你願意為多大的差異採取行動，並且針對品牌標誌，對訓練集做記憶檢查。

5. **編輯後的出處證明**。一張影像帶有硬綁定的 C2PA 清單，並由帶有 Stable Signature 浮水印的模型生成。之後它被裁切、重新存成 JPEG，並上傳到一個會剝除中介資料的網站。哪些關於它來源的證據可能留下來？為什麼？

提示

清單會隨中介資料一起消失；即使保留下來，硬綁定的雜湊也不再符合裁切並重新編碼後的像素。軟綁定或穩健的隱形浮水印可以撐過適度的裁切與壓縮（Stable Signature 報告在強烈裁切之後仍能偵測），但再生攻擊可以移除像素層級的浮水印。誠實的答案是「也許浮水印還在，但清單不在」，這就是兩者被設計成配合使用的原因。

參考文獻

1. T. Karras, S. Laine and T. Aila, “A Style-Based Generator Architecture for Generative Adversarial Networks,” *CVPR*, 2019. arXiv
2. A. Brock, J. Donahue and K. Simonyan, “Large Scale GAN Training for High Fidelity Natural Image Synthesis,” *ICLR*, 2019. arXiv

3. W. Peebles and S. Xie, “Scalable Diffusion Models with Transformers,” *ICCV*, 2023. CVF
4. C. Saharia, W. Chan, S. Saxena et al., “Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding,” *NeurIPS*, 2022. arXiv ·project page
5. R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” *CVPR*, 2022. CVF
6. C. Meng, Y. He, Y. Song et al., “SDEdit: Guided Image Synthesis and Editing with Stochastic Differential Equations,” *ICLR*, 2022. arXiv
7. L. Zhang, A. Rao and M. Agrawala, “Adding Conditional Control to Text-to-Image Diffusion Models,” *ICCV*, 2023. CVF
8. A. Hertz, R. Mokady, J. Tenenbaum et al., “Prompt-to-Prompt Image Editing with Cross-Attention Control,” *ICLR*, 2023. arXiv
9. T. Brooks, A. Holynski and A. A. Efros, “InstructPix2Pix: Learning to Follow Image Editing Instructions,” *CVPR*, 2023. CVF
10. A. Lugmayr, M. Danelljan, A. Romero et al., “RePaint: Inpainting using Denoising Diffusion Probabilistic Models,” *CVPR*, 2022. CVF
11. J. Ho and T. Salimans, “Classifier-Free Diffusion Guidance,” NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications; arXiv:2207.12598, 2022. arXiv
12. I. J. Goodfellow, J. Pouget-Abadie, M. Mirza et al., “Generative Adversarial Nets,” *NeurIPS*, 2014. arXiv
13. D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *ICLR*, 2014. arXiv
14. A. van den Oord, O. Vinyals and K. Kavukcuoglu, “Neural Discrete Representation Learning,” *NeurIPS*, 2017. arXiv
15. P. Esser, R. Rombach and B. Ommer, “Taming Transformers for High-Resolution Image Synthesis,” *CVPR*, 2021. CVF
16. J. Ho, A. Jain and P. Abbeel, “Denoising Diffusion Probabilistic Models,” *NeurIPS*, 2020. arXiv
17. P. Dhariwal and A. Nichol, “Diffusion Models Beat GANs on Image Synthesis,” *NeurIPS*, 2021. arXiv
18. Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel and M. Le, “Flow Matching for Generative Modeling,” *ICLR*, 2023. arXiv
19. X. Liu, C. Gong and Q. Liu, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow,” *ICLR*, 2023. arXiv
20. P. Esser, S. Kulal, A. Blattmann et al., “Scaling Rectified Flow Transformers for High-Resolution Image Synthesis,” *ICML*, PMLR 235, 2024. arXiv
21. T. Li, Y. Tian, H. Li, M. Deng and K. He, “Autoregressive Image Generation without Vector Quantization,” *NeurIPS*, 2024. arXiv
22. A. Sauer, D. Lorenz, A. Blattmann and R. Rombach, “Adversarial Diffusion Distillation,” *ECCV*, LNCS, pp. 87–103, 2024. doi ·arXiv
23. A. Radford, J. W. Kim, C. Hallacy et al., “Learning Transferable Visual Models From Natural Language Supervision,” *ICML*, PMLR 139, 2021. arXiv
24. C. Raffel, N. Shazeer, A. Roberts et al., “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” *Journal of Machine Learning Research*, vol. 21, 2020. arXiv
25. D. Podell, Z. English, K. Lacey et al., “SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis,” *ICLR*, 2024. arXiv
26. R. Liu, D. Garrette, C. Saharia et al., “Character-Aware Models Improve Visual Text Rendering,”

ACL, 2023. ACL Anthology
27. A. Ramesh, P. Dhariwal, A. Nichol, C. Chu and M. Chen, "Hierarchical Text-Conditional Image Generation with CLIP Latents," arXiv:2204.06125, 2022. arXiv
28. H. Chefer, Y. Alaluf, Y. Vinker, L. Wolf and D. Cohen-Or, "Attend-and-Excite: Attention-Based Semantic Guidance for Text-to-Image Diffusion Models," *ACM Transactions on Graphics*, vol. 42, no. 4 (SIGGRAPH), 2023. doi ·arXiv
29. O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *MICCAI*, 2015. arXiv
30. J. Yu, Y. Xu, J. Y. Koh et al., "Scaling Autoregressive Models for Content-Rich Text-to-Image Generation," *Transactions on Machine Learning Research*, 2022. arXiv
31. H. Chang, H. Zhang, J. Barber et al., "Muse: Text-To-Image Generation via Masked Generative Transformers," *ICML*, PMLR 202, 2023. arXiv
32. R. Gal, Y. Alaluf, Y. Atzmon et al., "An Image is Worth One Word: Personalizing Text-to-Image Generation using Textual Inversion," *ICLR*, 2023. arXiv
33. N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein and K. Aberman, "DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation," *CVPR*, 2023. CVF
34. E. J. Hu, Y. Shen, P. Wallis et al., "LoRA: Low-Rank Adaptation of Large Language Models," *ICLR*, 2022. arXiv
35. Y. Huang, J. Huang, Y. Liu et al., "Diffusion Model-Based Image Editing: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025. doi ·arXiv
36. J. Song, C. Meng and S. Ermon, "Denoising Diffusion Implicit Models," *ICLR*, 2021. arXiv
37. T. Salimans and J. Ho, "Progressive Distillation for Fast Sampling of Diffusion Models," *ICLR*, 2022. arXiv
38. Y. Song, P. Dhariwal, M. Chen and I. Sutskever, "Consistency Models," *ICML*, PMLR 202, 2023. arXiv
39. P. Sun, Y. Jiang, S. Chen et al., "Autoregressive Model Beats Diffusion: Llama for Scalable Image Generation," arXiv:2406.06525, 2024. arXiv
40. H. Chang, H. Zhang, L. Jiang, C. Liu and W. T. Freeman, "MaskGIT: Masked Generative Image Transformer," *CVPR*, 2022. CVF
41. K. Tian, Y. Jiang, Z. Yuan, B. Peng and L. Wang, "Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction," *NeurIPS*, 2024. arXiv
42. L. Yu, J. Lezama, N. B. Gundavarapu et al., "Language Model Beats Diffusion — Tokenizer is Key to Visual Generation," *ICLR*, 2024. arXiv
43. Chameleon Team, "Chameleon: Mixed-Modal Early-Fusion Foundation Models," arXiv:2405.09818, 2024. arXiv
44. X. Wang, Y. Cui, J. Wang et al., "Multimodal learning with next-token prediction for large multimodal models," *Nature*, vol. 650, pp. 327–333, 2026 (preprint "Emu3: Next-Token Prediction is All You Need," 2024). doi ·arXiv
45. C. Zhou, L. Yu, A. Babu et al., "Transfusion: Predict the Next Token and Diffuse Images with One Multi-Modal Model," *ICLR*, 2025. arXiv
46. J. Xie, W. Mao, Z. Bai et al., "Show-o: One Single Transformer to Unify Multimodal Understanding and Generation," *ICLR*, 2025. arXiv
47. C. Wu, X. Chen, Z. Wu et al., "Janus: Decoupling Visual Encoding for Unified Multimodal Understanding and Generation," *CVPR*, 2025. CVF

48. T.-Y. Lin, M. Maire, S. Belongie et al., “Microsoft COCO: Common Objects in Context,” *ECCV*, 2014. arXiv
49. M. J. Chong and D. Forsyth, “Effectively Unbiased FID and Inception Score and Where to Find Them,” *CVPR*, 2020. arXiv
50. G. Parmar, R. Zhang and J.-Y. Zhu, “On Aliased Resizing and Surprising Subtleties in GAN Evaluation,” *CVPR*, 2022. CVF
51. T. Kynkäänniemi, T. Karras, M. Aittala, T. Aila and J. Lehtinen, “The Role of ImageNet Classes in Fréchet Inception Distance,” *ICLR*, 2023. arXiv
52. G. Stein, J. C. Cresswell, R. Hosseinzadeh et al., “Exposing Flaws of Generative Model Evaluation Metrics and Their Unfair Treatment of Diffusion Models,” *NeurIPS*, 2023. arXiv
53. S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti and S. Kumar, “Rethinking FID: Towards a Better Evaluation Metric for Image Generation,” *CVPR*, 2024. CVF
54. J. Hessel, A. Holtzman, M. Forbes, R. Le Bras and Y. Choi, “CLIPScore: A Reference-free Evaluation Metric for Image Captioning,” *EMNLP*, 2021. ACL Anthology
55. D. Ghosh, H. Hajishirzi and L. Schmidt, “GenEval: An Object-Focused Framework for Evaluating Text-to-Image Alignment,” *NeurIPS Datasets and Benchmarks Track*, 2023. arXiv
56. K. Huang, K. Sun, E. Xie, Z. Li and X. Liu, “T2I-CompBench: A Comprehensive Benchmark for Open-world Compositional Text-to-image Generation,” *NeurIPS Datasets and Benchmarks Track*, 2023. proceedings
57. J. Xu, X. Liu, Y. Wu et al., “ImageReward: Learning and Evaluating Human Preferences for Text-to-Image Generation,” *NeurIPS*, 2023. arXiv
58. Y. Kirstain, A. Polyak, U. Singer et al., “Pick-a-Pic: An Open Dataset of User Preferences for Text-to-Image Generation,” *NeurIPS*, 2023. arXiv
59. T. Lee, M. Yasunaga, C. Meng et al., “Holistic Evaluation of Text-To-Image Models,” *NeurIPS Datasets and Benchmarks Track*, 2023. arXiv
60. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” *CVPR*, 2009. doi
61. C. Schuhmann, R. Beaumont, R. Vencu et al., “LAION-5B: An Open Large-Scale Dataset for Training Next Generation Image-Text Models,” *NeurIPS Datasets and Benchmarks Track*, 2022. arXiv
62. A. Birhane, V. U. Prabhu and E. Kahembwe, “Multimodal datasets: misogyny, pornography, and malignant stereotypes,” arXiv:2110.01963, 2021. arXiv
63. N. Carlini, J. Hayes, M. Nasr et al., “Extracting Training Data from Diffusion Models,” *USENIX Security Symposium*, 2023. USENIX ·arXiv
64. G. Somepalli, V. Singla, M. Goldblum, J. Geiping and T. Goldstein, “Diffusion Art or Digital Forgery? Investigating Data Replication in Diffusion Models,” *CVPR*, 2023. CVF
65. F. Bianchi, P. Kalluri, E. Durmus et al., “Easily Accessible Text-to-Image Generation Amplifies Demographic Stereotypes at Large Scale,” *ACM FAccT*, 2023. doi
66. A. S. Luccioni, C. Akiki, M. Mitchell and Y. Jernite, “Stable Bias: Analyzing Societal Representations in Diffusion Models,” *NeurIPS Datasets and Benchmarks Track*, 2023. arXiv
67. Coalition for Content Provenance and Authenticity (C2PA), “C2PA Technical Specification,” version 2.1, 2024. specification
68. Y. Wen, J. Kirchenbauer, J. Geiping and T. Goldstein, “Tree-Ring Watermarks: Fingerprints for Diffusion Images that are Invisible and Robust,” *NeurIPS*, 2023. arXiv

69. P. Fernandez, G. Couairon, H. Jégou, M. Douze and T. Furon, “The Stable Signature: Rooting Watermarks in Latent Diffusion Models,” *ICCV*, 2023. CVF
70. X. Zhao, K. Zhang, Z. Su et al., “Invisible Image Watermarks Are Provably Removable Using Generative AI,” *NeurIPS*, 2024. arXiv
71. M. Saberi, V. S. Sadasivan, K. Rezaei et al., “Robustness of AI-Image Detectors: Fundamental Limits and Practical Attacks,” *ICLR*, 2024. arXiv
72. W. Lu, Y. Xu, J. Zhang, C. Wang and D. Tao, “HandRefiner: Refining Malformed Hands in Generated Images by Diffusion-based Conditional Inpainting,” *ACM Multimedia*, 2024, pp. 7085–7093. doi
73. C. Zhang, C. Zhang, M. Zhang, I. S. Kweon and J. Kim, “Text-to-image Diffusion Models in Generative AI: A Survey,” arXiv:2303.07909, 2023. arXiv
74. F.-A. Croitoru, V. Hondru, R. T. Ionescu and M. Shah, “Diffusion Models in Vision: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. doi
75. J. Xiong, G. Liu, L. Huang et al., “Autoregressive Models in Vision: A Survey,” *Transactions on Machine Learning Research*, 2025. arXiv

DLCV 12A・影片生成

影片生成器如何建構與評量：影片自編碼器與 tokenizer、分解式與完整時空注意力、文字生成影片與影像生成影片、相機與運動控制、長影片、時間一致性、自迴歸與擴散生成、FVD 與基準套件、資料、風險與失效模式。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/

你將學到

- 區分影格品質、運動品質與時間一致性。
- 理解空間／時間壓縮及 token 預算取捨。
- 比較影像模型的時間擴充與原生影片注意力。
- 追蹤文字、影像、相機與運動條件如何作用於長影片生成。
- 以互補指標及人類研究評估整段聯合與串流模型。

先看全貌

影片生成器根據一個請求產生一小段會動的畫面：一句話（「一隻狗在日落時沿著海灘奔跑」）、一張要讓它動起來的靜態影像、一條相機路徑，或是要接續下去的一段短片開頭。結果必須每個影格都好看，動作要可信，而且東西從一個影格到下一個影格都要維持同一個東西。

> **用最簡單的話說**
>
> 想像一本翻頁動畫書。如果你憑記憶把每一頁各畫一次，狗身上的斑點會到處亂跑，耳朵忽大忽小，快速翻頁時整本書看起來像在閃爍。厲害的畫家會先一次把所有頁面的大致動作草擬出來，再補上細節，讓每個斑點在狗奔跑時都停留在牠身體的同一個位置。影片生成器做的是同一件事：它在壓縮後的草圖空間裡，把許多影格放在一起規劃，讓細節跟著物體移動，而不是到處跳動。

為什麼重要。影片生成用在創作工具、讓照片動起來、產生訓練資料與模擬場景，也被視為機器人與車輛可能的「世界模擬器」。它同時是最昂貴的一種視覺生成：五秒的片段包含幾十張必須彼此吻合的影像。大部分的工程都圍繞一個取捨：為了一致性，要把所有影格一起建模，但又不能讓每個影格的每個像素都和其他影格的每個像素互相作用。

本篇的位置。影像方面的內容在別處已經講過，這裡不重複。DLCV 12 說明文字編碼器、交叉注意力、實務上的引導、控制與編輯、文字生成影像的評估與風險。ML 08 推導擴散、無分類器引導、潛在擴散、擴散 Transformer、流匹配與少步取樣器，其中的小節影片加入時間壓縮與一致性問題是簡短的預告，本篇把它展開。ML 09 涵蓋自迴歸生成與視覺 tokenizer。DLCV 13 討論以動作為條件的影片預測、大型生成式世界模型，以及逼真的影片是否代表理解物理。DLCV 09A.3 介紹光流、扭曲與時間一致性指標。這裡只談**生成**影片特有的部分。所有程式都只用 NumPy、SciPy 與 scikit-image，在 CPU 上數秒內執行完畢，不需要下載網路權重。

影片為什麼比影像難

白話版。影片生成須兼顧單格品質、合理運動與身分一致性；獨立生成的好影格仍可能組成閃爍影片。

任務

下面每一種任務，都是從條件分布 $p(x \mid c)$ 取樣一支影片 $x \in \mathbb{R}^{F\times H\times W\times 3}$（$F$ 個影格，高 H、寬 W、三個色彩通道）；各任務之間唯一的差別是條件 c。

任務	條件 c	本篇中的例子
文字生成影片（T2V）	一段文字提示	Make-A-Video [1]、CogVideoX [2]、Wan [3]
影像生成影片（I2V）	第一個影格，常加上提示	Stable Video Diffusion [4]、DynamiCrafter [5]
影片延伸或預測	同一段片段較早的影格	VDM [6]、Video LDM [7]、StreamingT2V [8]
影格內插	要在中間補齊的關鍵影格	Make-A-Video [1]、Stable Video Diffusion [4]
影片編輯	輸入影片加上編輯要求	Tune-A-Video [9]、Text2Video-Zero [10]
受控生成	相機路徑或物體軌跡	MotionCtrl [11]、CameraCtrl [12]、DragNUWA [13]

和影像一樣（DLCV 12），同一個訓練好的模型常常能涵蓋好幾列：第一個影格、較早的影格或關鍵影格都是額外的條件，再用遮罩決定哪些影格要生成。

對每支影片的三項要求

- **逐影格品質**。每個影格單獨看都應該像一張真實的影像：清晰、沒有瑕疵、構圖良好。
- **動作品質**。片段應該會動，而且動作要合理：速度、方向與形變都要像真的。幾乎不動的片段即使每個影格都完美，仍然是失敗。
- **時間一致性**。同一個物體應該一直是同一個物體。有三種失敗各有名稱。**閃爍**（flicker）是亮度或細緻紋理在影格之間快速變化，而且沒有任何真實的動作可以解釋。**身分漂移**（identity drift）是主體的外觀（臉、標誌、襯衫上的花紋）在片段中緩慢改變。**物體恆存性**（object permanence）的失敗，是物體被遮住之後消失、變樣，或錯誤地重新出現。

這三項要求彼此拉扯。要消除閃爍，最容易的方法是幾乎不產生動作；要得到戲劇性的動作，最容易的方法是讓細節隨意變動。本篇後面的評估一節，必須同時檢查這三者。

成本

影片很大。一段 $F = 81$ 個影格、480×832 像素的片段（以每秒 16 個影格計，約五秒）有 $81 \cdot 480 \cdot 832 \cdot 3 \approx 9.7 \times 10^7$ 個數字，大約等於 81 張影像。生成器也不能把影格一個接一個地處

理：一致性要求資訊能在相距很遠的影格之間流動。使用注意力時，讓每個位置看到其他每個位置的成本，會隨位置數量的**平方**增長。有兩個設計手段可以控制它：生成之前先**壓縮**影片（下一節），以及**限制**哪些位置可以注意到哪些位置（再下一節）。

壓縮影片：3D VAE 與影片 tokenizer

白話版。影片自編碼器壓縮空間與時間以降低生成成本，其重建品質限制可恢復的細節與運動。

為什麼不直接用影像自編碼器逐影格壓縮？

最早的潛在影片模型沿用潛在影像擴散模型的自編碼器 [14]，把每個影格分別編碼。Video LDM 的作者報告，這會造成閃爍，因為影像自編碼器只用影像訓練，不知道連續的影格應該要解碼得一致；他們用影片微調**解碼器**，加入時間層與 3D 卷積判別器，並發現這一步對好的結果至關重要 [7]。Stable Video Diffusion 同樣建立在預訓練的影像潛在模型上，只在空間上壓縮 [4]、[3]。逐影格的自編碼器也是每個影格保留一個潛在影格，所以對時間方向的成本毫無幫助。

3D 因果 VAE

3D VAE 讓卷積同時在時間與空間上進行，因此兩個方向都能降取樣 [2]。最近的系統多半讓它在時間上具**因果性**：每個時間卷積只在片段開頭補零，所以一個潛在影格只依賴目前與更早的影格，絕不依賴後面的影格 [15]、[2]。一個有用的結果是，第一個影格會被單獨編碼。因此常用的影格排列是「$1 + T$」：第一個影格成為第一個潛在影格，之後每 f_t 個影格成為再多一個潛在影格。靜態影像就只是一支一個影格的影片，這讓同一個自編碼器、同一個生成器能同時用影像與影片訓練 [15]、[16]、[3]。

設一段片段有 $F = 1 + T$ 個影格，尺寸為 $H \times W$，在時間上壓縮 f_t 倍、在每個空間軸上壓縮 f_s 倍，並產生 C 個潛在通道。潛在網格與 transformer 的 token 數量為

$$T' = 1 + \frac{T}{f_t}, \qquad H' = \frac{H}{f_s}, \qquad W' = \frac{W}{f_s}, \qquad L = T' \cdot \frac{H'}{p} \cdot \frac{W'}{p},$$

其中 T'、H'、W' 分別是潛在影格數、列數與行數，p 是 transformer 合成一個 token 的正方形 patch 邊長（常取 $p = 2$），L 是序列長度。Wan 的技術報告對它的 $4\times8\times8$ 自編碼器與 2×2 patch，給出的正是這個數量：$L = (1 + T/4) \cdot H/16 \cdot W/16$ [3]。兩個數字可以概括一種設計：**數值的壓縮率** $\frac{3FHW}{CT'H'W'}$，以及**每個 token 的像素數**，約為 $f_t f_s^2 p^2$。由於潛在表徵的通道數多於 RGB 的 3 個，影片 VAE 對 token 的壓縮遠大於對數值的壓縮。

圖 1 示範 17 個影格的片段的計算。

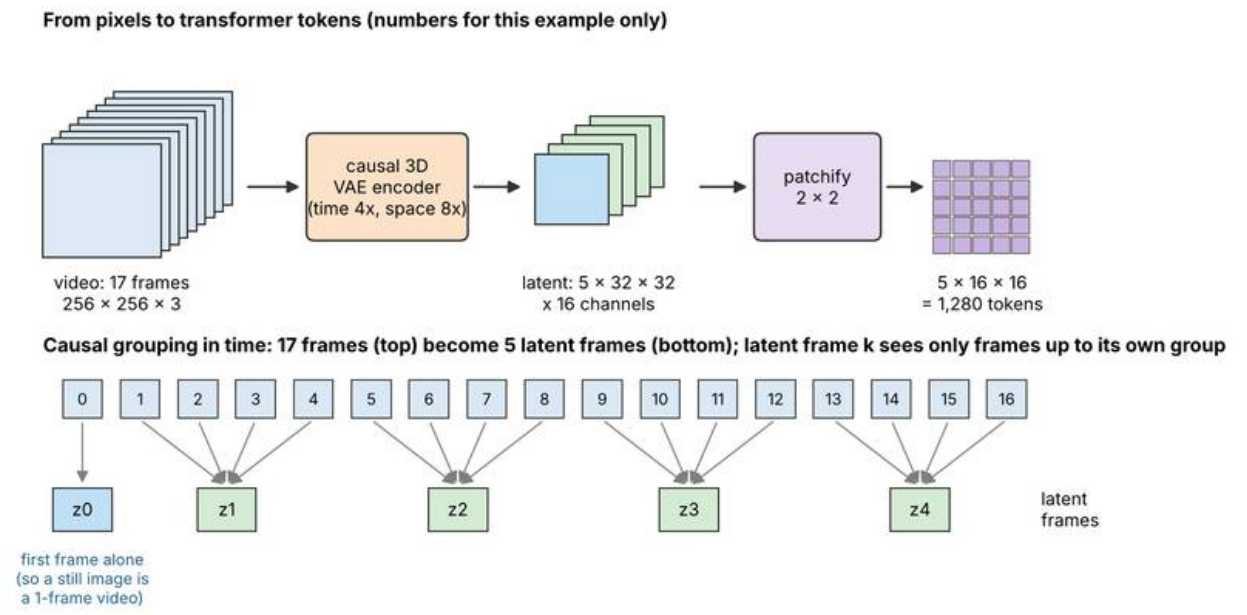


圖 *1* — 以數字看因果影片 *VAE*。第一個影格單獨編碼；之後每個潛在影格摘要四個影格，而且看不到未來的影格。接著 *2 × 2* 的 *patchify* 步驟決定 *transformer* 的序列長度。

已發表的系統都有記載它們的壓縮倍率。下表列出幾個（這是設計選擇的地圖，不是排名）。

系統	壓縮（時間 × 高 × 寬）	潛在通道數	備註
Stable Video Diffusion [4]	1 × 8 × 8	4	影像自編碼器，沒有時間壓縮（倍率依 [3] 所列）
CogVideoX [2]	4 × 8 × 8	16	3D 因果 VAE
HunyuanVideo [16]	4 × 8 × 8	16	因果 3D VAE，輸入 $(T+1)$ 個影格
Wan [3]	4 × 8 × 8	16	因果；第一個影格只在空間上壓縮
Movie Gen [17]	8 × 8 × 8	16	由影像自編碼器膨脹而來的時間自編碼器
LTX-Video [18]	8 × 32 × 32	128	整體約 1:192；沒有獨立的 patchify 步驟
VideoPoet 中的 MAGVIT-v2 [19]	128 × 128 的 17 個影格 → 5 × 16 × 16 個離散 token	詞彙表大小 2^{18}	供語言模型使用的離散 token

LTX-Video 用每個 token 的像素數來比較：4 × 8 × 8 自編碼器再加上 2 × 2 patch，是每個 token 1,024 個像素，而它自己的設計是 8,192 個 [18]。進一步提高壓縮是有代價的。CogVideoX 的消融實驗發現，8 × 16 × 16 的設定即使增加潛在通道數，生成器的訓練也很難收斂 [2]；而 LTX-Video 則要求解碼器在像素空間中執行最後一步去雜訊，以找回強壓縮所損失的細節 [18]。

僅線上版：因果 *3D* 卷積與 *1 + T* 影格排列 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/notes/因果-3d-卷積與-1–t-影格排列/*

連續潛在表徵或離散 token

和影像模型一樣（DLCV 12），影片生成器若不是在**連續**的潛在網格（VAE，供擴散與流模型使用）上運作，就是在學得的 codebook 產生的**離散 token**（供遮罩式與自迴歸 token 模型使用）上運作。MAGVIT 提出一種 3D tokenizer，把影片量化成時空 token，供遮罩式生成使用 [20]。MAGVIT-v2 以因果 3D CNN tokenizer 和**無查表量化器**（lookup-free quantizer；詞彙表可以變得很大）取代了它，並且在作者的測試中，搭配它的語言模型在影像與影片基準上勝過擴散模型 [15]；codebook 及其問題見 ML 09。

程式：潛在表徵大小與注意力成本

這段程式針對一段 81 個影格、480 × 832 的片段，在三種自編碼器下計算潛在網格、token 數量，以及一層注意力中的查詢–鍵配對數。配對數預告了下一節的內容：「完整」是讓每個 token 注意每個 token，「分解式」則把注意力拆成每個潛在影格內的空間部分，以及每個位置上的時間部分。

```
def latent_grid(frames, height, width, ft, fs, channels=16):
    """Latent shape of a causal video VAE: the first frame alone, then groups of ft
     ↪  frames."""
    t = 1 + (frames - 1) // ft
    return t, height // fs, width // fs, channels

def attention_pairs(t, s):
    """Query-key pairs in one attention layer over t latent frames of s tokens each."""
    full = (t * s) ** 2                 # every token attends to every token
    factorized = t * s**2 + s * t**2    # spatial attention per frame + temporal attention
     ↪  per position
    return full, factorized

frames, H, W, patch = 81, 480, 832, 2   # about 5 s at 16 fps; DiT patches of 2 x 2 latent
 ↪  positions
pixels = frames * H * W * 3
print(f"pixel values: {pixels:,}")
for name, ft, fs, c in [("image VAE, every frame", 1, 8, 4),
                        ("video VAE 4x8x8", 4, 8, 16),
                        ("video VAE 8x16x16", 8, 16, 64)]:
    t, h, w, c = latent_grid(frames, H, W, ft, fs, c)
    s = (h // patch) * (w // patch)     # tokens per latent frame after patchify
    full, fact = attention_pairs(t, s)
    print(f"{name:23s} latent {t}x{h}x{w}x{c} ({pixels / (t*h*w*c):5.1f}x fewer numbers)  "
          f"tokens {t*s:7,}  full {full:.1e}  factorized {fact:.1e}  ratio {full/fact:4.0f}")
```

輸出：

```
pixel values: 97,044,480
image VAE, every frame  latent 81x60x104x4 ( 48.0x fewer numbers)  tokens 126,360  full
 ↪  1.6e+10  factorized 2.1e+08  ratio   77
video VAE 4x8x8         latent 21x60x104x16 ( 46.3x fewer numbers)  tokens  32,760  full
 ↪  1.1e+09  factorized 5.2e+07  ratio   21
```

```
video VAE 8x16x16       latent 11x30x52x64 ( 88.4x fewer numbers)  tokens   4,290  full
 ↪  1.8e+07  factorized 1.7e+06  ratio   11
```

（所有數字都是我們自己的算術結果；帶有 64 個通道的 $8 \times 16 \times 16$ 那一列是假設的設定，不是某個特定模型。）有三點心得。第一，$4 \times 8 \times 8$ 影片 VAE 保留的數值數量和逐影格影像 VAE 差不多，但它把 **token** 減少為四分之一、把注意力配對數減少為十六分之一，因為每個潛在位置帶有更多通道。第二，32,760 個 token 與 Wan 針對這個片段大小的公式相符。第三，對 32,760 個 token 做完整注意力，每層大約有 10^9 個查詢–鍵配對，這就是為什麼完整注意力與分解式注意力的選擇，加上壓縮倍率，決定了一個模型負擔得起什麼。

架構：從膨脹的影像模型到影片 Transformer

白話版。一條路線為影像模型加入時間層，另一條訓練影片 Transformer；兩者都須決定資訊如何跨越時空。

膨脹預訓練的影像模型

想法是重用影像模型已經知道的外觀知識，只學習動作。Video Diffusion Models（VDM）把影像擴散 U-Net 變成**在空間與時間上分解的 3D U-Net**：每個 3×3 卷積變成對每個影格分別作用的 $1 \times 3 \times 3$ 卷積，每個空間注意力區塊仍在影格內做注意力，其後再加一個新的時間注意力區塊，在每個空間位置上對各影格做注意力 [6]。由於時間注意力可以關掉，同一個網路可以同時用單張影像與影片訓練，作者發現這種聯合訓練對樣本品質很重要 [6]。

後來的系統保留預訓練的影像層，並插入**初始化為什麼都不做**的新時間層，讓膨脹後的模型一開始與影像模型完全相同。令 h 為某個區塊的特徵，並重新排列成把各影格當作一批影像。膨脹後的區塊計算

$$h_{\text{out}} = h + Z(\,\text{TempAttn}(h)),$$

其中 TempAttn 在每個空間位置上沿影格軸做注意力，Z 是權重從零開始的線性投影，因此訓練一開始 $h_{\text{out}} = h$（與 DLCV 12 中 ControlNet 的零卷積是同一個想法）。訓練再把模型從「一批各自獨立的影像」逐步推向「一支影片」。Make-A-Video [1]、Video LDM [7]、AnimateDiff [21] 與 Open-Sora [22] 都用了這個想法的某個版本：恆等初始化的層、學得的混合權重，或零初始化的投影。Make-A-Video 也展示了好處：它從影像–文字配對學習外觀及其與文字的連結，只從無標註的影片學習動作，因此不需要成對的文字–影片資料 [1]。

僅線上版：從影像模型起步的四種方式 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/notes/從影像模型起步的四種方式/*

分解式或完整時空注意力

白話版。一個 token 可以看到片段中其他每個 token（完整注意力），也可以先看自己所在影格的 token，再看其他影格中同一個位置的 token（分解式注意力）。分解式注意力便宜得多；完整注意力則能追蹤在影格之間移動很遠的物體。

設一段片段有 T 個潛在影格，每個影格 S 個 token，總共 $N = TS$ 個 token。一層完整注意力比較 $N^2 = T^2S^2$ 個查詢–鍵配對。分解式注意力在每個影格內做空間注意力（T 個大小為 S 的問題，所以是 TS^2 個配對），並在每個位置做時間注意力（S 個大小為 T 的問題，所以是 ST^2 個配對）。比值為

$$\frac{T^2S^2}{TS^2 + ST^2} = \frac{TS}{S + T},$$

當 $S \gg T$ 時它接近 T，就像上面的程式那樣（21 個潛在影格時比值為 21）。分解式的樣式來自影片理解：TimeSformer 比較了幾種自注意力方案，發現「分離式」（divided）時空注意力在它試過的方案中，對影片分類最好 [23]，ViViT 則提出了幾種把空間與時間維度分解的變體 [24]。Latte 在潛在影片擴散 transformer 內比較了四種這類分解方式（交錯的空間與時間區塊、後段融合，以及在區塊內拆分注意力的兩種方式），發現交錯的空間與時間 transformer 區塊在它的實驗中表現最好 [25]。

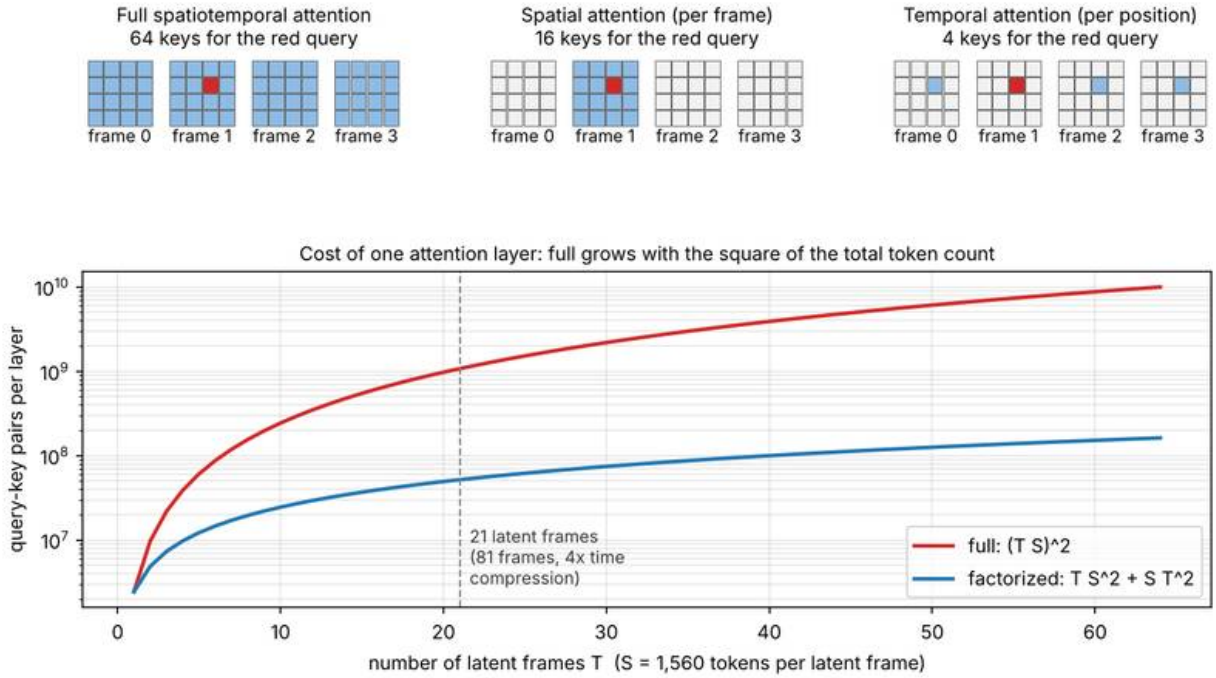


圖 2 — 在完整、空間與時間注意力下，一個查詢能看到哪些鍵，以及這要付出多少成本。分解式注意力要在不同影格、相距很遠的位置之間傳遞資訊，只能間接地透過好幾層來完成。

支持完整注意力的理由與動作有關。CogVideoX 主張，在空間與時間注意力分開時，一個在相鄰影格之間移動很遠的物體，無法直接注意到下一個影格中的它自己；資訊必須間接地經過其他 patch，這使得大幅度動作的一致性很難學；因此它對文字與影片 token 一起使用完整的 3D 注意力 [2]。HunyuanVideo 選擇完整注意力，理由是結果比分離式注意力更好、影像與影片可以用同一種設計，並且能重用為語言模型開發的加速方法 [16]。Wan 同樣使用完整時空注意力 [3]；Open-Sora Plan 從分解式的「2+1D」去雜訊器改成 3D 完整注意力，並加入更便宜的稀疏注意力運算 [26]；LTX-Video 使用極高的壓縮，正是為了讓完整注意力維持在負擔得起的範圍 [18]。實用的準則是：token 數量允許時，優先採用完整注意力，而自編碼器的設計目標，就是讓 token

數量允許這麼做。

級聯，還是整支影片用同一個模型

級聯（cascade）先生成一段短、小、低影格率的影片，再用另外的模型改善它。Imagen Video 使用七個子模型（一個基礎模型、三個空間超解析度模型和三個時間超解析度模型），得到 128 個影格、1280 × 768、每秒 24 個影格、約 5.3 秒的影片 [27]。Make-A-Video 結合一個基礎模型、一個影格內插網路和兩個超解析度模型 [1]；Video LDM 先生成稀疏的關鍵影格，再用內插模型補齊影格 [7]。級聯讓每個階段都保持小巧，但基礎階段只用少數影格就決定了整體動作。Lumiere 主張，先生成相距很遠的關鍵影格再內插，會讓整體的時間一致性變得困難，因此它提出**時空 U-Net**（Space-Time U-Net），在時間與空間上都做降取樣與上取樣，一次就生成完整影格率的片段 [28]。帶有時間壓縮的大型潛在 transformer 走的是同樣的單階段路線；Movie Gen 指出，它的 8× 時間壓縮讓它不必再使用早期工作中的影格內插模型 [17]。

代表性系統地圖

下表列出設計已記載於論文或技術報告的系統。這是設計選擇的地圖，不是排名；圖 3 則把它們放在時間軸上。

系統	生成器與骨幹	在哪裡生成	備註
VDM [6]	擴散，分解式 3D U-Net	像素	影像與影片聯合訓練
Make-A-Video [1]	擴散，膨脹的 U-Net，級聯	像素	不需成對的文字–影片資料
Imagen Video [27]	擴散，七個模型的級聯	像素	空間與時間超解析度
Video LDM [7]	擴散，凍結的影像 LDM 加時間層	影像自編碼器的潛在空間	微調過的時間解碼器
Stable Video Diffusion [4]	擴散，膨脹的潛在 U-Net	影像自編碼器的潛在空間	搭配資料篩選的分階段訓練
Lumiere [28]	擴散，時空 U-Net	像素	整支影片一次生成
Latte [25]	擴散 transformer，分解式	潛在空間	比較四種分解方式
CogVideoX [2]	擴散 transformer，完整 3D 注意力	3D 因果 VAE 的潛在空間	文字與影片 token 串接
Open-Sora [22]	擴散 transformer，分解式（STDiT）	3D 自編碼器的潛在空間	完全開放的訓練程式碼與權重
Open-Sora Plan [26]	擴散 transformer，3D 注意力與較便宜的稀疏變體（Skiparse）	wavelet-flow VAE 的潛在空間	附資料管線的開源專案
HunyuanVideo [16]	流匹配 transformer，完整注意力	因果 3D VAE 的潛在空間	超過 130 億個參數
Movie Gen [17]	流匹配 transformer	時間自編碼器的潛在空間	300 億個參數，同步音訊
Wan [3]	流匹配擴散 transformer，完整注意力	因果 3D VAE 的潛在空間	1.3B 與 14B 兩種模型
LTX-Video [18]	潛在擴散 transformer，完整注意力	高度壓縮的潛在空間	解碼器負責最後一步去雜訊
MAGVIT [20]、VideoPoet [19]	遮罩式或自迴歸 token transformer	離散 token	見自迴歸生成一節

流匹配與擴散 transformer 在 ML 08 與 ML 08，擴散 Transformer 推導 [29]、[30]。上表的參數量是論文中所陳述的數字。

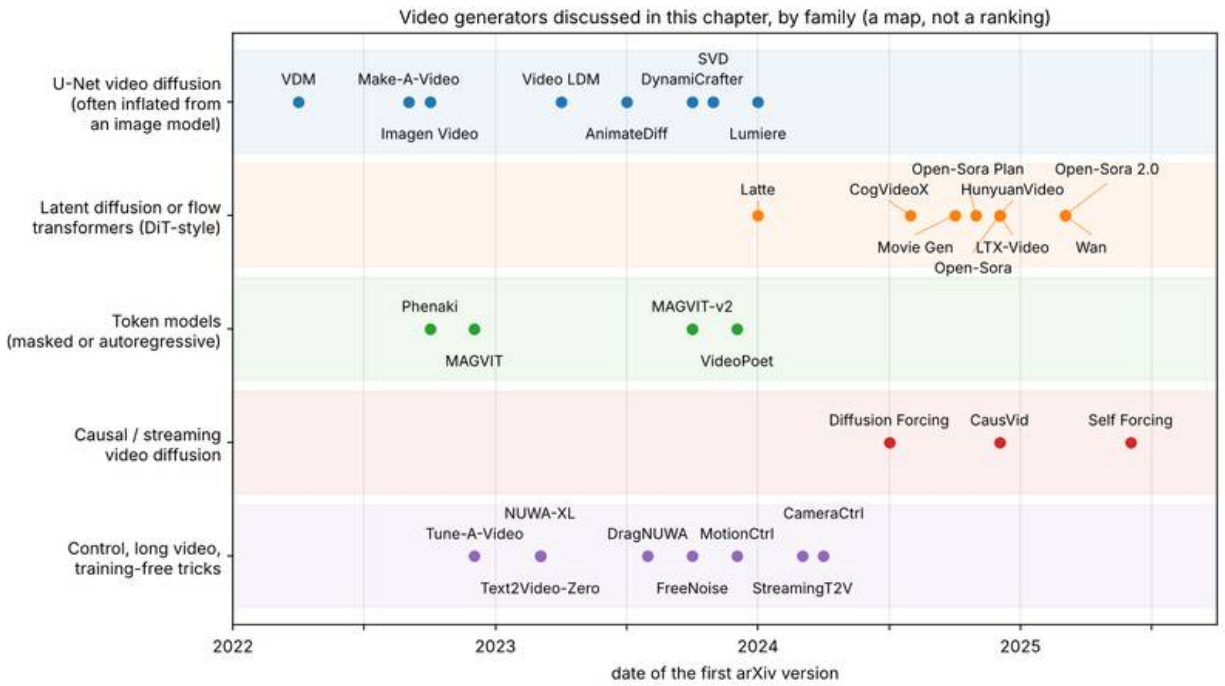


圖 3 — 本篇的系統依家族與其 *arXiv* 初版日期排列。重心從膨脹的 *U-Net*（*2022–2023*）移到使用完整注意力的潛在 *transformer*（*2024–2025*），*token* 模型與因果擴散則是並行的兩條線。

條件：文字生成影片與影像生成影片

白話版。提示進入影片模型的方式，和進入影像模型一樣，都是對文字編碼器的輸出做注意力。影像生成影片中的第一個影格，則以額外輸入的方式進入，每個生成的影格都看得到它。和影像的差別在於，條件必須對整支片段都成立，而且「動多少」本身就是要控制的東西。

文字生成影片

文字編碼器、交叉注意力與文字–影像聯合注意力的說明見 DLCV 12。影片系統使用同樣的機制。Wan 用從影片 token 到文字編碼器輸出的**交叉注意力**嵌入文字 [3]。CogVideoX 把文字與影片 token 串接成一個序列，並使用**專家式自適應 LayerNorm**（expert adaptive LayerNorm）對兩種模態分別調變，因為它們的特徵空間與數值尺度不同 [2]。HunyuanVideo 先在分開的串流中處理影片與文字 token，再串接起來做聯合處理（「雙串流到單串流」），並以多模態語言模型作為主要的文字編碼器 [16]。文字描述必須講出動作與事件，不能只講外觀（見「資料」一節）。

Make-A-Video 還把**影格率**輸入給模型，既作為資料增強，也作為動作速度的控制 [1]。引導的作法和影像相同（DLCV 12）。

影像生成影片

影像生成影片（「讓這張照片動起來」）必須保留輸入影格的內容，同時創造出動作。有三種已有文獻記載的設計：

- **Stable Video Diffusion** 把文字嵌入換成條件影格的影像嵌入（來自預訓練的影像–文字模型），並把該影格加過雜訊的副本沿通道軸串接到去雜訊器的輸入，對每個影格重複一次。作者發現固定的引導強度會造成偏離條件影格或過度飽和，因此改成讓引導強度沿影格線性增加 [4]。
- **DynamiCrafter** 用 query transformer 把影像投影到與文字對齊的上下文空間，讓影片模型能像注意提示一樣注意它，同時也把完整影像與初始雜訊串接，以保留視覺細節 [5]。
- **HunyuanVideo** 直接把參考影像的潛在表徵當作第一個潛在影格 [16]，當因果 VAE 把第一個影格單獨編碼時，這是很自然的選擇。

影像生成影片常見的失敗是**動作太少**：模型只要幾乎不改變輸入，就能在對輸入的保真度上得到高分。StreamingT2V 報告，當競爭的影像生成影片模型被反覆用來延伸片段時，容易出現「影片停滯」（video stagnation）[8]。

相機與運動控制

白話版。用「相機緩緩繞著雕像轉」這樣的提示來描述動作，是很弱的方式。控制方法直接把動作交給生成器：一條相機路徑（每個影格相機在哪裡），或是軌跡（物體或被拖曳的點該往哪裡去）。難處在於遵從動作，卻不改變內容。

- **把相機運動做成小型轉接器**。 Stable Video Diffusion 在基礎模型上，另外訓練了水平移動、變焦與固定相機的相機運動 LoRA 模組 [4]；AnimateDiff 的 **MotionLoRA** 則以很低的成本，把它的運動模組調整到新的運動樣式，例如不同的鏡頭類型 [21]。
- **把相機位姿當作稠密輸入**。 **CameraCtrl** 在影片擴散模型之上訓練一個隨插即用的相機模組，不更動基礎模型。它不直接輸入原始的相機矩陣，而是把每個影格的相機表示成逐像素的 **Plücker 嵌入**（Plücker embedding），也就是通過每個像素的視線射線的 6 個數字描述，把相機幾何與影像位置連結起來；作者發現，用相機路徑多樣、外觀與基礎模型資料相近的訓練影片，能在控制力與泛化之間取得最佳平衡 [12]。
- **相機與物體一起控制**。 **MotionCtrl** 獨立控制相機運動與物體運動，把相機位姿與物體軌跡當作不攜帶外觀資訊的條件 [11]。**DragNUWA** 結合文字、影像與軌跡，對開放領域的影片做細粒度控制 [13]。

相機路徑就是 3D 重建所估計的那種位姿序列（DLCV 11）。遵循它並不會讓生成的場景在幾何上一致，而且控制訊號的可靠程度，只能和它訓練時所用的位姿或軌跡一樣。

僅線上版：相機位姿與 *Plücker* 嵌入 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/notes/相機位姿與-plücker-嵌入/*

長影片

白話版。模型是用幾秒鐘的短片段訓練的，因為記憶體會隨長度增加。要做出較長的影片，要嘛一段一段不斷延伸片段（容易漂移），要嘛先粗略規劃整支影片再填入細節（需要長的訓練影片）。

無論哪一種，要在一分鐘之內維持劇情與身分一致，仍然很難。

有四種已有文獻記載的策略：

1. **一段一段自迴歸地延伸**。生成一段片段，再以前一段的最後幾個影格為條件，生成下一段。VDM 顯示，在取樣時單純替換已知的影格，得到的延續單獨看很好，卻和先前的影格對不上，因此它提出**重建引導**（reconstruction guidance），把模型對條件影格的重建所產生的引導項加進去 [6]。Video LDM 訓練預測模型，透過遮罩接收幾個上下文影格 [7]。**StreamingT2V** 加入短期記憶（對前一段特徵的注意力）以得到平順的銜接，以及長期記憶（第一段的特徵），讓模型不會忘記最初的場景 [8]。
2. **先粗後細**。 **NUWA-XL** 用一個全域擴散模型在整個時間範圍內生成關鍵影格，再用局部擴散模型遞迴地補上它們之間的影格，因此可以直接用長影片訓練，並且平行生成各段 [31]。
3. **改變雜訊，而不是模型**。 **FreeNoise** 不需微調就能延伸預訓練的短片段模型：它重新排程一串具有長程相關性的初始雜訊，並在視窗上做時間注意力，還加入一種動作注入方法，用於隨時間變化的提示 [32]。
4. **依故事生成可變長度的影片**。 **Phenaki** 用在時間上具因果性的 tokenizer 把影片壓縮成離散 token，因此能處理可變長度，並根據一連串隨時間變化的提示生成長影片 [33]。

核心的困難是**誤差累積**：每一段都是以生成的、略有錯誤的影格為條件，誤差會層層疊加，就像學得的世界模型的展開（rollout）一樣（DLCV 13）。自迴歸生成一節中的因果擴散方法，則是從訓練端來處理同一個問題。

僅線上版：重建引導：延伸影片 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/notes/重建引導延伸影片/*

時間一致性

白話版。一致的影片會讓每個細節都附著在它所屬的東西上：狗移動時，牠的斑點跟著移動；相機平移時，牆面的紋理跟著滑過。最簡單的測試是把動作還原，再檢查其他東西是否有變。陷阱在於，完全不動的影片會完美地通過這個測試。

不一致從哪裡來

生成器會創造條件沒有指定的細節。如果這些創造出來的細節在每個影格都重新畫一次，就會閃爍。如果它們固定在螢幕上而不是固定在物體上，就會在場景於其下移動時「黏住」。一致性意味著創造出來的細節必須**沿著動作傳遞**。架構靠時間層與所有影格的聯合生成來做到這一點；有幾種免訓練方法則在預訓練的影像模型中強制做到。Text2Video-Zero 讓每個影格的自注意力都去注意**第一個影格**（跨影格注意力），以保留前景物體的外觀與身分，並在潛在碼中加入動態以維持背景的一致 [10]。Make-A-Video 在最後逐影格的超解析度階段，對每個影格使用**相同的雜訊初始化**，以促成各影格之間創造出來的細節一致 [1]。Tune-A-Video 在影像模型中加入時空注意力機制，並在單一影片上微調它 [9]。

衡量一致性

常見的衡量有三種，每一種都只用生成影片本身就能算出的量來定義：

- **影格差異** $D = \frac{1}{(F-1)\,P}\sum_{t=2}^{F}\|x_t - x_{t-1}\|_1$，其中 x_t 是第 t 個影格，P 是每個影格的像素數值個數。它很便宜，但把真實動作與閃爍混在一起；VBench 只在靜態場景（所有變化都是閃爍）上用這種方式衡量時間閃爍 [34]。
- **扭曲誤差** $E = \frac{1}{(F-1)\,P}\sum_{t=2}^{F}\|x_t - \mathcal{W}(x_{t-1}, u_t)\|_1$，其中 u_t 是從影格 $t-1$ 到影格 t 的動作（光流），$\mathcal{W}$ 沿著它扭曲前一個影格。它在比較之前先扣除真實動作；光流、扭曲及其遮罩形式見 DLCV 09A.3。對生成的影片而言，光流必須**從影片本身估計**。
- **隨時間的特徵相似度**。 VBench 的主體一致性，把每個影格的特徵和第一個與前一個影格的特徵比較，$S = \frac{1}{F-1}\sum_{t=2}^{F}\frac{1}{2}(\langle d_1, d_t\rangle + \langle d_{t-1}, d_t\rangle)$，其中 d_t 是影格 t 由自監督影像模型得到的單位長度特徵向量，$\langle\cdot,\cdot\rangle$ 是內積（餘弦相似度）[34]。

這三種都會獎勵靜止的影片。因此 VBench 把它們與**動態程度**（dynamic degree）這個維度搭配，用光流網路衡量動作的多寡 [34]。

程式：閃爍、黏滯與凍結

這個玩具實驗建立一支「粗略影片」，讓影像每個影格平移 2 個像素（就像提示可能要求的那樣），再用四種方式加入創造出來的細緻細節：每個影格都用新的雜訊、固定在螢幕上的同一份雜訊、隨場景一起移動的同一份雜訊，以及重複第一個影格的凍結影片。它用相位相關估計每支影片自己的動作，並回報這三個量。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, transform
from skimage.registration import phase_cross_correlation

rng = np.random.default_rng(0)
T, v = 16, 2                                  # 16 frames; the prompt asks for a pan of 2 px
 ↪  per frame
scene = transform.resize(data.camera() / 255.0, (128, 128), anti_aliasing=True)
content = np.stack([np.roll(scene, v * t, axis=1) for t in range(T)])   # the "coarse" video

def detail():                                 # invented fine texture, as a generator would
 ↪  add
    n = ndi.gaussian_filter(rng.standard_normal(scene.shape), 0.7)
    return 0.04 * n / n.std()

d0 = detail()
videos = {
    "independent noise": content + np.stack([detail() for _ in range(T)]),  # new detail
     ↪  every frame
    "shared noise":      content + d0,                                      # same detail,
     ↪  fixed on screen
    "propagated noise":  content + np.stack([np.roll(d0, v * t, axis=1) for t in range(T)]),
     ↪  # detail moves with the scene
```

```
}
videos["frozen"] = np.repeat(videos["propagated noise"][:1], T, axis=0)      # a still image
 ↪  repeated

c = (slice(None), slice(8, -8), slice(8, -8))  # ignore borders (np.roll wraps around)
def measure(vid):
    shifts = [phase_cross_correlation(a, b, upsample_factor=10)[0][1] for a, b in
     ↪  zip(vid[1:], vid[:-1])]
    flicker = np.abs(np.diff(vid, axis=0))[c].mean()                         #
     ↪  frame-difference energy
    warped = np.stack([np.roll(f, int(round(s)), axis=1) for f, s in zip(vid[:-1], shifts)])
    warp_err = np.abs(vid[1:] - warped)[c].mean()                            # error after
     ↪  the video's own motion
    return flicker, warp_err, np.mean(np.abs(shifts))

for name, vid in videos.items():
    f, w, m = measure(vid)
    print(f"{name:18s} frame difference {f:.4f}   warping error {w:.4f}   estimated motion
     ↪  {m:.2f} px/frame")
```

輸出：

```
independent noise  frame difference 0.0808   warping error 0.0446   estimated motion 2.00
 ↪  px/frame
shared noise       frame difference 0.0528   warping error 0.0419   estimated motion 2.01
 ↪  px/frame
propagated noise   frame difference 0.0796   warping error 0.0000   estimated motion 2.00
 ↪  px/frame
frozen             frame difference 0.0000   warping error 0.0000   estimated motion 0.00
 ↪  px/frame
```

（所有數字都來自我們的玩具實驗。）把這四行放在一起讀：

- **只看影格差異會誤導**。它把固定在螢幕上的雜訊（0.0528）判定為比正確的、隨場景傳遞的細節（0.0796）更穩定，因為傳遞的細節會移動；它還給凍結的影片一個完美的零。
- **扭曲誤差能區分會動的幾種情況**。還原估計出的動作之後，只有隨場景傳遞的細節不留下任何殘差；每個影格都換新的雜訊，以及固定在螢幕上的雜訊，兩者的殘差仍然看得見。
- **還需要一個動作的衡量**。凍結的影片扭曲誤差同樣是零。只有估計出的動作（0.00，而不是要求的每影格 2 個像素）能揭露它，這正是 VBench 動態程度的作用。

圖 4 以經過動作補償的 x–t 切片顯示同樣的效果：正確、隨場景傳遞的細節會形成筆直的垂直線。

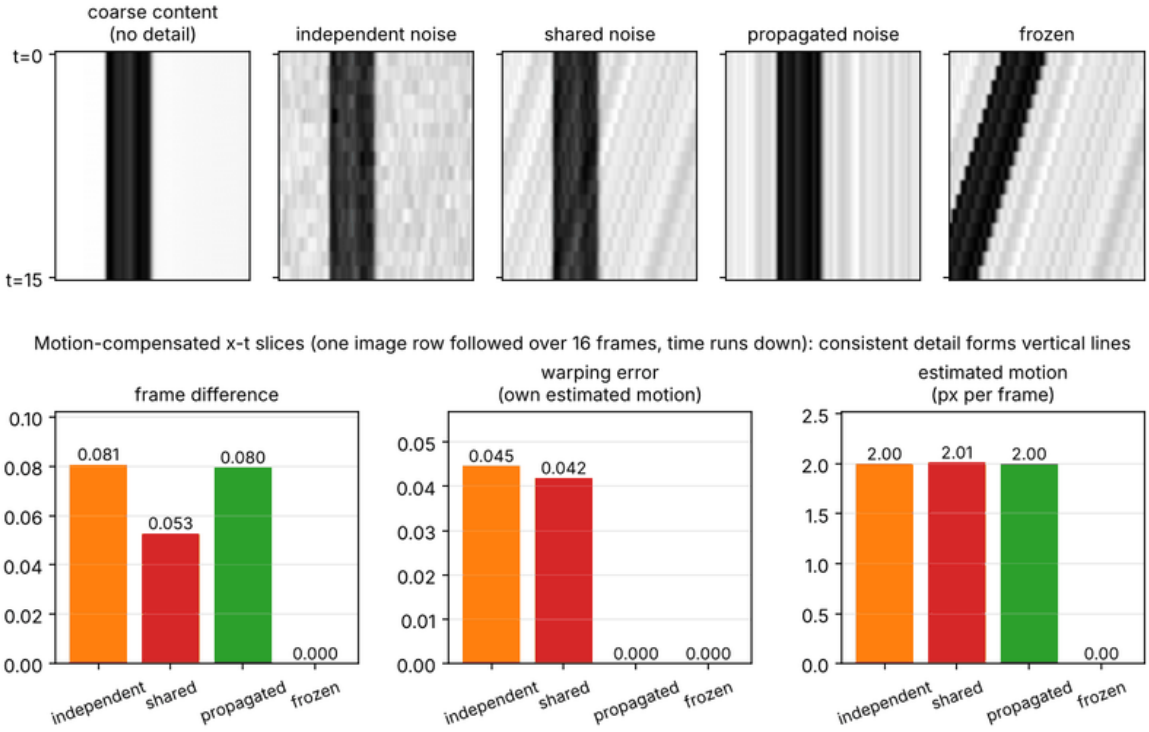


圖 4 — 一致性玩具實驗。還原要求的動作後，正確的細節變成垂直線；閃爍呈現為斑點，固定在螢幕上的細節呈現為傾斜的條紋，凍結的影片呈現為傾斜的帶狀。沒有任何單一衡量能把這四種情況全部正確排序。

自迴歸與擴散影片生成

白話版。產生一段片段有兩種方式。**擴散**（或流）模型把固定長度片段的所有影格放在一起，從雜訊精修成影片，所以每個影格既看得到過去，也看得到未來。**自迴歸**模型則一段一段地寫出影片，每一段都以先前的內容為條件，就像語言模型寫文字；它可以無限寫下去，也能串流，但早期的錯誤會留下來。最近的混合式做法試圖兩者兼得。

token 模型：遮罩式與自迴歸

有了離散的影片 tokenizer，影片生成就變成序列建模（ML 09）。**MAGVIT** 使用遮罩 token 建模，平行預測許多 token，並以單一模型支援許多生成任務（例如影格預測、內插與修補）[20]。**Phenaki** 用以文字為條件的雙向遮罩 transformer 生成它的因果影片 token [33]。**VideoPoet** 是處理文字、影像、影片與音訊 token 的 decoder-only transformer；MAGVIT-v2 把每秒 8 個影格、128 × 128 的 17 個影格轉成 5 × 16 × 16 的網格，也就是 1,280 個 token，詞彙表大小為 2^{18}，而模型像語言模型一樣訓練，先在多種生成任務的混合上預訓練，再針對特定任務調適 [19]。token 模型重用語言模型的基礎設施（KV 快取，見 ML 09），並且能自然地延伸片段，但每個生成的 token 都是一個承諾，誤差會累積（ML 09）。

擴散：整段影片一次生成

標準的影片擴散 transformer 以**雙向**注意力，一起對所有潛在影格去雜訊。CausVid 直白地說出其後果：生成單一影格就必須處理整個序列，包括未來，這讓這類模型很難用於互動場合 [35]。它們的長處就是讓擴散在影像領域稱霸的那些（訓練穩定、品質高、可引導），再加上片段內的整體一致性。它們的限制是最大長度固定，且延遲高。

混合式：因果、可串流的擴散

- **Diffusion Forcing** 訓練擴散模型對一串 token 去雜訊，每個 token 有**獨立的雜訊程度**，並把它當作因果的下一個 token 模型，在不把過去完全擴散完的情況下生成未來的影格；這讓它可以把影片展開到超過訓練視野的長度，而它的基線在那裡會發散 [36]。
- **CausVid** 把預訓練的雙向影片擴散 transformer 調整成因果、自迴歸的版本，即時逐影格生成，並以雙向教師監督因果學生，把它的 50 步取樣器蒸餾成 4 步，同時使用 KV 快取做串流生成；作者報告這能減輕誤差累積，並讓模型儘管只用短片段訓練，仍能產生長影片 [35]。
- **Self Forcing** 直接處理**曝光偏差**：訓練時，每個影格都以模型自己先前的輸出為條件來生成（以 KV 快取做自迴歸展開），而不是以真實影格為條件，並對整段生成的影片施加損失；它報告在單張 GPU 上達到延遲不到一秒的即時串流生成 [37]。

面向	自迴歸或因果	雙向擴散
長度	無上限，一段一段生成	由訓練片段長度決定，除非另外延伸
延遲	最前面的影格很早就可取得，可串流	必須先對整段片段去雜訊
未來資訊	沒有（因果）	每個影格都看得到過去與未來
主要風險	誤差累積、曝光偏差	長片段的記憶體與計算量
典型用途	互動式生成、世界模型、直播影片	離線的文字生成影片與影像生成影片

同樣的因果與雙向的區別，對世界模型也很重要，因為動作是一步一步到來的（DLCV 13）。也請記得 ML 08 講過的：因果**自編碼器**並不會讓生成器變成自迴歸；那是由生成器的注意力與取樣方式決定的。

評估影片生成模型

白話版。你需要四方面的證據：影片整體看起來像真的影片（FVD 這類分布指標）、它們符合提示、它們的動作與一致性良好（VBench 這類逐維度的套件），以及人們比較喜歡它們（人類研究）。結果發現，標準的分布指標主要看的是單一影格，所以可能恰好漏掉本篇關心的時間問題。

Fréchet Video Distance

FVD 把 FID（ML 08）延伸到影片 [38]。每支影片（真實的或生成的）都通過一個預訓練的影片網路，也就是在 Kinetics-400 [40] 上訓練的 I3D 動作辨識模型 [39]，並取它的輸出（logits）作為特徵向量。對每一組特徵擬合一個 Gaussian，然後

$$\mathrm{FVD} = \|\mu_r - \mu_g\|_2^2 + \mathrm{Tr}\big(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\big),$$

其中 μ_r, Σ_r 是真實影片特徵的平均與共變異數，μ_g, Σ_g 是生成影片的，Tr 是跡（trace），$(\cdot)^{1/2}$ 是矩陣平方根。越低越好。原始論文引入 FVD 時附了一項人類研究，顯示它與人們對生成影片的判斷有良好的相關性 [38]。類別條件與無條件的基準常在 UCF-101（一個包含 101 個人類動作類別的資料集）上報告 FVD [41]、[1]、[25]。

它的主要限制：內容偏差。 Ge 等人發現，在逐影格品質相同的情況下，FVD 在大幅度的時間破壞下只上升一點點，而且精心挑選的一組**沒有動作**的生成影片子集，可以讓 FVD 大幅下降 [42]。他們把偏差追溯到特徵：在 Kinetics 上訓練的動作分類器學到的是物體與場景的外觀，因為許多動作只看一個影格就能辨認；來自自監督影片模型 VideoMAE v2 [43] 的特徵，偏差就小得多 [42]。FID 的各種陷阱同樣適用（DLCV 12），此外還有影片特有的陷阱：片段長度、影格率，以及縮放到低解析度的正方形 [42]。

僅線上版：*Ge 等人：On the Content Bias in Fréchet Video Distance（CVPR 2024）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/papers/ge-等 人 on-the-content-bias-in-fréchet-video-distancecvpr-2024/*

程式：看不見時間的 Fréchet 距離

這個玩具實驗製作「真實」片段（一個沿直線移動的光斑），以及兩組被破壞的片段：一組是**模糊的影格**（影格較差，動作正確），另一組是**打亂的影格**（影格完美，動作被破壞）。它用兩個手工製作的特徵萃取器計算 Fréchet 距離：一個把每個影格的外觀統計量沿時間平均，就像主要只看內容的萃取器；另一個再加上兩個時間統計量。

```
import numpy as np
from scipy import linalg, ndimage as ndi

rng = np.random.default_rng(0)
T, S = 16, 32
yy, xx = np.mgrid[:S, :S]

def real_videos(n):
    """Toy 'real' clips: a bright blob moving in a straight line, plus mild sensor noise."""
    vids = np.empty((n, T, S, S))
    for i in range(n):
        p, vel, r = rng.uniform(10, 22, 2), rng.uniform(-0.7, 0.7, 2), rng.uniform(2.5, 4)
        for t in range(T):
            cy, cx = p + vel * t
            vids[i, t] = np.exp(-((yy - cy) ** 2 + (xx - cx) ** 2) / (2 * r**2))
    return vids + 0.02 * rng.standard_normal(vids.shape)

def frame_stats(f):                       # appearance of one frame
    gy, gx = np.gradient(f)
    return [f.mean(), f.std(), f.max(), np.abs(gx).mean() + np.abs(gy).mean()]

def appearance_features(v):               # frame statistics averaged over time: blind to
↪   frame order
    return np.mean([frame_stats(f) for f in v], axis=0)

def motion_features(v):                   # add how the content changes from frame to frame
    d = np.abs(np.diff(v, axis=0)).mean(axis=(1, 2))
```

```
    return np.concatenate([appearance_features(v), [d.mean(), d.std()]])

def frechet(a, b):                        # Fréchet distance between Gaussian fits of two
 ↪  feature sets
    mu1, mu2 = a.mean(0), b.mean(0)
    s1, s2 = np.cov(a, rowvar=False), np.cov(b, rowvar=False)
    covmean = linalg.sqrtm(s1 @ s2).real
    return float(((mu1 - mu2) ** 2).sum() + np.trace(s1 + s2 - 2 * covmean))

real, other_real = real_videos(400), real_videos(400)
blurred = ndi.gaussian_filter(other_real, sigma=(0, 0, 0.7, 0.7))          # worse frames,
 ↪  same motion
shuffled = np.stack([v[rng.permutation(T)] for v in other_real])          # perfect frames,
 ↪  broken motion

for name, feats in [("appearance features", appearance_features), ("+ motion features",
 ↪  motion_features)]:
    f = lambda vids: np.stack([feats(v) for v in vids])
    ref = f(real)
    scale = ref.std(0)                    # put all features on a comparable scale
    fd = {k: frechet(ref / scale, f(v) / scale) for k, v in
          [("second real set", other_real), ("blurred frames", blurred), ("shuffled frames",
           ↪  shuffled)]}
    print(f"{name:20s} " + "   ".join(f"{k} {d:6.2f}" for k, d in fd.items()))
```

輸出：

```
appearance features  second real set   0.01   blurred frames 115.10   shuffled frames   0.01
+ motion features    second real set   0.07   blurred frames 141.87   shuffled frames
 ↪  12397.17
```

（所有數字都來自我們的玩具實驗，使用的是玩具特徵，不是 I3D。）只用外觀特徵時，動作被破壞的打亂組，和第二組真實影片一樣接近真實組（0.01），而輕微的模糊卻被重重處罰。加入兩個時間統計量之後，打亂組變成遠遠最差的一組。真正的 FVD 沒有這麼極端，因為 I3D 確實看得到一些動作，但這正是 Ge 等人所測量的內容偏差的形狀 [42]：**這個指標有多「時間性」，取決於它的特徵**。圖 5 顯示這些片段與距離。

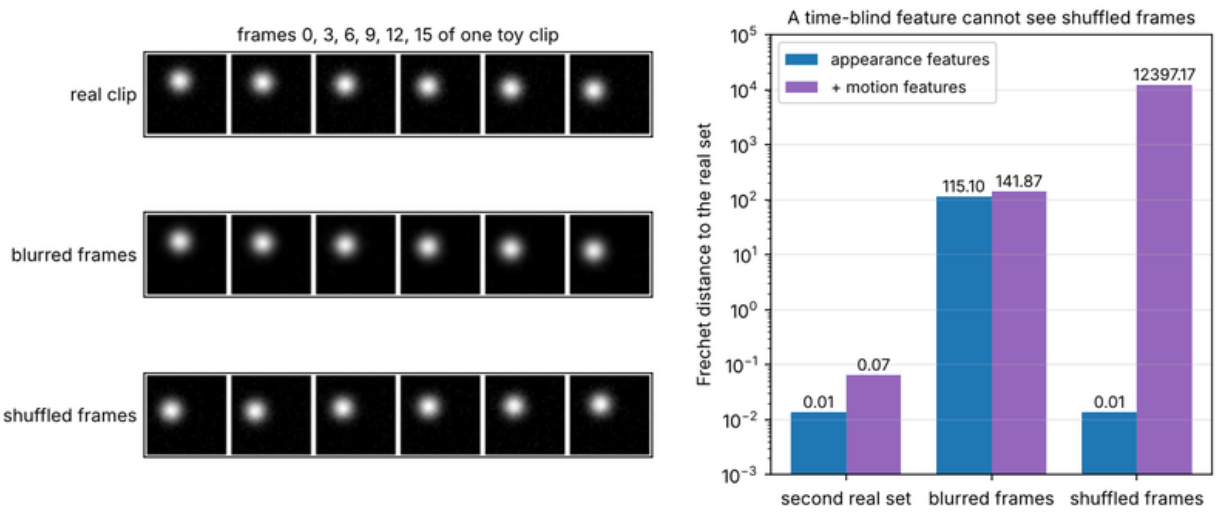


圖 *5* — 用看不見時間的特徵算出的 *Fréchet* 距離，分不出打亂的片段和真實的片段。玩具資料與玩具特徵。

基準套件與人類評估

白話版。現代的基準不給單一一個數字，而是問許多狹窄的問題，每個問題有自己的提示與自動檢查，並確認這些檢查和人的判斷一致。

- **VBench** 把「影片生成品質」拆成 16 個維度，每個維度都有量身訂做的提示與評估方法：時間品質（主體一致性、背景一致性、時間閃爍、動作平滑度、動態程度）、逐影格品質（美感與成像品質）、語意（物體類別、多個物體、人類動作、顏色、空間關係、場景）、風格（外觀風格與時間風格），以及整體一致性；它為每個維度提供人類偏好標註，用來驗證自動分數 [34]。**VBench++** 把它延伸到影像生成影片（含自適應長寬比的影像套件）以及可信賴度 [44]。
- **EvalCrafter** 從對真實使用者資料的分析中建立 700 個提示，用 17 個客觀指標評估視覺品質、內容品質、動作品質與文字–影片對齊，並擬合係數，使合併後的分數與使用者的意見一致 [45]。
- **人類評估**仍是基準。Stable Video Diffusion 用人類偏好研究來比較它的訓練選擇 [4]；HunyuanVideo 報告專業人員所做的評估 [16]；Movie Gen 則由專家標註者沿文字對齊、視覺品質，以及真實感與美感進行成對 A/B 測試，使用一組公開的 1,003 個提示，並報告 FVD 與 Inception Score 和它對影片品質的人類評估並不相關 [17]。Open-Sora 2.0 同時報告人類評估與 VBench 分數 [46]。

物理合理性需要自己的測試；DLCV 13 描述了一些基準，發現物理理解與視覺逼真度無關。

報告清單。說明提示以及它們如何挑選、片段長度、影格率與解析度、樣本數與隨機種子、取樣器、步數與引導（包括跨影格的任何排程）、每個分布指標的特徵萃取器與前處理、所用的維度套件及其版本，以及人類研究的流程（成對或絕對、標註者人數、多數決規則）。報告時間衡量要**搭配**動作衡量與逐影格品質衡量，並且展示未經挑選的片段。

資料

白話版。影片模型從訓練片段中學習外觀、動作與它和語言的連結。有良好文字描述的影片，比有文字描述的影像稀少得多，所以系統會結合影像與影片資料，並花費大量心力篩選片段與重新標註文字描述。

- **影像加影片**。 VDM [6]、Phenaki [33] 與許多後來的系統都用影像與影片聯合訓練；Make-A-Video 只從影像–文字配對學習文字與視覺的連結 [1]；因果「$1 + T$」自編碼器讓影像成為一個影格的影片 [16]、[3]。
- **網路影片集**。 WebVid 是帶有弱文字描述的網路影片，隨 Frozen in Time 檢索模型一起提出（論文中是 WebVid-2M；依其 arXiv 紀錄，後來釋出 10M 版本）[47]，Make-A-Video 就是用 WebVid-10M 訓練的 [1]。**Panda-70M** 整理了 380 萬個高解析度影片，把它們切成語意一致的片段，並用數個跨模態教師模型，加上一個挑選最佳文字描述的檢索模型為它們加上文字描述，得到 7,000 萬個附文字描述的片段 [48]。
- **資料篩選很重要**。 Stable Video Diffusion 的 Large Video Dataset 一開始有 5.8 億個已標註的片段，內容長達 212 年，接著用光流分數移除近乎靜止的片段，用 OCR 移除含大量文字的片段，並移除美感分數低的片段；用篩選後子集訓練的模型，比用未篩選資料訓練的模型更受人們偏好 [4]。CogVideoX 則表示，它自己的影片文字描述模型是其資料管線的關鍵部分 [2]。
- **評估資料集**。 UCF-101 [41] 與 Kinetics [40] 是人類動作資料集，提供 FVD 的基準，以及 FVD 的特徵網路。

僅線上版：*Blattmann 等人：Stable Video Diffusion: Scaling Latent Video Diffusion Models to Large Datasets（2023）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/video-generation/papers/blattmann-等人 stable-video-diffusion-scaling-latent-video-diffusion-models-to-large-datasets2023/*

網路影片和網路影像一樣，會引發著作權、同意與內容安全的問題；見 DLCV 12，資料。

風險與負責任的使用

白話版。逼真的生成影片可以讓人「說出」他沒說過的話，或呈現從未發生的事件。防禦方式和影像相同，有三種（偵測偽造、標示生成內容、從源頭限制濫用），但影片更難：它每次上傳都會被重新編碼，而且幾秒鐘令人信服的畫面就足以誤導人。

- **深偽與偵測**。 FaceForensics++ 是偵測臉部操弄的基準（四種操弄方法、隨機的壓縮程度與尺寸、超過 180 萬張被操弄的影像，並有不公開的測試集）；其作者發現，領域專屬的知識即使在強壓縮下也能改善偵測，而且在他們的基準上，偵測器勝過人類觀察者 [49]。在某一種生成方法上訓練的偵測器，可能對較新的方法失效，所以偵測是一場軍備競賽。
- **影片浮水印**。 **Video Seal** 聯合訓練浮水印嵌入器與萃取器，並在兩者之間施加影片編解碼器等轉換，同時沿時間傳遞浮水印，因此不必為每個高解析度影格都加上浮水印 [50]。簽章的出處資訊（provenance manifest）、不可見浮水印及其失效模式，在 DLCV 12，深偽、出處與浮水印 及其補充資料硬綁定、軟綁定與浮水印中說明；同樣的教訓也適用於影片，只是影片中重新編碼與裁切是家常便飯。
- **偏見、記憶與同意**和影像一樣，源自資料（DLCV 12，風險）；人物在網路影片中出現的頻率遠高於許多影像集，這讓同意問題的賭注更高。

- **虛假的權威**。看起來物理上正確的生成影片，很容易被當成證據。請把生成的片段視為插圖，而不是紀錄；為什麼逼真不等於物理正確，見 DLCV 13。

失效模式

白話版。今天的影片生成器擅長常見場景的短片、單一鏡頭。它們在隨時間保持不變、物理、細緻的動作，以及任何較長的內容上，仍會失敗。

- **閃爍與紋理黏滯**。細節在影格之間變動，或在場景移動時維持固定（圖 4）。
- **身分漂移與變形**。臉、文字與花紋緩慢改變；兩個物體融合，或一個物體分裂。
- **物體恆存性**。物體被遮住時消失，或回來時變了樣。
- **動作太少**。近乎靜止的片段在一致性上得高分；動態程度的衡量就是因此而存在的 [34]，而影像生成影片的延伸可能停滯 [8]。
- **不合理的物理與動力學**。動作看起來流暢，物理上卻是錯的（見 DLCV 13）。
- **長影片的漂移**。一段一段延伸片段時的誤差累積 [35]、[37]；對最初場景的遺忘 [8]。
- **控制薄弱**。相機路徑只被粗略地遵循，軌跡改變了物體的形狀；MotionCtrl 以不含外觀資訊的動作條件為設計目標，針對的正是這一點 [11]。
- **繼承自影像的失敗**。計數、屬性綁定、文字渲染與手部，會像在影像中一樣失敗（DLCV 12，失效模式），而且現在還多了隨時間變化的可能。

如何測試。針對每一類失敗建立小型的提示集（快速動作、遮擋、人轉身、書寫的文字、指定的相機動作），每個提示生成數個隨機種子，同時衡量一致性**與**動作，檢視經過動作補償的 x–t 切片以找出閃爍與黏滯，並實際觀看片段；影格拼貼會藏住大部分的時間性失敗。

現代觀點

值得閱讀的綜述。

- **Melnik 等人 [51]**（TMLR 2024）綜述影片擴散模型：數學公式、架構選擇、時間一致性的方法、依輸入模態（文字、影像、影片、音訊）分類的應用，以及含資料集與指標的訓練與評估實務，還有長影片與計算成本等開放挑戰。
- **Xing 等人 [52]**（2023）從三個領域回顧影片擴散模型：影片生成、影片編輯，以及其他影片理解任務，每個領域下又分更細的類別，並討論挑戰與未來趨勢。
- **Liu 等人 [53]**（2024）依公開的技術報告與逆向工程，回顧 OpenAI 在 2024 年 2 月發表的 Sora：背景、相關技術、應用，以及安全與無偏見的生成等限制。由於它依賴逆向工程，請把其中的技術主張視為有根據的假設來讀。

歷史模型與可用服務要分開。本篇論文用來解釋架構里程碑，不是現行採購清單。依 2026 年 11 月 5 日查核的 OpenAI 公告，Sora 網頁與 App 已於 2026 年 4 月 26 日停止服務，API 則於 2026 年 9 月 24 日停止服務 [54]。獨立的 Open-Sora 專案是不同專案。現行部署選項及鏡頭驗收重點，可參考影片生成模型選擇指南。

2023–2026 年的最新進展。有五個趨勢特別突出。(1) 搭配**完整注意力的潛在 transformer 與因果 3D VAE**，成為大型開放模型的主流配方（CogVideoX [2]、HunyuanVideo [16]、Wan [3]），其中數個採用流匹配目標 [16]、[17]、[3]。(2) **開放模型報告在它們自己的人類評估與 VBench 分**

數上，縮小了與閉源系統的大部分差距，而訓練成本也成為研究主題：Open-Sora 2.0 報告以約 $200k 訓練出商業等級的模型 [46]。(3) **更高的壓縮**減少 token，使完整注意力維持在負擔得起的範圍，由解碼器接手更多細節 [18]。(4) **因果、少步、串流的生成**帶來互動與即時的使用 [35]、[37]、[36]。(5) **評估超越了 FVD**，走向分維度套件 [34]、[44]、[45] 與大型人類研究 [17]，並認識到需要對時間敏感的特徵 [42]。同步的音訊生成也出現了 [17]、[19]。

尚未解決的問題。在數分鐘而非數秒的尺度上維持一致性與身分；物理正確的動力學與互動（DLCV 13）；對相機、物體與時間點做精確、解耦的控制；能衡量動作與一致性、不會獎勵靜止，並且與人的判斷一致的指標；訓練與取樣長的高解析度片段的成本；對事件與動作的文字描述品質；以及經得起重新編碼的出處資訊。

重點整理

- 清晰單格不足以代表好影片，運動與身分還須跨時間一致。
- 壓縮降低生成成本，也限制可恢復的細節與時間資訊。
- 時間層、分解式注意力與完整時空注意力提供不同資訊傳遞方式及成本。
- 註明條件是文字、首格、相機姿態或運動軌跡。
- 分塊延伸與因果生成可增加輸出長度，但須處理累積誤差。
- 同時評估外觀、運動、時間一致性與偏好，並報告資料整理、時長及完整成本。

練習

1. **token 預算**。某個模型使用因果 $4 \times 8 \times 8$ VAE 與 2×2 patch。一段 121 個影格、720×1280 的片段會產生多少個 token？一層完整注意力有多少個查詢–鍵配對，分解式注意力把它減少了多少倍？下面哪一個單一改變能更有效地降低完整注意力的成本：把時間壓縮加倍，還是把 patch 大小加倍？

   提示

   $T' = 1+120/4 = 31$，每個潛在影格有 $(720/16)\cdot(1280/16) = 45\cdot 80 = 3600$ 個 token，所以 $L = 111{,}600$ 個 token，約有 1.25×10^{10} 個配對。分解式的比值為 $TS/(T+S) = 31 \cdot 3600/3631 \approx 30.7$。把 f_t 加倍得到 $T' = 16$（配對數縮小約 3.8 倍）；把 p 加倍則讓每個影格的 token 數除以 4（配對數縮小 16 倍），代價是空間 token 變得更粗。可以用程式中的函式檢查。

2. **打破一致性衡量**。在一致性玩具實驗中，加入第五支影片：創造出來的細節每個影格平移 2 個像素，但粗略內容**不**移動（「靜止場景上移動的紋理」）。三種衡量各自回報什麼？你會相信哪一種能偵測出這個失敗？

   提示

   相位相關現在看到兩個互相競爭的動作：紋理在動，內容不動。依兩者相對強度而定，估計出的位移是 0 或 2，而沒被補償的那一層會留下扭曲誤差。對有多個移動部分的真實影片，單一的全域動作估計並不夠；實務上會使用帶有遮擋遮罩的稠密光流（DLCV 09A.3）。

3. **你自己的特徵中的內容偏差**。修改 Fréchet 玩具實驗，讓「生成」組的影格正確，但光斑移動速度是真實組的兩倍。哪一個特徵萃取器能偵測出來？接著設計第三個特徵，也能偵測以正確速度移動、但在中途反向的光斑。

   提示

   速度改變會改變平均影格差異，所以只有動作特徵能偵測到。反向則會讓平均絕對差異保持相近；需要一個比較動作方向隨時間變化的特徵（例如質心軌跡前後兩半的相關性，或淨位移除以路徑長度）。真實的指標也面臨同樣的問題：這些特徵編碼了哪些時間性質？

4. **選擇架構**。你要為商品照片建立影像生成影片模型（相機繞著物體緩慢轉動 3 秒），手上只有小型影片資料集，以及一個強大的預訓練影像模型。請在「用分解式時間層膨脹影像模型」與「從頭訓練完整注意力的影片 transformer」之間做選擇，並說明你會如何加入相機控制。

   提示

   影片少、影像模型強時，以零初始化或恆等初始化的時間層膨脹模型，能重用外觀知識，所需資料也較少；相機運動緩慢，分解式注意力在大幅度動作上的弱點就不那麼要緊。以照片為條件（串接的潛在表徵加上影像嵌入，如 Stable Video Diffusion），並加入使用逐像素 Plücker 嵌入的相機模組，用相機路徑多樣且已知的片段訓練。評估商品在各影格之間的身分是否保持、相機路徑的準確度，以及動態程度的衡量。

5. **評估計畫**。某團隊宣稱他們的新模型有「更好的時間一致性」，理由是它的 VBench 主體一致性分數較高。請列出三種其他解釋，以及你會要求的額外證據。

   提示

   模型可能動得比較少（檢查動態程度與動作平滑度）、可能生成較簡單或較短的片段（檢查提示、片段長度與影格率），或逐影格品質不同而影響特徵萃取器（檢查逐影格品質）。要求提供同一個 VBench 版本的完整維度表、每個提示的未經挑選片段、在動作相當的條件下對一致性做的人類成對研究，以及用對時間敏感的特徵萃取器算出的 FVD。

參考文獻

1. U. Singer, A. Polyak, T. Hayes et al., “Make-A-Video: Text-to-Video Generation without Text-Video Data,” arXiv:2209.14792, 2022. arXiv
2. Z. Yang, J. Teng, W. Zheng et al., “CogVideoX: Text-to-Video Diffusion Models with An Expert Transformer,” *ICLR*, 2025. arXiv
3. Team Wan, A. Wang, B. Ai, B. Wen et al., “Wan: Open and Advanced Large-Scale Video Generative Models,” arXiv:2503.20314, 2025. arXiv
4. A. Blattmann, T. Dockhorn, S. Kulal et al., “Stable Video Diffusion: Scaling Latent Video Diffusion Models to Large Datasets,” arXiv:2311.15127, 2023. arXiv
5. J. Xing, M. Xia, Y. Zhang et al., “DynamiCrafter: Animating Open-Domain Images with Video Diffusion Priors,” *ECCV*, LNCS, pp. 399–417, 2024. doi ·arXiv

6. J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi and D. J. Fleet, “Video Diffusion Models,” *NeurIPS*, 2022. proceedings ·arXiv
7. A. Blattmann, R. Rombach, H. Ling et al., “Align your Latents: High-Resolution Video Synthesis with Latent Diffusion Models,” *CVPR*, 2023. CVF
8. R. Henschel, L. Khachatryan, H. Poghosyan et al., “StreamingT2V: Consistent, Dynamic, and Extendable Long Video Generation from Text,” *CVPR*, 2025. CVF
9. J. Z. Wu, Y. Ge, X. Wang et al., “Tune-A-Video: One-Shot Tuning of Image Diffusion Models for Text-to-Video Generation,” *ICCV*, 2023. CVF
10. L. Khachatryan, A. Movsisyan, V. Tadevosyan et al., “Text2Video-Zero: Text-to-Image Diffusion Models are Zero-Shot Video Generators,” *ICCV*, 2023. CVF
11. Z. Wang, Z. Yuan, X. Wang et al., “MotionCtrl: A Unified and Flexible Motion Controller for Video Generation,” *SIGGRAPH 2024 Conference Papers*, 2024. doi ·arXiv
12. H. He, Y. Xu, Y. Guo et al., “CameraCtrl: Enabling Camera Control for Text-to-Video Generation,” *ICLR*, 2025. arXiv
13. S. Yin, C. Wu, J. Liang et al., “DragNUWA: Fine-grained Control in Video Generation by Integrating Text, Image, and Trajectory,” arXiv:2308.08089, 2023. arXiv
14. R. Rombach, A. Blattmann, D. Lorenz, P. Esser and B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” *CVPR*, 2022. CVF
15. L. Yu, J. Lezama, N. B. Gundavarapu et al., “Language Model Beats Diffusion — Tokenizer is Key to Visual Generation,” *ICLR*, 2024. arXiv
16. W. Kong, Q. Tian, Z. Zhang et al., “HunyuanVideo: A Systematic Framework For Large Video Generative Models,” arXiv:2412.03603, 2024. arXiv
17. A. Polyak, A. Zohar, A. Brown et al., “Movie Gen: A Cast of Media Foundation Models,” arXiv:2410.13720, 2024. arXiv
18. Y. HaCohen, N. Chiprut, B. Brazowski et al., “LTX-Video: Realtime Video Latent Diffusion,” arXiv:2501.00103, 2024. arXiv
19. D. Kondratyuk, L. Yu, X. Gu et al., “VideoPoet: A Large Language Model for Zero-Shot Video Generation,” *ICML*, PMLR 235, 2024. PMLR ·arXiv
20. L. Yu, Y. Cheng, K. Sohn et al., “MAGVIT: Masked Generative Video Transformer,” *CVPR*, 2023. CVF
21. Y. Guo, C. Yang, A. Rao et al., “AnimateDiff: Animate Your Personalized Text-to-Image Diffusion Models without Specific Tuning,” *ICLR*, 2024. arXiv
22. Z. Zheng, X. Peng, T. Yang et al., “Open-Sora: Democratizing Efficient Video Production for All,” arXiv:2412.20404, 2024. arXiv
23. G. Bertasius, H. Wang and L. Torresani, “Is Space-Time Attention All You Need for Video Understanding?,” *ICML*, PMLR 139, 2021. PMLR
24. A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić and C. Schmid, “ViViT: A Video Vision Transformer,” *ICCV*, 2021. CVF
25. X. Ma, Y. Wang, X. Chen et al., “Latte: Latent Diffusion Transformer for Video Generation,” *Transactions on Machine Learning Research*, 2025. arXiv
26. B. Lin, Y. Ge, X. Cheng et al., “Open-Sora Plan: Open-Source Large Video Generation Model,” arXiv:2412.00131, 2024. arXiv
27. J. Ho, W. Chan, C. Saharia et al., “Imagen Video: High Definition Video Generation with Diffusion Models,” arXiv:2210.02303, 2022. arXiv

28. O. Bar-Tal, H. Chefer, O. Tov et al., “Lumiere: A Space-Time Diffusion Model for Video Generation,” *SIGGRAPH Asia 2024 Conference Papers*, 2024. doi ·arXiv
29. Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel and M. Le, “Flow Matching for Generative Modeling,” arXiv:2210.02747, 2022. arXiv
30. W. Peebles and S. Xie, “Scalable Diffusion Models with Transformers,” *ICCV*, 2023. CVF
31. S. Yin, C. Wu, H. Yang et al., “NUWA-XL: Diffusion over Diffusion for eXtremely Long Video Generation,” *ACL*, pp. 1309–1320, 2023. doi ·arXiv
32. H. Qiu, M. Xia, Y. Zhang et al., “FreeNoise: Tuning-Free Longer Video Diffusion via Noise Rescheduling,” *ICLR*, 2024. arXiv
33. R. Villegas, M. Babaeizadeh, P.-J. Kindermans et al., “Phenaki: Variable Length Video Generation From Open Domain Textual Description,” arXiv:2210.02399, 2022. arXiv
34. Z. Huang, Y. He, J. Yu et al., “VBench: Comprehensive Benchmark Suite for Video Generative Models,” *CVPR*, 2024. CVF
35. T. Yin, Q. Zhang, R. Zhang et al., “From Slow Bidirectional to Fast Autoregressive Video Diffusion Models,” *CVPR*, 2025. CVF
36. B. Chen, D. M. Monso, Y. Du, M. Simchowitz, R. Tedrake and V. Sitzmann, “Diffusion Forcing: Next-token Prediction Meets Full-Sequence Diffusion,” *NeurIPS*, 2024. proceedings ·arXiv
37. X. Huang, Z. Li, G. He, M. Zhou and E. Shechtman, “Self Forcing: Bridging the Train-Test Gap in Autoregressive Video Diffusion,” *NeurIPS*, 2025. arXiv
38. T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski and S. Gelly, “Towards Accurate Generative Models of Video: A New Metric & Challenges,” arXiv:1812.01717, 2018. arXiv
39. J. Carreira and A. Zisserman, “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,” *CVPR*, 2017. CVF
40. W. Kay, J. Carreira, K. Simonyan et al., “The Kinetics Human Action Video Dataset,” arXiv: 1705.06950, 2017. arXiv
41. K. Soomro, A. R. Zamir and M. Shah, “UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild,” technical report CRCV-TR-12-01, arXiv:1212.0402, 2012. arXiv
42. S. Ge, A. Mahapatra, G. Parmar, J.-Y. Zhu and J.-B. Huang, “On the Content Bias in Fréchet Video Distance,” *CVPR*, 2024. CVF
43. L. Wang, B. Huang, Z. Zhao et al., “VideoMAE V2: Scaling Video Masked Autoencoders With Dual Masking,” *CVPR*, 2023. CVF
44. Z. Huang, F. Zhang, X. Xu et al., “VBench++: Comprehensive and Versatile Benchmark Suite for Video Generative Models,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 48, no. 3, pp. 3268–3285, 2026. doi ·arXiv
45. Y. Liu, X. Cun, X. Liu et al., “EvalCrafter: Benchmarking and Evaluating Large Video Generation Models,” *CVPR*, 2024. CVF
46. Z. Zheng, X. Peng, Y. Lou et al., “Open-Sora 2.0: Training a Commercial-Level Video Generation Model in $200k,” arXiv:2503.09642, 2025. arXiv
47. M. Bain, A. Nagrani, G. Varol and A. Zisserman, “Frozen in Time: A Joint Video and Image Encoder for End-to-End Retrieval,” *ICCV*, 2021. CVF ·arXiv
48. T.-S. Chen, A. Siarohin, W. Menapace et al., “Panda-70M: Captioning 70M Videos with Multiple Cross-Modality Teachers,” *CVPR*, 2024. CVF
49. A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies and M. Nießner, “FaceForensics++:

Learning to Detect Manipulated Facial Images,” *ICCV*, 2019. doi ·arXiv
50. P. Fernandez, H. Elsahar, I. Z. Yalniz and A. Mourachko, “Video Seal: Open and Efficient Video Watermarking,” arXiv:2412.09492, 2024. arXiv
51. A. Melnik, M. Ljubljanac, C. Lu, Q. Yan, W. Ren and H. Ritter, “Video Diffusion Models: A Survey,” *Transactions on Machine Learning Research*, 2024. arXiv
52. Z. Xing, Q. Feng, H. Chen et al., “A Survey on Video Diffusion Models,” arXiv:2310.10647, 2023. arXiv
53. Y. Liu, K. Zhang, Y. Li et al., “Sora: A Review on Background, Technology, Limitations, and Opportunities of Large Vision Models,” arXiv:2402.17177, 2024. arXiv
54. OpenAI, “What to know about the Sora discontinuation,” official service notice, 2026, accessed 5 November 2026. Notice.

DLCV 12B・單步生成

從可重現的數值實驗理解單步預測，分辨軌跡蒸餾、一致性、reflow 與分布匹配，並完整衡量速度、品質及多樣性。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/

你將學到

- 解釋為何一次局部去雜訊方向不一定能到達遠方端點。
- 比較端點迴歸、一致性、重流與分布匹配。
- 區分匹配教師軌跡與匹配輸出分布。
- 理解以步長為條件的捷徑與平均速度目標。
- 報告網路評估次數、完整延遲、對齊與多樣性。

先看全貌

用最簡單的話說

畫畫老師通常會給你許多小指令：往這邊移動、再看一下、修正曲線、補上細節。如果只保留第一個指令，再把動作放大，圖就容易畫錯。想畫得快，需要換一種學法：學會完整動作、預測老師最後畫到哪裡，或學會成品應該長什麼樣子。這些方法都能加快作畫，但學習的東西不同。

擴散去雜訊器通常預測某個雜訊程度下的局部修正；單步生成器則必須把隨機輸入變成完整樣本。它需要學會**有限位移**，不能只把局部修正做得更有信心。本章整合擴散蒸餾、一致性模型、rectified flow，以及 shortcut／平均速度模型。有些借助預訓練教師，有些直接訓練生成器。「單步」描述推論方式，並不代表大家使用同一種訓練演算法。

先讀 ML 08：擴散模型與流匹配，理解加雜訊排程、預測目標與機率流方程式；DLCV 12：影像生成 則介紹整體管線與評估。本章聚焦：**把反覆迭代壓縮成一次或少數幾次呼叫時，究竟要改變什麼？**超解析度以觀測影像為條件，另見單步擴散超解析度。

第一次先看數值實驗、方法家族表與評估原則，再挑一篇論文導讀。公式用來分辨訓練目標；不必先讀懂所有推導，才能掌握家族之間的差異。

為什麼放大一步會不同

白話說。局部切線提供目前方向，不是到遠方端點的位移；局部預測準確，單步跳躍仍可能很差。

將機率流或流匹配的常微分方程式寫成：

$$\frac{dx_t}{dt} = v(x_t, t), \qquad x_{t+h} \approx x_t + h\, v(x_t, t).$$

$x_t \in \mathbb{R}^d$ 是目前的影像或潛在向量，t 是連續時間，v 是瞬時速度，h 是帶正負號的時間增量。右式是 Euler 更新。局部誤差取決於路徑上速度如何隨時間與位置改變；放大 h 就忽略更多變化。即使神經網路更準確地預測同一個瞬時速度，也不會自動修好積分誤差。

必須區分三種誤差：學到的向量場偏離目標，形成**模型誤差**；有限步求解器近似連續動態，形成**數值誤差**；推論的起始雜訊、條件或解析度與訓練不同，形成**分布不匹配**。改善其中一項不會消除另外兩項。機率流理論 [1] 說明理想邊際分布的連結，並不保證有限的神經取樣器精確遵循它。

流匹配 [2] 也一樣。即使訓練配對沿直線內插，同一位置的邊際速度仍可能平均多組配對的方向，使實際軌跡彎曲。訓練內插線是直的，不等於學到的邊際流也筆直。

已知精確分數函數的小實驗

以下排除神經網路近似誤差，以等權重的四個二維高斯分布為目標。加入標準差 σ 的高斯雜訊後，各成分變異數為 $a^2 + \sigma^2$，其中原始標準差 $a = 0.12$。精確分數函數為：

$$s(x, \sigma) = -\frac{1}{a^2 + \sigma^2} \sum_{j=1}^{4} w_j(x, \sigma)(x - \mu_j).$$

μ_j 是第 j 個成分的中心，w_j 是在位置 x 的正規化後驗權重。這種加性雜訊族以雜訊尺度參數化時，機率流方程式為 $dx/d\sigma = -\sigma s(x, \sigma)$ [3]。積分方向從大 σ 到小 σ。

起點取自**精確的加雜訊混合分布**，並非直接假設為標準高斯，因此能控制數值比較的條件。實際生成器還要處理起始分布與訓練邊界是否吻合。我們從同一批起點比較粗略 Euler 終點與高精度自適應 ODE 解；此誤差不是影像品質指標，也不是訓練模型的基準成績。

```
import numpy as np
from scipy.integrate import solve_ivp

centers = np.array([[-1.7, 0.], [1.7, 0.], [0., -1.7], [0., 1.7]])
base_var = 0.12**2
sigma_start, sigma_end = 3.0, 0.04

def score(x, sigma):
    """Exact score of the equally weighted Gaussian mixture after Gaussian noising."""
    var = base_var + sigma**2
    delta = x[:, None, :] - centers[None, :, :]
    logits = -(delta**2).sum(-1) / (2 * var)
    weights = np.exp(logits - logits.max(1, keepdims=True))
    weights /= weights.sum(1, keepdims=True)
    return -(weights[:, :, None] * delta).sum(1) / var

def euler(initial, steps):
    x = initial.copy()
    grid = np.linspace(sigma_start, sigma_end, steps + 1)
    for s0, s1 in zip(grid[:-1], grid[1:]):
        x += (s1 - s0) * (-s0 * score(x, s0))
    return x
```

```
rng = np.random.default_rng(7)
initial = centers[rng.integers(4, size=512)] + np.sqrt(base_var + sigma_start**2) *
 ↪  rng.normal(size=(512, 2))
shape = initial.shape
solution = solve_ivp(lambda sigma, flat: (-sigma * score(flat.reshape(shape),
 ↪  sigma)).ravel(),
                     (sigma_start, sigma_end), initial.ravel(), rtol=1e-9, atol=1e-10,
                      ↪  dense_output=True)
reference = solution.y[:, -1].reshape(shape)
for steps in [1, 2, 4, 8, 16, 32, 64]:
    estimate = euler(initial, steps)
    rmse = np.sqrt(np.mean((estimate - reference)**2))
    print(f'{steps:2d} evaluations: endpoint RMSE {rmse:.4f}')
```

```
 1 evaluations: endpoint RMSE 0.8098
 2 evaluations: endpoint RMSE 0.5686
 4 evaluations: endpoint RMSE 0.3011
 8 evaluations: endpoint RMSE 0.1224
16 evaluations: endpoint RMSE 0.0316
32 evaluations: endpoint RMSE 0.0164
64 evaluations: endpoint RMSE 0.0085
```

這些數字來自固定種子的實際執行。即使用精確分數函數，一次 Euler 評估仍有很大的終點誤差；增加評估次數才能更準確地跟隨彎曲路徑。參考解也是容許誤差很小的數值解，並非解析的封閉形式傳輸映射。

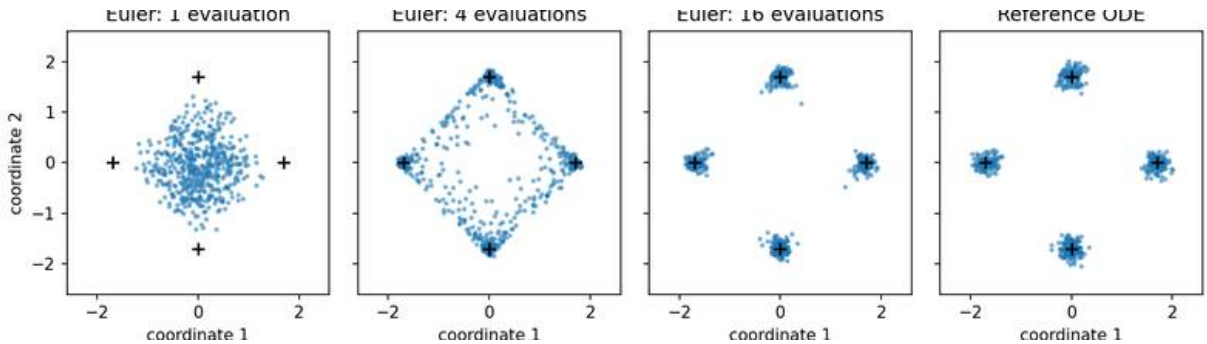


圖 *1* — 即使分數函數精確，只減少評估次數、沒有改變預測目標，仍會產生積分誤差。這是合成數值實驗，不是影像模型的輸出。

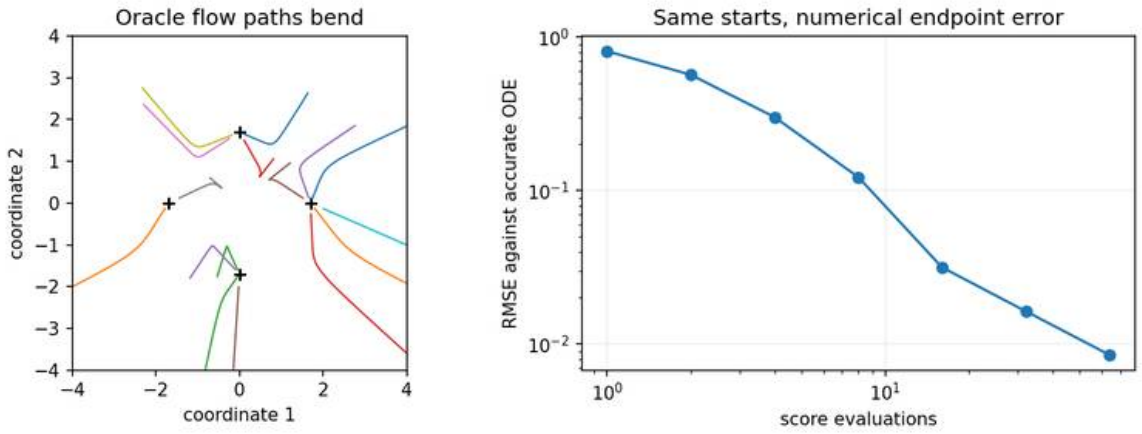


圖 2 — 切線與有限位移不同。蒸餾與大步長目標要學的資訊，並不存在於單步 *Euler* 基準的局部預測裡。

加速方法的地圖

白話說。分開看待更快積分既有模型，以及重新訓練端點、跳躍或分布預測器。

家族	學習或改變什麼	需要獨立教師嗎？	主要限制
快速求解器	改善既有向量場的數值更新	使用原模型	少步準確度仍取決於向量場與排程
漸進／軌跡蒸餾	把教師多步壓成學生一步	需要	遵循選定的教師映射
一致性模型	同一路徑不同位置預測相同乾淨端點	蒸餾需要；直接訓練不需要	不同雜訊程度的端點預測須一致
Reflow 與端點蒸餾	先改善雜訊—影像耦合，再學直接映射	大型影像應用通常需要	拉直路徑與近似映射是兩個階段
對抗蒸餾	寫實輸出與教師指引	ADD ／ LADD 需要	判別器穩定性與多樣性
分布匹配	學生輸出分布符合教師定義的目標	需要	學生分數模型必須追蹤持續變動的分布
Shortcut ／平均速度	以時間區間為條件的有限位移	可從頭訓練	自我目標準確度、時間條件與邊界行為

此表依擴散蒸餾綜述 [4] 區分目標，另加入直接訓練家族，因為本章談的是單步生成，不限於蒸餾。分類會重疊：DMD2 結合分布匹配與對抗目標；InstaFlow 結合 reflow 與端點蒸餾。檢查實際損失，比替模型貼上唯一標籤更有用。

訓練學生前，先以 DDIM [5]、DPM-Solver [6] 作為基準。它們改善每次評估的離散化誤差，卻不會自行學出完整端點映射。拿沒有效率的教師取樣器比較，可能誇大蒸餾加速；中等步數表現良好的求解器，也未必能縮到一次評估。

壓縮教師軌跡

白話說。教師為各含雜訊輸入指定目的地，端點迴歸讓學生直接重現選定的映射。

漸進蒸餾

Progressive Distillation [7] 反覆讓學生一步符合教師確定性的兩步。訓練後學生接任教師，再把步數減半，形成逐步放大跳躍的學習課程，避免新網路立刻近似完整映射。

設確定性教師從 z_t 經過中間時間到達 z_s。以乾淨預測 $\hat{x}$ 表示的 DDIM 式單步更新為：

$$z_s = \frac{\sigma_s}{\sigma_t} z_t + \left(\alpha_s - \frac{\sigma_s}{\sigma_t} \alpha_t \right) \hat{x}.$$

z_t, z_s 是含雜訊狀態，α_t, σ_t 是訊號與雜訊排程係數。反解這個仿射方程式，就得到能重現教師兩步終點的學生目標。目標取決於教師映射與起點；它不同於一般乾淨影像去雜訊，後者的平方誤差最優解可能平均多個相容樣本。

推論節省的計算轉移到訓練：教師要做多次評估來產生目標，且階段之間可能累積誤差。學生也依賴特定排程與輸出參數化；任意更換取樣器，不等於使用訓練好的程序。

僅線上版：*Progressive Distillation*：教師兩步壓成學生一步（*ICLR 2022*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/progressive-distillation 教師兩步壓成學生一步 iclr-2022/*

耦合也是目標的一部分

從雜訊分布到影像分布可以有許多合法映射。教師軌跡選定一種**耦合**（coupling），指定哪個隨機輸入對應哪個輸出。匹配這組配對，比只匹配輸出分布更嚴格。

例如對稱隨機輸入 z，$G_1(z) = \operatorname{sign}(z)$ 與 $G_2(z) = -\operatorname{sign}(z)$ 都有平衡的兩模態輸出分布，但逐樣本對應完全相反。配對迴歸會強烈懲罰交換兩者，即使邊際分布沒有改變。

因此，單步生成不必然「太確定」。**隨機潛在輸入**經確定性映射仍能產生多樣輸出；模式崩潰發生在模型忽略隨機輸入，或只涵蓋少數目標模態時。把不同種子的無關終點拿來平均又是另一個問題：平均值可能落在模態之間，成為不合理樣本。

一致性模型的端點一致性

白話說。不學下一個小修正，改學路徑最後到哪裡。同一路徑上的兩個狀態，應該給出同一個答案。

一致性函數滿足：

$$f(x_t, t) = f(x_s, s) = x_\varepsilon$$

x_t, x_s 位於同一條機率流軌跡，x_ε 是小正邊界時間 ε 的近乾淨端點。神經網路以以下參數化強制邊界條件：

$$f_\theta(x, t) = c_{\text{skip}}(t) x + c_{\text{out}}(t) F_\theta(x, t),$$

F_θ 是可學網路，$c_{\text{skip}}(\varepsilon) = 1$、$c_{\text{out}}(\varepsilon) = 0$。邊界因此回傳輸入，而非允許所有位置都輸出同一常數。邊界錨定與正確配對很重要；只要求兩個預測相等，仍有無用的平凡解。

一致性蒸餾使用預訓練分數模型與求解器產生相鄰路徑狀態，讓較吵狀態的學生輸出，符合較乾淨狀態上停止梯度的目標網路輸出。**一致性訓練**則從資料、鄰近雜訊程度與共用雜訊建構配對，不需要獨立預訓練擴散教師 [8]。後續研究 [9] 修改權重、雜訊抽樣及目標更新；不能把最初設定當成所有一致性模型的最終形式。

單步取樣在起始雜訊上評估 f_θ；少步取樣則在端點預測之間，於較低雜訊程度重新加雜訊再預測。它不是反覆把同一確定性映射套用到自己的輸出。模型訓練的雜訊範圍與取樣排程必須相符。

僅線上版：*Consistency Models*：沿同一路徑預測相同端點（*ICML 2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/consistency-models 沿同一路徑預測相同端點 icml-2023/*

潛在一致性與引導

Latent Consistency Models（LCM）在預訓練自動編碼器的潛在空間進行一致性蒸餾 [10]、[11]。空間計算減少，但仍有編解碼成本與重建限制。去雜訊器轉為受引導潛在機率流的端點預測器。

一般無分類器引導結合有條件與無條件預測。LCM 將引導強度作為學生條件，蒸餾已引導的流，使每一步不必另做兩次去雜訊預測。可用引導範圍取決於訓練；蒸餾與原管線的相同引導數字，也可能因慣例和參數化而代表不同效果。

LCM 的 skipping-step 配對跨越較遠的時間間隔，強化有限跳躍的端點目標，但仍依賴教師與訓練排程。一步可能比兩步或四步差；支援單步推論，不代表一步永遠是最佳設定。

僅線上版：*Latent Consistency Models*：受引導潛在流的端點（*2023*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/latent-consistency-models 受引導潛在流的端點 2023/*

先拉直路徑再蒸餾

白話說。教師的路線比較不繞，學生就較容易學完整動作。Reflow 先改變路徑，再以端點迴歸壓縮它。

Rectified Flow [12] 以端點配對的內插訓練瞬時速度。每輪 reflow 先抽取雜訊 z，沿目前的流到達 $x = \Phi(z)$，再以這些**有耦合關係**的 (z, x) 訓練下一個流，而非任意獨立配對雜訊和資料。目標為：

$$x_t = (1-t)z + tx, \qquad \mathcal{L}_{\text{reflow}} = \mathbb{E}\|v_\theta(x_t, t) - (x - z)\|_2^2.$$

此處時間從雜訊端的 0 增加到資料端的 1；v_θ 是瞬時速度，Φ 是前一模型的端點映射。它與前面實驗的雜訊尺度方向不同，必須明示慣例，避免正負號反轉。

理想 reflow 在假設成立時具分布保留與傳輸成本性質；有限網路和近似取樣仍會引入誤差。一輪 reflow 不保證所有軌跡都筆直，也可能仍需多次 Euler 評估。InstaFlow [13] 特別區分此階段與後續的單步端點蒸餾。

端點學生可最小化 $d(G_\theta(z, c), \Phi_{\text{reflow}}(z, c))$；$c$ 是文字條件，d 是重建或感知距離。Reflow 改變配對並在理想情況保留邊際分布，端點迴歸再近似改變後的映射，兩者並非同一目標。

僅線上版：*InstaFlow*：先 *reflow*，再蒸餾端點（*ICLR 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/instaflow 先-reflow 再蒸餾端點 iclr-2024/*

加入寫實度訊號

白話說。模仿教師終點也可能複製教師錯誤。判別器補上另一種訊號：輸出是否像目標資料？它能改善外觀，但仍須穩定訓練並檢查多樣性。

Adversarial Diffusion Distillation（ADD）結合預訓練學生、凍結擴散教師與判別器 [14]。擴散蒸餾項把學生輸出拉向教師去雜訊目標；對抗項則在預訓練特徵空間獎勵寫實外觀。教師與判別器用於訓練，部署時不必再執行迭代教師取樣。

一般生成器的對抗項可含 $-\mathbb{E}[D_\phi(G_\theta(z,c),c)]$，其中 D_ϕ 提供真實感分數，G_θ 是學生。判別器以自己的損失與正規化分辨真實、生成樣本；此分數不是影像真實或符合提示詞的校準機率。

ADD 可在訓練過的少數雜訊程度間細化。LADD [15] 把對抗蒸餾移到潛在擴散特徵，減少訓練時反覆解碼像素及執行大型影像特徵編碼器的需求。比較成本時，要分別計入訓練的教師、判別器與特徵抽取，以及推論的最終解碼器。

對抗與擴散損失都不自動保證多樣性。檢查不同種子的構圖、罕見概念與重複紋理；低 FID 仍可能伴隨條件覆蓋不足。實際資料、提示詞如何進入判別器與訓練穩定性，也都是比較條件。

僅線上版：*Adversarial Diffusion Distillation*：教師與判別器（*ECCV 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/adversarial-diffusion-distillation 教師與判別器 eccv-2024/*

匹配分布，而非每個指定終點

白話說。學生可以學著產生正確的一整群影像，不必每個雜訊種子都複製教師的同一張圖。但學生一直變動，訓練也必須持續估計它目前的輸出分布。

DMD [16] 訓練單步生成器，使用兩個擴散分數估計：凍結模型描述教師定義的目標分布；另一模型追蹤學生目前的輸出分布。在相同雜訊程度下，兩者的分數差提供把生成樣本移向目標的方向。

令 $x = G_\theta(z,c)$、$x_t = \alpha_t x + \sigma_t \epsilon$，代表性的更新形式為：

$$\nabla_\theta \mathcal{L} \approx \mathbb{E}\left[w(t)\alpha_t \left(s_{\text{student}}(x_t,t,c) - s_{\text{target}}(x_t,t,c) \right) \frac{\partial G_\theta(z,c)}{\partial \theta} \right].$$

z 是生成器的隨機輸入，c 是條件，ϵ 是新抽取的高斯雜訊，α_t, σ_t 是加雜訊係數；$s = \nabla_x \log p_t(x \mid c)$ 是分數函數，$w(t)$ 是訓練權重。此式簡寫回傳生成器參數的向量—Jacobian 乘積。它使用近似分數與時間權重，並非直接、精確計算影像密度的 KL 散度。

為什麼需要學生分數？只追隨目標分數可能把樣本推往少數吸引模態；分數差也考量學生已經放了多少機率質量。為什麼加雜訊？乾淨分布可能幾乎不重疊，加雜訊提供較平滑的比較空間。為什麼另訓練分數網路？學生更新後，其分布也變了，凍結教師無法描述它。

原始 DMD 另以教師產生的雜訊—影像配對做迴歸，維持穩定與覆蓋，因此並非完全沒有軌跡目標。SwiftBrush 的變分分數蒸餾與 SiD 的分數恆等式目標 [17]、[18] 相關，但目標、權重與資料需求不同，不能視為同一公式的相同實作。

DMD2 改變了什麼

DMD2 [19] 移除離線端點迴歸資料集，增加學生分數網路相對於生成器的更新頻率，並加入真實影像的對抗學習，以修正教師目標的偏差。圖中標為「real」的教師分數仍是近似，並非直接取得真實資料密度。

多步學生訓練也模擬自己的中間取樣狀態，減少教師加雜訊輸入與實際學生路徑不符的問題。大型模型的四步成績不能當成相同模型的一步成績；表格每列都要連同骨幹、引導、步數與評估規則一起讀。

僅線上版：*DMD*：匹配教師定義的分布（*CVPR 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/dmd 匹配教師定義的分布 cvpr-2024/*；*DMD2*：追蹤學生，加入真實資料（*NeurIPS 2024*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/one-step-generation/papers/dmd2 追蹤學生加入真實資料 neurips-2024/*

直接學習有限跳躍

白話說。網路可以學「這段時間要移動多遠」，不只學「現在往哪裡走」。輸入與目標改變後，就能不靠獨立教師，直接訓練大步長模型。

Shortcut 模型

Shortcut 模型 [20] 以指定區間 d 為條件，預測正規化位移 $s_\theta(x,t,d)$，更新為 $x' = x + d\,s_\theta(x,t,d)$。$d = 0$ 時用流匹配速度目標提供基礎；較長區間則要求一次長跳符合兩次連續短跳：

$$s(x,t,2d) = \tfrac{1}{2}\left[s(x,t,d) + s(x + d\,s(x,t,d), t+d, d)\right].$$

t 是目前時間，d 是短區間長度。第二項必須在**更新後的位置與時間**評估；以停止梯度的短跳目標訓練長跳。如果仍使用原時間，就改變了目標。

流監督避免只靠任意自我一致性。模型可在同一訓練階段混合局部目標與自舉長跳。「無獨立教師」仍有自我目標、訓練成本與近似誤差；大步品質受容量與短跳目標準確度限制。

MeanFlow 的平均速度

MeanFlow [21] 區分區間平均速度與端點的瞬時速度。以資料到雜訊的時間慣例定義：

$$u(z_t, r, t) = \frac{1}{t-r}\int_r^t v(z_\tau, \tau)\,d\tau, \qquad z_r = z_t - (t-r)u(z_t, r, t).$$

$r < t$，z_t 是流上的狀態，v 是瞬時速度，u 是同一軌跡的區間平均速度。積分效果不能用一次 v 評估取代；模型要直接學會這個平均效果。

固定 r，沿路徑微分該定義，可得：

$$u = v - (t - r)\,(\partial_t u + (\nabla_z u)v)\,.$$

這裡是沿狀態變動的全微分。訓練透過 Jacobian—向量乘積（JVP）及停止梯度目標計算，不必實體化整張影像的 Jacobian。流匹配建構用來替代未知邊際速度目標；它學的量與「放大 Euler 步長」不同。

模型必須表示起始與終止時間。當 $r = t$，平均速度趨近瞬時速度；完整區間則由一次平均速度預測給出整段位移。理想恆等式只啟發目標，有限網路仍是近似。原文 JVP 消融也顯示，對哪些變數微分會實質影響結果。

完整評估速度與品質的取捨

白話說。快速輸出仍必須完成原本任務。生成分布、條件、多樣性與總成本要一起評估。

步數、評估次數與時間

取樣一步是一次狀態更新；網路函數評估次數（NFE）計算網路執行。求解器每步可能多次評估，無分類器引導也可能各做有條件與無條件預測。把兩者合批雖只呼叫一次 API，仍處理雙倍樣本。應清楚說明計數慣例。

完整延遲包含文字編碼、條件抽取、生成／去雜訊、解碼及後處理。分開模型載入與穩態推論，固定硬體、精度、解析度、批次大小與同步方式。單次執行的大模型也可能超過記憶體預算。

已蒸餾進學生的引導是系統加速的一部分，但要報告可用範圍。若快取提示詞嵌入，也須明示；拿有快取的學生與沒有快取的教師比較，並非只測蒸餾效果。

品質、符合度與多樣性

FID [22] 比較影像特徵分布的平均與共變異數，不比較成對影像，也不直接檢查提示的關係是否成立。特徵模型、參考切分、樣本數與縮放都影響分數；FID-5k 與 FID-30k 是不同規則。

CLIP 類分數 [23] 比較影像與文字嵌入，實作的正規化、尺度及骨幹可能不同。0.3 與 30 可能只是縮放慣例。GenEval [24] 以結構化規則檢查物件、數量、顏色與關係，補足分布指標和人類評估，而非取代它們。

多樣性要跨提示詞檢查，也要固定提示詞、改變種子。模型可能涵蓋許多類別，卻每個提示都只畫一種構圖；反過來，變化很多也可能是忽略提示。保留隨機輸入不保證模型有善用它。

正確閱讀基準表格

CIFAR-10、類別條件 ImageNet、文字條件 COCO [25]、[26]、[27] 回答不同問題。低解析度無條件成績，不代表高解析度提示遵循能力；四步成績也不是一步的操作點。

比較時記錄任務、骨幹與初始化、資料與教師、解析度、引導、NFE、總延遲與記憶體、分布品質、條件符合度、多樣性及有限樣本不確定性。個別元件的因果主張，優先看同篇論文的控制消融；跨論文數字無法排除資料、架構與訓練預算的差異。

失敗模式與方法選擇

白話說。加速會轉移工作與假設，不會讓它們消失。先決定哪些行為必須保留，再挑目標。

- **固定種子要有相同終點：**可先考慮軌跡或端點迴歸；它保留特定耦合，也可能複製教師缺陷。
- **需要彈性步數：**一致性、shortcut 與區間條件模型支援多種預算，但須遵循訓練好的取樣程序。
- **要改變分布，不要求逐樣本配對：**分布匹配提供自由度；注意輔助分數落後、模態遺漏與穩定性。
- **重視感知外觀：**對抗項可能有幫助，仍須評估多樣性與條件符合度。
- **不能使用預訓練教師：**直接一致性、shortcut 或平均速度訓練可符合條件；仍要比較總訓練成本。

單步模型可能繼承偏差、遺失罕見概念、過度擬合評估提示詞，或在蒸餾後改動控制時失敗。分布目標與自我目標也可能遇到分數不準、目標崩潰、邊界條件不良及訓練／測試輸入不匹配。窄範圍時間或引導訓練的模型，並非教師的通用替代品。

現代觀點

白話說。研究逐漸轉向：有限推論預算下，應該學哪一種量？不同改進並不一定都在近似同一個去雜訊步驟。

Fan 等人的蒸餾綜述 [4] 區分保真／輸出目標、軌跡方法及對抗方法，幫助辨識學生的監督與教師保留部分；類別並不互斥，例如 DMD2 同時使用分布和對抗訊號。Yang 等人的擴散綜述 [28] 將高效率取樣放在整體模型與應用中；流匹配指南 [2] 則從傳輸角度區分局部速度、端點耦合及數值積分。

2023–2025 年的一致性、reflow、對抗與分布匹配，形成不同的少步影像生成路徑。Shortcut 與 MeanFlow [20]、[21] 另展示直接學有限跳躍的路線；類別條件基準不能直接外推為大型文字生成系統的能力。2026 年 DMAD 預印本 [29] 探討對抗分布匹配，避免另維護持續更新的學生分數網路；比較時要保留其預印本身分及較廣的視覺生成設定。

開放問題包括罕見模態、組合控制、蒸餾後的編輯能力、超越特徵分布摘要的評估、訓練成本，以及跨解析度／領域轉移。復原任務還要求輸出符合特定觀測；單步超解析度 將加入這個限制。

重點整理

- 有限跳躍需要端點或位移目標，不只是把局部方向使用一次。
- 區分求解器誤差、學得模型誤差與訓練／測試不匹配。
- 軌跡迴歸保留選定的耦合，分布匹配則可學到不同耦合。
- 一致性需要邊界錨定，重流則在壓縮步數前改變耦合。

- 分布、對抗與平均速度目標提供不同監督，也有不同估計難題。
- 比較品質、提示遵循與多樣性時，同時比較完整流程延遲及網路評估次數。

練習

1. **精確場與求解器誤差**。固定分數評估次數，比較非均勻與均勻雜訊尺度網格。哪些位置應放較多點？答案是否取決於起始尺度？

   提示

   觀察軌跡彎曲及成分歸屬急遽變化之處，從相同起點比較終點 RMSE。調整網格可以改善求解器，沒有改變分數函數；不能用特定起點調好網格後，就宣稱普遍最優。

2. **端點還是分布？**對稱高斯 z 下，比較 $G_1(z) = \operatorname{sign}(z)$ 與 $G_2(z) = -\operatorname{sign}(z)$ 的期望成對平方差與輸出機率。哪一種目標分得出兩者？

   提示

   除零機率邊界外，兩輸出相差 2，平方差為 4；兩者都各以二分之一機率輸出兩個模態。理想邊際分布比較無法分辨，配對迴歸可以。

3. **錯誤的 shortcut**。在兩次短跳公式中，把第二個時間 $t + d$ 改為 t。即使更新了位置，為什麼也不再是相同一致性關係？

   提示

   向量場也隨時間變化。例如純量場 $v(x, t) = t$，兩次方向的差異完全來自時間推進。第二跳仍問原時間，就問錯了區間。

4. **計算整條管線**。教師每次更新做兩次去雜訊預測，共 20 次更新；學生已蒸餾引導，共四次更新。兩者都各做一次文字編碼與解碼。求去雜訊評估比，說明實際時間比為何不同。

   提示

   教師與學生分別為 40 與 4 次，比值 10。共同編解碼成本、批次效率、精度與記憶體行為都會改變時間比，應先建立含固定成本的模型。

5. **可疑的進步**。學生 FID 較低，但提示關係正確率下降、不同種子輸出幾乎相同。在接受其保留教師品質前，提出三項檢查。

   提示

   檢查固定提示的種子多樣性、保留提示集的物件及關係條件，並以相同樣本數與參考切分重做分布評估。單一彙總統計無法確定條件覆蓋。

參考文獻

1. Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, B. Poole, “Score-Based Generative Modeling through Stochastic Differential Equations,” *ICLR*, 2021. paper
2. Y. Lipman, M. Havasi, P. Holderrieth, et al., “Flow Matching Guide and Code,” *arXiv tutorial*, 2024. paper
3. T. Karras, M. Aittala, T. Aila, S. Laine, “Elucidating the Design Space of Diffusion-Based Generative Models,” *NeurIPS*, 2022. source
4. X. Fan, Z. Wu, H. Wu, L. Cao, “A Survey on Pre-Trained Diffusion Model Distillations,” *arXiv preprint*, *version 2*, 2025. paper
5. J. Song, C. Meng, S. Ermon, “Denoising Diffusion Implicit Models,” *ICLR*, 2021. paper
6. C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, J. Zhu, “DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps,” *NeurIPS*, 2022. paper
7. T. Salimans, J. Ho, “Progressive Distillation for Fast Sampling of Diffusion Models,” *ICLR*, 2022. paper
8. Y. Song, P. Dhariwal, M. Chen, I. Sutskever, “Consistency Models,” *ICML*, 2023. paper
9. Y. Song, P. Dhariwal, “Improved Techniques for Training Consistency Models,” *ICLR*, 2024. paper
10. S. Luo, Y. Tan, L. Huang, J. Li, H. Zhao, “Latent Consistency Models: Synthesizing High-Resolution Images with Few-Step Inference,” *arXiv preprint*, 2023. paper
11. R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, “High-Resolution Image Synthesis with Latent Diffusion Models,” *CVPR*, 2022. source
12. X. Liu, C. Gong, Q. Liu, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow,” *ICLR*, 2023. paper
13. X. Liu, X. Zhang, J. Ma, J. Peng, Q. Liu, “InstaFlow: One Step is Enough for High-Quality Diffusion-Based Text-to-Image Generation,” *ICLR*, 2024. paper
14. A. Sauer, D. Lorenz, A. Blattmann, R. Rombach, “Adversarial Diffusion Distillation,” *ECCV*, 2024. paper
15. A. Sauer, F. Boesel, T. Dockhorn, A. Blattmann, P. Esser, R. Rombach, “Fast High-Resolution Image Synthesis with Latent Adversarial Diffusion Distillation,” *SIGGRAPH Asia Conference Papers*, 2024. paper ·published version
16. T. Yin, M. Gharbi, R. Zhang, E. Shechtman, F. Durand, W. T. Freeman, T. Park, “One-step Diffusion with Distribution Matching Distillation,” *CVPR*, 2024. paper
17. T. H. Nguyen, A. Tran, “SwiftBrush: One-Step Text-to-Image Diffusion Model with Variational Score Distillation,” *CVPR*, 2024. paper
18. M. Zhou, H. Zheng, Z. Wang, M. Yin, H. Huang, “Score identity Distillation: Exponentially Fast Distillation of Pretrained Diffusion Models for One-Step Generation,” *ICML*, 2024. paper
19. T. Yin, M. Gharbi, T. Park, R. Zhang, E. Shechtman, F. Durand, W. T. Freeman, “Improved Distribution Matching Distillation for Fast Image Synthesis,” *NeurIPS*, 2024. paper
20. K. Frans, D. Hafner, S. Levine, P. Abbeel, “One Step Diffusion via Shortcut Models,” *ICLR*, 2025. paper
21. Z. Geng, M. Deng, X. Bai, J. Z. Kolter, K. He, “Mean Flows for One-step Generative Modeling,” *NeurIPS*, 2025. paper
22. M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, “GANs Trained by a Two

Time-Scale Update Rule Converge to a Local Nash Equilibrium," *NeurIPS*, 2017. source
23. A. Radford, J. W. Kim, C. Hallacy, et al., "Learning Transferable Visual Models From Natural Language Supervision," *ICML*, 2021. paper
24. D. Ghosh, H. Hajishirzi, L. Schmidt, "GenEval: An Object-Focused Framework for Evaluating Text-to-Image Alignment," *NeurIPS Datasets and Benchmarks*, 2023. paper
25. A. Krizhevsky, V. Nair and G. Hinton, "CIFAR-10 and CIFAR-100 datasets," official dataset page; companion technical report: A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009. dataset and report
26. J. Deng, W. Dong, R. Socher, L. Li, Kai Li, Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *CVPR*, 2009. doi
27. T. Lin, M. Maire, S. Belongie, et al., "Microsoft COCO: Common Objects in Context," *ECCV*, 2014. paper
28. L. Yang, Z. Zhang, Y. Song, et al., "Diffusion Models: A Comprehensive Survey of Methods and Applications," *ACM Computing Surveys*, 2023. paper
29. Z. Yu, J. Yuan, H. Yang, et al., "DMAD: Distribution Matching as Adversarial Distillation for Fast Visual Generation," *arXiv preprint*, 2026. paper
30. R. Zhang, P. Isola, A. A. Efros, E. Shechtman and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," in *Proc. IEEE/CVF CVPR*, 2018, pp. 586–595. doi ·arXiv
31. C. Schuhmann, R. Beaumont, R. Vencu, et al., "LAION-5B: An open large-scale dataset for training next generation image-text models," *NeurIPS Datasets and Benchmarks*, 2022. paper

DLCV 13・世界模型：影片、潛在動態與規劃

世界模型是什麼、模型式強化學習如何使用它（World Models、Dreamer 系列、MuZero）、為什麼影片生成器被當成世界模擬器、潛在動態與 JEPA 式預測表示如何運作，以及如何評估它們，並附上完整的 CPU 實驗。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/world-models/

你將學到

- 區分觀測、隱藏狀態、信念與動作。
- 比較線上規劃、想像中的策略學習與價值等價搜尋。
- 區分合理影片預測與有用的動作條件化模擬。
- 理解循環與特徵預測式潛在動態。
- 量測多步誤差，並偵測規劃器利用模型錯誤的情況。

先看全貌

追蹤問的是物體去了哪裡。世界模型問的是接下來可能發生什麼，尤其是**如果代理人採取某個動作**會發生什麼。機器人靠近杯子時，光有物體標籤不夠：它需要預測夾爪和杯子會怎麼移動、哪些結果不確定，以及哪個動作較可能達成目標。

> **用最簡單的話說**
>
> 推玩具車之前，你會先想像它會跑到哪裡。你可以想像輕推和用力推，挑比較好的那一種，推下去，再看看實際發生了什麼。世界模型就是這種想像力的學習版本。規劃器則是在想像中試推、挑出一個動作的那部分。每推一次就重新看一眼，能讓想像中的小錯誤不會越積越多。

「世界模型」一詞涵蓋好幾個研究傳統。Ding 等人依兩種功能整理文獻：建立能捕捉世界運作方式的內部表示，以及預測未來狀態以模擬並引導決策 [1]。本章採用一個操作型定義：**世界模型是一個學習得到的模型，它表示足夠多的環境資訊，能在指定條件（通常包括代理人自己的動作）下預測相關的變化**。它的用處取決於任務。用來渲染場景的模型、用來學習表示的模型，以及用來控制機器人的模型，可能需要非常不同的資訊。

觀測、狀態、信念與動作

白話版。影像只揭露環境的一部分，記憶或信念須處理隱藏物件與未觀測運動。

令 s_t 表示時間 t 的真實環境狀態，o_t 為觀測，a_t 為執行的動作：

$$s_{t+1} \sim p(s_{t+1} \mid s_t, a_t), \qquad o_t \sim p(o_t \mid s_t).$$

第一個分布是**轉移**（動態）模型，第二個是**觀測**模型。狀態可能包含物體位置、速度與接觸條件。影像只呈現一個投影，帶有遮擋與感測器雜訊；Szeliski 的教科書涵蓋這個觀測過程中的幾何與運動 [3]。兩張完全相同的影像可能有不同的未來：同一個位置的球，可能正往左也可能正往右。

因此，學習得到的模型會從歷史建立表示：

$$z_t = E_\theta(o_{0:t}, a_{0:t-1}), \qquad p_\theta(z_{t+1} \mid z_t, a_t),$$

其中 E_θ 是參數為 θ 的編碼器，$o_{0:t}$ 是到時間 t 為止的觀測序列，$p_\theta(z_{t+1} \mid z_t, a_t)$ 是學到的潛在轉移。編碼器不會自動讓 z_t 成為充分的狀態。若它丟掉了速度、被遮住的物體或某個相關的材質性質，轉移模型就無法找回這些資訊。循環記憶或一小段影片上下文會有幫助，但必須在實際的遮擋與延遲下測試。

當狀態只能被部分觀測時，有原則的對象是**信念**：在已知歷史下，對可能狀態的一個機率分布。Bishop 對序列潛在變數模型（隱藏馬可夫模型與線性動態系統）的說明提供了機率背景 [2]。一個緊湊的潛在向量可以近似這個分布中有用的部分；它不必一對一地對應到人類看得懂的物理變數。

預測的分解

預測未來可以拆成三個問題：現在處在什麼狀態？在這些動作下狀態怎麼變化？攝影機會看到什麼？

對一個候選的動作序列，未來觀測的機率可分解為

$$p_\theta(o_{t+1:t+H} \mid o_{\le t}, a_{0:t+H-1}) = \int p_\theta(z_t \mid o_{\le t}, a_{<t}) \prod_{k=t}^{t+H-1} p_\theta(z_{k+1} \mid z_k, a_k)\, p_\theta(o_{k+1} \mid z_{k+1})\, dz_{t:t+H},$$

其中 H 是預測視野。三個因子分別是**推論**（給定過去，目前的潛在狀態是什麼？）、**轉移**（它在這些動作下如何演變？）與**渲染**（相機會看到什麼？）。如果目標與成本定義在潛在空間中，規劃器可以直接在 z 上運作，省略像素解碼器。本章許多設計選擇，其實就是決定要學哪些因子、怎麼學。

模型式強化學習

白話版。學得的動態模型讓代理人在模擬中測試動作；預測有用時，可能減少真實嘗試。

問題設定

RL 代理人以離散步驟與環境互動。每一步它觀測環境，依**策略** $\pi(a_t \mid s_t)$ 選擇 a_t，得到獎勵 r_t，環境隨之改變。目標是最大化期望**回報**

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

其中 $\gamma \in [0,1)$ 是折扣因子，讓近期獎勵比遠期獎勵更重要。**價值** $V^\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s]$ 是從狀態 s 出發、依循 π 時的期望回報。

無模型（model-free）方法直接從真實轉移學習策略或價值。**模型式**（model-based）方法另外學習轉移與獎勵的模型，並用它減少真實互動。這個想法由來已久：Sutton 的 Dyna 架構（1991）就已經把從真實經驗學習與在學到的模型所模擬的經驗上規劃結合起來，兩者使用同樣的價值更新 [4]。Moerland 等人的綜述沿兩個軸整理這個領域：如何學習動態模型（處理隨機性、不確定性、部分可觀測性與時間抽象），以及規劃與學習如何整合（從哪裡開始規劃、給規劃多少預算、怎麼規劃，以及如何把結果回饋到行動與學習中）[5]。

使用模型的三種方式

學到模型之後，主要有三種方式把它的預測變成決策。

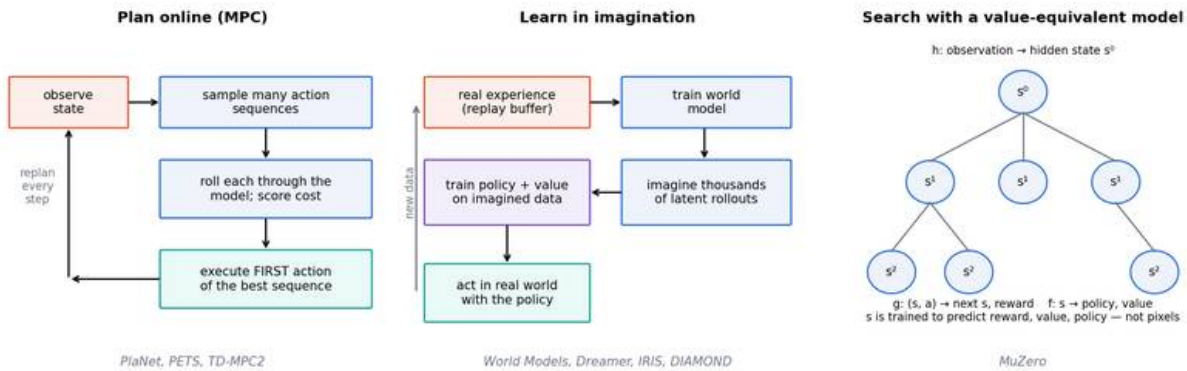


圖 *1* — 使用學到的模型的三種方式。同一個模型可以扮演不只一種角色；差別在於決策是在哪裡算出來的。

1. **線上規劃（模型預測控制）**。每一個真實步驟，都在模型中模擬候選動作序列以尋找好的序列，只執行第一個動作，觀測，再重複。PlaNet [7]、PETS [29] 與 TD-MPC2 [11] 屬於這一類。
2. **在想像中學習**。用模型產生的軌跡訓練一個策略（通常還有價值函數），執行時直接用策略行動，不再搜尋。World Models [6]、Dreamer 系列 [8], [9], [10]、IRIS [12] 與 DIAMOND [13] 屬於這一類。
3. **用價值等價模型搜尋**。學一個只預測規劃所需量（獎勵、價值、策略）、不預測觀測的模型，並在它的潛在空間中做樹搜尋。MuZero [14] 是代表。

三者有一個共同的危險：模型總會在某些地方出錯，而最佳化器非常擅長找到正好是那些地方。我們會在「不確定性與模型被利用」一節回到這點。

為什麼誤差會累積

每一步想像都從上一步想像的結果出發，所以小小的預測錯誤會被帶進下一步的預測，並可能每走一步就變大一些。

假設學到的轉移 $\hat{f}$ 的單步誤差最多為 ε，而真實動態 f 是常數為 L 的 Lipschitz 函數（狀態的小變化最多讓下一個狀態變化 L 倍）。從同一個狀態出發、施加相同動作，k 步後預測與真實狀態的差距 e_k 滿足 $e_{k+1} \le L e_k + \varepsilon$，因此

$$e_H \le \varepsilon \sum_{k=0}^{H-1} L^k = \varepsilon \, \frac{L^H - 1}{L - 1} \quad (L \neq 1).$$

當 $L > 1$ 時，這個上界隨視野 H 指數成長；當 $L < 1$ 時，它飽和於 $\varepsilon/(1-L)$。這是最壞情況的上界，但它解釋了兩個常見設計：頻繁重新規劃（MPC）與使用短的模型推演。Janner 等人把第二個想法具體化：他們的 MBPO 演算法從真實狀態分岔出短的模型推演，而不是從頭想像長軌跡，並從理論與實驗兩方面論證這能保留模型資料的好處、同時限制其偏差 [30]。

World Models：壓縮、預測、控制

白話版。先編碼觀測、預測表示的變化，再利用學得的動態訓練控制器。

Ha 與 Schmidhuber 的 *World Models*（以「Recurrent World Models Facilitate Policy Evolution」發表於 NeurIPS 2018，另有互動式線上版本）讓這個想法在深度學習中廣為人知 [6]。他們的代理人有三個部分：

- **V（視覺）**：一個變分自編碼器，把每張 64×64 的畫面壓縮成小的潛在向量 z_t。
- **M（記憶）**：一個帶混合密度輸出的循環網路，在給定 z_t、動作與隱藏狀態 h_t 時，預測 z_{t+1} 的分布。
- **C（控制器）**：一個刻意設計得極小的線性策略 $a_t = W_c [z_t; h_t] + b_c$，以演化策略而非反向傳播訓練。

在賽車任務中，使用 V 與 M 特徵的控制器解決了任務；在 VizDoom 任務中，控制器完全在 M 模擬的環境中訓練，再轉移到真實遊戲 [6]。這個設計把**感知**、**預測**與**決策**分開，並顯示當表示已經帶有相關動態時，控制器可以非常小。它也揭露了核心風險：在不完美夢境中訓練的控制器，可能學會利用夢境的錯誤。

潛在動態：從 PlaNet 到 Dreamer 系列

白話版。不預測未來的影像，而是預測一個緊湊的隱藏狀態，只在訓練時才把影像解碼出來。一個同時包含確定性記憶與隨機部分的隱藏狀態，可以表示代理人記得的東西，也能表示它不確定的東西。

循環狀態空間模型

Hafner 等人的 PlaNet 從影像學習環境動態，並在潛在空間中以快速線上規劃選擇動作；它的關鍵建模想法是同時包含確定性與隨機成分的潛在動態模型，即**循環狀態空間模型**（recurrent state-space model, RSSM）[7]。RSSM 保留一個確定性記憶 h_t 與一個隨機狀態 z_t：

$$h_t = f_\theta(h_{t-1}, z_{t-1}, a_{t-1}), \qquad z_t \sim p_\theta(z_t \mid h_t) \ \text{(prior)}, \qquad z_t \sim q_\theta(z_t \mid h_t, o_t) \ \text{(posterior)}.$$

這裡 f_θ 是一個循環單元；**先驗**（prior）在沒看到目前影像的情況下預測 z_t（想像時就用它）；**後驗**（posterior）還會看到影像 o_t（訓練時以及過濾真實觀測時使用）。多個輸出頭讀取模型狀態 (h_t, z_t)，預測影像 $\hat{o}_t$、獎勵 $\hat{r}_t$ 與延續旗標（回合是否繼續）。

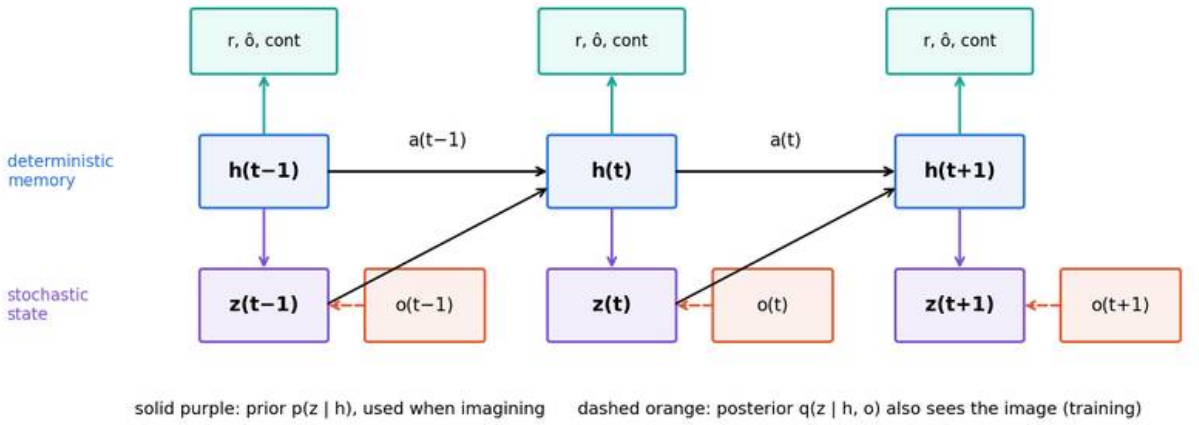


圖 2 — 展開的 *RSSM*。想像時只用先驗路徑（實線）；訓練時 z 還會以真實影像為條件（虛線）。

訓練時最大化觀測序列概似的變分下界。一個簡化的單步損失為

$$\mathcal{L}_t = -\log p_\theta(o_t \mid h_t, z_t) - \log p_\theta(r_t \mid h_t, z_t) + \beta\, D_{\mathrm{KL}}\big(q_\theta(z_t \mid h_t, o_t) \,\|\, p_\theta(z_t \mid h_t)\big),$$

其中 $\beta > 0$ 是 KL 項的權重。重建項讓狀態帶有資訊；KL 項則教先驗去預測後驗將會看到的東西，這正是想像所需要的能力。實際實作還會加入損失平衡與其他穩定化技巧，所以請把這個式子當成教學大綱，而不是任何一篇論文的精確損失。

僅線上版：*ELBO 與重新參數化技巧：訓練潛在狀態* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/world-models/notes/elbo-與重新參數化技巧訓練潛在狀態/*

Dreamer：在想像中學習行為

Dreamer（2020）保留 RSSM，但把線上規劃換成一個完全在想像出的潛在軌跡上訓練的 actor–critic。它的關鍵做法是把學到的狀態價值的解析梯度，沿著在緊湊潛在空間中想像出的軌跡反向傳播 [8]。代理人從由真實資料推得的潛在狀態出發，用 actor 想像 H 步，用學到的 critic 評分，並以自舉目標（例如 λ-回報）訓練 critic：

$$V_t^\lambda = \hat{r}_t + \gamma\, \hat{c}_t \Big[(1-\lambda)\, v_\psi(s_{t+1}) + \lambda\, V_{t+1}^\lambda \Big],$$

其中 $\hat{r}_t$ 與 $\hat{c}_t$ 是預測的獎勵與延續，v_ψ 是 critic，$\lambda \in [0,1]$ 在單步自舉（$\lambda = 0$）與較長的想像回報（$\lambda = 1$）之間取捨。actor 則被更新以提高這些價值。

這條路線持續發展：

- **DreamerV2**（2021）改用**離散**（類別型）潛在變數，成為第一個藉由在獨立訓練的世界模型中學習行為、在 55 款遊戲的 Atari 基準上達到人類水準的代理人 [9]。

- **DreamerV3**（Nature，2025）使用以正規化、平衡與轉換為基礎的穩健化技巧，使單一設定就能在超過 150 個任務上勝過專門方法，並且是第一個在沒有人類資料或課程設計的情況下、從零開始在 Minecraft 中收集到鑽石的演算法 [10]。

另外兩條相關路線說明 RSSM 只是眾多選擇之一。**TD-MPC2** 在一個**隱式**、沒有解碼器的世界模型（不重建影像）的潛在空間中規劃，以單一組超參數在 104 個線上 RL 任務上取得強勁結果，並訓練出一個能執行 80 個任務的 317M 參數代理人 [11]。**IRIS** 以離散自編碼器加上對影像 token 的自迴歸 Transformer 取代 RSSM，只用相當於兩小時的 Atari 遊戲經驗學習 [12]。**DIAMOND** 更進一步走向像素：它讓代理人在**擴散**世界模型中訓練，主張離散潛在變數可能丟掉對控制重要的視覺細節，並展示了這個模型可作為 3D 遊戲的互動式神經遊戲引擎 [13]。

MuZero：只預測規劃需要的東西

白話版。要下好一盤棋，你不需要預測畫面會長什麼樣子。你只需要預測局面有多好、哪些著法看起來有希望，以及會得到哪些獎勵。MuZero 正是學這些，然後進行搜尋。

AlphaZero 式的樹搜尋需要一個遊戲規則的模擬器。MuZero 移除了這個需求：它學一個反覆套用時能預測與規劃最直接相關之量的模型，也就是獎勵、動作選擇策略與價值 [14]。它有三個函數：

$$s^0 = h_\theta(o_1, \dots, o_t), \qquad (r^k, s^k) = g_\theta(s^{k-1}, a^k), \qquad (\mathbf{p}^k, v^k) = f_\theta(s^k),$$

其中 h_θ（表示函數）把觀測歷史編碼成隱藏狀態 s^0，g_θ（動態函數）把隱藏狀態與假設動作 a^k 映射到預測獎勵 r^k 與下一個隱藏狀態，f_θ（預測函數）輸出策略向量 $\mathbf{p}^k$ 與價值 v^k。蒙地卡羅樹搜尋展開 g_θ 來選擇動作。訓練時沿著真實軌跡把模型展開 K 步，並最小化

$$\mathcal{L} = \sum_{k=0}^{K} \Big[\ell^r(u_{t+k}, r^k) + \ell^v(z_{t+k}, v^k) + \ell^p(\boldsymbol{\pi}_{t+k}, \mathbf{p}^k) \Big] + c\|\theta\|^2,$$

其中 u 是觀測到的獎勵，z 是價值目標（自舉回報），$\boldsymbol{\pi}$ 是搜尋產生的造訪次數分布，ℓ^r, ℓ^v, ℓ^p 是各量的損失，c 是 L2 懲罰的權重。

隱藏狀態 s^k 從不被要求重建觀測。它只需要**價值等價**：用它規劃，必須得到與在真實環境中規劃相同的獎勵、價值與策略。以這個設計，MuZero 在 57 款 Atari 遊戲上創下新的最佳成績，並在沒有被告知規則的情況下，於圍棋、西洋棋與將棋上追平 AlphaZero 的超人表現 [14]。對視覺而言的教訓是：世界模型的預測目標應依用途而定；預測每個像素只是一種選擇，不是必要條件。

影片預測與生成：作為世界模擬器

白話版。如果模型能生成逼真的未來影片，也許它學會了世界如何運作。有時候確實如此；但更常見的是，它只學會了影片通常看起來像什麼。判斷是哪一種，才是難處所在。

以動作為條件的影片預測

Finn 等人在一個含 59,000 次機器人推動互動的資料集上，訓練了**以動作為條件的影片預測**模型；由於模型預測的是像素運動而非原始像素，它對沒見過的物體有部分泛化能力，而以不同的未來動作為條件，就產生了對不同未來的「視覺想像」[15]。這是最字面意義上的世界模型：影像進、動作進、影像出。

為什麼平方誤差預測器會模糊

真實的未來是不確定的。假設從同樣的過去出發，球有 $\frac{1}{2}$ 的機率往左、$\frac{1}{2}$ 的機率往右。對平方誤差而言，最佳的單一預測是**條件平均**：

$$\hat{o}^\star = \arg\min_{\hat{o}} \ \mathbb{E}[\|o_{t+1} - \hat{o}\|^2 \mid o_{\le t}] = \mathbb{E}[o_{t+1} \mid o_{\le t}],$$

在這裡就是兩顆半亮的球，一張永遠不會發生的影像。一個從兩種未來中**取樣**的生成模型會產生逼真的影像，但平均而言平方誤差**更高**。

```
import numpy as np

x = np.arange(64)
def frame(center):                      # a 1-D "image" with one bright ball
    return np.exp(-0.5 * ((x - center) / 2.0) ** 2)

# Same past, two equally likely futures: the ball bounces left or right.
futures = np.stack([frame(20), frame(44)])
mean_pred = futures.mean(axis=0)        # what an MSE-trained predictor converges to
sample_pred = futures[0]                # what a good generative sampler might output

def expected_mse(pred):                 # average over the two possible futures
    return np.mean([(pred - f) ** 2 for f in futures])

print(f"peak brightness: true future 1.00, mean prediction {mean_pred.max():.2f}")
print(f"expected MSE of mean prediction:   {expected_mse(mean_pred):.4f}")
print(f"expected MSE of a sharp sample:    {expected_mse(sample_pred):.4f}")
```

輸出：

```
peak brightness: true future 1.00, mean prediction 0.50
expected MSE of mean prediction:   0.0277
expected MSE of a sharp sample:    0.0554
```

清晰的樣本的期望誤差恰好是模糊平均的兩倍。這就是為什麼像素 MSE 會獎勵模糊、為什麼會出現隨機式影片模型，以及為什麼只用一個平方誤差數字評估世界模型，可能選錯模型。

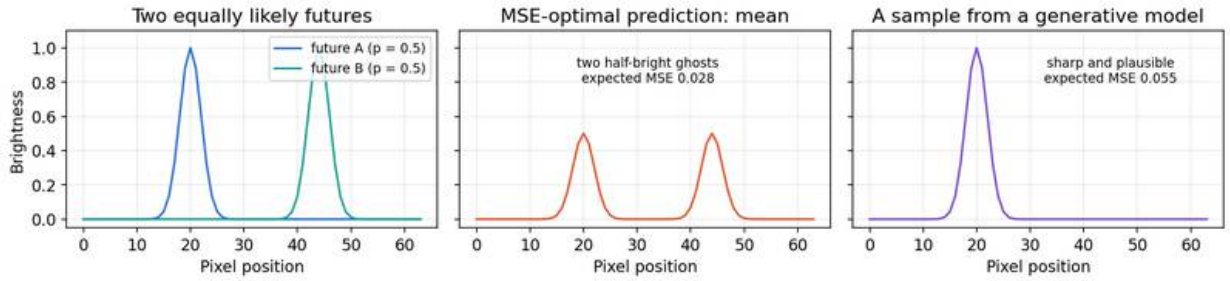


圖 *3* — 上方程式碼的輸出。*MSE* 最佳的預測把互不相容的未來平均在一起；取樣結果合理，但 *MSE* 分數較差。

Babaeizadeh 等人的 SV2P 為潛在變數的每個樣本預測一個不同的可能未來，結果比沒有隨機性的同一模型好得多 [16]。現代影片生成器在大得多的規模上使用同樣的原理，採用自迴歸 token 模型或擴散與流模型（見 ML 08 與 ML 09）。

大型生成式世界模型

有幾個大型模型能以動作、文字或從影片本身推斷出的動作為條件來生成影片，並被當作世界模擬器提出。

- **GAIA-1** 把駕駛世界建模視為對影片、文字與動作離散 token 的下一個 token 預測，能產生逼真的駕駛情境，並可細部控制自車行為與場景特徵 [17]。
- **UniSim** 結合在不同面向上豐富的資料集（影像中的物體、機器人資料中的密集動作、導航資料中的運動），學習真實世界互動的「通用模擬器」；它能模擬高階指令與低階控制，而完全在其中訓練的策略可零樣本部署到真實機器人上 [18]。
- **Genie** 是一個在未標註的網路影片上、不使用動作標籤訓練出的生成式互動環境。它由影片 tokenizer、自迴歸動態模型與一個從畫面變化推斷動作的**潛在動作模型**組成，因此即使訓練時沒有動作標籤，使用者仍能在生成的世界中逐格行動 [19]。這類潛在動作不一定對應到經過校正的實體致動器。
- **Genie 3** 由 Google DeepMind 於 2025 年 8 月發布，能從文字生成可即時導覽的世界，每秒 24 格、720p，並維持數分鐘的一致性。公告列出了限制：動作空間受限、難以模擬多個獨立代理人、真實地點不精確，以及互動只能持續數分鐘而非數小時 [20]。
- **Cosmos** 是 NVIDIA 的開放「世界基礎模型」平台：包含預訓練影片模型、影片 tokenizer、資料整理流程與後訓練範例，目的是微調成機器人與車輛專用的世界模型 [21]。

逼真的影片就代表理解物理嗎？

兩項仔細的研究給出的答案是：不一定。Motamed 等人的 **Physics-IQ** 基準測試流體力學、光學、固體力學、磁學與熱力學；在目前的影片模型（包括 Sora、Runway、Pika、Lumiere、Stable Video Diffusion 與 VideoPoet）中，物理理解嚴重不足，且與視覺真實感無關 [22]。Kang 等人在由古典力學支配的 2D 模擬器上訓練擴散影片模型，發現分布內完美泛化、組合式泛化可觀察到隨規模改善，但分布外則失敗；模型的行為像是在找最相近的訓練樣本，而不是抽象出規則 [23]。

用於機器人控制時，也要問「動作」指的是什麼：力、關節目標、夾爪指令、相機移動，還是推斷出的 token。單位錯誤、時間偏移或控制頻率不同，都會改變條件預測問題，即使影片看起來很有說服力。

預測式表示：JEPA 家族

白話版。預測每一個像素，會把力氣浪費在不重要的細節上，例如樹葉的紋理。不如預測缺失或未來部分的抽象**描述**。如果描述夠好，這個預測就對規劃有用。

聯合嵌入預測架構（joint-embedding predictive architecture, JEPA）用編碼器分別編碼可見的**上下文** x 與被隱藏的**目標** y，並訓練一個預測器把上下文嵌入映射到目標嵌入：

$$\mathcal{L} = \| P_\phi(E_\theta(x),\, m) - \mathrm{sg}[\bar{E}(y)] \|^2,$$

其中 E_θ 是上下文編碼器，$\bar{E}$ 是目標編碼器，P_ϕ 是以目標位置資訊 m（例如在影像或時間中的位置）為條件的預測器，sg（stop-gradient）阻止梯度流入目標分支。沒有任何東西在像素空間中重建。危險在於**崩塌**（collapse）：如果兩個編碼器都輸出常數，損失就是零。實務上用不對稱設計（只有一側有預測器、stop-gradient，以及不以同一梯度訓練的目標編碼器）來避免崩塌；ML 06 更詳細地討論這些自監督機制。

JEPA 系列一開始是表示學習：先用在影像上（I-JEPA [24]），再用在影片上（V-JEPA [25]），之後 V-JEPA 2.1 [27] 又改進了它的密集特徵。下面兩個模型說明它如何變成世界模型：以動作為條件的預測，以及在特徵空間中朝目標規劃。

- **V-JEPA 2** 先在超過一百萬小時的網路影片上預訓練不含動作的 JEPA，再用不到 62 小時的未標註機器人影片，後訓練出一個潛在的**以動作為條件**的世界模型 V-JEPA 2-AC；以影像目標進行規劃後，它在兩個實驗室的 Franka 機械手臂上實現零樣本的拾取與放置，且沒有使用來自這些環境的資料、任務專屬訓練或獎勵 [26]。被動影片與動作資料的分工，讓本章所說的觀測／動作區別變得具體。
- **DINO-WM** 用現成的編碼器展示同樣的想法：它從離線軌跡預測未來的 DINOv2 patch 特徵，並把動作序列最佳化到接近目標影像的特徵來規劃，不重建像素，也不學獎勵模型 [28]。

僅線上版：*JEPA 系列的其他模型：I-JEPA、V-JEPA 與 V-JEPA 2.1 — https://blog.shyandram.dev/zh-tw/learn/deep-learning/world-models/notes/jepa-系列的其他模型 i-jepav-jepa-與-v-jepa-21/*

特徵空間中的目標需要小心。距離 $\|z - z_{\text{goal}}\|^2$ 只會獎勵表示中保留下來的東西。如果表示忽略了夾爪是否閉合，規劃器可能到達一個看起來相似的狀態，卻沒有真的抓住物體。單憑特徵品質更好，並不能證明編碼器能預測任意動作的後果。

各家族之間的關係

這些家族的主要差別在於預測什麼，而每種預測目標都帶出一個需要檢驗的問題。

家族	預測目標	主要好處	要檢驗的問題
潛在動態（RSSM、TD-MPC2）	緊湊狀態、獎勵、價值	便宜、可重複推演	潛在狀態是否保留任務相關動態？
價值等價（MuZero）	只有獎勵、價值、策略	不在無關細節上花力氣	獎勵改變時還能遷移嗎？
自迴歸影片（GAIA-1、Genie、IRIS）	下一張畫面或 token	明確的序列分解	長推演時誤差會漂移嗎？
擴散／流影片（DIAMOND、Cosmos）	一起生成的畫面或片段	表達力強、能表示不確定的未來	能否在控制延遲內以動作操控結果？
預測式嵌入（JEPA、DINO-WM）	目標特徵	不需像素重建就能學到結構	動作是否被表示？目標是否有意義？

這些選擇彼此相容。Transformer 可以參數化其中任何一種；潛在世界模型可以透過擴散解碼；影片生成器可以外層用自迴歸、內層用擴散。架構、預測目標、時間分解與訓練目標是各自獨立的設計決策。

從資料學習動態

白話版。動態模型的好壞取決於資料：影格、動作與獎勵必須保持順序並正確對齊，測試回合也必須真正與訓練回合分開。

訓練樣本應保留順序與動作對齊：

$$(o_t,\, a_t,\, o_{t+1},\, r_t,\, \text{done}_{t+1}),$$

其中 done_{t+1} 標記回合結束。記錄每個動作發生在每個相機時間戳之前或之後。依完整回合、環境實例與相關受試者切分；隨機切分相鄰畫面會洩漏幾乎相同的場景，讓轉移預測器看起來比實際更好（資料洩漏的一般討論見 ML 11）。

資料收集策略很重要。在緩慢、輕柔互動上訓練的模型，可能在劇烈運動時失敗。當資料中沒有動作時，觀察到的相關性特別麻煩：模型可能學到手靠近時杯子會動，卻完全沒有證據知道其他推法會怎樣。這就是被動影片與介入式資料的差別。

單步損失衡量從真實歷史出發的預測。**多步損失**展開模型自己的預測並懲罰累積誤差。展開會改變訓練難度與梯度路徑；它並不能消除觀測互動範圍之外的分布偏移。

完整 CPU 實驗：擬合、預測、規劃

白話版。現在我們在一個玩具系統上跑完整個流程：收集資料、擬合模型、檢查它一步與多步的預測，再用它來規劃。

我們使用二維狀態 $x_t = (\text{position}, \text{velocity})$（位置與速度）、純量動作 u_t，以及一個有阻尼的線性系統：

$$x_{t+1} = Ax_t + Bu_t + \epsilon_t,$$

其中 A 是 2×2 狀態轉移矩陣，B 是動作輸入向量，ϵ_t 是微小的高斯雜訊。這個系統刻意設計成完全可觀測，以便在加入影像編碼器與部分可觀測性之前，先單獨檢視動態與規劃。訓練與測試

回合各自獨立產生。我們只在訓練轉移上擬合，並用相同的保留動作同時評估單步預測與開迴路推演。

規劃器使用**交叉熵方法**（cross-entropy method, CEM）：從高斯分布取樣候選動作序列，保留模型下成本最低的「菁英」序列，把高斯的平均與分散重新擬合到它們，然後重複。它只執行一個動作，取得新的真實狀態後重新規劃。這就是模型預測控制。

僅線上版：交叉熵方法：用取樣與重新擬合來規劃 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/world-models/notes/交叉熵方法用取樣與重新擬合來規劃/*

```
import numpy as np

rng = np.random.default_rng(7)
A = np.array([[1.0, 0.1], [0.0, 0.94]])
B = np.array([0.005, 0.1])

def collect(n, steps=40):
    x = np.zeros((n, steps + 1, 2))
    x[:, 0] = rng.normal(size=(n, 2)) * [0.8, 0.3]
    u = rng.uniform(-1, 1, size=(n, steps))
    for t in range(steps):
        x[:, t + 1] = (x[:, t] @ A.T + u[:, t, None] * B
                       + rng.normal(scale=0.002, size=(n, 2)))
    return x, u

train, train_u = collect(120)
test, test_u = collect(80)
phi = np.column_stack([train[:, :-1].reshape(-1, 2), train_u.ravel()])
target = train[:, 1:].reshape(-1, 2)
W = np.linalg.lstsq(phi, target, rcond=None)[0]
W_no_action = np.linalg.lstsq(phi[:, :2], target, rcond=None)[0]

def predict(x, u):
    return np.concatenate([x, np.asarray(u)[..., None]], axis=-1) @ W

one_step = predict(test[:, :-1], test_u)
rollout = np.zeros_like(test)
rollout[:, 0] = test[:, 0]
for t in range(test_u.shape[1]):
    rollout[:, t + 1] = predict(rollout[:, t], test_u[:, t])
rmse = lambda a, b: np.sqrt(np.mean((a - b) ** 2))
print("One-step:", rmse(one_step, test[:, 1:]))
print("Without actions:", rmse(test[:, :-1] @ W_no_action, test[:, 1:]))
print("Final-horizon rollout:", rmse(rollout[:, -1], test[:, -1]))

def plan(x, mean, horizon=20, population=256):
    std = np.ones(horizon)
    for _ in range(4):
        candidates = np.clip(rng.normal(mean, std, (population, horizon)), -1, 1)
        imagined = np.broadcast_to(x, (population, 2)).copy()
        costs = np.zeros(population)
        for t in range(horizon):
            imagined = predict(imagined, candidates[:, t])
            costs += (imagined[:, 0] ** 2 + 0.15 * imagined[:, 1] ** 2
```

```
                        + 0.02 * candidates[:, t] ** 2)
        costs += 20 * imagined[:, 0] ** 2
        elite = candidates[np.argsort(costs)[:32]]
        mean = elite.mean(axis=0)
        std = elite.std(axis=0) + 0.05
    return mean

real = np.array([1.5, 0.0])
mean = np.zeros(20)
for _ in range(40):
    sequence = plan(real, mean)
    real = A @ real + B * sequence[0]  # real system; never used inside plan()
    mean = np.r_[sequence[1:], 0.0]
print("Controlled final state:", real)
assert rmse(one_step, test[:, 1:]) < 0.01
assert abs(real[0]) < 0.1
```

輸出：

```
One-step: 0.0019866110218121832
Without actions: 0.04106321931941812
Final-horizon rollout: 0.013772759824274517
Controlled final state: [-0.00453985 -0.0210205 ]
```

單步 RMSE（0.0020）落在雜訊水準；拿掉動作輸入後增加到二十倍（0.0411），可見動作確實重要。即使是這個幾乎完美的線性模型，視野 40 的開迴路推演誤差（0.0138）也約為單步誤差的七倍，是上面推導的誤差累積的溫和例子。MPC 把位置從 1.5 帶到約 −0.0045，而零動作時位置停在 1.5。這些是本腳本的合成結果，不是已發表的比較。

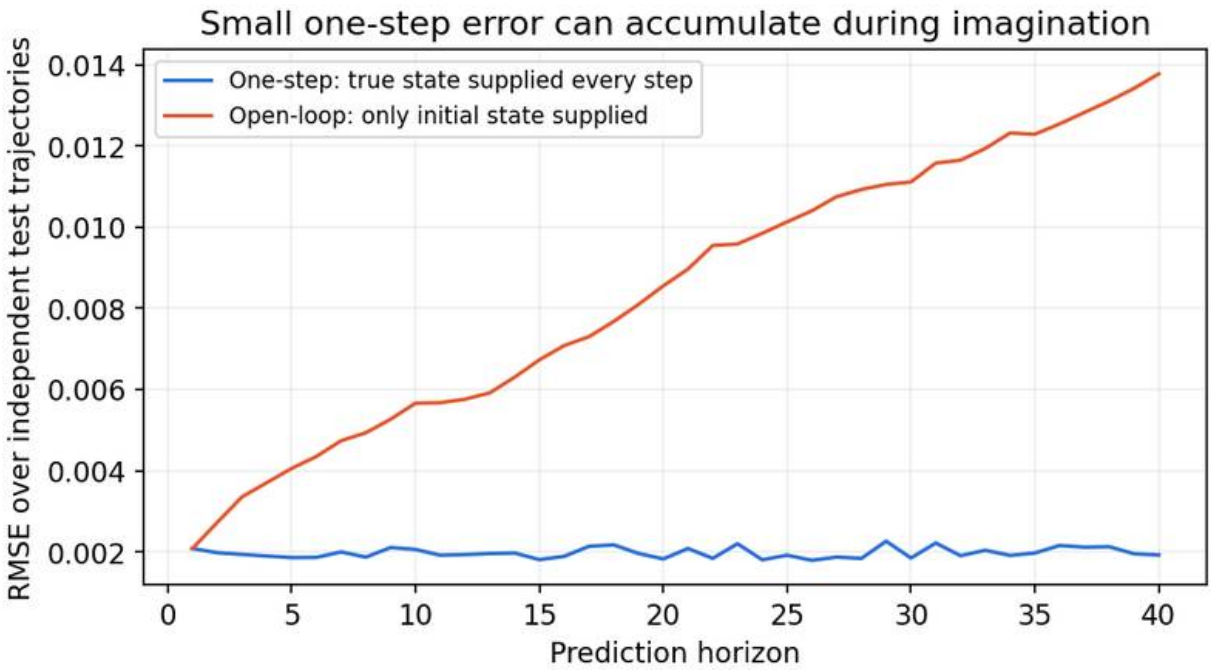


圖 *4* — 獨立的合成測試回合。單步評估每一步都拿到真實狀態；開迴路評估只拿到初始狀態與保留的動作。

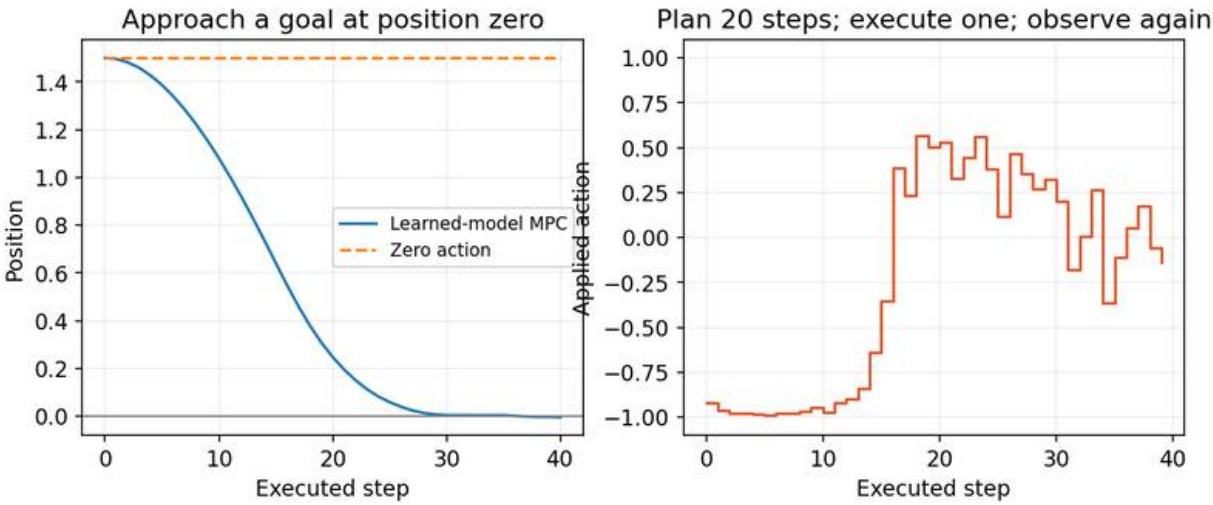


圖 *5* — *CEM* 使用學到的轉移模型規劃。在合成環境中只執行第一個動作，再從新的觀測重新規劃。

把 x_t 換成影像嵌入，並不能重現 Dreamer 或 V-JEPA 2-AC：那會多出編碼器學習問題、潛在不確定性，以及需要一個有效的目標或獎勵。

規劃、不確定性與模型被利用

白話版。規劃器會去找模型最喜歡的動作。如果模型在某處是錯的，規劃器會很樂意往那裡跑。知道模型**在哪裡**不確定，才能讓它保持誠實。

有限視野的規劃目標可寫成

$$a^{*}_{t:t+H-1} = \arg\min_{a_{t:t+H-1}} \mathbb{E}_{\theta}\Big[\sum_{k=1}^{H} c(z_{t+k}, a_{t+k-1}) + \lambda_{\mathrm{T}}\, c_{\mathrm{T}}(z_{t+H})\Big],$$

其中 c 是每步成本，c_{T} 是以 λ_{T} 加權的終端成本，期望值針對模型預測的潛在狀態 z。**模型預測控制**執行第一個動作、重新觀測、再重新規劃。回饋可以修正漂移，但無法挽救一個其直接後果就被嚴重誤判的動作。

兩種不確定性

偶然（aleatoric）不確定性是本質上的隨機性（擲硬幣、可能往左也可能往右的球）；資料再多也消除不了。**認知**（epistemic）不確定性來自資料不足（模型從沒見過這種情況）；更多資料可以消除它。對生成模型取樣可以表示偶然不確定性，但樣本多樣不代表已校準。認知不確定性常以**集成**（ensemble）近似：在 bootstrap 重抽樣上或以不同初始化訓練的多個模型，在資料密集處意見一致，在資料稀少處則傾向分歧。Chua 等人的 PETS 把機率式網路（處理偶然雜訊）組成集成（處理認知不確定性），搭配以取樣為基礎的不確定性傳遞與 CEM 規劃，在數個基準上以少得多的樣本追平強大無模型方法的最終表現 [29]。

程式碼：規劃器利用學到的模型

代理人只有動作落在 $[-1, 1]$ 的記錄資料，但規劃器可以選擇 $[-3, 3]$ 之間的任何動作。

```
import numpy as np
from sklearn.neural_network import MLPRegressor

rng = np.random.default_rng(0)
true_reward = lambda a: np.sin(3 * a) * np.exp(-0.3 * a ** 2)   # unknown to the agent

# Logged interactions only cover actions in [-1, 1].
a_data = rng.uniform(-1, 1, 200)
r_data = true_reward(a_data) + 0.05 * rng.normal(size=200)

ensemble = []
for k in range(5):                       # bootstrap + different initialization
    idx = rng.integers(0, 200, 200)
    m = MLPRegressor(hidden_layer_sizes=(64, 64), max_iter=3000, random_state=k)
    ensemble.append(m.fit(a_data[idx, None], r_data[idx]))

grid = np.linspace(-3, 3, 601)
preds = np.stack([m.predict(grid[:, None]) for m in ensemble])
mean, std = preds.mean(0), preds.std(0)
```

```
for name, score in [("single model", preds[0]),
                    ("ensemble mean", mean),
                    ("pessimistic: mean - 2 std", mean - 2 * std)]:
    a_star = grid[np.argmax(score)]       # the "planner": pick the best-looking action
    print(f"{name:26s} picks a = {a_star:+.2f}, predicted {score.max():.2f}, "
          f"true {true_reward(a_star):.2f}")
print(f"mean std inside data {std[np.abs(grid) <= 1].mean():.3f}, "
      f"outside {std[np.abs(grid) > 1.5].mean():.3f}")
```

輸出：

```
single model               picks a = -3.00, predicted 1.79, true -0.03
ensemble mean              picks a = -3.00, predicted 1.58, true -0.03
pessimistic: mean - 2 std  picks a = +0.43, predicted 0.87, true 0.91
mean std inside data 0.025, outside 0.321
```

單一模型與集成平均都在資料範圍外外插出一條上升直線，規劃器於是衝到動作空間的邊緣，而那裡真實獎勵約為零。集成在資料外的分歧是資料內的十三倍；**悲觀**分數（平均減兩倍標準差）讓規劃器留在資料範圍內，找到真實獎勵為 0.91 的動作。限制動作、縮短視野、懲罰分歧以及用真實回饋測試，是實務上的對策。不過，集成成員意見一致並不保證正確：五個模型可能共享同樣的偏差。

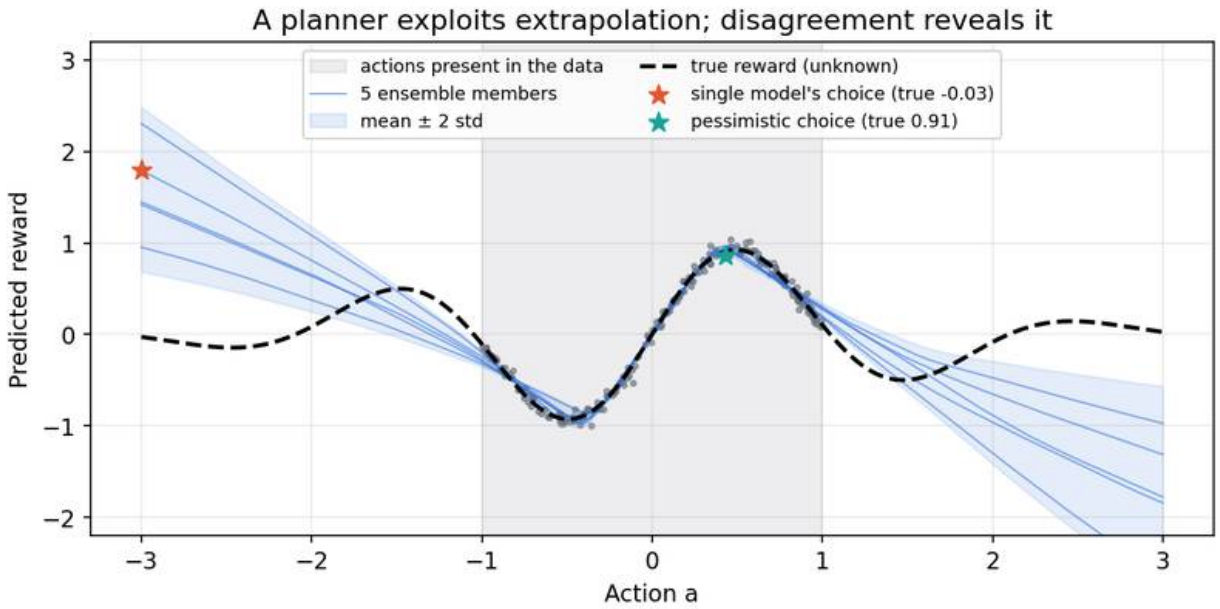


圖 *6* — 上方程式碼的輸出。對學到的模型做最佳化，會把規劃器推向模型最不可靠的地方；集成的分散程度能揭露這些區域。

如何評估世界模型

白話版。沒有任何單一數字能說明世界模型好不好。要問好幾個問題，每個問題各有自己的量測方式與控制條件。

問題	量測	重要的控制條件
能預測已知動態嗎？	單步與依視野索引的狀態或特徵誤差	獨立回合；固定動作序列
外觀能保持一致嗎？	重建與影片品質指標、身分一致性	比較相機運動與遮擋
動作的作用正確嗎？	相同初始上下文、指定不同動作	真實的以動作為條件的參考軌跡
遵守物理限制嗎？	碰撞、物體恆存、守恆；Physics-IQ 等基準 [22]	任務專屬容許誤差；分布外案例 [23]
對決策有幫助嗎？	成功率、回報、人工介入次數、控制延遲	在真實或可信的環境中測試

像素保真度與決策用處可能不一致，模糊的例子已經說明了這點。平均出來的預測可能平方誤差很低，卻呈現一個不可能的場景；準確的潛在預測器可能省略視覺上無關的紋理。每個指標都要一起報告預測目標、視野、條件與輸出解析度。

比較時，請納入持續性基準（預測「什麼都不變」）、無動作預測器，以及一個適合任務的簡單動態模型。分別評估保留的物體、視角、動作範圍與環境性質。除了想像中的結果，也要報告閉迴路控制結果，並計入編碼器與取樣器的延遲：每個動作要花十秒規劃的規劃器，對以 10 Hz 運作的機器人可能毫無用處。

現代觀點

白話版。現在有三條研究路線都被稱為「世界模型」：模型式強化學習、被重新定位為模擬器的影片生成器，以及預測式表示。尚未解決的問題，大多在於它們的預測是否可靠到足以據此行動。

綜述。 Moerland 等人對模型式 RL 提供了最系統化的整理：動態模型學習（隨機性、不確定性、部分可觀測性、時間抽象）、規劃與學習整合方式的分類，以及隱式（價值等價、MuZero 式）方法 [5]。Ding 等人（ACM Computing Surveys）依功能整理更廣泛的「世界模型」文獻，也就是理解世界與預測未來兩類，並回顧在生成式遊戲、自動駕駛、機器人與社會模擬上的應用 [1]。Zhu 等人從影片生成的角度綜述「通用世界模型」，涵蓋影片生成器、自動駕駛世界模型以及自主代理人中的世界模型，並討論其限制 [31]。Guan 等人則提供了專門針對自動駕駛世界模型的初步綜述 [32]。

2023–2026 年的現況。

- **模型式 RL 趨於成熟**。 DreamerV3 展示了單一設定在超過 150 個任務上都能運作，並於 2025 年發表在 Nature [10]；TD-MPC2 展示了無解碼器的潛在規劃能擴展到 317M 參數的多任務代理人 [11]；IRIS 與 DIAMOND 顯示 Transformer 與擴散世界模型在 Atari 100k 基準上具競爭力 [12], [13]。
- **影片生成器被重新定位為模擬器**。 GAIA-1、UniSim、Genie、Genie 3 與 Cosmos 在帶有動作、文字或推斷出之潛在動作的影片上訓練大型模型 [17], [18], [19], [20], [21]。它們能幫助代理人的最強證據，是 UniSim 把在模擬器中訓練的策略零樣本轉移到真實世界 [18]；最強的警告則是：真實感不代表懂物理 [22], [23]。

- **預測式表示走向控制**。JEPA 系列從影像（I-JEPA）走到影片（V-JEPA），再到真實機器人上以動作為條件的規劃（V-JEPA 2-AC）與更密集的特徵（V-JEPA 2.1）[24], [25], [26], [27]；DINO-WM 顯示凍結的自監督特徵已能支援零樣本規劃 [28]。

開放問題。

- **長視野一致性**。誤差會累積；生成的世界會漂移；物體身分與畫面外的狀態很難維持數分鐘，更別說數小時 [20]。
- **因果與物理正確性**。在被動影片上訓練的模型學到的是相關性；分布外的物理會失敗 [22], [23]。介入式（動作）資料稀少又昂貴。
- **動作介面**。從影片推斷的潛在動作、機器人關節指令與文字指令是不同的東西；可靠地在它們之間轉換仍未解決。
- **被利用與不確定性**。在想像中訓練的規劃器與策略會利用模型錯誤；大型影片模型大多缺乏校準過的認知不確定性。
- **評估**。目前沒有公認的基準能跨領域把預測品質與後續決策品質連結起來；多數論文只報告影片指標或任務回報其中之一，很少兩者都有。
- **成本**。對大型影片模型取樣可能遠慢於即時，這與閉迴路控制相衝突。

重點整理

- 預測目標與動作介面決定世界模型能支援什麼。
- 局部觀測需要記憶或信念估計，一張影像不是完整狀態。
- 規劃、想像中的策略訓練與價值等價搜尋，以不同方式使用學得動態。
- 像素逼真不代表物理正確；特徵預測也需要有意義目標與避免坍縮的機制。
- 長展開會累積誤差，最佳化可能利用這些誤差；須檢查不確定性並重新規劃。
- 評估保留資料預測、受控動作效果，以及真實環境中的行為。

練習

1. **動作消融**。在 CPU 實驗中拿掉動作欄位。比較單步誤差、視野 40 的誤差與控制行為。為什麼一個不看動作的短期預測器仍可能看起來還不錯？

   提示

   想像的動作不再影響無動作預測器，所以規劃器完全無法比較它們的後果：每個候選都得到相同的預測成本。在一步之內，有阻尼的動態只讓狀態改變一點點，因此類似「保持不變」的預測仍能得到中等分數，這也是為什麼只看單步誤差是很弱的測試。

2. **部分觀測**。隱藏速度，擬合「位置到位置」的動態。然後改用最後兩個位置與前一個動作作為特徵。這段歷史何時有幫助？觀測雜訊在哪裡造成傷害？

   提示

   兩個位置的差可以估計速度（有限差分），所以雙位置模型能恢復大部分狀態。每個位置上的雜訊在除以時間步長時會被放大，因此觀測雜訊越大，好處越小。

3. **規劃視野**。比較視野 5、20 與 40，並把初始平均調整成對應長度。在多個獨立種子上量測最終距離、動作力度與執行時間。

   提示

   保持成本定義不變，並計算所有模型呼叫次數。太短的視野「看不到」提早煞車可以避免衝過頭；太長的視野成本更高，也累積更多模型誤差，不過在這個線性系統中誤差很小。

4. **領域偏移**。以速度阻尼 0.94 收集訓練回合，然後在阻尼為 0.75 的環境中執行控制。比較開迴路執行一個計畫與每一步重新規劃。找出一個訓練資料涵蓋不足的動作範圍。

   提示

   把學到的動態（以 0.94 擬合）與真實執行的動態（0.75）分開；不要在評估轉移上重新擬合。重新規劃使用新的觀測，所以能部分修正有偏差的模型；開迴路執行則做不到。

5. **平均與樣本**。把模糊的例子延伸成三個同樣可能的未來，位置在 10、32 與 54。計算平均預測與單一清晰樣本的期望 MSE。再提出一個會偏好樣本的指標。

   提示

   平均現在有三個三分之一高度的峰；期望 MSE 約為 0.037（平均）與 0.074（樣本），又是兩倍。這個倍數是普遍的：若 X 與 Y 是同一分布的獨立抽樣，則 $\mathbb{E}\|X-Y\|^2 = 2\,\mathbb{E}\|X-\mathbb{E}X\|^2$，所以完美的取樣器的 MSE 永遠是平均的兩倍。分布層級的指標（例如比較生成樣本集合與真實未來集合，就像 FID 類指標對影像所做的）或多個樣本中取最佳的 best-of-N 誤差，會獎勵清晰且合理的預測。

6. **悲觀的取捨**。在集成程式碼中，把「mean − 2 std」的係數 2 換成 0、0.5、1 與 4。畫出每種情況下選到的動作與其真實獎勵。係數太大時會出什麼問題？

   提示

   係數為 0 時規劃器會利用外插；係數增大時它會留在資料範圍內。過度悲觀會讓規劃器避開任何有些微分歧的動作，可能錯過資料範圍內的最佳動作，而在 RL 情境中還會停止探索。

參考文獻

1. J. Ding et al., “Understanding world or predicting future? A comprehensive survey of world models,” *ACM Computing Surveys*, vol. 58, no. 3, 2025. doi arXiv
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, ch. 13 “Sequential Data”. book
3. R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., Springer, 2022. book
4. R. S. Sutton, “Dyna, an integrated architecture for learning, planning, and reacting,” *ACM SIGART Bulletin*, vol. 2, no. 4, 1991. doi
5. T. M. Moerland, J. Broekens, A. Plaat and C. M. Jonker, “Model-based reinforcement learning: a survey,” *Foundations and Trends in Machine Learning*, vol. 16, no. 1, 2023. doi arXiv

6. D. Ha and J. Schmidhuber, “Recurrent world models facilitate policy evolution,” NeurIPS, 2018; interactive version “World Models,” arXiv:1803.10122. paper interactive
7. D. Hafner et al., “Learning latent dynamics for planning from pixels,” ICML, 2019. paper
8. D. Hafner, T. Lillicrap, J. Ba and M. Norouzi, “Dream to control: learning behaviors by latent imagination,” ICLR, 2020. arXiv
9. D. Hafner, T. Lillicrap, M. Norouzi and J. Ba, “Mastering Atari with discrete world models,” ICLR, 2021. arXiv
10. D. Hafner, J. Pasukonis, J. Ba and T. Lillicrap, “Mastering diverse control tasks through world models,” *Nature*, vol. 640, 2025. doi arXiv
11. N. Hansen, H. Su and X. Wang, “TD-MPC2: scalable, robust world models for continuous control,” ICLR, 2024. arXiv
12. V. Micheli, E. Alonso and F. Fleuret, “Transformers are sample-efficient world models,” ICLR, 2023. arXiv
13. E. Alonso et al., “Diffusion for world modeling: visual details matter in Atari,” NeurIPS, 2024. arXiv
14. J. Schrittwieser et al., “Mastering Atari, Go, chess and shogi by planning with a learned model,” *Nature*, vol. 588, 2020. doi arXiv
15. C. Finn, I. Goodfellow and S. Levine, “Unsupervised learning for physical interaction through video prediction,” NIPS, 2016. arXiv
16. M. Babaeizadeh, C. Finn, D. Erhan, R. H. Campbell and S. Levine, “Stochastic variational video prediction,” arXiv:1710.11252, 2017. arXiv
17. A. Hu et al., “GAIA-1: a generative world model for autonomous driving,” technical report, arXiv:2309.17080, 2023. arXiv
18. S. Yang et al., “Learning interactive real-world simulators,” arXiv:2310.06114, 2023. arXiv
19. J. Bruce et al., “Genie: generative interactive environments,” arXiv:2402.15391, 2024. arXiv
20. Google DeepMind, “Genie 3: a new frontier for world models,” blog post, August 5, 2025. blog
21. NVIDIA (N. Agarwal et al.), “Cosmos world foundation model platform for physical AI,” arXiv:2501.03575, 2025. arXiv
22. S. Motamed, L. Culp, K. Swersky, P. Jaini and R. Geirhos, “Do generative video models understand physical principles?,” arXiv:2501.09038, 2025. arXiv
23. B. Kang et al., “How far is video generation from world model: a physical law perspective,” ICML, 2025. arXiv
24. M. Assran et al., “Self-supervised learning from images with a joint-embedding predictive architecture,” CVPR, 2023. doi arXiv
25. A. Bardes et al., “Revisiting feature prediction for learning visual representations from video,” arXiv:2404.08471, 2024. arXiv
26. M. Assran et al., “V-JEPA 2: self-supervised video models enable understanding, prediction and planning,” arXiv:2506.09985, 2025. arXiv
27. L. Mur-Labadia et al., “V-JEPA 2.1: unlocking dense features in video self-supervised learning,” arXiv:2603.14482, 2026. arXiv
28. G. Zhou, H. Pan, Y. LeCun and L. Pinto, “DINO-WM: world models on pre-trained visual features enable zero-shot planning,” arXiv:2411.04983, 2024. arXiv
29. K. Chua, R. Calandra, R. McAllister and S. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” NeurIPS, 2018. arXiv

30. M. Janner, J. Fu, M. Zhang and S. Levine, “When to trust your model: model-based policy optimization,” NeurIPS, 2019. arXiv
31. Z. Zhu et al., “Is Sora a world simulator? A comprehensive survey on general world models and beyond,” arXiv:2405.03520, 2024. arXiv
32. Y. Guan et al., “World models for autonomous driving: an initial survey,” arXiv:2403.02622, 2024. arXiv
33. D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” *International Conference on Learning Representations* (*ICLR*), 2014; arXiv:1312.6114. arXiv
34. D. J. Rezende, S. Mohamed and D. Wierstra, “Stochastic backpropagation and approximate inference in deep generative models,” ICML, 2014. paper
35. R. Y. Rubinstein, “The cross-entropy method for combinatorial and continuous optimization,” *Methodology and Computing in Applied Probability*, vol. 1, no. 2, 1999. doi
36. P.-T. de Boer, D. P. Kroese, S. Mannor and R. Y. Rubinstein, “A tutorial on the cross-entropy method,” *Annals of Operations Research*, vol. 134, no. 1, 2005. doi

DLCV 14・穩健性：分布偏移、對抗樣本與異常

視覺模型為什麼在測試集之外會失效，以及該怎麼辦：分布偏移的種類與衡量它的基準、資料擴增、預訓練、測試時適應與領域適應；對抗威脅模型、FGSM、PGD、對抗訓練、誠實的評估與認證穩健性；分布外偵測與工業異常偵測的分數、指標與陷阱。

網站版本: https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/

你將學到

- 區分分布轉移、對抗擾動與不熟悉輸入。
- 讓轉移處理方式符合其資料假設。
- 指定威脅模型、實作攻擊並解讀穩健性憑證。
- 比較輸出式及特徵式 OOD 分數，以及區塊式異常偵測。
- 檢查資料污染、測試集調參與誤導性的總體指標。

先看全貌

一個在測試集上拿到 98% 的模型，只被檢查過「長得像訓練資料」的資料。部署後的模型會遇到別的資料：新相機拍的照片、被攻擊者修改過的影像，以及從來不在標籤集裡的物體。穩健性研究的就是這時候會發生什麼，以及怎麼讓答案不那麼令人擔心。

> **用最簡單的話說**
>
> 想像一位只讀過一本課本的學生。考試時有三件事可能出錯。第一，考卷印得模糊、褪色：題目還是同一類，但看起來不一樣（**分布偏移**）。第二，有個惡作劇的人把題目裡的幾個字改掉，讓學生讀錯，而且專挑最會讓這位學生搞混的字（**對抗樣本**）。第三，考卷上冒出一題完全不同科目的題目：正確答案是「這不是我的科目」，學生卻還是信心滿滿地寫了答案（**分布外輸入**）。好學生能應付模糊的字跡、不容易被騙，而且知道什麼時候該說「我不知道」。

為什麼重要。這三種失效造成的傷害不同。偏移會悄悄拉低現場的準確率，所以一個看起來已經完成的模型，在上線後會持續犯更多錯。對抗樣本是安全問題：任何決策值得被操縱的系統（內容過濾、身分檢查、交通標誌辨識）都會被攻擊。無法回答的輸入會產生信心十足的胡說，這是安全關鍵系統所能給出的最危險的輸出。解法也不一樣，所以第一項實務技能，就是說得出你面對的是哪一種問題。

本篇的位置。 DLCV 02 已經介紹過 ImageNet-C、-R 與 -A、一步式的 FGSM 攻擊、捷徑學習，以及用溫度縮放做校準；這裡在它們的基礎上往下談，不重複。ML 11 用重要性權重定義了共變量、標籤與概念偏移，也談到部署後的監控。PR 01 推導了拒絕選項，PR 10 則在一個小型手寫數字專案上，區分選擇性分類、開放集合辨識與異常偵測。ML 01 與 ML 06A 談了以重建為基礎的異常分數，ML 10 則解釋為什麼微調會侵蝕基礎模型的穩健性。本篇所有程式碼用 NumPy、

SciPy、scikit-image 與 scikit-learn 就能在 CPU 上幾秒內跑完；其中的「網路」是 8×8 手寫數字上的小型多層感知器，所以我們印出的每個數字都來自玩具實驗，並且會標明。

三種穩健性問題，一張地圖

白話版。先診斷失敗來自資料改變、刻意擾動，或任務外輸入；三者需要不同評估。

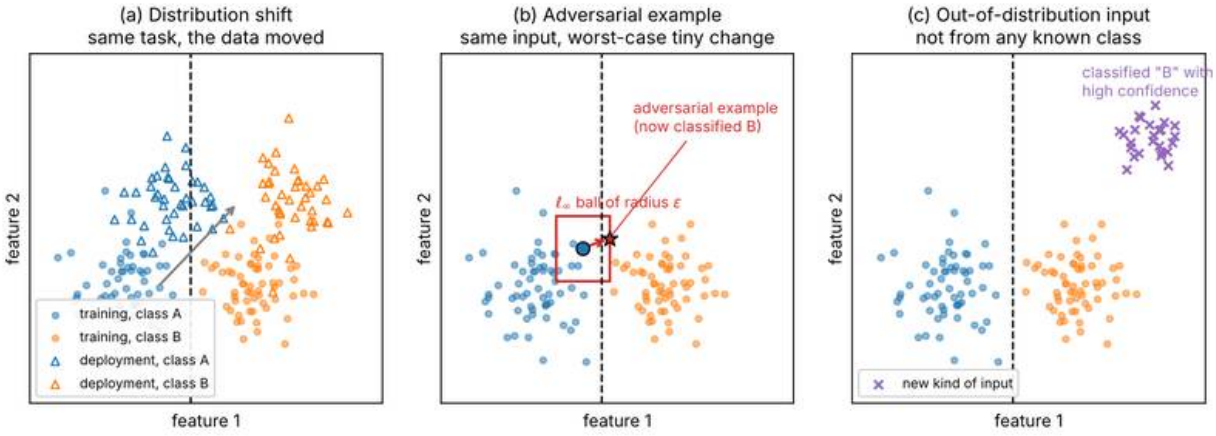


圖 *1* — 在同一個合成的二類問題上看三種穩健性問題。(*a*) 分布偏移：部署資料（三角形）離開了訓練資料，原本對訓練資料沒問題的邊界現在會出錯。(*b*) 對抗樣本：在正確輸入周圍的小方框內，最壞情況的點越過了邊界。(*c*) 分布外輸入：訓練時從未見過的那類輸入，仍然得到信心很高的類別標籤。

精確版。設分類器 f 用取自分布 $p_{\text{train}}(\mathbf{x}, y)$ 的樣本訓練，其中 $\mathbf{x}$ 是影像，標籤 $y \in \mathcal{Y} = \{1, \dots, K\}$。用 $\text{acc}_p(f) = \Pr_{(\mathbf{x},y)\sim p}[f(\mathbf{x}) = y]$ 表示它在分布 p 上的準確率。

1. **分布偏移**。測試資料來自 $p_{\text{test}} \neq p_{\text{train}}$，但標籤集 $\mathcal{Y}$ 相同。問題是 $\text{acc}_{p_{\text{test}}}(f)$ 比 $\text{acc}_{p_{\text{train}}}(f)$ 低多少，以及怎麼減少這個落差。
2. **對抗穩健性**。對每個測試配對 $(\mathbf{x}, y)$，對手可以把 $\mathbf{x}$ 換成允許集合 $\mathcal{S}(\mathbf{x})$ 中的任何 $\mathbf{x}'$，這個集合通常是 $\mathbf{x}$ 周圍的小球。若對 $\mathcal{S}(\mathbf{x})$ 中的**每一個** $\mathbf{x}'$ 都有 $f(\mathbf{x}') = y$，模型在 $(\mathbf{x}, y)$ 上就是穩健的。問題問的是最壞情況，不是平均。
3. **OOD 與異常偵測**。有些測試輸入在 $\mathcal{Y}$ 中沒有正確標籤（新的物體類別），或來自訓練資料從未涵蓋的輸入空間區域（有瑕疵的零件、損毀的影格）。我們想要一個分數 $s(\mathbf{x})$：分布內（in-distribution, ID）的輸入分數高，這些輸入分數低，讓系統可以拒答或發出警報。

	分布偏移	對抗樣本	OOD / 異常
什麼變了	資料分布	某一個輸入，且是刻意的	輸入在任務之外
誰造成的	外在世界（相機、地點、時間）	帶著目標的對手	外在世界（新類別、瑕疵）
我們要什麼	維持準確率	在每一種允許的改動下標籤都不變	標記該輸入，不要作答
典型指標	偏移後測試集上的準確率	預算 ε 下的穩健準確率	AUROC、95% TPR 下的 FPR

	分布偏移	對抗樣本	OOD / 異常
本篇位置	分布偏移	對抗樣本	分布外與異常偵測

三者在邊緣會重疊。嚴重的損壞可以讓輸入實質上變成 OOD；對抗樣本是由對手挑選的偏移；ImageNet-A 是一組因為會騙倒分類器而被挑出來的自然照片 [1]，介於偏移與對抗樣本之間。不過這張表說明了為什麼單一個穩健性數字永遠無法涵蓋三者：它們問的問題不同，需要的測試資料也不同。

分布偏移

白話版。熟悉任務也可能來自新相機、光線、風格或地點；轉移時準確率與信心未必一起變化。

偏移的種類

常用的詞彙用兩種方式拆解聯合分布，$p(\mathbf{x}, y) = p(y \mid \mathbf{x})\, p(\mathbf{x}) = p(\mathbf{x} \mid y)\, p(y)$，並依哪個因子改變來為偏移命名：

- **共變量偏移（covariate shift）：** $p(\mathbf{x})$ 改變，$p(y \mid \mathbf{x})$ 不變。新的掃描儀、起霧、另一家醫院的染色方式。
- **標籤（先驗）偏移（label shift）：** $p(y)$ 改變，$p(\mathbf{x} \mid y)$ 不變。疫情期間某種罕見疾病變得常見。
- **概念偏移（concept shift）：** $p(y \mid \mathbf{x})$ 本身改變。某個類別的定義被修訂。

ML 11 用重要性權重與標籤偏移校正推導了這幾種偏移，這裡不重複。視覺領域還有兩個常用的詞。**領域偏移**（domain shift）是指名稱明確的資料來源之間的共變量偏移（合成對真實、這家醫院對那家醫院）；**子群體偏移**（subpopulation shift）則保持相同種類的影像，但改變各個群體出現的頻率；WILDS 兩者都有 [2]。Yang 等人的綜述加了一個區分，後面會變得很重要：**共變量偏移**改變輸入的外觀而標籤仍然有效，**語意偏移**（semantic shift）則帶來標籤落在 $\mathcal{Y}$ 之外的輸入 [3]。我們希望模型對共變量偏移保持穩健，並偵測出語意偏移。

衡量偏移穩健性的基準

白話版。真正的部署資料很少能事先拿到，所以研究者建立了一次只改變一件事的測試集，或是收集同樣類別但自然地不太一樣的照片。

主要的幾個家族，除非另外說明，都以 ImageNet 類別標註：

- **合成損壞。** **ImageNet-C** 以五種嚴重程度加上雜訊、模糊、天氣與數位損壞，作者也提出了用於擾動序列的 ImageNet-P [4]。細節與平均損壞錯誤率在 DLCV 02。
- **風格化表現。** **ImageNet-R** 收集藝術品、卡通、雕塑與其他表現形式；同一篇論文還提出其他真實世界的偏移資料集，並發現對紋理與局部影像統計有幫助的方法，對其他偏移（例如地理變化）沒有幫助 [5]。
- **自然對抗影像。** **ImageNet-A** 保留未經修改、但被 ResNet-50 判錯的照片；它們能轉移到其他模型，顯示模型有共同的弱點 [1]。

- **受控收集。** **ObjectNet** 以隨機的背景、旋轉與視角拍攝物體，只當測試集，沒有訓練切分 [6]。
- **重新收集。** **ImageNetV2** 依照原本的收集流程重建測試集；即使盡可能照抄流程，準確率還是下降了 [7]（見 ML 11）。
- **真實環境。** **WILDS** 收集十個來自真實應用的偏移資料集，例如不同醫院的組織切片與新地點的野外相機陷阱，並報告標準訓練在分布外的表現遠低於分布內，而現有的穩健性方法也沒有縮小這個差距 [2]。

怎麼讀這些數字。乾淨準確率較高的模型，偏移後的準確率通常也較高，所以「它在 ImageNet-R 上分數更高」可能只是在說「它是個更好的模型」。Taori 等人在 213 種測試條件下評估了 204 個 ImageNet 模型，並提出**有效穩健性**（effective robustness）：偏移測試集上的準確率，超出由標準模型擬合出的趨勢線根據乾淨準確率所預測的部分 [8]。Miller 等人發現，對許多偏移而言，不同架構與訓練選擇下的偏移準確率與乾淨準確率落在接近一條直線上（「線上的準確率」，accuracy on the line），只有某些合成偏移與一個醫療資料集的相關性較弱 [9]。實務上的結論：比較穩健性介入手段時，要在相同的乾淨準確率下比，或同時列出兩個數字。

僅線上版：有效穩健性與線上的準確率 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/notes/有效穩健性與線上的準確率/*

程式碼：損壞、資料擴增與測試批次正規化

這個玩具實驗在 scikit-learn 的 8×8 手寫數字上訓練一個小型 MLP，並用三種損壞、嚴重程度 1 到 5 去損壞測試影像。網路用訓練集算出的統計量把每個像素標準化，做法就像批次正規化層在測試時使用的滑動統計量。我們比較三個模型：一般模型；用雜訊擴增訓練的模型（每張訓練影像多一份隨機嚴重程度的雜訊副本）；以及帶有**測試批次正規化**（test-batch normalization）的一般模型，它在損壞後的測試批次上重新計算每個像素的平均與標準差，而不是使用訓練統計量。

```
import numpy as np
from scipy.ndimage import gaussian_filter
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

X, y = load_digits(return_X_y=True)
X = X / 16.0                                          # 8x8 images, pixels in [0, 1]
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
rng = np.random.default_rng(0)

corruptions = {                                       # severity s = 1..5
    "noise":    lambda X, s: np.clip(X + rng.normal(0, 0.1 * s, X.shape), 0, 1),
    "blur":     lambda X, s: gaussian_filter(X.reshape(-1, 8, 8), (0, 0.25 * s, 0.25 *
    ↪  s)).reshape(-1, 64),
    "contrast": lambda X, s: 0.5 + (X - 0.5) * (1 - 0.15 * s),
}

def norm(X, ref):                                     # standardize each pixel with ref's
↪  statistics
    return (X - ref.mean(0)) / (ref.std(0) + 0.05)

def fit(X, y):
```

```
    return MLPClassifier(hidden_layer_sizes=(64,), alpha=1e-3, max_iter=400,
     ↪  random_state=0).fit(norm(X, Xtr), y)

plain = fit(Xtr, ytr)
s_rand = rng.integers(1, 6, (len(Xtr), 1))            # random noise severity per training
 ↪  image
augmented = fit(np.vstack([Xtr, corruptions["noise"](Xtr, s_rand)]), np.r_[ytr, ytr])

for name, corrupt in corruptions.items():
    rows = {"plain": [], "noise-aug": [], "test-norm": []}
    for s in range(6):
        Xs = Xte if s == 0 else corrupt(Xte, s)
        rows["plain"].append(plain.score(norm(Xs, Xtr), yte))
        rows["noise-aug"].append(augmented.score(norm(Xs, Xtr), yte))
        rows["test-norm"].append(plain.score(norm(Xs, Xs), yte))   # statistics of the test
         ↪  batch
    for k, v in rows.items():
        print(f"{name:8s} {k:9s}", " ".join(f"{a:.2f}" for a in v))
```

它印出嚴重程度 0（乾淨）到 5 的測試準確率：

```
noise    plain     0.98 0.94 0.75 0.54 0.41 0.31
noise    noise-aug 0.97 0.96 0.93 0.82 0.74 0.62
noise    test-norm 0.98 0.97 0.90 0.83 0.73 0.61
blur     plain     0.98 0.98 0.96 0.82 0.46 0.21
blur     noise-aug 0.97 0.97 0.97 0.90 0.78 0.58
blur     test-norm 0.98 0.98 0.97 0.95 0.89 0.81
contrast plain     0.98 0.92 0.46 0.20 0.19 0.19
contrast noise-aug 0.97 0.98 0.96 0.93 0.84 0.60
contrast test-norm 0.98 0.98 0.98 0.98 0.98 0.98
```

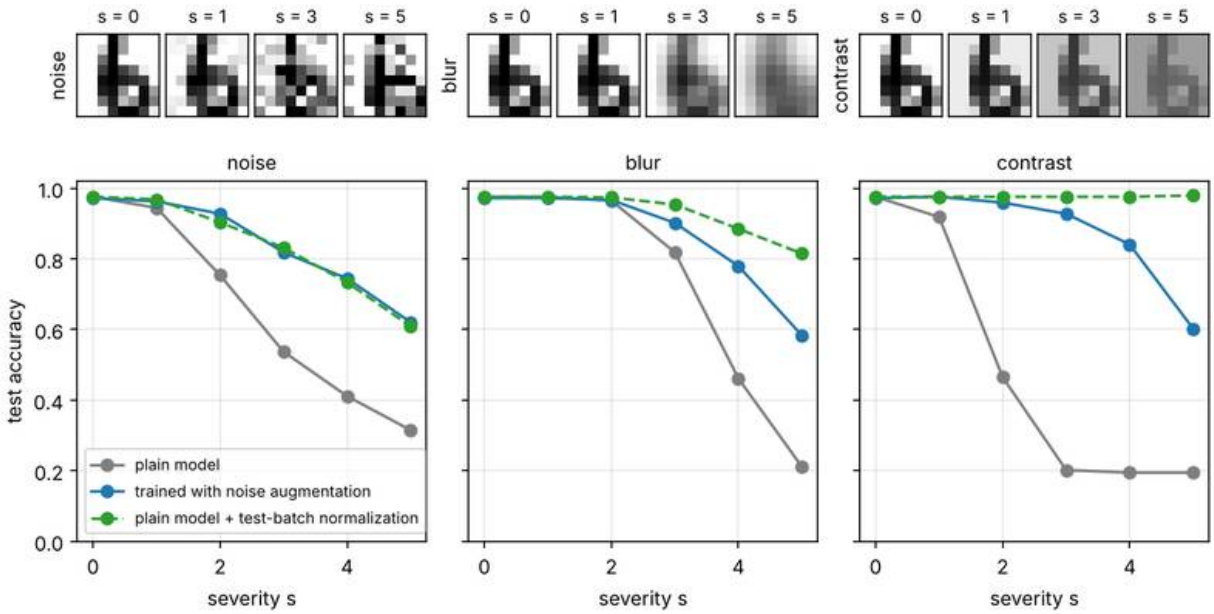


圖 2 — 小型 *MLP* 在損壞的手寫數字上的準確率（我們的玩具實驗）。上：每種損壞與嚴重程度各取一張測試影像。下：一般模型、用雜訊擴增訓練的模型，以及在損壞的測試批次上重新計算每像素統計量的一般模型。

（所有數字都來自我們的玩具實驗。）有四個教訓可以帶到真實網路上：

- **看起來溫和的改變就足夠了**。把對比降到原本範圍的 70%（嚴重程度 2），就讓一般模型的準確率減半，即使人可以輕鬆讀出這些數字。
- **資料擴增有幫助，而且不只對它用過的那種損壞**。用雜訊訓練，在這裡對模糊與對比也有幫助，這正是多樣化擴增所寄望的。它犧牲了一點乾淨準確率（0.98 變成 0.97），而真實基準顯示，轉移到其他偏移遠遠不能保證 [5], [8]。
- **在假設成立時，重新估計正規化統計量是便宜又有效的修正**。對比損壞是對每個像素的仿射變換，逐像素標準化可以完全還原它，所以準確率維持在 0.98。雜訊改變統計量的方式是標準化還原不了的，所以收益較小。
- **測試批次正規化需要一個批次**。它一次使用了同一種損壞的全部 540 張測試影像。如果只有幾張影像，或批次混合了好幾種條件，統計量就會是錯的（練習 1）。

什麼有幫助，什麼沒有

白話版。處理方式包括多樣訓練、廣泛預訓練、測試時調適、目標領域調適與多領域訓練；應依可用資料及部署限制選擇。

資料擴增。資料擴增讓訓練分布變寬，使測試分布更有可能落在其中。**AugMix** 把同一張影像的幾條隨機擴增鏈混合起來，並加上 Jensen–Shannon 一致性損失，要求網路對原圖與擴增版本給出相近的預測 [10]。作者刻意移除了與 ImageNet-C 損壞重疊的運算，讓這個基準衡量的是對沒見過的損壞的轉移能力，而不是對測試損壞的背誦 [10]。風格化訓練把影像紋理換成繪畫風格，讓 ResNet-50 更依賴形狀，並改善了對多種失真的穩健性 [11]。要小心的是 ImageNet-R 那項研究：擴增與更大的模型在某些真實偏移（風格、模糊）上有幫助，在另一些（地理）上沒有，而且沒有任何受評估的方法能在所有偏移上一致地改善穩健性 [5]。通用的擴增配方在 DLCV 02。

在更多樣的資料上預訓練。在 Taori 等人的測試平台中，大多數穩健性介入手段對自然偏移幾乎沒有帶來有效穩健性；主要的例外是用更大、更多樣的資料訓練 [8]。以網路上的影像–文字配對訓練的零樣本 CLIP 模型，在自然的 ImageNet 偏移上大幅提高了有效穩健性，但 CLIP 論文也指出，一旦用完全監督的方式在 ImageNet 上擬合分類器，這個好處幾乎消失 [12]。Fang 等人檢驗了五個可能的原因（資料集大小、訓練分布、訓練時的語言監督、測試時的語言監督、對比損失），發現**更多樣的訓練分布**是主因 [13]。所以微調會侵蝕穩健性；WiSE-FT 把零樣本模型與微調後模型的權重取平均，保留兩者的大部分優點 [14]（另見 ML 10 的 LP-FT）。

測試時適應（test-time adaptation, TTA）。如果未標註的測試影像是一批一批到達，模型可以在預測前先調整自己。最簡單的形式就是我們玩具實驗中的做法：Schneider 等人把 25 個預訓練模型的批次正規化統計量換成損壞測試影像的統計量，一致地改善了對損壞的穩健性，甚至只用單一樣本來適應也有效 [15]。**TENT** 再往前一步：它重新估計正規化統計量，並且在每個測試批次上，以最小化模型自身預測的熵，線上更新正規化層中逐通道的仿射參數 [16]。TTA 假設測試資料是一串偏移相當穩定的資料流；它在測試時增加運算量，如果資料流改變，還可能漂移。

僅線上版：測試時的熵最小化 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/notes/測試時的熵最小化/*

非監督式領域適應（unsupervised domain adaptation, UDA）。如果訓練期間有目標領域的未標註影像，就可以訓練模型在沒有目標標籤的情況下在那裡運作。**領域對抗訓練（DANN）**在特徵上加一個小型領域分類器，兩者之間放一個**梯度反轉層**（gradient-reversal layer），使特徵既對有標註來源領域上的標籤任務有用，同時又無法分辨來源與目標 [17]。這個想法根據領域適應理論：如果分類器在特徵空間中分不出兩個領域，好的來源分類器就應該能轉移過去。

僅線上版：領域對抗訓練與梯度反轉層 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/notes/領域對抗訓練與梯度反轉層/*

領域泛化（domain generalization, DG）。沒有任何目標資料時，在多個來源領域上訓練模型，期望它能在沒見過的領域上運作。Zhou 等人的綜述把 DG 方法分成領域對齊、元學習、資料擴增與集成學習等類別 [18]。Gulrajani 與 Lopez-Paz 接著在同一個測試平台 DomainBed 中，以相同的調參力氣與明確的模型選擇規則，重跑了許多 DG 演算法，發現搭配強網路與資料擴增的單純經驗風險最小化（ERM，也就是在合併後的來源資料上做一般訓練）表現與它們相當甚至更好 [19]。他們的重點在方法論：一個 DG 方法，如果沒有一套不偷看測試領域的超參數選擇規則，就是不完整的。

僅線上版：*Gulrajani 與 Lopez-Paz：In Search of Lost Domain Generalization（ICLR 2021）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/papers/gulrajani-與-lopez-pazin-search-of-lost-domain-generalizationiclr-2021/*

沒有幫助，或不可靠的做法。在 Taori 等人的測試平台中，對合成損壞的穩健性只能微弱地預測對自然偏移的穩健性，ℓ_p 對抗穩健性也是如此 [8]。在某個基準上調好的方法，常常只對那個基準有幫助 [5]。在固定資料下更換架構，影響小於資料本身 [8], [13]。實務規則由此而來：要在好幾種偏移上測試，其中至少包含一種來自你自己部署環境的真實偏移。

偏移下的信心

白話版。在偏移下，模型犯錯變多，但它通常不會等比例地變得沒有信心，所以它的機率不再代表它說的意思。

Ovadia 等人在好幾個任務上對資料集偏移下的不確定性方法做了基準測試，發現偏移越大校準越差，在分布內驗證資料上擬合的事後校準（例如溫度縮放）在偏移下不夠用，而對多個模型取平均的方法（例如深度集成）表現最穩 [20]。溫度縮放在 DLCV 02 中解釋過，ML 11 則用玩具實驗示範了擬合好的溫度如何在偏移下失靈。實務上的結論：在乾淨驗證資料上選好的信心門檻，到了現場會接受更多錯誤答案，所以部署後要監控信心的分布（ML 11）。

對抗樣本

白話版。對幾乎任何一張被正確分類的影像，附近都有另一張影像（人眼常常分辨不出來）會被網路分類錯誤。攻擊者的做法，是直接問網路：每個像素該往哪個方向移動。

什麼是對抗樣本

Szegedy 等人報告，只要加上一個經由最大化網路預測誤差而找到、人眼無法察覺的擾動，就能讓深度網路誤分類一張影像，而且同一個擾動往往也能騙過用另一個資料子集訓練的不同網路 [21]。Goodfellow 等人用網路在高維度下的近似線性來解釋這個現象，並提出一步式的快速梯度符號法（fast gradient sign method）[22]，DLCV 02 已推導並測試過。本節補上評估與改善穩健性所需的部分：威脅模型、多步攻擊、對抗訓練、誠實的評估，以及認證。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

威脅模型

白話版。在說清楚「對什麼穩健」之前，「穩健」沒有任何意義：攻擊者可以改多少，攻擊者又知道些什麼？

威脅模型（threat model）固定了三件事。

1. 允許的擾動。多數研究使用乾淨影像 $\mathbf{x} \in [0,1]^d$ 周圍半徑為 ε 的 ℓ_p 球：

$$\mathcal{S}_p(\mathbf{x}) = \{\mathbf{x}' \in [0,1]^d : \|\mathbf{x}' - \mathbf{x}\|_p \le \varepsilon\}, \qquad \|\boldsymbol{\delta}\|_\infty = \max_i |\delta_i|, \qquad \|\boldsymbol{\delta}\|_2 = \Big(\sum_i \delta_i^2\Big)^{1/2},$$

其中 d 是像素值的個數，$\boldsymbol{\delta} = \mathbf{x}' - \mathbf{x}$ 是擾動，δ_i 是它的第 i 個分量。ℓ_∞ 球讓每個像素最多改變 ε（到處都改一點點）；ℓ_2 球限制改變的總能量；ℓ_0「範數」則計算有多少像素改變（只改幾個像素，但改很多）。在 CIFAR-10 上，ℓ_∞ 的 $\varepsilon = 8/255$ 是標準預算：Madry 等人在 0–255 的像素尺度上以 $\varepsilon = 8$ 訓練並攻擊 [23]，RobustBench 則為 ℓ_∞ 與 ℓ_2 威脅模型以及常見損壞維護排行榜 [24]。小的 ℓ_p 範數是「難以察覺」的方便替代，但不是它的定義。

2. 攻擊者的知識。在**白箱**（white-box）攻擊中，攻擊者知道模型與權重，可以計算梯度。在**黑箱**（black-box）攻擊中，攻擊者只能查詢模型。Papernot 等人展示了一種實用的黑箱攻擊：查詢目標模型來標註輸入，用這些輸入訓練一個本地的替代模型，在替代模型上製作對抗樣本，再靠**轉移**（transfer）攻擊目標 [25]。其他黑箱攻擊只靠查詢來估計決策邊界。穩健性的主張應該針對白箱攻擊來做，因為今天某個攻擊者做不到的事，明天資訊更充分的攻擊者可能就做到了。

3. 目標。無目標（untargeted）攻擊只要任何錯誤標籤；**有目標**（targeted）攻擊要的是指定的標籤（停車標誌被讀成速限標誌）。有目標攻擊比較難，所以兩者給出的數字不同。

FGSM 與 PGD 的精確版本

白話版。 FGSM 沿著損失往上爬一大步。PGD 則爬很多小步，每一步之後把影像拉回允許的球內。PGD 能找到更強的對抗樣本，是標準的一階攻擊。

令 $\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}, y)$ 是參數為 $\boldsymbol{\theta}$ 的網路在輸入 $\mathbf{x}$、標籤 y 上的交叉熵損失。對 ℓ_∞ 球的 **FGSM** [22] 為

$$\mathbf{x}_{\text{adv}} = \Pi_{[0,1]^d}\big(\mathbf{x} + \varepsilon\ \operatorname{sign}(\nabla_{\mathbf{x}} \mathcal{L}(\boldsymbol{\theta}, \mathbf{x}, y))\big),$$

其中 $\nabla_{\mathbf{x}}\mathcal{L}$ 是損失對輸入像素的梯度，sign 逐項作用（得到 -1、0 或 $+1$），$\Pi_{[0,1]^d}$ 把每個像素截到合法範圍。把每個像素恰好往增加損失的方向移動 $\pm\varepsilon$，是損失的線性近似在 ℓ_∞ 球內所能做到的最好。

投影梯度下降（projected gradient descent, PGD）[23] 從球內的隨機點開始，重複 T 次較小的步伐：

$$\mathbf{x}^{(0)} = \mathbf{x} + \mathbf{u}, \quad \mathbf{u} \sim \mathcal{U}[-\varepsilon,\varepsilon]^d, \qquad \mathbf{x}^{(t+1)} = \Pi_{\mathcal{S}(\mathbf{x})}\big(\mathbf{x}^{(t)} + \alpha\ \mathrm{sign}(\nabla_{\mathbf{x}}\mathcal{L}(\boldsymbol{\theta},\mathbf{x}^{(t)},y))\big),$$

其中 $\mathbf{u}$ 是球內的均勻雜訊（隨機起點），α 是步長，$t = 0,\dots,T-1$ 計算步數，$\Pi_{\mathcal{S}(\mathbf{x})}$ 是對允許集合的投影。對與合法像素範圍相交的 ℓ_∞ 球，投影是兩次截斷：先做 $x_i' \leftarrow \min(\max(x_i', x_i - \varepsilon), x_i + \varepsilon)$，再做 $x_i' \leftarrow \min(\max(x_i', 0), 1)$。最後的迭代點 $\mathbf{x}^{(T)}$ 就是對抗樣本。PGD 是「對損失做梯度上升」；這個名字來自最佳化領域，同一個演算法在那裡是最小化。因為損失曲面不是凹的，隨機起點與幾次重新啟動會有幫助；Madry 等人觀察到，從許多隨機起點到達的損失值相當集中，這支持了 PGD 是可靠的一階對手 [23]。

另一個常見的損失是 Carlini 與 Wagner 的**邊際損失**（margin loss），它直接最大化最大的錯誤 logit 與正確 logit 之間的差距，而不是交叉熵 [26]。交叉熵飽和時它就派上用場，這也是認真的評估要跑好幾種攻擊的原因之一。

僅線上版：決定評估結果的攻擊設定 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/notes/決定評估結果的攻擊設定/*

對抗訓練

白話版。如果模型在附近最壞的影像上失敗，就用附近最壞的影像訓練它。訓練時，對每個批次先用 PGD 攻擊，然後用被攻擊過的影像而不是乾淨影像來擬合。

Madry 等人把穩健訓練寫成最小–最大（鞍點）問題 [23]：

$$\min_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta}), \qquad \rho(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\Big[\max_{\boldsymbol{\delta}\in\mathcal{S}} \mathcal{L}(\boldsymbol{\theta},\mathbf{x}+\boldsymbol{\delta},y)\Big],$$

其中 $\mathcal{D}$ 是資料分布，$\mathcal{S}$ 是允許的擾動集合（例如 $\|\boldsymbol{\delta}\|_\infty \le \varepsilon$），內層的最大化是攻擊者的最佳擾動，外層的最小化是防禦者的訓練。實務上，內層最大化用幾步的 PGD 近似，外層最小化則對得到的對抗批次做一般的隨機梯度下降。

它的代價：

- **運算量。**每個訓練步驟都需要一次 T 步的攻擊，所以訓練的成本大約是 $T+1$ 倍。
- **容量。** Madry 等人發現，穩健分類需要比乾淨分類更大的網路；小網路對 PGD 訓練時可能崩潰成只預測一個類別，而用一步式 FGSM 訓練則會對它狹窄的擾動集合過擬合，在大 ε 下無法抵擋 PGD [23]。
- **乾淨準確率。**穩健模型在乾淨資料上通常比較不準。Tsipras 等人證明了這個取捨在一個簡單而自然的設定中確實存在，並主張穩健模型學到的是不同的特徵，而這些特徵恰好和人類感知更一致 [27]。**TRADES** 把穩健錯誤拆成自然錯誤加上邊界錯誤，並用一個損失透過單一權重明確地在兩者間取捨 [28]。

僅線上版：*Madry 等人：Towards Deep Learning Models Resistant to Adversarial Attacks*（*ICLR*

2018）— https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/papers/madry-等 人 towards-deep-learning-models-resistant-to-adversarial-attacksiclr-2018/

程式碼：在手寫數字上做 FGSM、PGD 與對抗訓練

NumPy 手動計算單隱藏層 MLP 的輸入梯度：對 softmax 輸出 $\mathbf{p}$ 與 one-hot 標籤 $\mathbf{e}_y$，交叉熵對 logits 的梯度是 $\mathbf{p} - \mathbf{e}_y$，再把它依序反傳過輸出層、ReLU 與第一層。兩個模型都對訓練資料做同樣的 50 趟；穩健模型擬合的是針對它當前權重製作的 PGD 樣本（$\varepsilon = 0.15$，5 步）。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

X, y = load_digits(return_X_y=True)
X = X / 16.0                                            # pixels in [0, 1]
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, stratify=y, random_state=0)
rng = np.random.default_rng(0)

def input_grad(m, X, y):
    """Gradient of the cross-entropy loss with respect to the input pixels."""
    (W1, W2), (b1, b2) = m.coefs_, m.intercepts_
    h = np.maximum(X @ W1 + b1, 0)                      # hidden ReLU layer
    z = h @ W2 + b2                                     # logits
    g = np.exp(z - z.max(1, keepdims=True)); g /= g.sum(1, keepdims=True)
    g[np.arange(len(y)), y] -= 1                        # dL/dz = softmax(z) - one-hot(y)
    return ((g @ W2.T) * (h > 0)) @ W1.T                # back through ReLU and first layer

def fgsm(m, X, y, eps):
    return np.clip(X + eps * np.sign(input_grad(m, X, y)), 0, 1)

def pgd(m, X, y, eps, steps=10):
    alpha = 2.5 * eps / steps                           # step size
    Xa = np.clip(X + rng.uniform(-eps, eps, X.shape), 0, 1)        # random start in the ball
    for _ in range(steps):
        Xa = Xa + alpha * np.sign(input_grad(m, Xa, y))             # ascent step on the loss
        Xa = np.clip(np.clip(Xa, X - eps, X + eps), 0, 1)           # project: eps-ball, then
         ↪  [0, 1]
    return Xa

def train(adversarial, epochs=50, eps=0.15):
    m = MLPClassifier(hidden_layer_sizes=(128,), random_state=0)
    m.partial_fit(Xtr, ytr, classes=np.arange(10))    # first pass on clean images
    for _ in range(epochs - 1):                       # Madry et al.: fit the worst case
     ↪  found by PGD
        m.partial_fit(pgd(m, Xtr, ytr, eps, steps=5) if adversarial else Xtr, ytr)
    return m

standard, robust = train(False), train(True)
for eps in [0, 0.05, 0.1, 0.15, 0.2]:
    acc = [m.score(atk(m, Xte, yte, eps), yte) for m in (standard, robust) for atk in (fgsm,
     ↪  pgd)]
```

```
    print(f"eps={eps:.2f}  standard: FGSM {acc[0]:.3f} PGD {acc[1]:.3f}"
          f"   adversarially trained: FGSM {acc[2]:.3f} PGD {acc[3]:.3f}")
```

它印出（在筆電 CPU 上約 10–20 秒）：

```
eps=0.00  standard: FGSM 0.967 PGD 0.967   adversarially trained: FGSM 0.956 PGD 0.956
eps=0.05  standard: FGSM 0.861 PGD 0.854   adversarially trained: FGSM 0.891 PGD 0.889
eps=0.10  standard: FGSM 0.574 PGD 0.528   adversarially trained: FGSM 0.757 PGD 0.746
eps=0.15  standard: FGSM 0.237 PGD 0.178   adversarially trained: FGSM 0.626 PGD 0.593
eps=0.20  standard: FGSM 0.057 PGD 0.007   adversarially trained: FGSM 0.415 PGD 0.361
```

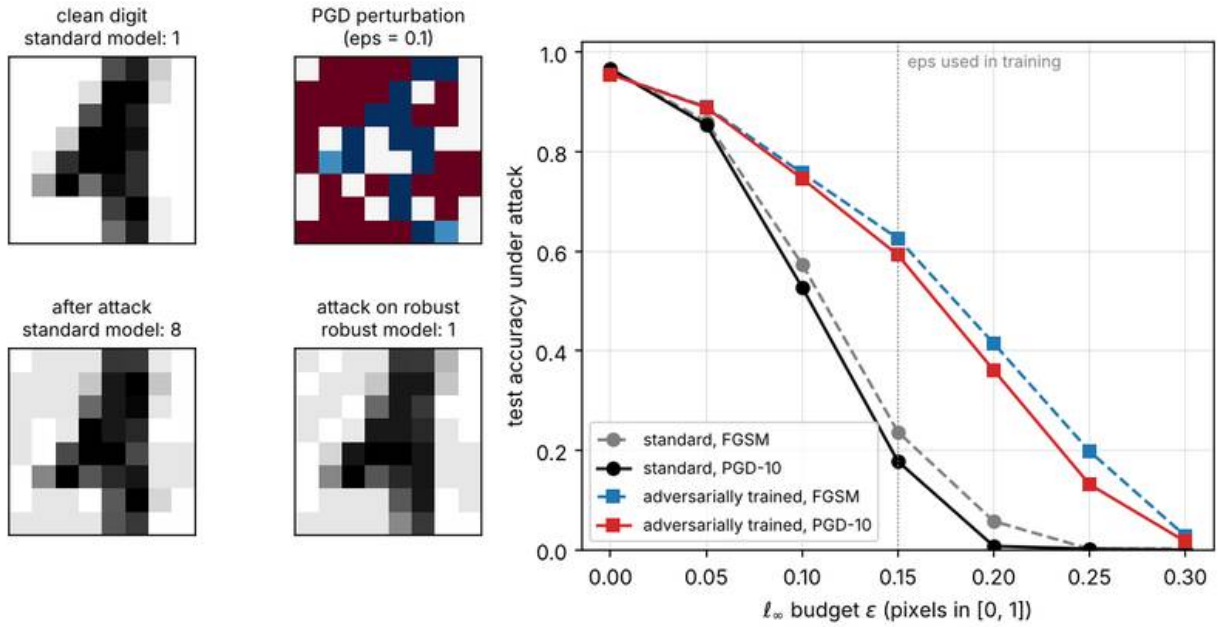


圖 *3* — 兩個 *MLP* 上的 *FGSM* 與 *PGD*（我們的玩具實驗）。左：$\varepsilon = 0.1$ 的 *PGD* 擾動讓標準模型的答案從 *1* 變成 *8*，但沒有改變對抗訓練模型的答案。右：預算 ε 下受攻擊的準確率；*PGD* 永遠至少和 *FGSM* 一樣強，對抗訓練在每個 ε 都提高了穩健準確率，代價是少許乾淨準確率。

（所有數字都來自我們的玩具實驗。）這張表可以從三個方向讀：

- **沿著一欄往下：**準確率隨 ε 快速下降。在 $\varepsilon = 0.1$ 時，每個像素可以改變資料集 16 個灰階中的 1.6 個，標準模型在 PGD 下從 0.967 降到 0.528。
- **橫跨一組攻擊：** PGD 從不比 FGSM 弱，而且差距隨 ε 擴大（標準模型在 $\varepsilon = 0.2$ 時是 0.057 對 0.007）。只用 FGSM 的評估會高估穩健性。
- **橫跨模型：**對抗訓練把 $\varepsilon = 0.15$ 時的 PGD 準確率從 0.178 提高到 0.593，並犧牲約一個百分點的乾淨準確率（0.967 變成 0.956），是那個取捨的小規模版本。超出訓練預算後，穩健性就消退了（圖中 $\varepsilon = 0.25$ 時是 0.131）：對於訓練時所針對的威脅模型之外的任何事，穩健性都沒有保證。

誠實地評估穩健性

白話版。很容易造出只會弄壞攻擊者工具的「防禦」，例如讓梯度變得沒用，但模型其實一樣容易被騙。這種穩健性只存在於紙面上，直到有人用更好的攻擊為止。

Athalye、Carlini 與 Wagner 把這個現象稱為**混淆梯度**（obfuscated gradients），是梯度遮蔽（gradient masking）的一種，並描述了三種類型 [29]：**破碎梯度**（shattered gradients；不可微或數值不穩定的運算，例如對輸入做量化，會讓梯度為零或錯誤）；**隨機梯度**（stochastic gradients；測試時的隨機輸入變換讓單一次的梯度具有誤導性）；以及**消失或爆炸梯度**（vanishing or exploding gradients；執行迭代流程的防禦，例如把輸入投影到學到的流形上，行為就像一個極深的網路）。他們為每一種類型各給出一個反制攻擊：**BPDA**（backward pass differentiable approximation，反向傳遞可微分近似）在前向傳遞照常執行防禦，反向傳遞則換成可微分的近似（通常是恆等函數）；**EOT**（expectation over transformation，變換的期望）對隨機變換的梯度取平均；以及**重新參數化**（reparameterization），藉由變數變換讓迭代步驟從梯度路徑中消失。在針對 ICLR 2018 收錄的九個白箱防禦所做的案例研究中，有七個依賴混淆梯度，他們的攻擊完全繞過了其中六個，部分繞過一個 [29]。

同一篇論文還列出評估有問題的警訊：一步式攻擊比迭代式攻擊更有效；黑箱攻擊比白箱攻擊更有效；無界攻擊的成功率沒有達到 100%；隨機取樣找得到梯度攻擊找不到的對抗樣本；以及增加預算並沒有提高成功率 [29]。我們的玩具實驗通過了它能通過的檢查：PGD 勝過 FGSM，而且 ε 增大時準確率會降到零。

為了讓評估較不依賴評估者的細心程度，Croce 與 Hein 提出了 **AutoAttack**，這是一個免調參數的集成，由兩個改良的 PGD 變體（有自動的步長排程與不同的損失）和兩個互補的既有攻擊組成；用在 50 多個已發表的防禦上，除了一個之外，所有防禦的穩健準確率都比原本報告的更低 [30]。**RobustBench** 以 AutoAttack 建立了公開排行榜，並對允許的模型訂有規則，同時也邀請在 AutoAttack 可能高估穩健性的地方使用適應性攻擊 [24]。所以，報告穩健準確率時，應該說明威脅模型、ε、所用的攻擊（至少要有 AutoAttack 或針對該防禦設計的適應性攻擊）、步數與重新啟動次數，以及測試集。

僅線上版：*Athalye* 等人：*Obfuscated Gradients Give a False Sense of Security*（*ICML 2018*）— *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/papers/athalye-等人_obfuscated-gradients-give-a-false-sense-of-securityicml-2018/*

認證穩健性

白話版。攻擊只能證明模型在某個點不穩健。認證則能證明，在某個半徑內沒有任何擾動能改變答案，無論用哪種攻擊。

據作者所述，**隨機平滑**（randomized smoothing）[31] 是當時唯一被證明能在 ImageNet 上實行的認證防禦，至今仍是標準的參照點。取任何基底分類器 f，定義平滑分類器 g，回傳 f 在高斯雜訊下最常預測的類別：

$$g(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} \Pr_{\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})} [f(\mathbf{x} + \boldsymbol{\eta}) = c],$$

其中 $\boldsymbol{\eta}$ 是每個像素標準差為 σ 的各向同性高斯雜訊，$\mathbf{I}$ 是單位矩陣。Cohen、Rosenfeld 與 Kolter 證明，如果最高的類別 c_A 在雜訊下的機率至少為 $\underline{p_A}$，其他每個類別至多為 $\overline{p_B}$，那麼對每個滿足

下式的擾動 δ，g 都預測 c_A：

$$\|\delta\|_2 < R = \frac{\sigma}{2}\left(\Phi^{-1}(\underline{p_A}) - \Phi^{-1}(\overline{p_B})\right),$$

其中 Φ^{-1} 是標準常態累積分布函數的反函數 [31]。實務上，$\underline{p_A}$ 是由蒙地卡羅樣本估計的信賴下界，而且無法認證時，方法會棄權。這個保證適用於 ℓ_2 威脅模型，附帶選定的失敗機率，並且是針對平滑分類器 g，不是針對 f。較大的 σ 允許較大的半徑，但會降低乾淨準確率，同樣是一種取捨。

僅線上版：隨機平滑的實作 — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/notes/隨機平滑的實作/*

實體攻擊與貼片攻擊

白話版。攻擊不必隱形，也不必是數位的。一張列印出來的貼紙就能騙過相機。

Eykholt 等人產生了能在視角、距離與光線改變下依然有效的擾動，把它們印成黑白貼紙貼在真實的停車標誌上，結果讓道路標誌分類器在多數實驗室影像中，以及在經過路旁的影片的許多影格中，輸出攻擊者指定的目標類別 [32]。Brown 等人做出了**對抗貼片**（adversarial patch）：通用、可列印的影像區域，放在場景中任何位置，都會讓分類器回報指定的類別 [33]。這類攻擊離開了 ℓ_p 威脅模型：它們很大，但位置集中。它們對任何以相機為基礎的系統都很重要，也與 DLCV 02A 討論的人臉辨識呈現攻擊有關。

分布外與異常偵測

白話版。分類器總是會從它認識的類別中挑一個，即使面對的是從沒見過的東西的照片。OOD 偵測多加了第二個輸出，一個分數，用來說「這個輸入看起來像我的訓練資料」或「不像」，讓系統可以拒絕作答。

你要解決的是哪一個任務？

好幾個密切相關的任務有著不同的名稱。Yang 等人把它們放進同一個框架，稱為**廣義 OOD 偵測**（generalized OOD detection），並依需要偵測哪一種偏移，以及 ID 資料是單一類別還是多個類別來區分 [3]：

任務	分布內資料	必須標記	典型例子
OOD 偵測	一個 K 類的訓練集	來自其他類別的輸入（語意偏移），同時仍要分類 ID 輸入	寵物分類器看到一輛車
開放集合辨識（open-set recognition）	K 個已知類別	未知類別，同時分類已知的	PR 10、DLCV 02A
感官異常偵測（sensory anomaly detection）	同一種類的正常影像	看起來不同的輸入（共變量偏移）	產線上有刮痕的零件

任務	分布內資料	必須標記	典型例子
語意異常／新穎性偵測（semantic anomaly / novelty detection）	一個或少數幾個類別	來自任何其他類別的輸入	相機陷阱影像中出現一個新物種

OOD 偵測與開放集合辨識的差別主要在於歷史與基準；該綜述把開放集合辨識視為多類別的新穎性偵測 [3]。它們都不同於 PR 01 的**拒絕選項**（reject option），後者是對已知類別之間模稜兩可的輸入棄權，而不是對新穎的輸入棄權。

分數

白話版。每個偵測器最後都化為每個輸入一個數字。以輸出為基礎的分數問分類器有多有信心；以特徵為基礎的分數問輸入的特徵離訓練特徵有多遠。

令 $\mathbf{z}(\mathbf{x}) \in \mathbb{R}^K$ 為訓練好的分類器的 logits，$\mathbf{h}(\mathbf{x}) \in \mathbb{R}^m$ 為它的倒數第二層特徵（最後一個線性層的輸入），並約定**分數越大代表越像分布內**。四個標準的事後分數，不需要重新訓練：

最大 softmax 機率（maximum softmax probability, MSP） [34]：

$$s_{\text{MSP}}(\mathbf{x}) = \max_k \frac{e^{z_k(\mathbf{x})}}{\sum_{j=1}^{K} e^{z_j(\mathbf{x})}},$$

其中 z_k 是第 k 個 logit。Hendrycks 與 Gimpel 指出，被正確分類的樣本，其 MSP 往往高於被誤分類的樣本與 OOD 樣本，這讓 MSP 成為標準的基準線 [34]。

能量分數（energy score） [35]：

$$E(\mathbf{x}) = -T \log \sum_{k=1}^{K} e^{z_k(\mathbf{x})/T}, \qquad s_{\text{energy}}(\mathbf{x}) = -E(\mathbf{x}),$$

其中 $T > 0$ 是溫度（通常 $T = 1$）。負能量是 logits 的 log-sum-exp，是一種平滑的最大值，它保留了 logits 的大小，而這是 softmax 正規化掉的資訊。Liu 等人主張能量與輸入密度的對應比 softmax 信心更好，並報告在他們的 CIFAR 基準上，95% TPR 下的 FPR 比 MSP 更低 [35]。

馬氏距離（Mahalanobis distance） [36]：對訓練特徵為每個類別擬合一個高斯分布，各類別平均為 $\boldsymbol{\mu}_c$，並共用一個（「綁定」的）共變異數 $\boldsymbol{\Sigma}$，分數為

$$s_{\text{Maha}}(\mathbf{x}) = -\min_c \, (\mathbf{h}(\mathbf{x}) - \boldsymbol{\mu}_c)^{\top} \boldsymbol{\Sigma}^{-1} (\mathbf{h}(\mathbf{x}) - \boldsymbol{\mu}_c),$$

其中最小值遍歷 K 個類別。這就是 PR 01 的線性判別分析，放到學到的特徵上。Lee 等人還加上了小幅的輸入擾動，並結合多個層；我們只用最後一個隱藏層。

k 近鄰（kNN）距離 [37]：把特徵正規化成單位長度，$\hat{\mathbf{h}} = \mathbf{h}/\|\mathbf{h}\|_2$，分數為

$$s_{\text{kNN}}(\mathbf{x}) = -\|\hat{\mathbf{h}}(\mathbf{x}) - \hat{\mathbf{h}}_{(k)}\|_2,$$

其中 $\hat{\mathbf{h}}_{(k)}$ 是離 $\hat{\mathbf{h}}(\mathbf{x})$ 第 k 近的正規化訓練特徵。Sun 等人發現，這個不需要高斯假設的非參數分數表現良好，而且特徵正規化至關重要 [37]。

這些分數不更動分類器。訓練時的方法則會改變它：**離群值暴露**（Outlier Exposure）訓練分類器對一大批輔助的離群影像給出均勻（低信心）的預測，在原始實驗中改善了對沒見過的 OOD 資料的偵測 [38]。

指標

白話版。把偵測當成二元決策，ID 對 OOD，問分數把兩群分開得多好：看所有門檻，或看一個能保留大多數 ID 輸入的門檻。

把 ID 測試輸入當作正例、OOD 輸入當作負例，當 $s(\mathbf{x}) \geq \tau$ 時接受該輸入。則 TPR(τ) 是被接受的 ID 輸入的比例，FPR(τ) 是被接受的 OOD 輸入的比例。有三個標準指標 [35]：

- **AUROC**，也就是 τ 變化時 TPR 對 FPR 曲線下的面積。它等於隨機一個 ID 輸入的分數高於隨機一個 OOD 輸入的機率；0.5 是隨機猜測，1 是完美分開。
- **FPR@95TPR** = FPR(τ_{95})，其中 τ_{95} 是使 TPR(τ_{95}) = 0.95 的門檻：保留 95% 的 ID 輸入時，仍被接受的 OOD 輸入所占的比例。越低越好。它描述的是單一個工作點，也就是實際部署的系統所使用的那種。
- **AUPR**，精確率–召回率曲線下的面積，以 ID 輸入為正類（AUPR-In）或以 OOD 輸入為正類（AUPR-Out）來報告 [34]。它取決於 ID 與 OOD 測試輸入的比例，所以應該把這個比例一併報告。

ROC 曲線與精確率–召回率曲線本身，在 DLCV 02 中有說明。

程式碼：四個分數，近與遠的 OOD

ID 資料是數字 0–4。**近 OOD** 輸入是沒見過的數字 5–9：同一種影像，新的類別。**遠 OOD** 輸入是 scikit-image 中照片的 8×8 區塊，縮放到相同的像素範圍：完全不同種類的影像。分類器在訓練時兩者都沒見過。

```
import numpy as np
from scipy.special import logsumexp, softmax
from skimage import data, transform
from sklearn.datasets import load_digits
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

X, y = load_digits(return_X_y=True)
X = X / 16.0
known = y < 5                                          # in-distribution (ID): digits 0-4
Xtr, Xid, ytr, yid = train_test_split(X[known], y[known], test_size=0.3, stratify=y[known],
 ↪  random_state=0)
X_near = X[~known]                                     # near-OOD: unseen digits 5-9
rng = np.random.default_rng(0)
photos = [transform.resize(im / 255.0, (256, 256)) for im in (data.camera(), data.moon(),
 ↪  data.coins(), data.text())]
X_far = []                                             # far-OOD: 8x8 patches of photographs
for _ in range(600):
```

```
    im = photos[rng.integers(4)]; r, c = rng.integers(0, 224, 2)
    p = transform.resize(im[r:r + 32, c:c + 32], (8, 8), anti_aliasing=True)
    X_far.append(((p - p.min()) / (np.ptp(p) + 1e-8)).ravel())
X_far = np.array(X_far)

m = MLPClassifier(hidden_layer_sizes=(64,), alpha=1e-3, max_iter=500,
 ↪  random_state=0).fit(Xtr, ytr)
(W1, W2), (b1, b2) = m.coefs_, m.intercepts_
feat = lambda X: np.maximum(X @ W1 + b1, 0)           # penultimate (hidden) features
unit = lambda F: F / (np.linalg.norm(F, axis=1, keepdims=True) + 1e-12)
F = feat(Xtr)
mu = np.stack([F[ytr == c].mean(0) for c in range(5)])           # class means
cov = sum(np.cov(F[ytr == c].T, bias=True) * np.sum(ytr == c) for c in range(5)) / len(F)
P = np.linalg.inv(cov + 1e-3 * np.eye(len(cov)))                # shared precision matrix

def scores(X):                                        # every score: larger = "more
 ↪  in-distribution"
    H = feat(X); z = H @ W2 + b2; d = H[:, None] - mu[None]
    return {"MSP": softmax(z, axis=1).max(1),
            "energy": logsumexp(z, axis=1),           # = -E(x) with temperature 1
            "Mahalanobis": -np.einsum("ncd,de,nce->nc", d, P, d).min(1),
            "kNN": -np.sort(np.linalg.norm(unit(H)[:, None] - unit(F)[None], axis=2), 1)[:,
             ↪  9]}

s_id = scores(Xid)
print(f"ID test accuracy {m.score(Xid, yid):.3f}")
for name, X_ood in [("near", X_near), ("far", X_far)]:
    s_ood = scores(X_ood)
    for k in s_id:
        auroc = roc_auc_score(np.r_[np.ones(len(Xid)), np.zeros(len(X_ood))], np.r_[s_id[k],
         ↪  s_ood[k]])
        fpr95 = np.mean(s_ood[k] >= np.quantile(s_id[k], 0.05))  # OOD accepted when 95% of
         ↪  ID is kept
        print(f"{name:4s} {k:11s} AUROC {auroc:.3f}  FPR@95TPR {fpr95:.3f}")
```

它印出：

```
ID test accuracy 1.000
near MSP         AUROC 0.941  FPR@95TPR 0.291
near energy      AUROC 0.955  FPR@95TPR 0.258
near Mahalanobis AUROC 0.961  FPR@95TPR 0.209
near kNN         AUROC 0.970  FPR@95TPR 0.152
far  MSP         AUROC 0.870  FPR@95TPR 0.448
far  energy      AUROC 0.811  FPR@95TPR 0.660
far  Mahalanobis AUROC 1.000  FPR@95TPR 0.000
far  kNN         AUROC 0.995  FPR@95TPR 0.010
```

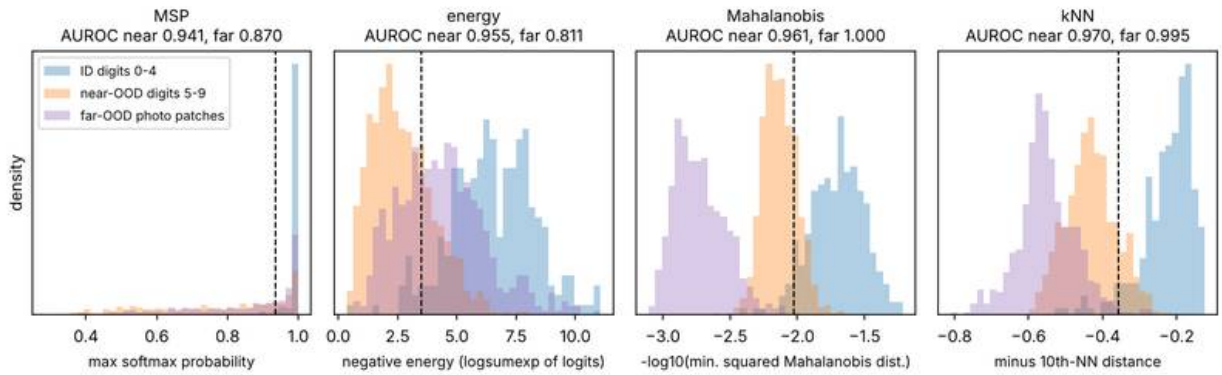


圖 *4 — ID* 數字 *0–4*、近 *OOD* 數字 *5–9* 與遠 *OOD* 照片區塊的分數分布（我們的玩具實驗）。虛線是接受 *95% ID* 測試輸入的門檻；它右邊的一切都被接受。馬氏距離面板的對數尺度只用於顯示，不會改變任何排序。

（所有數字都來自我們的玩具實驗。）有三個教訓：

- **內容上的「遠」並不代表每種分數都「容易」**。照片區塊和數字毫無相似之處，但在 95% TPR 的門檻下，MSP 接受了其中的 45%，能量接受了 66%。分類器信心十足地把它們對應到某個數字，因為信心衡量的是離決策邊界有多遠，而不是離資料有多遠。以特徵距離為基礎的分數看得出這些區塊的特徵離所有訓練特徵都很遠，因此幾乎全部拒絕。
- **分數的排名會隨 OOD 資料集改變**。在沒見過的數字上，能量勝過 MSP，在照片區塊上卻輸給它。大型基準研究 OpenOOD v1.5 同樣發現，沒有哪個方法能在所有資料集上都勝出 [39]。
- **AUROC 與 FPR@95TPR 在大小上並不一致**。 AUROC 0.941 聽起來很好，但在可部署的門檻下，MSP 仍然接受 29% 的沒見過的數字。

PR 10 用另一個分類器、以保留的數字跑了一個相關的開放集合實驗，並示範了代理的未知集合如何誤導人（PR 10）。

工業異常偵測

白話版。在生產線上，瑕疵稀少而且五花八門，所以你不可能收集到每一種瑕疵的樣本。你改為只學正常零件看起來是什麼樣子，一個區塊一個區塊地學，然後標記任何和你見過的所有正常區塊都不像的區塊。

MVTec AD 讓這種設定成為標準基準：5354 張物體與紋理類別的高解析度影像，訓練用的無瑕疵影像，含有超過 70 種瑕疵（刮痕、凹陷、汙染、結構變化）的測試影像，以及瑕疵區域的像素級精確遮罩 [40]。方法在兩個層次上評分：**影像層級**（這個零件有瑕疵嗎？）與**像素層級**（瑕疵在哪裡？），通常用 AUROC；也會使用區域重疊的量測，例如 AU-PRO，MVTec AD 2 就是一例 [41]。用上表的說法，這是感官異常偵測。

PatchCore [42] 是建立在 kNN 想法上的強大而簡單的方法。它把無瑕疵的訓練影像送進在 ImageNet 上預訓練的網路，保留**中層的區塊特徵**（最後幾層對 ImageNet 類別太特定，對小瑕疵又太粗糙），每個特徵都在一個小鄰域內取平均以納入上下文。所有這些特徵放進一個**記憶庫**（memory bank），再用**貪婪核心集（coreset）子取樣**把它縮成一個仍能涵蓋罕見正常模式的小子集。測試時，每個區塊在記憶庫中的**最近鄰距離**構成異常圖，圖的最大值（稍加重新加權）就是

影像分數。這個方法是為「冷啟動」（cold-start）設定設計的，也就是只有正常（無瑕疵）影像可用，它在 MVTec AD 上達到當時最先進的偵測與定位表現 [42]。

僅線上版：*PatchCore 逐步解析* — *https://blog.shyandram.dev/zh-tw/learn/deep-learning/robustness/notes/patchcore-逐步解析/*

程式碼：磚牆上的區塊記憶庫

沒有網路權重，所以我們用原始的 7×7 像素區塊作為特徵，這保留了 PatchCore 的機制（記憶庫、核心集、最近鄰距離圖、最大值池化的影像分數），只拿掉學出來的特徵。「產品」是 scikit-image 磚牆紋理的 64×64 裁切：正常的訓練裁切取自影像上半部，測試裁切取自下半部，一半的測試裁切會被加上一條細的合成刮痕。

```
import numpy as np
from skimage import data, draw, transform
from skimage.util import view_as_windows
from sklearn.metrics import roc_auc_score
from sklearn.neighbors import NearestNeighbors

rng = np.random.default_rng(0)
tex = transform.rescale(data.brick() / 255.0, 0.5)   # 256x256 brick-wall "product surface"
S, P = 64, 7                                          # image size, patch size

def crop(top):                                        # normal images: top half; test images:
 ↪  bottom half
    r = rng.integers(0, 128 - S) + (0 if top else 128); c = rng.integers(0, 256 - S)
    return tex[r:r + S, c:c + S].copy()

def scratch(im):                                      # a thin line, 0.5 brighter or darker
    r, c = rng.integers(16, S - 16, 2); a = rng.uniform(0, np.pi); L = rng.integers(8, 16)
    rr, cc = draw.line(r, c, int(r + L * np.sin(a)), int(c + L * np.cos(a)))
    mask = np.zeros_like(im, bool); mask[rr, cc] = True
    im[mask] = np.clip(im[mask] + rng.choice([-0.5, 0.5]), 0, 1)
    return im, mask

def patches(im, step=1):                              # one 7x7 patch centred on each pixel
    return view_as_windows(np.pad(im, P // 2, mode="reflect"), (P, P), step).reshape(-1, P *
     ↪  P)

def greedy_coreset(M, n):                             # keep n patches that cover the bank
 ↪  well
    keep, d = [0], np.linalg.norm(M - M[0], axis=1)
    for _ in range(n - 1):
        i = int(d.argmax()); keep.append(i)
        d = np.minimum(d, np.linalg.norm(M - M[i], axis=1))
    return M[keep]

bank = np.vstack([patches(crop(top=True), step=2) for _ in range(16)])
memory = NearestNeighbors(n_neighbors=1).fit(greedy_coreset(bank, len(bank) // 10))

is_defect, img_score, pix_score, pix_true = [], [], [], []
for k in range(40):
```

```
    im, mask = scratch(crop(top=False)) if k % 2 else (crop(top=False), np.zeros((S, S),
     ↪  bool))
    amap = memory.kneighbors(patches(im))[0].reshape(S, S)   # nearest-neighbour distance per
     ↪  pixel
    is_defect.append(k % 2); img_score.append(amap.max())
    pix_score.append(amap.ravel()); pix_true.append(mask.ravel())
print(f"memory bank {len(bank)} -> {len(bank) // 10} patches")
print(f"image-level AUROC {roc_auc_score(is_defect, img_score):.3f}")
print(f"pixel-level AUROC {roc_auc_score(np.concatenate(pix_true),
 ↪  np.concatenate(pix_score)):.3f}")
```

它印出：

```
memory bank 16384 -> 1638 patches
image-level AUROC 0.955
pixel-level AUROC 0.996
```

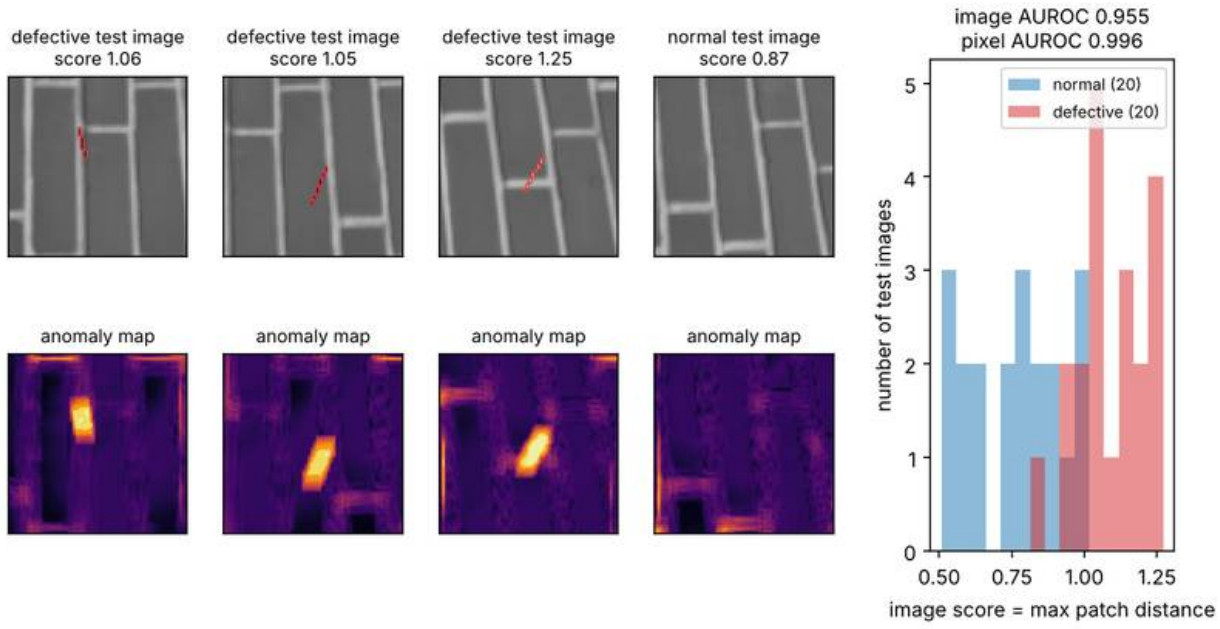


圖 *5* — 磚牆裁切上的區塊記憶庫（我們的玩具實驗）。上：測試影像（紅色外框：合成刮痕）。下：最近鄰距離圖。右：*20* 張正常與 *20* 張有瑕疵測試影像的影像分數（圖的最大值）。

（所有數字都來自我們的玩具實驗。）像素層級的 AUROC 幾乎完美，影像層級的 AUROC 卻較低：一個影像分數是 4096 個區塊距離的最大值，所以單一個不尋常、卻是正常的區塊（記憶庫裡沒有的灰漿轉角），會把正常影像的分數拉高，程度和真正的瑕疵一樣。更多正常訓練影像會減少這類誤報；瑕疵越淡，誤報越多（練習 4）。在異常圖中，灰漿邊緣有些發亮：它們是最罕見的正常結構，這正是 PatchCore 要保留涵蓋它們的核心集，而不是隨機子樣本的原因。

OOD 與異常評估的陷阱

白話版。 OOD 的數字取決於選了哪些 OOD 資料，勝過取決於方法，而且許多已發表的 OOD 測試集有瑕疵。讀每個 OOD 結果時，都要連同它的測試集一起讀。

- **近 OOD 對遠 OOD。** OpenOOD 把 OOD 測試集分成近 OOD（與 ID 資料相似）和遠 OOD（明顯不同），並發現所有偵測器在這兩組之間的偵測表現都有很大的差距 [39]。我們的玩具實驗顯示這個標籤可能誤導：對以輸出為基礎的分數，「遠」的照片區塊比「近」的數字更難。
- **被汙染的 OOD 測試集。** Bitterwolf 等人發現，許多與 ImageNet-1K 搭配使用的 OOD 測試集，包含很大比例實際上顯示 ID 類別的影像，有些情況超過一半，這扭曲了評估；他們建立了 NINCO，一個每張影像都經過檢查、確認不含 ID 物體的 OOD 測試集 [43]。
- **預訓練重疊。**建立在以網路規模資料預訓練的模型上的偵測器，可能在預訓練時就見過「未知」類別。這樣一來，測試就不再與未知有關，必須據此描述設定。
- **門檻與模型選擇。**和任何超參數一樣，門檻與分數的選擇必須用驗證資料來定。在 OOD 測試集上（或在與它太相似的代理上）調整它們，會得到和 DomainBed 警告的「神諭式模型選擇」同樣的樂觀 [19]。
- **ID 集合內的共變量偏移。**在部署中，ID 影像可能以模糊或新光線的樣子出現。OpenOOD v1.5 評估「全頻譜」（full-spectrum）偵測，其中共變量偏移的 ID 影像必須被接受，而語意 OOD 影像必須被拒絕，並發現這對目前所有方法都是重大挑戰 [39]。這裡正是本篇第一個與第三個問題交會的地方。

穩健性評估計畫

白話版。在宣稱模型穩健之前，先寫下它對什麼穩健，用手邊最強的工具去測試，並把結果和乾淨的數字並列報告。

1. **為問題命名。**偏移（哪一種？）、對手（哪個威脅模型？），還是未知的輸入（哪一種新穎性？）。如果不只一種，分別評估。
2. **先建立測試資料。**至少有一組來自預期部署環境（新地點、裝置或時間）的真實偏移資料，合成損壞作為便宜的補充，以及一組確認過沒有 ID 汙染的保留 OOD 集。
3. **使用強力的評估。**對抗性的主張：白箱攻擊、帶重新啟動的 PGD，以及 AutoAttack 或針對該防禦的適應性攻擊，上面提到的警訊檢查，還有確切的 ε 與範數。對偏移：有效穩健性或相同的乾淨準確率。對 OOD：在近與遠 OOD 集上的 AUROC 與 FPR@95TPR。
4. **別讓測試資料進入迴圈。**超參數、門檻與檢查點都用來自訓練分布的驗證資料來選，或清楚說明用了神諭。
5. **報告取捨。**乾淨準確率、穩健準確率、校準與運算成本要一起報告；一項的進步不是免費的。
6. **上線後要監控。**像 ML 11 那樣追蹤輸入統計量與信心分布，在多感測器系統中，還要追蹤 DLCV 03B 列出的感測器故障。

現代觀點

值得一讀的綜述。

- **Yang、Zhou、Li 與 Liu [3]**（IJCV 2024）把異常偵測、新穎性偵測、開放集合辨識、OOD 偵測與離群值偵測統一為廣義 *OOD* 偵測的子任務，依需要偵測的是共變量或語意偏移，以及 ID 資料是一個還是多個類別來整理；接著逐一回顧各個領域，其中對 OOD 偵測方法（以分類為基礎、以密度為基礎與以距離為基礎）著墨最多。先讀它，可以釐清詞彙。
- **Zhou 等人 [18]**（TPAMI）定義領域泛化，把它與領域適應和遷移學習連結起來，並回顧依領域對齊、元學習、資料擴增、集成學習等分組的方法，另外也涵蓋理論和視覺以外的應用。
- **Liang、He 與 Tan [44]**（IJCV）依測試資料的形式，把測試時適應分成測試時領域適應、測試時批次適應與線上測試時適應，並為每組演算法整理分類，討論尚未解決的挑戰。
- **Akhtar、Mian、Kardan 與 Shah [45]**（IEEE Access 2021）回顧 2018 年以來電腦視覺中的對抗攻擊與防禦：對分類器的攻擊、分類之外的攻擊、實體世界的攻擊、對抗樣本為何存在的理論，以及依模型修改、輸入修改與外部偵測器分組的防禦，認證防禦則另外處理。
- **Pang、Shen、Cao 與 van den Hengel [46]**（ACM Computing Surveys 2021）把深度異常偵測整理成三種框架：用深度學習做通用特徵抽取、學習正常性的表示，以及端到端的異常分數學習，細分為十一個類別，並討論每一種的假設、優點與缺點。
- **Liu 等人 [47]**（Machine Intelligence Research 2024）依網路架構、監督程度、損失函數、指標與資料集回顧工業影像異常偵測，並提出一個更貼近實際製造的設定。

它們共通的結論。分布偏移、對抗穩健性與 OOD 偵測是各自發展的社群，有各自的基準，而這些綜述一再發現，基準的選擇、模型選擇與評估協定，對結果的影響不亞於方法本身。公平比較時，簡單的基準線（訓練得好的 ERM、MSP 或 kNN 分數、搭配 PGD 的對抗訓練）仍然很難被超越。

2023–2026 年的最新進展。可以看到五個趨勢。(1) **基礎模型改變了起點**。大規模、多樣的預訓練，帶來了專門方法做不到的大部分自然偏移穩健性 [12], [13]，而權重內插這類穩健微調嘗試保住它 [14]；OOD 基準現在也納入了 CLIP 與 DINOv2 骨幹 [39]。(2) **用生成資料做對抗訓練**。在不使用外部真實資料的情況下，用現代擴散模型生成的影像訓練，在 RobustBench 的 CIFAR-10 與 CIFAR-100 上創下新的結果 [48]；這些基準上的穩健準確率仍遠低於乾淨準確率。(3) **更好的 OOD 評估**。具有近與遠 OOD 切分和全頻譜設定的統一基準 [49], [39]，以及清理過的測試集 [43]，取代了臨時拼湊的比較。(4) **零樣本與少樣本異常偵測**。 WinCLIP 使用 CLIP，搭配狀態詞的提示集成與視窗層級的特徵，在沒有或只有極少正常影像的情況下分類並分割異常 [50]。(5) **更難的工業基準**。由於 MVTec AD 上的結果已經飽和，MVTec AD 2 加入了透明與重疊的物體、不尋常的照明、正常資料的高變異、非常小的瑕疵，以及含光線變化的測試情境，並報告最先進的方法離解決它仍然很遠 [41]。

尚未解決的問題。對任何基準都沒預料到的自然偏移保持穩健；超出小 ℓ_p 球之外的對抗穩健性（貼片、實體改變、語意編輯），以及在基礎模型規模上的對抗穩健性；在大影像上、有用的半徑下仍然夠緊的認證；骨幹在預訓練時可能什麼都見過的情況下的 OOD 偵測；能接受共變量偏移的 ID 輸入、同時拒絕語意新穎性的偵測器；以及一套能在同一個模型上測試本篇三個問題的共通評估協定。

重點整理

- 分布轉移、對抗穩健性與 OOD 偵測，需要不同測試條件及指標。
- 比較調適與泛化前，先說明可用的目標資料。
- 對抗主張須註明威脅模型、攻擊存取方式與擾動預算。

- 強攻擊或適應性攻擊有助揭露梯度遮蔽；憑證只涵蓋其明定假設。
- 在多個近／遠 OOD 集合評估分數並檢查污染，沒有普遍最佳分數。
- 異常偵測須區分影像層級決策與定位，並驗證正常資料假設。
- 在明確協定下比較乾淨資料表現、穩健性、操作門檻與成本。

練習

1. **用小批次或混合批次做測試批次正規化**。在損壞的程式碼中，改用 8 張測試影像組成的批次（而不是全部 540 張）做測試批次正規化，接著再對一個把三種損壞都以嚴重程度 3 混在一起的批次做一次。與一般模型和雜訊擴增模型相比，準確率有什麼變化？這對於把測試時適應部署在會在白天與夜間之間切換的相機上，說明了什麼？

   提示

   用 `np.array_split(Xs, len(Xs) // 8)` 把 `Xs` 切成區塊，每個區塊用自己的統計量正規化，再把預測串起來。在我們的執行中，嚴重程度 3 時，8 張一批保留了一部分增益（雜訊：整批 0.86，8 張一批 0.74，一般模型 0.56；模糊：0.95、0.87、0.82）。混合三種損壞的批次在這裡仍然有幫助（0.91，一般模型為 0.55），因為三種損壞都讓每個像素的統計量朝相近的方向改變（對比變低、分散變大）；一般情況下別期待這樣。小批次的統計量雜訊很大，而混合了截然不同條件（白天與夜晚）的批次，其統計量對兩者都不合用，所以真正的 TTA 方法需要來自同一種條件的、夠大的批次，或是能偵測資料流發生變化並重設的辦法。

2. **一個只弄壞梯度的防禦**。在對抗程式碼中加入一個「防禦」：標準模型看到每個輸入像素之前，先把它四捨五入到最近的 0.25 的倍數，並透過四捨五入重新計算輸入梯度（它幾乎處處為零）。FGSM 與 PGD 報告了什麼？接著改用 BPDA 再攻擊一次，只在反向傳遞時把四捨五入當作恆等函數。第一個結果顯示了評估一節中的哪些警訊？

   提示

   梯度為零時，`np.sign` 回傳零，PGD 就永遠不會離開它的隨機起點。在我們的執行中，「有防禦」的模型在 $\varepsilon = 0.15$ 的這個 PGD 下保持了 0.96 的準確率，而隨機的 $\pm\varepsilon$ 符號雜訊作為攻擊反而表現更好（0.93）：隨機取樣勝過梯度攻擊，是警訊之一。使用 BPDA 時，用 `input_grad(m, quantize(Xa), y)` 作為 `Xa` 處的梯度。在 $\varepsilon = 0.1$ 時，攻擊比對沒有防禦的模型弱（0.79 對 0.53），因為以 0.25 為間隔四捨五入，會抹掉小於半個間隔的擾動；在 $\varepsilon = 0.15$ 時，BPDA 讓準確率降到 0.01，低於沒有防禦模型的 0.18。這就是 Athalye 等人所說的破碎梯度情形。

3. **準確率–穩健性的取捨**。用訓練預算 $\varepsilon \in \{0.05, 0.1, 0.15, 0.2\}$ 訓練穩健模型，並畫出乾淨準確率對 $\varepsilon = 0.1$ 時 PGD 準確率的圖。曲線的拐點在哪裡？如果把隱藏層從 128 個單元增加到 512 個，會有什麼變化？

   提示

   在我們用 128 個隱藏單元的執行中，訓練預算從 0.05 增加到 0.2 時，乾淨準確率從 0.972 降到 0.941，而 $\varepsilon = 0.1$ 時的 PGD 準確率從 0.706 升到 0.754（在 0.1 與 0.15），然後降到 0.733：訓練預算遠大於測試預算，只會損失乾淨準確率而沒有進一步的收益。用 512 個隱藏單元時，每個預算下兩個數字都變好了（例如訓練預算 0.1 時，乾淨 0.980、穩健 0.806），符合 Madry 等人所說穩健性需要容量的發現。每種設定都要用相同的訓練輪數與相同的評估攻擊；確切數值會隨隨機種子略有不同。

4. **淡的瑕疵與更多正常資料**。在磚牆程式碼中，把刮痕強度從 0.5 改成 0.35，然後再把正常訓練裁切的數量從 16 增加到 32。報告每種情況下的影像層級與像素層級 AUROC，並解釋這兩個指標的差異。

   提示

   在我們的執行中，強度 0.35、16 張裁切時，影像層級的 AUROC 明顯下降（約 0.70），32 張裁切時部分回復（約 0.88），而兩種情況下像素層級的 AUROC 都維持在 0.99 以上。影像層級的分數取 4096 個區塊的最大值，所以會被最罕見的正常區塊拖累；更多正常資料會把更多這類區塊放進記憶庫。像素層級的 AUROC 彙整了數百萬個像素，幾乎都是正常的，即使有少數正常區塊分數很高，也能把大多數刮痕像素排在它們之前。預期結果會隨隨機種子有些變動，因為只有 40 張測試影像。

5. **選對工具**。對每個情境，說出它屬於三種穩健性問題中的哪一種、你會先跑哪一種評估、先試哪一種方法：(a) 一個皮膚病變分類器被搬到使用不同皮膚鏡的診所；(b) 使用者刻意用小幅影像編輯來繞過的內容過濾器；(c) 一個野外相機陷阱分類器，開始看到不在它標籤集裡的物種；(d) 一個沒有任何瑕疵焊接影像的焊接檢查相機。

   提示

   (a) 共變量（領域）偏移：從新診所收集有標註的測試集；試試正規化統計量適應，或用少數有標註的目標影像微調，並在相同的乾淨準確率下與更多樣的預訓練骨幹比較。(b) 對抗性，需要明確的威脅模型（允許哪些編輯？）；用白箱與轉移攻擊評估，並試試對抗訓練；對貼片類的編輯，ℓ_p 穩健性不夠。(c) 語意偏移：OOD 或開放集合偵測；用保留的物種評估 AUROC 與 FPR@95TPR，在好的骨幹上使用特徵距離分數。(d) 感官異常偵測：對正常焊接建立區塊記憶庫；用之後收集到的少量真實瑕疵，在影像與像素層級評估。

參考文獻

1. D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt and D. Song, “Natural Adversarial Examples,” *CVPR*, 2021. CVF
2. P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani et al., “WILDS: A Benchmark of in-the-Wild Distribution Shifts,” *ICML*, PMLR 139, 2021. PMLR
3. J. Yang, K. Zhou, Y. Li and Z. Liu, “Generalized Out-of-Distribution Detection: A Survey,” *International Journal of Computer Vision*, vol. 132, no. 12, pp. 5635–5662, 2024. doi ·arXiv
4. D. Hendrycks and T. Dietterich, “Benchmarking Neural Network Robustness to Common Corruptions and Perturbations,” *ICLR*, 2019. arXiv
5. D. Hendrycks, S. Basart, N. Mu, S. Kadavath, F. Wang, E. Dorundo et al., “The Many Faces of Robustness: A Critical Analysis of Out-of-Distribution Generalization,” *ICCV*, 2021. CVF
6. A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum and B. Katz, “ObjectNet: A large-scale bias-controlled dataset for pushing the limits of object recognition models,” *NeurIPS*, 2019. proceedings
7. B. Recht, R. Roelofs, L. Schmidt and V. Shankar, “Do ImageNet Classifiers Generalize to ImageNet?,” *ICML*, PMLR 97, 2019. PMLR

8. R. Taori, A. Dave, V. Shankar, N. Carlini, B. Recht and L. Schmidt, “Measuring Robustness to Natural Distribution Shifts in Image Classification,” *NeurIPS*, 2020. proceedings
9. J. Miller, R. Taori, A. Raghunathan, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon and L. Schmidt, “Accuracy on the Line: on the Strong Correlation Between Out-of-Distribution and In-Distribution Generalization,” *ICML*, PMLR 139, 2021. PMLR
10. D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer and B. Lakshminarayanan, “AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty,” *ICLR*, 2020. arXiv
11. R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann and W. Brendel, “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness,” *ICLR*, 2019. arXiv
12. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal et al., “Learning Transferable Visual Models From Natural Language Supervision,” *ICML*, PMLR 139, 2021. PMLR
13. A. Fang, G. Ilharco, M. Wortsman, Y. Wan, V. Shankar, A. Dave and L. Schmidt, “Data Determines Distributional Robustness in Contrastive Language Image Pre-training (CLIP),” *ICML*, PMLR 162, 2022. PMLR
14. M. Wortsman, G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs et al., “Robust Fine-Tuning of Zero-Shot Models,” *CVPR*, 2022. CVF
15. S. Schneider, E. Rusak, L. Eck, O. Bringmann, W. Brendel and M. Bethge, “Improving robustness against common corruptions by covariate shift adaptation,” *NeurIPS*, 2020. proceedings
16. D. Wang, E. Shelhamer, S. Liu, B. Olshausen and T. Darrell, “Tent: Fully Test-Time Adaptation by Entropy Minimization,” *ICLR*, 2021. arXiv
17. Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand and V. Lempitsky, “Domain-Adversarial Training of Neural Networks,” *Journal of Machine Learning Research*, vol. 17, no. 59, pp. 1–35, 2016. JMLR
18. K. Zhou, Z. Liu, Y. Qiao, T. Xiang and C. C. Loy, “Domain Generalization: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022. doi ·arXiv
19. I. Gulrajani and D. Lopez-Paz, “In Search of Lost Domain Generalization,” *ICLR*, 2021. arXiv
20. Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. V. Dillon, B. Lakshminarayanan and J. Snoek, “Can you trust your model’s uncertainty? Evaluating predictive uncertainty under dataset shift,” *NeurIPS*, 2019. proceedings
21. C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow and R. Fergus, “Intriguing properties of neural networks,” *ICLR*, 2014. arXiv
22. I. J. Goodfellow, J. Shlens and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” *ICLR*, 2015. arXiv
23. A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, “Towards Deep Learning Models Resistant to Adversarial Attacks,” *ICLR*, 2018. arXiv
24. F. Croce, M. Andriushchenko, V. Sehwag, E. Debenedetti, N. Flammarion, M. Chiang, P. Mittal and M. Hein, “RobustBench: a standardized adversarial robustness benchmark,” *NeurIPS Datasets and Benchmarks Track*, 2021. proceedings ·leaderboard
25. N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik and A. Swami, “Practical Black-Box Attacks against Machine Learning,” *ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pp. 506–519, 2017. doi
26. N. Carlini and D. Wagner, “Towards Evaluating the Robustness of Neural Networks,” *IEEE Symposium on Security and Privacy (SP)*, pp. 39–57, 2017. doi

27. D. Tsipras, S. Santurkar, L. Engstrom, A. Turner and A. Madry, “Robustness May Be at Odds with Accuracy,” *ICLR*, 2019. arXiv
28. H. Zhang, Y. Yu, J. Jiao, E. P. Xing, L. El Ghaoui and M. I. Jordan, “Theoretically Principled Trade-off between Robustness and Accuracy,” *ICML*, PMLR 97, 2019. PMLR
29. A. Athalye, N. Carlini and D. Wagner, “Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples,” *ICML*, PMLR 80, 2018. PMLR
30. F. Croce and M. Hein, “Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks,” *ICML*, PMLR 119, 2020. PMLR
31. J. Cohen, E. Rosenfeld and J. Z. Kolter, “Certified Adversarial Robustness via Randomized Smoothing,” *ICML*, PMLR 97, 2019. PMLR
32. K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno and D. Song, “Robust Physical-World Attacks on Deep Learning Visual Classification,” *CVPR*, 2018. CVF
33. T. B. Brown, D. Mané, A. Roy, M. Abadi and J. Gilmer, “Adversarial Patch,” arXiv:1712.09665, 2017. arXiv
34. D. Hendrycks and K. Gimpel, “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks,” *ICLR*, 2017. arXiv
35. W. Liu, X. Wang, J. D. Owens and Y. Li, “Energy-based Out-of-distribution Detection,” *NeurIPS*, 2020. proceedings
36. K. Lee, K. Lee, H. Lee and J. Shin, “A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks,” *NeurIPS*, 2018. proceedings
37. Y. Sun, Y. Ming, X. Zhu and Y. Li, “Out-of-Distribution Detection with Deep Nearest Neighbors,” *ICML*, PMLR 162, 2022. PMLR
38. D. Hendrycks, M. Mazeika and T. Dietterich, “Deep Anomaly Detection with Outlier Exposure,” *ICLR*, 2019. arXiv
39. J. Zhang, J. Yang, P. Wang, H. Wang, Y. Lin, H. Zhang et al., “OpenOOD v1.5: Enhanced Benchmark for Out-of-Distribution Detection,” arXiv:2306.09301, 2023. arXiv
40. P. Bergmann, M. Fauser, D. Sattlegger and C. Steger, “MVTec AD — A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection,” *CVPR*, 2019. CVF
41. L. Heckler-Kram, J.-H. Neudeck, U. Scheler, R. König and C. Steger, “The MVTec AD 2 Dataset: Advanced Scenarios for Unsupervised Anomaly Detection,” *International Journal of Computer Vision*, vol. 134, no. 4, 2026. doi ·arXiv
42. K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox and P. Gehler, “Towards Total Recall in Industrial Anomaly Detection,” *CVPR*, 2022. CVF
43. J. Bitterwolf, M. Müller and M. Hein, “In or Out? Fixing ImageNet Out-of-Distribution Detection Evaluation,” *ICML*, PMLR 202, 2023. PMLR
44. J. Liang, R. He and T. Tan, “A Comprehensive Survey on Test-Time Adaptation Under Distribution Shifts,” *International Journal of Computer Vision*, vol. 133, no. 1, pp. 31–64, 2025. doi · arXiv
45. N. Akhtar, A. Mian, N. Kardan and M. Shah, “Advances in Adversarial Attacks and Defenses in Computer Vision: A Survey,” *IEEE Access*, vol. 9, pp. 155161–155196, 2021. doi ·arXiv
46. G. Pang, C. Shen, L. Cao and A. van den Hengel, “Deep Learning for Anomaly Detection: A Review,” *ACM Computing Surveys*, vol. 54, no. 2, pp. 1–38, 2021. doi ·arXiv
47. J. Liu, G. Xie, J. Wang, S. Li, C. Wang, F. Zheng and Y. Jin, “Deep Industrial Image Anomaly

Detection: A Survey," *Machine Intelligence Research*, vol. 21, no. 1, pp. 104–135, 2024. doi ·arXiv
48. Z. Wang, T. Pang, C. Du, M. Lin, W. Liu and S. Yan, "Better Diffusion Models Further Improve Adversarial Training," *ICML*, PMLR 202, 2023. PMLR
49. J. Yang, P. Wang, D. Zou, Z. Zhou, K. Ding, W. Peng et al., "OpenOOD: Benchmarking Generalized Out-of-Distribution Detection," *NeurIPS Datasets and Benchmarks Track*, 2022. proceedings
50. J. Jeong, Y. Zou, T. Kim, D. Zhang, A. Ravichandran and O. Dabeer, "WinCLIP: Zero-/Few-Shot Anomaly Classification and Segmentation," *CVPR*, 2023. CVF