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 1 冊（共 4 冊）

基礎與影像處理

繁體中文版・第 1 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 1 冊（共 4 冊），主題為「基礎與影像處理」，收錄 15 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 基礎 1

從這裡開始・學習路徑與推薦教科書 2

影像就是陣列 12

II 影像處理 24

IP 01・緒論 25

IP 02・數位影像基礎 41

IP 03・強度轉換與空間濾波 61

IP 04・頻域濾波 83

IP 05・影像復原與重建 107

IP 06・色彩影像處理 131

IP 07・小波與其他影像轉換 153

IP 08・影像壓縮與浮水印 172

IP 09・形態學影像處理 194

IP 10・影像分割（一）：邊緣偵測、閾值化與區域偵測 215

IP 11・影像分割（二）：主動輪廓——蛇模型與水平集 235

IP 12・特徵擷取 254

IP 13・影像圖樣分類 275

篇 I

基礎

從這裡開始・學習路徑與推薦教科書

八個單元如何彼此銜接、依背景與目標該走哪條路線、如何執行程式碼與筆記本，以及每個單元適合搭配的教科書。

網站版本: https://blog.shyandram.dev/zh-tw/learn/foundations/getting-started/

你將學到

- 本站八個單元彼此的關係，以及各自的用途。
- 如何閱讀一篇教學頁面：難度、先備知識、單元側欄、筆記本與互動實驗。
- 依背景（剛學程式、數學生疏、已會訓練網路）與目標（視覺、理論、生成模型、語言、世界模型）選擇學習路徑。
- 如何建立一個小型 Python 環境，在 CPU 上執行程式碼區塊與筆記本。
- 每個單元適合搭配哪本主教材，以及教科書、綜述與論文各該怎麼讀。
- 如何把閱讀轉成可以檢驗的小里程碑。

先看全貌

你不必先學完所有數學才有資格碰影像，但你需要知道每個運算背後的假設。最快的學法，是從一個可以測試的問題開始：「為什麼模糊會削弱邊緣？」「分類器換到另一台相機的影像還有效嗎？」「模型改成預測下一個 token 而不是類別，會有什麼不同？」

> **用最簡單的話說**
>
> 學這個領域就像學做菜。教科書介紹食材與技巧，小實驗讓你嚐到實際的味道。先挑一道菜，一次只改一種食材，把觀察記下來，再去挑戰整桌宴席。本站就是一本把食譜依合理順序排好的食譜書，而且每道菜都能在小廚房（一台筆電的 CPU）裡做出來。

本站分成**八個單元**。單元是一系列依序排列、圍繞同一主題的教學。編號說明的是各單元的角色，不代表一定要讀完一個才能進入下一個。

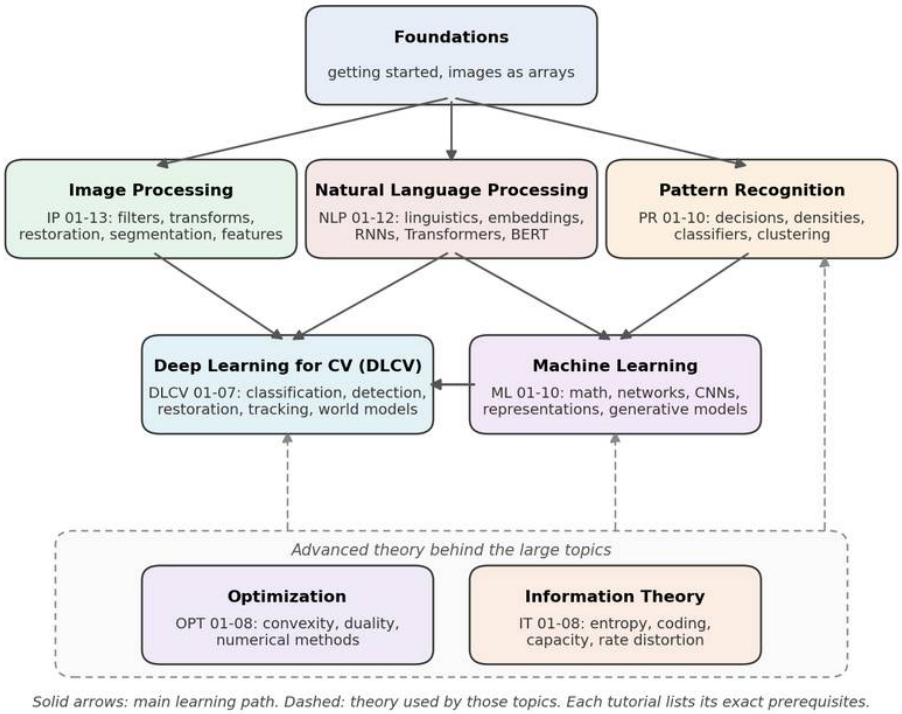


圖 *1* — 八個單元如何銜接。先讀基礎，再選一個入門單元（*IP*、*NLP* 或 *PR*）；機器學習與電腦視覺深度學習（*DLCV*）建立在它們之上；最佳化與資訊理論則是這些大主題背後的進階理論。箭頭只是建議，不是限制：每篇教學都會列出自己的先備知識。

單元	核心問題	內容	適合的第一個成果
基礎・起點	資料是什麼？	本頁與影像就是陣列	檢查影像的形狀、dtype 與座標慣例
影像處理（IP）・入門	如何變換或量測影像？	IP 01–13，從基礎到圖樣分類	解釋一個濾波器及其邊界行為
自然語言處理（NLP）・入門	文字與語言如何建模？	NLP 01–12，從導論與語言學基礎，經過詞嵌入、RNN、Transformer 與 BERT，到句法與語意任務	建立並評估一個小型語言應用
圖形識別（PR）・入門	如何用特徵在不確定性下做決策？	PR 01–10，從貝氏決策到辨識專題	用正確的訓練／測試切分比較分類器
機器學習（ML）・核心	表示、預測器與生成器如何被學出來？	ML 01–11、數學章節 ML 01A–01C，以及因子模型與自動編碼器的 ML 06A	訓練並評估一個小模型
電腦視覺深度學習（DLCV）・核心	學習式模型如何解決視覺任務？	DLCV 01–14：辨識、幾何、復原、生成、世界模型與穩健性	從頭到尾理解一條分類或偵測流程
最佳化（OPT）・理論	如何建立並求解帶限制的決策問題？	OPT 01–08，從凸集合到求解器與應用	建立凸最佳化問題並檢查最適條件
資訊理論（IT）・理論	表示與傳遞不確定性的極限是什麼？	IT 01–08，從熵到編碼、通道、率失真與網路	計算熵，說明壓縮或通訊的極限

用一段話說明它們的關係：**影像處理**把影像當成訊號，以人工設計運算；**圖形識別**加入統計與決策理論：有了特徵之後該怎麼決定、又能有多確定？**機器學習／深度學習**用學出來的特徵取代人工特徵，並提供各處共用的機制（損失函數、最佳化、架構、生成模型）；**視覺深度學習**把這套機制用在具體的視覺任務上，其中復原與增強章節大量借用影像處理的成像概念；**自然語言處理**則以文字為輸入，重複使用相同的機率、最佳化與 Transformer 概念。**最佳化**進一步發展估計與訓練背後的建模規則、限制、對偶性與數值方法。**資訊理論**解釋不確定性、編碼與通訊極限；熵與散度也串起分類、語言建模和壓縮。

教學頁面怎麼看

每篇教學都採用相同版面，看懂一頁就看懂全部。

- **難度標籤**。分為入門、中階、進階。ML 數學章節與前幾篇 IP 屬入門；DLCV 各篇則假設你已懂 CNN 與反向傳播。
- **先備知識**。標題下方的「先備知識」列出本頁預設你已讀過的教學。先備知識與語言無關，中英文頁面相同。遇到不熟的連結，先讀那一頁，至少讀它的「重點整理」。
- **單元側欄**。左欄依序列出目前單元的所有教學；頁尾的「上一篇／下一篇」也照同樣順序。
- **段落結構**。每篇以「你將學到」和「用最簡單的話說」開場，每個主題講兩遍：先用白話說，再給出含公式與程式碼的精確版本。結尾是「重點整理」、「練習」（提示預設收合，先自己試再打開）與編號的「參考文獻」，內文以 [1]、[2] 引用。
- **筆記本按鈕**。有配套筆記本的教學，標題旁會出現「在 Colab 開啟」，頁尾則有「查看筆記本」連結。目前 ML 03 有一本（卷積與影像濾波）。
- **互動實驗**。許多頁面內嵌即時實驗，在瀏覽器中對真實影像或資料集執行真正的演算法：濾波器、傅立葉轉換、門檻、最近鄰等等。每個實驗都先請你預測結果，再讓你親自驗證。所有實驗也都收錄在互動實驗頁面。
- **搜尋與語言**。搜尋頁會在你目前閱讀的語言中尋找教學與部落格文章；頁首的語言切換可跳到另一種語言的同一頁。若某頁尚未翻譯，中文網站會顯示英文版並附上提示。

選擇你的入口

挑一個最符合你的描述。各路線刻意有所重疊。

如果程式設計也是新主題。先熟悉 Python 的函式、迴圈、串列與陣列索引，以及基本繪圖。接著讀影像就是陣列與 IP 01。建立一張合成影像，檢查它的列與欄，並說明兩張 `uint8` 影像相加為什麼會溢位。之後讀 IP 02 與 IP 03，這兩章的每條公式都有對應的 NumPy 程式。

如果會寫程式，但數學有點生疏。先讀 ML 01 熟悉機器學習的詞彙，再讀三篇數學章節：線性代數、機率與資訊理論與數值計算。它們依循 Goodfellow、Bengio、Courville [7] 的「應用數學」部分，並為每個概念搭配簡短的 NumPy 驗證。目標是能檢查矩陣形狀、解讀條件機率，並認出數值不穩定的計算。

如果已學過入門機率與線性代數。想學統計決策與傳統方法，從 PR 01 開始；想學習式表示，從 ML 02 開始。推導遇到明確缺口時，再回頭補數學章節。

如果已經會訓練神經網路。快速瀏覽 ML 03 與 ML 05，再從視覺深度學習選一篇應用章節，並搭配 ML 11 的評估與部署一起讀。

依目標建議的學習路徑

有了入口之後，再選一個目標。每條路徑是一個閱讀順序，通常八到十二篇。讀每頁時留意「先備知識」，有缺口就隨時補上。

目標	建議順序
傳統影像處理	影像就是陣列 → IP 01–05（基礎、濾波、頻率域、復原）→ IP 06–09（色彩、小波、壓縮、形態學）→ IP 10–13（分割、主動輪廓、特徵、分類）
統計圖形識別	ML 01A、ML 01B → PR 01 → PR 02（密度估計）→ PR 03（線性模型）→ PR 04（核方法）→ PR 05（特徵）→ PR 07（分群、EM）→ PR 08（近似推論）→ PR 09（集成、驗證）→ PR 10
視覺深度學習	ML 01 → ML 01A–01C → ML 02 → ML 03 → ML 05 → DLCV 01 → DLCV 02 → DLCV 03 → 選讀 DLCV 09 復原或 DLCV 07 追蹤 → ML 11
生成模型	ML 01B → ML 02 → ML 06 → ML 06A → ML 07 → ML 08 → ML 09 → ML 10
語言	NLP 01 → NLP 02 → NLP 03–04（類神經網路、詞嵌入）→ NLP 05–06（RNN、序列標註）→ NLP 07–09（注意力與 Transformer、seq2seq、語言模型）→ NLP 10–11（BERT、分類）→ NLP 12；網路與序列模型基礎可搭配 ML 02 與 ML 04
凸建模與求解器	ML 01A、ML 01C → OPT 01 → OPT 02–04（函數、問題、對偶性）→ OPT 05–07（估計、數值方法、限制式求解器）→ OPT 08
資訊、壓縮與通訊	ML 01B → IT 01 → IT 02–03（典型性、來源編碼）→ IT 04–06（容量、高斯通道、率失真）→ IT 07–08（統計決策、網路）
世界模型	ML 04 → ML 05 → ML 07 → ML 08 → DLCV 13

若每週讀一篇、執行程式碼並做兩題練習，這是可以持續的步調；較長的教學可能需要兩週。速度不如能用自己的話說明每頁的重點整理重要。

執行程式碼

網站與 Python 範例使用不同的環境。只讀網站什麼都不用安裝；要執行程式碼需要 Python；要在本機預覽網站本身才需要 Node。

為教學建立 Python 環境

為所有教學建立一個專用的虛擬環境。Python 3.11 以上是實用的起點；若使用其他版本，請核對各套件的需求。

```
python3 -m venv .venv-study
source .venv-study/bin/activate
python -m pip install --upgrade pip
python -m pip install numpy scipy matplotlib scikit-learn scikit-image opencv-python
 ↪  PyWavelets jupyterlab
```

Windows PowerShell 的啟用方式是 `.venv-study\Scripts\Activate.ps1`。這些套件涵蓋入門陣列與傳統學習範例；神經網路章節另用 PyTorch，專門範例可能還會列出額外相依套件。請使用 PyTorch 官方安裝選擇器 [11] 產生的安裝指令；沒有 GPU 就選 CPU 版本。本站每個範例都設計成能在筆電 CPU 上於數秒到約一分鐘內跑完。

執行一個程式碼區塊

有些範例可獨立執行，有些則刻意接續同一章前面的區塊。例如影像就是陣列要先在第一段載入 `gray` 與 `rgb`，才能執行後面的切片範例；PR 10 綜合專題的主要流程要依序執行，後續才會沿用原本的開發／測試切分。開始新章節時使用新的筆記本核心，信任儲存的結果前先重新啟動並執行全部儲存格。名稱尚未定義，通常表示前面的設定區塊還沒跑，不應隨意換成另一份資料。提供的 `assert` 用來檢查特定主張，並不保證涵蓋所有輸入。

好習慣是一次只改一件事再重跑：kernel 大小、學習率、亂數種子。練習題會提示該調哪個參數。

筆記本與 Colab

有配套筆記本的教學會出現「在 Colab 開啟」按鈕。Colab 在瀏覽器中、以雲端主機執行筆記本，已預先安裝 NumPy、OpenCV、scikit-image 與 Matplotlib，本機不必做任何設定。若要在本機執行同一本筆記本，從「查看筆記本」連結下載，於已啟用的環境中執行 `jupyter lab` 並開啟檔案。筆記本放在 repository 的 `notebooks/` 資料夾。

預覽網站（選用）

只有想修改網站時才需要。本 repository 需要 Node 22.12 以上 [12]：

```
npm install
npm run dev
```

開發伺服器會印出本機網址。教學是 `src/content/tutorials/<lang>/<track>/` 底下的 MDX 檔案。

第一個 CPU 實驗

以下完整範例自行建立影像、加入雜訊再平滑，不需要下載任何東西。

```
import numpy as np
import cv2

rng = np.random.default_rng(0)
clean = np.zeros((32, 32), dtype=np.float32)
clean[8:24, 8:24] = 1.0
noisy = clean + rng.normal(0, 0.1, clean.shape).astype(np.float32)
smooth = cv2.GaussianBlur(noisy, (5, 5), 1.0,
                          borderType=cv2.BORDER_REFLECT_101)
print("Shape and type:", smooth.shape, smooth.dtype)
print("Noise-region standard deviation:",
      noisy[:5, :5].std(), smooth[:5, :5].std())
print("Across the edge:", np.round(smooth[16, 5:12], 3))
assert smooth.shape == clean.shape
assert np.isfinite(smooth).all()
```

輸出顯示兩個效果。平坦、全為背景的角落，標準差從約 0.074 降到約 0.026：平滑把雜訊平均掉了。在第 8 欄的邊緣處，數值不再從 0 直接跳到 1，而是在三、四個像素內從約 0.04 爬升到 0.93：平滑也把銳利的轉折擴散開了。因此雜訊估計變低，不等於所有重要細節都變好。IP 03 會解釋原因，練習 2 則請你實際量測這個取捨。

選一本主教材

用一本主教材維持連貫的進度，再用另一本提供同一概念的不同解釋。下列連結指向作者或出版社頁面，取得方式依各網站說明。這是精選的起點，不是排名。

需求	推薦教科書	建議用法
數學準備	Deisenroth、Faisal、Ong，*Mathematics for Machine Learning* [1]	從線性代數、矩陣分解、向量微積分與機率讀起，再透過迴歸與 PCA 應用
影像操作	Gonzalez、Woods，*Digital Image Processing*，第 4 版 [2]	把影像基礎、空間與頻率域濾波、復原與分割，搭配影像處理單元閱讀
圖形識別主教材	Theodoridis、Koutroumbas，*Pattern Recognition*，第 4 版 [3]	依序學分類器設計、特徵與分群；實際動手解題，而不只讀摘要
圖形識別配套教材	Bishop，*Pattern Recognition and Machine Learning* [4]	從機率、潛在變數與圖模型的角度重新理解同一主題
第一本現代深度學習書	Prince，*Understanding Deep Learning* [5]	搭配圖解、數學與隨書筆記本，理解網路與生成建模
另一種現代深度學習解釋	C. M. Bishop、H. Bishop，*Deep Learning: Foundations and Concepts* [6]	用它的機率觀點加深對架構與訓練原理的理解
深度學習基礎與參考	Goodfellow、Bengio、Courville，*Deep Learning* [7]	第一部分（應用數學）對應 ML 01A–01C；後續章節涵蓋最佳化、CNN、序列模型與表示學習
分類之外的電腦視覺	Szeliski，*Computer Vision: Algorithms and Applications*，第 2 版 [8]	串起成像、運動、幾何、辨識與重建

需求	推薦教科書	建議用法
機率式機器學習	Murphy，*Probabilistic Machine Learning: An Introduction* [9]	以機率為主軸的現代參考書，適合機器學習與圖形識別單元
自然語言處理應用主教材	Hagiwara，*Real-World Natural Language Processing* [13]	依序學習應用流程、文字分類、序列模型、注意力、預訓練模型、訓練實務與部署
自然語言處理語言學配套教材	McRoy，*Principles of Natural Language Processing* [14]	搭配對應章節學習句法、文法、語意、篇章、檢索、生成與翻譯
文字與語言補充教材	Jurafsky、Martin，*Speech and Language Processing*，第 3 版草稿 [10]	深入 token、n-gram、分類、嵌入（embedding）、Transformer 與評估等主題
凸最佳化	Boyd、Vandenberghe，*Convex Optimization* [15]	OPT 01–05 依第 1–8 章學建模與應用；OPT 06–08 深入第 9–11 章的數值方法與現代延伸
資訊理論	Cover、Thomas，*Elements of Information Theory*，第 2 版 [16]	從熵與典型性讀到來源／通道編碼、連續變數、率失真、統計決策、通用編碼與網路

圖形識別單元以 **Theodoridis 與 Koutroumbas [3] 及 Bishop 的 PRML [4] 為兩本核心教材**。先用 Theodoridis 建立完整的辨識流程，再比較 Bishop 對同一問題的機率處理方式。出版年份較早，不會讓貝氏決策理論或高斯模型失效；架構、資料規模或應用改變的部分，再用新論文補充。

若只選一本深度學習入門書，從 Prince [5] 開始，需要時搭配本站數學章節。偏好一路以機率觀點推導，就選 Bishop 與 Bishop [6]。Goodfellow 等人 [7] 仍是重要基礎，但 2016 年版早於視覺 Transformer、擴散模型與基礎模型；這些內容由機器學習／深度學習單元補上。

自然語言處理單元以 **Hagiwara [13] 與 McRoy [14] 為兩本核心教材**。Hagiwara 串起模型與應用，McRoy 補足理解語言系統所需的語言學與符號式概念。可以圍繞同一問題並讀兩書：分類器需要資料流程，剖析器或對話系統也需要明確的語言結構。兩書的 2021 年版再搭配新論文補充。

最佳化單元以 **Boyd 與 Vandenberghe [15]** 學習哪些問題變換保留凸性，以及最適性憑證的意義。資訊理論單元以 **Cover 與 Thomas [16]** 分辨可達成的構造與任何構造都無法突破的界限。兩個單元延伸 ML 01B–01C 的數學入門；每篇教學都有對應的教科書章節地圖。

Jurafsky 與 Martin 的草稿會持續更新，目前版本日期為 2026 年 8 月 19 日 [10]。做章節筆記時請記下版本日期，因為章號可能隨版本變動。其他教科書也要記下版次，讓練習題的出處能被重新找到。

教科書、綜述與論文怎麼讀

教科書提供定義與連貫的順序。每讀一段，至少重推一條公式、跑一個小例子、解一道練習，再往下讀。

綜述（survey ／ review）提供方法地圖、假設與未解問題。每篇教學的「現代觀點」都會摘要幾篇。閱讀時可整理成表格：它涵蓋的每個方法的預測目標、學習目標、所需資料、計算成本與評估方式。也要檢查涵蓋日期：「近期」是相對於作者當時而言。

原始論文提出一個方法及其證據。讀問題、方法、實驗規則與限制，把重要主張追到實際支持它的表格或消融實驗。知道它相對公平基準改變了什麼，比記住方法名稱更有用。

每篇閱讀筆記保留五項：問題、數學物件、主要假設、檢驗該假設的實驗，以及一個尚未解決的疑問。每篇教學的參考文獻是下一步閱讀的方向；不必讀完每篇引用的論文才開始跑程式。

用小里程碑安排進度

1. **表示資料**：檢查一張影像，說明各軸、數值範圍與 dtype。
2. **變換資料**：實作一個濾波器，與函式庫結果比對，並處理邊界。
3. **做出決策**：訓練一個簡單分類器，前處理參數只從訓練資料學。
4. **學習特徵**：訓練一個小型神經網路，說明損失函數與驗證方式的選擇理由。
5. **研究現代方法**：在合成資料上重現注意力、遮罩或加雜訊等機制。
6. **完成小專題**：報告基準方法、獨立測試結果、失敗案例與限制。

能不照抄原文、用自己的話說明上一個里程碑時，再往下走。一個比較乾淨的小實驗，比一個看不清資料與評估方式的大模型更能建立理解。

重點整理

- 八個單元：基礎、影像處理、自然語言處理、圖形識別、機器學習／深度學習、視覺深度學習、最佳化、資訊理論。它們互相支援，不必讀完一個才開始另一個。
- 每頁的「先備知識」、難度標籤與單元側欄告訴你它的位置；用它們取代從頭讀到尾。
- 依背景選入口、依目標選路徑；每週一篇並執行程式碼與練習，是可持續的步調。
- 範例以 CPU 學習為目標；有相依關係的區塊依章節順序執行，各章使用獨立筆記本，並依個別範例需求補裝套件。
- 每個單元搭配一本主教材與一本配套教材，並以不同目的閱讀教科書、綜述與論文。

練習

1. **規劃**。選一個入口、一條學習路徑與一本主教材。寫下第一個實驗要回答的問題，以及什麼具體結果算是進步。

   提示

   問題要小到能在 CPU 上一次做完，例如「5×5 高斯模糊在平坦區域降低多少雜訊？又讓邊緣變寬多少？」進步應該是一個可以量測的數字，而不是一種感覺。

2. **掃描 kernel 大小**。把平滑範例改成 kernel 大小 3、5、9（σ 跟著加大，例如 0.8、1.0、2.0）。回報平坦區域的標準差，以及邊緣從 0.1 升到 0.9 所需的像素數。

   提示

固定亂數種子與含雜訊的影像，只改濾波器。雜訊下降與邊緣變寬應該同時發生：這就是線性平滑的偏差與變異取捨。

3. **讀懂頁面資訊**。打開 ML 02。它的難度是什麼？「先備知識」指向哪一頁？單元側欄中下一篇是哪一篇？

   提示

   「先備知識」就在標題下方。沿著它一路往回追，直到遇到你已具備先備知識的頁面；這條鏈就是你個人的閱讀順序。

4. **為來源分類**。從任一篇教學挑一筆參考文獻，判斷它是教科書、綜述、原始論文還是軟體文件。這種來源最適合回答什麼問題？

   提示

   綜述回答「有哪些選擇、彼此如何比較？」；原始論文回答「這個特定改動是否有幫助、在什麼實驗規則下？」；軟體文件回答「這個函式實際上算了什麼？」

5. **用正確的單位切分**。有一個資料集包含 50 段影片、每段有許多影格。請規劃訓練／驗證／測試切分，並說明為什麼隨機切分單一影格可能造成誤導。

   提示

   相鄰影格幾乎相同，隨機切影格會讓近乎重複的樣本同時出現在訓練集與測試集，使測試分數虛高。應改以影片（或錄製時段）為單位切分。

參考文獻

1. M. P. Deisenroth, A. A. Faisal and C. S. Ong, *Mathematics for Machine Learning*, Cambridge University Press, 2020. Book site
2. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018. Book site
3. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009. Companion site
4. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006（官方提供 PDF）. Book page
5. S. J. D. Prince, *Understanding Deep Learning*, MIT Press, 2023. Book and notebooks
6. C. M. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, 2024（提供免費線上版）. Book site
7. I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016. Book site
8. R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., Springer, 2022（PDF 限個人使用）. Book site
9. K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022. Book site

10. D. Jurafsky and J. H. Martin, *Speech and Language Processing*, 3rd ed. draft, release of August 19, 2026. Draft site
11. PyTorch, “Get Started Locally,” installation documentation. Documentation
12. Node.js, “Download Node.js.” Download page
13. M. Hagiwara, *Real-World Natural Language Processing*, Manning, 2021. Publisher
14. S. McRoy, *Principles of Natural Language Processing*, open textbook, 2021. Open textbook
15. S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004. Author site and book
16. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, 2nd ed., Wiley, 2006. Publisher

連結與草稿日期於 2026 年 10 月 4 日查核。配套網站不代表整本教科書可免費取得；取得方式以各網站說明為準。

影像就是陣列

數位影像到底是什麼？NumPy 與 OpenCV 如何儲存它？以及能避開大多數常見錯誤的索引、資料型別與通道順序習慣。

網站版本: https://blog.shyandram.dev/zh-tw/learn/foundations/images-as-arrays/

你將學到

- 灰階與彩色影像如何變成 NumPy 陣列，以及 `shape`、`dtype` 與 `strides` 告訴你什麼。
- 為什麼影像是連續場景經過取樣與量化後的結果，以及保留較少像素或較少灰階時會失去什麼。
- 座標慣例（先列、後行），以及切片、翻轉、遮罩與攤平如何運作，包括視圖（view）與複本（copy）的差別。
- 每個人都會踩到的三個坑：BGR 與 RGB 的通道順序、`uint8` 回繞（wraparound），以及不小心透過視圖改寫原始資料。
- 同樣的陣列觀點如何延伸到深度學習的張量、影像區塊（patch）與神經網路影像表示。

先看全貌

本站所有方法，從 3×3 模糊到視覺 Transformer，起點都是同一個東西：一張數字網格。只要你能有把握地讀取、切片與轉換這張網格，其餘課程就只是選擇要組合哪些數字、如何組合。

> **用最簡單的話說**
>
> 想像一幅由許多小彩色磁磚拼成的馬賽克。站遠一點，你看到一張臉；走近一點，你只看到一塊塊方格，每格只有一種顏色。數位照片也是如此：一個長方形的磁磚陣列，每一塊磁磚用一個數字（或三個數字，分別代表紅、綠、藍）描述。要找到某一塊，你說「第幾列、第幾行」，就像在電影院找座位。電腦把這幅馬賽克存成一張數字表，而你對圖片做的每件事，例如調亮、模糊或找出人臉，都是對這張表做運算。

為什麼重要：初學者的影像程式錯誤，幾乎都是陣列的問題，而不是演算法的問題。顏色偏藍，是因為通道被對調了；明亮區域變黑，是因為 8 位元數值回繞了；修改裁切區域卻改到原圖，是因為裁切得到的是視圖而不是複本。本頁用可執行的程式把這些陷阱一次講清楚，讓你之後一眼就能認出來。

影像是取樣並量化後的訊號

白話版。真實世界沒有像素。相機測量每個感測器小區域收到多少光，再把測量結果寫成一個整數。兩個決定把連續場景變成數位影像：在哪裡測量（有幾個點），以及每個測量值寫得多細（有幾種可能的值）。

精確版。令單色場景是一個連續的亮度函數。**取樣**（sampling）在 H 列、W 行的規則網格上對它求值；**量化**（quantization）則把每個樣本四捨五入到 $L = 2^b$ 個整數階層之一，其中 b 是每個像素的位元數。結果是定義在有限網格上的函數：

$$I : \{0, \dots, H-1\} \times \{0, \dots, W-1\} \to \{0, \dots, L-1\}, \qquad (r, c) \mapsto I[r, c],$$

其中 r 是**列**（row）索引（從上往下數），c 是**行**（column）索引（從左往右數），$I[r, c]$ 是**像素值**，常見的 8 位元情況下 $L = 256$，數值從 0（黑）到 255（白）。彩色影像多一個通道索引 k，寫成 $I[r, c, k]$，三個色彩通道的 $k \in \{0, 1, 2\}$。Gonzalez 與 Woods [1, 第 2 章] 以及 Torralba、Isola 與 Freeman [2] 等教科書都以這種方式處理影像形成與數位化。

寫成陣列時，網格的**形狀**（shape）在灰階時是 `(H, W)`，彩色時是 `(H, W, 3)`。儲存的數值個數是 $H \times W \times C$（C 個通道），所需記憶體則是這個數量乘以每個數值的位元組數。

> **座標：先列後行**
>
> 陣列以 `[列, 行]` 索引，也就是畫面自己座標軸上的 `[y, x]`：第一個索引沿頁面往下，第二個索引沿頁面橫向。繪圖函式常把點寫成 `(x, y)`，也就是 `(行, 列)`。本系列的影像處理章節沿用教科書慣例，把 $f(x, y)$ 寫成 x 是列索引，所以 $f(x, y)$ 對應 NumPy 的 `f[x, y]`。每當程式與數學對不上，先確認哪個軸是哪個，再去除錯其他地方。

形狀、dtype 與記憶體

白話版。陣列知道自己的三件事：每個軸上有多少個數字（`shape`）、每個數字是哪一種（`dtype`），以及這些數字在記憶體中如何排列（`strides`）。

scikit-image 附帶幾張小型測試照片 [6]、[7]，因此本頁所有例子都不必下載任何東西就能執行。下面的程式載入一張灰階與一張彩色影像，並印出它們的基本性質。

```
import numpy as np
from skimage import data

gray = data.camera()          # grayscale test photo bundled with scikit-image
rgb = data.astronaut()        # color test photo, also bundled

for name, a in [("gray", gray), ("rgb", rgb)]:
    print(name, a.shape, a.dtype, a.ndim, a.size, a.nbytes)
print(gray.min(), gray.max())
print(gray.strides, rgb.strides)
```

輸出：

```
gray (512, 512) uint8 2 262144 262144
rgb (512, 512, 3) uint8 3 786432 786432
0 255
(512, 1) (1536, 3, 1)
```

解讀輸出：灰階影像有 512 列、512 行，所以 `size` 是 512 × 512 = 262,144 個數值；每個 `uint8` 佔一個位元組，所以 `nbytes` 是同一個數字。彩色影像的數值是它的三倍。像素範圍是完整的 0 到 255。

`strides` 是沿各軸前進一步時要跳過的位元組數。對 `gray` 而言，往右移一行花 1 個位元組，往下移一列花 512 個位元組，表示 NumPy 是**一列接著一列**儲存影像（C 順序）。在 `rgb` 中，同一個像素的三個通道值在記憶體中相鄰。這解釋了為什麼沿著列掃描很快，也解釋了為什麼把影像重塑成長向量時各列會維持在一起，下一節會看到。NumPy 論文 [3] 與文件 [4] 完整描述了陣列模型。

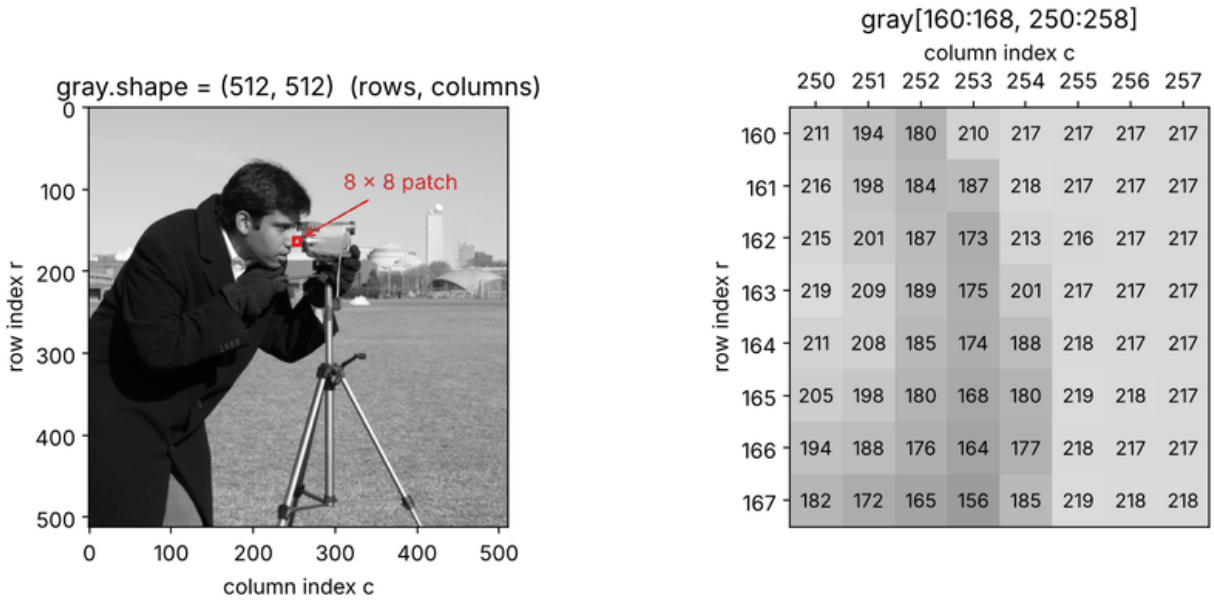


圖 *1* — 影像是一張整數表。左：*512 × 512* 測試照片，框出一個 *8 × 8* 區塊。右：同一個區塊，呈現為電腦實際儲存的數字，並標示列與行索引。*Matplotlib* 預設把第 *0* 列放在最上方 [*9*]。

索引、切片與視圖

白話版。因為影像就是陣列，你可以用與任何 NumPy 陣列相同的中括號語法取出單一像素、矩形區域或整列。令人意外的是，這樣取出的矩形是同一塊記憶體上的一扇窗，而不是獨立的複本。

精確版。對陣列 `a`，`a[r, c]` 回傳單一像素，`a[r0:r1, c0:c1]` 回傳第 $r_0, \dots, r_1 - 1$ 列與第 $c_0, \dots, c_1 - 1$ 行組成的子區塊（結尾索引不含），而 `a[::2, ::2]` 這類寫法則每隔一列、一行取一個。基本切片是**視圖**：與原陣列共用記憶體的陣列物件。若整數索引已選定每個軸，例如 `gray[r, c]`，回傳的是純量值；`rgb[r, c]` 則保留通道軸，因此回傳視圖。布林或整數陣列索引在讀取時回傳複本，但 `a[mask] = 0` 這種直接索引賦值仍會寫入 `a`。呼叫 `.copy()` 可以明確讓切片與來源分離 [13]。

```
r, c = 100, 200
```

```
print(gray[r, c])             # row first, then column
print(rgb[r, c])              # three numbers: R, G, B
patch = gray[100:108, 200:208]            # rows 100..107, columns 200..207
print(patch.shape, np.shares_memory(patch, gray))

work = gray.copy()
work[100:108, 200:208] = 0                # writing through a slice edits `work`
print(work[100:108, 200:208].max(), gray[100:108, 200:208].max())

H, W = gray.shape
vec = gray.reshape(-1)                    # flatten: row-major order
print(vec.shape, vec[r * W + c] == gray[r, c], np.shares_memory(vec, gray))
print(np.array_equal(gray[::-1], np.flipud(gray)), gray[:, ::-1].shape)
mask = gray > 200
print(mask.dtype, mask.sum())
```

輸出：

```
54
[81 57 17]
(8, 8) True
0 109
(262144,) True True
True (512, 512)
bool 55112
```

每一行告訴我們什麼：

- `gray[100, 200]` 是 54，`rgb[100, 200]` 是 `[81 57 17]`：彩色像素是一個短向量，每個通道一個數字。
- 8×8 區塊與 `gray` 共用記憶體，所以它是視圖。因為 `work` 是 `gray` 的複本，把 `work` 的區塊歸零不會動到 `gray`（它的區塊最大值仍是 109）。如果當初是對 `gray` 本身的視圖寫入，原始照片就會被改掉。
- 攤平後得到 $HW = 262144$ 個數字的向量，像素 (r, c) 位於位置 $rW + c$。這種攤平形式就是傳統分類器與全連接網路第一層看到的輸入向量（見 ML 01A 與 ML 01）。
- 把影像上下翻轉是 `gray[::-1]`，這是列步長為負的視圖，不需要迴圈。與 `np.flipud` 的比較結果是 `True`。
- 像 `gray > 200` 這樣的比較會得到同形狀的布林陣列，此處有 55,112 個 `True`。布林遮罩是你選取、計數或編輯符合條件像素的方式。

彩色影像與通道順序

白話版。彩色影像每個像素存三個數字，分別代表紅的程度、綠的程度與藍的程度。麻煩在於，兩種常見慣例對這三個數字的排列順序不同，而陣列本身不會告訴你它是哪一種。

精確版。多數 Python 影像程式碼（scikit-image [6] 的 NumPy 陣列、Matplotlib、Pillow）使用紅、綠、藍（**RGB**）順序。OpenCV 的 `imread` 回傳 **BGR**：藍色是通道 0，紅色是通道 2。混用

兩者不會報錯，只會悄悄把紅與藍對調，於是膚色變藍、天空變橘。為了不依賴磁碟上的檔案就能看見這個效果，下面的程式把彩色測試影像寫成暫存 PNG，再用 OpenCV 讀回來比較。

```
import pathlib, tempfile
import cv2
from skimage import io

with tempfile.TemporaryDirectory() as d:
    path = pathlib.Path(d) / "astronaut.png"
    io.imsave(path, rgb)                    # a PNG written in R, G, B order
    bgr = cv2.imread(str(path))             # OpenCV hands the channels back as B, G, R
print(rgb[100, 200], bgr[100, 200])
print(np.array_equal(bgr[..., ::-1], rgb))
print(np.array_equal(cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB), rgb))

weights = np.array([0.299, 0.587, 0.114])
manual = rgb @ weights                      # weighted sum of R, G, B -> float64 (H, W)
cv_gray = cv2.cvtColor(rgb, cv2.COLOR_RGB2GRAY)
print(cv_gray.dtype, np.abs(manual - cv_gray).max())
from skimage.color import rgb2gray
print(np.abs(rgb2gray(rgb) * 255 - cv_gray).max())
```

輸出：

```
[81 57 17] [17 57 81]
True
True
uint8 0.5010000000000332
15.529299999999992
```

同一個像素以 RGB 讀是 `[81 57 17]`，以 BGR 讀是 `[17 57 81]`。反轉最後一個軸 `bgr[..., ::-1]` 是不花成本的順序翻轉方式，而 `cv2.cvtColor` 搭配 `COLOR_BGR2RGB` 則是明確做同一件事。兩者都能完全還原原圖。

要把彩色轉成灰階，標準做法是對通道做**加權和**，因為人眼對綠色比對藍色敏感。權重 0.299, 0.587, 0.114 來自 ITU-R BT.601 電視標準的亮度（luma）定義 [8, 第 2.5.1 節]，而上面的檢查顯示，OpenCV 的灰階轉換與這個加權和的差距在 0.501 個灰階以內，那只是四捨五入成整數造成的。不同函式庫使用不同的權重：在這張影像上，scikit-image 的 `rgb2gray` 與 OpenCV 的結果最多相差約 15.5 個灰階。所以不要假設「灰階」在不同函式庫中是同一組數字，請對照你所呼叫函式的文件。



圖 2 — 彩色影像是三個平面的堆疊。每個通道本身就是一張灰階影像：橘色太空衣在紅色平面很亮、在藍色平面很暗。最後一個面板是加權灰階轉換。

資料型別與回繞陷阱

白話版。影像最常見的儲存型別是 8 位元無號整數，只能存 0 到 255。如果計算超過 255，它不會停在 255，而是悄悄從 0 重新開始。一個幾乎全白的像素可能變成幾乎全黑。

精確版。對無號整數型別 `uint8`，運算是以 $2^8 = 256$ 為模進行的。所以 $250 + 10 = 260 \equiv 4$，而 $250 - 251 = -1 \equiv 255$。由於數值不會飽和，在 `uint8` 中對影像做調亮或相減並不安全。NumPy 2 也不再允許陣列與放不進該陣列型別的 Python 整數混合運算：它會丟出錯誤，而不是悄悄擴大型別，這是依照 NEP 50 [5]。安全的做法有三個步驟：先轉成較寬的型別（`int16`、`float32`），再計算，最後裁切回有效範圍並轉回 `uint8`。

```
from skimage.util import img_as_float

a = np.array([250], dtype=np.uint8)
print(a + 10, a - 251, a.astype(np.int16) + 10)
try:
    a + 300
except OverflowError as e:
    print("OverflowError:", e)

wrapped = gray + np.uint8(60)
clipped = np.clip(gray.astype(np.int16) + 60, 0, 255).astype(np.uint8)
print((gray >= 196).sum(), (wrapped < gray).sum(), clipped.max())

x = gray.astype(np.float32) / 255.0
print(x.dtype, x.min(), x.max(), np.allclose(x, img_as_float(gray)))
y = np.clip(1.2 * x, 0, 1)
out = np.round(y * 255).astype(np.uint8)
print(out.dtype, out.max())
```

輸出：

```
[4] [255] [260]
OverflowError: Python integer 300 out of bounds for uint8
72087 72087 255
float32 0.0 1.0 True
uint8 255
```

解讀：在 `uint8` 中，`250 + 10` 得到 4，`250 - 251` 得到 255；在 `int16` 中，和則正確地是 260。對每個像素加 60，回繞的恰好是數值至少為 196 的像素（因為 $196 + 60 = 256$）：共有 72,087 個，而這同樣 72,087 個像素算完後反而比原來更暗。裁切版本則讓它們全部停在最大值 255。最後一段是常用的浮點流程：除以 255 以便在 $[0, 1]$ 中運算、計算、裁切、乘回去並四捨五入。`skimage.util.img_as_float` 對 `uint8` 影像做的是同樣的重新縮放。

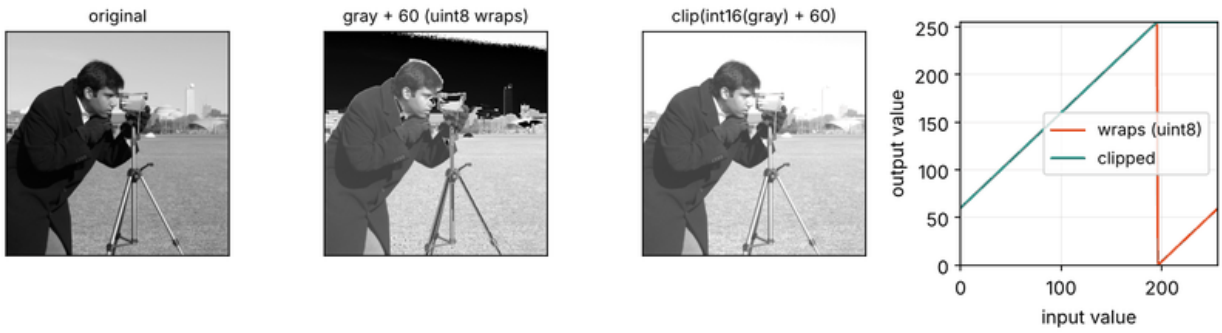


圖 3 — 調亮 *60* 個灰階。在 *uint8* 中，明亮的天空回繞成黑色（橘色曲線）。先轉成較寬型別、相加再裁切，則讓它飽和成白色（綠色曲線）。

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

先選資料型別，再做運算

溢位之後才轉型，無法恢復遺失的值。應先決定運算用的型別：濾波與學習用 $[0,1]$ 的 `float32`、需要精確整數差值用 `int16`，只有在存檔或顯示時才轉回 `uint8`。OpenCV 自己的運算函式（例如 `cv2.add`）會飽和而不是回繞，但 NumPy 的 `+` 運算子不會，所以要知道某一行程式用的是哪一個。上方的實驗在照片上執行同樣的運算，你可以親自試試每一種選擇。

在真實影像上做取樣與量化

白話版。兩個獨立的旋鈕控制場景能保留多少：像素的數量（空間解析度）與灰階的階數（亮度解析度）。降低前者，畫面變成方塊狀；降低後者，平滑的明暗過渡會裂成一塊塊平坦的色帶。

精確版。每隔 s 列、s 行取一個，也就是 `gray[::s, ::s]`，會把 $H \times W$ 的影像縮成約 $H/s \times W/s$ 個樣本。若事先沒有平滑，可能會產生**混疊**（aliasing）：較粗的網格無法表示的細節，不會消失，而是變成假的圖樣（正式處理見 IP 02 與頻率域各章）。把位元深度從 8 降到 b，就只剩 2^b 個相異數值。下面的輔助函式把每個像素四捨五入到所屬區間的中心，所以 4 位元時有 16 個允許的灰階。

```
def quantize(img, bits):
    step = 256 // 2 ** bits
    return (img // step) * step + step // 2    # replace each bin by its center

for bits in (8, 4, 2, 1):
    print(bits, len(np.unique(quantize(gray, bits))))

small = gray[::4, ::4]                          # keep every 4th row and column
print(gray.shape, "->", small.shape, small.nbytes, gray.nbytes)

def invert_loop(img):
    out = np.empty_like(img)
```

```
    for r in range(img.shape[0]):
        for c in range(img.shape[1]):
            out[r, c] = 255 - img[r, c]
    return out

print(np.array_equal(invert_loop(gray), 255 - gray))
chw = np.transpose(rgb, (2, 0, 1))
print(rgb.shape, "->", chw.shape, np.shares_memory(chw, rgb))
print(np.mean(rgb, axis=(0, 1)))
```

輸出：

```
8 256
4 16
2 4
1 2
(512, 512) -> (128, 128) 16384 262144
True
(512, 512, 3) -> (3, 512, 512) True
[141.56249237 105.75944519  96.47507477]
```

相異數值的個數以 2^b 下降。每隔 4 列、4 行取一個，會把影像從 262,144 位元組縮到 16,384 位元組，也就是記憶體的十六分之一。巢狀迴圈與單一運算式 `255 - gray` 的結果完全相同（`True`），但陣列運算式是以一次編譯過的運算執行，而不是二十六萬多次 Python 步驟：能寫成陣列運算式的影像程式就盡量這樣寫，把明確的迴圈留給檢查驗證用。轉置成通道優先 `(3, 512, 512)` 也是一個視圖，這是多數深度學習函式庫期望的排列。最後一行對兩個空間軸取平均，得到每個色彩通道一個平均值。

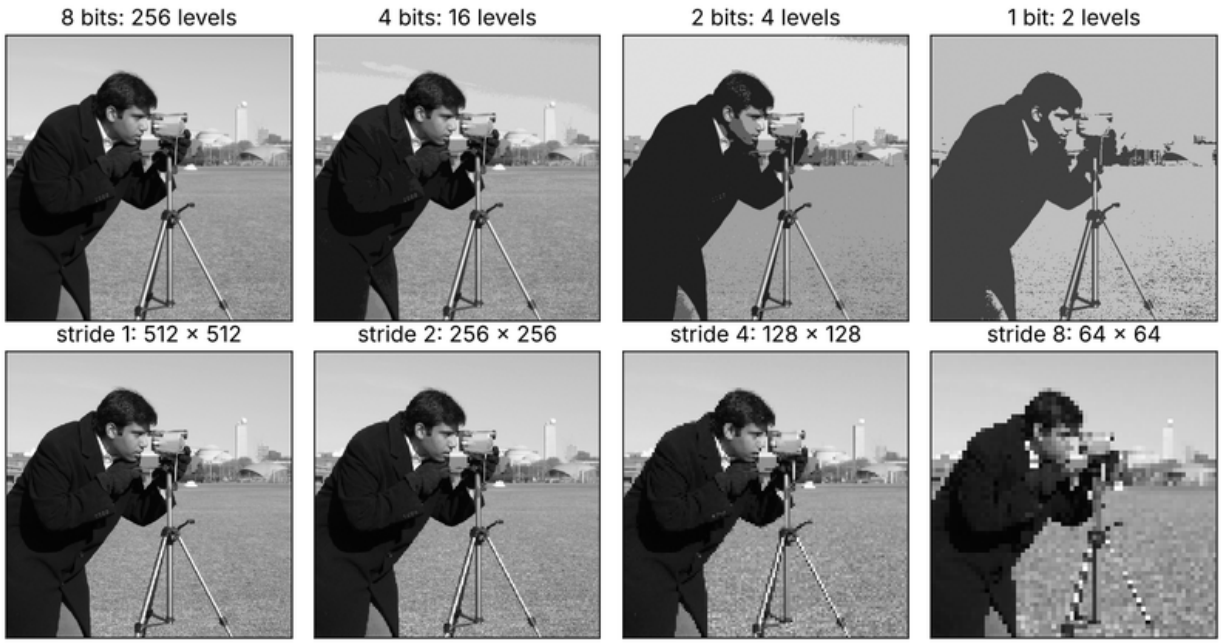


圖 4 — 兩種失去資訊的方式。上：較少灰階（量化）把平滑的明暗變成平坦的色帶。下：較少樣本（此處是單純點取樣，並以相同大小顯示）讓影像變成方塊狀，也破壞腳架腳這類細長結構。

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

反過來想也很有用。8 位元每通道通常足以顯示影像，但不一定足以處理影像：把一張偏暗又有色帶的影像拉伸對比，只會放大色帶。這就是為什麼存在每通道位元數更多的影像格式與感測器，也是為什麼處理流程會先轉成浮點數再做運算。上方的實驗讓你在照片與一張會暴露混疊的測試圖樣上同時調整這兩個旋鈕。

處理影像前的簡短檢查表

檢查	原因	一行程式
印出 `shape` 與 `dtype`	確認是灰階或彩色，以及數值範圍	`print(a.shape, a.dtype, a.min(), a.max())`
確認通道順序	RGB 與 BGR 看起來很像，直到顏色錯了	`rgb = bgr[..., ::-1]`
選擇運算用的 dtype	`uint8` 會回繞；[0,1] 的 `float32` 對濾波安全	`x = a.astype(np.float32) / 255`
編輯前先複製	切片是視圖，編輯會寫回原陣列	`b = a.copy()`
轉回前先裁切	超出範圍的浮點數轉成 `uint8` 會變成亂碼	`np.clip(y, 0, 1)`
記得 `[列, 行]`	繪圖 API 的座標對是 `(x, y)`	`a[y, x]`

現代觀點

本章沒有任何內容過時，因為深度學習框架把影像視為張量（tensor），而張量的行為就像 NumPy 陣列。改變的是排列方式、正規化，以及近年研究中對「影像是什麼」的想法。

張量與排列方式。訓練批次會多一個前導軸，成為 `(N, H, W, C)`，或許多函式庫期望的通道優先 `(N, C, H, W)`，就像上面用 `np.transpose` 得到的視圖。像素值通常轉成 `float32` 並正規化，例如除以 255 再對每個通道標準化。上面所有陷阱都原封不動地延續：以 RGB 輸入訓練的模型若被餵入 BGR，會悄悄變差；在 `uint8` 中回繞的前處理步驟，可能不報任何錯誤就毒化整個資料集。卷積與 Transformer 各章直接建立在這個觀點上：ML 03 與 ML 05。

影像視為區塊序列。 Dosovitskiy 等人的 Vision Transformer [10]（ICLR 2021）把影像切成固定大小區塊的網格，用本章的重塑方式把每個區塊攤平成向量，再把這個序列送進標準 Transformer。該論文報告，這類模型在大規模預訓練後，於影像分類基準上可以追平或超越卷積網路，同時需要較少的訓練運算量。對本章而言，重點在於陣列運算（切成區塊、重塑、轉置）就是模型的前端。

影像視為函式。第二個現代想法，是用學習得到的函式取代網格。Sitzmann 等人 [11]（NeurIPS 2020）提出正弦表示網路（sinusoidal representation networks），這是使用週期性活化函數的小型網路，輸入座標 (r, c)、輸出像素值，並展示它們能細緻地表示影像、音訊、影片及其導數。

Essakine 等人 [12]（TMLR 2025）對這個隱式神經表示（implicit neural representation）領域做了綜述：他們針對四類方法（活化函數、位置編碼、組合策略與網路結構最佳化）提出分類架構、做實驗比較，並以開放問題作結，例如更具表現力的活化函數、更好的位置編碼，以及對高維資料的可擴展性。陣列仍然是這類模型訓練與評估的方式，因為網路是擬合在取樣後的像素上；但解析度成為查詢函式時的屬性，而不是儲存表格的屬性。

哪些仍未解決、哪些已經確定。仍未定論的問題包括：如何在不同工具之間一致地儲存高動態範圍與廣色域色彩，以及在相同儲存成本下，學習得到的表示應如何與一般陣列比較。已經確定、值得練到自動反應的，就是本章這些：shape、dtype、通道順序、視圖與複本，以及先選運算型別再計算的習慣。

重點整理

- 數位影像是取樣並量化後的訊號，儲存成形狀 `(H, W)` 或 `(H, W, C)` 的陣列；$I[r, c]$ 表示第 r 列、第 c 行。
- `shape`、`dtype` 與 `strides` 描述陣列。一張 512 × 512 的 8 位元灰階影像有 262,144 個位元組；彩色版本是它的三倍。
- 切片回傳與原陣列共用記憶體的視圖。編輯前先用 `.copy()`，並且用陣列運算式取代逐像素迴圈。
- 各函式庫對通道順序（OpenCV 用 BGR，多數其他用 RGB）以及灰階轉換權重的看法不同；要檢查，不要假設。
- `uint8` 運算以 256 為模回繞。先轉成 `int16` 或 `float32`，計算、裁切，最後才轉回去。
- 空間解析度與灰階解析度是兩個獨立的選擇，降低任一個都會以不同方式失去資訊：方塊化與混疊，對上色帶。
- 深度學習張量、影像區塊與隱式神經表示，都建立在同一個影像陣列觀點上。

練習

1. **通道平均。**載入 `data.astronaut()`，用 `rgb.mean(axis=(0, 1))` 計算每個色彩通道的平均值。哪個通道最大？接著寫出同一張影像的 BGR 版本結果，並說明最大值現在位於哪個索引。

   提示

   這張影像的 R、G、B 平均值約為 141.6、105.8 與 96.5，所以紅色最大。在 BGR 陣列中，同樣的數字以相反順序出現，因此最大值在索引 2，而不是索引 0。

2. **用切片翻轉。**只用切片（不用 `np.flip`、不用迴圈），產生 `gray` 的上下翻轉、左右鏡像與旋轉 180° 的版本。分別與 `np.flipud`、`np.fliplr` 和 `np.rot90(gray, 2)` 比對。

   提示

   在你想反轉的軸上使用步長 -1：`gray[::-1]`、`gray[:, ::-1]` 與 `gray[::-1, ::-1]`。這三個都是視圖，所以如果要編輯，請先複製。

3. **找出回繞**。對本頁調亮 60 的例子，哪些輸入灰階值會回繞？輸入 200 會得到什麼輸出值？寫一行檢查程式，在不計算回繞後影像的情況下，計算被回繞的像素數。

   提示

   輸入為 196 以上的會回繞，因為 $196 + 60 = 256 \equiv 0$。輸入 200 得到 $260 \bmod 256 = 4$。計數是 `(gray >= 196).sum()`。

4. **安全的差值**。兩個 `uint8` 畫面 A 與 B 應以絕對差值 $|A - B|$ 比較。用一對像素值說明 `A - B` 會出錯，並寫出 NumPy 的安全版本。

   提示

   在 `uint8` 中，若 $A = 10$、$B = 20$，`A - B` 等於 246，而不是 10。安全版本是 `np.abs(A.astype(np.int16) - B.astype(np.int16)).astype(np.uint8)`。

5. **設計位元深度預算**。一張灰階影像以 8 位元儲存，再量化到 b 位元。若每個像素剛好壓縮成 b 位元，請預測 $b = 4$ 與 $b = 2$、512×512 時的相異數值個數與所需記憶體。為什麼圖 4 的 4 位元影像看得到色帶，而 8 位元看不到？

   提示

   有 2^b 個數值：16 與 4。壓縮儲存為 $512 \times 512 \times b/8$ 位元組，4 位元時是 131,072 位元組，2 位元時是 65,536 位元組。色帶出現在平滑漸層跨越兩個區間邊界的地方：8 位元時每一階只差 1 個灰階，在多數顯示器上低於人眼可分辨的程度，而 4 位元時每階相差 16 個灰階。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018 (Chapter 2, image sensing, sampling and quantization). publisher page
2. A. Torralba, P. Isola and W. T. Freeman, *Foundations of Computer Vision*, MIT Press, 2024 (open-access edition). book site
3. C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020. doi
4. NumPy developers, "Data types," NumPy user guide, current documentation. docs
5. NumPy developers, "NEP 50: Promotion rules for Python scalars," NumPy Enhancement Proposals. NEP 50
6. S. van der Walt et al., "scikit-image: image processing in Python," *PeerJ*, vol. 2, e453, 2014. doi
7. scikit-image developers, "skimage.data: test images and example data," scikit-image documentation. docs
8. 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 page
9. Matplotlib developers, "matplotlib.pyplot.imshow," Matplotlib documentation. docs

10. A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *International Conference on Learning Representations* (*ICLR*), 2021. arXiv:2010.11929
11. V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell and G. Wetzstein, “Implicit Neural Representations with Periodic Activation Functions,” *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 33, 2020. proceedings
12. A. Essakine et al., “Where Do We Stand with Implicit Neural Representations? A Technical and Performance Survey,” *Transactions on Machine Learning Research* (*TMLR*), 2025. arXiv:2411.03688
13. NumPy developers, “Indexing on ndarrays,” NumPy user guide, accessed 2026-11-05. documentation

篇 II

影像處理

IP 01・緒論

什麼是數位影像處理、它從哪裡來、它處理電磁波譜中的哪些訊號，以及本系列依循的基本步驟路線圖。

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

你將學到

- 什麼是數位影像——用文字說明，也用函數 $f(x,y)$ 表示——以及「處理」一張影像是什麼意思
- 從低階、中階、高階處理的角度，看影像處理在哪裡結束、電腦視覺從哪裡開始
- 這個領域的起源：用電報傳送的新聞照片、月球探測器，以及醫學掃描儀
- 我們處理的影像來自電磁波譜的哪些波段，以及哪些非光學的訊號
- 數位影像處理的基本步驟，同時也是本系列 13 章的路線圖
- 一套影像處理系統由哪些硬體與軟體組成，以及深度學習如何重塑相機的處理流程

先看全貌

> **用最簡單的話說**
>
> 想像一幅由許多小方磚拼成的大馬賽克。每塊磚只有一種灰色，並寫著一個數字：0 是黑色，255 是白色。電腦「看」不到這幅馬賽克，它只能讀那一長串數字。**數位影像處理**就是一整套改動這些數字的技巧，讓畫面變得更清楚、更小，或更容易理解：擦掉灰塵造成的斑點、把太暗的照片調亮，或是替桌上每一枚硬幣描出輪廓，好讓我們數清楚有幾枚。

今天你看到的幾乎每一張影像，在抵達你眼前之前都經過了影像處理。從光線打到手機感測器，到照片出現在螢幕上，手機已經執行了幾十道運算。醫院的電腦斷層掃描儀根本不「拍照」：它量測 X 光沿著成千上萬條直線被吸收了多少，再把影像算出來。氣象衛星記錄的是人眼看不見的能量，再把它變成你看得懂的地圖。

這一章是出發前的地圖。它界定這個領域、說明它的來歷、展示它能處理的訊號種類，並列出後續各章逐一介紹的步驟 [1]。本章沒有艱深的內容，但後面每一章都會用到我們在這裡建立的詞彙。

什麼是數位影像處理？

白話版。把一張圖切成由小方格組成的網格，每個方格存一個數字（彩色影像則存幾個數字），這張圖就變成了數位影像。處理的意思是：把這個數字網格交給電腦，得到某種有用的結果。

影像是一個函數

在數學上，灰階影像是一個二維函數

$$f(x, y), \qquad (x, y) \in \mathbb{R}^2.$$

其中 x 與 y 是空間座標，函數值 $f(x, y)$ 是影像在該點的**強度**（intensity），也稱為灰階值（gray level）。對相機而言，強度正比於抵達感測器的光能；對 X 光底片而言，則正比於穿透人體的輻射量。在真實世界中，x、y 與 f 都可以連續變化。

當這三個量都是有限且離散的，影像就是**數位影像**。我們在一個 $M \times N$ 的網格上取樣，並把每個值量化成 L 個等級之一：

$$f : \{0, 1, \ldots, M-1\} \times \{0, 1, \ldots, N-1\} \to \{0, 1, \ldots, L-1\}, \qquad L = 2^k.$$

M 是列數，N 是行數，k 是每個值所用的位元數（$k = 8$ 就得到熟悉的 0 到 255）。網格中的每個元素稱為**像素**（pixel，即 *picture element* 的縮寫）。第 2 章會說明取樣與量化該如何選擇；現在只要記得：影像就是一個矩陣。

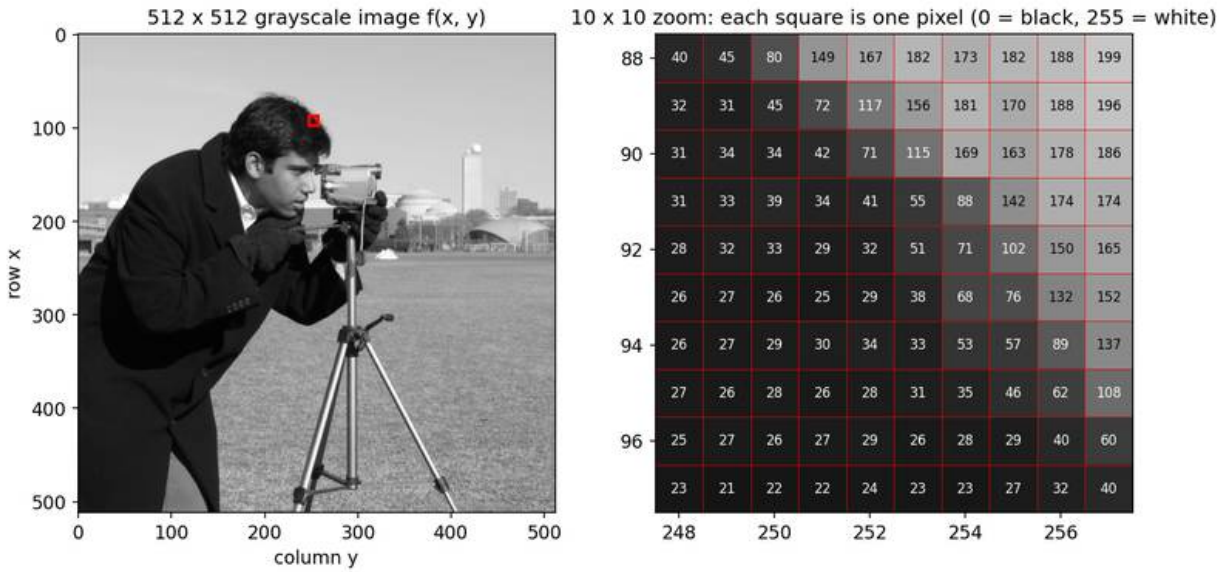


圖 *1.1* — 左：一張 *512 × 512*、*8* 位元的灰階影像。右：紅框內 *10 × 10* 區域的放大圖。每個方格就是一個像素，裡面的數字是它的灰階值。深色頭髮與明亮天空之間的「邊緣」，其實只是數值從 *30* 左右跳到 *180* 左右而已。

在程式中，這個矩陣就是一個 NumPy 陣列。用 `[列, 行]` 索引可以讀取單一像素，用切片則取得一塊鄰域：

```
from skimage import data

f = data.camera()            # a 512 x 512 grayscale photograph
print(f.shape, f.dtype)      # (512, 512) uint8
print(f[100, 250])           # intensity at row 100, column 250
print(f.min(), f.max())      # darkest and brightest values
```

```
patch = f[88:92, 248:252]    # a 4 x 4 neighborhood is just a sub-array
print(patch)
```

哪一軸是 x？

本系列沿用教科書的慣例 [1]：x 是列的索引（由上往下），y 是行的索引（由左往右），因此 $f(x,y)$ 正好對應 NumPy 的 `f[x, y]`。許多繪圖函式庫，以及先修教學〈Images as Arrays〉，用的是相反的順序（列 $= y$、行 $= x$）。當程式和數學對不上時，先檢查座標軸。

彩色影像會多一個通道索引，寫成 $f(x,y,c)$；醫學體積資料則多一個深度索引，寫成 $f(x,y,z)$。本章的觀念完全適用。

處理在哪裡結束，視覺從哪裡開始

影像處理在哪裡結束、影像分析或電腦視覺從哪裡開始，大家看法不一。一種極端的說法是：影像處理只包含輸入與輸出都是影像的運算。這個規則很乾淨，卻太狹隘——連計算一張照片的平均亮度這麼簡單的事都會被排除在外。另一個極端是電腦視覺，它的目標是模仿人類視覺：理解場景、辨識物體，並據此行動。電腦視覺的教科書把它視為一個獨立領域，核心是幾何、三維重建與辨識 [7]。

比較實用的看法，是把它想成一條連續光譜，上面有三種處理 [1]：

- **低階處理**：輸入影像，輸出影像。例如去除雜訊、提高對比、銳化。不涉及任何解讀。
- **中階處理**：輸入影像，輸出屬性（attributes）：區域、邊緣、輪廓，或對個別物體的量測。影像分割（把影像切成有意義的部分），以及把這些部分描述成電腦能分類的形式，都是典型的中階工作。
- **高階處理**：替一群已辨識出的物體「賦予意義」。讀懂一整頁文字，或判斷一張醫學影像裡有沒有腫瘤，都屬於高階工作。光譜的這一端，就是影像分析與電腦視覺交會之處。

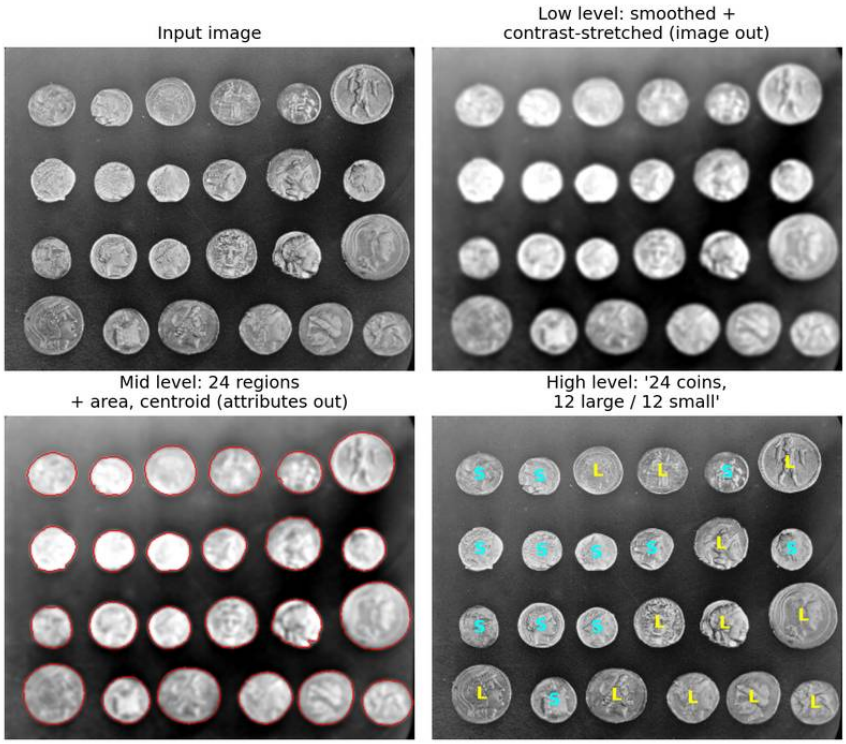


圖 *1.2* — 在同一張影像上看處理的連續光譜。低階：去雜訊並拉伸對比（輸出仍是影像）。中階：分割出硬幣並逐一量測（輸出是一串區域與數字）。高階：解讀這些量測值（輸出是一句話：共 *24* 枚硬幣，大的佔一半）。

在本系列中，「數位影像處理」涵蓋從低階一路到辨識個別區域或物體為止。這個範圍足以包含分割、特徵擷取與分類（第 10–13 章），但不包括完整的場景理解。

數位影像處理的起源

白話版。最早的數位圖片，是用打孔電報紙帶橫越大西洋傳送的新聞照片。真正用電腦處理影像始於 1960 年代，當時太空機構需要修正月球探測器傳回的影像；到了 1970 年代，能從量測資料「算出」影像的醫學掃描儀，讓這個領域蓬勃發展。

經由電報電纜傳送的圖片

1920 年代初期，**Bartlane 電纜圖片傳輸系統**透過橫越大西洋的海底電報電纜，在倫敦與紐約之間傳送新聞照片 [2]。照片先被轉換成打在一般五單位電報紙帶上的代碼，用標準電報設備傳送，再於接收端離線重組。根據 Kobayashi 對這套系統的歷史整理，直到 1939 年戰爭爆發為止，它一共橫越大西洋傳送了將近 500 張新聞照片 [2]。最早的 Bartlane 圖片只有五種灰階；到 1920 年代末，增加到十五種 [1]。

嚴格說來，這些圖片沒有經過任何電腦，所以這算是數位影像處理的史前史。但核心觀念已經出

現：把圖片轉成一組有限的代碼，傳送出去，再重建回來；而灰階等級的多寡，決定了結果有多逼真。



圖 *1.3* — 灰階數有多重要？同一張照片分別量化成 *5* 階（早期 *Bartlane*）、*15* 階（*1920* 年代末的 *Bartlane*）與 *256* 階（現代 *8* 位元）。只有 *5* 階時，大衣與草地這類平滑區域塌成一片片平坦色塊；即使有 *15* 階，天空中仍看得到一條條色帶。這種瑕疵稱為假輪廓（*false contouring*），第 *2* 章會解釋它。

幾行程式就能重現這個實驗。把強度 $r \in [0,1]$ 均勻量化成 L 個等級的映射為

$$Q_L(r) = \frac{\operatorname{round}\left(r\,(L-1)\right)}{L-1},$$

其中 round 表示四捨五入到最接近的整數，因此輸出只會是 $0, \frac{1}{L-1}, \ldots, 1$ 這 L 個值之一。

```
import numpy as np
from skimage import data

def quantize(img, levels):
    """Map a uint8 image onto `levels` evenly spaced gray levels."""
    x = img.astype(np.float64) / 255.0
    q = np.round(x * (levels - 1)) / (levels - 1)
    return (q * 255).astype(np.uint8)

f = data.camera()
for L in (5, 15, 256):
    g = quantize(f, L)
    print(L, "levels ->", len(np.unique(g)), "distinct values,",
          f"mean abs error {np.abs(g.astype(int) - f).mean():.2f}")
```

在這張測試影像上，平均絕對誤差從 $L = 5$ 時的約 18 個灰階，降到 $L = 15$ 時的約 5，再到 $L = 256$ 時的 0。

真正用電腦處理圖片，始於 1960 年代的太空計畫：人們用電腦修正月球探測器傳回影像中的失真，這成為數位影像處理的奠基計畫之一 [1]。

僅線上版：電腦遇上影像：太空計畫 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/introduction/notes/電腦遇上影像太空計畫/*

由量測算出的影像：電腦斷層

第二個奠基應用來自醫學。1973 年，Godfrey Hounsfield 描述了一套**電腦化橫向軸向掃描**（computerized transverse axial scanning）系統，也就是今天所說的電腦斷層掃描（computed tomography，CT）[4]。它不是只曝光一張底片，而是讓 X 光源與偵測器繞著病人旋轉，記錄光束沿著許多角度、許多直線被吸收的程度，再由電腦算出一個橫切面上每一點的吸收值。Hounsfield 指出，這能呈現傳統 X 光底片看不出的軟組織差異 [4]。

CT 之所以是里程碑，是因為影像不是拍出來的，而是重建出來的。掃描儀量測的是未知切面 $f(x, y)$ 的線積分：

$$p(s,\theta) = \iint_{\mathbb{R}^2} f(x,y)\,\delta(x\cos\theta + y\sin\theta - s)\,dx\,dy,$$

其中 θ 是光束的角度，s 是偵測器在投影方向上的位置，δ 是狄拉克 delta 函數，它只保留落在直線 $x\cos\theta + y\sin\theta = s$ 上的點。所有投影 $p(s,\theta)$ 合起來稱為**正弦圖**（sinogram）。從 p 還原出 f 是一個反問題（inverse problem），也是第 5 章後半「重建」的主題。這類演算法的標準測試物件，是 Shepp 與 Logan 在 1974 年提出、由多個橢圓組成的合成頭部切面 [5]。

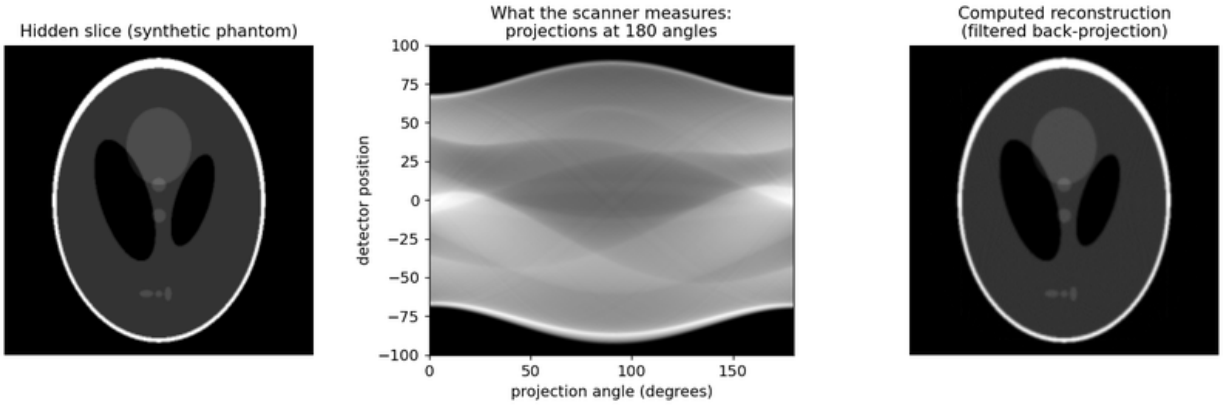


圖 *1.4* — 迷你版的電腦斷層。左：未知的切面（*Shepp–Logan* 假體 [5]）。中：掃描儀實際記錄的資料，每個角度一條投影。右：以濾波反投影（*filtered back-projection*）還原的切面，這個演算法會在第 *5* 章推導。

```
import numpy as np
from skimage.data import shepp_logan_phantom
from skimage.transform import radon, iradon, rescale

slice_ = rescale(shepp_logan_phantom(), 0.5)          # 200 x 200 synthetic head slice
angles = np.linspace(0, 180, 180, endpoint=False)
sinogram = radon(slice_, theta=angles)                 # what the scanner records
recon = iradon(sinogram, theta=angles, filter_name="ramp")
err = np.sqrt(np.mean((recon - slice_) ** 2))
```

```
print(sinogram.shape, recon.shape, f"RMS error {err:.3f}")   # (200, 180) (200, 200) ~0.025
```

從 1960 年代起，同樣的想法擴散到遙測、天文、生物、工業檢測、執法，最後走進每個人口袋裡的手機。所有這些應用背後有兩大目標：改善影像以便人來判讀，以及處理影像資料以便儲存、傳輸與機器感知。

使用數位影像處理的領域舉例

白話版。我們的眼睛只看得到「光」的一小段。其他種類的光，從伽瑪射線到無線電波，也都能形成影像；聲波和電子束也可以。影像處理能處理所有這些來源，因為它們最後都會變成一個數字網格。

整理應用最好的方式，是依影像的**來源**來分類 [1]。主要來源是電磁（electromagnetic，EM）能量；其他還有聲波、電子，以及純粹由計算產生的影像。

電磁波譜

電磁波可以用波長 λ 或頻率 ν 描述，兩者的關係為

$$\lambda = \frac{c}{\nu},$$

其中 $c \approx 2.998 \times 10^8$ m/s 是光速。光也以一包一包的**光子**（photon）形式存在，每個光子帶有能量

$$E = h\nu = \frac{hc}{\lambda},$$

其中 $h \approx 6.626 \times 10^{-34}$ J·s 是普朗克常數。波長短代表頻率高、光子能量大；波長長代表頻率低、光子能量小。光是這一點就能解釋成像的許多現象：高能光子能穿透組織與金屬（X 光），低能量的波則能穿透雲層與黑暗（雷達與無線電）。

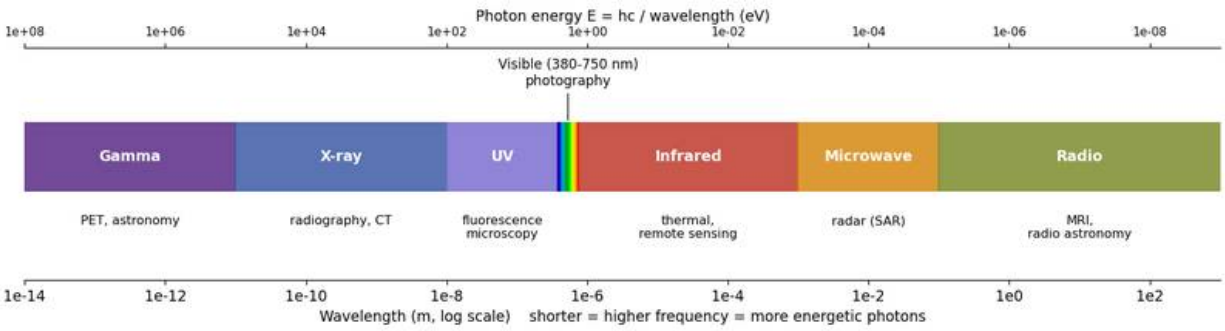


圖 *1.5* — 對數尺度下的電磁波譜，上方座標軸為光子能量。可見光只是其中細細的一條。各波段的分界只是慣例，不同資料來源略有差異。

```
h, c, eV = 6.626e-34, 2.998e8, 1.602e-19   # Planck, speed of light, joules per eV

def photon_energy_ev(wavelength_m):
    return h * c / wavelength_m / eV

for name, lam in [("X-ray (0.1 nm)", 1e-10), ("green light (550 nm)", 550e-9),
                  ("thermal IR (10 um)", 10e-6), ("radar (3 cm)", 0.03)]:
    print(f"{name:22s} {photon_energy_ev(lam):10.3g} eV")
# X-ray ~1.24e4 eV, green ~2.25 eV, thermal IR ~0.124 eV, radar ~4.1e-5 eV
```

一個 X 光光子的能量大約是綠光光子的一萬倍，而雷達光子的能量還不到綠光光子的萬分之一。以下各波段依能量由高到低排列。

從伽瑪射線到無線電波

白話版。每個波段擅長呈現不同的東西，也各自帶有不同的雜訊。以下依光子能量由高到低快速走一遍：

- **伽瑪射線：**核子醫學（影像呈現放射性示蹤劑聚集在哪裡，顯示的是功能而不是解剖構造）與天文學。由於每個像素只累積到有限數量的光子，影像雜訊很多。
- **X 光：**醫學上的 X 光攝影、血管攝影與 CT [4]；工業上檢查焊接點與電路板裡看不見的瑕疵。
- **紫外線：**螢光顯微術，被標記的構造會在黑色背景上發亮。
- **可見光與紅外線：**攝影、顯微術、機器視覺與多光譜遙測；熱紅外線感測的是物體放出的熱，而不是反射光。
- **微波：**成像雷達，自帶照明，能穿透雲層與黑暗 [8]。
- **無線電波：**醫學上的 MRI [6]，以及電波天文學——第一張黑洞陰影的影像，就是結合許多望遠鏡的資料重建出來的 [9]。

要記住的一點：CT、MRI、雷達與電波天文學都不是直接拍下影像，而是從量測資料算出影像。

僅線上版：逐一走訪電磁波段 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/introduction/notes/逐一走訪電磁波段/*

其他成像方式

並非所有影像都來自電磁輻射。

- **聲波成像**。聲波會在不同材料的交界處反射。地質學家把低頻聲波送入地底並記錄回波，藉此描繪地下岩層，用於石油、天然氣與礦產探勘（震測成像，seismic imaging）。醫學**超音波**使用高頻聲波（每秒數百萬次振動），能即時、且不使用游離輻射地拍攝胎兒、心臟與腹部器官。這兩種情況都是靠回波的時間來形成影像。
- **電子顯微術**。電子可以像光一樣被聚焦，但它的等效波長短得多，因此電子顯微鏡能解析比光學顯微鏡細得多的細節。穿透式電子顯微鏡讓電子束穿過薄標本；掃描式電子顯微鏡則讓聚焦的電子束掃過表面，記錄被反彈或撞擊出來的電子。這類影像常帶有雜訊，需要強化與復原。

- **合成影像**。電腦也會產生沒有任何感測器拍過的影像：碎形、算繪出來的三維場景、模擬結果，以及越來越多由生成模型產生的影像。這些影像用同一套工具處理，也大量用來製作訓練資料與測試案例。

這趟巡禮的重點是：一旦資料變成數字網格，同一套工具就都適用，不論這些數字來自伽瑪射線、聲波回音或電腦程式。真正改變的是雜訊、解析度與成像的物理原理，而這些差異決定了你該拿出哪一種工具。

數位影像處理的基本步驟

白話版。讓一張影像派上用場就像做菜：先備料（擷取影像），再清洗、處理（強化、復原），有時改變食材的形態（色彩、轉換、壓縮），然後切塊（分割），描述每一塊，最後判斷每一塊是什麼（分類）。不是每道菜都需要每個步驟。

把這些步驟分成兩組會比較好理解 [1]：輸出通常是**影像**的方法，以及輸出通常是從影像擷取出的**屬性**的方法。關於問題領域的**知識庫**（knowledge base）——例如瑕疵通常出現在哪裡、什麼樣的標籤才合理——引導著每一個步驟。這份清單並不是非走不可的流程；實際應用只會用到需要的步驟，並依合理的順序排列。

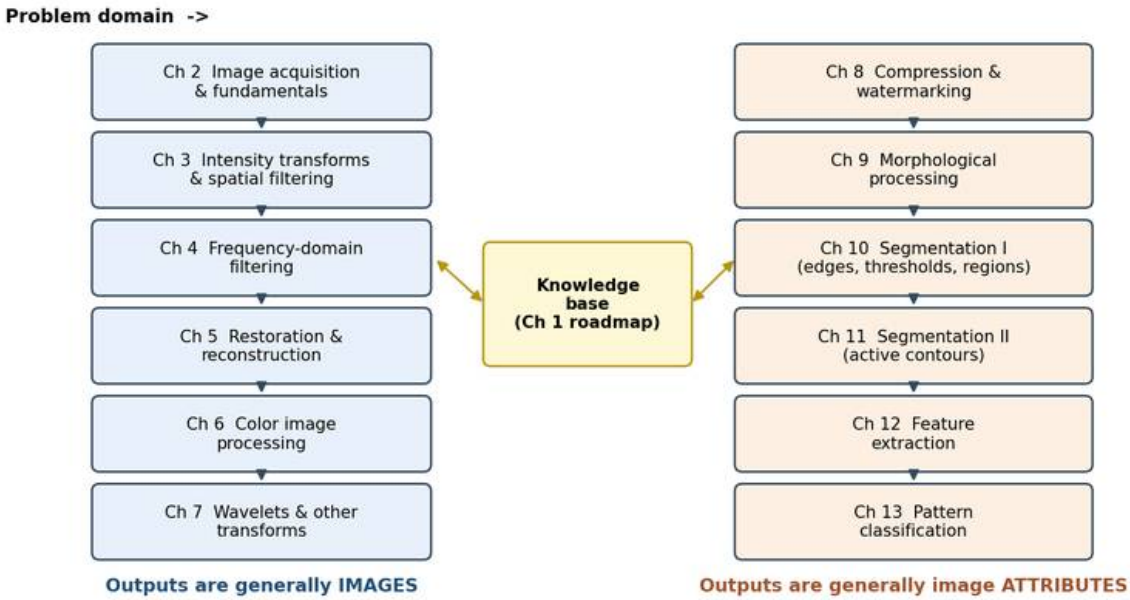


圖 *1.6* — 數位影像處理的基本步驟，標上本系列中負責介紹它們的章節。左側步驟的輸出通常是影像，右側步驟的輸出通常是屬性。中央的領域知識影響所有步驟。

以下逐一介紹各步驟，以及負責的章節。

1. **影像擷取**（image acquisition）——取得數位形式的影像，包括感測、取樣與量化。第 2 章也會建立之後處處用到的數學工具（像素間的關係、影像算術、幾何轉換）。→ IP 02・數位影像基礎

2. **影像強化**（image enhancement）——讓影像更適合特定用途，例如提高對比或銳化細節。強化是主觀的：「更好」取決於觀看者與任務。→ IP 03・強度轉換與空間濾波 與 IP 04・頻域濾波
3. **影像復原**（image restoration）——消除已知或估計出的劣化，例如模糊或雜訊。和強化不同，復原是客觀的：它依賴描述影像如何劣化的數學模型。同一章也介紹由投影重建影像（CT）。→ IP 05・影像復原與重建
4. **彩色影像處理**（color image processing）——色彩模型、虛擬色彩，以及全彩影像的處理。→ IP 06・彩色影像處理
5. **小波與其他影像轉換**——同時以多種解析度表示影像，是現代壓縮與金字塔方法的基礎。→ IP 07・小波與其他影像轉換
6. **壓縮與浮水印**——減少影像所需的儲存空間或頻寬，以及嵌入看不見的資訊以保護所有權。→ IP 08・影像壓縮與浮水印
7. **形態學處理**（morphological processing）——擷取與整理形狀成分的工具，例如細線化、填補孔洞、分開相黏的物體。從這裡開始，輸出逐漸變成屬性。→ IP 09・形態學影像處理
8. **影像分割**（segmentation）——把影像切分成組成它的部分或物體。這是最困難的步驟之一，這裡的錯誤會延續到之後的每一步。→ IP 10・影像分割（一）（邊緣、閾值、區域）與 IP 11・影像分割（二）（主動輪廓：snakes 與 level sets）
9. **特徵擷取**（feature extraction）——把分割出的區域轉成數字：邊界與區域描述子，以及角點這類點特徵。它包含兩部分：偵測特徵與描述特徵。→ IP 12・特徵擷取
10. **影像樣式分類**（image pattern classification）——根據物體的特徵替它貼上標籤，從最小距離分類器一路談到類神經網路與深度學習。→ IP 13・影像樣式分類

第 1 章（也就是本頁）是把它們串在一起的總覽。

下面的程式把其中四個步驟串起來，用在圖 1.2 的硬幣影像上。每個步驟在這裡只需一兩個函式呼叫；後面的章節會解釋每一步內部在做什麼。

```
from scipy import ndimage as ndi
from skimage import data, filters, measure, segmentation

img = data.coins()

# 1) enhancement: low level, image in -> image out
smooth = filters.gaussian(img, sigma=1, preserve_range=True)

# 2) segmentation: mid level, image in -> regions out
markers = (smooth < 30) * 1 + (smooth > 150) * 2       # sure background / sure coin
regions = segmentation.watershed(filters.sobel(smooth), markers) == 2
regions = ndi.binary_fill_holes(regions)
labels = measure.label(regions)

# 3) feature extraction: regions in -> numbers out
props = [p for p in measure.regionprops(labels) if p.area > 300]
areas = [p.area for p in props]

# 4) classification / interpretation: numbers in -> a statement out
median = sorted(areas)[len(areas) // 2]
n_large = sum(a >= median for a in areas)
print(f"{len(areas)} coins found, {n_large} at or above the median size")
# 24 coins found, 12 at or above the median size
```

注意藏在這段程式裡的知識庫：閾值 30 與 150、最小面積 300，以及「大於等於中位數就算大」這條規則，全都編碼了這個特定問題的知識。換了光源或換了硬幣，這些數字就得跟著改。這個領域的大半歷史——包括深度學習的崛起——說穿了，就是把這類知識從手動設定的常數，搬進從資料中學出來的模型。

影像處理系統的組成元件

白話版。一套影像處理系統就像一支團隊：感測器負責接住影像，快速的專用晶片負責粗重的運算，電腦執行程式，記憶體存放圖片，螢幕讓人觀看，印表機輸出紙本，網路把一切連在一起。

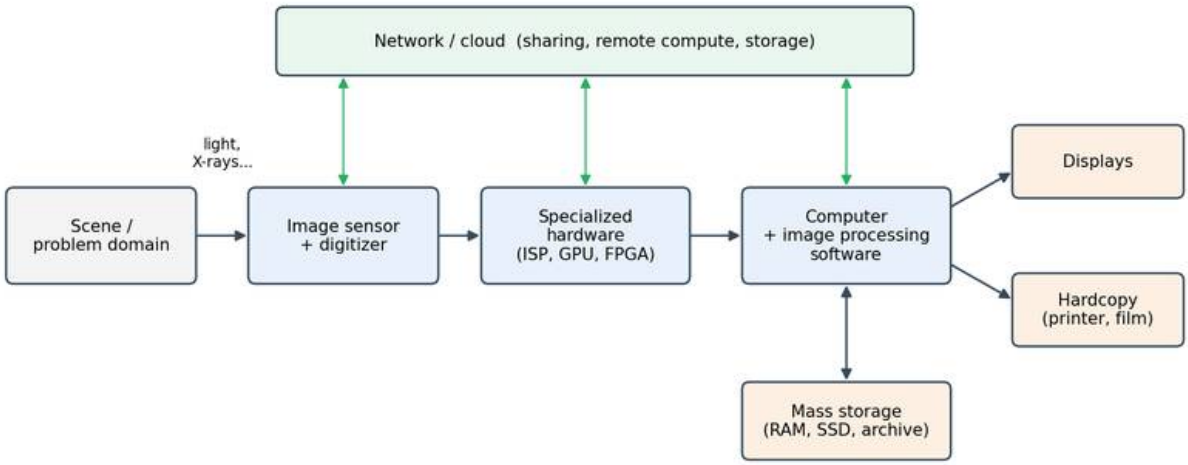


圖 *1.7* — 通用影像處理系統的組成元件。箭頭表示主要的資料流向；網路把每個階段連到遠端的儲存與運算資源。

- **影像感測器**。擷取影像需要兩個元件：一個會對物體輻射出的能量產生反應的實體裝置（CCD 或 CMOS 晶片、X 光偵測器、超音波探頭），以及把裝置的電訊號轉成數字的**數位轉換器**（digitizer）。在手機中，兩者做在同一顆晶片上。
- **專用影像處理硬體**。有些運算必須以影片速率對每個像素執行，比通用處理器能負擔的還快。早期系統使用專門的算術運算板；今天這個角色由每顆相機晶片裡的影像訊號處理器（image signal processor，ISP）、繪圖處理器（GPU）、現場可程式化邏輯閘陣列（FPGA），以及手機中的神經網路處理器擔任。
- **電腦**。從智慧門鈴裡的嵌入式微控制器，到整個伺服器叢集都有可能。專用任務的電腦可以量身訂做；研究與一般用途則一台普通工作站就夠了。
- **軟體**。執行特定任務的模組，再加上把它們組合起來的方法。本系列使用 Python 搭配 NumPy、SciPy、scikit-image 與 OpenCV。本系列希望培養的，是自己寫程式的能力，而不只是呼叫現成的函式。
- **大量儲存裝置**。影像很大，而且大批湧入。儲存分成三層：處理期間的短期儲存（RAM、GPU 記憶體、影格緩衝區）、可快速取用的線上儲存（SSD 與磁碟陣列），以及很少存取的封存儲存（磁帶、冷儲存雲端服務）。

- **影像顯示器**。今天大多是彩色平面顯示器。醫學診斷使用經過校正的專用螢幕；立體視覺或虛擬實境則使用頭戴式顯示器。
- **硬拷貝裝置**。雷射與噴墨印表機、底片相機、熱感印表機。底片仍提供極高的解析度，而紙本依然是書面資料與許多報告的首選媒介。
- **網路與雲端**。幾乎每套系統都已連網。傳輸影像是頻寬問題，這正是壓縮（第 8 章）是核心主題的原因。雲端服務把儲存與繁重的運算（例如訓練深度模型）從裝置上移走。

「很大」到底有多大？一張有 M 列、N 行、C 個通道、每個值 k 位元的影像，在壓縮前需要

$$b = M \times N \times C \times k \text{ bits}$$

其中灰階影像 $C = 1$，RGB 彩色影像 $C = 3$。

```
def raw_size_mb(height, width, channels=1, bits=8, frames=1):
    return height * width * channels * bits * frames / 8 / 1e6

print(f"{raw_size_mb(512, 512):.2f} MB   - this chapter's 8-bit test photo")         # 0.26
print(f"{raw_size_mb(3000, 4000, 3):.0f} MB   - one 12-megapixel RGB photo")         # 36
print(f"{raw_size_mb(512, 512, bits=16, frames=300):.0f} MB  - a 300-slice 16-bit CT volume")
 ↪  # 157
print(f"{raw_size_mb(2160, 3840, 3, frames=30*60):.0f} MB - one minute of uncompressed 4K
 ↪  video")  # 44790
```

一分鐘、每秒 30 影格、未壓縮的 4K 影片大約是 45 GB。沒有壓縮，串流影片與充滿圖片的網頁都將窒礙難行。

現代觀點

教科書把這個領域看成一條由理解透徹、人工設計的步驟組成的流程，這仍然是學習影像處理的正確方式。但大約 2010 年以後，有兩項發展改變了這個領域的實務：計算被搬進了成像過程本身，而學習出來的模型取代或強化了許多人工設計的步驟。以下的綜述論文是很好的入門。

計算成像與可見光以外成像的綜述

- **Mait、Euliss 與 Athale，〈Computational imaging〉（2018）** [8] 回顧了二十年來光學與計算一起設計的系統：感測器記錄下的資料，要經過處理才會成為認得出來的圖片。這篇綜述涵蓋空間光調變器、合成孔徑雷達與計算顯微術等主題。主要結論是：圖 1.6 中「擷取」與「處理」的分界，如今是一個設計選擇——CT、MRI 與雷達是早期的例子，而手機讓這種做法成為主流。
- **Delbracio、Kelly、Brown 與 Milanfar，〈Mobile computational photography: a tour〉（2021）** [10] 追溯手機相機如何靠演算法克服微小的感測器與鏡頭：連拍多張影格再合併、強力的降噪，以及多影格超解析度。這趟巡禮顯示，現代手機照片其實是一條長長處理流程的輸出，它幾乎可以當作本系列每一章的搭配讀物。

- **Gallego 等人，〈Event-based vision: a survey〉（2020）**[11] 介紹一種完全打破圖 1.1 那種「影格」模型的感測器。事件相機（event camera）的每個像素在自己的亮度改變時就非同步地觸發，提供微秒等級的時間解析度與非常高的動態範圍。這篇綜述說明，當輸入是一串事件而不是一個數字網格時，特徵偵測、光流、影像重建等經典任務都必須重新思考。
- **Wang、Ye 與 De Man，〈Deep learning for tomographic image reconstruction〉（2020）**[12] 回顧類神經網路如何進入 CT、MRI、PET 等成像方式——正是 1970 年代開啟這個領域的那些重建問題。內容涵蓋清理傳統重建結果的網路、把量測模型保留在流程中的物理引導（physics-informed）方法、直接學習的重建，以及生成模型，目標包括更快的掃描、更低的輻射劑量與更好的影像品質。作者的展望強調：大型、經過妥善整理的資料集將是未來進展的關鍵。

深度學習如何重塑處理流程

LeCun、Bengio 與 Hinton（2015）的回顧文章 [13] 概括了這個大轉變：深度網路直接從範例中學習多層次抽象的資料表示，而不再依賴人工設計的特徵。在影像處理中，這個轉變在圖 1.2 的連續光譜各層級上，呈現出不同的樣貌。

在**高階**，改變幾乎是全面的。分類與辨識（第 13 章）如今由學習模型主導；第 12、13 章會介紹在那之前的人工特徵與分類器，它們在資料稀少或重視可解釋性時依然有用。

在**低階**，最清楚的案例是相機自己的處理流程。傳統 ISP 是一串分別調校的階段，把感測器的原始數值變成完成的照片：黑階校正、去馬賽克（demosaicing）、去雜訊、白平衡、色彩校正、色調映射與壓縮。**Ramanath 等人（2005）**[14] 清楚地概述了這條流程與其中的取捨，很適合在讀第 3、5、6 章時放在手邊。

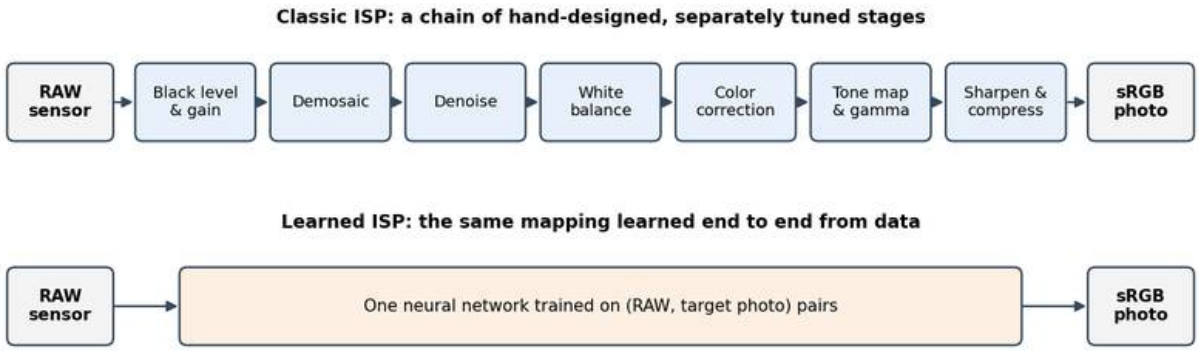


圖 *1.8* — 傳統相機 *ISP*（上）是一串人工設計的階段，每一個都會在本系列某處介紹。學習式 *ISP*（下）則用一個以原始資料與目標照片配對訓練的網路，取代整串或部分階段。

深度學習從幾個方向切入這條流程：

- **端到端學習困難的階段。** Chen 等人（2018）訓練一個全卷積網路，把在極暗環境下以短曝光拍攝的原始影像，直接映射成乾淨的照片，並以同一場景的長曝光照片作為目標 [15]。傳統流程在這種情況下會嚴重放大雜訊。
- **讓訓練資料更逼真。** Brooks 等人（2018）提出把一般照片「反處理」（unprocess）的方法：反轉相機流程的各個步驟來合成逼真的原始資料，讓以此訓練的去雜訊網路能用在真實的感測器

輸出上 [16]。這是傳統流程知識讓學習模型變得更好的好例子。

- **取代整個 ISP**。 Ignatov、Van Gool 與 Timofte（2020）提出 PyNET，一個金字塔形的卷積網路，把手機感測器的原始資料直接映射成高品質照片，在單一模型中學會去馬賽克、去雜訊、色彩與色調 [17]。他們以手機原始影像與高階單眼相機拍攝同一場景的照片配對來訓練。
- **整體綜述**。 da Silva 等人（2023）綜述了用於影像訊號處理的深度學習方法，比較只取代個別階段的做法與用單一網路取代整個 ISP 的做法 [18]。

有什麼**沒有**改變？第 2、4、5 章的成像數學，依然決定了任何方法——不論是不是學習而來——能還原多少資訊。學習模型的訓練與評估，用的仍是經典章節定義的品質指標、轉換與色彩空間。而且許多實際部署的系統是混合式的：在傳統流程中放進一個學習式去雜訊器，或是以物理為基礎的重建再交給網路精修 [12]。理解經典步驟，才能看出這類系統中哪一部分在做什麼，以及它失敗時為什麼失敗。

重點整理

- 數位影像是在 $M \times N$ 網格上取樣、並量化為 $L = 2^k$ 個等級的函數 $f(x, y)$；在程式中它就是一個陣列。
- 影像處理是一條連續光譜：低階（影像到影像）、中階（影像到屬性）與高階（屬性到意義）。電腦視覺位於高階那一端。
- 這個領域源自電報圖片傳輸（1920 年代）、太空探測器影像的電腦修正（1960 年代），以及 CT 這類由計算產生的醫學影像（1970 年代）。
- 影像來自電磁波譜的每個部分，從伽瑪射線到無線電波，也來自超音波、震波與電子束等非電磁來源。來源的物理特性決定了雜訊與解析度，也就決定了你需要哪些工具。
- 基本步驟分成輸出為影像的步驟（從擷取到壓縮）與輸出為屬性的步驟（從形態學到分類），全部由領域知識引導。第 2–13 章依循這份路線圖。
- 一套可運作的系統結合了感測器、專用硬體、電腦、軟體、儲存、顯示器、硬拷貝與網路。
- 深度學習已大致接管高階辨識，並正在重塑相機 ISP 這類低階步驟；但經典的成像模型依然是學習方法賴以建立的基礎。

練習

1. **儲存預算**。一台顯微鏡以縮時方式拍攝 2048 × 2048、16 位元的灰階影像，每 30 秒一張，連續 48 小時。整個實驗需要多少未壓縮的儲存空間？一顆 1 TB 的硬碟可以放幾張？

   提示

   每張影像是 2048 × 2048 × 16 位元 = 8,388,608 位元組，約 8.4 MB。共有 48 × 120 = 5760 張，總計約 48 GB。1 TB 硬碟大約可放 $10^{12}/8.39 \times 10^{6} \approx 119{,}000$ 張。

2. **放到光譜上**。判斷下列工作屬於低階、中階或高階，並說明理由：(a) 去除掃描文件上的椒鹽雜訊；(b) 在顯微影像中描出每個細胞的輪廓並回報每個細胞的面積；(c) 判斷一張胸部 X 光片是否正常；(d) 把一張太暗的照片調亮。

提示

看輸出是什麼。輸出影像就是低階（(a)、(d)）。輸出區域或量測值是中階（(b)）。輸出對場景的判斷是高階（(c)）。

3. **為什麼用 X 光而不是可見光？**用 $E = hc/\lambda$ 計算 0.05 nm X 光與 600 nm 橙光的光子能量。用兩句話解釋為什麼前者能用來拍攝骨骼，後者卻不行。

提示

約 24.8 keV 對約 2.07 eV，比值大約 12,000。可見光光子在組織中幾公釐內就被吸收或散射；高能的 X 光光子能穿過軟組織，而骨骼吸收的比例較高，因此產生對比。

4. **需要幾個灰階？**使用本章的 `quantize` 函式，畫出 `data.camera()` 與其量化版本之間的平均絕對誤差，$L = 2, 3, \ldots, 32$。接著實際看看這些影像：大約在哪個 L 以後，你不再明顯看到假輪廓？誤差曲線能不能準確預測你看到的結果？

提示

均勻量化的誤差大約以 $1/L$ 的速度下降。看得見的色帶主要取決於影像中平滑的區域（這裡是天空），而不是平均誤差——這也是第 2 章把感知品質與數值誤差分開討論的原因之一。

5. **設計一套系統。**一間工廠想在輸送帶上剔除有裂痕的陶瓷磁磚，速度是每秒兩片。列出你需要圖 1.7 中的哪些元件；並針對圖 1.6 的每個基本步驟，說明你是否會使用它以及原因。

提示

你需要一個搭配受控照明的感測器、每秒能處理兩張影像的運算能力，以及連到剔除機構的連線。可能用到的步驟：擷取、強化（對比）、或許形態學、分割出像裂痕的結構、特徵（裂痕長度），以及一條分類規則。色彩、小波與壓縮大概用不到，除非影像需要存檔備查。

6. **學習式還是傳統式？**從圖 1.8 的傳統 ISP 中挑一個階段。提出一個用學習模型取代它的理由，以及一個保留人工設計版本的理由。

提示

支持學習：網路能適應實際的感測器雜訊與場景統計特性。支持保留：人工設計的階段行為可預測、執行成本低、容易調校，而且不需要配對的訓練資料。現代觀點中的幾篇參考文獻正好討論了這個取捨。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 1. publisher page

2. K. Kobayashi, “Birth of a Digital Phototelegraph — the Bartlane System,” *Journal of the Institute of Image Electronics Engineers of Japan*, vol. 31, no. 2, pp. 244–249, 2002. J-STAGE
3. NASA, “First Image of the Moon Taken by a U.S. Spacecraft” (Ranger 7, PIA02975), NASA Photojournal. NASA
4. G. N. Hounsfield, “Computerized transverse axial scanning (tomography): Part 1. Description of system,” *British Journal of Radiology*, vol. 46, no. 552, pp. 1016–1022, 1973. doi:10.1259/0007-1285-46-552-1016
5. L. A. Shepp and B. F. Logan, “The Fourier reconstruction of a head section,” *IEEE Transactions on Nuclear Science*, vol. 21, no. 3, pp. 21–43, 1974. doi:10.1109/TNS.1974.6499235
6. P. C. Lauterbur, “Image formation by induced local interactions: examples employing nuclear magnetic resonance,” *Nature*, vol. 242, pp. 190–191, 1973. doi:10.1038/242190a0
7. R. Szeliski, *Computer Vision: Algorithms and Applications*, 2nd ed., Springer, 2022. book site
8. J. N. Mait, G. W. Euliss and R. A. Athale, “Computational imaging,” *Advances in Optics and Photonics*, vol. 10, no. 2, pp. 409–483, 2018. doi:10.1364/AOP.10.000409
9. The Event Horizon Telescope Collaboration, “First M87 Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole,” *The Astrophysical Journal Letters*, vol. 875, no. 1, 2019. doi:10.3847/2041-8213/ab0ec7
10. M. Delbracio, D. Kelly, M. S. Brown and P. Milanfar, “Mobile Computational Photography: A Tour,” arXiv:2102.09000, 2021. arXiv
11. G. Gallego et al., “Event-based Vision: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. arXiv
12. G. Wang, J. C. Ye and B. De Man, “Deep learning for tomographic image reconstruction,” *Nature Machine Intelligence*, vol. 2, no. 12, pp. 737–748, 2020. doi:10.1038/s42256-020-00273-z
13. Y. LeCun, Y. Bengio and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015. doi:10.1038/nature14539
14. R. Ramanath, W. E. Snyder, Y. Yoo and M. S. Drew, “Color image processing pipeline,” *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 34–43, 2005. doi:10.1109/MSP.2005.1407713
15. C. Chen, Q. Chen, J. Xu and V. Koltun, “Learning to See in the Dark,” *CVPR*, 2018. arXiv
16. T. Brooks, B. Mildenhall, T. Xue, J. Chen, D. Sharlet and J. T. Barron, “Unprocessing Images for Learned Raw Denoising,” arXiv:1811.11127, 2018. arXiv
17. A. Ignatov, L. Van Gool and R. Timofte, “Replacing Mobile Camera ISP with a Single Deep Learning Model,” arXiv:2002.05509, 2020. arXiv
18. M. H. M. da Silva et al., “ISP meets Deep Learning: A Survey on Deep Learning Methods for Image Signal Processing,” arXiv:2305.11994, 2023. arXiv

IP 02・數位影像基礎

眼睛與相機如何把光變成數字、取樣與量化如何決定一張數位影像，以及之後每一章都會用到的基本數學工具。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/

你將學到

- 人眼如何感知光，以及為什麼我們「看到的」不一定是「實際存在的」（亮度適應、Weber 比、馬赫帶、同時對比）
- 影像在物理上從哪裡來：電磁波譜、感測器，以及照度–反射率模型 $f = i \cdot r$
- 取樣與量化在做什麼、它們如何決定空間解析度與強度解析度，以及最近鄰、雙線性、雙三次內插如何縮放影像
- 像素之間關係的術語：鄰居、相鄰、連通、區域、邊界，以及三種距離量測
- 之後每一章都會用到的數學工具箱：逐元素運算與矩陣運算、線性、算術運算、集合與邏輯運算、幾何轉換、影像轉換，以及把直方圖當成機率

先看全貌

第 1 章說明了數位影像處理「是什麼」。這一章要回答「數位影像到底是什麼」：從離開場景的光開始，一路走到記憶體裡的一格格整數。途中我們會定義像素、鄰居、解析度、線性運算子等詞彙——之後整個系列都會直接使用它們，不再另外解釋。

> **用最簡單的話說**
>
> 想像你要用一盒蠟筆，把一幅畫臨摹到方格紙上。首先你得決定方格紙有幾格：格子越多，越能畫出細小的細節，這叫做**取樣**。接著你得決定可以用幾種顏色的蠟筆：蠟筆越多，深淺變化越平滑，這叫做**量化**。一張數位影像就是那張方格紙，每一格裡寫著一個數字，代表你選了哪一支蠟筆。影像處理的其他一切，都是在教你怎麼改變這些數字。

這一章不只是記帳。第一，大多數影像最後是給人看的，所以我們必須知道眼睛在哪裡容易被騙、在哪裡又特別敏銳。第二，拍攝影像時做的每個選擇——取樣多少點、量化成幾階、用哪一種內插——都會留下痕跡，後續的演算法只能與之共處。深度學習並沒有讓這些選擇消失；它多半只是把這些選擇搬進了訓練資料與損失函數裡，「現代觀點」一節會說明這點。

視覺感知的要素

白話說：你的眼睛是一台由膠狀物質和神經細胞組成的相機。它非常擅長看出「差異」與「邊緣」，卻不太會判斷絕對亮度。

眼睛的構造與成像

光線穿過角膜，通過瞳孔（大小由虹膜控制），再由水晶體聚焦到**視網膜**上——那是眼球後方一層薄薄的感光細胞 [1]。水晶體是有彈性的：睫狀肌改變它的曲率，也就改變了焦距，讓遠近物體都能對焦。視網膜上的像是倒立的實像，和相機感測器上的像一樣。

視網膜上有兩種感光細胞 [1]：

- **錐細胞（cones）**——每隻眼睛有數百萬個，最密集地分布在中央一個小凹處，稱為**中央窩（fovea）**。它們在明亮環境下運作（明視覺，photopic vision），分成三種光譜敏感度不同的類型，帶給我們色彩與細節。許多錐細胞擁有自己專屬的神經連結，所以中央窩的視覺特別清晰。
- **桿細胞（rods）**——數量多出許多倍，散布在整個視網膜上，但中央窩正中央沒有。它們在昏暗環境下運作（暗視覺，scotopic vision），無法分辨顏色，而且許多桿細胞共用一條神經末梢，用細節換取靈敏度。這就是為什麼月光下的物體看起來沒有顏色、輪廓也比較柔和。

視神經離開眼球的地方完全沒有感光細胞，稱為**盲點**。你平常察覺不到，是因為大腦會自動把空缺補上——這是一個早期的線索：知覺是一種主動的重建，而不是單純的記錄。

亮度適應與亮度辨別

視覺系統能應付極大範圍的光強度，從星光到陽光都行。它靠的是**亮度適應（brightness adaptation）**：在任一時刻，它只在以目前適應水準為中心的一個窄窗內運作，而這個窗會隨整體光線改變而滑動 [1]。在窗內，主觀亮度大致隨物理光強度的對數成長，而不是線性成長。

我們能察覺多小的變化？讓觀察者看一片強度為 I 的均勻背景，再閃一個強度為 $I + \Delta I$ 的小光點。有一半機率被察覺到的增量記為 ΔI_c。比值

$$W = \frac{\Delta I_c}{I}$$

稱為 **Weber 比（Weber ratio）**，其中 I 是背景強度，ΔI_c 是剛好能察覺的增量。W 越小代表辨別能力越好（很小的相對變化就看得出來）。在很寬的中間強度範圍內，W 大致是常數，這就是 Weber 定律：我們感受到的是相對變化，而不是絕對變化。在低強度、由桿細胞主導時，W 很大，辨別能力很差 [1]。

這有直接的工程意義。因為我們感受的是相對變化，物理上等間隔的灰階在知覺上並不等間隔。這也是影像通常以非線性（gamma）編碼儲存的原因之一，以及 8 位元通常就足以顯示的原因之一。

馬赫帶、同時對比與錯覺

感受到的亮度並不是強度的簡單函數。兩個經典示範讓這件事一目了然 [1]：

- **馬赫帶（Mach bands）**。給人看一組階梯狀、各自均勻的灰色條紋。每一條在物理上都是均勻的，但觀察者會說在每個邊緣的暗側看到一條更暗的細帶、亮側看到一條更亮的細帶。視覺系

統會誇大邊緣，這與相鄰感受野之間的側向抑制（lateral inhibition）相符——本質上就是一個內建的銳化濾波器。

- **同時對比（simultaneous contrast）**。同一塊中灰色方塊，放在暗背景上看起來較亮，放在亮背景上看起來較暗，儘管兩塊方塊完全相同。感受到的亮度取決於周圍環境。

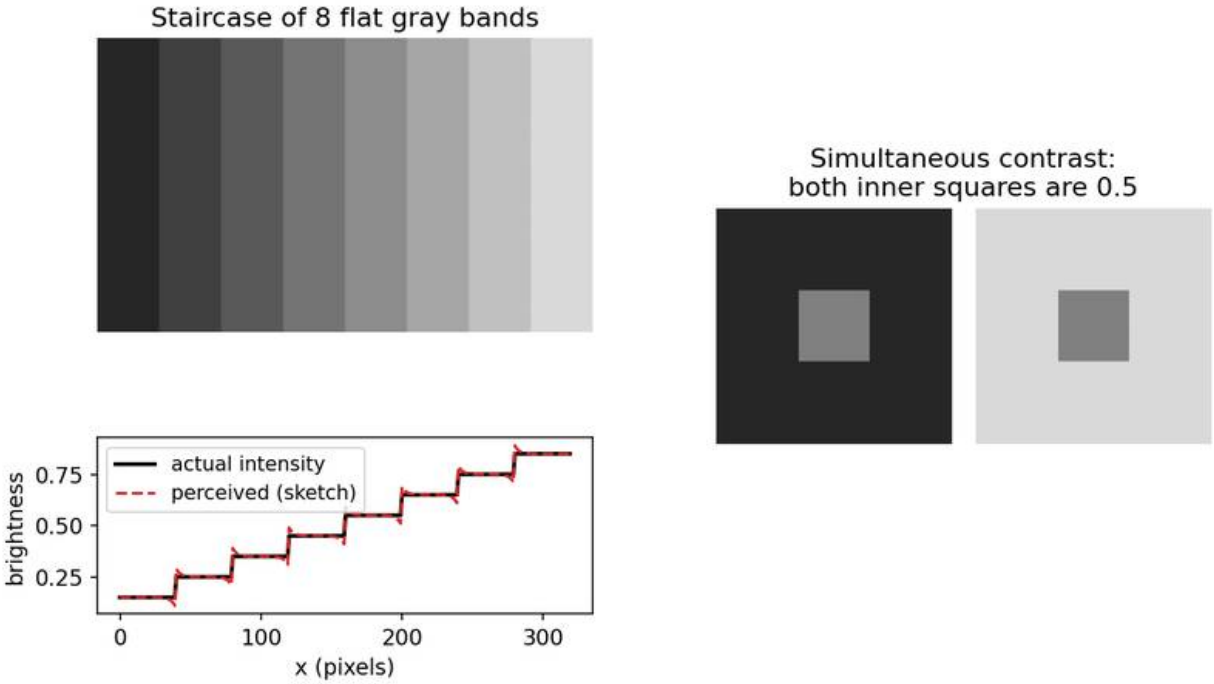


圖 *2.1* — 左：一組均勻灰帶；虛線示意觀察者在每個邊緣感受到的過衝與下衝（馬赫帶）。右：同時對比——兩塊內部方塊的強度都是 *0.5*。

一般的視錯覺——看見沒畫出來的輪廓、等長線段看起來不等長——都顯示眼睛會補足並重新詮釋。這對影像處理有兩層意義：一個把平方誤差降到最低的演算法，產生的影像仍可能看起來不對；而數值上很小的瑕疵，若剛好落在邊緣上，可能非常顯眼。這正是感知品質指標的出發點，會在「現代觀點」中討論。

光與電磁波譜

白話說：光是一種能量波。可見光只是寬廣波譜中薄薄的一片，這個波譜還包括無線電波、微波、紅外線、紫外線、X 射線與伽瑪射線——而這些都可以拿來成像。

波長、頻率與能量

電磁輻射可以描述成正弦波，也可以描述成一串無質量的粒子，稱為光子 [1]。兩種描述的關係是

$$\lambda = \frac{c}{\nu}, \qquad E = h\nu,$$

其中 λ 是波長（公尺），ν 是頻率（赫茲），$c \approx 3 \times 10^8$ m/s 是光速，E 是單一光子的能量（焦耳），h 是普朗克常數。波長越短，頻率越高，每個光子的能量也越高。所以伽瑪射線和 X 射線能穿透組織，而無線電波需要很大的天線才能偵測。

可見光大約從 400 nm（紫）到 700 nm（紅）。物體看起來是什麼顏色，取決於它反射哪些波長：綠葉主要反射波段中段的光，其餘被吸收。沒有顏色、只有強度的光稱為**無色光（achromatic）**或**單色光（monochromatic）**，它的強度稱為**灰階（gray level）**。

量測光的三個詞

「光有多少」可以指三件不同的事。**輻射度（radiance）**是光源發出的物理能量，**輝度（luminance）**是其中眼睛感受得到的部分，**亮度（brightness）**則是主觀的感覺，會受適應狀態與周圍環境影響。要記住的一點：感測器和眼睛一樣，只看得到它敏感波段裡的能量。

僅線上版：輻射度、輝度與亮度 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/notes/輻射度輝度與亮度/*

影像感測與擷取

白話說：感測器就是一個把光轉成電壓的小小測光表。把很多個排成一列或一個網格，就成了掃描器或相機。

單一、線型與陣列感測器

每一種成像裝置都結合了**照明光源**、**場景**，以及由對目標能量有反應的材料做成的**感測器** [1]。感測器輸出的電壓經數位化後成為一個數字。感測器有三種排列方式：

- **單一感測器**。一個偵測器（例如光二極體）加上兩個方向的機械移動，一次一個取樣點地掃出整張影像。速度慢，但便宜又非常精確，一些高精度掃描器採用這種方式。
- **線型（條狀）感測器**。一排偵測器一次擷取一條線，再靠垂直於感測條的移動提供第二個維度。平台式掃描器和航空推掃式（push-broom）成像儀就是這樣運作。把感測器排成環狀、圍繞一個旋轉的光源，就是電腦斷層掃描（CT）的幾何架構；CT 的影像是重建出來的，而不是直接讀出。
- **陣列感測器**。一個 2-D 的偵測器網格，例如 CCD 或 CMOS 晶片，一次擷取整張影像，這就是數位相機。鏡頭把場景聚焦到陣列上，每個偵測器在曝光時間內累積光量，讀出電路為每個偵測器產生一個值。

彩色相機通常在陣列上覆蓋一層紅、綠、藍濾光片組成的馬賽克，所以每個偵測器只量到一種顏色；每個像素缺少的另外兩個值，會在相機的影像處理管線中以內插補齊（去馬賽克，demosaicking）[8]。

簡單的成像模型

我們把一張單色影像寫成 2-D 函數 $f(x,y)$，其值正比於抵達 (x,y) 的能量。因為它來自物理能量，所以 $0 \le f(x,y) < \infty$。對反射光而言，把 f 拆成兩個因子很有幫助 [1]：

$$f(x,y) = i(x,y)\, r(x,y), \qquad 0 < i(x,y) < \infty, \qquad 0 < r(x,y) < 1,$$

其中 $i(x,y)$ 是**照度（illumination）**（照到該場景點的光有多少），$r(x,y)$ 是**反射率（reflectance）**（表面反射了其中多少比例，從接近 0 的完全吸收到接近 1 的完全反射）。照度取決於光源；反射率則是物體本身的性質。在 X 射線之類的穿透式成像中，以**穿透率（transmissivity）**取代反射率。

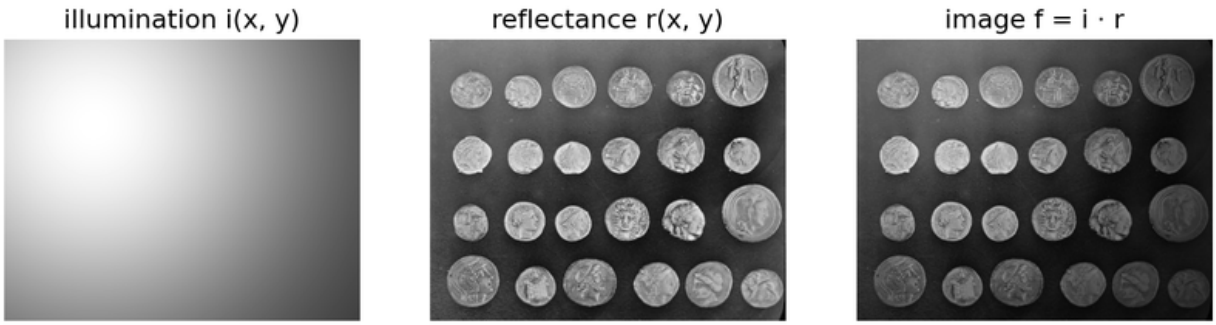


圖 2.2 — 照度–反射率模型。平滑的照度場（左）乘上反射率（中），得到記錄下來的影像（右）。

```
import numpy as np
from skimage import data, img_as_float

r = img_as_float(data.coins())          # reflectance in [0, 1]
h, w = r.shape
yy, xx = np.mgrid[0:h, 0:w]
i = 0.2 + 0.8 * xx / (w - 1)            # illumination ramp: dim left, bright right
f = i * r                               # what the sensor records
```

這個模型很簡單，卻很實用。照度通常在場景中緩慢變化，反射率則在物體邊緣處劇烈改變。後面的章節會利用這一點：同態濾波（homomorphic filtering，第 4 章）取 $\log f = \log i + \log r$ 把兩者分開；陰影校正（本章稍後）則把已知的 i 除掉。

影像取樣與量化

白話說：相機只在有限個位置量測連續的場景（取樣），再把每個量測值四捨五入到有限個階層之一（量化）。

基本概念

感測器的輸出是在連續平面上的連續電壓波形。要變成數位的，座標和振幅都必須離散化 [1]：

- **取樣（sampling）**把座標數位化：我們只保留網格位置上的值。
- **量化（quantization）**把振幅數位化：每個保留的值都對應到有限個階層中最接近的那一個。

對陣列感測器而言，偵測器的數量與間距決定了取樣；類比數位轉換器決定了量化。需要取樣多密，取決於影像內容：根據取樣定理，一個頻寬有限的訊號，只要取樣頻率大於其最高頻率的兩倍，就能完美重建。第 4 章會正式推導；Unser 的回顧論文則提供了以樣條（spline）為基礎、較現代的取樣與重建觀點 [2]。

數位影像的表示法

經過取樣與量化後，影像是一個 M 列、N 行的矩陣：

$$f = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

每個元素是一個**像素（pixel，picture element）**。依照教科書的慣例 [1]，x 是列索引、y 是行索引，原點在左上角。（許多函式庫，包括 OpenCV 的點座標 API，用的是 $(x, y) =$（行, 列）。務必確認。）

強度階數通常是 2 的冪次 $L = 2^k$，所以值落在 $[0, L-1]$。儲存一張影像需要

$$b = M \times N \times k \text{ 位元},$$

其中 k 是每像素的位元數。一張 512×512、$k = 8$ 的影像需要 262,144 位元組。成像系統的**動態範圍（dynamic range）**是可量測的最大強度（受飽和限制）與最小強度（受雜訊限制）之比；某一張影像的**對比（contrast）**則是其最高與最低強度之差。

空間解析度與強度解析度

空間解析度（spatial resolution）是能辨識的最小細節尺寸，必須帶單位才有意義：每毫米線對數，或印刷時的每英寸點數（dpi）。只說影像是「1024 × 1024」，在不知道它涵蓋多大實際面積前，並沒有說出任何解析度資訊。**強度解析度（intensity resolution）**是能辨識的最小階層變化，通常以位元數 k 表示。

```
import numpy as np
from skimage import data

img = data.camera()                       # 512 x 512, uint8 (k = 8)
small = img[::4, ::4]                     # keep every 4th sample -> 128 x 128

def quantize(x, bits):
    step = 2 ** (8 - bits)
    return (x // step) * step + step // 2 # map to the centre of each bin

q3 = quantize(img, 3)                     # 8 gray levels
print(small.shape, np.unique(q3).size)    # (128, 128) 8
```

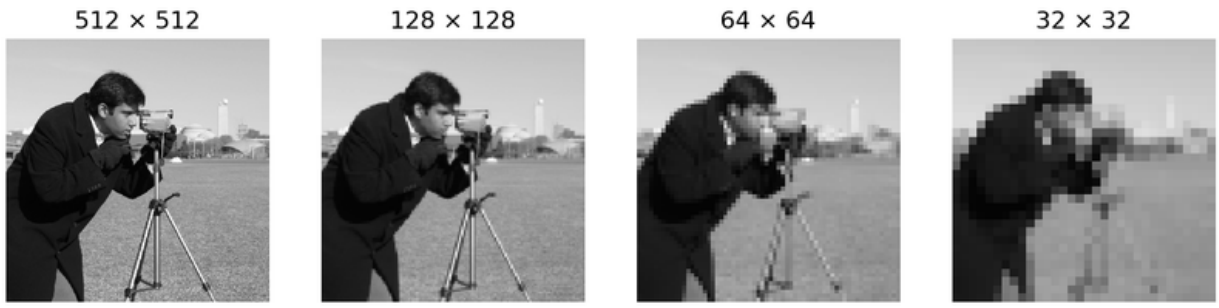


圖 2.3 — 降低空間解析度。每個版本都以相同大小顯示，因此取樣點越少，像素看起來就越大、越像方塊。



圖 2.4 — 降低強度解析度。大約在 8 階時，天空這類平滑區域開始出現假輪廓；只剩 2 階時，只看得到剪影。

由此得到兩個教訓。減少取樣點時，首先被破壞的是細緻紋理（草、頭髮），接著才是邊緣。減少階層時，首先受損的是平滑區域：圖 2.4 的天空會碎成一階一階，稱為**假輪廓（false contouring）**——恰好出現在 Weber 比說眼睛最敏感的地方。細節多的影像比平滑的影像更能容忍較少的灰階，因為繁雜的紋理會把階梯掩蓋掉 [1]。

> **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

影像內插

內插（interpolation）是在沒有取樣值的位置估計數值。每當影像被縮放、旋轉、變形或對位時，都會用到它 [1]、[3]、[4]。

最近鄰（nearest neighbor）直接複製最近取樣點的值。它快速、保留原始數值，但會讓邊緣呈鋸齒方塊；縮小影像時，甚至可能讓細線整條消失。

雙線性（bilinear）內插使用最近的四個取樣點。對於距左上取樣點 f_{00} 的小數偏移量為 $a, b \in [0, 1)$ 的位置，

$$v = (1-a)(1-b)\,f_{00} + a(1-b)\,f_{01} + (1-a)b\,f_{10} + ab\,f_{11},$$

其中 f_{01}、f_{10}、f_{11} 分別是右、下、右下的鄰居。等價地，$v(x,y) = c_1 x + c_2 y + c_3 xy + c_4$，四個係數由四個取樣點決定。結果是連續的，但略顯模糊。

```
import numpy as np

def bilinear(img, x, y):
    """Sample img (H, W) at real-valued column x, row y."""
    if img.ndim != 2 or not img.size:
        raise ValueError("img must be a nonempty 2-D array")
    if not (0 <= x <= img.shape[1] - 1 and 0 <= y <= img.shape[0] - 1):
        raise ValueError("sample coordinates must lie inside the image")
    x0, y0 = int(np.floor(x)), int(np.floor(y))
    x1, y1 = min(x0 + 1, img.shape[1] - 1), min(y0 + 1, img.shape[0] - 1)
    a, b = x - x0, y - y0
    top = (1 - a) * img[y0, x0] + a * img[y0, x1]
    bot = (1 - a) * img[y1, x0] + a * img[y1, x1]
    return (1 - b) * top + b * bot

print(bilinear(np.array([[0., 10.], [20., 30.]]), 0.5, 0.5))   # 15.0
```

這個函式只接受影像內的座標，包含最後一列與最後一行。它會拒絕越界座標，避免負的 NumPy 索引意外讀取另一側像素。幾何變形若需要影像外的數值，必須另選常數補值或鏡射等邊界政策。

雙三次（bicubic）內插使用最近的 16 個取樣點。它用一個平滑的分段三次核來加權這些取樣點——最常用的是 Keys 的三次卷積核 [3]——先沿列、再沿行套用。這個核帶有小小的負瓣，所以雙三次比雙線性更能保持邊緣清晰，但可能在強邊緣旁產生輕微的過衝（振鈴，ringing）。

僅線上版：三次卷積核 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/notes/三次卷積核/*

```
import cv2
from skimage import data

crop = data.camera()[150:214, 230:294]    # 64 x 64
up = {name: cv2.resize(crop, None, fx=4, fy=4, interpolation=flag)
      for name, flag in [("nearest", cv2.INTER_NEAREST),
                         ("bilinear", cv2.INTER_LINEAR),
                         ("bicubic", cv2.INTER_CUBIC)]}
```



圖 2.5 — 同一裁切區放大四倍。最近鄰呈方塊狀；雙線性平滑但偏軟；雙三次讓邊緣稍微清晰一些。

拖動滑桿，比較把 96 × 96 裁切區放大成 384 × 384 時，最近鄰與雙三次的差別。注意相機的邊緣與臉部輪廓。



最近鄰



雙三次

攝影師裁切區以最近鄰與雙三次內插放大四倍的比較

沒有任何內插方法能創造出取樣時已丟掉的細節。Thévenaz、Blu 與 Unser 的回顧 [4] 把這點說得很精確：內插器的品質取決於它的逼近階數，以及它的核有多接近理想（sinc）重建器；在相近的計算成本下，較高階的 B-spline 方法優於傳統的三次卷積。「現代觀點」中討論的學習式超解析度（super-resolution）更進一步，會根據訓練資料腦補出看似合理的細節。

像素之間的基本關係

白話說：在談論影像裡的物體之前，我們需要規定哪些像素彼此「相碰」，以及兩個像素相距多遠。

像素的鄰居

位於 (x, y) 的像素 p 有四個水平與垂直鄰居，

$$N_4(p) = \{(x+1, y),\ (x-1, y),\ (x, y+1),\ (x, y-1)\},$$

四個對角鄰居，

$$N_D(p) = \{(x+1, y+1),\ (x+1, y-1),\ (x-1, y+1),\ (x-1, y-1)\},$$

以及八個鄰居 $N_8(p) = N_4(p) \cup N_D(p)$ [1]。在影像邊界上，有些鄰居會落在影像外；要怎麼處理（忽略、補值、鏡射）是一個選擇，在濾波時會再遇到。

相鄰、連通、區域與邊界

令 V 為被視為「相似」的強度值集合——二值影像中 $V = \{1\}$；灰階影像中可能是 100 到 120 之間的所有值。兩個值都在 V 中的像素 p 與 q [1]：

- 若 $q \in N_4(p)$，則為 **4-相鄰（4-adjacent）**；
- 若 $q \in N_8(p)$，則為 **8-相鄰（8-adjacent）**；
- 若 $q \in N_4(p)$，或者 $q \in N_D(p)$ 且集合 $N_4(p) \cap N_4(q)$ 中沒有任何值屬於 V 的像素，則為 **m-相鄰（m-adjacent，混合相鄰）**。

為什麼需要 m-相鄰？在 8-相鄰下，一步對角線和兩步「繞過轉角」的路線可能同時連接同一對像素，使路徑產生歧義。m-相鄰只在不存在 4-連通路線時才保留對角連結，消除了多餘的迴圈（圖 2.6 右）。



圖 2.6 — 像素 p 的鄰域，以及二值區塊上的 m-相鄰：虛線對角不構成 m-相鄰，因為這兩個像素已經共享一個屬於 V 的 4-鄰居。

從 p 到 q 的**數位路徑（digital path）**是一串互不相同的像素 $(x_0, y_0), \ldots, (x_n, y_n)$，相鄰兩個像素彼此相鄰；$n$ 是路徑長度，若第一個與最後一個像素相同，則稱為**封閉路徑**。依所用的相鄰定義，我們稱之為 4-、8- 或 m-路徑。對於像素子集 S，若 p 與 q 之間存在一條完全位於 S 中的路徑，就說它們**在 S 中連通**。在 S 中與 p 連通的所有像素構成一個**連通分量（connected component）**；若 S 只有一個連通分量，則 S 是**連通集合**。

連通集合稱為**區域（region）**。若兩個區域的聯集是連通的，就說它們相鄰。區域 R 的**邊界（boundary，或 border）**是 R 中至少有一個鄰居位於 R 外的像素集合。注意邊界與**邊緣（edge）**的差別：邊界是區域整體、封閉的性質；邊緣則是局部的強度不連續，未必會閉合（第 10 章）。

相鄰的選擇會改變「有幾個物體？」這類簡單問題的答案：

```
import numpy as np
from skimage.measure import label

b = np.array([[1, 0, 0],
              [0, 1, 0],
```

```
              [0, 0, 1]])
print(label(b, connectivity=1).max(),     # 4-connectivity: 3 objects
      label(b, connectivity=2).max())     # 8-connectivity: 1 object
```

一個經典結論：如果前景用 8-連通，背景就應該用 4-連通（反之亦然），否則一條封閉的數位曲線可能無法把內外分開。這是數位拓樸（digital topology）的基本結果 [1]。

距離量測

對像素 $p = (x, y)$、$q = (u, v)$ 與 z，若函數 D 滿足 $D(p,q) \geq 0$ 且僅在 $p = q$ 時等號成立、$D(p,q) = D(q,p)$，以及 $D(p,z) \leq D(p,q) + D(q,z)$，則 D 是一個**距離度量（distance metric）**。三種標準度量是 [1]、[5]：

$$
\begin{aligned}
D_E(p,q) &= \sqrt{(x-u)^2 + (y-v)^2} && \text{(Euclidean)} \\
D_4(p,q) &= |x-u| + |y-v| && \text{(city-block)} \\
D_8(p,q) &= \max(|x-u|,\, |y-v|) && \text{(chessboard)}
\end{aligned}
$$

即歐氏距離、城市街區距離與棋盤距離。與 p 的 D_E 不超過某定值的像素構成圓盤；D_4 構成菱形；D_8 構成正方形。$D_4 = 1$ 的像素恰好是 $N_4(p)$，$D_8 = 1$ 的像素恰好是 $N_8(p)$。D_4 與 D_8 只取決於座標，與像素值無關；反之，沿著 m-路徑量的距離會受像素值影響，因為路徑本身受像素值影響。

```
import numpy as np
p, q = np.array([2, 3]), np.array([5, 7])
d = np.abs(p - q)
print("D_E =", np.hypot(*d), " D_4 =", d.sum(), " D_8 =", d.max())   # 5.0, 7, 4
```

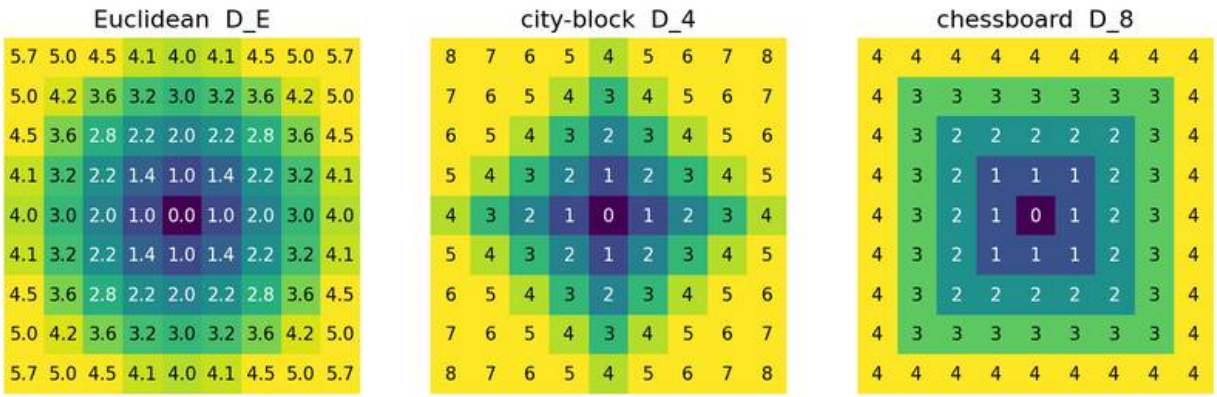


圖 2.7 — 三種度量下與中心像素的距離。等距輪廓分別是圓形、菱形與正方形。

Rosenfeld 與 Pfaltz [5] 提出了高效率的兩趟（two-pass）演算法，一次算出每個像素到最近物體像素的距離——也就是**距離轉換（distance transform）**，至今仍用於骨架化、形態學與形狀比對（第 9 章）。

數位影像處理的數學工具簡介

白話說：影像就是矩陣，所以大部分影像處理都是對矩陣做運算。本節列出我們會一再用到的運算，以及決定哪些工具適用的關鍵性質——線性。

逐元素運算與矩陣運算

逐元素（elementwise，又稱陣列）運算逐一像素作用。兩張 2×2 影像的逐元素乘積為

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix},$$

而矩陣乘積則遵守列乘行的規則。除非特別說明，影像運算——包括「兩張影像相乘」——都是逐元素的 [1]。在 NumPy 中，`A * B` 是逐元素乘積，`A @ B` 是矩陣乘積。

線性與非線性運算

考慮一個把輸入影像 f 對應到輸出 $g = \mathcal{H}[f]$ 的運算子 $\mathcal{H}$。若對所有影像 f_1, f_2 與所有常數 a, b 都有

$$\mathcal{H}[a f_1 + b f_2] = a\,\mathcal{H}[f_1] + b\,\mathcal{H}[f_2]$$

則它是**線性**的。這個性質的兩部分是**可加性**（和的輸出等於輸出的和）與**齊次性**（輸入縮放，輸出也同比例縮放）。把所有像素加總是線性的；取最大值則不是：

```
import numpy as np
rng = np.random.default_rng(0)
f1, f2 = rng.random((4, 4)), rng.random((4, 4))
a, b = 2.0, -3.0
print(np.isclose(np.sum(a*f1 + b*f2), a*np.sum(f1) + b*np.sum(f2)))   # True
print(np.isclose(np.max(a*f1 + b*f2), a*np.max(f1) + b*np.max(f2)))   # False
```

線性之所以重要，是因為線性運算子完全由它對簡單輸入的反應決定，而且有大量理論（卷積、傅立葉轉換，第 3–5 章）可以套用。中值濾波器或形態學等非線性運算子能做到線性運算子做不到的事，但較難分析。

算術運算

兩張同尺寸影像的和、差、積、商都是逐元素運算 [1]。其中三種用途特別重要。

平均多張含雜訊的影像。假設我們對一個靜止場景拍了 K 張影像，$g_i(x,y) = f(x,y) + \eta_i(x,y)$，其中 f 是乾淨影像，η_i 是各張之間互不相關、平均為零、變異數為 σ_η^2 的雜訊。平均值 $\bar{g} = \frac{1}{K}\sum_{i=1}^{K} g_i$ 滿足

$$E\{\bar{g}(x,y)\} = f(x,y), \qquad \sigma_{\bar{g}}^2(x,y) = \frac{1}{K}\,\sigma_\eta^2(x,y),$$

所以雜訊標準差以 $1/\sqrt{K}$ 的速度下降。天文學家疊加多次曝光就是這個道理；手機的「夜間模式」在對齊一連串連拍畫面後，做的也是同一件事 [7]。

```
import numpy as np
from skimage import data, img_as_float

g = img_as_float(data.camera())
rng = np.random.default_rng(1)
for K in (1, 4, 16, 64):
    avg = np.mean([g + rng.normal(0, 0.2, g.shape) for _ in range(K)], axis=0)
    print(K, round(float(np.std(avg - g)), 3))    # 0.2, 0.1, 0.05, 0.025
```

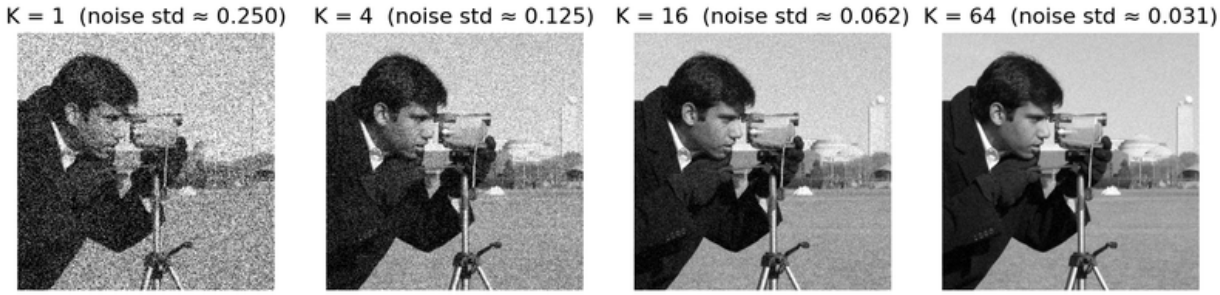


圖 *2.8* — 平均 K 張彼此獨立的含雜訊畫面。K 每乘以 *4*，雜訊標準差大約減半。

相減能凸顯差異。把視訊的兩張畫面相減，會凸顯移動；在醫學影像中，把顯影前的「遮罩」影像從顯影後的影像中減去，主要只會留下吸收了顯影劑的血管。

相乘與相除處理陰影與遮罩。若已知感測器的陰影型態 $h(x,y)$（例如拍攝均勻目標取得），觀察到的影像 $g = f \cdot h$ 可以用 $f = g/h$ 校正。乘上二值遮罩則可保留感興趣區域、把其餘歸零。

一條實用規則：算術結果常常超出有效範圍（相減可能為負，相加可能超過 255）。請用浮點數運算，再重新縮放。縮放到 $[0, K]$ 的簡單方法是先算 $f_m = f - \min f$，再算 $f_s = K f_m / \max f_m$。

集合與邏輯運算

把二值影像看成像素座標的**集合** A（前景），一般的集合運算就都適用 [1]：聯集 $A \cup B$、交集 $A \cap B$、補集 $A^c = \{w \mid w \notin A\}$，以及差集 $A - B = A \cap B^c$。在二值陣列上，它們分別對應邏輯運算 OR、AND、NOT 與 AND-NOT。對灰階影像，聯集與交集通常定義為逐元素的最大值與最小值。模糊集合（fuzzy sets）更進一步，讓隸屬度可以介於 0 與 1 之間。

```
import numpy as np
from skimage import data

coins = data.coins()
A = coins > 100                                      # set A: bright pixels
B = np.zeros_like(A); B[:, : coins.shape[1] // 2] = True   # set B: left half
union, inter, diff = A | B, A & B, A & ~B
```

集合運算是數學形態學（第 9 章）以及合併分割遮罩時的共通語言。

空間運算

空間運算直接作用在像素值與位置上，分為三類 [1]。

單像素運算各自獨立地改變每個值：$s = T(z)$，其中 z 是輸入強度、s 是輸出。影像負片 $s = (L-1) - z$ 就是一例。這些就是第 3 章的強度轉換。

鄰域運算由 (x, y) 周圍的鄰域 S_{xy} 計算每個輸出。局部平均

$$g(x,y) = \frac{1}{mn} \sum_{(r,c) \in S_{xy}} f(r,c),$$

（其中 S_{xy} 是以 (x, y) 為中心的 $m \times n$ 視窗）是最簡單的例子；第 3 章會把它推廣成空間濾波。

幾何轉換移動像素。它分兩步：座標的空間轉換，以及用強度內插填滿新網格。最常見的一族是**仿射（affine）**轉換。以齊次座標表示，

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

其中 (x, y) 是輸入位置、(x', y') 是輸出位置，a_{ij} 編碼旋轉、縮放與剪切，(t_x, t_y) 是平移。仿射轉換保持直線與平行性，串接多個轉換就等於把矩陣相乘。

實作上幾乎都用**反向映射（inverse mapping）**：對每個輸出像素，套用反轉換找出它在輸入中的來源位置，再在該處內插。正向映射（把每個輸入像素推到輸出）會留下空洞與碰撞。

```
import cv2
from skimage import data

img = data.camera()
h, w = img.shape
M = cv2.getRotationMatrix2D(center=(w / 2, h / 2), angle=30, scale=0.8)  # 2x3 affine
rot = cv2.warpAffine(img, M, (w, h), flags=cv2.INTER_LINEAR)  # inverse mapping inside
```

有兩個相關的任務建立在這些轉換之上。**影像對位（registration）**在轉換未知時，利用兩張影像中都找得到的點把它估計出來，讓同一場景的兩張影像對齊。**單應性（homography）**是在仿射轉換之外再加上透視效果的 3×3 轉換；從不同視角拍攝同一個平面的兩張照片，就是以它相互對應。

僅線上版：影像對位 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/notes/影像對位/*

僅線上版：單應性：投影轉換 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/notes/單應性投影轉換/*

向量與矩陣運算

彩色像素是一個向量 $\mathbf{z} = [z_1, z_2, z_3]^T$，分量是紅、綠、藍值。向量工具因此可直接套用；例如 $\mathbf{z}$ 到參考色 $\mathbf{a}$ 的歐氏距離

$$D(\mathbf{z}, \mathbf{a}) = \left[(\mathbf{z} - \mathbf{a})^T (\mathbf{z} - \mathbf{a}) \right]^{1/2},$$

是簡單彩色分割（第 6 章）的基礎。把 $M \times N$ 影像堆疊成 $MN \times 1$ 的向量，也能把任何線性運算寫成 $\mathbf{g} = \mathbf{H}\mathbf{f}$，這個形式會在影像復原（第 5 章）中用到。

影像轉換

有些問題換到另一個域會比較好解。2-D 線性轉換的一般形式為 [1]

$$T(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, r(x, y, u, v),$$

其中 f 是輸入影像，r 是**正向轉換核**，(u, v) 是轉換域的變數。反轉換使用反向核 $s(x, y, u, v)$。當 $r(x, y, u, v) = r_1(x, u)\, r_2(y, v)$ 時，核是**可分離的**，2-D 轉換可以先沿列、再沿行做 1-D 轉換，或寫成矩陣形式 $\mathbf{T} = \mathbf{A}\mathbf{F}\mathbf{B}$。傅立葉轉換（第 4 章）與小波轉換（第 7 章）是你最常遇到的兩種。

機率方法

強度可以當成隨機量來處理。若 $M \times N$ 影像中有 n_k 個像素的強度為 z_k，則該強度的機率估計為

$$p(z_k) = \frac{n_k}{MN}, \qquad \sum_{k=0}^{L-1} p(z_k) = 1,$$

這就是**正規化直方圖（normalized histogram）**。強度的平均值與變異數為

$$m = \sum_{k=0}^{L-1} z_k\, p(z_k), \qquad \sigma^2 = \sum_{k=0}^{L-1} (z_k - m)^2\, p(z_k),$$

其中 m 衡量平均亮度，σ（標準差）衡量對比。更高階的動差描述偏斜與尖峰程度。

```
import numpy as np
from skimage import data

img = data.camera()
p = np.bincount(img.ravel(), minlength=256) / img.size   # p(z_k)
z = np.arange(256)
m = np.sum(z * p)
sigma = np.sqrt(np.sum((z - m) ** 2 * p))
print(round(m, 2), round(sigma, 2))                      # 129.06 73.64
```

直方圖處理（第 3 章）與雜訊建模（第 5 章）都從這裡出發。

現代觀點

本章的基礎概念並沒有被深度學習取代，反而成了深度學習的介面。網路吃的是經過取樣與量化的陣列，用編碼了某種知覺模型的損失函數訓練，並在相機管線產生的資料上評估。以下幾個方向的回顧文獻勾勒出這片領域。

取樣與內插理論。Unser 的回顧 [2] 把取樣看成以實用的基底（例如 B-spline）來逼近，而不再要求理想的頻寬有限訊號與 sinc 重建器。重點是一起分析取樣基底、前置濾波與重建核；這些選擇都不能消除取樣率造成的資訊限制。Thévenaz、Blu 與 Unser [4] 在相同條件下比較各種內插核，發現以 B-spline 為基礎的內插，比 Keys 的三次卷積 [3] 有更好的品質與成本取捨。

學習式超解析度。Wang、Chen 與 Hoi [9] 以及 Moser 等人 [10] 綜述了深度超解析度，從 CNN 設計一路談到 transformer 與擴散模型。有兩個方法顯示古典概念如何在網路中延續：LIIF [11] 是可在任意座標查詢的學習式內插器；Real-ESRGAN [12] 則把模糊、雜訊、縮放與壓縮重複施加多次來合成訓練資料，因為用雙三次縮小產生的訓練資料對與真實退化並不相符。教訓是：雙三次內插既是比較基準，也是藏在訓練資料裡的假設。

知覺與影像品質。SSIM [13] 把感知一節的觀察轉成一個比較局部亮度、對比與結構的品質指標；Zhai 與 Min [14] 綜述了整個領域。深度網路特徵之間的距離（LPIPS）在該研究的人類判斷測試中，較所比較的傳統指標更吻合 [15]；Ding 等人 [16] 則發現，一個指標在品質資料集上的排名，並不能可靠預測它當作訓練損失時是否好用。簡單說，馬赫帶與 Weber 比如今活在損失函數裡。

僅線上版：*MSE、PSNR 與資料範圍* — *https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/notes/msepsnr-與資料範圍/*

相機感測器管線。Ramanath 等人 [8] 描述了傳統的機內管線（去馬賽克、白平衡、色彩轉換、gamma、壓縮）；Delbracio 等人 [7] 說明手機如何以連拍、對齊與合併重塑這條管線——也就是大規模的「平均含雜訊影像」。如今已有學習式模型取代其中一部分甚至全部 [17]、[18]。

僅線上版：各篇回顧的詳細介紹 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/fundamentals/papers/各篇回顧的詳細介紹/*

什麼沒有改變？像素網格、位元深度、相鄰與連通（至今仍是每個分割後處理步驟中連通分量標記的基礎）、仿射變形（如今成了網路中可微分的層），以及直方圖。正因為懂這些，你才能察覺模型的輸入是否用錯了縮放核、量化得太粗，或是用了一個與人類判斷不符的指標來比較。

重點整理

- 人類視覺感受的是相對變化（Weber 定律），並會誇大邊緣（馬赫帶）；感受到的亮度取決於周圍。好的影像處理會尊重這些事實。
- 影像可以由電磁波譜的任何部分形成；對反射光而言，$f = i \cdot r$ 把照度與反射率分開。
- 取樣決定空間解析度、量化決定強度解析度，儲存量是 MNk 位元。階數太少會在平滑區域造成假輪廓。
- 內插——最近鄰、雙線性、雙三次——估計取樣點之間的值，在速度、平滑度與銳利度之間取捨，但無法找回已遺失的細節。
- 4-、8- 與 m-相鄰定義了路徑、連通分量、區域與邊界；D_4 與 D_8 是距離度量，其單位球分別是菱形與正方形。

- 除非特別說明，影像運算都是逐元素的；線性決定了卷積與傅立葉工具是否適用。
- 平均 K 張獨立的含雜訊畫面，可讓雜訊標準差降為 $1/\sqrt{K}$；幾何轉換使用反向映射加上內插。
- 深度學習建立在這些基礎上：學習式內插器、感知損失與學習式相機管線，都承襲了本章的概念。

練習

1. 一台灰階掃描器產生 2400 × 3300、每像素 12 位元的影像。一張未壓縮的掃描需要多少 MB？若降為 8 位元呢？

   提示

   用 $b = MNk$，再除以 8×10^6（若要 MiB 則除以 8×2^{20}）。12 位元：$2400 \cdot 3300 \cdot 12/8 = 11.88$ MB。8 位元：7.92 MB。

2. 把 `skimage.data.moon()` 量化成 4、5、6 位元。對你而言，在哪個位元數時假輪廓消失？接著在以 4 位元量化之前，先加入少量均勻隨機雜訊（振幅為半個量化間距）。有什麼改變？為什麼？

   提示

   這種雜訊（稱為抖動，dither）會打散大片平坦的階梯，眼睛會把細小的雜訊平均掉，而不是看到輪廓。平均誤差差不多，但可見的結構改變了——再次提醒我們，平方誤差與感受到的品質是兩回事。

3. 在下面的二值影像中，分別以 4-、8- 與 m-連通計算 1 像素的連通分量數。再分別以 4- 與 8-連通計算 0 像素的連通分量數。哪一種搭配能給出一致的「內／外」答案？

```
0 0 0 0 0
0 1 1 1 0
0 1 0 1 0
0 1 1 0 0
0 0 0 0 0
```

   提示

   在三種相鄰定義下，1 都只構成一個連通分量；但因為右下角有個對角缺口，只有允許對角步時，這個環才是封閉的。中央的 0 在 4-連通下是孤立的（一個洞），在 8-連通下則與外部相連。4/4 搭配得到「不封閉的曲線卻有洞」；8/8 得到「封閉的曲線卻沒有洞」。只有前景 8-連通搭配背景 4-連通（或反過來）才一致。可用 `skimage.measure.label` 驗證。

4. 證明對任意兩個像素都有 $D_8(p,q) \le D_E(p,q) \le D_4(p,q)$，並找出讓每個不等式取等號的像素對。

   提示

   令 $d_x = |x-u|$、$d_y = |y-v|$，則 $\max(d_x,d_y)^2 \le d_x^2 + d_y^2 \le (d_x+d_y)^2$。當 d_x, d_y 其中之一為零時取等號。

5. 寫一個用正向映射旋轉影像 θ 角的函式（把每個輸入像素推到四捨五入後的輸出位置）。顯示 $\theta = 17°$ 的結果，並解釋為什麼會有空洞。然後改用反向映射加雙線性內插修正它。

   提示

   旋轉保持距離不變，但旋轉後的像素中心通常落在輸出網格點之間。前向映射再四捨五入，可能留下空洞，或把多個取樣點送到同一像素。反向映射會逐一拜訪輸出像素，在對應的來源座標內插；來源座標若落在影像之外，仍須明訂邊界處理方式。

6. 你有一個靜止場景的 10 張含雜訊畫面，雜訊標準差為 12 個灰階。要讓它低於 2 個灰階，需要幾張？在真實的手持連拍中，哪個假設可能不成立？手機管線怎麼處理？

   提示

   $12/\sqrt{K} < 2$ 需要 $K > 36$，所以要 37 張。「場景與相機都靜止」這個假設不成立；平均前必須先對齊（對位）各畫面，移動中的物體則需要穩健的合併方式 [7]。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 2. publisher page
2. M. Unser, “Sampling—50 years after Shannon,” *Proceedings of the IEEE*, vol. 88, no. 4, pp. 569–587, 2000. doi
3. 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
4. P. Thévenaz, T. Blu, and M. Unser, “Interpolation revisited,” *IEEE Transactions on Medical Imaging*, vol. 19, no. 7, pp. 739–758, 2000. doi
5. A. Rosenfeld and J. L. Pfaltz, “Distance functions on digital pictures,” *Pattern Recognition*, vol. 1, pp. 33–61, 1968. doi
6. B. Zitová and J. Flusser, “Image registration methods: a survey,” *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, 2003. doi
7. M. Delbracio, D. Kelly, M. S. Brown, and P. Milanfar, “Mobile computational photography: A tour,” arXiv:2102.09000, 2021. arXiv
8. R. Ramanath, W. E. Snyder, Y. Yoo, and M. S. Drew, “Color image processing pipeline,” *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 34–43, 2005. doi
9. Z. Wang, J. Chen, and S. C. H. Hoi, “Deep learning for image super-resolution: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. arXiv
10. B. Moser, F. Raue, S. Frolov, J. Hees, S. Palacio, and A. Dengel, “Hitchhiker’s guide to super-resolution: Introduction and recent advances,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. arXiv
11. Y. Chen, S. Liu, and X. Wang, “Learning continuous image representation with local implicit image function,” CVPR, 2021. arXiv
12. X. Wang, L. Xie, C. Dong, and Y. Shan, “Real-ESRGAN: Training real-world blind super-resolution with pure synthetic data,” arXiv:2107.10833, 2021. arXiv

13. 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. project page
14. G. Zhai and X. Min, “Perceptual image quality assessment: A survey,” *Science China Information Sciences*, vol. 63, no. 11, 2020. doi
15. R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” CVPR, 2018. arXiv
16. K. Ding, K. Ma, S. Wang, and E. P. Simoncelli, “Comparison of full-reference image quality models for optimization of image processing systems,” *International Journal of Computer Vision*, 2021. arXiv
17. C. Chen, Q. Chen, J. Xu, and V. Koltun, “Learning to see in the dark,” CVPR, 2018. arXiv
18. A. Ignatov, L. Van Gool, and R. Timofte, “Replacing mobile camera ISP with a single deep learning model,” arXiv:2002.05509, 2020. arXiv
19. 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
20. R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, 2004. doi
21. scikit-image contributors, “ProjectiveTransform.estimate,” scikit-image 0.25.2 documentation, accessed 5 November 2026. API 文件.

IP 03・強度轉換與空間濾波

用曲線與直方圖重新對應像素值，再用小小的卷積核平滑與銳化影像：空間域影像增強的日常工具箱。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/

你將學到

- 一條曲線 $s = T(r)$ 如何讓影像變亮、變暗、反相或拉開對比，以及何時該選對數、gamma 或分段線性曲線。
- 直方圖等化（histogram equalization）為什麼有效（一步步推導），以及直方圖匹配、CLAHE 這類區域方法與區域統計量如何建立在它之上。
- 什麼是線性空間濾波、相關（correlation）與卷積（convolution）差在哪裡，以及可分離卷積核為什麼快。
- 平滑（低通）濾波器，包括非線性的中值濾波器，如何去除雜訊；以微分為基礎的（高通）濾波器如何銳化。
- 如何從低通卷積核推導出高通、帶通與帶拒卷積核，以及如何把多個方法串成一條處理流程。

先看全貌

這一章談的是：不離開像素格子，直接改善影像。本章所有方法都直接作用在像素值上，這就是「空間域」（spatial domain）的意思。方法分成兩大類。**點運算**（point operation）只用每個像素自己的值來改變它；**鄰域運算**（neighborhood operation，也就是空間濾波器）則參考周圍像素的值來改變它。

> **用最簡單的話說**
>
> 想像一盒蠟筆畫，每一張都畫得太淡了。**點運算**就像替每一種顏色訂一條魔法規則：「淡粉紅變成紅色，淡藍變成深藍。」你只看一個點，查一下規則，然後重新上色。**濾波器**則是在決定之前，先看看這個點和它的鄰居。想讓畫柔和一點，就把每個點塗成鄰居的平均色；想讓輪廓更清楚，就讓每個點和鄰居差得更多。

為什麼重要：這些是大家第一個會拿出來用的工具，手機相機、醫學影像檢視器、顯微鏡，以及電腦視覺模型的前處理都會用到。它們也是卷積層在概念上的祖先：CNN 的第一層就是一組空間濾波器，而許多學習式的影像增強網路，輸出的仍然是一條曲線或一個濾波器，這點會在「現代觀點」中看到。

背景

白話版。我們把輸出影像寫成「對輸入影像套用某個運算」。如果這個運算一次只看一個像素，它就是一張查找表；如果它看一個小視窗，它就是濾波器。

精確版。令 $I(x,y)$ 為輸入影像、$G(x,y)$ 為輸出影像，x,y 為整數像素座標（與卷積教學使用相同的符號）。空間域處理可寫成

$$G(x,y) = T[I](x,y),$$

其中 T 是定義在 (x,y) **鄰域**（neighborhood）上的運算子。鄰域通常是以 (x,y) 為中心的小正方形或長方形。把鄰域中心逐一移過每個像素，就能在每個位置套用這個運算子。

當鄰域只有一個像素（1×1）時，T 只取決於該像素的值，這時稱為**強度轉換**（intensity transformation，也稱灰階轉換或點轉換），並以純量寫成：

$$s = T(r),$$

其中 r 是某像素的輸入強度，s 是同一像素的輸出強度，兩者都落在 $[0, L-1]$，L 為強度階數（8 位元影像 $L = 256$）。因為 r 只有 L 種可能值，任何 T 都能存成長度為 L 的查找表（lookup table），這也是點運算極快的原因。

基本強度轉換函數

白話版。畫一條曲線，橫軸是輸入亮度、縱軸是輸出亮度。曲線陡的地方，相近的灰階會被拉開，對比增加；曲線平的地方，灰階被擠在一起，對比下降。

斜率 $T'(r)$ 就是局部的對比增益：兩個相鄰像素間的小差異 Δr，轉換後大約變成 $T'(r)\,\Delta r$。下面每一條曲線，都是在決定要把這份增益花在哪裡。

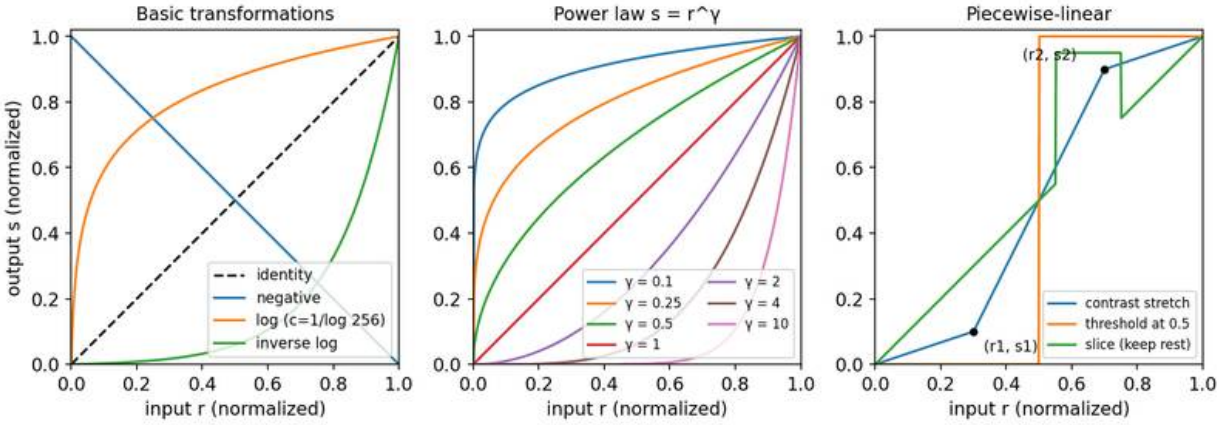


圖 *3.1* — 常見的強度轉換曲線（強度正規化到 *[0, 1]*）。左：反相、對數與反對數。中：冪次（*gamma*）曲線。右：分段線性曲線。

影像反相

$$s = (L - 1) - r.$$

暗變亮、亮變暗。斜率處處為 -1，所以對比的大小不變，只是極性反轉。當大片暗區中有細小的亮細節時，反相很有幫助，因為人通常比較容易在淺色背景上判讀灰階細節。

對數轉換

$$s = c \log(1 + r),$$

其中 $c > 0$ 為縮放常數，通常取 $c = (L - 1)/\log L$，讓最大輸入對應到 $L - 1$。加 1 是為了避開 $\log 0$。斜率 $c/(1 + r)$ 在 r 小時大、在 r 大時小，因此對數曲線會擴張暗部、壓縮亮部。

它的經典用途是顯示動態範圍極大的資料，例如傅立葉頻譜的強度，數值可以橫跨好幾個數量級。線性縮放時只看得到最亮的那一點；取對數之後，結構就浮現了（圖 3.2 下排）。**反對數**（指數）曲線的作用則恰好相反。

冪次（gamma）轉換

$$s = c\, r^{\gamma},$$

其中 $c > 0$、$\gamma > 0$ 為常數（若 r 已正規化到 $[0, 1]$，取 $c = 1$）。$\gamma < 1$ 時曲線往上拱，像對數一樣提亮暗部；$\gamma > 1$ 時曲線往下彎，壓暗暗部。和對數不同的是，一個參數就能產生一整族曲線（圖 3.1 中）。

Gamma 的重要性不只在增強。顯示器與相機都有冪次型的響應，影像檔通常儲存的是經過 gamma 編碼的值，而不是與光量成正比的值。預先補償裝置響應的做法稱為 **gamma 校正**（gamma correction）。一個實務上的後果是：對 gamma 編碼過的像素做平均、模糊或縮放，和在線性光上做並不相同，所以嚴謹的流程會先線性化。

```
import numpy as np
from skimage import data, img_as_float

r = img_as_float(data.camera())        # intensities scaled to [0, 1]
negative = 1.0 - r                     # s = (L-1) - r, in normalized form
log_t = np.log1p(r) / np.log(2.0)      # s = c log(1 + r), c chosen so s(1) = 1
bright = r ** 0.4                      # gamma < 1 lifts the shadows
dark = r ** 2.5                        # gamma > 1 deepens them
```

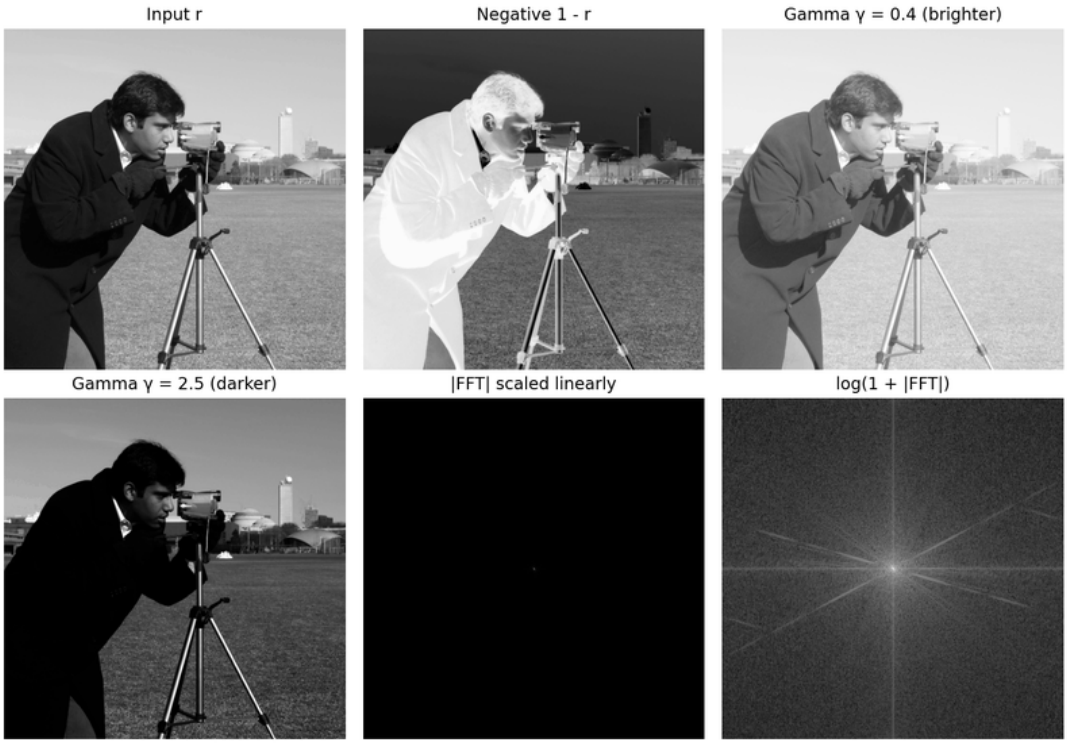


圖 *3.2* — 上排：輸入、反相、*gamma 0.4*。下排：*gamma 2.5*；線性縮放的傅立葉強度（幾乎全黑）；同一強度取 $log(1+\cdot)$ 之後。

分段線性轉換

不用單一公式，而是把幾段直線接起來。好處是控制精確，代價是參數較多。

對比拉伸（contrast stretching）。選兩個控制點 (r_1, s_1) 與 (r_2, s_2)，用直線連接 $(0,0) \to (r_1, s_1) \to (r_2, s_2) \to (L-1, L-1)$。中間那段的斜率是 $(s_2 - s_1)/(r_2 - r_1)$。只要 $r_1 \le r_2$ 且 $s_1 \le s_2$，曲線就是單調的，不會顛倒灰階的順序。有兩個特例值得記住：

- $r_1 = r_{\min}$、$s_1 = 0$、$r_2 = r_{\max}$、$s_2 = L-1$：把影像實際的範圍對應到完整範圍（**最小–最大拉伸**）。實務上改用第 1 與第 99 百分位數取代最小值與最大值，避免少數離群像素主導結果。
- $r_1 = r_2 = m$、$s_1 = 0$、$s_2 = L-1$：產生二值影像，也就是在 m 處做**門檻化**（threshold）。

```
def stretch(img, r1, s1, r2, s2, L=256):
    lut = np.interp(np.arange(L), [0, r1, r2, L - 1], [0, s1, s2, L - 1])
    return lut.astype(np.uint8)[img]

img = data.camera()
lo, hi = np.percentile(img, [1, 99])
out = stretch(img, lo, 0, hi, 255)     # robust min-max stretch
```

強度切片（intensity-level slicing）。突顯某一段強度 $[A, B]$。做法有兩種：把這段設成高值、其他設成低值（得到二值結果）；或只提亮這段、其他保持不變。當感興趣的灰階範圍已知時（例如 X 光中的某種組織或材料），切片就很好用。

位元平面切片（bit-plane slicing）。8 位元像素值是八個位元的加總，把每個像素的同一個位元收集起來，就得到一張二值影像，稱為**位元平面**（bit plane）。最高幾個平面承載看得見的結構，最低幾個平面則像雜訊。圖 3.3 顯示，只用最高四個平面就能重建出與原圖非常接近的影像，這說明了位元平面為何與壓縮有關，也說明最低幾個平面為何常被拿來藏浮水印。

僅線上版：位元平面詳解 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/notes/位元平面詳解/*

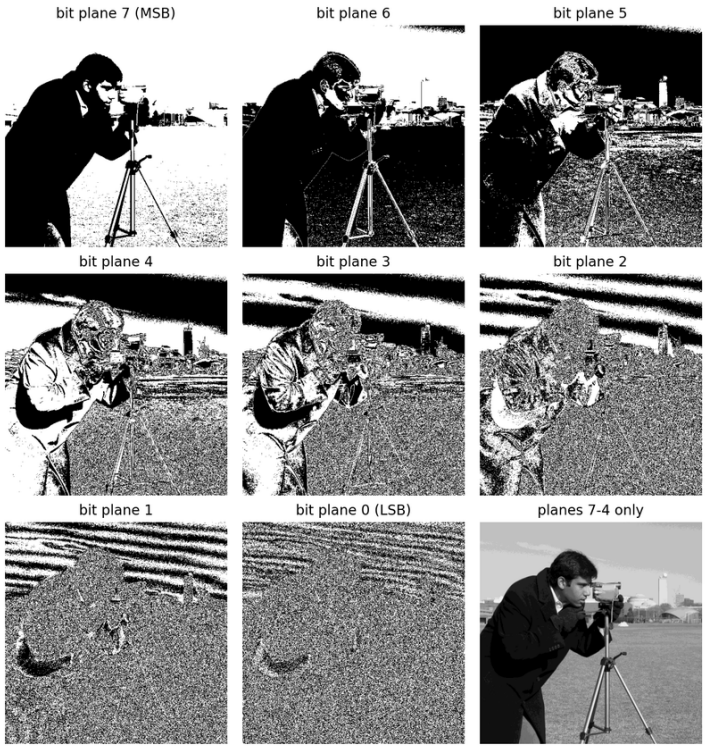


圖 *3.3 — 8* 位元影像的八個位元平面，從最高有效位元（左上）到最低有效位元，以及只用平面 *7–4* 重建的影像。

直方圖處理

白話版。直方圖（histogram）統計每種亮度各有多少像素。暗影像的計數堆在左邊；灰濛濛的影像則擠在一條窄帶裡。直方圖處理就是根據這些計數自動選出曲線 T，讓輸出的直方圖變成我們想要的形狀。

精確版。對一張 $M \times N$、灰階為 r_k（$k = 0, \ldots, L-1$）的影像，令 n_k 為值等於 r_k 的像素數。**正規化直方圖**

$$p_r(r_k) = \frac{n_k}{MN}$$

估計了「隨機挑一個像素，其值為 r_k」的機率，總和為 1。

直方圖等化

目標是找一條曲線 T，讓輸出直方圖盡可能平坦，也就是每個灰階被使用的頻率大致相同。

推導（連續情形）。把強度視為連續隨機變數 $r \in [0, L-1]$，其機率密度函數（PDF）為 $p_r(r)$。假設 T 在此區間嚴格遞增。那麼像素落在小區間 $[r, r+dr]$，恰好等於輸出落在 $[s, s+ds]$，其中 $s = T(r)$。機率相等給出 $p_s(s)\,|ds| = p_r(r)\,|dr|$，因此

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right|.$$

現在把 T 選為 r 的累積分布函數（CDF）乘上比例：

$$s = T(r) = (L-1)\int_0^r p_r(w)\,dw,$$

其中 w 是積分用的虛擬變數。由微積分基本定理，$ds/dr = (L-1)\,p_r(r)$。代入得

$$p_s(s) = p_r(r)\cdot\frac{1}{(L-1)\,p_r(r)} = \frac{1}{L-1}, \qquad 0 \le s \le L-1.$$

不論輸入的 PDF 長什麼樣，輸出的 PDF 都是均勻分布。CDF 正是那條能把機率平均攤開的單調曲線：像素多的地方它陡（把像素拉開），像素少的地方它平。

離散版本。把積分換成加總：

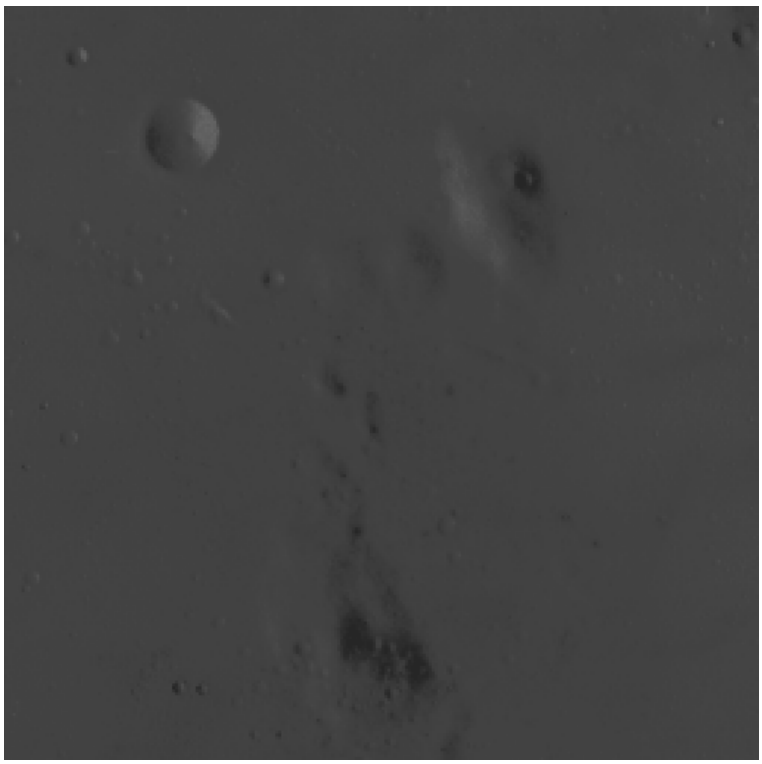
$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j), \qquad k = 0, 1, \dots, L-1,$$

再把 s_k 四捨五入到最近的整數。由於多個輸入灰階可能捨入到同一個輸出灰階，而一個灰階又無法拆開，離散結果只會近似平坦。輸出直方圖通常會出現高聳的尖峰，中間夾著空隙（圖 3.4 中）。真正能保證的是：輸出涵蓋完整範圍，且輸出的 CDF 接近一條直線。

```
def equalize(img, L=256):
    hist = np.bincount(img.ravel(), minlength=L)      # n_k
    cdf = np.cumsum(hist) / img.size                  # sum of p_r(r_j), j <= k
    lut = np.round((L - 1) * cdf).astype(np.uint8)    # s_k
    return lut[img]
```

OpenCV 的 `cv2.equalizeHist` 使用稍微不同的正規化 $(\mathrm{cdf}_k - \mathrm{cdf}_{\min})/(MN - \mathrm{cdf}_{\min})$，讓最暗的有值灰階對應到 0，結果大約只差一階。

拖動滑桿，比較一張刻意壓低對比的影像與它的全域等化結果。注意等化並不是「免費的」：它會誇大大片平滑區域與顆粒感，因為它把對比花在像素數量多的地方，而不是有意義的細節所在之處。



輸入



等化後

一張昏暗、平淡的月面影像，與其直方圖等化後對比強烈的版本。

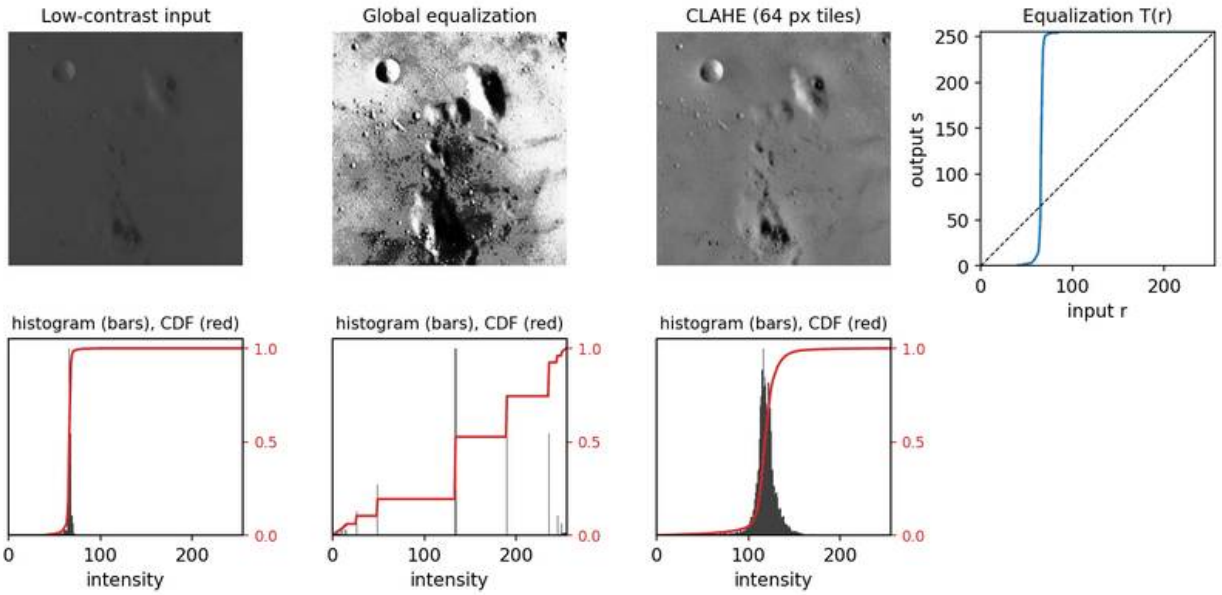


圖 *3.4* — 全域直方圖等化與 *CLAHE* 的比較。輸入只占據很窄的灰階範圍，所以等化曲線 $T(r)$（右）幾乎是階梯，輸出直方圖被拉開且出現空隙。*CLAHE* 限制斜率，並依區域調整。

直方圖匹配（規定化）

有時候平坦的直方圖並不是我們要的目標。我們可能希望影像看起來像某張參考影像，或希望直方圖是手動指定的形狀。**直方圖匹配**（histogram matching，也稱 histogram specification）把輸入對應成直方圖接近指定 PDF $p_z(z)$ 的影像。

技巧是用兩次等化。等化輸入得到均勻的 $s = T(r)$；對目標分布做等化，得到

$$G(z) = (L-1)\int_0^z p_z(t)\,dt,$$

它同樣產生均勻變數。兩個均勻變數可以互相對應，因此令 $G(z) = T(r)$ 並解出：

$$z = G^{-1}(T(r)).$$

其中 z 是輸出強度、G 是目標分布的縮放 CDF、t 是虛擬變數。離散情形下 G 是階梯函數，未必可逆，所以對每個輸入灰階取滿足 $G(z) \geq T(r)$ 的最小 z，只要一個 `searchsorted` 就能完成：

```
def match(img, ref, L=256):
    cdf_in = np.cumsum(np.bincount(img.ravel(), minlength=L)) / img.size
    cdf_ref = np.cumsum(np.bincount(ref.ravel(), minlength=L)) / ref.size
    # For each input level, the smallest z with G(z) >= T(r).
    lut = np.searchsorted(cdf_ref, cdf_in).clip(0, L - 1).astype(np.uint8)
    return lut[img]
```

`skimage.exposure.match_histograms` 實作了相同的概念，也支援彩色影像的逐通道匹配。

等化與匹配都是查找表，所以輸入值相同的像素一定得到相同的輸出值；一大群相同值的像素無法被拆開，輸出直方圖只能近似目標。精確直方圖指定（exact histogram specification）[5] 先用區域平均打破平手、把像素嚴格排序，從而去除這個限制。

僅線上版：精確直方圖匹配 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/notes/精確直方圖匹配/*

區域直方圖處理

全域方法對整張影像只用一條曲線。如果重要細節藏在暗角落、而那裡的像素又不多，全域直方圖幾乎不會注意到它。

區域直方圖等化對每個像素周圍的鄰域計算直方圖，由此算出轉換，只套用到中心像素，再移到下一個像素。**自適應直方圖等化**（adaptive histogram equalization, AHE）及其變體由 Pizer 等人做了系統性的研究，其中包括 AHE 在近乎均勻的區域會過度放大雜訊的問題：那裡的區域直方圖只是一根尖峰，CDF 變成近乎垂直的階梯 [2]。

對比受限自適應直方圖等化（contrast-limited AHE, CLAHE）透過限制對應曲線的斜率來解決這個問題。由於等化曲線的斜率與直方圖高度成正比，把每個區域直方圖在某個上限處截斷，再把截掉的計數重新分配到所有 bin，就能為對比增益設上限。為了加速，影像被切成小塊（tile），每塊算一條對應曲線，每個像素的輸出由最近四塊的對應結果做雙線性內插，避免出現明顯的接縫。Zuiderveld 在 Graphics Gems 的章節給出了被廣泛沿用的實作 [3]，Reza 則描述了即時的硬體實現 [4]。

```
import cv2
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
out = clahe.apply(data.moon())         # uint8 in, uint8 out
```

在 scikit-image 中，對應的是 `skimage.exposure.equalize_adapthist(img, clip_limit=0.01)`。

利用直方圖統計量增強影像

直方圖的平均衡量平均亮度，標準差衡量對比。若改在每個像素周圍的小視窗中計算，就得到**區域平均**與**區域標準差**，讓我們只增強需要增強的區域。典型的規則是：只有當鄰域偏暗（區域平均低），而且有一些但不太多的對比（區域標準差適中）時，才放大該像素，藉此跳過平坦背景與強烈邊緣。兩個區域統計量只需要兩次方框濾波，一次作用在 I、一次作用在 I^2。

僅線上版：以區域直方圖統計量增強影像 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/notes/以區域直方圖統計量增強影像/*

空間濾波的基礎

白話版。空間濾波器把一小格權重（**卷積核**，kernel）滑過整張影像。在每個位置，把權重乘上底下的像素再加總，結果就是中心像素的新值。

卷積教學已經用從零寫起的 NumPy 迴圈、OpenCV 呼叫與邊界處理，把操作細節講清楚了。這裡專注於本章後面需要的理論。

線性空間濾波與相關

沿用該教學的符號，對大小為 $(2k+1)\times(2k+1)$ 的卷積核 K，K 與 I 的**相關**（correlation）為

$$(K \star I)(x,y) = \sum_{i=-k}^{k}\sum_{j=-k}^{k} K(i,j)\, I(x+i,\, y+j),$$

其中 (i,j) 是相對於卷積核中心的位移，k 是卷積核的半寬。奇數尺寸讓卷積核有明確的中心。這個濾波器是**線性**的，因為輸出是輸入值的線性組合：對 $aI_1 + bI_2$ 濾波，等於 a 倍的 I_1 結果加上 b 倍的 I_2 結果。它也是**平移不變**的，因為每個位置用的是同一組權重。

相關與卷積

卷積是同樣的計算，只是先把卷積核旋轉 180°：

$$(K * I)(x,y) = \sum_{i=-k}^{k}\sum_{j=-k}^{k} K(i,j)\, I(x-i,\, y-j).$$

對旋轉 180° 後不變的對稱卷積核（方框、高斯、拉普拉斯），兩者相同；對反對稱的卷積核（Sobel、任何方向導數），兩者結果相同但正負號相反。真正有用的差別出現在**脈衝**（impulse，一堆 0 中的一個 1）上：卷積核與脈衝做相關，得到的是旋轉過的卷積核；做卷積，得到的則是一模一樣的卷積核。這就是為什麼代數性質漂亮的是卷積而不是相關：

- **交換律：** $K * I = I * K$，
- **結合律：** $K_1 * (K_2 * I) = (K_1 * K_2) * I$，所以一連串濾波器可以合併成單一卷積核，
- **分配律：** $K * (I_1 + I_2) = K * I_1 + K * I_2$。

接下來兩小節都靠結合律。就像卷積教學提到的，`cv2.filter2D` 計算的是相關；大多數深度學習的「卷積」層其實也是相關，但因為權重是學出來的，翻不翻轉並不重要。

可分離卷積核

若卷積核可以寫成行向量 $\mathbf{c}$ 與列向量 $\mathbf{r}$ 的外積，就稱它是**可分離的**（separable）：

$$K = \mathbf{c}\,\mathbf{r}^{\top}.$$

於是 $K * I = \mathbf{c} * (\mathbf{r}^\top * I)$：先用 $\mathbf{r}$ 濾每一列，再用 $\mathbf{c}$ 濾每一行。對 $m \times n$ 的卷積核，每個像素的乘法次數從 mn 降為 $m + n$，加速比為 $mn/(m+n)$，7×7 時是 3.5 倍，且隨尺寸線性成長。矩陣恰好是外積若且唯若它的**秩**（rank）為 1，這也提供了一個檢驗方法：做奇異值分解，確認只有一個奇異值不為零。方框與高斯卷積核都可分離；Sobel 也可以，它是一個平滑向量乘上一個差分向量。

```
g1 = cv2.getGaussianKernel(7, 1.5)          # 7x1 column, sums to 1
K = g1 @ g1.T                               # 7x7 Gaussian kernel
print(np.linalg.matrix_rank(K))             # 1  ->  separable
f = img_as_float(data.camera()).astype(np.float32)
a = cv2.filter2D(f, -1, K.astype(np.float32))
b = cv2.sepFilter2D(f, -1, g1, g1)          # two 1-D passes
print(np.abs(a - b).max())                  # ~1e-7
```

空間域與頻域

卷積定理指出，空間中的卷積等於頻率中的相乘：若 $\mathcal{F}$ 表示傅立葉轉換，則 $\mathcal{F}\{K * I\} = \mathcal{F}\{K\} \cdot \mathcal{F}\{I\}$。因此每個線性、平移不變的濾波器都有兩種等價描述：空間中的卷積核 K，以及說明每個頻率保留多少的**轉移函數**（transfer function）$H = \mathcal{F}\{K\}$。平滑卷積核的轉移函數在零頻率附近大、在高頻小，所以是**低通**（lowpass）；銳化卷積核相反，是**高通**（highpass）。第 4 章會正式發展這個觀點；目前只要記得這組詞彙：「低通」就是「平滑」，「高通」就是「邊緣與細節」。

小卷積核通常在空間域計算比較便宜；非常大的卷積核則常透過快速傅立葉轉換在頻域計算比較划算。

建構空間濾波卷積核

常見的做法有三種：

1. **從數學性質出發**。取平均得到方框卷積核；取導數得到差分卷積核（拉普拉斯、Sobel）。
2. **對連續函數取樣**。在整數格點上對二維高斯取樣，再把權重正規化成總和為 1。
3. **從頻率規格出發**。在頻域設計 H，做反轉換得到空間卷積核，通常再截斷成可處理的大小。

所有做法都適用兩個檢查。平滑卷積核的權重總和應為 1，平坦區域才能保持亮度；導數卷積核的權重總和應為 0，平坦區域才會輸出 0。

平滑（低通）空間濾波器

白話版。把每個像素換成鄰域的平均。隨機雜訊被拉向平均而淡去，但銳利的邊緣也會被抹開，因為平均跨過了邊緣兩側。

方框濾波器

$m \times n$ 方框卷積核（box kernel）的每個權重都是 $1/(mn)$。它是最簡單的低通濾波器，可分離，而且搭配累加和（running sum）時，每個像素的計算量與尺寸無關。它的弱點在頻率響應：方框

的傅立葉轉換是類似 sinc 的函數，帶有旁瓣（side lobe），所以會讓部分高頻以反相的方式通過，而且它以正方形、與方向有關的方式抹開影像，細小文字與細線可能變成塊狀條紋。

高斯濾波器

高斯卷積核是對下式取樣：

$$w(s,t) = C\,e^{-\frac{s^2+t^2}{2\sigma^2}},$$

其中 (s,t) 是相對於卷積核中心的位移，σ（標準差）決定模糊程度，C 讓權重總和為 1。它有三個性質，讓它成為預設的平滑卷積核：

- **等向性**（isotropy）。它只取決於距離 $\sqrt{s^2+t^2}$，所以各方向模糊程度相同。
- **可分離性**。$e^{-(s^2+t^2)/2\sigma^2} = e^{-s^2/2\sigma^2}\,e^{-t^2/2\sigma^2}$。
- **平滑且非負的頻率響應**。它的傅立葉轉換也是高斯，沒有旁瓣。

一維高斯約 99.7% 的質量落在 $\pm 3\sigma$ 之內，所以約 6σ 大小（向上取到最近的奇數）的卷積核就幾乎涵蓋全部。更大只是浪費計算，更小則會截掉尾巴。兩個標準差分別為 σ_1、σ_2 的高斯做卷積，得到 $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$ 的高斯，所以重複的小模糊可以預測地組合起來。

順序統計（非線性）濾波器

順序統計濾波器（order-statistic filter）把鄰域中的值**排序**後挑一個出來。它們是非線性的：沒有任何卷積核能表示它們，卷積定理也不適用。

- **中值濾波器**（median filter）：輸出中間值（第 50 百分位數）。它是處理**脈衝雜訊**（impulse noise，又稱椒鹽雜訊 salt-and-pepper）的標準方法，也就是被強制變成全黑或全白的孤立像素。脈衝是極端值，排序後會跑到兩端，幾乎不可能是中值。只要視窗中被汙染的像素少於一半，中值就會忽略它們。在同樣大小下，它保留階梯邊緣的能力也遠勝線性平均。
- **最大值濾波器**（第 100 百分位數）：找出最亮的點；能去除胡椒（暗）雜訊，但會讓亮點長大。
- **最小值濾波器**（第 0 百分位數）：相反；能去除鹽（亮）雜訊，但會讓暗點長大。

最大值與最小值濾波器會在第 9 章以灰階膨脹與侵蝕的形式再次出現。

```
from scipy import ndimage as ndi

f = img_as_float(data.camera()).astype(np.float32)
noisy = f.copy()
u = np.random.default_rng(0).random(f.shape)
noisy[u < 0.05], noisy[u > 0.95] = 0.0, 1.0       # 10% salt-and-pepper

box = cv2.blur(noisy, (5, 5))
gauss = cv2.GaussianBlur(noisy, (0, 0), sigmaX=1.5)
med = cv2.medianBlur(noisy, 3)
mn, mx = ndi.minimum_filter(noisy, 3), ndi.maximum_filter(noisy, 3)
```

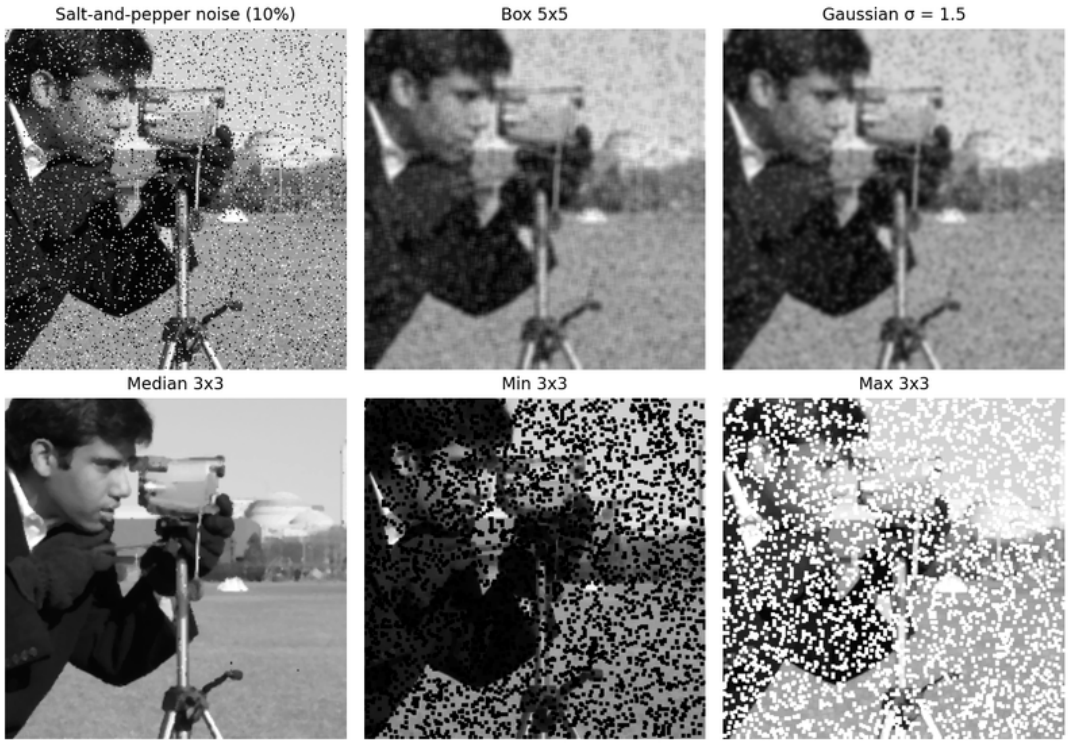


圖 *3.5* — 平滑 *10%* 椒鹽雜訊。線性濾波器（方框、高斯）把每個脈衝攤成灰色汙漬；*3×3* 中值濾波器幾乎完全去除雜訊；最小值與最大值各自去除一種極性，卻放大另一種。

銳化（高通）空間濾波器

白話版。平滑是取平均；銳化則相反，要突顯差異。鄰居之間的差異就是導數，所以銳化是用導數做出來的。

一階與二階導數

對像素格點上的一維訊號 $f(x)$，最簡單的近似是

$$\frac{\partial f}{\partial x} = f(x+1) - f(x), \qquad \frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x).$$

圖 3.6 把兩者套用在一條手工製作的剖面上，其中有一段斜坡、一條一個像素寬的細線和一個階梯。從圖中可以讀出選擇導數的規則：

- 在**平坦**區域，兩種導數都是 0。
- 沿著**斜坡**，一階導數一路都不為零，因此響應很粗；二階導數只在斜坡的起點與終點不為零。
- 在**細線**處，二階導數有強烈的雙重響應（正、強負、正），而且幅度比一階導數大。細小的細節正是銳化該強調的東西。

- 在**階梯**處，二階導數產生相鄰的一正一負值，兩者之間的**零交越**（zero crossing）可精確定位邊緣，第 10 章會把它用在邊緣偵測上。

所以要增強細節時，二階導數是比較自然的選擇。

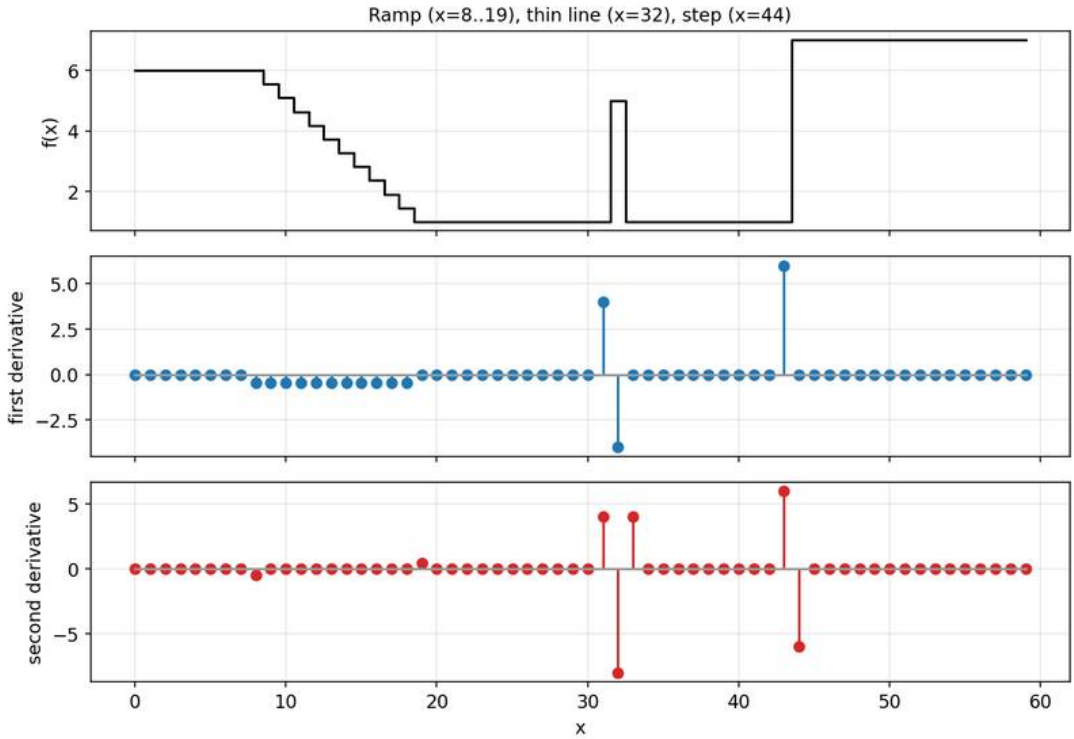


圖 *3.6* — 合成剖面的一階與二階差分。二階導數忽略斜坡內部、對細線反應最強，並在階梯兩側改變正負號。

拉普拉斯運算子

最簡單的**等向性**二階導數運算子（影像旋轉時響應不變）是**拉普拉斯運算子**（Laplacian）：

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}.$$

把上面的一維差分分別用在 x 與 y 方向並相加，得到

$$\nabla^2 I(x,y) = I(x+1,y) + I(x-1,y) + I(x,y+1) + I(x,y-1) - 4I(x,y),$$

也就是與下列卷積核做相關：

$$K_{\nabla^2} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

若再加入兩個對角方向，會得到外圈八格都是 1、中心為 −8 的版本，它以 45° 而非 90° 為間隔具有等向性。把所有正負號翻轉，同樣是有效的卷積核。權重總和為 0，所以平坦區域會消失，只留下變化。

用拉普拉斯銳化。拉普拉斯影像大多是灰色（零），在邊緣處有亮暗的鑲邊。把它加回原圖，就把這些鑲邊疊到邊緣上：

$$G(x,y) = I(x,y) + c\,\nabla^2 I(x,y),$$

其中中心為負的卷積核（如上面的 K_{∇^2}）取 $c = -1$，中心為正的卷積核取 $c = +1$。正負號很重要：選錯號，邊緣反而會被模糊。由於卷積具有分配律，$I - \nabla^2 I$ 本身就是單一卷積核：中心為 5、上下左右為 −1，正是下方實驗室中的 `sharpen` 預設。

> **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

試試 `gaussian` 與 `box-blur` 預設來平滑，再試 `laplacian` 看純粹的二階導數響應。把 sharpen 卷積核的中心從 5 改成 6，權重總和就變成 2，整張影像會變亮，這說明了總和為什麼重要。

反銳化遮罩與高增強濾波

這是來自傳統暗房的技巧：減去一份模糊的副本來分離出細節，再把細節加回去。

$$g_{\text{mask}}(x,y) = I(x,y) - \bar{I}(x,y), \qquad G(x,y) = I(x,y) + \alpha\, g_{\text{mask}}(x,y),$$

其中 $\bar{I}$ 是 I 的模糊（低通）版本，g_{mask} 是**反銳化遮罩**（unsharp mask，即被模糊去掉的細節），$\alpha \geq 0$ 是權重。$\alpha = 1$ 時稱為**反銳化遮罩**（unsharp masking）；$\alpha > 1$ 時稱為**高增強濾波**（highboost filtering）；$0 < \alpha < 1$ 時效果溫和。因為 g_{mask} 是高通訊號，這是另一種加入高通細節的方式；若 $\bar{I}$ 是高斯模糊，模糊的 σ 就決定了哪種大小的細節會被增強。

α 太大會產生明顯的**光暈**（halo）：強烈邊緣兩側出現過衝，輸出也可能超出有效範圍，所以要裁切（clip）。

梯度與 Sobel 運算子

一階導數是以**梯度**（gradient）的形式使用，也就是偏導數組成的向量

$$\nabla I = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \partial I/\partial x \\ \partial I/\partial y \end{bmatrix},$$

它指向強度增加最快的方向。它的長度，即**梯度大小**

$$M(x,y) = \sqrt{g_x^2 + g_y^2} \approx |g_x| + |g_y|,$$

在邊緣處大、在平坦區域接近 0。右邊的近似省去了開根號，但不具等向性。

Sobel 運算子用 3×3 卷積核估計 g_x 與 g_y：

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad K_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

兩者都可分離：$K_x = [1,2,1]^\top [-1,0,1]$，即一個方向的中央差分乘上另一個方向的輕微平滑。中間的 2 讓中央那一列或那一行的權重較大，降低運算子對雜訊的敏感度。（卷積教學就是用 K_x 當作邊緣偵測的例子。）梯度大小很少直接拿來銳化，更常用來找出邊緣，或當作遮罩來決定哪裡允許銳化，就像下面的綜合範例。

```
f = img_as_float(data.chelsea()[..., 1]).astype(np.float32)
lap_k = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]], np.float32)
lap = cv2.filter2D(f, -1, lap_k)
sharp = np.clip(f - lap, 0, 1)                # c = -1 because the center is negative

blur = cv2.GaussianBlur(f, (0, 0), 2)
mask = f - blur                               # g_mask
unsharp = np.clip(f + 1.0 * mask, 0, 1)       # alpha = 1
highboost = np.clip(f + 3.0 * mask, 0, 1)     # alpha = 3

gx = cv2.Sobel(f, cv2.CV_32F, 1, 0, ksize=3)
gy = cv2.Sobel(f, cv2.CV_32F, 0, 1, ksize=3)
M = np.hypot(gx, gy)                          # gradient magnitude
```

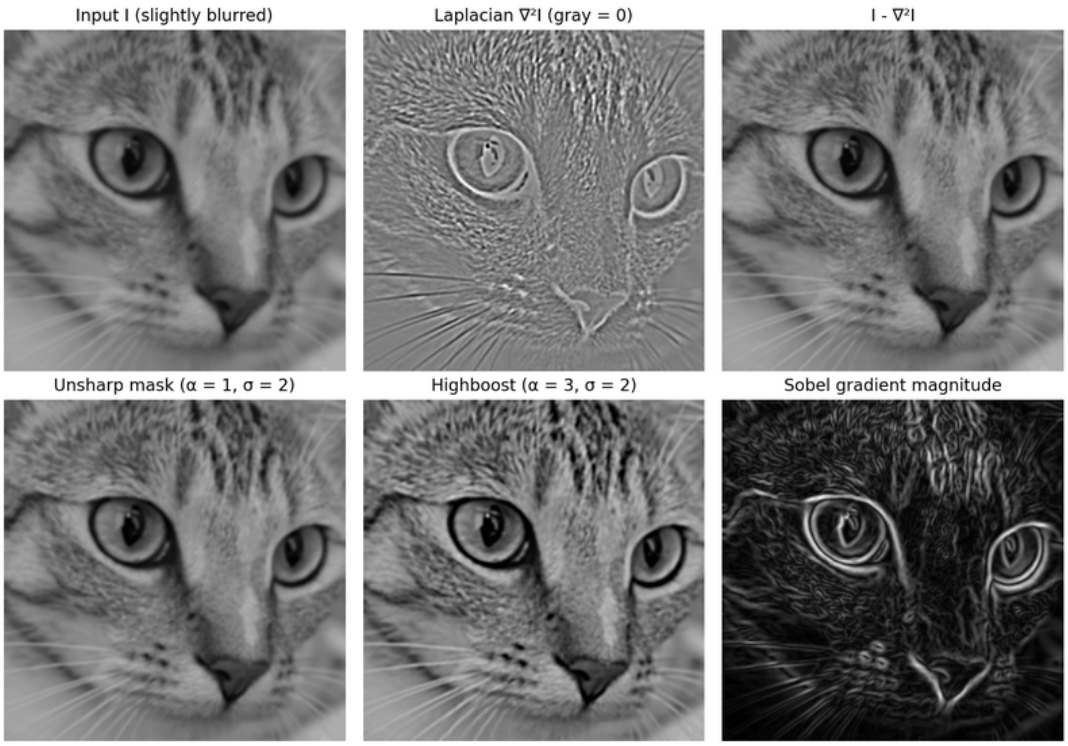


圖 *3.7* — 銳化一張稍微模糊的影像。上排：輸入、拉普拉斯（灰色代表零）、$I - \nabla^2 I$。下排：反銳化遮罩、高增強濾波、*Sobel* 梯度大小。

濾波時一律使用浮點數。`uint8` 的流程會把拉普拉斯或 Sobel 響應的負值裁成 0，悄悄丟掉一半的資訊。

由低通濾波器構成高通、帶拒與帶通濾波器

白話版。如果你知道怎麼保留平滑的部分，你也就知道怎麼保留平滑部分以外的一切：相減就好。

令 δ 為**單位脈衝卷積核**：中心為 1、其餘為 0，因此 $\delta * I = I$。它的轉移函數在所有頻率都等於 1（全通濾波器）。若 K_{LP} 是轉移函數為 H_{LP} 的低通卷積核，則

$$K_{\mathrm{HP}} = \delta - K_{\mathrm{LP}} \quad \Longleftrightarrow \quad H_{\mathrm{HP}} = 1 - H_{\mathrm{LP}}.$$

反銳化遮罩正是如此：$g_{\mathrm{mask}} = I - \bar{I} = (\delta - K_{\mathrm{LP}}) * I$。

要隔離某一**頻帶**，就組合兩個低通濾波器。令 K_1 為 σ_1 較小的高斯（截止頻率高，保留較多細節），K_2 為 σ_2 較大的高斯（截止頻率低）。則

$$K_{\mathrm{BP}} = K_1 - K_2, \qquad K_{\mathrm{BR}} = \delta - K_{\mathrm{BP}} = K_2 + (\delta - K_1).$$

帶通（bandpass）卷積核 K_{BP} 保留 K_1 讓過、但 K_2 擋下的頻率。**帶拒**（bandreject）卷積核 K_{BR} 是一個低通加一個高通，只去除中間那段頻帶。帶通 $K_1 - K_2$ 就是**高斯差**（difference of Gaussians, DoG），它也是高斯拉普拉斯的良好近似，並且是第 12 章尺度空間偵測器的基礎。權重總和可以驗證設計：高通與帶通卷積核總和為 0，帶拒卷積核總和為 1。

```
def gauss_kernel(sigma):
    n = 2 * int(np.ceil(3 * sigma)) + 1          # about 6 sigma, odd
    g = cv2.getGaussianKernel(n, sigma)
    return g @ g.T

def delta_like(K):
    D = np.zeros_like(K); D[K.shape[0] // 2, K.shape[1] // 2] = 1
    return D

lp2 = gauss_kernel(4.0)                          # low cutoff
lp1 = gauss_kernel(1.0)                          # high cutoff
lp1 = np.pad(lp1, (lp2.shape[0] - lp1.shape[0]) // 2)   # zero-pad to the same size
hp = delta_like(lp1) - lp1                       # highpass, sums to 0
bp = lp1 - lp2                                   # bandpass (DoG), sums to 0
br = delta_like(bp) - bp                         # bandreject, sums to 1
```

結合多種空間增強方法

白話版。真實影像很少只有一個問題。有用的增強通常是一條短短的處理流程，而且步驟的順序很重要。

以下是一個實作範例。輸入（圖 3.8a）是一張曝光不足的人像，帶有少量高斯雜訊，以及 4% 的椒鹽脈衝。我們希望得到更亮、更清晰的影像，又不放大雜訊。

1. **先用 3×3 中值濾波器去除脈衝**（b）。脈衝是極端值，後續任何拉對比或銳化都會讓它們更糟，而線性模糊只會把它們攤開。中值濾波能去除它們又保留邊緣。
2. **用 gamma 0.45 提亮暗部**（c）。全域的亮度問題，正適合用點運算處理。在中值之後才做，曲線就不會被黑白脈衝干擾。
3. **建立邊緣權重圖**（d）。用高斯（$\sigma = 2$）模糊影像，計算模糊後影像的 Sobel 梯度大小 M，再正規化成 $w = \min(M/M_{90}, 1)$，其中 M_{90} 是 M 的第 90 百分位數。因為是在平滑之後計算，w 在真正的邊緣上接近 1，在平坦區域殘留的雜訊上接近 0。
4. **套用由權重圖控制的反銳化遮罩**（f）：$G = I_c + \alpha\, w \cdot (I_c - \bar{I}_c)$，其中 I_c 是 gamma 校正後的影像、$\bar{I}_c$ 是它的高斯模糊、$\alpha = 2$，乘法逐像素進行。

(e) 是用相同 α 的一般反銳化遮罩：背景與深色衣領都變得粗糙，因為雜訊同樣是高頻細節。受控版本 (f) 對臉、頭髮與太空衣輪廓的銳化程度幾乎一樣，卻讓平滑的背景保持乾淨。這種「用平滑後的一階導數影像來決定哪裡要做二階銳化」的模式，可以看作「現代觀點」中保邊處理的手工版本。

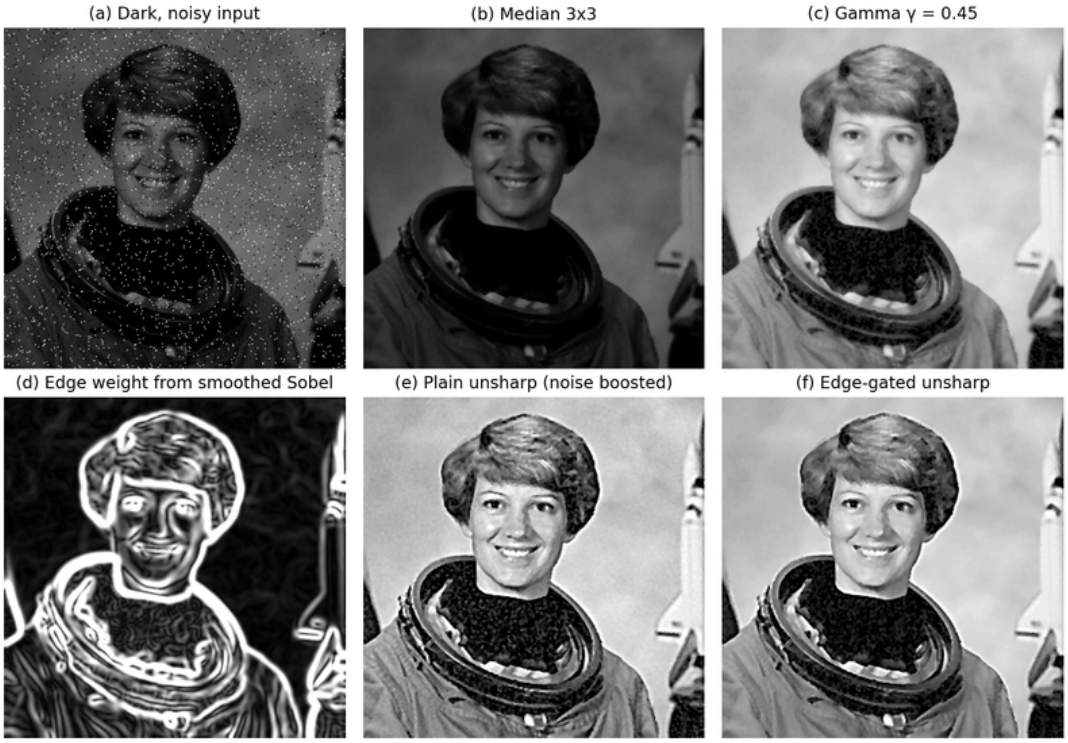


圖 *3.8* — 四步驟的增強流程：(*a*) 輸入、(*b*) *3×3* 中值、(*c*) *gamma 0.45*、(*d*) 由平滑後 *Sobel* 大小得到的邊緣權重、(*e*) 供比較的一般反銳化遮罩、(*f*) 由 (*d*) 控制的反銳化遮罩。

每一步都用到本章的一個概念。步驟順序遵循三條經驗法則：在任何會放大差異的步驟之前先去除離群值；用點運算修正整體色調；只在訊號強於雜訊的地方銳化。

現代觀點

本章的技術已有數十年歷史，卻至今隨處可見：CLAHE 是醫學影像與遙測的標準前處理步驟，gamma 與色調曲線存在於每一條相機處理流程，高斯、中值與 Sobel 濾波器則存在於每一套視覺函式庫。後來的研究朝三個方向推進：更聰明的對比增強、能平滑卻不模糊邊緣的濾波器，以及兩者的學習式替代方案。下面的綜述是很好的入門點。

對比增強與直方圖方法。 Pizer 等人 [2] 至今仍是自適應直方圖等化的代表性分析，Zuiderveld 的章節 [3] 是各函式庫沿用的精簡 CLAHE 實作，Coltuc、Bolon 與 Chassery [5] 則說明如何得到完全符合目標的直方圖。若想要更全面的領域地圖，Qi 等人 [11] 綜述了過去二十年的影像增強方法，分為監督式與非監督式演算法，並把品質評估列為第三個面向。

保邊濾波器。方框與高斯濾波器會跨越邊緣取平均。**雙邊濾波器**（bilateral filter）[6] 再乘上一個隨強度差異遞減的權重，讓邊緣另一側的像素幾乎沒有貢獻。**導引濾波器**（guided filter）[8] 把輸出建模為導引影像的區域線性函數，只需要方框濾波器，而且在強烈邊緣附近表現較好。Paris 等人 [7] 綜述了雙邊濾波器及其應用，例如為了色調映射把影像拆成基底層與細節層；Milanfar [9] 把這些濾波器統一看作資料自適應的加權平均；Gingichashvili 與 Lischinski [10] 則提出比較它們的共同方法。

僅線上版：導引濾波器：沿著導引影像平滑 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/notes/導引濾波器沿著導引影像平滑/*；雙邊濾波器：只和長得像的像素平均 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/notes/雙邊濾波器只和長得像的像素平均/*

僅線上版：*Bilateral Filter*：平均鄰近像素，但不跨越強邊緣 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/papers/bilateral-filter 平均鄰近像素但不跨越強邊緣/*；*Guided Filter*：擬合局部線性模型，取代逐對相似度計算 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/papers/guided-filter 擬合局部線性模型取代逐對相似度計算/*

深度學習改變了什麼。最有意思的發現是，本章有多少內容在學習式系統內部存活下來。許多成功的增強網路並不直接輸出像素，而是預測某個傳統運算子的參數，再套用它。

- **學習曲線。** Zero-DCE [13] 用一個小型 CNN 為每個像素預測提亮曲線，而且訓練時不需要參考影像：這就是 gamma 與對比拉伸的概念，只是曲線逐像素決定。
- **學習式保邊濾波器。** HDRNet [15] 在低解析度的雙邊網格（bilateral grid）中預測區域仿射色彩轉換，再於全解析度套用，把雙邊濾波器的保邊能力變成可學習的層。
- **低光增強。** Li 等人的綜述 [12] 依學習策略、網路結構、損失函數與訓練資料，回顧了深度學習的低光影像與影片增強。

僅線上版：現代觀點的詳細介紹 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/intensity-spatial-filtering/papers/現代觀點的詳細介紹/*

沒有改變的部分：中值濾波器仍是脈衝雜訊的首選，Sobel 卷積核仍出現在損失函數與邊緣圖中，CLAHE 仍是預設的前處理步驟，而 gamma 處理仍是混用線性與編碼像素值的學習式流程中常見的錯誤來源。深度模型也往往繼承了傳統方法的失敗模式：光暈、暗部雜訊放大與過度增強，仍是審查者會特別檢查的瑕疵。

重點整理

- 強度轉換 $s = T(r)$ 就是一張查找表，斜率就是局部對比增益。對數與 $\gamma < 1$ 的曲線擴張暗部，$\gamma > 1$ 擴張亮部，分段線性曲線則提供精確控制。
- 直方圖等化以縮放後的 CDF 作為 T，在連續情形下讓輸出 PDF 成為均勻分布，在離散情形下則近似平坦。匹配是把一次等化接上另一次等化的反函數；精確匹配則先用區域平均打破平手。
- 區域方法（AHE、CLAHE、區域平均與變異數規則）會依區域調整；CLAHE 藉由截斷每個區域直方圖來限制對比增益。
- 線性空間濾波就是與卷積核做相關；卷積會翻轉卷積核，並具有漂亮的代數性質。可分離卷積核每個像素只需 $m + n$ 次運算，而非 mn 次。
- 低通濾波器（方框、高斯）以平均去除雜訊，但會模糊邊緣；非線性的中值濾波器是對付脈衝雜訊的工具。
- 銳化就是加入高通細節：拉普拉斯（二階導數）、反銳化遮罩與高增強；梯度與 Sobel 運算子則用來衡量邊緣強度。
- 高通、帶通與帶拒卷積核都能由低通卷積核從脈衝 δ 相減得到；好的處理流程會安排步驟順序，不讓後面的步驟放大前面本該去除的東西。

練習

1. **替霧濛濛的照片設計曲線**。一張 8 位元影像的像素值全落在 90 到 160 之間。寫出把這個範圍對應到 $[0, 255]$ 的分段線性轉換、它的斜率，並說明值為 120 與 122 的兩個相鄰像素會怎樣。

   提示

   $90 \le r \le 160$ 時 $T(r) = 255\,(r - 90)/70$，範圍外則裁切到 $[0, 255]$。斜率為 $255/70 \approx 3.64$，所以相差 2 會變成約 7.3 個灰階。四捨五入後 $T(120) = 109$、$T(122) = 117$。

2. **手算等化**。一張 3 位元（$L = 8$）、共 16 個像素的小影像，灰階 0 到 7 的計數為 $n = [0, 8, 4, 2, 2, 0, 0, 0]$。計算等化查找表與輸出直方圖。為什麼輸出不是平坦的？

   提示

   CDF 為 $[0, 0.5, 0.75, 0.875, 1, 1, 1, 1]$，所以 $s_k = \text{round}(7 \cdot \text{CDF})$ 得到 $[0, 4, 5, 6, 7, 7, 7, 7]$（3.5 捨入為 4）。輸出直方圖為：灰階 4 有 8 個像素、5 有 4 個、6 有 2 個、7 有 2 個。灰階 1 的 8 個像素共用同一個輸入值，查找表無法拆開它們，只有精確直方圖規定化才能做到。

3. **順序重要嗎？**先套用 gamma 曲線再做方框（平均）濾波，和先做方框濾波再套用 gamma 曲線，結果一樣嗎？請證明，或用 2 個像素的平均舉出反例。

   提示

   不一樣。取兩個像素 0 與 1、$\gamma = 2$。先 gamma：$(0 + 1)/2 = 0.5$；先平均：$0.5^2 = 0.25$。非線性的點運算與線性濾波器不能交換順序。這也是為什麼模糊 gamma 編碼的影像，不等於模糊線性光。

4. **可不可以分離？**判斷下列卷積核是否可分離，若可以，寫出兩個向量：(a) 中心為 -4 的 3×3 拉普拉斯；(b) 全部為 1 的 3×3 卷積核；(c) $\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$。

   提示

   (a) 不行：它的秩為 2（但它是兩個可分離卷積核的和，每個軸各一個）。(b) 可以：$[1,1,1]^{\top}[1,1,1]$。(c) 可以：$[1,2,1]^{\top}[1,2,1]$，是高斯的小型二項式近似。可用 `np.linalg.matrix_rank` 驗證。

5. **邊緣上的中值與平均**。一維訊號 $[10,10,10,10,50,50,50,50]$ 是完美的階梯。分別做 3 點平均與 3 點中值（忽略兩端樣本）。接著把位置 1（從 0 起算）的樣本換成值為 255 的脈衝，再做一次。你得到什麼結論？

   提示

   在乾淨的階梯上，平均得到一段斜坡（$\ldots, 10, 23.3, 36.7, 50, \ldots$），中值則完整保留階梯。加入脈衝後，平均在位置 1 與 2 產生兩個約 92 的升高值；中值在這兩處輸出 10，完全去除脈衝。

6. **設計帶通卷積核**。用 $\sigma_1 = 1$ 與 $\sigma_2 = 3$ 的高斯，在 NumPy 中建立帶通與帶拒卷積核。驗證兩者的權重總和，用它們濾波 `skimage.data.brick()`，並描述各自保留了哪些結構。

   提示

   先把兩個卷積核補成相同大小（較小者補零），再計算 $K_1 - K_2$ 與 $\delta - (K_1 - K_2)$。總和應約為 0 與 1。帶通輸出保留灰縫線條與磚塊尺度的紋理，去除整體明暗與最細的顆粒；帶拒輸出保留明暗與細顆粒，但減弱了磚塊圖樣。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 3. publisher page
2. S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics, and Image Processing*, 1987. doi
3. K. Zuiderveld, "Contrast limited adaptive histogram equalization," in P. S. Heckbert (ed.), *Graphics Gems IV*, Morgan Kaufmann, 1994, pp. 474–485. entry
4. A. M. Reza, "Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement," *Journal of VLSI Signal Processing*, 2004. doi
5. D. Coltuc, P. Bolon and J.-M. Chassery, "Exact histogram specification," *IEEE Transactions on Image Processing*, 2006. doi
6. C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 1998. pdf

7. S. Paris, P. Kornprobst, J. Tumblin and F. Durand, “Bilateral filtering: Theory and applications,” *Foundations and Trends in Computer Graphics and Vision*, vol. 4, no. 1, pp. 1–73, 2009. doi
8. K. He, J. Sun and X. Tang, “Guided image filtering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(6):1397–1409, 2013 (first presented at ECCV 2010). doi
9. P. Milanfar, “A tour of modern image filtering,” *IEEE Signal Processing Magazine*, vol. 30, no. 1, pp. 106–128, 2013. IEEE Xplore
10. S. Gingichashvili and D. Lischinski, “Evaluation and comparison of edge-preserving filters,” arXiv:2012.13778, 2020. arXiv
11. Y. Qi, Z. Yang, W. Sun et al., “A comprehensive overview of image enhancement techniques,” *Archives of Computational Methods in Engineering*, 2021. doi
12. 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*, 2021 (arXiv:2104.10729). arXiv
13. C. Guo, C. Li, J. Guo, C. C. Loy, J. Hou, S. Kwong and R. Cong, “Zero-reference deep curve estimation for low-light image enhancement,” in *Proc. IEEE/CVF CVPR*, 2020. arXiv
14. 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*, 2020. arXiv
15. 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* (*SIGGRAPH*), 2017. arXiv
16. H. Wu, S. Zheng, J. Zhang and K. Huang, “Fast end-to-end trainable guided filter,” in *Proc. IEEE/CVF CVPR*, 2018. arXiv
17. 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
18. 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 (first presented at ECCV 2006). doi

IP 04・頻域濾波

傅立葉轉換如何把影像拆成一份「波的配方」，以及修改這份配方如何做到平滑、銳化與去除週期性雜訊。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/frequency-domain/

你將學到

- 傅立葉轉換（Fourier transform）在描述訊號的什麼，以及為什麼「頻率」是描述影像的好方法
- 取樣（sampling）如何運作、什麼是奈奎斯特取樣率（Nyquist rate），以及取樣太稀疏為什麼會產生混疊（aliasing）與摩爾紋（moiré）
- 一維與二維的離散傅立葉轉換（DFT），以及每天都會用到的性質：中心化、對稱性、頻譜與相位、卷積定理
- 頻域濾波的標準步驟，包括避免環繞誤差（wraparound error）所需的填補
- 低通、高通、同態、帶拒與陷波濾波器，以及「理想」濾波器為什麼會產生振鈴（ringing）
- 快速傅立葉轉換（FFT）如何讓這一切變得便宜，以及傅立葉的想法在現代深度學習中出現在哪裡

先看全貌

在第 3 章，我們讓一個小卷積核在像素上滑動來濾波。這一章換一個完全不同的角度看同一張影像。我們不再問「這個像素有多亮？」，而是問「這張影像裡含有多少各種不同的波？」第二種描述叫做影像的**頻譜**（spectrum），產生它的工具就是**傅立葉轉換**。

> **用最簡單的話說**
>
> 想像一個樂團在演奏。你的耳朵聽到的是混在一起的聲音，但樂手可以告訴你：「這裡有低沉的貝斯、中音的鋼琴和弦，還有高音的鈸。」傅立葉轉換對一張圖片做的是同一件事：它告訴你圖裡有多少「慢慢變化」的成分（大片平滑的區域，像貝斯），有多少「快速變化」的成分（銳利的邊緣和細紋理，像鈸）。知道配方之後，你可以把鈸調小聲讓圖變平滑，或把鈸調大聲讓圖變清晰，然後再把整首歌重新播放出來。

為什麼要這麼麻煩？有三個理由。第一，有些操作用「保留這些頻率、拿掉那些頻率」來描述，比設計一個卷積核容易得多。第二，根據卷積定理，大型卷積會變成簡單的乘法，而 FFT 讓它跑得很快。第三，頻率的觀點能解釋許多在像素觀點下很神祕的現象：為什麼縮小影像會冒出奇怪的花紋、為什麼截止得很陡的濾波器會產生漣漪，以及為什麼重複出現的條紋雜訊幾乎可以被完美移除。本章依照 Gonzalez 與 Woods 教科書的章節安排 [1]。

背景

白話版。任何合理的訊號，都可以由許多不同頻率的正弦波和餘弦波相加而成，每個波有自己的強度和起始位置。傅立葉轉換就是要找出這些強度和起始位置。

十九世紀初，傅立葉（Joseph Fourier）主張：週期函數可以寫成正弦與餘弦的加權總和；即使是非週期函數，只要曲線下的面積有限，也能寫成正弦與餘弦的積分。最驚人的是，這個過程不會遺失任何資訊：你可以轉到頻率描述，再一模一樣地轉回來。

對影像而言，這件事有兩層意義。**低頻**是橫跨影像緩慢變化的波，它承載整體亮度與大片平滑區域。**高頻**變化得很快，它承載邊緣、細紋理與雜訊。平滑影像就是削弱高頻；銳化影像就是加強高頻。本章要把這個說法講得精確。

傅立葉分析要到 1965 年快速傅立葉轉換出現之後 [4]，才真正成為實用的影像處理工具；在那之前，對一張影像做轉換實在太慢了。

預備概念

白話版。在轉換影像之前，我們需要五樣工具：複數（把一個波的強度和起始位置一起存起來）、傅立葉級數（波的總和）、脈衝（完美的「尖峰」）、傅立葉轉換本身，以及卷積。

複數

複數寫成 $C = R + jI$，其中 R 是實部，I 是虛部，$j = \sqrt{-1}$。它也可以寫成極座標形式

$$C = |C|\, e^{j\theta}, \qquad |C| = \sqrt{R^2 + I^2}, \qquad \theta = \operatorname{atan2}(I, R)$$

其中 $|C|$ 是大小（長度），θ 是角度。歐拉公式 $e^{j\theta} = \cos\theta + j\sin\theta$ 把兩種寫法連在一起。這就是傅立葉轉換使用複指數的原因：一個複數就能同時記錄一個波的**振幅**（amplitude）與**相位**（phase，也就是波從哪裡開始）。複數的共軛為 $C^* = R - jI$。

傅立葉級數

一個以週期 T 重複的函數 $f(t)$ 可以寫成

$$f(t) = \sum_{n=-\infty}^{\infty} c_n\, e^{j\frac{2\pi n}{T}t}, \qquad c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t)\, e^{-j\frac{2\pi n}{T}t}\, dt$$

其中 n 是整數，c_n 是在一個週期內完成 n 個循環的那個波的（複數）係數，t 是連續變數（時間或位置）。

脈衝與篩選性質

連續的**單位脈衝**（unit impulse）$\delta(t)$ 是一個理想化的尖峰：除了 $t = 0$ 以外處處為零，而總面積為 1，即 $\int_{-\infty}^{\infty} \delta(t)\, dt = 1$。它最有用的性質是**篩選性質**（sifting property）：

$$\int_{-\infty}^{\infty} f(t)\,\delta(t-t_0)\,dt = f(t_0)$$

其中 t_0 是任意位置。乘上一個平移過的脈衝再積分，就把 f 在 t_0 的值「挑」出來。離散脈衝 $\delta(x)$ 在 $x=0$ 時為 1、其他地方為 0，篩選性質變成 $\sum_x f(x)\,\delta(x-x_0) = f(x_0)$。

脈衝串（impulse train）$s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta T)$ 是一排間距為 ΔT 的脈衝，它就是取樣的數學模型。

連續函數的傅立葉轉換

$f(t)$ 的傅立葉轉換與反轉換為

$$F(\mu) = \int_{-\infty}^{\infty} f(t)\,e^{-j2\pi\mu t}\,dt, \qquad f(t) = \int_{-\infty}^{\infty} F(\mu)\,e^{j2\pi\mu t}\,d\mu$$

其中 μ 是連續頻率（每單位 t 的循環數），$F(\mu)$ 一般是複數。這兩個式子合稱**傅立葉轉換對**，記作 $f(t) \Leftrightarrow F(\mu)$。

用一個例子建立直覺。取一個高度為 A、只在 $|t| \le W/2$ 時不為零的方框函數，它的轉換是

$$F(\mu) = AW\,\frac{\sin(\pi\mu W)}{\pi\mu W} = AW\,\operatorname{sinc}(\mu W)$$

其中 $\operatorname{sinc}(m) = \sin(\pi m)/(\pi m)$。這裡有兩個教訓。第一，有銳利邊緣的函數，其轉換會向外擴散，帶著緩慢衰減的漣漪。第二，兩個域之間存在反比關係：方框越寬，sinc 越窄。在一個域裡窄，在另一個域裡就寬。等一下「理想」濾波器產生振鈴時，這兩個教訓會再次出現。

另外兩組轉換對值得記住。脈衝的轉換是常數：$\delta(t) \Leftrightarrow 1$。間距為 ΔT 的脈衝串，其轉換是另一個間距為 $1/\Delta T$ 的脈衝串：

$$s_{\Delta T}(t) \Leftrightarrow S(\mu) = \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty} \delta\!\left(\mu - \frac{n}{\Delta T}\right)$$

卷積

兩個連續函數的卷積為

$$(f \star h)(t) = \int_{-\infty}^{\infty} f(\tau)\,h(t-\tau)\,d\tau$$

其中 τ 是積分用的虛擬變數。**卷積定理**（convolution theorem）說

$$(f \star h)(t) \Leftrightarrow H(\mu)F(\mu), \qquad f(t)\,h(t) \Leftrightarrow (H \star F)(\mu)$$

一個域裡的卷積，就是另一個域裡的乘法。頻域濾波之所以存在，全靠這一個事實。

取樣與取樣函數的傅立葉轉換

白話版。相機並不會記錄連續的場景，它只在一個格點上量測。格點太疏的話，快速的細節會被誤認成緩慢的變化，你會看到其實不存在的花紋。只要取樣頻率超過最快頻率的兩倍，原則上就能把原訊號完整重建。

取樣就是相乘

把取樣模型化為 $f(t)$ 乘上脈衝串：

$$\tilde{f}(t) = f(t)\, s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} f(n\Delta T)\, \delta(t - n\Delta T)$$

其中 ΔT 是取樣間隔，$f(n\Delta T)$ 是取樣值。由卷積定理，相乘變成與 $S(\mu)$ 卷積，而與一串脈衝卷積，效果就是把頻譜複製很多份：

$$\tilde{F}(\mu) = \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\!\left(\mu - \frac{n}{\Delta T}\right)$$

所以，**取樣後函數的頻譜，是原頻譜以 $1/\Delta T$ 為間隔、無限重複的週期性複本**。

取樣定理與奈奎斯特取樣率

假設 f 是**帶限**（band-limited）的：當 $|\mu| > \mu_{\max}$ 時 $F(\mu) = 0$。只要複本之間的間距大於一份複本的寬度，它們就不會重疊：

$$\frac{1}{\Delta T} > 2\mu_{\max}$$

這就是**取樣定理**，因 Shannon 而在工程界廣為人知 [2]。$2\mu_{\max}$ 稱為**奈奎斯特取樣率**。取樣速度超過它，取樣值就包含 f 的全部資訊。

重建

如果複本沒有重疊，把 $\tilde{F}(\mu)$ 乘上一個高度為 ΔT、涵蓋 $[-\mu_{\max}, \mu_{\max}]$ 的方框 $H(\mu)$，就能分離出原本的頻譜。在時域，這相當於與 sinc 函數卷積，得到內插公式

$$f(t) = \sum_{n=-\infty}^{\infty} f(n\Delta T)\, \operatorname{sinc}\!\left[\frac{t - n\Delta T}{\Delta T}\right]$$

每個取樣點都貢獻一個以自己為中心的 sinc，加總後的曲線剛好通過所有取樣點。Unser 的回顧論文 [3] 說明了現代實務如何用樣條（spline）等緊湊的核心，來取代不切實際的無限長 sinc，並把它近似得很好。

混疊

若 $1/\Delta T \le 2\mu_{\max}$，相鄰的複本會重疊，尾巴互相疊加。之後不論用什麼濾波器都無法把它們分開：高頻偽裝成了低頻。這就是**混疊**。唯一的解法是預防：在取樣之前用低通濾波器（**抗混疊**濾波器，anti-aliasing filter）去除高頻。

一個小實驗就能看到這種偽裝。以每秒 10 個樣本取樣一個 9 Hz 的餘弦波，此時奈奎斯特上限是 5 Hz：

```
import numpy as np

fs = 10.0                       # samples per second
t = np.arange(0, 2, 1 / fs)     # 2 seconds of samples
x = np.cos(2 * np.pi * 9 * t)  # a 9 Hz tone: above fs/2 = 5 Hz
X = np.abs(np.fft.rfft(x))
freqs = np.fft.rfftfreq(len(x), d=1 / fs)
print("peak at", freqs[X.argmax()], "Hz")   # -> 1.0 Hz, not 9 Hz
```

在這些取樣時間點上，9 Hz 的音和 1 Hz 的音完全無法區分。電影裡馬車車輪看起來倒著轉，也是同一個效應。

一維離散傅立葉轉換

白話版。電腦手上只有有限個取樣值，所以它使用有限版本的傅立葉轉換。DFT 吃進 M 個數字，吐出 M 個數字，每個數字代表某一個特定的波有多少。

定義

對取樣值 $f(0), f(1), \dots, f(M-1)$：

$$F(u) = \sum_{x=0}^{M-1} f(x)\, e^{-j2\pi ux/M}, \qquad f(x) = \frac{1}{M} \sum_{u=0}^{M-1} F(u)\, e^{j2\pi ux/M}$$

其中 $x = 0, \dots, M-1$ 是取樣點的索引，$u = 0, \dots, M-1$ 是離散頻率的索引（頻率 u 代表在 M 個樣本內完成 u 個循環），係數 $1/M$ 放在反轉換上（也有慣例把它拆成兩邊各 $1/\sqrt{M}$）。

與取樣的關係

這個公式從哪裡來？取樣後函數的轉換 $\tilde{F}(\mu)$ 是連續的，並以 $1/\Delta T$ 為週期。在其中一個週期內等間隔地取 M 個點，$\mu = m/(M\Delta T)$，$m = 0, \dots, M-1$，得到的正好就是上面的 DFT。所以 DFT 就是**取樣資料之轉換、在一個週期內的取樣值**。DFT 相鄰頻率之間的間距為

$$\Delta u = \frac{1}{M\Delta T}$$

其中 $M\Delta T$ 是全部樣本所涵蓋的總長度。紀錄越長，頻率解析度越細。

實務上有兩個後果。因為在頻率上也取了樣，DFT 會把資料當作以 M 為週期重複。而且 $F(u)$ 與 $f(x)$ 都是週期性的：$F(u) = F(u+M)$、$f(x) = f(x+M)$。

這個定義其實就是一次矩陣與向量的乘法，很容易驗證：

```
import numpy as np

def dft(f):
    M = len(f)
    x = np.arange(M)
    u = x.reshape(-1, 1)
    W = np.exp(-2j * np.pi * u * x / M)   # M x M matrix of complex exponentials
    return W @ f

f = np.random.default_rng(0).standard_normal(64)
print(np.allclose(dft(f), np.fft.fft(f)))  # True
```

推廣到二變數函數

白話版。影像有兩個方向，所以波也有兩個頻率：一個橫向、一個縱向。二維的波看起來像一組平行條紋，兩個頻率決定了條紋有多密、朝哪個方向。

二維脈衝與連續轉換

二維脈衝 $\delta(t,z)$ 的體積為 1，篩選方式和一維相同：$\iint f(t,z)\,\delta(t-t_0,z-z_0)\,dt\,dz = f(t_0,z_0)$。二維連續轉換為

$$F(\mu,\nu) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(t,z)\,e^{-j2\pi(\mu t+\nu z)}\,dt\,dz$$

其中 t,z 是空間座標，μ,ν 是對應的頻率。

二維取樣與混疊

以間距 ΔT、ΔZ 在格點上取樣，會把頻譜複製到一個二維格子上。一張帶限影像（在 $|\mu| \le \mu_{\max}$、$|\nu| \le \nu_{\max}$ 之外為零）可以被完整復原的條件是

$$\frac{1}{\Delta T} > 2\mu_{\max}, \qquad \frac{1}{\Delta Z} > 2\nu_{\max}$$

影像經常違反這條規則。真實場景含有銳利邊緣，而銳利邊緣根本不是帶限的，所以每張數位影像多少都有混疊。當你用丟棄像素的方式**降取樣**（縮小）影像時，混疊就會變得明顯。圖 4.1 使用一張波帶片（zone plate），它的頻率隨著與中心的距離增加。每 4 個像素只留 1 個，外圈的細環就變成原圖中根本不存在的假圓圈。先模糊再取樣，就能移除那些無法保留的頻率，假圓圈也隨之消失。

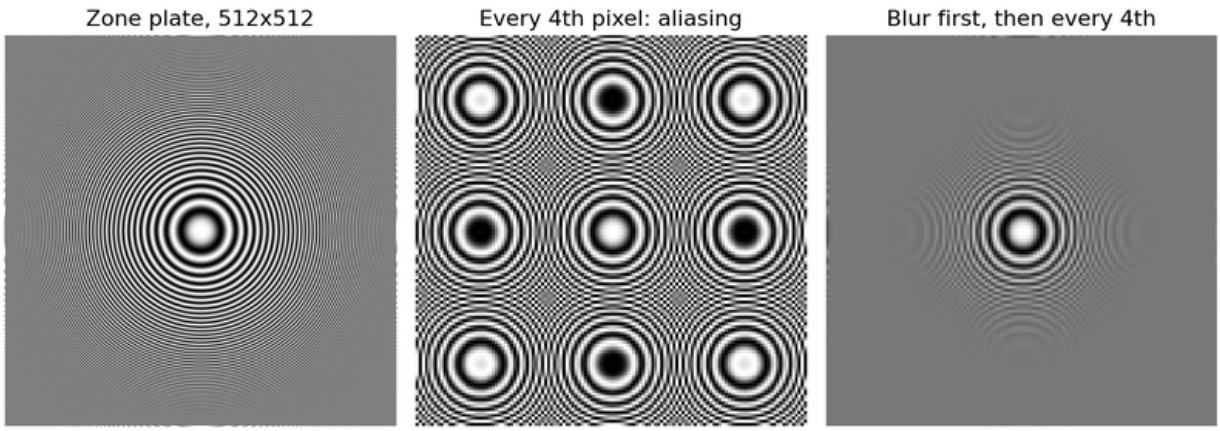


圖 *4.1* — 二維混疊。左：頻率向外遞增的波帶片。中：每 *4* 個像素取 *1* 個，產生假的圓環（混疊）。右：降取樣前先做高斯模糊，移除無法表示的頻率，原本細節所在之處變成一片均勻的灰。

摩爾紋

摩爾紋是日常生活中看得到的混疊：當細密的週期紋理（襯衫的織紋、紗窗、印刷照片的網點）被間距相近的感測器格點取樣時，兩種週期互相「拍」出又大又慢的波浪狀條帶。圖 4.1 中間那張就是這種摩爾紋。相機在感測器前面放一片光學抗混疊濾鏡來減輕它；軟體則在縮放之前先模糊。

二維 DFT

對一張 $M \times N$ 的影像 $f(x, y)$：

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}, \qquad f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v)\, e^{j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

其中 x, y 是像素座標（列、行），u, v 是離散頻率索引，$F(u,v)$ 是 $M \times N$ 的複數陣列。F 的每一個元素都描述一個覆蓋整張影像的二維波。

二維 DFT 的性質

白話版。頻譜的行為是可以預測的。移動物體不會改變頻譜的大小；旋轉物體會讓頻譜跟著旋轉；（移位後的）頻譜中心存的是平均亮度。最令人意外的是：我們在圖中認得出來的東西，大部分存在相位裡，而不是在大小裡。

空間間隔與頻率間隔

如果影像是以間距 ΔT、ΔZ 取樣的，DFT 的頻率間距為

$$\Delta u = \frac{1}{M\Delta T}, \qquad \Delta v = \frac{1}{N\Delta Z}$$

視野越大，頻率解析度越細；像素越小，DFT 能表示的最高頻率就越高。

平移與旋轉

平移影像等於把轉換乘上一個線性相位，反之亦然：

$$f(x - x_0, y - y_0) \Leftrightarrow F(u,v)\, e^{-j2\pi\left(\frac{ux_0}{M} + \frac{vy_0}{N}\right)}$$

其中 (x_0, y_0) 是平移量。因為 $|e^{-j\cdot}| = 1$，**大小 $|F|$ 不受平移影響**，只有相位改變。若使用極座標 $x = r\cos\theta$、$y = r\sin\theta$ 以及 $u = \omega\cos\varphi$、$v = \omega\sin\varphi$，把 f 旋轉 θ_0，F 也會旋轉同樣的角度：$f(r, \theta + \theta_0) \Leftrightarrow F(\omega, \varphi + \theta_0)$。

週期性與中心化

DFT 與其反轉換在兩個方向上都是週期性的：對任意整數 k_1, k_2，$F(u,v) = F(u + k_1 M, v + k_2 N)$。因此，原始的 DFT 陣列把零頻率放在左上角，四個四分之一週期在陣列中央相遇。不論是觀察頻譜還是設計濾波器，把零頻率移到中央都方便得多。利用平移性質，令 $x_0 = M/2$、$y_0 = N/2$：

$$f(x,y)\,(-1)^{x+y} \Leftrightarrow F(u - M/2,\ v - N/2)$$

此式在 M、N 為偶數時成立。在 NumPy 中，轉換之後呼叫 `np.fft.fftshift` 可以達到同樣的重新排列。

對稱性

對實數影像而言，轉換具有**共軛對稱性**：$F(-u,-v) = F^*(u,v)$。因此大小是偶函數，$|F(-u,-v)| = |F(u,v)|$，相位則是奇函數。頻譜中的每一個亮點，都有一個對中心鏡像的雙胞胎。設計陷波濾波器時會用到這一點。

傅立葉頻譜與相位角

令 $F(u,v) = R(u,v) + jI(u,v)$，則

$$|F(u,v)| = \sqrt{R^2 + I^2}, \qquad \phi(u,v) = \operatorname{atan2}(I(u,v), R(u,v)), \qquad P(u,v) = |F(u,v)|^2$$

分別是**傅立葉頻譜**（大小）、**相位角**與**功率譜**（power spectrum）。零頻率項稱為 **DC 成分**，$F(0,0) = MN\bar{f}$，其中 $\bar{f}$ 是平均灰階值。由於 $F(0,0)$ 通常比其他值大上數千倍，顯示頻譜時會用 $\log(1 + |F|)$。

```
import numpy as np
from skimage import data, img_as_float
```

```
f = img_as_float(data.camera())
F = np.fft.fft2(f)
Fc = np.fft.fftshift(F)                          # move (0,0) to the centre
spectrum = np.log1p(np.abs(Fc))                  # log(1+|F|) for display
phase = np.angle(Fc)                            # atan2(imaginary part, real part)
print(np.angle(np.array([1+1j, -1+1j]), deg=True))  # [ 45. 135.]

M, N = f.shape
y, x = np.mgrid[0:M, 0:N]
print(np.allclose(np.fft.fft2(f * (-1.0) ** (x + y)), Fc))  # True for even M, N
print(np.isclose(F[0, 0].real, f.sum()))                    # DC term = sum of pixels
```

圖 4.2 顯示簡單圖案與頻譜的對應。矩形的轉換是二維 sinc：一個明亮的十字，沿著矩形短邊方向的那一臂比較長（空間中窄、頻率中寬）。平移矩形，頻譜完全不變；旋轉矩形，頻譜跟著旋轉。餘弦光柵只含一個二維頻率，所以頻譜只有三個點：DC 項與一對對稱的點。高斯函數轉換後仍是高斯函數。

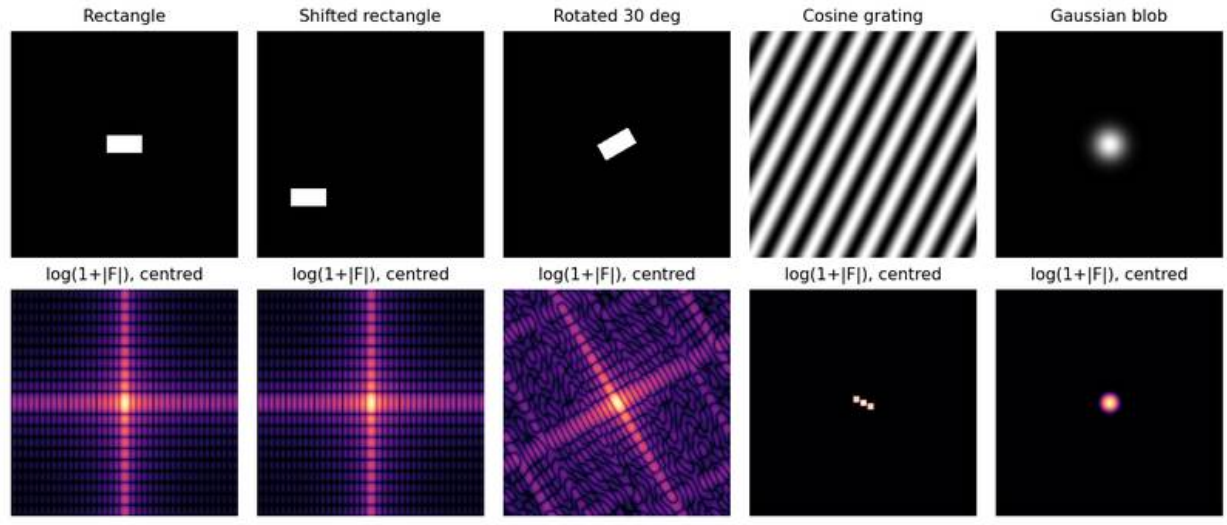


圖 4.2 — 簡單圖案（上）與中心化的對數頻譜（下）。平移不改變大小；旋轉使頻譜跟著旋轉。光柵頻譜的三個點已經放大，方便觀察。

雙參數函式 `atan2(I, R)` 保留兩個分量的正負號，才能辨識正確象限；`arctan(I/R)` 無法區分 $1 + j$ 與 $-1 - j$。大小為零時，數學上的相位沒有定義，即使軟體仍回傳某個約定數值。NumPy 的 `angle` 透過 `arctan2` 計算 [21]。

相位與大小的交換實驗

轉換中的哪一部分，承載了我們認得出來的內容？取兩張影像，保留其中一張的大小、另一張的相位，再做反轉換。Oppenheim 與 Lim 讓這個實驗廣為人知 [7]。

```
import numpy as np
```

```
from skimage import color, data, img_as_float

a = img_as_float(data.camera())
b = img_as_float(color.rgb2gray(data.astronaut()))
A, B = np.fft.fft2(a), np.fft.fft2(b)
mixed = np.real(np.fft.ifft2(np.abs(A) * np.exp(1j * np.angle(B))))
# 'mixed' looks like the astronaut, not the cameraman
```

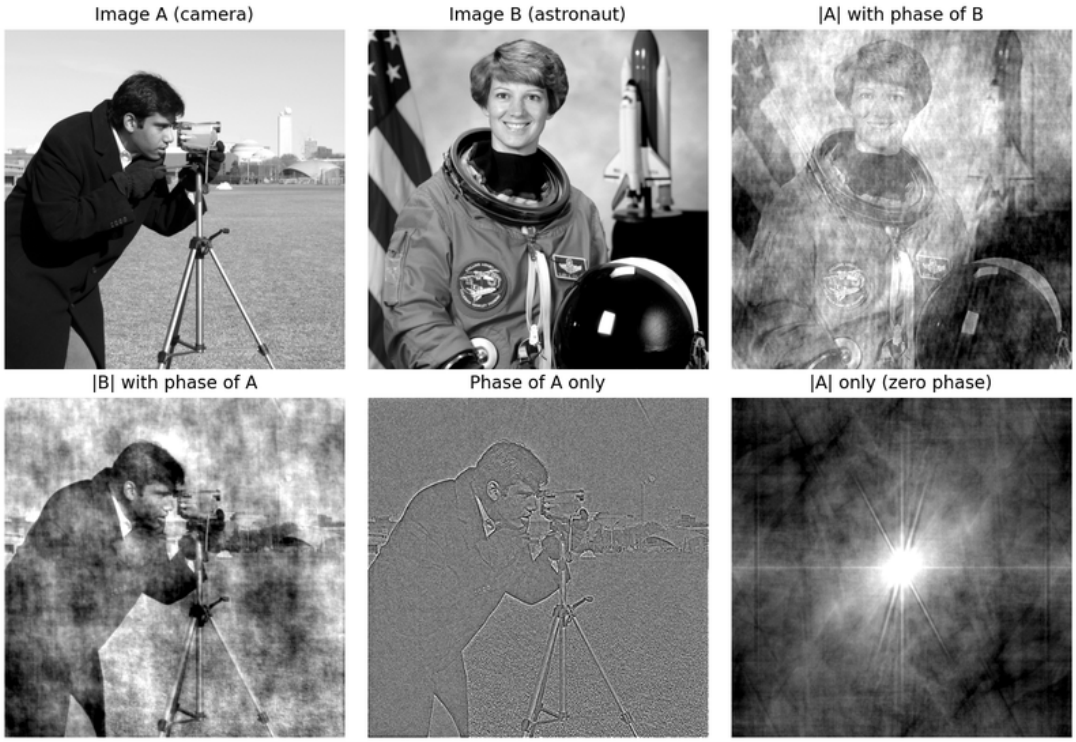


圖 *4.3* — 交換大小與相位。每張混合影像看起來都像「提供相位」的那一張。只有相位（大小設為 *1*）仍保留邊緣與構圖；只有大小（相位設為零）則是一團認不出來的光斑。

結果非常明確：**結構由相位承載**。大小告訴你每個頻率上有多少能量，而大多數自然照片的這部分都差不多。相位告訴你各個波在哪裡對齊，而邊緣恰好就是許多波對齊的地方。這也是為什麼後面的濾波器都設計成不動相位。

二維卷積定理

離散版本的卷積定理為

$$(f \star h)(x,y) \;\Leftrightarrow\; F(u,v)\,H(u,v), \qquad (f \star h)(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)\,h(x-m,\,y-n)$$

其中索引 $x-m$ 與 $y-n$ 要對 M、N **取餘數**。這是**循環卷積**（circular convolution）：DFT 假設影像無限重複，所以從右邊緣推出去的東西會從左邊冒出來。

環繞誤差與填補

循環卷積並不是我們要的，它會把影像左右邊界的內容混在一起。這就是**環繞誤差**。解法是把兩個陣列都補零，讓週期之間的距離大到複本永遠不會碰在一起。若 f 大小為 $A \times B$、h 大小為 $C \times D$，則填補後的大小 $P \times Q$ 應滿足

$$P \geq A + C - 1, \qquad Q \geq B + D - 1$$

如果濾波器直接在頻域中指定、而且和影像一樣大，常見的選擇就是 $P = 2M$、$Q = 2N$。

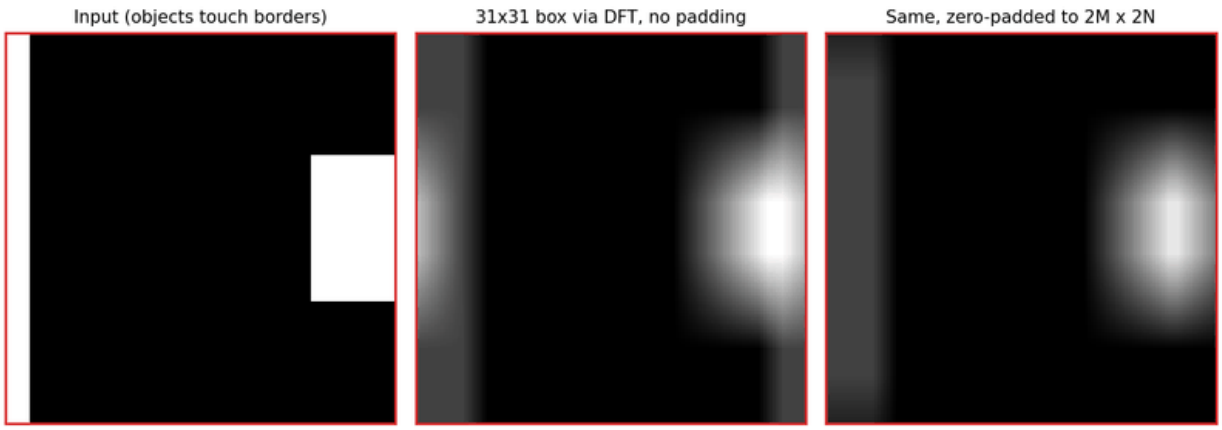


圖 *4.4* — 環繞誤差。不填補時（中），模糊讓右側方塊的光漏到左邊界，左側長條的光也漏到右邊界。補零到兩倍大小（右）才得到正確的線性卷積。

頻域濾波基礎

白話版。先轉換影像，再把每個頻率乘上一個代表「保留」或「移除」的數字，最後轉換回來。這一整份數字清單就是濾波器。

濾波步驟

給定 $M \times N$ 的影像 $f(x, y)$ 與濾波器轉移函數（transfer function）$H(u, v)$：

1. 選擇填補大小 $P = 2M$、$Q = 2N$。
2. 在影像後方補零（或補鏡像像素），得到 $P \times Q$ 的填補影像 $f_p(x, y)$。
3. 把 f_p 乘上 $(-1)^{x+y}$，使其轉換中心化。
4. 計算 DFT，得到 $F(u, v)$。
5. 建立大小為 $P \times Q$、中心位於 $(P/2, Q/2)$ 的實數對稱濾波器 $H(u, v)$，並計算 $G(u, v) = H(u, v)\, F(u, v)$。
6. 計算 $g_p(x, y) = \mathrm{Re}\{\mathcal{F}^{-1}[G(u, v)]\}\,(-1)^{x+y}$。
7. 裁切左上角 $M \times N$ 的區域，得到 $g(x, y)$。

第 6 步取實部，只是丟掉浮點數捨入誤差造成的微小虛部；對實數影像與實數對稱濾波器而言，精確的結果本來就是實數。

```
import numpy as np

def filter_freq(f, H_of_D):
    """Filter image f with a radially symmetric, zero-phase transfer function.
    H_of_D maps a distance array D (from the centre of the P x Q grid) to H."""
    M, N = f.shape
    P, Q = 2 * M, 2 * N                                  # 1. padding size
    fp = np.zeros((P, Q)); fp[:M, :N] = f                # 2. zero-pad
    y, x = np.mgrid[0:P, 0:Q]
    fp = fp * (-1.0) ** (x + y)                          # 3. centre the transform
    F = np.fft.fft2(fp)                                  # 4. DFT
    D = np.hypot(y - P / 2, x - Q / 2)
    G = H_of_D(D) * F                                    # 5. multiply
    gp = np.real(np.fft.ifft2(G)) * (-1.0) ** (x + y)    # 6. inverse, undo centring
    return gp[:M, :N]                                    # 7. crop
```

補零之後再模糊，右側與下側邊緣會出現一條變暗的帶狀區域，因為濾波器把黑色像素也平均進來了。改用鏡像填補（`np.pad(f, ..., mode="reflect")`）可以避免這個問題，本章的繪圖程式就是這樣做的。

零相位濾波器

對這裡使用的實數對稱轉移函數，只有**非負**增益才會在增益為正的頻率上保留相位。負增益會增加 π 的相位；增益為零則移除該頻率，輸出相位沒有定義。因此下文的非負低通與高通遮罩是零相位濾波器，但後面的負值 Laplacian 轉移函數會反轉符號。僅有實數與對稱性，不能保證零相位。設計不造成空間位移的濾波器，有助於讓特徵保持與輸入對齊。

空間濾波器與頻域濾波器的對應

由卷積定理，每個頻域濾波器 $H(u,v)$ 都對應一個空間卷積核 $h(x,y) = \mathcal{F}^{-1}\{H(u,v)\}$，每個空間卷積核也都有它的轉移函數。高斯是最乾淨的例子。在一維中，

$$H(u) = A\,e^{-u^2/2\sigma^2} \quad \Leftrightarrow \quad h(x) = \sqrt{2\pi}\,\sigma A\,e^{-2\pi^2\sigma^2 x^2}$$

其中 σ 是高斯在頻域中的寬度。頻域中寬的高斯，對應到空間中窄的卷積核，反之亦然。因此，一個常見的實務策略是：在頻域中設計（「讓 40 個循環以下通過」很容易描述），然後實作一個近似 h 的小型空間卷積核。

什麼時候走頻域比較快？用 $k \times k$ 的卷積核做空間卷積，每個像素大約要 k^2 次乘法；走 FFT 的話，每個像素的成本只隨 $\log(MN)$ 增加。小卷積核（3×3、5×5）走空間域划算；寬達數十個像素的卷積核，走 FFT 划算。

用低通頻域濾波器平滑影像

白話版。低通濾波器保留慢的波、移除快的波，所以影像會變模糊。濾波器從「保留」降到「移除」的方式非常重要：降得太突然，就會產生漣漪。

令 $D(u,v)$ 為與 $P \times Q$ 頻率矩形中心的距離，

$$D(u,v) = \sqrt{(u - P/2)^2 + (v - Q/2)^2}$$

並令 $D_0 > 0$ 為**截止頻率**（cutoff frequency）。

理想低通濾波器（ILPF）

$$H(u,v) = \begin{cases} 1 & D(u,v) \le D_0 \\ 0 & D(u,v) > D_0 \end{cases}$$

它讓半徑 D_0 的圓內所有頻率通過，圓外全部擋掉。選擇 D_0 的一個好方法，是看它保留了總功率的多少比例：$\alpha = 100 \sum_{D \le D_0} P(u,v) / \sum_{u,v} P(u,v)$。

巴特沃斯低通濾波器（BLPF）

$$H(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}}$$

其中 n 是**階數**（order）。在 $D = D_0$ 時濾波器值恰為 0.5。n 小時衰減平緩；$n \to \infty$ 時 BLPF 趨近理想濾波器。

高斯低通濾波器（GLPF）

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

其中 D_0 扮演 σ 的角色。在 $D = D_0$ 時濾波器值為 $e^{-1/2} \approx 0.607$。

振鈴

圖 4.5 以相同的截止頻率比較三者。理想濾波器產生明顯的**振鈴**：邊緣的淡淡分身一圈圈向外重複，就像往池塘丟石頭激起的漣漪。原因正是前面提過的方框與 sinc 轉換對。頻域中的陡峭截止，對應到一個中央主瓣、外圍環繞正負交替旁瓣的空間卷積核。邊緣與這種卷積核卷積之後，旁邊就會畫出一串淡淡的明暗回音。

高斯則是另一個極端：它的空間卷積核也是高斯，永遠為正，所以絕不會振鈴。巴特沃斯介於兩者之間：1 階完全沒有振鈴，2 階有幾乎看不見的負旁瓣，階數越高振鈴越明顯。2 階是在陡峭截止與乾淨結果之間常見的折衷。

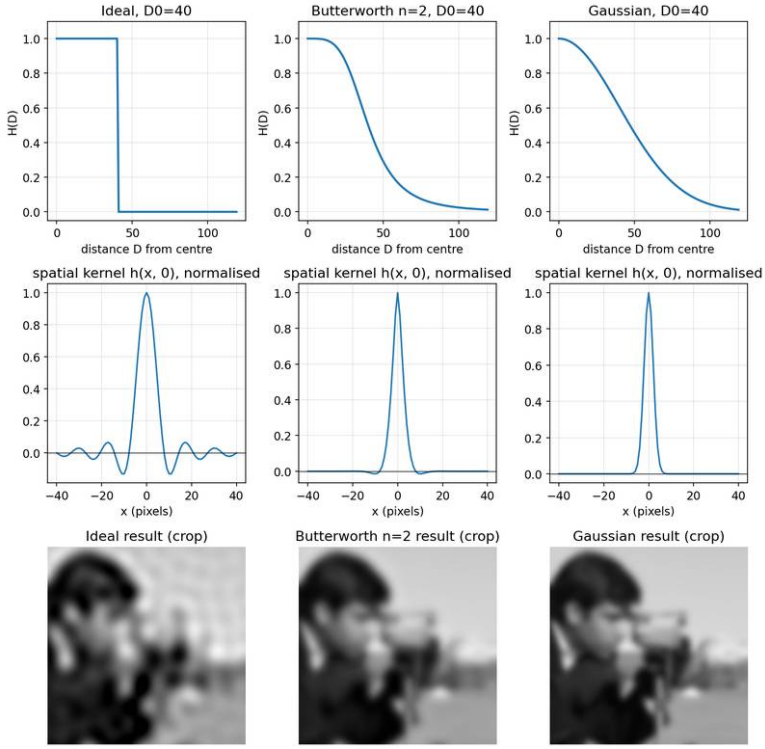


圖 *4.5* — 截止頻率皆為 $D0 = 40$ 的理想、巴特沃斯（$n = 2$）與高斯低通濾波器。上：徑向剖面 $H(D)$。中：對應的空間卷積核，只有理想濾波器有明顯的負旁瓣。下：理想濾波器的結果在邊緣周圍出現漣漪（振鈴），高斯則沒有。

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

```
D0 = 40
ideal  = lambda D: (D <= D0).astype(float)
butter = lambda D, n=2: 1 / (1 + (D / D0) ** (2 * n))
gauss  = lambda D: np.exp(-D**2 / (2 * D0**2))
smooth = filter_freq(f, gauss)
```

低通濾波有很多不起眼但實用的用途：把低解析度掃描中斷掉的字跡筆畫連起來、柔化人像的皮膚紋理，以及壓抑遙測影像中的掃描線紋路。

用高通濾波器銳化影像

白話版。高通濾波器正好相反：它丟掉慢的波、保留快的波，所以只剩下邊緣與紋理。把其中一部分加回原圖，影像看起來就更清晰。

理想、巴特沃斯與高斯高通濾波器

每個高通濾波器都可以由低通濾波器得到：

$$H_{\text{HP}}(u,v) = 1 - H_{\text{LP}}(u,v)$$

所以高斯高通為 $1 - e^{-D^2/2D_0^2}$，巴特沃斯高通為 $1/(1+[D_0/D]^{2n})$，理想高通則是圓內為 0、圓外為 1。振鈴的道理一樣：理想的會振鈴，高斯的不會。由於高通濾波器令 $H(0,0)=0$，輸出的平均值為零，所以平坦區域在顯示縮放後會變成灰色（零）。

頻域中的拉普拉斯運算子

第 3 章的拉普拉斯運算子（Laplacian）$\nabla^2 f = \partial^2 f/\partial x^2 + \partial^2 f/\partial y^2$ 也有轉移函數。微分等於把轉換乘上 $j2\pi u$，在兩個方向各微分兩次，得到

$$H(u,v) = -4\pi^2\left[(u-P/2)^2 + (v-Q/2)^2\right] = -4\pi^2 D^2(u,v)$$

此處頻率以「每張影像的循環數」為單位。銳化後的影像為 $g(x,y) = f(x,y) - \nabla^2 f(x,y)$；用減號是因為 H 的中心為負。實務上拉普拉斯的輸出比 f 大得多，相減前必須先把兩者縮放到可比較的範圍（例如 f 在 $[0,1]$，$\nabla^2 f$ 除以其最大絕對值）。

反銳化遮罩、高提升與高頻強調

反銳化遮罩（unsharp masking）先減去模糊版本得到細節「遮罩」，再把它加回去：

$$g_{\text{mask}}(x,y) = f(x,y) - f_{\text{LP}}(x,y), \qquad g(x,y) = f(x,y) + k\, g_{\text{mask}}(x,y)$$

其中 f_{LP} 是低通濾波後的 f，$k \geq 0$ 是權重（$k=1$：反銳化遮罩；$k>1$：高提升，high-boost）。在頻域中，整個操作只是一個轉移函數 $H = 1 + k\,H_{\text{HP}}$。把它一般化，就得到**高頻強調**（high-frequency emphasis）：

$$H_{\text{HFE}}(u,v) = k_1 + k_2\, H_{\text{HP}}(u,v)$$

其中 $k_1 \geq 0$ 控制低頻（以及 DC 項）留下多少，$k_2 \geq 0$ 控制高頻被放大多少。選 $k_1 < 1$ 也會降低整體對比，所以高頻強調之後常接著做直方圖等化（histogram equalization）。

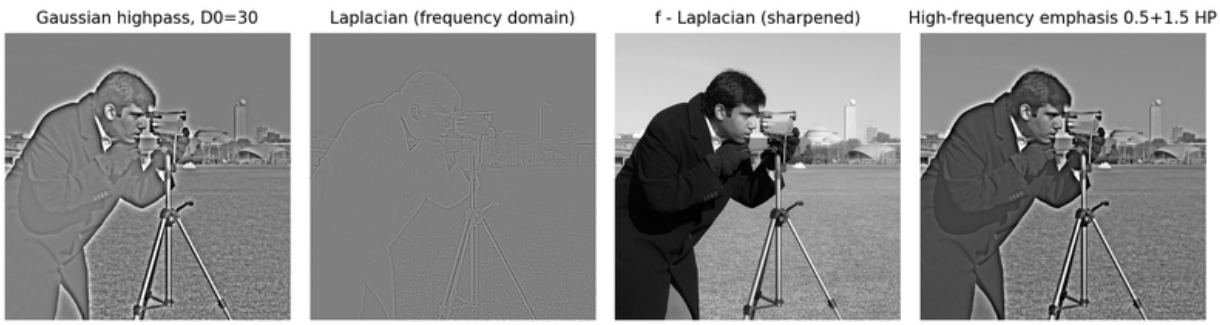


圖 *4.6* — 頻域銳化。由左至右：高斯高通（$D0 = 30$）、以 $H = -4\pi^2 D^2$ 計算的拉普拉斯、原圖減去縮放後的拉普拉斯，以及 $k1 = 0.5$、$k2 = 1.5$ 的高頻強調。

同態濾波

一個簡單的成像模型說，每個像素是**照度**（illumination）$i(x,y)$（照到場景上的光有多少）與**反射率**（reflectance）$r(x,y)$（表面反射多少光）的乘積：$f = i \cdot r$。照度在場景中變化緩慢；反射率則在物體邊界處劇烈變化。我們想壓低照度、加強反射率，但乘積的傅立葉轉換並不等於轉換的乘積。

Oppenheim、Schafer 與 Stockham 提出的技巧 [8] 是先取對數，把乘積變成和：

$$z(x,y) = \ln f(x,y) = \ln i(x,y) + \ln r(x,y)$$

接著用一個對低頻與高頻有不同處理的轉移函數對 z 濾波，最後取指數：

$$s = \mathcal{F}^{-1}\{H\,Z\}, \qquad g(x,y) = e^{s(x,y)}, \qquad H(u,v) = (\gamma_H - \gamma_L)\left[1 - e^{-c\,D^2(u,v)/D_0^2}\right] + \gamma_L$$

其中 $\gamma_L < 1$ 削弱低頻（照度），$\gamma_H > 1$ 放大高頻（反射率），c 控制過渡的陡峭程度，D_0 控制過渡的位置。結果同時壓縮了動態範圍、又增強了對比。

```
def homomorphic(f, gL=0.4, gH=1.6, c=1.0, D0=30, eps=1e-3):
    z = np.log(f + eps)                       # eps avoids log(0)
    H = lambda D: (gH - gL) * (1 - np.exp(-c * D**2 / D0**2)) + gL
    return np.exp(filter_freq(z, H)) - eps
```

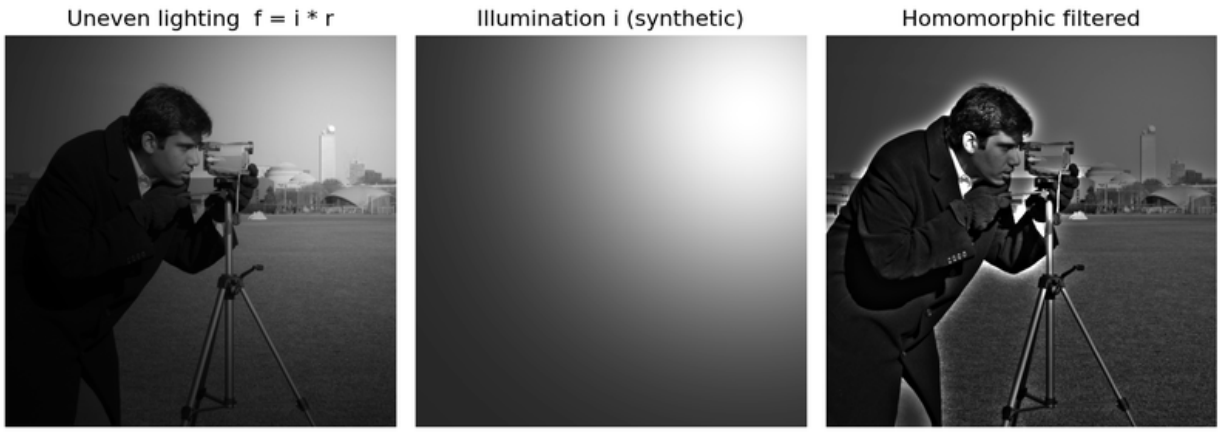


圖 *4.7* — 同態濾波。合成的照度場讓影像左側變暗（左、中）。以 $\gamma L = 0.4$、$\gamma H = 1.6$ 在對數域濾波後，照明變得均勻，暗部細節也被提升（右），代價是強邊緣周圍出現輕微光暈。

選擇性濾波

白話版。有時候問題不在於「細節太多」或「太少」，而是某個特定、重複出現的花紋，例如電子干擾造成的條紋。在頻譜中，這種花紋只是幾個亮點。選擇性濾波器只移除那幾個點，其他什麼都不動。

帶拒與帶通濾波器

帶拒（bandreject）濾波器移除以半徑 C_0 為中心、寬度為 W 的一圈頻率。高斯版本為

$$H_{\mathrm{BR}}(u,v) = 1 - \exp\left[-\left(\frac{D^2(u,v) - C_0^2}{D(u,v)\,W}\right)^2\right]$$

n 階巴特沃斯版本為

$$H_{\mathrm{BR}}(u,v) = \frac{1}{1 + \left[\dfrac{D(u,v)\,W}{D^2(u,v) - C_0^2}\right]^{2n}}$$

其中 C_0 是環的中心半徑，W 是環的寬度。**帶通**(bandpass)濾波器只保留那一圈：$H_{\mathrm{BP}} = 1 - H_{\mathrm{BR}}$。環狀濾波器適合在所有方向上都出現在固定距離的雜訊，但這在實務中並不常見。

陷波濾波器

陷波濾波器（notch filter）只作用在選定頻率周圍的小鄰域。由於實數影像的頻譜共軛對稱，每個陷波都必須搭配它對中心的鏡像。具有 Q 對陷波的陷波拒斥濾波器，是由多個中心移到亮點位置的高通濾波器相乘而成：

$$H_{\mathrm{NR}}(u,v) = \prod_{k=1}^{Q} H_k(u,v)\,H_{-k}(u,v)$$

其中 H_k 是中心位於 (u_k, v_k) 的高通濾波器（理想、巴特沃斯或高斯），H_{-k} 是中心位於 $(-u_k, -v_k)$ 的同一個濾波器，兩者都相對於頻率矩形的中心量測。距離為

$$D_k(u,v) = \sqrt{(u - P/2 - u_k)^2 + (v - Q/2 - v_k)^2}, \qquad D_{-k}(u,v) = \sqrt{(u - P/2 + u_k)^2 + (v - Q/2 + v_k)^2}$$

陷波通過濾波器為 $1 - H_{\mathrm{NR}}$。套用陷波通過濾波器再反轉換，可以單獨取出干擾花紋本身，用來確認你移除的是雜訊而不是影像內容。

```
import numpy as np
from skimage import data, img_as_float

f = img_as_float(data.camera())
M, N = f.shape
y, x = np.mgrid[0:M, 0:N]
g = f + 0.2 * np.cos(2 * np.pi * (40 * x / N + 25 * y / M))   # add a ripple
```

```
G = np.fft.fftshift(np.fft.fft2(g))
u, v = y - M // 2, x - N // 2              # centred frequency coordinates
H = np.ones((M, N))
for uk, vk in [(25, 40)]:                   # spike location (row, column offset)
    for s in (+1, -1):                      # the conjugate-symmetric partner
        Dk = np.hypot(u - s * uk, v - s * vk)
        H *= 1 - np.exp(-Dk**2 / (2 * 4.0**2))   # Gaussian notch reject
out = np.real(np.fft.ifft2(np.fft.ifftshift(G * H)))
```

這裡刻意不做填補。干擾在整張影像上是週期性的，填補反而會把它的亮點抹散到許多頻率樣本上，降低陷波的效果。

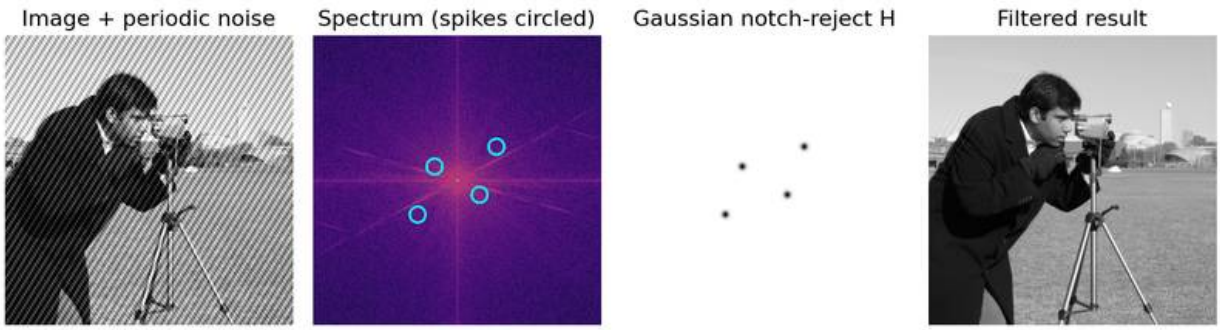


圖 *4.8* — 以陷波濾波去除合成的週期性雜訊。兩組正弦波紋產生四個亮點（兩對共軛點，已圈出）。高斯陷波拒斥濾波器在每個亮點放一個暗點把它們移除，干擾隨之消失。

拖動滑桿，直接比較受干擾與濾波後的影像：



含週期性雜訊



陷波濾波後

帶有斜向條紋干擾的攝影師影像與陷波濾波結果的比較

實作

白話版。二維 DFT 可以拆成許多一維 DFT 來算；反 DFT 可以用正向轉換的程式來算；而 FFT 演算法讓每一次一維轉換快上非常多。

可分離性

二維的核 $e^{-j2\pi(ux/M+vy/N)}$ 可以分解成 $e^{-j2\pi ux/M}\,e^{-j2\pi vy/N}$，所以

$$F(u,v)=\sum_{x=0}^{M-1} e^{-j2\pi ux/M}\underbrace{\sum_{y=0}^{N-1} f(x,y)\,e^{-j2\pi vy/N}}_{F(x,v)}$$

其中 $F(x,v)$ 是第 x 列的一維 DFT。先對每一列做一維 DFT，再對結果的每一行做一維 DFT。只要有一維的 FFT 函式，就能組出二維的。

反 DFT 與正向 DFT 的形式相同，只差指數的正負號與 $1/MN$ 因子，所以一個正向 FFT 函式也能用來計算反轉換。

僅線上版：用正向轉換計算反 *DFT* — *https://blog.shyandram.dev/zh-tw/learn/image-processing/frequency-domain/notes/用正向轉換計算反-dft/*

快速傅立葉轉換

照定義計算一維 DFT，大約需要 M^2 次複數乘法與加法：M 個輸出，每個都是 M 項的和。Cooley 與 Tukey 的 **FFT** [4] 在 M 為 2 的冪次時，把成本降到大約 $M\log_2 M$。核心想法是**分而治之**：把總和拆成偶數索引與奇數索引兩半，

$$F(u)=F_{\text{even}}(u)+W_M^u\,F_{\text{odd}}(u),\qquad F(u+M/2)=F_{\text{even}}(u)-W_M^u\,F_{\text{odd}}(u)$$

其中 $W_M=e^{-j2\pi/M}$，F_{even}、F_{odd} 是長度 $M/2$ 的 DFT。每個半長度的 DFT 再繼續拆，直到長度為 1。總共有 $\log_2 M$ 層，每層的成本約為 M 次運算。

兩者的成本比 $M^2/(M\log_2 M)=M/\log_2 M$ 成長得很快。$M=1024$ 時約為 100；對 1024×1024 的影像，直接做二維轉換（成本 $(MN)^2$）與用 FFT（成本 $MN\log_2 MN$）相比，比值約為 50,000。在筆電上就能明顯看出差異：

```
import time
x = np.random.default_rng(2).standard_normal(4096)
t0 = time.perf_counter(); dft(x); t1 = time.perf_counter(); np.fft.fft(x); t2 =
 ↪  time.perf_counter()
print(f"direct {t1 - t0:.3f}s  fft {t2 - t1:.5f}s")   # e.g. ~0.7 s vs ~0.2 ms
```

實務上請使用 FFTW [6] 這類函式庫提供的 FFT，它能處理任意長度。質因數都很小（2、3、5、7）的長度很快，大質數則較慢；`scipy.fft.next_fast_len` 會傳回一個合適的填補大小。

僅線上版：*FFT 的歷史與函式庫 — https://blog.shyandram.dev/zh-tw/learn/image-processing/frequency-domain/notes/fft-的歷史與函式庫/*

現代觀點

傅立葉轉換比本系列中任何其他工具都古老，卻老得特別好。深度學習並沒有取代它；相反地，傅立葉的想法以三種角色回歸：作為計算卷積的方法、作為建構具有全域視野之網路層的方法，以及作為理解網路學到什麼、在哪裡失敗的透鏡。

值得一讀的綜述與回顧

- **Unser，〈Sampling — 50 years after Shannon〉[3]** 回顧取樣理論從 Shannon 的帶限模型到以樣條為基礎的表述。重點：實際系統以緊湊的核心搭配相應的前置濾波器，取代理想的 sinc。
- **Xu、Zhang 與 Luo [13]** 綜述了**頻率原則**（frequency principle）：以梯度下降訓練的網路，傾向先擬合低頻、之後才擬合高頻，有點像一個慢慢打開的低通濾波器。
- **Kovachki 等人 [15]** 深入介紹神經算子（neural operator），它學習的是函數之間的映射。用頻率響應來參數化一層網路，可以讓它在很大程度上不受網格解析度影響。

網路內部的傅立葉概念

- **快速卷積**。依據卷積定理，卷積可以算成 FFT 之間的乘積；Mathieu、Henaff 與 LeCun [9] 用這一點加速 CNN 的訓練。如今的 3×3 小卷積核通常直接做空間卷積較快，所以 FFT 卷積主要用於大型卷積核。
- **全域濾波層**。頻域中的乘積會影響每一個像素，所以單一個傅立葉層就擁有整張影像的感受野（receptive field）。Global Filter Networks [11] 用本章的步驟（FFT、乘上可學習的濾波器、反 FFT）取代自注意力；Fourier Neural Operator [14] 則在低頻上學習濾波器，訓練好的模型可以在比訓練時更細的網格上使用。
- **頻譜偏差**。深度 ReLU 網路先學會低頻，高頻學得較晚、也較不穩健 [12]。這解釋了為什麼以座標為輸入的網路，常常需要先把輸入座標映射成正弦形式的傅立葉特徵，才能表現細緻的紋理 [20]。
- **混疊**。步幅卷積與池化在沒有低通前置濾波的情況下降取樣，因此會混疊，跟圖 4.1 一樣。Zhang [17] 指出，在每次降取樣前插入一個小模糊，能讓 CNN 對一個像素的平移更穩定，在他的實驗中也提升了分類準確率；無混疊（alias-free）的 GAN 生成器 [18] 則在每個重取樣步驟周圍套用精心設計的低通濾波器。
- **頻譜指紋**。生成器的上取樣會在頻譜中留下週期性的假影，很像圖 4.8 中的亮點；Frank 等人 [19] 利用它們建立了簡單又準確的 GAN 生成影像偵測器。

僅線上版：深度學習中的傅立葉概念詳解 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/frequency-domain/papers/深度學習中的傅立葉概念詳解/*

沒有改變的部分

對於已經很清楚的問題，深度學習並沒有取代傳統的頻域濾波。移除已知的週期性干擾、縮放前的抗混疊、快速的大卷積核模糊，以及同態照明校正，至今仍然用本章這些明確的濾波器來做，因為它們精確、快速、而且不需要訓練資料。改變的是，這些同樣的想法如今被內建到網路架構中，也被用來解釋網路的行為。

重點整理

- 傅立葉轉換把訊號改寫成波的總和。低頻承載平滑結構；高頻承載邊緣、紋理與雜訊。
- 取樣會把頻譜週期性地複製。取樣頻率要超過最高頻率的兩倍（奈奎斯特取樣率），或先模糊，否則會出現無法復原的混疊與摩爾紋。
- DFT 把影像當成週期性的，因此得到循環卷積；要填補到至少 $A + C - 1$（通常是 $2M \times 2N$）以避免環繞誤差。
- 平移只改變相位；旋轉使頻譜旋轉；實數影像的頻譜共軛對稱；而大部分可辨識的結構存在相位中。
- 濾波就是「轉換、乘上實數對稱的 H、反轉換、裁切」。陡峭截止會振鈴；高斯不會；巴特沃斯介於兩者之間。
- 高通、拉普拉斯、高頻強調與同態濾波器用來銳化或平衡照明；陷波濾波器能精準移除週期性干擾。
- FFT 把每次一維轉換的成本從 $O(M^2)$ 降到 $O(M \log M)$，可分離性則讓二維轉換由一維轉換組成。
- 在深度學習中，傅立葉的想法支撐了快速卷積、全域濾波層與神經算子，也解釋了頻譜偏差、CNN 中的混疊與 GAN 的假影。

練習

1. 一個一維訊號含有每公尺 3、7、12 個循環的成分。你每 0.1 公尺取樣一次。哪些成分能正確保留？每個混疊成分會出現在什麼頻率？

   提示

   取樣率是每公尺 10 個樣本，所以奈奎斯特上限是每公尺 5 個循環。3 個循環的成分沒問題。其他成分要減去 10 的倍數，摺回 $[-5, 5]$：7 會變成 $7 - 10 = -3$，也就是每公尺 3 個循環；12 會變成 $12 - 10 = 2$ 個循環。這時 7 個循環的成分和真正 3 個循環的成分已無法區分。

2. 證明：若 $f(x, y)$ 是實數且對稱，即 $f(x, y) = f(-x, -y)$（索引對 M、N 取餘數），則它的 DFT 是實數。用 NumPy 建一個對稱陣列加以驗證。

   提示

   對任何實數 f，共軛對稱給出 $F(-u, -v) = F^*(u, v)$。在定義中代入 $x \to -x$、$y \to -y$，由 f 的對稱性得到 $F(-u, -v) = F(u, v)$。兩者合起來 $F = F^*$，所以 F 是實數。驗證時，對隨機陣列 `a` 建立 `g = a + np.roll(a[::-1, ::-1], 1, axis=(0, 1))`，確認 `np.abs(np.fft.fft2(g).imag).max()` 趨近於零。

3. 使用 `filter_freq` 函式，以 $D_0 = 30$、階數 1、2、5、20 的巴特沃斯低通濾波器處理攝影師影像。從第幾階開始能明顯看到振鈴？畫出每個濾波器空間卷積核的中央列，並與你看到的現象對照。

   提示

   把中心化的濾波器做反轉換，再移回中央，就得到卷積核。當卷積核出現明顯的負旁瓣時就會振鈴。1 階完全沒有，2 階只有很小的旁瓣，到了 5 至 20 階，旁瓣已接近理想濾波器。

4. 你要用 FFT 以 51×51 的空間卷積核濾波一張 300×400 的影像。避免環繞誤差的最小填補大小是多少？你會選這個大小，還是稍大一點的？為什麼？

   提示

   最小值是 $(300 + 51 - 1) \times (400 + 51 - 1) = 350 \times 450$。兩者都可以，但只含小質因數的長度在 FFT 函式庫中較快。$350 = 2 \cdot 5^2 \cdot 7$、$450 = 2 \cdot 3^2 \cdot 5^2$ 本身就已符合，所以是好選擇；`scipy.fft.next_fast_len` 可以確認這一點。

5. 拍攝（或合成）一張含有兩種不同週期干擾的影像。設計一個只移除其中一種的陷波拒斥濾波器，再用對應的陷波通過濾波器顯示被你移除的花紋。陷波的半徑該怎麼選？

   提示

   在對數頻譜中找出亮點，忽略 DC 峰值與座標軸上的線條，並記得每個亮點都有鏡像雙胞胎。半徑剛好能蓋住亮點擴散範圍時效果最好：太小會留下殘餘條紋，太大會移除那些頻率附近的影像內容。陷波通過的輸出應該是乾淨的條紋，看不到任何場景。

6. 運用本章的觀念解釋：為什麼在每個步幅為 2 的層之前插入模糊，可以讓 CNN 在輸入影像平移一個像素時輸出更穩定？

   提示

   步幅 2 的降取樣讓取樣率減半，高於新奈奎斯特上限的頻率會以混疊的形式摺回來。平移一個像素會改變這些成分的相位，經過混疊後就改變了低頻輸出，於是後面的層看到不同的訊號。先用低通濾波器移除這些頻率，平移對輸出的影響就小得多。參見 [17]。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 4. publisher page
2. C. E. Shannon, “Communication in the presence of noise,” *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949. doi
3. M. Unser, “Sampling—50 years after Shannon,” *Proceedings of the IEEE*, vol. 88, no. 4, pp. 569–587, 2000. doi

4. J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation*, vol. 19, no. 90, 1965. doi
5. M. T. Heideman, D. H. Johnson and C. S. Burrus, “Gauss and the history of the fast Fourier transform,” *IEEE ASSP Magazine*, vol. 1, no. 4, pp. 14–21, 1984. doi
6. M. Frigo and S. G. Johnson, “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, 2005. doi
7. A. V. Oppenheim and J. S. Lim, “The importance of phase in signals,” *Proceedings of the IEEE*, vol. 69, no. 5, 1981. doi
8. A. V. Oppenheim, R. W. Schafer and T. G. Stockham, “Nonlinear filtering of multiplied and convolved signals,” *Proceedings of the IEEE*, vol. 56, no. 8, pp. 1264–1291, 1968. doi
9. M. Mathieu, M. Henaff and Y. LeCun, “Fast training of convolutional networks through FFTs,” arXiv:1312.5851, 2013. arXiv
10. L. Chi, B. Jiang and Y. Mu, “Fast Fourier convolution,” *Advances in Neural Information Processing Systems* (*NeurIPS*), vol. 33, 2020. proceedings
11. Y. Rao, W. Zhao, Z. Zhu, J. Lu and J. Zhou, “Global filter networks for image classification,” *NeurIPS*, 2021. arXiv
12. N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio and A. Courville, “On the spectral bias of neural networks,” *ICML*, 2019. arXiv
13. Z.-Q. J. Xu, Y. Zhang and T. Luo, “Overview frequency principle/spectral bias in deep learning,” arXiv:2201.07395, 2022. arXiv
14. Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart and A. Anandkumar, “Fourier neural operator for parametric partial differential equations,” arXiv:2010.08895, 2020. arXiv
15. N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart and A. Anandkumar, “Neural operator: Learning maps between function spaces,” *Journal of Machine Learning Research*, vol. 24, 2023. arXiv
16. K. Xu, M. Qin, F. Sun, Y. Wang, Y.-K. Chen and F. Ren, “Learning in the frequency domain,” *CVPR*, 2020. arXiv
17. R. Zhang, “Making convolutional networks shift-invariant again,” *ICML*, 2019. arXiv
18. T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen and T. Aila, “Alias-free generative adversarial networks,” arXiv:2106.12423, 2021. arXiv
19. J. Frank, T. Eisenhofer, L. Schönherr, A. Fischer, D. Kolossa and T. Holz, “Leveraging frequency analysis for deep fake image recognition,” *ICML*, 2020. arXiv
20. 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,” arXiv:2006.10739, 2020. arXiv
21. NumPy developers, “numpy.angle,” NumPy reference documentation, accessed 5 November 2026. API 文件.

IP 05・影像復原與重建

為模糊與雜訊建立退化模型，再用次序統計、陷波、逆濾波、Wiener 與受限最小平方濾波器把傷害還原，並從投影重建 CT 斷層影像。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/

你將學到

- 如何把影像受到的傷害寫成模型 $g = h \star f + \eta$，以及這個模型為什麼把「復原」和「增強」區分開來
- 常見的雜訊模型、如何從直方圖辨認它們，以及如何從平坦區塊量測雜訊參數
- 不同雜訊該配哪一種空間濾波器：平均濾波器、次序統計濾波器與自適應濾波器
- 如何在頻域用陷波濾波器（notch filter）去除週期性干擾
- 如何估計模糊並用逆濾波、Wiener 濾波、受限最小平方與幾何平均濾波器還原，以及為什麼直接相除會失敗
- CT 掃描儀如何把一組一維投影變回二維斷層：Radon 轉換、傅立葉切片定理與濾波反投影

先看全貌

第 3、4 章的目標是讓影像「看起來」更好。本章問的是更嚴格的問題：影像在受損之前原本長什麼樣子？要回答它，必須先有一個描述傷害的模型。有了模型，還原就變成一個定義清楚（至少是被充分理解）的數學問題。本章依照 Gonzalez 與 Woods 教科書的編排 [1]。

> **用最簡單的話說**
>
> 想像有人用濕手指抹過你的畫，又在上面撒了一些沙。如果只是「讓它變漂亮」，你可能會整張重新塗色。但如果你知道手指怎麼移動、沙長什麼樣，就能小心地把沙撥掉，再把被抹開的線條推回原位。復原就是這種細心的做法：先查清楚發生了什麼，再精準地把那件事倒轉回來。

只要真實場景才是重點，就需要復原：天文、顯微鏡、鑑識照片、醫學影像，以及每一支手機的相機處理流程。本章最後的「從投影重建影像」是同一個想法推到極致：CT 掃描儀根本看不到斷層本身，只看得到它從許多角度投下的「影子」，我們必須從這些影子重建出斷層。

影像退化／復原過程的模型

白話說：相機看到的是場景稍微模糊後的版本，上面再疊一層隨機雜訊。復原就是設法把場景找回來。

我們把退化寫成

$$g(x,y) = h(x,y) \star f(x,y) + \eta(x,y),$$

其中 $f(x,y)$ 是理想（未知）的影像，$h(x,y)$ 是**退化函數**，也稱為點擴散函數（point spread function, PSF），$\star$ 代表二維卷積，$\eta(x,y)$ 是加成性雜訊，$g(x,y)$ 是我們實際觀測到的影像。依第 4 章的卷積定理，同一個模型在頻域是

$$G(u,v) = H(u,v)\,F(u,v) + N(u,v),$$

其中大寫字母代表對應小寫函數的二維離散傅立葉轉換（DFT），(u,v) 是頻率座標。$H(u,v)$ 稱為**光學轉移函數**（optical transfer function, OTF）。

復原的目標是求出一個估計 $\hat{f}(x,y)$，讓它盡可能接近 $f(x,y)$。我們對 h 和 η 知道得越多，就能越接近。

復原與增強的差別

增強（第 3、4 章）大多是主觀的：調整對比或銳化，直到觀看者覺得好看為止。復原則是客觀的：它使用退化模型，並以結果和真實影像有多接近來評判，通常用可量測的準則，例如均方誤差。有些工具在兩章都會出現（例如中值濾波器），但使用的理由不同。在復原裡，我們挑某個濾波器，是因為雜訊模型告訴我們它是對的選擇。

本章先處理 $h = \delta$（沒有模糊、只有雜訊）的情況，再加入模糊。

雜訊模型

白話說：雜訊是疊在畫面上的隨機「顆粒」。不同的感測器與情境會產生不同的顆粒，每一種在直方圖上都有自己的指紋。

雜訊的空間與頻率特性

描述雜訊要問兩件事。空間上：某個像素的雜訊和鄰居的雜訊有沒有關聯？和影像本身有沒有關聯？除非特別說明，本章假設雜訊與位置無關，也與影像不相關。頻率上：雜訊的功率如何分布在 (u,v) 上？傅立葉頻譜大致為常數的雜訊稱為**白雜訊**（white noise），名稱來自包含所有可見光頻率的白光。週期性雜訊正好相反：它的功率集中在少數幾個孤立的頻率。

在這些假設下，雜訊完全由其數值的機率密度函數（probability density function, PDF）$p(z)$ 描述，z 代表強度值。

常見的 PDF

高斯（Gaussian）。最常用的模型，因為數學上方便，也很符合電子感測器的雜訊：

$$p(z) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-(z-\bar{z})^2/2\sigma^2},$$

其中 $\bar{z}$ 為平均值，σ 為標準差（σ^2 為變異數）。約 68% 的值落在 $\bar{z} \pm \sigma$ 之內，約 95% 落在 $\bar{z} \pm 2\sigma$ 之內。

Rayleigh。單邊、偏斜的密度，用來描述距離成像（range imaging）中的雜訊：

$$p(z) = \begin{cases} \dfrac{2}{b}(z-a)\, e^{-(z-a)^2/b}, & z \geq a \\ 0, & z < a \end{cases} \qquad \bar{z} = a + \sqrt{\pi b/4}, \quad \sigma^2 = \frac{b(4-\pi)}{4},$$

其中 a 決定起點位置，$b > 0$ 控制分散程度。

Erlang（gamma）。同樣偏斜，但尾巴更長：

$$p(z) = \begin{cases} \dfrac{a^b z^{b-1}}{(b-1)!}\, e^{-az}, & z \geq 0 \\ 0, & z < 0 \end{cases} \qquad \bar{z} = \frac{b}{a}, \quad \sigma^2 = \frac{b}{a^2},$$

其中 $a > 0$ 是速率，b 是正整數（形狀參數）。允許 b 為非整數，就得到一般的 gamma 密度。

指數（exponential）。即 $b = 1$ 的 Erlang 密度：

$$p(z) = a\, e^{-az} \ (z \geq 0), \qquad \bar{z} = \frac{1}{a}, \quad \sigma^2 = \frac{1}{a^2}.$$

它出現在雷射成像（斑點強度）中。

均勻（uniform）。區間 $[a, b]$ 內每個值出現的機率相同：

$$p(z) = \frac{1}{b-a} \ (a \leq z \leq b), \qquad \bar{z} = \frac{a+b}{2}, \quad \sigma^2 = \frac{(b-a)^2}{12}.$$

它主要用於模擬（量化誤差近似均勻分布）。

椒鹽（脈衝）雜訊（salt-and-pepper / impulse noise）。每個像素要嘛不變，要嘛以機率 P_s 被換成很亮的值（鹽），或以機率 P_p 被換成很暗的值（胡椒）。被破壞的像素比例 $P = P_s + P_p$ 稱為**雜訊密度**。故障的感測元件與傳輸位元錯誤會產生這種雜訊。它和其他雜訊不同，不是加上去的，而是取代原本的像素值。

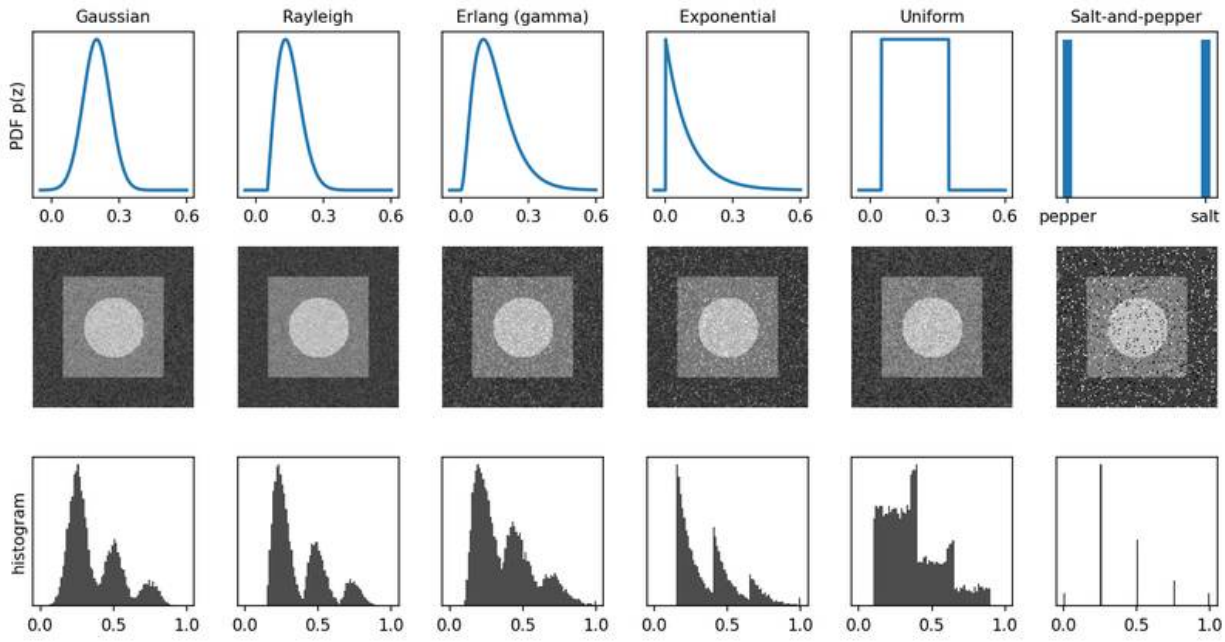


圖 *5.1* — 上：六種雜訊 *PDF*。中：一張具有三個平坦灰階的測試圖，分別被各種雜訊污染。下：直方圖。雜訊影像看起來差不多，但每個直方圖都包含三份雜訊 *PDF*（每個灰階一份），因此直方圖能辨認雜訊類型。脈衝雜訊是例外：它在 *0* 與 *1* 加上尖峰。

週期性雜訊

擷取影像時的電氣或機電干擾，可能在影像上疊加一個像 $A\sin(2\pi(u_0 x/M + v_0 y/N))$ 的正弦波，其中 A 為振幅，(u_0, v_0) 為頻率，$M \times N$ 為影像大小。它與位置相關，所以無法用 PDF 描述。不過在傅立葉頻譜中，每個正弦波都是位於 $\pm(u_0, v_0)$ 的一對亮點，因此在頻域很容易去除（見下方的週期性雜訊消除）。

估計雜訊參數

如果你能控制成像系統，就拍一張均勻照明的平坦目標：看到的一切都是雜訊。如果手上只有影像，就找一個本來應該是常數的小區域 S（天空、空白牆面、背景），計算它的直方圖 $p_S(z_i)$，也就是 S 中強度為 z_i 的像素比例。接著

$$\bar{z} = \sum_i z_i \, p_S(z_i), \qquad \sigma^2 = \sum_i (z_i - \bar{z})^2 \, p_S(z_i).$$

區塊直方圖的形狀告訴你該用哪一種 PDF，平均值與變異數再決定參數（Rayleigh、Erlang 的 a 和 b 等）。對脈衝雜訊，則計算落在極端值的像素數量來估計 P_s 與 P_p。

```
import numpy as np
from skimage import data, img_as_float
```

```
rng = np.random.default_rng(0)
f = img_as_float(data.camera())                  # values in [0, 1]

g_gauss = f + rng.normal(0.0, 0.05, f.shape)     # additive Gaussian, sigma = 0.05
g_unif = f + rng.uniform(-0.1, 0.1, f.shape)     # additive uniform on [-0.1, 0.1]
g_sp = f.copy()
r = rng.random(f.shape)
g_sp[r < 0.05] = 0.0                             # 5 % pepper
g_sp[r > 0.95] = 1.0                             # 5 % salt

# Estimate the noise from a patch that should be flat (here: sky, top-left)
patch = g_gauss[10:60, 10:60]
print(f"mean = {patch.mean():.3f}, std = {patch.std():.3f}")   # std close to 0.05
```

只有雜訊時的復原：空間濾波

白話說：如果唯一的問題是雜訊，就把每個像素換成它鄰域的一個「聰明摘要」。最好的摘要方式取決於雜訊種類。

沒有模糊時，模型簡化為 $g(x,y) = f(x,y) + \eta(x,y)$（脈衝雜訊則是取代）。以下 S_{xy} 是以 (x,y) 為中心的 $m \times n$ 視窗，(r,c) 走遍視窗內的像素。

平均濾波器

算術平均（arithmetic mean）：一般的平均：

$$\hat{f}(x,y) = \frac{1}{mn} \sum_{(r,c)\in S_{xy}} g(r,c).$$

它能降低高斯或均勻雜訊，但會模糊邊緣。

幾何平均（geometric mean）：乘積開 mn 次方根：

$$\hat{f}(x,y) = \Big[\prod_{(r,c)\in S_{xy}} g(r,c) \Big]^{1/mn}.$$

平滑效果與算術平均相近，但細節損失較少。

調和平均（harmonic mean）：

$$\hat{f}(x,y) = \frac{mn}{\sum_{(r,c)\in S_{xy}} 1/g(r,c)}.$$

它能處理鹽雜訊與類高斯雜訊，但對胡椒雜訊完全失效，因為只要一個接近 0 的像素，就會主宰倒數和。

Q 階反調和平均（contraharmonic mean）：

$$\hat{f}(x,y) = \frac{\sum_{(r,c)\in S_{xy}} g(r,c)^{Q+1}}{\sum_{(r,c)\in S_{xy}} g(r,c)^{Q}}.$$

Q 稱為階數。Q 為正時去除胡椒雜訊，為負時去除鹽雜訊，無法同時去除兩者。$Q = 0$ 即算術平均，$Q = -1$ 即調和平均。正負號選錯會造成災難，如圖 5.2 所示。

次序統計濾波器

這類濾波器把 S_{xy} 內的像素排序，再從排序結果中挑值。

- **中值（median）**：$\hat{f}(x,y) = \text{median}_{(r,c)\in S_{xy}}\, g(r,c)$。處理脈衝雜訊最好用的通用工具。它能去除孤立的離群值，模糊程度遠小於同尺寸的平均濾波器。
- **最大值（max）**與**最小值（min）**：取最大與最小值。最大值濾波器去除胡椒（暗）雜訊，最小值濾波器去除鹽（亮）雜訊。
- **中點（midpoint）**：$\frac{1}{2}(\max + \min)$。結合排序與平均，適合高斯或均勻雜訊。
- **alpha 修剪平均（alpha-trimmed mean）**：去掉最低與最高各 $d/2$ 個值，再對剩下的 $mn - d$ 個值 $g_R(r,c)$ 取平均：

$$\hat{f}(x,y) = \frac{1}{mn - d} \sum_{(r,c)\in S_{xy}} g_R(r,c).$$

$d = 0$ 時是算術平均，$d = mn - 1$ 時是中值。介於兩者之間時，它能處理混合雜訊，例如高斯雜訊加上少量脈衝。

```
import numpy as np
from scipy import ndimage as ndi

def arithmetic_mean(g, m=3):
    return ndi.uniform_filter(g, m)

def geometric_mean(g, m=3):
    return np.exp(ndi.uniform_filter(np.log(g + 1e-6), m))

def harmonic_mean(g, m=3):
    return 1.0 / ndi.uniform_filter(1.0 / (g + 1e-6), m)

def contraharmonic_mean(g, m=3, Q=1.5):
    g = g + 1e-6                                  # avoid 0 ** negative
    return ndi.uniform_filter(g ** (Q + 1), m) / ndi.uniform_filter(g ** Q, m)

def alpha_trimmed_mean(g, m=5, d=6):
    def trim(v):
        v = np.sort(v)
        return v[d // 2: v.size - d // 2].mean()
    return ndi.generic_filter(g, trim, size=m)

median = lambda g, m=3: ndi.median_filter(g, m)
maxf = lambda g, m=3: ndi.maximum_filter(g, m)    # removes pepper
```

```
minf = lambda g, m=3: ndi.minimum_filter(g, m)    # removes salt
midpoint = lambda g, m=3: 0.5 * (maxf(g, m) + minf(g, m))
```

自適應濾波器

上面的濾波器對每個像素一視同仁。自適應濾波器則依 S_{xy} 內的局部統計量改變行為，通常只需多花一點計算就能表現更好。

自適應局部雜訊消除濾波器（adaptive, local noise reduction filter）。令 σ_η^2 為雜訊變異數（整張影像估計一次），$\bar{z}_S$ 與 σ_S^2 為 S_{xy} 內的局部平均與局部變異數。濾波器為

$$\hat{f}(x,y) = g(x,y) - \frac{\sigma_\eta^2}{\sigma_S^2}\left[g(x,y) - \bar{z}_S\right].$$

可以分三種情況理解。若 $\sigma_\eta^2 = 0$，表示沒有雜訊，濾波器原封不動傳回 g。若局部變異數遠大於雜訊變異數，視窗裡很可能有邊緣，濾波器就保持接近 g，保住邊緣。若兩個變異數相等，視窗是平坦區域，濾波器傳回局部平均。實務上會把比值上限截在 1，因為局部變異數的估計值可能碰巧比 σ_η^2 小。

自適應中值濾波器（adaptive median filter）。固定尺寸的中值濾波器在脈衝稀疏時效果很好；雜訊密度一高，一個視窗裡的脈衝可能比真實像素還多，而加大固定視窗又會把一切都模糊掉。自適應中值濾波器在需要時才放大視窗，而且不去動沒有被污染的像素。令 $z_{\min}$、$z_{\max}$、z_{med} 為 S_{xy} 內的最小、最大與中值，z_{xy} 為中心像素，$S_{\max}$ 為允許的最大視窗尺寸 [2]。

- **階段 A：**若 $z_{\min} < z_{\text{med}} < z_{\max}$，中值不是脈衝，進入階段 B。否則放大視窗；若尺寸超過 $S_{\max}$，輸出 z_{med}；否則重複階段 A。
- **階段 B：**若 $z_{\min} < z_{xy} < z_{\max}$，中心像素不是脈衝，原樣輸出 z_{xy}。否則輸出 z_{med}。

這樣同時達成三個目標：去除脈衝、平滑其他雜訊，並減少一般中值濾波造成的邊緣變細與模糊。

```
import numpy as np
from scipy import ndimage as ndi

def adaptive_local(g, noise_var, m=7):
    """Adaptive local noise reduction: f = g - (s_n^2 / s_L^2)(g - m_L)."""
    mean = ndi.uniform_filter(g, m)
    var = ndi.uniform_filter(g * g, m) - mean ** 2
    ratio = np.minimum(noise_var / np.maximum(var, 1e-12), 1.0)  # never above 1
    return g - ratio * (g - mean)

def adaptive_median(g, s_max=7):
    out, done = g.copy(), np.zeros(g.shape, bool)
    for s in range(3, s_max + 1, 2):
        zmin, zmax = ndi.minimum_filter(g, s), ndi.maximum_filter(g, s)
        zmed = ndi.median_filter(g, s)
        stage_a = (zmin < zmed) & (zmed < zmax) & ~done          # median is not an impulse
```

```
        keep = (zmin < g) & (g < zmax)                          # centre pixel is not an
         ↪  impulse
        out[stage_a] = np.where(keep, g, zmed)[stage_a]
        done |= stage_a
    out[~done] = zmed[~done]                                    # window grew to s_max
    return out
```

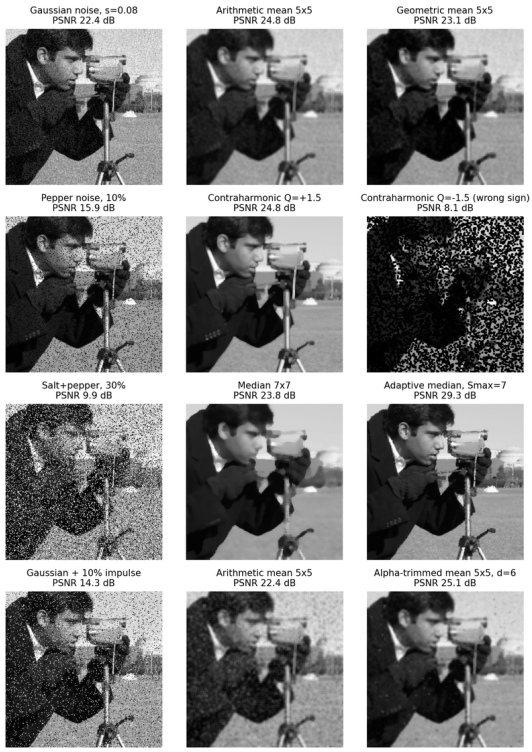


圖 *5.2* — 讓濾波器對上雜訊。第 *1* 列：高斯雜訊，算術與幾何平均。第 *2* 列：胡椒雜訊；$Q = +1.5$ 的反調和濾波器能去除它，$Q = -1.5$ 則讓情況更糟。第 *3* 列：*30%* 椒鹽雜訊；*7×7* 中值濾波器把細節抹平，自適應中值保留了多得多的細節。第 *4* 列：高斯加脈衝雜訊；*alpha* 修剪平均勝過一般平均。*PSNR* 為相對乾淨影像的峰值訊雜比（越高越好）。

以頻域濾波消除週期性雜訊

白話說：週期性雜訊是藏在影像裡的幾個純「音調」。它們在頻譜上是亮點，所以我們只挖掉那幾個點，其他一律保留。

陷波拒斥與陷波通過濾波器

陷波濾波器（notch filter）會拒斥（或通過）以指定中心為圓心的小鄰域內的頻率。由於實數影像的 DFT 具有共軛對稱性，陷波一定成對出現，相對頻譜中心位於 (u_k, v_k) 與 $(-u_k, -v_k)$。具有 Q 對陷波的陷波拒斥濾波器，是一組中心被移到陷波位置的高通濾波器的乘積：

$$H_{NR}(u,v) = \prod_{k=1}^{Q} H_k(u,v)\, H_{-k}(u,v),$$

其中 H_k 與 H_{-k} 是中心分別在 (u_k, v_k) 與 $(-u_k, -v_k)$ 的高通濾波器（理想、Butterworth 或高斯，同第 4 章）。例如高斯陷波使用 $H_k = 1 - e^{-D_k^2/2D_0^2}$，$D_k(u,v)$ 是 (u,v) 到 (u_k, v_k) 的距離，D_0 是陷波半徑。對應的陷波通過濾波器為

$$H_{NP}(u,v) = 1 - H_{NR}(u,v).$$

陷波通過濾波器本身也很有用：把它乘上 G 再轉回空間域，就得到只有干擾圖樣的影像。其他形狀的陷波（例如沿座標軸的細線）可去除掃描線圖樣。

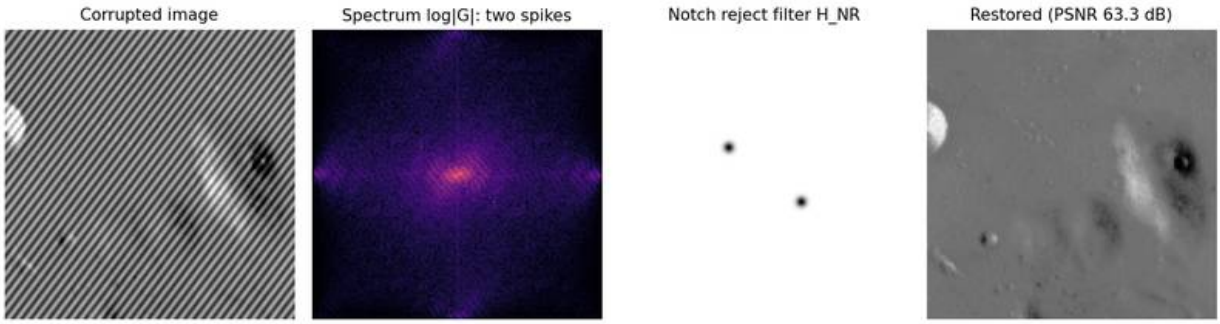


圖 *5.3* — 正弦干擾圖樣（左）在頻譜中是兩個孤立的亮點。用兩個高斯陷波去除它們，反 *DFT* 就還原出乾淨的影像。這裡幾乎完美還原，是因為干擾剛好是單一正弦波；真實的干擾通常分散在好幾個頻率上。

```
import numpy as np
from skimage import data, img_as_float

f = img_as_float(data.moon())
M, N = f.shape
y, x = np.mgrid[:M, :N]
g = f + 0.2 * np.sin(2 * np.pi * (40 * y / M + 25 * x / N))    # periodic interference

G = np.fft.fftshift(np.fft.fft2(g))
U, V = np.mgrid[-M // 2:M // 2, -N // 2:N // 2]
H = np.ones((M, N))
for u0, v0 in [(40, 25), (-40, -25)]:                          # a spike and its mirror
    D2 = (U - u0) ** 2 + (V - v0) ** 2
    H *= 1 - np.exp(-D2 / (2 * 3.0 ** 2))                      # Gaussian notch, radius ~3
f_hat = np.real(np.fft.ifft2(np.fft.ifftshift(H * G)))
```

僅線上版：最佳陷波濾波 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/最佳陷波濾波/*

線性、位置不變的退化

白話說：我們假設模糊是「公平」的。光線加倍，模糊後的光也加倍；而且左上角的模糊和右下角的一樣。

把（不含雜訊的）退化寫成運算子 $\mathcal{H}$，即 $g = \mathcal{H}[f] + \eta$。若對任意影像 f_1, f_2 與常數 a, b 都有

$$\mathcal{H}[af_1 + bf_2] = a\,\mathcal{H}[f_1] + b\,\mathcal{H}[f_2],$$

則 $\mathcal{H}$ 是**線性**的。若平移輸入只會平移輸出，

$$\mathcal{H}[f(x-\alpha, y-\beta)] = g(x-\alpha, y-\beta)$$

對任意平移量 (α, β) 都成立，則 $\mathcal{H}$ 是**位置不變**的。對這種運算子，把 f 寫成許多平移脈衝的和。線性讓我們可以分別對每個脈衝套用 $\mathcal{H}$，位置不變則保證每個脈衝都產生相同的響應 h，只是位置不同。把這些響應加總，正是卷積：

$$g(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\alpha,\beta)\,h(x-\alpha, y-\beta)\,d\alpha\,d\beta + \eta(x,y) = (h \star f)(x,y) + \eta(x,y).$$

這裡 $h(x,y) = \mathcal{H}[\delta(x,y)]$ 是**脈衝響應**，在光學中就是點擴散函數：單一光點所成的像。這就是為什麼還原模糊影像常被稱為**反卷積**（deconvolution），也是下面的濾波器都在頻域建構的原因。

許多真實退化只是近似線性位置不變（LPI）。鏡頭在影像角落可能更模糊，在靜止背景前移動的物體也和背景模糊得不一樣。空間變異的復原方法存在，但代價高得多；常見的變通是把影像切成大致符合 LPI 的小塊。

估計退化函數

白話說：要還原模糊之前，得先知道模糊是什麼。可以從照片裡找線索、自己測試相機，或從物理推導。

觀察法

在模糊影像中找一個結構簡單而明顯的小區域 $g_s(x,y)$，例如一條邊或一個小亮點，而且訊號遠強於雜訊。用手工（或銳化加上判斷）建立該區域應有樣貌的估計 $\hat{f}_s(x,y)$。則

$$H_s(u,v) = \frac{G_s(u,v)}{\hat{F}_s(u,v)}$$

就是從這個小區域估計出的 OTF。若模糊是位置不變的，H_s 告訴我們整張影像 H 的形狀，再放大到全尺寸即可。這很費工，主要用於只有影像可用的情況，例如歷史照片。

實驗法

如果手上有相同（或等效）的成像系統，就在相同設定下拍一個極小的亮點。夠小的亮點近似強度為 A 的脈衝，其傅立葉轉換是常數 A。觀測到的亮點影像直接給出 OTF：

$$H(u,v) = \frac{G(u,v)}{A}.$$

顯微鏡使用者正是用小於解析度的螢光微珠這樣做。

數學建模法

有時物理會直接給答案。本章會用到兩個經典模型。

大氣擾流（atmospheric turbulence）。透過擾動空氣長時間曝光的成像，常以下式建模：

$$H(u,v) = e^{-k\,(u^2+v^2)^{5/6}},$$

其中常數 k 隨擾流強度增加。除了 5/6 次方之外，它看起來像高斯低通濾波器，而且永遠不會剛好等於零。

均勻直線運動模糊（uniform linear motion blur）。假設影像在長度為 T 的曝光期間等速移動，x 方向總共移動 a、y 方向總共移動 b。感測器把場景每一個平移後的副本都累加起來，這個平均動作的轉移函數為

$$H(u,v) = \frac{T}{\pi(ua+vb)}\,\sin\!\big[\pi(ua+vb)\big]\,e^{-j\pi(ua+vb)}.$$

僅線上版：推導運動模糊的轉移函數 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/推導運動模糊的轉移函數/*

這是沿運動方向的 sinc 函數，再乘上線性相位項。關鍵是：只要 $ua + vb$ 是非零整數，它就剛好等於**零**，因此整條整條的頻率線被完全摧毀（圖 5.4）。

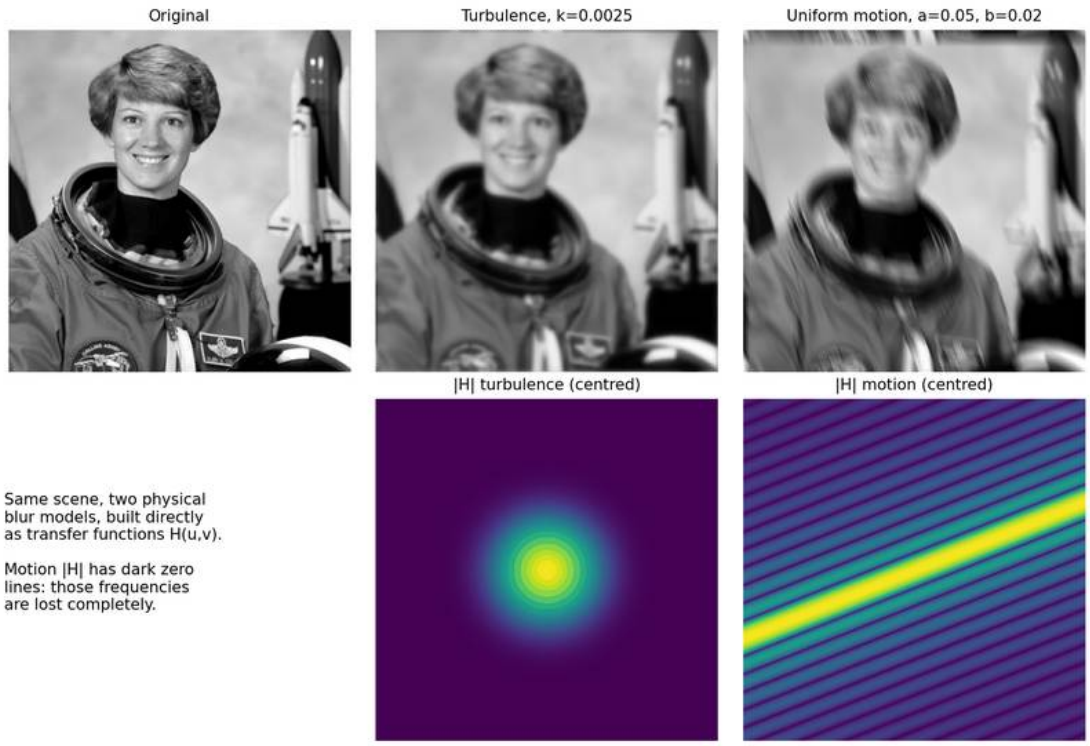


圖 *5.4* — 由物理推導出的兩種退化函數。擾流的 *OTF* 是一個平滑的隆起，會衰減但永遠不為零。運動的 *OTF* 是沿運動方向的 *sinc* 脊線，旁邊有一條條平行、剛好為零的暗線：在這些頻率上，模糊影像完全不含場景的資訊。

```
import numpy as np

def freq_grid(shape):
    u = np.fft.fftfreq(shape[0]) * shape[0]            # 0, 1, ..., -1 (un-centred)
    v = np.fft.fftfreq(shape[1]) * shape[1]
    return np.meshgrid(u, v, indexing="ij")

def motion_blur_otf(shape, a, b, T=1.0):
    U, V = freq_grid(shape)
    s = np.pi * (U * a + V * b)
    return T * np.sinc(s / np.pi) * np.exp(-1j * s)    # np.sinc(x) = sin(pi x) / (pi x)

def turbulence_otf(shape, k):
    U, V = freq_grid(shape)
    return np.exp(-k * (U ** 2 + V ** 2) ** (5 / 6))
```

實作細節

像上面這樣直接在頻域建構 H，得到的是循環卷積：模糊會從影像邊界繞回另一邊。做實驗時這很理想，因為模型和濾波器完全吻合。處理真實照片時，要先補邊或讓邊界漸弱再做反卷積，否則邊緣會出現強烈的振鈴（ringing）。

逆濾波

白話說：既然模糊把每個頻率乘上了 H，那就除回去。這是最直覺的想法，而只要有雜訊，它就會慘敗。

直接逆濾波器（direct inverse filter）的估計為

$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}.$$

代入 $G = HF + N$ 就看出問題：

$$\hat{F}(u,v) = F(u,v) + \frac{N(u,v)}{H(u,v)}.$$

即使完全知道 H，也無法精確還原 F，因為 N 是未知的。更糟的是，模糊讓 $|H|$ 在高頻非常小（運動模糊甚至剛好為零），而雜訊大致是白的，在高頻仍保有功率。於是 N/H 變得極大，淹沒整張影像。在圖 5.5 中，雖然雜訊標準差只有強度範圍的 0.3% 與 1%，完整的逆濾波仍把兩張影像都變成純雜訊。

一個局部的補救是只在頻譜原點附近（$|H|$ 夠大的地方）做逆濾波，例如把 $\hat{F}$ 乘上半徑 D_0 的陡峭 Butterworth 低通濾波器。對 H 平滑遞減的擾流模糊有效；對運動模糊則不太行，因為 H 的零線也穿過低頻。半徑還得手動調整。我們需要一個能自動決定每個頻率該信任多少的方法。

最小均方誤差（Wiener）濾波

白話說：在每個頻率問一句「這裡畫面比較多，還是雜訊比較多？」畫面佔優勢的地方就還原模糊；雜訊佔優勢的地方就收手。

Wiener 濾波器把影像與雜訊都視為隨機場，尋找使均方誤差

$$e^2 = E\{(f - \hat{f})^2\}$$

最小的估計，$E\{\cdot\}$ 代表期望值。假設雜訊與影像不相關、其中之一平均為零，而且 $\hat{f}$ 是 g 的線性函數。對平穩訊號，DFT 會讓各頻率彼此解耦，因此最佳增益可以一個頻率一個頻率地選。結果為

$$\hat{F}(u,v) = \left[\frac{1}{H(u,v)}\,\frac{|H(u,v)|^2}{|H(u,v)|^2 + S_\eta(u,v)/S_f(u,v)}\right] G(u,v),$$

其中 $S_f(u,v) = E\{|F(u,v)|^2\}$ 是影像的**功率頻譜**（power spectrum），$S_\eta(u,v) = E\{|N(u,v)|^2\}$ 是雜訊的功率頻譜，而下文用到的 * 表示共軛複數。

僅線上版：推導 *Wiener* 濾波器 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/推導-wiener-濾波器/*

第一個因子就是逆濾波器。第二個因子介於 0 和 1 之間。訊號遠強於雜訊時，$S_\eta/S_f \approx 0$，這個因子接近 1，濾波器表現得像逆濾波器。雜訊佔優勢時，因子趨近 0，把那個頻率關掉。若某頻率 $H = 0$，括號在該處為 0（寫成 $H^*/(|H|^2 + S_\eta/S_f)$ 就不必除以零）。沒有雜訊時 $S_\eta = 0$，Wiener 濾波器就是逆濾波器。

常數 K 近似

我們很少知道 S_f，因為那正是我們要找的影像的功率頻譜。常見的捷徑是把比值 S_η / S_f 換成一個常數 K：

$$\hat{F}(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + K} G(u,v).$$

K 可以目視調整，或選擇讓某個品質指標最佳的值。雜訊變異數與影像變異數的比值（也就是訊雜比的倒數）是不錯的起點。K 小，結果較銳利但雜訊多；K 大，結果較平滑。

```
import numpy as np
from skimage import data, img_as_float

rng = np.random.default_rng(0)
f = img_as_float(data.camera())[::2, ::2]                # 256 x 256
H = motion_blur_otf(f.shape, a=0.1, b=0.1)               # from the snippet above
g = np.real(np.fft.ifft2(H * np.fft.fft2(f))) + rng.normal(0, 0.01, f.shape)
G = np.fft.fft2(g)

def wiener(G, H, K):
    return np.real(np.fft.ifft2(np.conj(H) / (np.abs(H) ** 2 + K) * G))

f_hat = wiener(G, H, K=0.01)
```

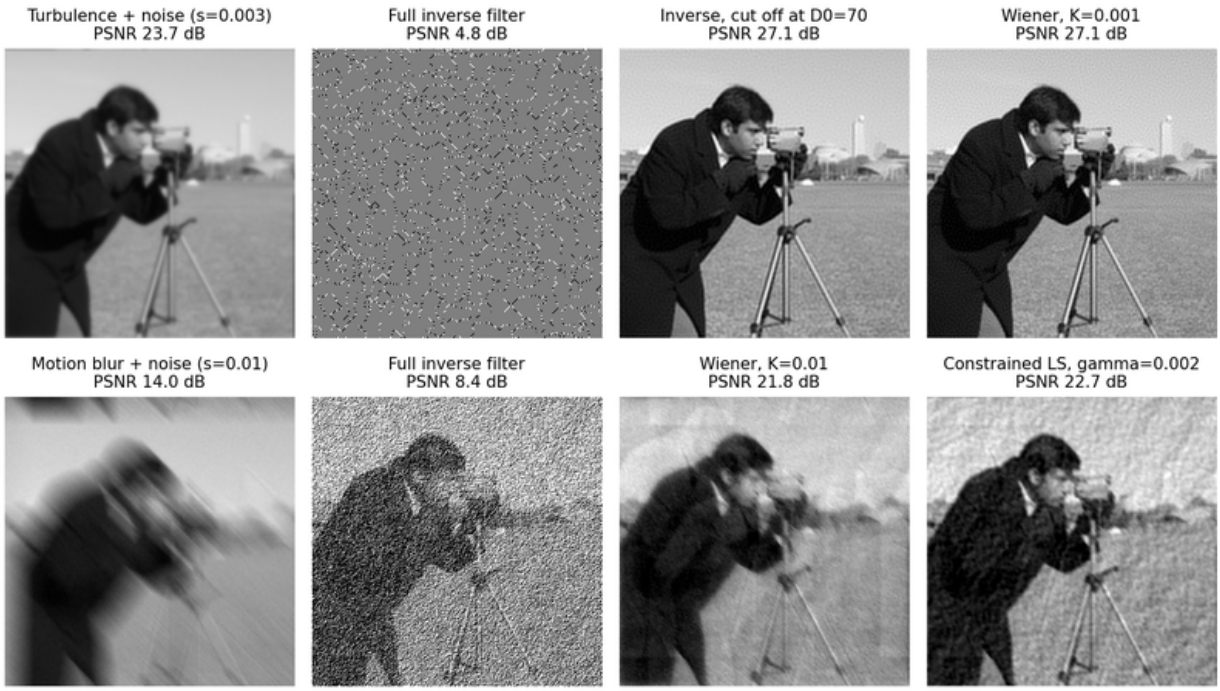


圖 *5.5* — 上列：擾流模糊（$k = 0.0025$）加微弱雜訊。完整逆濾波被放大的雜訊摧毀；限制在半徑 *70* 內的逆濾波可行，而 *Wiener* 濾波器不必挑半徑就達到相同品質。下列：斜向運動模糊加雜訊。

逆濾波失敗；*Wiener* 與受限最小平方濾波器找回了場景，但留下一些振鈴與顆粒。

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

拖動滑桿，比較運動模糊加雜訊的影像與其 Wiener 復原結果（512 × 512 完整版，$K = 0.01$）。



運動模糊 + 雜訊



Wiener，K=0.01

帶有斜向運動模糊與雜訊的 cameraman 影像，與 Wiener 濾波結果比較

受限最小平方濾波

白話說：在所有「扣掉預期雜訊量後，能解釋觀測結果」的影像之中，挑最平滑的那一張。

Wiener 濾波器需要功率頻譜，或一個意義不明確的常數 K。**受限最小平方**（constrained least squares, CLS）濾波只需要雜訊的平均值與變異數 [3]。以矩陣向量形式，把像素排成向量，模型為

$$\mathbf{g} = \mathbf{H}\mathbf{f} + \boldsymbol{\eta},$$

其中 $\mathbf{g}, \mathbf{f}, \boldsymbol{\eta}$ 是 $MN \times 1$ 向量，$\mathbf{H}$ 是 $MN \times MN$ 的模糊矩陣。這些矩陣大到無法直接求反矩陣，但因為 $\mathbf{H}$ 代表循環卷積，它可以被 DFT 對角化，這正是頻域閉式解得以存在的原因。

這類反問題是**病態**的（ill-conditioned）：$\mathbf{g}$ 的微小變化會造成解的巨大變化。CLS 的對策是偏好平滑的解。它以拉普拉斯運算子 ∇^2 衡量粗糙程度，求解

$$\min_{\hat{\mathbf{f}}} \; C = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[\nabla^2 \hat{f}(x,y) \right]^2 \quad \text{subject to} \quad \|\mathbf{g} - \mathbf{H}\hat{\mathbf{f}}\|^2 = \|\boldsymbol{\eta}\|^2,$$

其中 $\|\cdot\|$ 為歐氏範數。這個限制條件的意思是：殘差應該剛好和雜訊一樣大，不能更大（等於忽略資料），也不能更小（等於在擬合雜訊）。用拉格朗日乘數求得的解為

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \gamma\,|P(u,v)|^2}\right] G(u,v),$$

其中 $\gamma \geq 0$ 是乘數，$P(u,v)$ 是拉普拉斯卷積核

$$p(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

補零到 $M \times N$ 之後的 DFT。$\gamma = 0$ 時回到逆濾波器。和常數 K 的 Wiener 濾波器比較：CLS 把平坦的 K 換成 $\gamma|P|^2$，它在低頻小、高頻大，因此正好在雜訊容易佔優勢的地方收手。

γ 不是可以隨意轉的旋鈕：它要讓限制條件成立，也就是讓殘差 $\|\mathbf{g} - \mathbf{H}\hat{\mathbf{f}}\|^2$ 等於雜訊能量 $\|\boldsymbol{\eta}\|^2 = MN\,(\sigma_\eta^2 + \bar{\eta}^2)$，$\sigma_\eta^2$ 與 $\bar{\eta}$ 是雜訊的變異數與平均值，而我們已經知道如何從平坦區塊估計它們。

僅線上版：選擇 *CLS* 乘數 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/選擇-cls-乘數/*

```
def cls(G, H, gamma):
    p = np.zeros(H.shape)
    p[[0, 0, 0, 1, -1], [0, 1, -1, 0, 0]] = [4, -1, -1, -1, -1]   # Laplacian centred at (0,
    ↪  0)
    P = np.fft.fft2(p)
    return np.real(np.fft.ifft2(np.conj(H) / (np.abs(H) ** 2 + gamma * np.abs(P) ** 2) * G))

f_hat = cls(G, H, gamma=0.002)
```

Wiener 和 CLS 哪個比較好？一般而言沒有定論。Wiener 是對一群具有假設功率頻譜的影像平均而言最佳；CLS 則針對手上這一張影像，當雜訊統計量已知時，對這張影像可能得到更好的結果。若 K 或 γ 是手動調的（這很常見），兩者看起來往往差不多（圖 5.5）。

幾何平均濾波器

白話說：一個旋鈕，在「完全還原模糊」和「對雜訊保守一點」之間滑動。

幾何平均濾波器（geometric mean filter）結合了逆濾波器與（參數化）Wiener 濾波器：

$$\hat{F}(u,v) = \left[\frac{H^*(u,v)}{|H(u,v)|^2}\right]^{\alpha} \left[\frac{H^*(u,v)}{|H(u,v)|^2 + \beta\,\dfrac{S_\eta(u,v)}{S_f(u,v)}}\right]^{1-\alpha} G(u,v),$$

其中 α 與 β 是非負實數常數。特殊情況：

- $\alpha = 1$：逆濾波器。
- $\alpha = 0$：**參數化 Wiener 濾波器**；$\beta = 1$ 時就是標準 Wiener 濾波器。

- $\alpha = 1/2$、$\beta = 1$：逆濾波器與 Wiener 濾波器的幾何平均，這也是這個家族名稱的由來。它又稱為**頻譜等化濾波器**（spectrum equalization filter）：可以驗證此時輸出的期望功率頻譜 $|W|^2(|H|^2 S_f + S_\eta)$ 剛好等於原始影像的功率頻譜 S_f。

$\beta > 1$ 時濾波器比 Wiener 更保守；$\beta < 1$ 時更大膽。

```
def geometric_mean_filter(G, H, alpha, beta, K):
    """K stands in for S_eta / S_f, as in the constant-K Wiener filter."""
    H2 = np.abs(H) ** 2
    inv = np.conj(H) / np.maximum(H2, 1e-12)
    wie = np.conj(H) / (H2 + beta * K)
    # Both factors have the phase of conj(H): combine magnitudes, keep that phase.
    return np.real(np.fft.ifft2(np.abs(inv) ** alpha * np.abs(wie) ** (1 - alpha)
                                * np.exp(1j * np.angle(wie)) * G))
```

從投影重建影像

白話說：從許多角度讓 X 光穿過身體。每個角度得到一個「影子」，它只告訴我們光線一路總共穿過了多少東西。把夠多的影子巧妙地組合起來，就能重建出身體內部的切面。

電腦斷層（CT）原理

在 X 光 CT 中，射源與一排偵測器繞著病人旋轉。在每個角度，每個偵測器量測 X 光沿其直線路徑被衰減了多少。沿一條射線的衰減是累加的（強度比的對數等於衰減係數的線積分），所以每個量測值都是未知斷層 $f(x, y)$ 的一個**線積分**。同一角度的一組量測值稱為一個**投影**（projection）。CT 掃描儀從平移加旋轉的單一射源－偵測器對（平行射束），演進到配有多個偵測器的扇形射束，再到能擷取整個體積的螺旋與錐形射束；但下面的重建數學是它們共同的核心 [4]。

投影與 Radon 轉換

以法線式描述一條直線：$x\cos\theta + y\sin\theta = \rho$，$\rho$ 是它到原點的距離，θ 是法線的角度。f 沿該直線的線積分為

$$g(\rho, \theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\,\delta(x\cos\theta + y\sin\theta - \rho)\,dx\,dy,$$

其中 δ 是 Dirac 脈衝，只保留直線上的點。把 g 視為 (ρ, θ) 的函數，就是 f 的 **Radon 轉換**（Radon transform）。以 θ 為一軸、ρ 為另一軸畫成影像，稱為**正弦圖**（sinogram）：斷層中的單一點會畫出一條正弦曲線 $\rho = x_0\cos\theta + y_0\sin\theta$。

最簡單的逆向做法是**反投影**（backprojection）：把每個投影沿著它來的方向抹回整張影像，再把所有角度的抹痕加起來：

$$f_\theta(x, y) = g(x\cos\theta + y\sin\theta,\ \theta), \qquad f_{BP}(x, y) = \int_0^{\pi} f_\theta(x, y)\,d\theta.$$

這會把質量放到正確的位置，但也會放到每條射線經過的所有地方，所以結果是 f 被嚴重模糊的版本（圖 5.6 第三格）。這個模糊並非隨機：它等於與 $1/r$ 做卷積，r 是到中心的距離。傅立葉切片定理告訴我們如何去除它。

傅立葉切片定理

令 $G(\omega,\theta)$ 為固定角度 θ 下，投影 $g(\rho,\theta)$ 對 ρ 的一維傅立葉轉換：

$$G(\omega,\theta)=\int_{-\infty}^{\infty} g(\rho,\theta)\,e^{-j2\pi\omega\rho}\,d\rho.$$

傅立葉切片定理（Fourier-slice theorem，又稱投影切片定理）指出

$$G(\omega,\theta)=F(\omega\cos\theta,\ \omega\sin\theta),$$

其中 $F(u,v)$ 是 f 的二維傅立葉轉換。換句話說：一個投影的一維轉換，等於物體二維轉換中通過原點、角度為 θ 的一條徑向切片。證明很短：把 Radon 積分代入 G，積掉 delta，指數就變成 $-j2\pi\omega(x\cos\theta+y\sin\theta)$，這正是二維轉換在 $(u,v)=(\omega\cos\theta,\omega\sin\theta)$ 的值。

所以原則上，所有角度的投影會一片一片填滿二維頻譜，再做二維反轉換就得到 f。實務上這些切片是在極座標網格上取樣：原點附近密、遠處稀，內插到直角座標網格容易出錯。這種不均勻取樣也是單純反投影會模糊的原因：它過度加權了低頻。

平行射束的濾波反投影

把二維反傅立葉轉換寫成極座標 $u=\omega\cos\theta$、$v=\omega\sin\theta$。面積元素 $du\,dv$ 變成 $|\omega|\,d\omega\,d\theta$，再用傅立葉切片定理：

$$f(x,y)=\int_0^{\pi}\left[\int_{-\infty}^{\infty}|\omega|\,G(\omega,\theta)\,e^{j2\pi\omega\rho}\,d\omega\right]_{\rho=x\cos\theta+y\sin\theta} d\theta.$$

由內往外讀：對每個角度，把投影的一維頻譜乘上 $|\omega|$（**斜坡濾波器**，ramp filter），再轉回來；接著把濾波後的投影反投影，並對所有角度加總。這就是**濾波反投影**（filtered backprojection, FBP）。斜坡濾波器正好抵消了單純反投影的 $1/r$ 模糊。

斜坡濾波器會無限增長，所以會放大高頻雜訊，而且不可積分。實務上會限制頻寬並乘上平滑的窗函數，例如 Hamming 窗：在頻帶 $|\omega|\le W/2$ 內 $h(\omega)=c+(1-c)\cos(2\pi\omega/W)$，頻帶外為 0，$W$ 是頻寬，$c=0.54$。窗函數在 $\omega=0$ 時為 1，在頻帶邊緣降到 $2c-1=0.08$。它犧牲一點銳利度，換來少得多的振鈴與雜訊。Shepp 與 Logan 在提出那個著名頭部假體（phantom）的論文中一併提出的濾波器 [5]，至今仍是標準選項之一。視角數量也很重要：角度太少時，FBP 會產生條紋（圖 5.6 最後一格）。

```
import numpy as np
from skimage.data import shepp_logan_phantom
from skimage.transform import radon, iradon, rescale

f = rescale(shepp_logan_phantom(), 0.64)                 # 256 x 256
theta = np.linspace(0.0, 180.0, 180, endpoint=False)     # projection angles in degrees
```

```
sinogram = radon(f, theta=theta)                          # shape: (detector bins, angles)
backproj = iradon(sinogram, theta=theta, filter_name=None)    # blurry, no filter
fbp = iradon(sinogram, theta=theta, filter_name="ramp")       # filtered backprojection
rmse = np.sqrt(np.mean((fbp - f) ** 2))
```

`iradon` 也接受 `"shepp-logan"`、`"cosine"`、`"hamming"` 與 `"hann"` 窗函數，而 `skimage.transform.iradon_sart` 提供迭代式的替代方案 [6]。

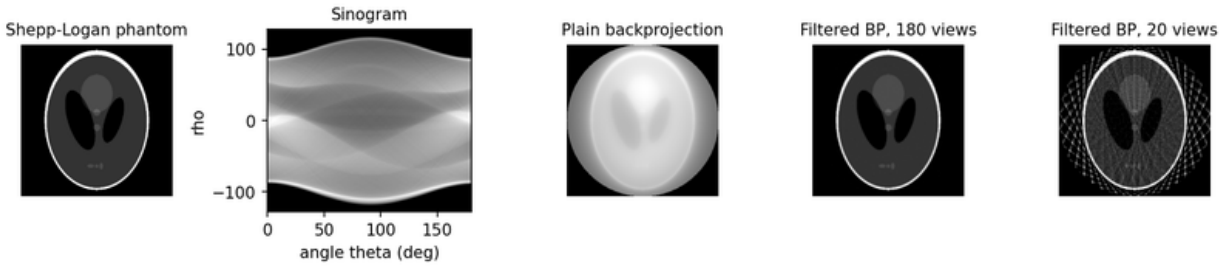


圖 *5.6* — 從斷層到正弦圖，再回到斷層。假體中的每一點在正弦圖中畫出一條正弦曲線。單純反投影得到模糊的光暈；加上斜坡濾波器（*FBP*）就還原出斷層。只有 *20* 個視角時，*FBP* 出現條紋偽影，這正是學習式重建方法要解決的問題（見現代觀點）。

僅線上版：扇形射束濾波反投影 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/扇形射束濾波反投影/*

現代觀點

本章的工具不是博物館裡的展品。$g = h \star f + \eta$ 仍是這個領域陳述問題的方式，類似 Wiener 的步驟也仍藏在現代網路裡。深度學習改變的是先驗（prior）：我們不再假設「影像是平滑的」（CLS）或「影像的功率頻譜是 S_f」（Wiener），而是從資料中學習乾淨影像長什麼樣子。想更全面地了解本章以外的深度影像復原（超解析度、除霧、除雨、低光增強、all-in-one 復原與擴散先驗），可以參考 DLCV 09・深度影像復原。以下的綜述聚焦在本章的三個主題：去雜訊、去模糊與 CT 重建。

僅線上版：現代觀點綜述詳讀 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/papers/現代觀點綜述詳讀/*

去雜訊：從區塊到網路

兩個經典方法代表了手工設計去雜訊的巔峰。**非局部平均**（non-local means, NLM）[7] 把每個像素換成一群「周圍區塊看起來相似」的像素的加權平均，不管它們在影像的哪裡：這是把自適應濾波器的想法推得更遠。**BM3D** [8] 把相似區塊堆成三維堆疊，在轉換域中聯合濾波，最後再做一次 Wiener 型的步驟。它當了大約十年的標準基準。

僅線上版：非局部平均去雜訊 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/非局部平均去雜訊/*

僅線上版：*Non-Local Means*：用相似區塊決定哪些像素可以平均 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/papers/non-local-means 用相似區塊決定哪些像素可以平均/*；*BM3D*：把相似區塊分組，協同去除雜訊 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/papers/bm3d 把相似區塊分組協同去除雜訊/*

DnCNN [9] 證明了一個訓練來預測雜訊（殘差學習）的單純深度卷積網路，在高斯雜訊上勝過 BM3D，而且單一模型能處理未知的雜訊強度。之後的發展可以從兩篇綜述看出：

- **Tian 等人（2020）** [10] 依雜訊類型整理深度去雜訊方法：合成高斯雜訊、真實相機雜訊、未知（盲）雜訊與混合情況。本章的雜訊模型仍是整個分類的主軸。
- **Elad、Kawar 與 Vaksman（2023）** [11] 主張去雜訊器的意義早已超過去雜訊本身：好的去雜訊器可以當作其他反問題的先驗，也是擴散式影像生成的引擎。

去模糊與一般反問題

- **Zhang 等人（IJCV 2022）** [12] 綜述深度去模糊。像本章這樣模糊已知時，工作主要在壓制逆濾波造成的雜訊放大與振鈴；盲去模糊還得估計 h，仍是比較難的問題。
- **Ongie 等人（2020）** [13] 依「對 **H** 了解多少」與訓練方式，整理 $\mathbf{g} = \mathbf{H}\mathbf{f} + \boldsymbol{\eta}$ 問題的深度方法。對 **H** 了解多少，是第一個設計決策。
- **Kamilov 等人（2023）** [14] 回顧隨插即用（plug-and-play, PnP）方法：在物理步驟（$\mathbf{g} \approx \mathbf{H}\mathbf{f}$）與呼叫去雜訊器（代替先驗）之間交替。DPIR [15] 是廣泛使用的例子。

這和本章的關係很直接。對模糊 H，這種分裂方法的物理步驟要在每個頻率求使 $|G - HF|^2 + \mu|F - Z|^2$ 最小的 $\hat{F}$，其中 Z 是目前去雜訊後的估計，μ 是懲罰權重。它的解是 $\hat{F} = (H^*G + \mu Z)/(|H|^2 + \mu)$，也就是一個被拉向 Z（而不是拉向零）的常數 K Wiener 濾波器。

CT 重建：學習式與展開式

資料完整、雜訊低時，FBP 又快又準。但為了降低輻射劑量而減少資料時，它就吃力了：視角變少會產生條紋（圖 5.6），每個視角的光子變少會產生雜訊。經典解法是搭配全變分（total variation）等手工先驗的迭代方法（選讀：全變分正則化 (*https://blog.shyandram.dev/zh-tw/learn/image-processing/restoration-reconstruction/notes/全變分正則化/*)，見「補充資料」頁）。深度學習提供兩種主要模式，Wang、Ye 與 De Man [19] 針對 CT 及其他斷層成像模態做了全面回顧：

- **後處理：**先做 FBP，再用 CNN 去除偽影，例如 **FBPConvNet** [16]。
- **展開式重建：**固定迭代演算法的迭代次數，把每次迭代的部分步驟換成訓練過的小型網路，例如 **Learned Primal-Dual** [17]。Monga、Li 與 Eldar [18] 綜述了這種設計模式。

沒有改變的是：Radon 轉換與傅立葉切片定理定義了上述每個方法都使用的前向模型，而 FBP 仍是臨床掃描儀的預設方法，也是許多學習式方法的輸入或起點。學習式方法也帶來本章線性濾波器沒有的新風險：網路可能產生看似合理、但資料中並不存在的結構。這就是為什麼在醫療應用上，必須在真實且分布外的掃描上做驗證。

重點整理

- 復原是客觀的：把退化建模為 $g = h \star f + \eta$（或 $G = HF + N$），再把這個模型反過來。
- 從平坦區塊的直方圖辨認雜訊類型，再從同一區塊估計平均值與變異數。
- 讓空間濾波器對上雜訊：類高斯雜訊用平均類濾波器；單邊脈衝用正負號正確的反調和濾波器；椒鹽雜訊用中值與自適應中值；混合雜訊用 alpha 修剪平均。
- 週期性雜訊是頻譜中的幾個亮點；陷波濾波器去除它們，最佳陷波濾波則在局部自適應地調整減去的量。
- 逆濾波 G/H 會在 $|H|$ 小的地方放大雜訊。Wiener、受限最小平方與幾何平均濾波器都在分母加上一項，抑制這些頻率。
- CT 量測的是線積分（Radon 轉換）。依傅立葉切片定理，每個投影提供二維頻譜的一條切片，而濾波反投影（先斜坡濾波、再反投影）重建出影像。
- 深度學習用學來的先驗取代手工先驗，但最好的方法仍把前向模型保留在內（隨插即用、展開式網路）。

練習

1. 一張暗視野顯微影像的背景區域直方圖是單邊的：在 12 處陡然開始，右側拖著長尾；平均值為 20，變異數為 64（灰階單位）。你會先試本章的哪一種 PDF？參數是多少？

   提示

   陡然開始加上長尾，指向指數、Rayleigh 或 Erlang 家族。試試平移到 12 開始的指數分布：需要 $1/a = 20 - 12 = 8$ 且 $1/a^2 = 64$，兩者在 $a = 1/8$ 時同時成立。Rayleigh 則不一致：$b(4 - \pi)/4 = 64$ 得到 $b \approx 298$，它的起點會是 $20 - \sqrt{\pi b/4} \approx 4.7$，而不是 12。

2. 一張影像只受到鹽雜訊污染。先不做實驗，預測下列做法的結果：(a) 最大值濾波器、(b) 最小值濾波器、(c) $Q = 2$ 的反調和濾波器、(d) $Q = -2$ 的反調和濾波器。再用本章的程式驗證。

   提示

   鹽像素是視窗中最大的值。最大值濾波器會把它們擴散（更糟），最小值濾波器會去除它們（但會讓亮的細節變暗、變細）。正的 Q 加重大值的權重，所以 $Q = 2$ 讓鹽雜訊更糟；$Q = -2$ 則能去除它。

3. 證明常數 K 的 Wiener 濾波器在 $K = 0$ 時退化成逆濾波器，而在 $|H|^2 \ll K$ 的頻率上輸出約為 $(H^*/K)\,G$。這說明 Wiener 濾波如何對待被模糊摧毀的頻率？

   提示

   當 $|H|^2 \ll K$ 時分母約為 K。輸出因此很小，與 $|H|$ 成正比，所以濾波器不會試圖找回這些頻率。它體面地放棄，而不是放大雜訊。

4. 一張 256×256 的影像沿 x 方向有 8 個像素的等速運動模糊，y 方向沒有。頻率以整數（每張影像的週期數）表示，與上面的程式一致。利用運動模糊 OTF，找出所有 $H(u, v) = 0$ 的頻率 (u, v)。解釋為什麼沒有任何線性濾波器能還原這些頻率的細節，以及這在復原影像中會呈現什麼樣子。

   提示

   當 u 以每張影像的週期數表示時，a 是模糊長度佔影像尺寸的比例，所以 $a = 8/256 = 1/32$、$b = 0$。當 $u/32$ 為非零整數時 $H = 0$：$u = \pm 32, \pm 64, \pm 96, \pm 128$，對所有 v 都成立。由於 $G = HF + N$ 在這些頻率不含任何 F 的成分，任何線性濾波器在那裡只能輸出雜訊（或零）。預期會看到沿 x 方向、以該空間週期出現的淡淡振鈴。

5. 用上面的 CT 程式，分別以 180、60、20、10 個視角，以及 `ramp` 與 `hann` 濾波器重建。畫出 RMSE 對視角數的關係。接著在正弦圖加上高斯雜訊再重複一次。什麼時候較平滑的 `hann` 濾波器會勝出？

   提示

   沒有雜訊、視角很多時，`ramp` 應該勝出。雜訊增加後，斜坡濾波器在高頻放大雜訊，而在高頻衰減的 `hann` 就會勝出。這和逆濾波器對 Wiener 濾波器是同一種取捨。

6. 推導 $\min_F |G - HF|^2 + \mu |F - Z|^2$ 的逐頻率解 $\hat{F} = (H^{*}G + \mu Z)/(|H|^2 + \mu)$。當 $Z = 0$ 時，你得到本章的哪一個濾波器？

   提示

   對 F^{*} 微分：$-H^{*}(G - HF) + \mu(F - Z) = 0$。$Z = 0$ 時就是常數 $K = \mu$ 的 Wiener 濾波器。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 5. publisher page
2. H. Hwang and R. A. Haddad, “Adaptive median filters: new algorithms and results,” *IEEE Transactions on Image Processing*, vol. 4, no. 4, pp. 499–502, 1995. DOI
3. B. R. Hunt, “The application of constrained least squares estimation to image restoration by digital computer,” *IEEE Transactions on Computers*, vol. C-22, no. 9, pp. 805–812, 1973. DOI
4. A. C. Kak and M. Slaney, *Principles of Computerized Tomographic Imaging*, IEEE Press, 1988 (SIAM reprint 2001). free online edition
5. L. A. Shepp and B. F. Logan, “The Fourier reconstruction of a head section,” *IEEE Transactions on Nuclear Science*, vol. 21, no. 3, pp. 21–43, 1974. DOI
6. scikit-image developers, “Radon transform,” scikit-image documentation (gallery example). docs
7. A. Buades, B. Coll and J.-M. Morel, “A non-local algorithm for image denoising,” *IEEE CVPR*, vol. 2, pp. 60–65, 2005. DOI

8. K. Dabov, A. Foi, V. Katkovnik and K. Egiazarian, “Image denoising by sparse 3-D transform-domain collaborative filtering,” *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007. DOI
9. 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
10. 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
11. 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. arXiv
12. 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
13. G. Ongie, A. Jalal, C. A. Metzler, R. G. Baraniuk, A. G. Dimakis and R. Willett, “Deep learning techniques for inverse problems in imaging,” arXiv:2005.06001, 2020. arXiv
14. U. S. Kamilov, C. A. Bouman, G. T. Buzzard and B. Wohlberg, “Plug-and-play methods for integrating physical and learned models in computational imaging: Theory, algorithms, and applications,” *IEEE Signal Processing Magazine*, vol. 40, no. 1, pp. 85–97, 2023. arXiv
15. K. Zhang, Y. Li, W. Zuo, L. Zhang, L. Van Gool and R. Timofte, “Plug-and-play image restoration with deep denoiser prior,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6360–6376, 2022. arXiv
16. K. H. Jin, M. T. McCann, E. Froustey and M. Unser, “Deep convolutional neural network for inverse problems in imaging,” *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4509–4522, 2017. arXiv
17. J. Adler and O. Öktem, “Learned primal-dual reconstruction,” *IEEE Transactions on Medical Imaging*, 2018. arXiv
18. V. Monga, Y. Li and Y. C. Eldar, “Algorithm unrolling: Interpretable, efficient deep learning for signal and image processing,” arXiv:1912.10557, 2019. arXiv
19. G. Wang, J. C. Ye and B. De Man, “Deep learning for tomographic image reconstruction,” *Nature Machine Intelligence*, vol. 2, no. 12, pp. 737–748, 2020. DOI
20. A. Buades, B. Coll and J.-M. Morel, “A review of image denoising algorithms, with a new one,” *Multiscale Modeling & Simulation*, vol. 4, no. 2, pp. 490–530, 2005. DOI
21. L. I. Rudin, S. Osher and E. Fatemi, “Nonlinear total variation based noise removal algorithms,” *Physica D: Nonlinear Phenomena*, vol. 60, no. 1–4, pp. 259–268, 1992. DOI
22. 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
23. A. Buades, B. Coll and J.-M. Morel, “Non-Local Means Denoising,” *Image Processing On Line*, vol. 1, pp. 208–212, 2011. DOI ·article and implementation

IP 06・色彩影像處理

色彩如何量測與編碼、如何在 RGB、HSI 與 L*a*b* 之間轉換，以及如何對彩色影像做轉換、濾波與分割。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/color/

你將學到

- 為什麼對人類觀察者來說，三個數字就足以描述一種顏色；CIE 色度圖與裝置色域是什麼
- RGB、CMY(K)、HSI 與 CIE L*a*b* 之間的關係與精確轉換公式，以及 sRGB gamma 與 YCbCr 的定位
- 偽彩色如何把灰階影像變成好讀的彩色圖，以及為什麼 viridis 這類感知均勻色彩對應表勝過彩虹色
- 如何對全彩影像做轉換、白平衡、等化、平滑、銳化與分割，以及何時該逐通道處理、何時該當成向量處理
- 彩色影像中的雜訊有什麼特性，以及向量濾波器如何去除雜訊
- 近年研究（基於學習的色彩恆常性、色彩對應表設計、深度學習上色）為經典工具帶來了什麼

先看全貌

到目前為止，每個像素都只有一個數字：亮度。彩色影像的每個像素則有三個數字。色彩幫助我們找到物體（「那台紅色的車」）、區分材質，也是大家看照片時第一個察覺「不對勁」的地方。

> **用最簡單的話說**
>
> 想像你只有三支手電筒：一支紅的、一支綠的、一支藍的。把它們照在白牆上，調亮調暗，幾乎可以混出任何顏色：紅加綠變黃，三支一起變白。你的眼睛剛好反過來運作：眼睛裡有三種小小的色彩感測器，大腦把三個訊號混合成你看到的每一種顏色。一張彩色影像，其實就是替每一個點記下「三支手電筒各要多亮」。

依照教科書的編排 [1]，本章回答三個問題：怎麼精確描述色彩（色彩基礎與色彩模型）、怎麼用色彩呈現本身沒有顏色的資料（偽彩色），以及怎麼處理真正的彩色照片（轉換、濾波、分割、雜訊、壓縮）。第 3 到 5 章的灰階工具大多能沿用，但有一個反覆出現的關鍵：彩色像素是一個**向量**，把它當成三個互不相干的數字，有時沒問題，有時錯得離譜。

色彩基礎

白話版。光是許多波長的混合。眼睛不會量測整個混合，而是把它壓縮成三個數字，每種色彩感測器一個。兩種差很多的光，只要產生相同的三個數字，在我們看來就一模一樣。

光、視錐細胞與三色視覺

可見光的波長大約在 400 到 700 奈米之間。光源或物體表面可以用**光譜功率分布**（spectral power distribution）$S(\lambda)$ 描述：在每個波長 λ 發出或反射多少能量。

視網膜有三種**視錐細胞**（cones，對色彩敏感的感光細胞），分別對短、中、長波長最敏感。每一種視錐細胞 $c \in \{S, M, L\}$ 都有一條敏感度曲線 $s_c(\lambda)$，其反應為

$$r_c = \int S(\lambda)\, s_c(\lambda)\, d\lambda.$$

其中 $S(\lambda)$ 是入射光譜，$s_c(\lambda)$ 是第 c 種視錐細胞的敏感度。整條光譜被化約成三個數字，所以人類色覺是三**色視覺**（trichromatic）。直接的結果是**同色異譜**（metamerism）：兩條不同的光譜 $S_1 \neq S_2$ 只要產生相同的三個反應，看起來就是同一種顏色。這正是螢幕能運作的原因：螢幕從來沒有重現檸檬的光譜，只重現了它的三個視錐反應。

亮度、色相與飽和度

人用三個知覺屬性描述顏色：

- **亮度**（brightness）：看起來有多少光（對無色的灰來說，這就是全部）。
- **色相**（hue）：主波長，也就是我們用紅、橘、藍這些字叫出來的東西。
- **飽和度**（saturation）：顏色有多純，也就是摻了多少白。粉紅色是低飽和度的紅。

色相與飽和度合稱**色度**（chromaticity），所以顏色＝「色度＋亮度」，這正是 HSI 模型背後的想法。

CIE 三刺激值與色度圖

為了讓色彩可以量測，國際照明委員會（CIE）在 1931 年根據人類配色實驗，制定了三條**配色函數**（color-matching functions）$\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$。一條光譜的三**刺激值**（tristimulus values）為

$$X = \int S(\lambda)\, \bar{x}(\lambda)\, d\lambda, \qquad Y = \int S(\lambda)\, \bar{y}(\lambda)\, d\lambda, \qquad Z = \int S(\lambda)\, \bar{z}(\lambda)\, d\lambda.$$

X, Y, Z 依設計都是非負的，而 $\bar{y}$ 被選成符合人眼的亮度敏感度，所以 Y 就是**輝度**（luminance）。把色度與亮度分開的方法是正規化：

$$x = \frac{X}{X+Y+Z}, \qquad y = \frac{Y}{X+Y+Z}, \qquad z = \frac{Z}{X+Y+Z} = 1 - x - y.$$

x, y 是**色度座標**，z 是多餘的。把每一個純光譜色（單一波長）畫在 (x, y) 平面上，會得到一條馬蹄形曲線，稱為**光譜軌跡**（spectral locus）。連接兩端的直線是**紫線**（line of purples；紫色不是單一波長）。人類看得到的所有顏色都落在這個區域內。

色度圖好用，是因為兩個幾何性質：

1. **混色是線性的**。兩種光的任意混合落在兩者色度之間的線段上；三種光的混合落在它們圍成的三角形內。

2. **白色在中間**。參考白點，例如日光標準光源 D65，位於 $(x, y) \approx (0.3127, 0.3290)$，靠近中央。從白點往軌跡移動，飽和度增加；移動的方向決定色相。

色域

使用三個原色的顯示器，只能產生三原色圍成的三角形內的顏色。這個三角形就是裝置的**色域**（gamut）。由於光譜軌跡是彎的，沒有任何角落可實際製造的三角形能完全涵蓋它，所以每一台三原色顯示器都會漏掉某些可見顏色，尤其是高飽和的青色與綠色。

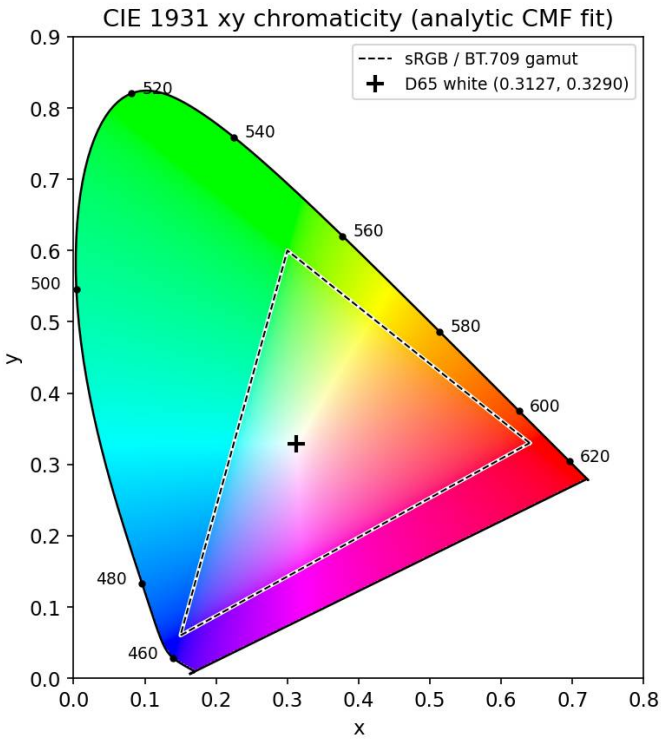


圖 *6.1* — 以 *Wyman* 等人 [2] 的解析配色函數擬合計算出的 *CIE 1931 xy* 色度圖（*430–650 nm*；擬合在兩端尾巴不夠準，故省略）。虛線三角形是 *sRGB / BT.709* 色域 [3], [4]；三角形外的顏色在你的螢幕上只能近似呈現。

僅線上版：圖 *6.1* 背後的配色函數解析擬合 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/color/notes/圖-61-背後的配色函數解析擬合/*

色彩模型

白話版。色彩模型就是顏色的座標系統。不同工作需要不同座標：螢幕用紅綠藍光思考，印表機用墨水思考，人則用「什麼顏色、多鮮豔、多亮」思考。

RGB

在 **RGB 模型**中，每種顏色是單位立方體裡的一個點 (R, G, B)。黑色是 $(0,0,0)$，白色是 $(1,1,1)$，主對角線上是各種灰。每通道 8 位元的影像有 $2^{24} \approx 1{,}670$ 萬種可能顏色。RGB 是**加法混色**：描述光的相加，也就是顯示器與相機感測器的運作方式。

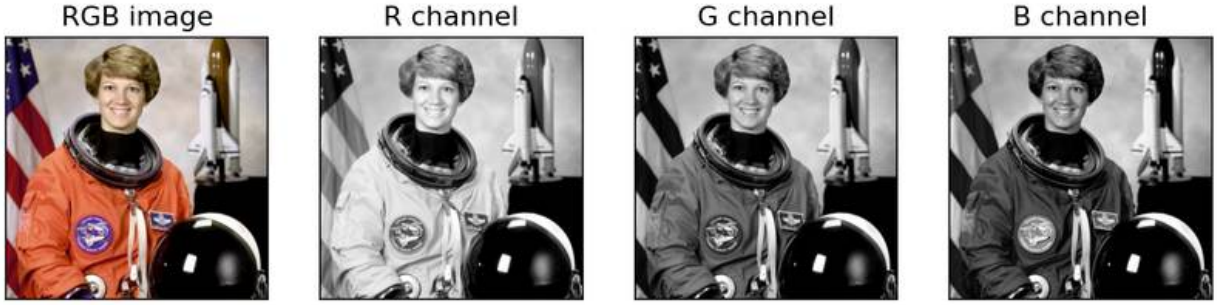


圖 *6.2 — RGB* 影像就是三張疊在一起的灰階影像。橘色太空衣在 R 通道很亮、在 B 通道很暗；太空衣的白色部分在三個通道都很亮。

單說「RGB」並不是精確的顏色：$(1,0,0)$ 在每台螢幕上都是不同的紅。**sRGB** 標準 [3] 規定了原色（與 ITU-R BT.709 [4] 相同的色度）、D65 白點與非線性編碼來解決這個問題。這個編碼常被籠統地稱為「gamma」，把儲存值 $C' \in [0,1]$ 轉成線性光 C：

$$C = \begin{cases} C'/12.92, & C' \le 0.04045, \\ \left(\dfrac{C' + 0.055}{1.055}\right)^{2.4}, & C' > 0.04045. \end{cases}$$

C' 是某個通道的儲存（編碼）值，C 是線性光的值。這條曲線把較多的碼值分配給暗部，因為人眼對暗部比較敏感。**實用原則：**模擬物理的運算（模糊、縮放、混光）在線性值上做才比較正確，所以在乎精度時要先解碼。

CMY 與 CMYK

印刷剛好相反。白紙上的墨水會吸收光，所以印刷是**減法混色**。青（cyan）、洋紅（magenta）、黃（yellow）分別吸收紅、綠、藍。數值在 $[0,1]$ 時：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

這裡 C, M, Y 是墨量（別和 CIE 的 Y 搞混）。真實的 C、M、Y 墨水等量混合會變成混濁的褐色而不是黑色，所以印表機加上第四種黑色墨水 K。一個簡單的轉換是

$$K = \min(C, M, Y), \qquad C_K = \frac{C - K}{1 - K}, \quad M_K = \frac{M - K}{1 - K}, \quad Y_K = \frac{Y - K}{1 - K} \quad (K < 1),$$

當 $K = 1$（純黑）時令 $C_K = M_K = Y_K = 0$。實際的印表機改用量測得到的色彩描述檔，但道理一樣：把顏色中共同的「灰」移到黑色墨水。

HSI

RGB 適合硬體，卻不符合人的直覺：沒有人會說一個顏色是「70% 紅、40% 綠、20% 藍」。**HSI 模型**（色相 hue、飽和度 saturation、強度 intensity）把亮度與色度分開，既貼近我們談論顏色的方式，也讓許多演算法變簡單。

從幾何上看，把 RGB 立方體用黑色那個角立起來，讓灰色對角線朝上。**強度**是沿這條軸的高度。用垂直於軸的平面切開立方體：**色相**是繞軸的角度（紅在 0°、綠在 120°、藍在 240°），**飽和度**是離軸的距離。

RGB 轉 HSI（$R, G, B \in [0,1]$）：

$$\theta = \cos^{-1}\left(\frac{\tfrac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right), \qquad H = \begin{cases} \theta, & B \le G \\ 360^\circ - \theta, & B > G \end{cases}$$

$$S = 1 - \frac{3\,\min(R,G,B)}{R+G+B}, \qquad I = \frac{R+G+B}{3}.$$

θ 是像素的色彩分量與紅色軸的夾角。H 是以度為單位的色相，$S \in [0,1]$ 是飽和度，$I \in [0,1]$ 是強度。程式裡有兩個邊界情況要小心：灰色像素（$R = G = B$）時 θ 的分母為零，色相**沒有定義**，依慣例設為 0；黑色（$I = 0$）時飽和度沒有定義，也設為 0。

HSI 轉 RGB 是上述公式的反運算。轉換是精確的，但要分扇區處理：用哪一組公式，取決於色相落在色環的哪三分之一（0°–120°、120°–240° 或 240°–360°），而與色相「相對」的那個通道永遠得到最小值 $I(1-S)$。

僅線上版：*從 HSI 轉回 RGB — https://blog.shyandram.dev/zh-tw/learn/image-processing/color/notes/從-hsi-轉回-rgb/*

```
import numpy as np

def rgb_to_hsi(rgb):
    """rgb: float array in [0, 1] with shape (..., 3). Returns H (degrees), S, I."""
    R, G, B = rgb[..., 0], rgb[..., 1], rgb[..., 2]
    num = 0.5 * ((R - G) + (R - B))
    den = np.sqrt((R - G) ** 2 + (R - B) * (G - B))
    theta = np.degrees(np.arccos(np.clip(num / (den + 1e-12), -1, 1)))
    H = np.where(B > G, 360.0 - theta, theta)
    I = rgb.mean(axis=-1)
    S = np.where(I > 0, 1 - rgb.min(axis=-1) / (I + 1e-12), 0.0)
    H = np.where(den > 1e-12, H, 0.0)      # gray pixels: hue undefined -> 0
    return H, S, I

prim = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1], [0.5, 0.5, 0.5]], float)
print(np.round(np.stack(rgb_to_hsi(prim), -1), 3))
# [[  0.   1.   0.333] [120.  1.  0.333] [240.  1.  0.333] [  0.  0.  0.5 ]]
```

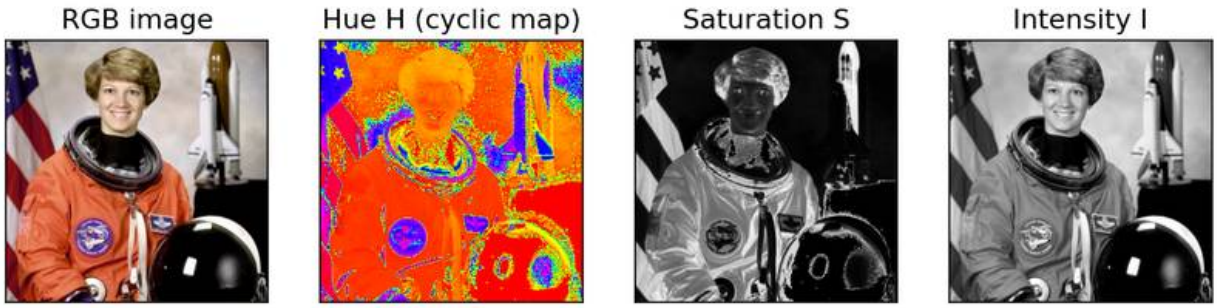


圖 *6.3* — *HSI* 分量。色相用循環色彩對應表顯示，因為 *0°* 與 *360°* 是同一個色相。在灰色區域（背景、白色太空衣）色相幾乎沒有定義，所以看起來很雜亂；只有橘色太空衣、國旗與徽章的飽和度高。

> **HSI、HSV、HSL 是表親，不是雙胞胎**
>
> 函式庫通常實作 HSV（value = $\max(R,G,B)$）或 HSL，而不是上面的 HSI 公式。三者都把色相分離出來，但飽和度與亮度的定義不同，數值不能互換。OpenCV 還把 8 位元的色相存成 $H/2$，好塞進 $[0, 179]$。

CIE L*a*b*

HSI 把亮度和色度分開了，但 HSI 中相同的步長看起來並不一樣大。**CIE 1976 L*a*b*** 色彩空間 [5] 的設計目標，是讓歐氏距離大致對應感知上的差異。它有三個軸：

- $L^* \in [0, 100]$ 是**明度**（lightness），從黑到白；
- a^* 從綠（負）到紅（正）；
- b^* 從藍（負）到黃（正）。

它由三刺激值 X, Y, Z 除以參考白點的三刺激值，再取立方根計算而得，立方根模仿人眼的壓縮式反應。色差 ΔE^*_{ab} 就是兩個 (L^*, a^*, b^*) 點之間的歐氏距離。寫程式時，`skimage.color.rgb2lab` 一行就能轉換 sRGB 影像。

僅線上版：*CIELAB 公式與從頭實作的轉換* — *https://blog.shyandram.dev/zh-tw/learn/image-processing/color/notes/cielab-公式與從頭實作的轉換/*

一段話看懂 YCbCr

影像與視訊編碼器會把**亮度訊號**（luma）Y' 和兩個**色差訊號**（chroma）分開，因為人眼對亮度細節的解析力遠高於色彩細節。ITU-R BT.601 [6] 對 gamma 編碼後的 $R', G', B' \in [0, 1]$ 定義

$$Y' = 0.299\,R' + 0.587\,G' + 0.114\,B',$$

$C_b \propto B' - Y'$、$C_r \propto R' - Y'$，再縮放並平移到 8 位元的「studio」範圍（Y' 在 $[16, 235]$，C_b, C_r 在 $[16, 240]$）。HDTV 使用 BT.709 [4]，其亮度權重不同（$0.2126, 0.7152, 0.0722$）。兩者搞混會造成輕微但看得出來的偏色。`skimage.color.rgb2ycbcr` 實作的是 BT.601 studio 範圍版本。

偽彩色影像處理

白話版。偽彩色（pseudocolor，又稱假色）就是幫灰階影像上色，讓細微的亮度差異變得容易看見。這些顏色不是「真的」，而是一種代碼，就像氣象圖上的顏色。

動機來自知覺：人能分辨的色彩組合遠多於灰階層次。熱像儀、X 光、深度圖或高程模型每個像素只有一個值，但用顏色呈現這個值，可以顯露在灰階中會被忽略的結構。

強度切片

最簡單的做法是把灰階範圍切成數段，每段給一個顏色。設門檻 $0 = l_0 < l_1 < \cdots < l_P = L - 1$ 與顏色 $c_1, \dots, c_P$：

$$g(x,y) = c_k \quad \text{if} \quad l_{k-1} \le f(x,y) < l_k.$$

f 是輸入灰階值，L 是灰階數，P 是分段數，g 是輸出顏色。若把 f 畫成 3-D 地形，每個門檻就是一個水平面切過「強度地形」，名稱由此而來。$P = 2$ 時就是用兩種顏色畫出的二值化。當分段本身有意義時（「溫度高於 40 °C」、「骨骼與軟組織」），強度切片（intensity slicing）最合適。

強度到色彩的轉換

更一般的做法是對灰階值套用三個獨立函數：

$$g_R(x,y) = T_R(f(x,y)), \qquad g_G(x,y) = T_G(f(x,y)), \qquad g_B(x,y) = T_B(f(x,y)).$$

T_R, T_G, T_B 把灰階值對應到紅、綠、藍輸出。強度切片是三者都是階梯函數的特例；平滑的函數則構成**連續色彩對應表**（colormap）。同樣的想法可推廣到多張輸入影像：對多光譜資料，把三個波段（例如近紅外、紅、綠）分別指定給顯示的三個通道，就得到假色合成影像。

感知均勻的色彩對應表

T_R, T_G, T_B 怎麼選，比看起來重要得多。常見的彩虹色對應表（`jet`）中段經過明亮的黃與青，兩端則變暗。它的**明度不是單調的**：資料上相等的步長在視覺上不相等，平坦區段看起來像邊緣，真正的邊緣反而可能消失。Kovesi [7] 指出，一些廣泛使用的色彩對應表有感知上的「平坦區」，可以藏住大到資料範圍十分之一的特徵，並主張色彩對應表應該設計成明度均勻變化。

viridis 由 van der Walt 與 Smith 設計，現在是 Matplotlib 的預設 [8]。它在 CAM02-UCS 色貌空間中被最佳化為感知均勻，轉成灰階列印時仍然均勻，對常見的色覺異常也易於辨讀。Crameri、Shephard 與 Heron [9] 在各科學領域提出同樣的主張：彩虹色與紅綠色對應表會扭曲資料，並把約 8% 有色覺異常的男性排除在外。

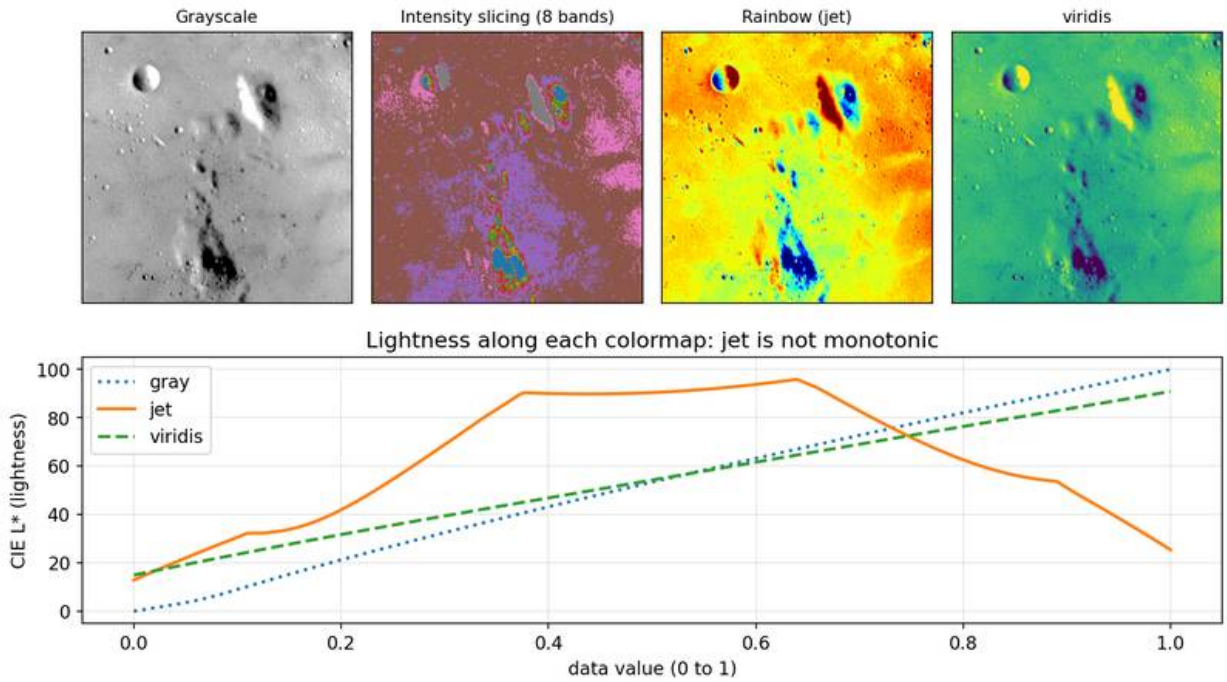


圖 *6.4* — 對比拉伸後的月面影像的偽彩色。上排：灰階、*8* 段強度切片、彩虹色 *jet*、*viridis*。下排：沿各色彩對應表的 *L** 明度。*viridis* 像灰階一樣穩定上升；*jet* 先升、持平、再下降，因而產生假邊緣。

```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data, util, color

g = util.img_as_float(data.moon())
# intensity slicing into 8 equal bands
bands = np.digitize(g, np.linspace(0, 1, 9)[1:-1])          # 0..7
palette = plt.get_cmap("tab10")(np.arange(8))[:, :3]
sliced = palette[bands]                                      # (H, W, 3)
# continuous intensity-to-color transformation
rgb = plt.get_cmap("viridis")(g)[..., :3]
# check a colormap's lightness profile
L = color.rgb2lab(plt.get_cmap("jet")(np.linspace(0, 1, 256))[None, :, :3])[0, :, 0]
print(np.all(np.diff(L) >= 0))                               # False: jet is not monotonic
```

經驗法則：有順序的資料用**循序型**（sequential）均勻對應表（例如 viridis）；有意義的中點（零、門檻）時用**發散型**（diverging）；角度與色相用**循環型**（cyclic）；只有無序類別才用**定性型**（qualitative）調色盤，例如強度切片。

全彩影像處理基礎

白話版。處理彩色影像有兩種方式：當成三張獨立的灰階影像，或把每個像素看成色彩空間中的小箭頭，直接處理箭頭。有時兩者結果相同，但常常不同。

一個全彩像素就是向量

$$\mathbf{c}(x,y) = \begin{bmatrix} c_R(x,y) \\ c_G(x,y) \\ c_B(x,y) \end{bmatrix} = \begin{bmatrix} R(x,y) \\ G(x,y) \\ B(x,y) \end{bmatrix}.$$

處理它有兩種方式：

1. **逐通道**（per-channel，componentwise）。對每個通道各自套用灰階方法，再疊回去。
2. **向量處理**（vector processing）。直接對向量 $\mathbf{c}(x,y)$ 運算，使用同時考慮三個分量的操作，例如向量距離、向量中值或彩色梯度。

逐通道處理要和向量處理結果相同，需要兩個條件：運算同時適用於純量與向量，而且對每個分量的作用相同、不會混在一起。**線性**鄰域運算兩者都滿足。對每個通道在視窗內取平均，等於對向量在視窗內取平均：

$$\frac{1}{|S_{xy}|}\sum_{(s,t)\in S_{xy}} \mathbf{c}(s,t) = \begin{bmatrix} \frac{1}{|S_{xy}|}\sum R(s,t) \\ \frac{1}{|S_{xy}|}\sum G(s,t) \\ \frac{1}{|S_{xy}|}\sum B(s,t) \end{bmatrix},$$

其中 S_{xy} 是以 (x,y) 為中心的鄰域，$|S_{xy}|$ 是其像素數。**非線性**運算會打破這個等價關係。逐通道中值可能從第一個像素取紅、從第二個取綠、從第三個取藍，組出鄰域裡根本不存在的顏色。門檻、邊緣偵測與直方圖等化也都是需要向量觀點的例子，後面各節會說明。

色彩轉換

白話版。色彩轉換就是一個只看單一像素、改變它顏色的配方：讓它變暗、翻成相反色、調暖一點，或者只有「夠橘」時才保留它。

數學表述

和第 3 章一樣，點轉換寫成 $g(x,y) = T[f(x,y)]$，只是現在 f 與 g 都是色彩向量。逐分量寫出來是

$$s_i = T_i(r_1, r_2, \ldots, r_n), \qquad i = 1, \ldots, n,$$

其中 r_i、s_i 是第 i 個輸入與輸出分量，n 是分量數（RGB 與 HSI 為 $n = 3$，CMYK 為 $n = 4$）。每個輸出都可以依賴所有輸入。

任何效果都能在任何模型中寫出來，但成本不同。把強度乘上 k，在 HSI 中只需一次乘法（$s_3 = k\, r_3$），在 RGB 中要三次（$s_i = k\, r_i$），在 CMY 中則變成仿射映射（$s_i = k\, r_i + (1-k)$）。選擇讓運算

最自然的模型，但別忘了轉換本身要花時間，也會在暗部或灰色像素放大雜訊，因為那裡的色相與飽和度不穩定。

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

補色

色環上位置相對的色相互為**補色**（complements）：紅與青、綠與洋紅、藍與黃。在 RGB 中，補色就是照相負片：

$$s_i = 1 - r_i, \qquad i = R, G, B,$$

也就是第 3 章灰階負片的彩色版。在 HSI 中，色相補色是 $H \mapsto (H + 180°) \bmod 360°$，但 RGB 負片同時反轉了強度，因此不能只靠色相算出來。補色有助於顯露暗部細節。

色彩切片

色彩切片（color slicing）是彩色版的強度切片：保留顏色接近某個**原型**（prototype）$\mathbf{a} = (a_R, a_G, a_B)$ 的像素，其餘推向中性色。「接近」常見的兩種定義：

$$\text{cube:}\quad s_i = \begin{cases} 0.5, & \text{if } |r_j - a_j| > W/2 \text{ for any } j \\ r_i, & \text{otherwise} \end{cases} \qquad \text{sphere:}\quad s_i = \begin{cases} 0.5, & \text{if } \|\mathbf{r} - \mathbf{a}\| > R_0 \\ r_i, & \text{otherwise} \end{cases}$$

W 是以 $\mathbf{a}$ 為中心的立方體寬度，R_0 是球的半徑，$\mathbf{r}$ 是輸入色彩向量，0.5 是中灰。球對各方向一視同仁；立方體計算較便宜，也方便逐通道推理。比較友善的變形是讓沒選中的像素保留自己的灰階值，而不是一片平灰，這樣仍看得到上下文。

```
import numpy as np
from skimage import data, util, color

img = util.img_as_float(data.astronaut())
a = np.array([0.85, 0.40, 0.20])            # prototype: the orange of the suit
d = np.linalg.norm(img - a, axis=-1)        # distance of every pixel to a
gray = color.gray2rgb(color.rgb2gray(img))
sliced = np.where((d <= 0.25)[..., None], img, gray)
print((d <= 0.25).mean())                   # fraction kept, about 0.165
```

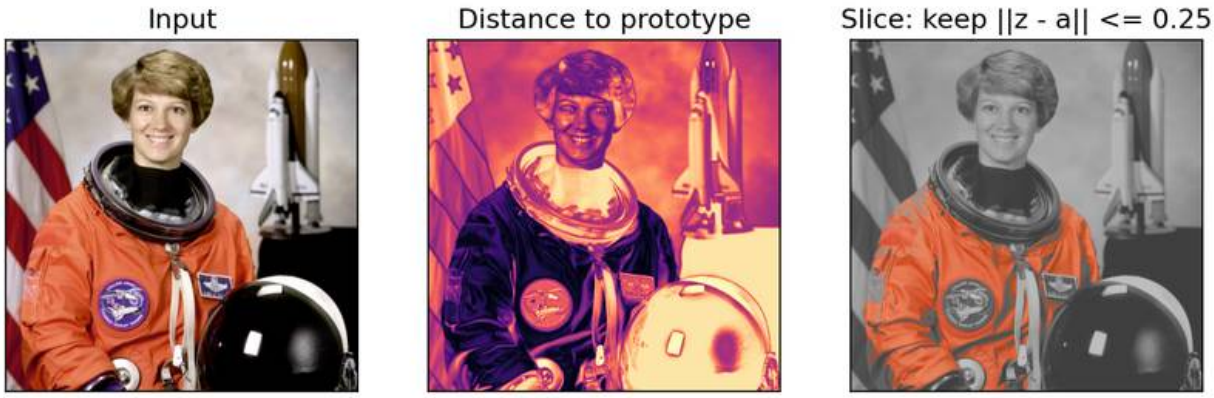


圖 *6.5* — 以太空衣的橘色為中心、半徑 *0.25* 的球做色彩切片。中：到原型的距離（越暗越近）。右：符合的像素保留顏色，其餘以灰階顯示。

色調與色彩校正

色調校正（tone correction）調整亮度與對比，最好只作用在類似強度的通道（I、V 或 L^*）上，以保留色相。偏亮（high-key）的影像需要拉開亮部的曲線，偏暗（low-key）的影像需要提亮暗部的曲線，平淡的影像則需要 S 形曲線。

色彩平衡（color balance）校正用來去除色偏：要減少某個顏色，就增加它的補色，或減少它在色環上的鄰居。青色太多？增加紅色，或同時減少藍與綠。場景中放一張灰卡就有精確的目標：校正後灰卡的 R、G、B 應該相等。

實務流程會透過校正過的色彩描述檔，把各裝置的顏色轉到與裝置無關的空間（例如 L*a*b*），讓顏色在螢幕與紙上看起來一致；超出某裝置色域（圖 6.1）的顏色必須往內映射。

白平衡

白平衡（white balance）是最常見的自動色彩校正。相機記錄的是表面顏色乘上光源顏色，所以同一面白牆在鎢絲燈下偏橘、在陰影中偏藍。我們的視覺系統大多能把光源的影響扣掉（**色彩恆常性**，color constancy），相機則必須用計算來做到。

標準模型是對角模型：光源顏色為 $\mathbf{e} = (e_R, e_G, e_B)$ 時，觀察到的像素為 $c_i = e_i \rho_i$，其中 ρ_i 是表面在白光下的反應。校正就是除以估計的光源，$\hat{\rho}_i = c_i / \hat{e}_i$（通常再縮放以維持整體亮度）。困難在於只靠一張影像估計 $\hat{\mathbf{e}}$。經典的估計法都對場景做了某種假設：

- **灰色世界**（gray world）[10]：場景的平均反射率是無彩色的，所以 $\hat{\mathbf{e}} \propto$ 影像的平均 RGB。
- **白色區塊**（white patch，max-RGB）[11], [14]：最亮的表面是白色，所以 $\hat{\mathbf{e}} \propto$ 各通道最大值（實務上用高百分位數，以忽略少數飽和像素）。
- **灰色邊緣**（gray edge）[11]：平均邊緣（影像導數）是無彩色的。van de Weijer、Gevers 與 Gijsenij 證明三者屬於同一個家族：

$$\left(\int \left| \frac{\partial^n \mathbf{c}_\sigma(\mathbf{x})}{\partial \mathbf{x}^n} \right|^p d\mathbf{x} \right)^{1/p} = k\,\hat{\mathbf{e}},$$

逐通道計算，其中 $\mathbf{c}_\sigma$ 是以尺度 σ 的高斯平滑過的影像，n 是導數階數，p 是 Minkowski 範數，k 是常數。$n=0, p=1$ 是灰色世界；$n=0, p=\infty$ 是 max-RGB；$n=1$ 是灰色邊緣。

```
import numpy as np
from skimage import data, util

def white_balance(img, method="gray-world", p=99.5):
    """Diagonal white balance. img: float RGB in [0, 1]."""
    px = img.reshape(-1, 3)
    e = px.mean(0) if method == "gray-world" else np.percentile(px, p, axis=0)
    gain = e.mean() / e                 # one gain per channel
    return np.clip(img * gain, 0, 1), e

img = util.img_as_float(data.astronaut()) * 0.8
biased = np.clip(img * [0.75, 0.95, 1.25], 0, 1)      # add a bluish cast
fixed, e = white_balance(biased, "white-patch")
print(e.round(3))                                      # [0.598 0.757 0.996]
```

拖動滑桿，比較偏藍的輸入與白色區塊法的校正結果。



偏藍



白色區塊法校正

加上人工藍色偏色的太空人影像，以及白色區塊法白平衡後的結果

這些假設都會在某些場景失敗。下方的貓照片真實平均色是橘褐色，灰色世界法把溫暖的毛色「校正」成灰，留下藍色調；白色區塊法則靠白色鬍鬚與高光表現好得多。每張結果上方標示的是估計光源與真實光源之間的角度誤差（angular error，這個領域的標準指標）。



圖 *6.6* — 在平均色不是灰色的影像上比較灰色世界法與白色區塊法。角度誤差來自我們自己的合成實驗（見圖表程式），並非基準測試結果。

彩色影像的直方圖處理

對 R、G、B 三個直方圖分別做等化，通常會產生錯誤的顏色，因為它改變了通道之間的比例，也就改變了色相。比較安全的做法是只等化強度通道、保留色度：

1. 轉到能分離亮度的模型（HSI、HSV 或 L*a*b*）。

2. 用第 3 章的方法等化 I（或 V、L^*）。
3. 轉回來。

```
import numpy as np
from skimage import data, util, exposure, color

img = util.img_as_float(data.astronaut())
per_channel = np.stack([exposure.equalize_hist(img[..., k]) for k in range(3)], -1)  # shifts
 ↪  hues
lab = color.rgb2lab(img)
lab[..., 0] = 100 * exposure.equalize_hist(lab[..., 0] / 100)                        #
 ↪  lightness only
better = np.clip(color.lab2rgb(lab), 0, 1)
```

色相保住了，但有些顏色可能看起來變淡，或在轉回時超出色域，因此有時需要裁切或稍微提高飽和度。區域直方圖方法（第 3 章）也能用同樣方式套用在 L^* 上。

彩色影像的平滑與銳化

白話版。要模糊或銳化彩色照片，可以對三個通道分別處理，因為這些是線性運算。不過只處理亮度，結果往往更乾淨。

平滑

彩色影像的鄰域平均為

$$\bar{\mathbf{c}}(x,y) = \frac{1}{K} \sum_{(s,t)\in S_{xy}} \mathbf{c}(s,t),$$

其中 S_{xy} 是鄰域，K 是鄰域大小。如前所述，這等於對每個通道分別平均，所以對 3 通道陣列呼叫 `cv2.GaussianBlur` 是正確的。

另一種做法是只平滑 HSI（或 L^*）影像的強度，保留色相與飽和度。兩者在邊緣附近不同：逐通道平滑會在紅綠交界產生中間色（一條橄欖色的線），只平滑強度則保留原本的色相、只柔化亮度。要選哪種取決於任務：去雜訊通常會對色度做比亮度更強的平滑，因為人眼對色度模糊較不敏感（這也正是 YCbCr 色度降取樣所利用的事實）。

銳化

向量的 Laplacian 就是各分量 Laplacian 組成的向量：

$$\nabla^2 \mathbf{c}(x,y) = \begin{bmatrix} \nabla^2 R(x,y) \\ \nabla^2 G(x,y) \\ \nabla^2 B(x,y) \end{bmatrix},$$

所以 Laplacian 銳化 $\mathbf{g} = \mathbf{c} - k\nabla^2\mathbf{c}$（強度 $k > 0$）可以逐通道進行。反銳化遮罩（unsharp masking，第 3 章）也一樣。實務上只銳化強度通道，可以避免高對比邊緣周圍出現彩色光暈（色邊），這種光暈來自三個通道過衝的程度不同。

在影像分割中使用色彩

白話版。色彩是把影像切成幾塊最容易的線索之一：「褐色的都是染色組織，藍色的都是背景」。問題在於怎麼量化「夠褐」。

在 HSI 空間中分割

如果目標是用色相描述的，HSI 是自然的選擇。典型流程：先對飽和度設門檻，做出「真的有顏色」的像素遮罩（低飽和度時色相沒有意義），把色相影像乘上遮罩，再對遮罩後的色相設門檻或看直方圖，挑出目標範圍。弱點在於色相的奇異點：接近灰色的像素中，雜訊會產生隨機色相，所以飽和度遮罩要先做。

在 RGB 向量空間中分割

RGB 向量常常效果更好，因為它避開了不穩定的色相計算。從目標中選一組樣本像素，算出平均色 $\mathbf{a}$，再依每個像素 $\mathbf{z}$ 到 $\mathbf{a}$ 的距離分類。**歐氏距離**為

$$D(\mathbf{z}, \mathbf{a}) = \|\mathbf{z} - \mathbf{a}\| = \left[(\mathbf{z} - \mathbf{a})^{\mathsf{T}}(\mathbf{z} - \mathbf{a})\right]^{1/2},$$

$D \le D_0$ 的像素標為目標，這個集合是半徑 D_0 的實心球。但樣本顏色通常在某些方向上分散得比較開（以染色組織來說，主要是由亮到暗的方向）。**Mahalanobis 距離**把這點納入考量：

$$D(\mathbf{z}, \mathbf{a}) = \left[(\mathbf{z} - \mathbf{a})^{\mathsf{T}}\mathbf{C}^{-1}(\mathbf{z} - \mathbf{a})\right]^{1/2},$$

其中 $\mathbf{C}$ 是樣本的 3×3 共變異數矩陣。此時 $D \le D_0$ 是順著資料方向的橢球。更便宜的近似是以 $\mathbf{a}$ 為中心的**外接盒**（bounding box），每個軸的半寬取該通道標準差的倍數；判斷是否落在盒內只需比較，不需平方或開根號。

```
import numpy as np
from skimage import data, util

img = util.img_as_float(data.immunohistochemistry())
samples = img[60:100, 60:100].reshape(-1, 3)        # a brown (stained) region
a, C = samples.mean(0), np.cov(samples.T)
z = img.reshape(-1, 3) - a
d_euc = np.sqrt((z ** 2).sum(1)).reshape(img.shape[:2])
d_mah = np.sqrt(np.einsum("ij,jk,ik->i", z, np.linalg.inv(C), z)).reshape(img.shape[:2])
mask_euc, mask_mah = d_euc <= 0.25, d_mah <= 3.0
```

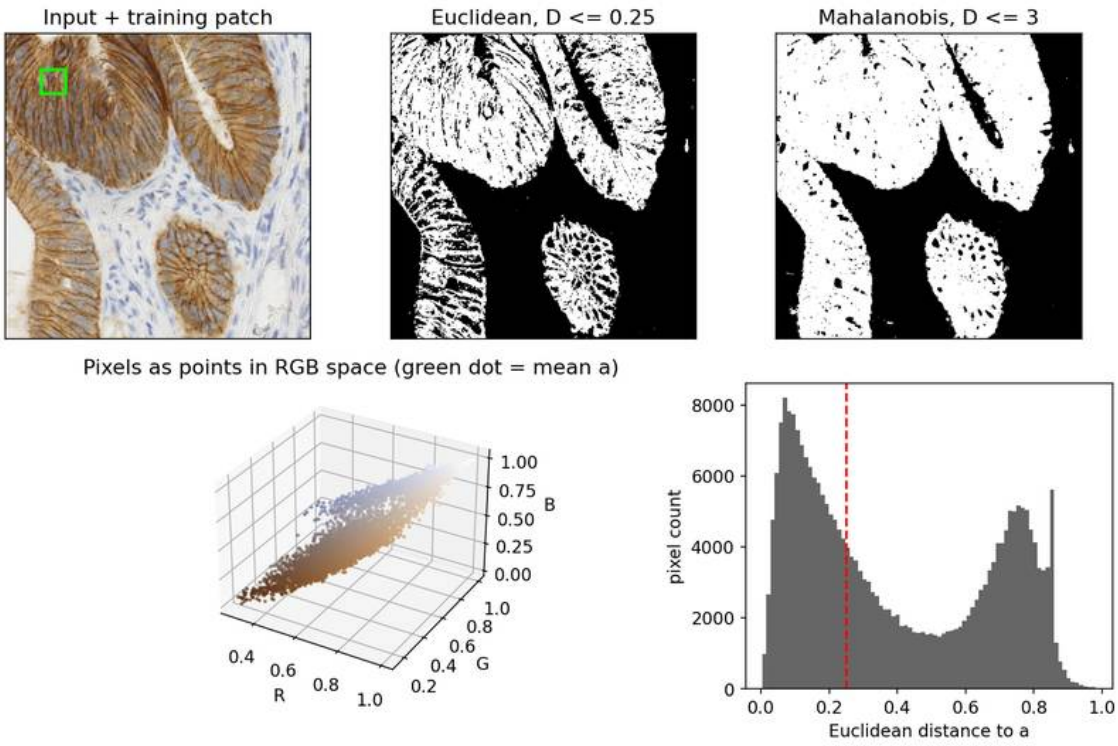


圖 *6.7* — 在 *RGB* 向量空間中分割。上排：染色組織樣本與訓練區塊（綠框）；半徑 *0.25* 的歐氏球；半徑 *3* 的 *Mahalanobis* 橢球。橢球順著拉長的褐色群集，把染色區域填得更完整。下排：像素形成從深褐到淡藍的細長雲團；距離直方圖呈雙峰，可用來選門檻。

彩色邊緣偵測

對每個通道各算梯度再相加是常見的捷徑，但那並不是向量影像的梯度。更糟的是，在灰階版本上做邊緣偵測，會漏掉**等亮度**（isoluminant）邊緣，也就是兩種亮度相同的顏色之間的邊界。Di Zenzo [13] 為多通道影像定義了正確的梯度。令 $\mathbf{u} = (\partial R/\partial x, \partial G/\partial x, \partial B/\partial x)$、$\mathbf{v} = (\partial R/\partial y, \partial G/\partial y, \partial B/\partial y)$，定義

$$g_{xx} = \mathbf{u}\cdot\mathbf{u}, \qquad g_{yy} = \mathbf{v}\cdot\mathbf{v}, \qquad g_{xy} = \mathbf{u}\cdot\mathbf{v}.$$

它們組成 2×2 結構張量 $\begin{bmatrix} g_{xx} & g_{xy} \\ g_{xy} & g_{yy} \end{bmatrix}$，總結了色彩向量在各個方向上變化得多快。邊緣強度是它最大特徵值的平方根，

$$\lambda_{\max} = \tfrac{1}{2}\Big[g_{xx} + g_{yy} + \sqrt{(g_{xx}-g_{yy})^2 + 4g_{xy}^2}\Big],$$

所以 $\sqrt{\lambda_{\max}}$ 就是色彩向量在所有方向中最大的變化率。和灰階影像的梯度不同，只要任何一個通道有變化，它就會有反應，所以看得到等亮度邊緣。

僅線上版：*Di Zenzo* 梯度的方向 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/color/notes/di-zenzo-梯度的方向/*

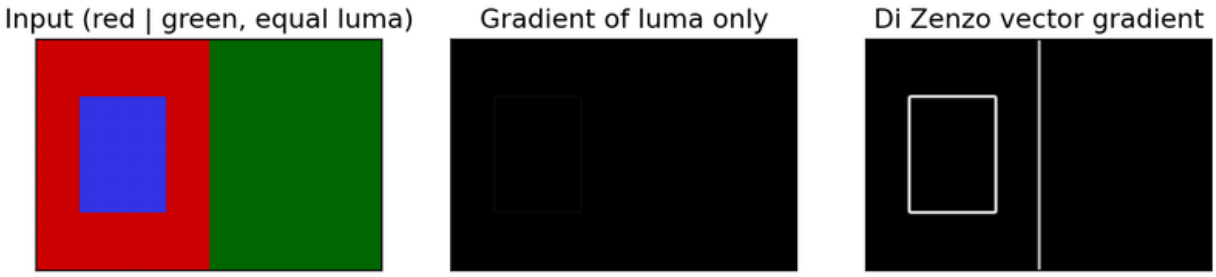


圖 *6.8* — 等亮度邊緣。紅綠兩半的 *BT.601* 亮度相同，所以亮度影像的梯度看不到兩者間的邊界（方塊也只隱約可見）。*Di Zenzo* 向量梯度兩者都找得到。

```
import numpy as np
from skimage import filters

def di_zenzo(img):
    """Edge strength of a float RGB image via the largest structure-tensor eigenvalue."""
    gx = np.stack([filters.sobel_v(img[..., k]) for k in range(3)], -1)   # d/dx
    gy = np.stack([filters.sobel_h(img[..., k]) for k in range(3)], -1)   # d/dy
    gxx, gyy, gxy = (gx * gx).sum(-1), (gy * gy).sum(-1), (gx * gy).sum(-1)
    return np.sqrt(0.5 * (gxx + gyy + np.sqrt((gxx - gyy) ** 2 + 4 * gxy ** 2)))
```

彩色影像中的雜訊

白話版。每個色彩通道都有自己的雜訊。把有雜訊的通道轉回色相與飽和度時，雜訊會混在一起，看起來可能糟得多，尤其是在暗部或灰色區域。

第 5 章的雜訊模型可套用到每個通道。相機中的雜訊主要產生於感測器，也就是在色彩重建之前。大多數感測器前面有一片彩色濾光陣列，每個感光點只量一種顏色，另外兩種靠內插補出來（**去馬賽克**，demosaicking）；內插會讓雜訊擴散並在通道之間產生相關。Lukac 等人 [12] 回顧了這整條處理鏈。

兩個觀察影響了實務做法：

- **色彩轉換會重新分配雜訊**。強度 $I = (R+G+B)/3$ 是三個通道的平均，所以每通道變異數為 σ^2 的獨立零均值雜訊，在 I 中變成 $\sigma^2/3$：強度比任何單一通道都乾淨。相反地，色相與飽和度是在灰色軸附近導數很大的非線性函數，因此會變得更雜。這就是為什麼去雜訊器常在亮度—色度空間中運作，並對色度做更強的平滑。
- **單一通道的脈衝雜訊是一個彩色斑點**。只出現在綠色通道的「鹽」雜訊會變成一個亮綠點。逐通道中值濾波器可以去除它，但在混合情況下可能產生鄰域裡沒有的顏色。

向量中值濾波器（vector median filter）避免了捏造顏色。在每個視窗中，它輸出與視窗內所有其他向量距離總和最小的那個輸入向量 [12]：

$$\mathbf{c}_{\mathrm{VM}} = \arg\min_{\mathbf{c}_i \in W} \sum_{\mathbf{c}_j \in W} \lVert \mathbf{c}_i - \mathbf{c}_j \rVert,$$

其中 W 是視窗內色彩向量的集合。輸出一定是輸入之一；對一維資料，它就退化成一般的中值。

```
import numpy as np
from skimage import data, util

def vector_median(img, k=3):
    r = k // 2
    H, W, _ = img.shape
    p = np.pad(img, ((r, r), (r, r), (0, 0)), mode="edge")
    nb = np.stack([p[i:i + H, j:j + W] for i in range(k) for j in range(k)], 2)  # (H, W,
     ↪  k*k, 3)
    D = np.linalg.norm(nb[:, :, :, None] - nb[:, :, None, :], axis=-1).sum(-1)    # (H, W,
     ↪  k*k)
    return np.take_along_axis(nb, D.argmin(-1)[..., None, None], 2)[:, :, 0]

img = util.img_as_float(data.astronaut())
rng = np.random.default_rng(0)
noisy = img.copy()
hit = rng.random(img.shape[:2]) < 0.05
noisy[hit] = rng.random((hit.sum(), 3))          # 5% random color impulses
out = vector_median(noisy)
psnr = lambda x: 10 * np.log10(1 / ((x - img) ** 2).mean())
print(round(psnr(noisy), 1), round(psnr(out), 1))  # about 20.4 -> 31.3 dB
```

這種向量化寫法每個像素使用的記憶體與 k^4 成正比；視窗大時請改用迴圈。

彩色影像壓縮

白話版。彩色影像的資料量是灰階的三倍，但其中很多是冗餘的：三個通道長得很像，而且人眼看不太到色彩細節。編碼器同時利用這兩點：

1. **去除通道間的相關性**。R、G、B 高度相關（圖 6.2 中三張影像非常相似）。轉成一個亮度加兩個色差，例如 YCbCr [6]，能把大部分能量集中在一個通道。
2. **色度降取樣**（chroma subsampling）。人眼對色彩細節的解析力不如亮度細節，所以色度通道可以只在水平方向用一半解析度（4:2:2），或在兩個方向都用一半（4:2:0）儲存，看起來幾乎沒有損失。光是 4:2:0 就讓原始資料減半：每個像素 $1 + 2 \cdot \frac{1}{4} = 1.5$ 個樣本，而不是 3 個。

之後每個通道再用第 8 章的轉換、量化與熵編碼方法壓縮。常見的色彩瑕疵有：銳利邊緣處的滲色（來自色度降取樣），以及高壓縮率下的塊狀色斑。

現代觀點

上面的經典工具至今仍在每一台相機與每一個影像函式庫裡運作。大約 2010 年後改變的，主要是參數怎麼決定：從固定的假設，變成從資料中學習。以下五篇回顧與綜述性質的論文描繪了這個領域的版圖。

僅線上版：現代觀點詳讀 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/color/papers/現代觀點詳讀/*

色彩恆常性：從假設到學習。 Gijsenij、Gevers 與 van de Weijer [14] 綜述了計算色彩恆常性，把方法分成三類：採用固定假設的方法（灰色世界、白色區塊、灰色邊緣）、色域式方法，以及用已知光源影像訓練的學習式方法。反覆出現的結論是，沒有任何單一固定假設適用所有場景。之後出現了學習式估計器：Fast Fourier Color Constancy（FFCC）[15] 能在手機上即時執行，FC4 [16] 讓每個影像區塊以學到的信心度替光源投票，Afifi 與 Brown [17] 則能在相機已經把影像渲染成非線性 sRGB 之後才校正白平衡。對角模型仍是骨幹，但光源估計現在是學出來的。

彩色向量濾波。 Lukac 等人 [12] 回顧了把彩色像素當成向量的濾波器。他們的結論與本章一致：逐分量處理可能產生色彩瑕疵，而向量順序統計量能保留輸入的顏色。在有訓練資料與運算資源時，學習式去雜訊器如今更強，但這些濾波器在講求速度與可預測性的場合仍是標準基準。

色彩對應表設計。 Kovesi [7]、viridis 系列 [8] 與 Crameri 等人 [9] 都遵循同一個原則：感知明度必須沿著對應表均勻變化，彩虹與紅綠色對應表應從軟體預設中消失。深度學習在這裡沒有改變任何事：問題在於人類知覺，答案是在感知色彩空間中量測的細心設計。

以深度學習上色。偽彩色用固定規則把灰階對應到顏色；上色（colorization）則試圖預測真實的顏色。Zhang、Isola 與 Efros [18] 把上色視為對量化色彩的分類問題，Anwar 等人 [19] 綜述了這個領域，DDColor [20]（ICCV 2023）則減少了顏色跨越物體邊界的滲色。兩個經典想法始終存在：在 L*a*b* 中運作（預測色度、保留給定的明度），以及把顏色當成一個分布，因為一件灰色襯衫原本可能是任何顏色。

沒有改變的部分。色度學（CIE XYZ、色度、色域）、編碼（sRGB、BT.601/709 YCbCr）與色差空間是標準，不是要學的東西。網路用 sRGB 資料訓練，也在這些空間中被評估。清楚知道你的張量處於哪個空間（編碼或線性、BT.601 或 BT.709、studio 或 full range），仍是色彩工作中最實用的技能之一。

重點整理

- 人類視覺是三色視覺，所以三個數字就能為人類觀察者描述顏色；反應相同的不同光譜（同色異譜）看起來一樣。
- CIE xy 色度圖呈現所有可見色度；三原色裝置只能涵蓋其中一個三角形（色域）。
- RGB 適合硬體，CMY(K) 適合墨水，HSI 把強度與色相、飽和度分開，L*a*b* 讓歐氏距離大致符合感知。注意灰色像素的色相沒有定義、sRGB gamma，以及 BT.601 與 BT.709 的差異。
- 偽彩色讓灰階資料更好讀；請用 viridis 這類感知均勻的色彩對應表，而非彩虹色。
- 線性運算（平滑、Laplacian 銳化）可以逐通道做；非線性運算（中值、直方圖等化、邊緣偵測、分割）應把像素當向量處理，或只處理強度通道。
- 白平衡就是除以估計的光源；灰色世界、白色區塊與灰色邊緣是同一個框架的特例，現代系統則用學習來估計。
- 用向量距離（歐氏、Mahalanobis、外接盒）分割色彩，用能看見等亮度邊界的 Di Zenzo 梯度偵測彩色邊緣。
- 強度比任何單一通道雜訊更少，色相與飽和度則更雜；向量中值濾波器能去除彩色脈衝雜訊而不捏造新顏色。

練習

1. 某像素 $(R, G, B) = (0.6, 0.6, 0.2)$。請手算它的 H、S、I。接著不計算，預測 B 增加到 0.3 時 H 會怎麼變。

   提示

   $I = 0.4667$。$\min = 0.2$，所以 $S = 1 - 0.2/0.4667 \approx 0.571$。計算 θ：分子 $= 0.5(0 + 0.4) = 0.2$，分母 $= \sqrt{0 + 0.4 \cdot 0.4} = 0.4$，所以 $\theta = \cos^{-1}(0.5) = 60°$；因為 $B \le G$，$H = 60°$（黃色）。B 增加到 0.3 時仍有 $R = G$，所以色相維持 $60°$，只有 S 下降。

2. 證明 RGB 負片 $s_i = 1 - r_i$ 會把 HSI 座標為 (H, S, I) 的像素，映射到強度為 $1 - I$、色相為 $H + 180°$ 的像素（非灰色像素）。飽和度會保持不變嗎？

   提示

   強度由線性性質直接得到。對色相而言，色彩分量 $\mathbf{c} - I\,\mathbf{1}$ 只是變號，相當於繞灰色軸旋轉 $180°$。飽和度是 $1 - \min/I$；取負片後最小值變成 $1 - \max(R, G, B)$，強度變成 $1 - I$，所以飽和度一般會改變。試試 $(0.9, 0.1, 0.1)$。

3. 你對一張藍天照片的 R、G、B 直方圖分別做等化。解釋為什麼天空可能變成灰青色，並提出兩種修正方式。

   提示

   等化會把每個通道拉滿整個範圍。原本幾乎一致偏高的通道（藍）相對其他通道會被壓低，而弱的通道被拉高，所以 R:G:B 比例改變。修正方式：只等化 I、V 或 L^*；或用平均直方圖算出一個共同的映射，套用到三個通道。

4. 在灰色世界／白色區塊的框架中，$n = 0$、$p = 2$ 會得到什麼估計法？把 `white_balance` 的想法用在貓的影像上，取 $p = 1, 2, 6, \infty$（∞ 用高百分位數），畫出角度誤差對 p 的關係。

   提示

   $p = 2$ 用每個通道的均方根值估計光源，介於平均值與最大值之間。以本章的合成偏色而言，誤差從約 $12°$（$p = 1$）降到約 $11°$（$p = 2$）、$7°$（$p = 6$）與 $4°$（$p = 20$）。在這張影像上誤差隨 p 增大而下降，因為明亮的白色細節比橘色的平均更能透露光源。在沒有白色物體的場景中，趨勢可能相反。

5. 設計一個在戶外照片中分割綠色植物、而且不受陰影影響的色彩分割器。你會用哪個空間與哪種距離？為什麼單純的 RGB 歐氏距離可能失敗？

   提示

   陰影會把三個通道等比例縮小，使像素沿著一條通過原點的直線移動，RGB 歐氏距離因此大幅改變。改用不受縮放影響的色度特徵（正規化的 $r = R/(R + G + B)$、$g = G/(R + G + B)$，或加上飽和度遮罩的色相，或 a^*, b^*），或用以陽光下與陰影中樣本訓練出共變異數的 Mahalanobis 距離。

6. 解釋為什麼向量中值濾波器永遠不會產生新顏色，並舉一個 3 個像素的一維例子，說明逐通道中值會產生新顏色。

提示

依定義，向量中值是視窗內的輸入向量之一。對三個像素 $(1,1,0)$、$(1,0,1)$、$(0,1,1)$（黃、洋紅、青），逐通道中值是 $(1,1,1)$，純白色，而視窗中根本沒有白色。

本章所有圖片都由 `scripts/figures/dip_ch06.py` 從 `skimage.data` 影像與合成陣列產生。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 6. publisher page
2. C. Wyman, P.-P. Sloan, and P. Shirley, "Simple Analytic Approximations to the CIE XYZ Color Matching Functions," *Journal of Computer Graphics Techniques*, vol. 2, no. 2, 2013. jcgt.org
3. IEC 61966-2-1:1999, "Multimedia systems and equipment – Colour measurement and management – Part 2-1: Colour management – Default RGB colour space – sRGB," IEC, 1999. IEC webstore
4. Recommendation ITU-R BT.709-6, "Parameter values for the HDTV standards for production and international programme exchange," ITU, 2015. ITU
5. ISO/CIE 11664-4:2019, "Colorimetry – Part 4: CIE 1976 L*a*b* colour space," ISO/CIE, 2019. standard page
6. Recommendation ITU-R BT.601-7, "Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios," ITU, 2011. ITU
7. P. Kovesi, "Good Colour Maps: How to Design Them," arXiv:1509.03700, 2015. arXiv
8. S. van der Walt and N. Smith, viridis colormap design notes (BIDS project page; presented at SciPy 2015), 2015. project page
9. F. Crameri, G. E. Shephard, and P. J. Heron, "The misuse of colour in science communication," *Nature Communications*, vol. 11, art. 5444, 2020. PMC
10. G. Buchsbaum, "A spatial processor model for object colour perception," *Journal of the Franklin Institute*, vol. 310, no. 1, pp. 1–26, 1980. DOI
11. J. van de Weijer, T. Gevers, and A. Gijsenij, "Edge-Based Color Constancy," *IEEE Transactions on Image Processing*, vol. 16, no. 9, pp. 2207–2214, 2007. DOI
12. R. Lukac, B. Smolka, K. Martin, K. N. Plataniotis, and A. N. Venetsanopoulos, "Vector Filtering for Color Imaging," *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 74–86, Jan. 2005. PDF
13. S. Di Zenzo, "A note on the gradient of a multi-image," *Computer Vision, Graphics, and Image Processing*, vol. 33, pp. 116–125, 1986. DOI
14. A. Gijsenij, T. Gevers, and J. van de Weijer, "Computational Color Constancy: Survey and Experiments," *IEEE Transactions on Image Processing*, vol. 20, no. 9, pp. 2475–2489, 2011. DOI
15. J. T. Barron and Y.-T. Tsai, "Fast Fourier Color Constancy," *CVPR*, 2017. arXiv
16. Y. Hu, B. Wang, and S. Lin, "FC4: Fully Convolutional Color Constancy with Confidence-Weighted Pooling," *CVPR*, 2017. paper
17. M. Afifi and M. S. Brown, "Deep White-Balance Editing," *CVPR*, 2020. arXiv

18. R. Zhang, P. Isola, and A. A. Efros, “Colorful Image Colorization,” arXiv:1603.08511, 2016. arXiv
19. S. Anwar, M. Tahir, C. Li, A. Mian, F. S. Khan, and A. W. Muzaffar, “Image Colorization: A Survey and Dataset,” arXiv:2008.10774, 2020. arXiv
20. X. Kang, T. Yang, W. Ouyang, P. Ren, L. Li, and X. Xie, “DDColor: Towards Photo-Realistic Image Colorization via Dual Decoders,” *ICCV*, 2023. arXiv

IP 07・小波與其他影像轉換

把每一種影像轉換都看成「換一組基底」，從 DCT、沃爾什–哈達瑪轉換一路走到哈爾轉換、多解析度分析與快速小波轉換。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/wavelets-transforms/

你將學到

- 為什麼所有線性影像轉換（傅立葉、餘弦、沃爾什–哈達瑪、哈爾、小波）其實是同一件事：用另一組積木來描述影像，這組積木叫做**基底**（basis）。
- 如何把轉換寫成兩次矩陣乘法，以及為什麼可分離（separable）的二維轉換很省計算。
- DCT、沃爾什–哈達瑪、斜轉換（slant）與哈爾轉換的基底影像長什麼樣子，以及 DCT 為何是 JPEG 的主力。
- 如何用時頻平面（time–frequency plane）理解傅立葉分析與小波分析的差別。
- 多解析度分析、尺度函數與小波函數如何導出快速小波轉換，也就是一個由兩個濾波器組成的濾波器組。
- 如何用 PyWavelets 做二維離散小波轉換，並應用在去雜訊、邊緣偵測與壓縮。

先看全貌

第 4 章把影像拆成許多正弦波的總和。這只是其中一種積木，並不是唯一的選擇。本章退一步問一個更一般的問題：給定任意一組積木，我要怎麼算出影像裡每塊積木各有多少？又該挑哪一組積木最適合手上的工作？

> **用最簡單的話說**
>
> 想像你在電話裡向朋友描述一個樂高模型。你可以一顆一顆凸點地講：「第 1 排第 1 格紅色、第 1 排第 2 格紅色……」。這一定行得通，但很慢。你也可以說：「這裡放一塊 2×4 的紅色積木，那裡放一塊 1×2 的藍色積木。」描述一下子變短了，因為積木的形狀剛好符合模型的形狀。
>
> 影像轉換就是在挑積木。像素網格用的是最小的 1×1 積木；傅立葉和餘弦轉換用的是條紋狀的積木；小波則準備了各種尺寸：大而模糊的拿來拼天空，小而銳利的拿來拼邊緣。積木越貼合畫面，需要的塊數就越少，壓縮、去雜訊和快速分析都是靠這一點。

「挑積木」的嚴謹說法是線性代數：內積、基底與基底轉換。這套工具一旦建立，本章的每一種轉換都只是一個不同的矩陣。小波會占最多篇幅，因為它是第一種在空間與頻率兩邊同時具有局部性的基底。這個特性讓小波成為 JPEG 2000 標準的核心 [15]，也是現代稀疏表示，以及近年部分神經網路層背後的重要想法。

預備知識

白話版：一個有 N 個取樣的訊號，就是 N 維空間中的一支箭頭（向量）。轉換就是量這支箭頭在 N 支參考箭頭上各投下多長的影子。如果參考箭頭兩兩垂直且長度為 1，影子就包含全部資訊，把影子加回去就能還原原本的箭頭。

內積空間

內積（inner product）$\langle \mathbf{f}, \mathbf{g} \rangle$ 接收兩個向量、回傳一個數，用來衡量兩者指向同一方向的程度。對長度為 N 的實數或複數向量：

$$\langle \mathbf{f}, \mathbf{g} \rangle = \sum_{x=0}^{N-1} f(x)\, g^{*}(x), \qquad \|\mathbf{f}\| = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle}.$$

其中 $f(x)$、$g(x)$ 是 $\mathbf{f}$、$\mathbf{g}$ 的第 x 個元素，g^{*} 是複數共軛（對實數資料沒有作用），$\|\mathbf{f}\|$ 是 $\mathbf{f}$ 的長度，也就是**範數**（norm）。對連續函數，總和改成積分：$\langle f, g \rangle = \int f(x) g^{*}(x)\, dx$。當兩個向量的內積為零時稱為**正交**（orthogonal）：它們彼此毫無共同成分。

正交歸一基底

若一組向量 $\{\mathbf{s}_0, \ldots, \mathbf{s}_{N-1}\}$ 每一個長度都是 1，而且兩兩正交，即 $\langle \mathbf{s}_u, \mathbf{s}_v \rangle = \delta(u - v)$（$u = v$ 時 δ 為 1，否則為 0），就稱為**正交歸一基底**（orthonormal basis）。此時任何向量都能乾淨地拆開：

$$t(u) = \langle \mathbf{f}, \mathbf{s}_u \rangle, \qquad \mathbf{f} = \sum_{u=0}^{N-1} t(u)\, \mathbf{s}_u.$$

$t(u)$ 稱為**轉換係數**。第一式是**分析**（正向）步驟，第二式是**合成**（反向）步驟。正交歸一基底還會保持能量：$\sum_x |f(x)|^2 = \sum_u |t(u)|^2$（帕塞瓦定理，Parseval's theorem），所以我們才能談「每個係數裝了多少能量」。

雙正交基底

有時最好用的積木並不互相垂直。這時需要兩組向量：一組 $\{\tilde{\mathbf{s}}_u\}$ 負責量測（分析），一組 $\{\mathbf{s}_u\}$ 負責重建（合成）。若滿足

$$\langle \mathbf{s}_u, \tilde{\mathbf{s}}_v \rangle = \delta(u - v), \qquad t(u) = \langle \mathbf{f}, \tilde{\mathbf{s}}_u \rangle, \qquad \mathbf{f} = \sum_u t(u)\, \mathbf{s}_u,$$

就稱為**雙正交**（biorthogonal）基底對，量測用的 $\tilde{\mathbf{s}}_u$ 稱為**對偶**（dual）基底。雙正交換來的是設計自由度：例如 JPEG 2000 採用雙正交小波濾波器，可以同時做到長度短且左右對稱，這個組合用正交小波很難達成 [15]。

框架

框架（frame）是一組向量數量多於維度的生成集合，所以帶有冗餘。它的定義性質是：量到的能量被固定的上下界夾住，

$$A\,\|\mathbf{f}\|^2 \;\le\; \sum_k |\langle \mathbf{f}, \mathbf{s}_k\rangle|^2 \;\le\; B\,\|\mathbf{f}\|^2, \qquad 0 < A \le B < \infty.$$

A、B 稱為**框架界**（frame bounds）。若 $A = B$，則稱為**緊框架**（tight frame），重建公式就是 $\mathbf{f} = \frac{1}{A}\sum_k \langle \mathbf{f}, \mathbf{s}_k\rangle \mathbf{s}_k$。一個經典的例子：平面上三支夾角 120° 的單位箭頭。它們能張成整個平面，任兩支都不正交，卻構成 $A = B = 3/2$ 的緊框架。當你需要穩健性或平移不變性時，多花一些係數換冗餘是值得的，例如不抽樣小波轉換（undecimated wavelet transform），以及「現代觀點」會提到的散射轉換。

```
import numpy as np

v = np.array([3.0, 1.0, 2.0])
e = np.eye(3)                          # standard orthonormal basis
c = e @ v                              # coefficients = inner products <v, e_k>
print(c, np.allclose(c @ e, v))        # synthesis: sum_k c_k e_k

# A biorthogonal pair in 2-D: analysis rows A, synthesis columns B = A^{-1}
A = np.array([[1.0, 0.0], [1.0, 1.0]])
B = np.linalg.inv(A)
x = np.array([2.0, 5.0])
print(A @ x, B @ (A @ x))              # [2. 7.] -> back to [2. 5.]
```

以矩陣表示的轉換

白話版：把量測用的向量疊成矩陣的每一列。訊號乘上這個矩陣，就一次量完所有係數；再乘上反矩陣，就把訊號拼回來。

對長度為 N 的一維訊號 $\mathbf{f}$，把基底向量 $\mathbf{s}_u^{*T}$ 依序放進 $N \times N$ **轉換矩陣** $\mathbf{A}$ 的各列，則

$$\mathbf{t} = \mathbf{A}\,\mathbf{f}, \qquad \mathbf{f} = \mathbf{A}^{-1}\mathbf{t}.$$

若基底是正交歸一的，$\mathbf{A}$ 就是**么正矩陣**（unitary，$\mathbf{A}^{-1} = \mathbf{A}^{*T}$；實數情形稱為**正交矩陣**，$\mathbf{A}^{-1} = \mathbf{A}^{T}$），反轉換幾乎不用成本。

對 $M \times N$ 的影像 $\mathbf{F}$，一般的二維線性轉換需要 $MN \times MN$ 的巨大矩陣。實務上幾乎所有影像轉換都是**可分離的**：先對每一行做一維轉換，再對每一列做一維轉換。寫成矩陣：

$$\mathbf{T} = \mathbf{A}\,\mathbf{F}\,\mathbf{B}^{T}, \qquad \mathbf{F} = \mathbf{A}^{-1}\,\mathbf{T}\,(\mathbf{B}^{T})^{-1},$$

其中 $\mathbf{A}$（$M \times M$）作用在行上，$\mathbf{B}$（$N \times N$）作用在列上；方形影像且兩方向使用同一種轉換時 $\mathbf{B} = \mathbf{A}$。可分離性把直接計算 $N \times N$ 轉換的成本從 $O(N^4)$ 降到 $O(N^3)$，而快速演算法（FFT、快速 DCT、快速小波轉換）還能再進一步降低。

```
import numpy as np
```

```
from scipy.fft import dct, dctn
from skimage import data, img_as_float

N = 8
A = dct(np.eye(N), norm="ortho", axis=0)  # row u = u-th cosine basis vector

print(np.allclose(A @ A.T, np.eye(N)))    # orthonormal -> True

F = img_as_float(data.camera())[200:208, 200:208]
T = A @ F @ A.T                           # forward 2-D transform
print(np.allclose(T, dctn(F, norm="ortho")))   # same as scipy -> True
print(np.allclose(A.T @ T @ A, F))             # inverse -> True
```

相關運算

白話版：每個係數都在回答一個問題：「我的影像有多像這個圖樣？」這個問題本身就是相關（correlation）運算。

把分析公式和第 3 章的互相關比一比。係數

$$t(u) = \langle \mathbf{f}, \mathbf{s}_u \rangle = \sum_x f(x)\, s_u^*(x)$$

正是 f 與基底函數 s_u 在位移為零時的相關值。$|t(u)|$ 大代表 f 和 s_u 很像；$t(u) = 0$ 代表 f 完全不含 s_u 的成分。由此得到兩個結論：

- **轉換就是樣板比對**。每個基底函數都是一個樣板，轉換一次替所有樣板打分數。傅立葉轉換替正弦波打分數，哈爾轉換替階梯狀圖樣打分數。
- **小波是在許多位移與尺度上做相關**。小波基底包含同一個圖樣的各種平移與伸縮版本。和某一尺度的所有平移版本做相關，就是一次濾波，所以本章後面的小波轉換會變成一組濾波器。

時頻平面上的基底函數

白話版：有些積木能告訴你事情何時發生，有些能告訴你它的音高，但沒有一塊能把兩者都說得完美。時頻平面就是描繪這種取捨的地圖。

對一個能量為 1 的基底函數 $s(x)$，用 $|s(x)|^2$ 與 $|S(\omega)|^2$ 的標準差定義它在時間（空間）與頻率上的展開程度：

$$\sigma_x^2 = \int (x - \bar{x})^2 |s(x)|^2 \, dx, \qquad \sigma_\omega^2 = \frac{1}{2\pi} \int (\omega - \bar{\omega})^2 |S(\omega)|^2 \, d\omega,$$

其中 $S(\omega)$ 是 s 的傅立葉轉換，$\bar{x}$、$\bar{\omega}$ 分別是兩個能量分布的質心。訊號版的**海森堡測不準原理**（Heisenberg uncertainty principle）指出

$$\sigma_x \, \sigma_\omega \;\geq\; \frac{1}{2},$$

只有高斯形狀的函數能取到等號。把每個基底函數畫成一個以 $(\bar{x}, \bar{\omega})$ 為中心、寬 σ_x、高 σ_ω 的矩形，稱為**海森堡盒**（Heisenberg box）或時頻磚。磚的面積不能小於某個下限，但形狀可以自由

選擇。

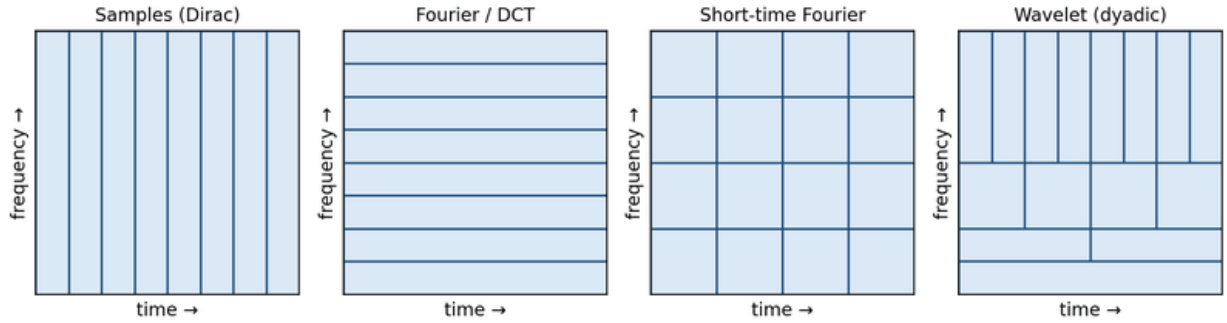


圖 *7.1* — 四種基底如何鋪滿時頻平面，每塊磚的面積都相同。取樣在時間上最精確；傅立葉／*DCT* 基底在頻率上最精確；短時傅立葉轉換對所有頻率採用同一種折衷；小波則依頻率調整磚的形狀。

圖 7.1 呈現四種標準情形：

- **取樣（單位基底）：**位置完全精確，沒有任何頻率資訊。
- **傅立葉或 DCT：**頻率完全精確，沒有位置資訊。一條銳利的邊緣會擴散到所有係數上。
- **短時傅立葉轉換**（short-time Fourier transform, STFT）：把訊號切成一個個窗，各自做傅立葉轉換。每塊磚形狀相同，一次決定就套用到所有頻率 [2]。
- **小波：**高頻的磚在時間上窄、頻率上高；低頻的磚則寬而矮。這很符合自然影像：邊緣是短暫的高頻事件，平緩的明暗變化則是緩慢的低頻成分。

基底影像

白話版：二維轉換的每個係數都對應一張小圖。影像就是這些小圖的加權總和，係數就是權重。

對矩陣 $\mathbf{A}$（第 u 列為 $\mathbf{a}_u^T$）定義的可分離轉換，可以把反轉換改寫成外積的總和：

$$\mathbf{F} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u,v)\, \mathbf{S}_{u,v}, \qquad \mathbf{S}_{u,v} = \mathbf{a}_u^* \mathbf{a}_v^{*T}.$$

每個 $N \times N$ 矩陣 $\mathbf{S}_{u,v}$ 就是一張**基底影像**（basis image）。$T(u,v)$ 告訴你要加入多少第 (u,v) 張基底影像；u 對應垂直方向的圖樣，v 對應水平方向。把 N^2 張基底影像排在一起，是理解一種轉換「看到什麼」最快的方法。

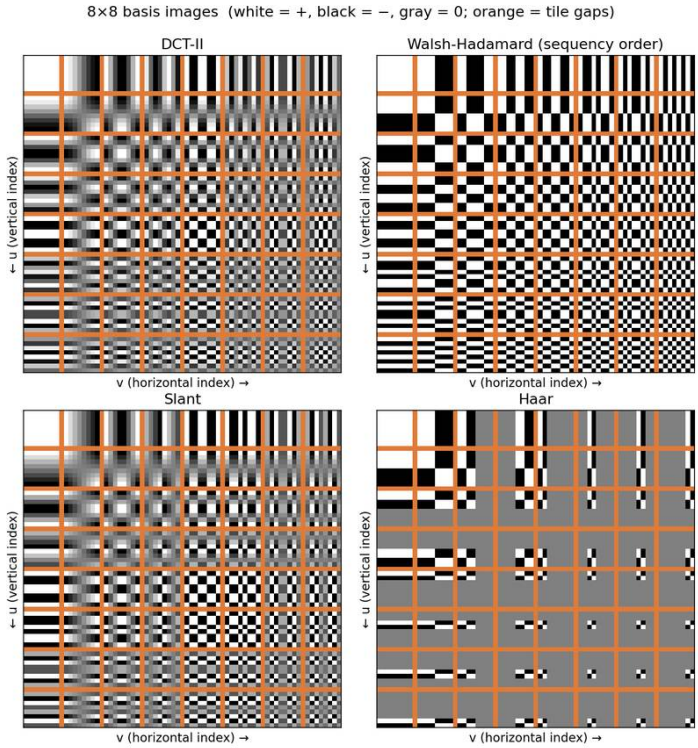


圖 7.2 — 四種轉換的 *8×8* 基底影像，向下代表垂直頻率（序率）增加，向右代表水平頻率增加；每格縮放到 $[-1, 1]$。注意哈爾基底影像和其他三種不同，大部分區域為零（灰色）：它們是局部的。

四種轉換左上角的基底影像都是常數，它量的是平均亮度，稱為 **DC** 係數。越往右、往下，圖樣振盪越快。DCT 平滑地振盪，沃爾什–哈達瑪在 ± 1 之間突然切換，斜轉換多了線性斜坡，哈爾則把每個圖樣限制在格子中的一小塊區域。

與傅立葉相關的轉換

白話版：傅立葉家族用波當積木。其中餘弦轉換在區塊邊界的表現最好，所以你手機裡的 JPEG 檔用的就是它。

以矩陣看離散傅立葉轉換

第 4 章的一維 DFT 是一個矩陣轉換：

$$A_{u,x} = \frac{1}{\sqrt{N}} e^{-j2\pi ux/N}, \qquad u, x = 0, \dots, N-1.$$

$A_{u,x}$ 是第 u 列第 x 行的元素，$j = \sqrt{-1}$。乘上 $1/\sqrt{N}$ 後矩陣是么正的。對壓縮來說它有兩個缺點：係數是複數，而且 DFT 默默把訊號當成週期性的。若區塊左右邊界的值不同，週期延伸就會出現跳躍，而跳躍需要大量高頻係數來表示。

離散餘弦轉換

（第二型）**離散餘弦轉換**（discrete cosine transform, DCT）[3] 使用實數餘弦：

$$t(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)\,u\,\pi}{2N}\right], \qquad \alpha(0) = \sqrt{\tfrac{1}{N}},\ \alpha(u) = \sqrt{\tfrac{2}{N}}\ (u \geq 1).$$

u 是頻率索引，$\alpha(u)$ 讓基底正交歸一。DCT 等於（差一個縮放）把訊號鏡射成長度 $2N$ 的偶對稱延伸後再做 DFT。鏡射消除了邊界跳躍，能量因此集中在少數低頻係數。這種**能量集中**（energy compaction）特性，正是 JPEG 標準對 8×8 影像區塊做 DCT 的原因 [4]。Ahmed、Natarajan 與 Rao 在 1974 年提出 DCT，並指出它在維納濾波與率失真（rate–distortion）準則下的表現，與統計上最佳的 Karhunen–Loève 轉換非常接近 [3]。

僅線上版：離散正弦轉換 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/wavelets-transforms/notes/離散正弦轉換/*

沃爾什–哈達瑪轉換

白話版：每個基底向量都只由 $+1$ 和 -1 組成。這樣轉換完全不需要乘法，只要加減。

$N = 2^n$ 階的哈達瑪矩陣以遞迴方式建構：

$$\mathbf{H}_1 = [\,1\,], \qquad \mathbf{H}_{2N} = \begin{bmatrix} \mathbf{H}_N & \mathbf{H}_N \\ \mathbf{H}_N & -\mathbf{H}_N \end{bmatrix}, \qquad \mathbf{A}_{\mathrm{WHT}} = \frac{1}{\sqrt{N}}\,\mathbf{H}_N.$$

$\mathbf{H}_N$ 的每個元素都是 ± 1，各列互相正交，因此 $\mathbf{A}_{\mathrm{WHT}}$ 是正交歸一（而且對稱）的。$\mathbf{H}_N$ 的列天生是**自然順序**（哈達瑪順序）。做分析時，改依**序率**（sequency，一列中正負號改變的次數，扮演頻率的角色）排序會更好讀。依序率排列的列稱為沃爾什函數，圖 7.2 就是它們的二維基底影像。

沃爾什–哈達瑪轉換（Walsh–Hadamard transform, WHT）因為計算便宜，很早就被用於影像編碼 [5]。對自然影像而言，它的能量集中能力不如 DCT，因為方波基底不擅長逼近平滑的明暗變化（見後面的圖 7.7）。在講求速度或硬體簡單的場合，以及作為快速隨機化演算法的元件時，它仍然很有用。

```
import numpy as np
from scipy.linalg import hadamard

H = hadamard(8)                                  # natural (Hadamard) order
changes = (np.diff(np.sign(H), axis=1) != 0).sum(1)
W = H[np.argsort(changes)] / np.sqrt(8)          # sequency order
print((np.diff(np.sign(W), axis=1) != 0).sum(1))   # [0 1 2 3 4 5 6 7]

x = np.array([10, 12, 11, 13, 40, 42, 41, 43], float)
print(np.round(W @ x, 2))
```

這個測試訊號是一個帶有小漣漪的階梯。它的 WHT 幾乎全落在係數 0（平均值）與係數 1（恰好一次正負號變化，對應那個階梯）；其餘只剩幾個小數值。

斜轉換

白話版：影像中很多區域不是平的，而是逐漸變亮或變暗。斜轉換準備了一塊「斜坡」積木，讓這種區域只需要一兩個係數。

Pratt、Chen 與 Welch 為影像編碼設計了**斜轉換**（slant transform）[6]。它和 WHT 一樣是正交歸一、遞迴建構的：每個尺寸都由兩份半尺寸的矩陣組成，再經一個稀疏矩陣混合；這個矩陣的兩個常數經過挑選，讓第二個基底向量成為等差遞減的階梯（「斜」向量），同時保持矩陣正交歸一。圖 7.2 中，斜轉換第一列與第一行的基底影像都是看得見的斜坡。斜轉換有 $O(N \log_2 N)$ 的快速演算法，但實務上同樣能處理斜坡的 DCT 取代了它。

僅線上版：建構斜轉換矩陣 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/wavelets-transforms/notes/建構斜轉換矩陣/*

哈爾轉換

白話版：把相鄰的兩個值拿來，記下它們的平均和差。再對平均值重複同樣的步驟。這就是哈爾轉換，也是最簡單的小波。

哈爾轉換（Haar transform）[1] 的基底函數要不是常數，就是一個「先上後下」的階梯，具有不同的寬度與位置。對 $N = 2^n$ 個取樣，用尺度 p 與位置 q 來編號，$k \geq 1$ 時 $k = 2^p + q$。不計歸一化，第 k 個基底向量在區間 $\left[\frac{q}{2^p}, \frac{q+1}{2^p}\right)$ 的前半段為 $+1$、後半段為 -1，其餘為 0（位置以訊號長度的比例表示）；$k = 0$ 的向量是常數。$N = 4$ 時：

$$\mathbf{A}_{\text{Haar}} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}.$$

第 0、1 列看整段訊號；第 2、3 列各自只看一半。這是它和前面所有轉換最關鍵的不同：**哈爾基底函數是局部的**。邊緣這類局部事件只會改變覆蓋到它的少數幾個哈爾係數。在時頻圖上，哈爾正好對應圖 7.1 的二進位（dyadic）鋪法。它是通往小波轉換的橋梁：以哈爾小波做的離散小波轉換，就是哈爾轉換。

小波轉換

白話版：用許多種縮放倍率看同一張影像。每往下一層，保留一張更模糊的版本，並只記下模糊時被抹掉的細節。模糊版本疊成一座金字塔，而那些細節就是小波係數。

多解析度分析

多解析度分析（multiresolution analysis, MRA）由 Mallat 形式化 [7]，把上面的想法變成一組基底。先取一個**尺度函數**（scaling function）$\varphi(x)$，做出平移與伸縮的版本：

$$\varphi_{j,k}(x) = 2^{j/2}\,\varphi(2^j x - k),$$

其中 j 是**尺度**（j 越大，函數越窄、細節越精細），k 是整數平移量，$2^{j/2}$ 讓能量維持為 1。令 V_j 為 $\{\varphi_{j,k}\}_k$ 所張成的空間，也就是在解析度 j 下能表示的所有訊號。MRA 要求四件事：

1. $\varphi_{0,k}$ 彼此正交歸一（或至少是 V_0 的穩定基底）。
2. 空間是巢狀的，$\cdots \subset V_{-1} \subset V_0 \subset V_1 \subset \cdots$：粗尺度看得到的東西，細尺度也看得到。
3. 所有 V_j 唯一的共同元素是 $f(x) = 0$。
4. 當 $j \to \infty$，任何平方可積函數都能被逼近到任意精確。

因為 $V_0 \subset V_1$，尺度函數本身必須能由更細的版本組合出來，於是得到**精細化方程式**（refinement equation，亦稱 dilation equation）：

$$\varphi(x) = \sum_{n} h_\varphi(n)\,\sqrt{2}\,\varphi(2x - n),$$

係數 $h_\varphi(n)$ 稱為**尺度函數係數**，稍後它會變成低通濾波器。

小波函數

從 V_{j+1} 降到 V_j 時遺失的細節，落在互補空間 W_j 中，使得 $V_{j+1} = V_j \oplus W_j$（$\oplus$ 表示 V_{j+1} 中每個元素都能唯一拆成 V_j 的一部分加上 W_j 的一部分；對正交小波，兩部分互相正交）。W_j 由**小波函數**（wavelet function）張成：

$$\psi_{j,k}(x) = 2^{j/2}\,\psi(2^j x - k), \qquad \psi(x) = \sum_{n} h_\psi(n)\,\sqrt{2}\,\varphi(2x - n).$$

對正交小波，**小波函數係數**可由尺度係數經調變與時間反轉得到：$h_\psi(n) = (-1)^n\,h_\varphi(1 - n)$，它們構成高通濾波器。一再重複這個拆分，得到

$$L^2(\mathbb{R}) = V_{j_0} \oplus W_{j_0} \oplus W_{j_0+1} \oplus \cdots,$$

其中 $L^2(\mathbb{R})$ 是所有有限能量函數構成的空間：一個粗略的近似，加上每個更細尺度的細節。

哈爾小波的 $h_\varphi = [\tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}]$、$h_\psi = [\tfrac{1}{\sqrt{2}}, -\tfrac{1}{\sqrt{2}}]$：就是平均與差。Daubechies 展示了如何建構具有**緊支撐**（compact support，即有限長度濾波器）且平滑度可調的正交小波，其正則性隨濾波器長度線性增加 [8]。PyWavelets [10] 中的 `dbN` 家族就是這些小波，`N` 是**消失矩**（vanishing moments）的個數：對 $m = 0, \dots, N-1$ 都有 $\int x^m \psi(x)\,dx = 0$。具有 N 個消失矩的小波對次數低於 N 的多項式完全「視而不見」，因此平滑區域的細節係數幾乎為零。

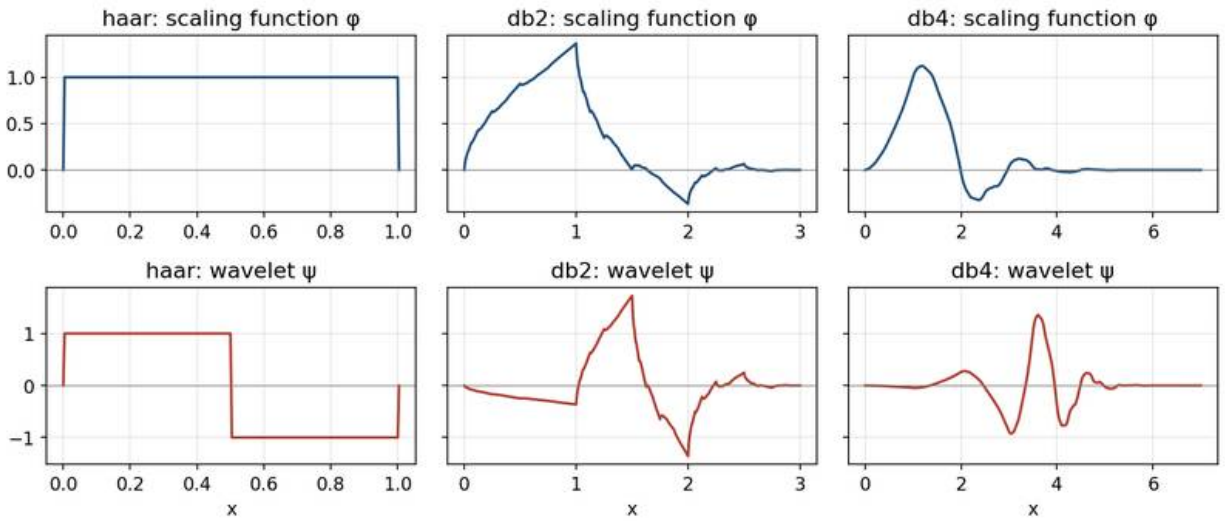


圖 7.3 — 以 *PyWavelets* 的級聯演算法（*cascade algorithm*）算出的尺度函數（上）與小波（下）：*Haar*、*db2*、*db4*。濾波器越長，函數越平滑、也越長。

一維離散小波轉換

有了這兩族函數，任何有限能量的訊號都能拆成一個粗略的近似，加上每個更細尺度的細節。**近似**（尺度）係數是訊號與選定最粗尺度 j_0 上的 $\varphi_{j_0,k}$ 的內積，**細節**（小波）係數則是訊號與每個更細尺度 j 上的 $\psi_{j,k}$ 的內積，與預備知識中完全相同。對長度 $M = 2^J$ 的取樣訊號，總和變成有限項，結果就是**離散小波轉換**（discrete wavelet transform, DWT）：近似係數 $W_\varphi(j_0,k)$ 與細節係數 $W_\psi(j,k)$，以尺度 j 與位置 k 編號。對採臨界取樣、並使用相容有限長度邊界構造的正交小波，係數總數恰好等於 M：這種 DWT 是基底轉換，不是擴張。但函式庫可能保留額外的邊界係數。PyWavelets 的 `periodization` 產生最少的係數，預設的對稱延拓則可能有冗餘；分析與合成必須使用相同的延拓模式 [21]。

僅線上版：小波級數與 *DWT* 公式 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/wavelets-transforms/notes/小波級數與-dwt-公式/*

快速小波轉換

白話版：你完全不需要去算 φ 或 ψ 的值。兩個短濾波器加上「每隔一個取一個」，一層一層做下去就夠了。

把精細化方程式代入 DWT，就得到 Mallat 的金字塔演算法，今天稱為**快速小波轉換**（fast wavelet transform, FWT）[7]：

$$W_\psi(j,k) = \sum_n h_\psi(n-2k)\, W_\varphi(j+1,n), \qquad W_\varphi(j,k) = \sum_n h_\varphi(n-2k)\, W_\varphi(j+1,n).$$

讀法：把較細一層的近似 $W_\varphi(j+1,\cdot)$ 與 h_ψ（或 h_φ）做相關，再每隔一個取一個輸出（**2 倍降取樣**）。用訊號處理的語言，這就是雙通道**分析濾波器組**（analysis filter bank）：低通分支產生下一

層近似，高通分支產生細節。把低通輸出再送進同一組濾波器，如此反覆。每一層資料量減半，所以總成本是 $O(M)$，比 FFT 的 $O(M \log M)$ 還快。

反向 FWT 是**合成濾波器組**：每個分支先 2 倍升取樣（插零）、經合成濾波器濾波，再相加。對正交小波，合成濾波器就是分析濾波器的時間反轉，整組濾波器能達成**完美重建**（perfect reconstruction）：即使每個分支的降取樣都造成混疊（aliasing），兩個分支的混疊項會互相抵消，輸出與輸入完全相同。Vetterli 與 Kovačević 的教科書從子頻帶編碼（subband coding）的角度深入發展了這個觀點 [2]。

```
import numpy as np
import pywt

x = np.array([4.0, 6.0, 10.0, 12.0, 8.0, 6.0, 5.0, 5.0])
w = pywt.Wavelet("haar")
h0, h1 = np.array(w.dec_lo), np.array(w.dec_hi)   # [.707 .707], [-.707 .707]

def analysis(x):
    lo = np.convolve(x, h0)[1::2]   # filter, then keep every 2nd sample
    hi = np.convolve(x, h1)[1::2]
    return lo, hi

a1, d1 = analysis(x)
a2, d2 = analysis(a1)
print("a2", a2, "d2", d2, "d1", d1.round(3))

ref = pywt.wavedec(x, "haar", level=2)            # [a2, d2, d1]
print(all(np.allclose(p, q) for p, q in zip(ref, [a2, d2, d1])))
print(np.allclose(pywt.waverec(ref, "haar"), x))
```

這兩行濾波器組與 `pywt.wavedec` 的結果完全一致。注意 $(5, 5)$ 這一對的細節係數為 0：沒有變化，就沒有細節。

二維小波轉換

白話版：先沿著列跑一維濾波器組，再沿著行跑一次。你會得到一張小而模糊的影像，以及三張分別對應不同方向的「邊緣圖」。

二維 DWT 是可分離的。一層分解使用一個二維尺度函數與三個二維小波，各由一維函數相乘而成：

$$\varphi(x,y) = \varphi(x)\varphi(y), \quad \psi^{H}(x,y) = \psi(x)\varphi(y), \quad \psi^{V}(x,y) = \varphi(x)\psi(y), \quad \psi^{D}(x,y) = \psi(x)\psi(y).$$

依照課本慣例，x 是列索引（垂直位置），y 是行索引。ψ^{H} 沿垂直方向變化、水平方向平滑，所以對**水平**邊緣有反應；ψ^{V} 對**垂直**邊緣有反應；ψ^{D} 則對對角細節與角點有反應。一層分解把 $M \times N$ 影像變成四個 $\frac{M}{2} \times \frac{N}{2}$ 的**子頻帶**（subband）：

子頻帶	垂直方向濾波	水平方向濾波	常見稱呼	PyWavelets 名稱
近似	低通	低通	LL	`cA`
水平細節	高通	低通	LH 或 HL（慣例不一）	`cH`
垂直細節	低通	高通	HL 或 LH	`cV`
對角細節	高通	高通	HH	`cD`

由於不同教科書與函式庫對 LH ／ HL 的標記剛好相反，本章一律使用不會混淆的 H、V、D。對近似子頻帶重複分解，就得到圖 7.4 那種熟悉的巢狀排列。

```
import numpy as np
import pywt
from skimage import data, img_as_float

img = img_as_float(data.camera())
LL1, (H1, V1, D1) = pywt.dwt2(img, "haar")   # one level
LL2, (H2, V2, D2) = pywt.dwt2(LL1, "haar")   # second level on LL1
print(img.shape, LL1.shape, LL2.shape)       # (512, 512) (256, 256) (128, 128)

e = lambda a: float((a ** 2).sum())
total = e(img)
print(f"energy in LL2: {e(LL2) / total:.4f}")

coeffs = pywt.wavedec2(img, "haar", level=2)  # same thing in one call
print(np.allclose(pywt.waverec2(coeffs, "haar"), img))
```

兩層之後，128 × 128 的近似子頻帶只占係數的十六分之一，卻裝了 camera 影像約 99% 的能量。其餘 15/16 的係數大多接近零，這正是壓縮與去雜訊所利用的稀疏性。

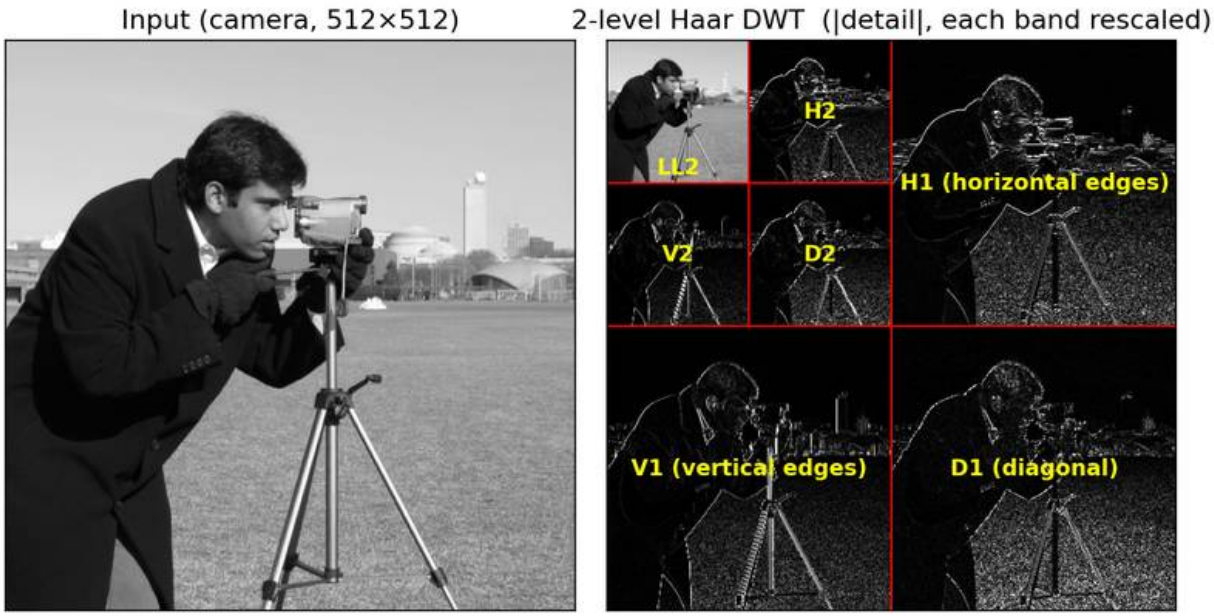


圖 *7.4 — camera* 影像的兩層哈爾 *DWT*。細節子頻帶顯示的是絕對值，且各自以其 *99.5* 百分位數重新縮放，好讓小係數看得見；若用同一個比例尺，它們幾乎全黑。三腳架垂直的腳在 *V1* 中特別亮，建築物的水平輪廓則出現在 *H1*。

僅線上版：小波封包 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/wavelets-transforms/notes/小波封包/*

應用

以門檻值去雜訊

白話版：在小波域裡，乾淨的影像只是少數幾個大數字，白雜訊則是散布各處的許多小數字。把小的歸零，再轉回來就好。

正交轉換會把標準差為 σ 的白高斯雜訊，變成每個子頻帶中標準差同樣為 σ 的白高斯雜訊，而影像的能量則集中在少數係數上。對細節係數 w 與門檻值 T，兩種門檻規則為：

$$\eta_{\text{hard}}(w) = \begin{cases} w, & |w| > T \\ 0, & |w| \le T \end{cases} \qquad \eta_{\text{soft}}(w) = \operatorname{sign}(w)\ \max(|w| - T,\, 0).$$

硬門檻（hard thresholding）只決定留或砍；軟門檻（soft thresholding）還會把留下的係數往零縮 T，得到連續的規則，假影較少，但對比會稍微損失。兩種經典的 T：

- **通用門檻值**（universal threshold，VisuShrink）：n 個取樣時 $T = \sigma\sqrt{2\ln n}$，出自 Donoho 與 Johnstone [12]。Donoho 證明用此 T 做軟門檻，所得估計有很高機率至少與真實訊號一樣平滑 [13]。雜訊強度通常由最細的對角子頻帶穩健地估計：$\hat{\sigma} = \text{median}(|w|)/0.6745$ [12]。
- **BayesShrink**：Chang、Yu 與 Vetterli 為每個子頻帶設定各自的門檻 $T = \hat{\sigma}^2/\hat{\sigma}_X$，其中 $\hat{\sigma}_X$ 是該子頻帶中乾淨係數的估計標準差 [14]。它會依各子頻帶含有多少訊號自動調整，通常優於通用門檻值。

```
import numpy as np
import pywt
from skimage import data, img_as_float
from skimage.metrics import peak_signal_noise_ratio as psnr

rng = np.random.default_rng(7)
clean = img_as_float(data.camera())
noisy = clean + rng.normal(0, 0.1, clean.shape)

def wavelet_denoise(y, wavelet="db4", level=3, mode="soft"):
    c = pywt.wavedec2(y, wavelet, mode="periodization", level=level)
    sigma = np.median(np.abs(c[-1][2])) / 0.6745       # noise level from finest HH
    T = sigma * np.sqrt(2 * np.log(y.size))            # universal threshold
    c = [c[0]] + [tuple(pywt.threshold(d, T, mode) for d in lvl) for lvl in c[1:]]
    return pywt.waverec2(c, wavelet, mode="periodization")

for mode in ["hard", "soft"]:
    out = np.clip(wavelet_denoise(noisy, mode=mode), 0, 1)
    print(mode, round(psnr(clean, out, data_range=1), 1), "dB")
```

```
print("noisy", round(psnr(clean, np.clip(noisy, 0, 1), data_range=1), 1), "dB")
```

在這個例子中，含雜訊影像的 PSNR 為 20.4 dB，硬門檻 25.7 dB，軟門檻 24.7 dB。通用門檻值本身就很保守（設計目標是幾乎清除所有雜訊），再加上軟門檻的收縮，就會過度平滑；圖 7.5 那種卡通般的外觀就是代價。BayesShrink 這類依子頻帶調整的門檻，或改用不抽樣（平移不變）的轉換，可以同時減輕模糊與方塊假影。scikit-image 在 `skimage.restoration.denoise_wavelet` 中包裝了這些變體。

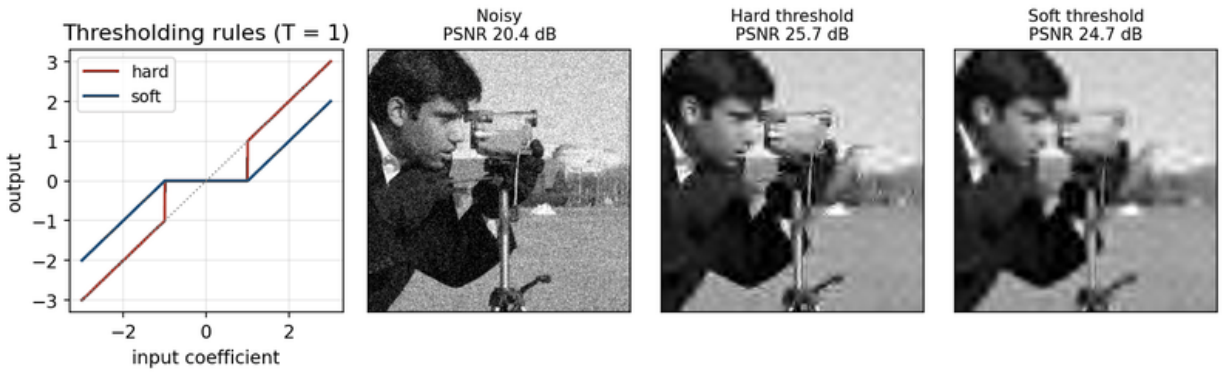
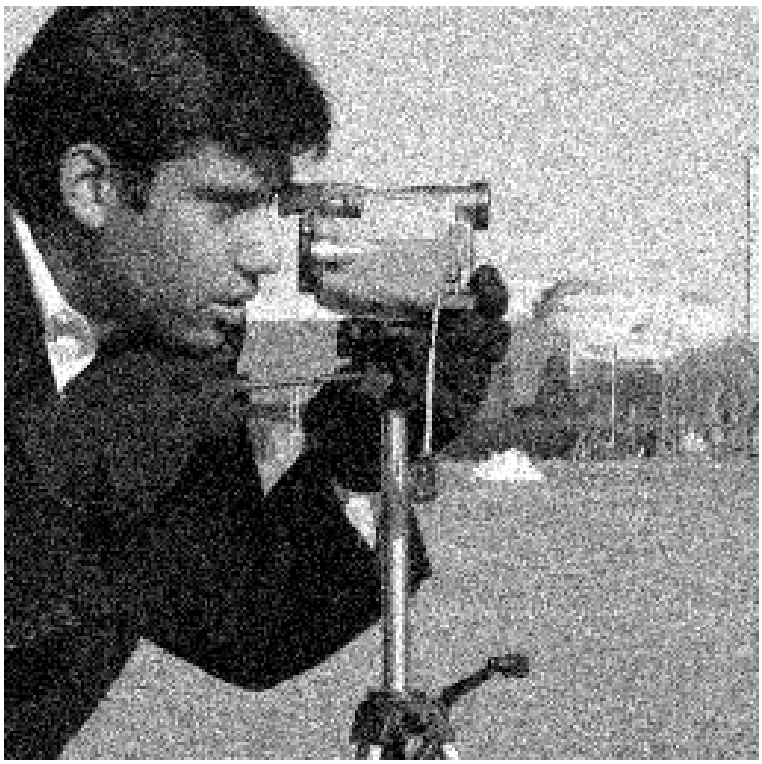


圖 *7.5 — camera* 影像的小波去雜訊（高斯雜訊，在 $[0, 1]$ 範圍內 $\sigma = 0.1$），使用 *db4*、三層分解與通用門檻值。硬門檻保留較銳利的邊緣，但留下零星的亮點；軟門檻較平滑，但也較模糊。

拖動滑桿，比較含雜訊的輸入與軟門檻的結果：



含雜訊



軟門檻

camera 影像含雜訊與小波去雜訊後的裁切比較

邊緣偵測

細節子頻帶是三個方向的高通響應，因此小波能當成快速的邊緣偵測器：把近似子頻帶設為零，再做反轉換。剩下的是影像減去它的粗略版本，主要集中在邊緣。只保留部分細節子頻帶就能挑選邊緣方向，如圖 7.6。越粗的層級得到的邊緣越粗，但對雜訊越穩健，這種多尺度行為類似於選擇高斯微分邊緣偵測器的 σ。

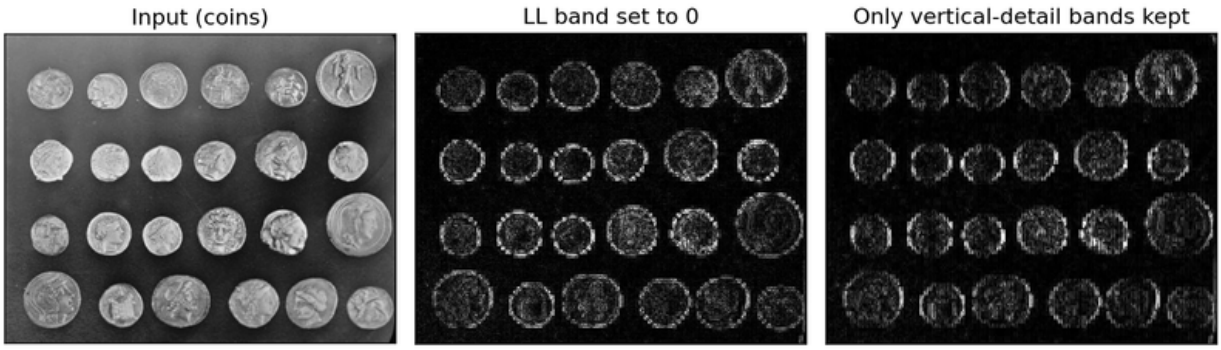


圖 *7.6* — 以 *coins* 影像兩層哈爾 *DWT* 得到的邊緣。中：近似子頻帶設為零。右：只保留垂直細節子頻帶，凸顯每枚硬幣左右兩側的邊緣。

壓縮預告

壓縮建立在能量集中之上：轉換、保留少數大係數、再有效率地編碼。圖 7.7 對整張 512 × 512 影像做四種轉換，只保留最大的 5% 係數。兩種小波比 DCT 高出 2 dB 以上、比 WHT 高出 4 dB 以上，而且誤差的樣貌不同：傅立葉類的基底會在邊緣周圍產生振鈴，並到處損失紋理；小波則保持邊緣清晰，主要在平滑或紋理區域損失細節。

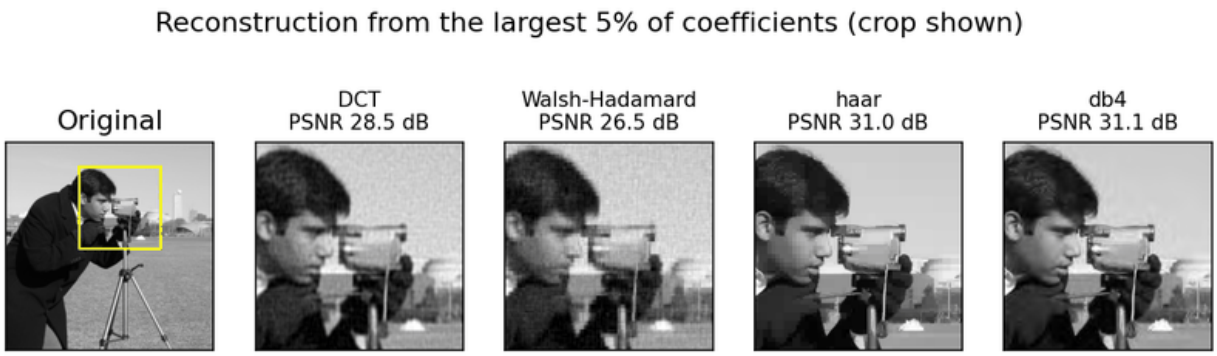


圖 *7.7* — 只用最大 *5%* 係數的重建結果（整張影像轉換，顯示裁切）。*PSNR*：*DCT 28.5 dB*、沃爾什–哈達瑪 *26.5 dB*、*Haar 31.0 dB*、*db4 31.1 dB*。

真正的編解碼器比「留下前 5%」細緻得多。JPEG 使用區塊 DCT 加上量化與熵編碼 [4]；JPEG 2000 則使用多層二維 DWT 與雙正交濾波器，再對每個子頻帶做位元平面編碼 [15]。IP 08 會詳細介紹這些流程，其中的轉換編碼實驗可以量化一張照片的 DCT 與 Haar 係數，看看誤差出現在哪裡。

現代觀點

小波在 1980 年代末以統一理論之姿進入影像處理，1990 與 2000 年代成為去雜訊與 JPEG 2000 的骨幹，如今則多半是一種成分：存在於稀疏模型、受訊號處理啟發的網路層，以及高效率的生成模型之中。以下是值得一讀的綜論與里程碑論文，以及各自的貢獻。

僅線上版：現代觀點詳讀 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/wavelets-transforms/papers/現代觀點詳讀/*

- **基礎**。 Mallat [7] 為影像引入多解析度分析、它與雙通道濾波器組的連結，以及 $O(N)$ 的金字塔演算法；Daubechies [8] 提供了具有緊支撐的正交小波，讓濾波器又短又實用。
- **綜述**。 Vetterli 與 Kovačević 免費公開的教科書 [2] 發展了本章的濾波器組觀點；Unser 與 Blu [9] 說明消失矩、逼近階數與平滑度從何而來；Rubinstein、Bruckstein 與 Elad [16] 追溯了從解析式轉換到從資料學習字典的演變。
- **實務**。 Skodras、Christopoulos 與 Ebrahimi [15] 說明 DWT、雙正交濾波器、子頻帶量化與嵌入式編碼如何在 JPEG 2000 中組合。

- **去雜訊理論**。收縮（shrinkage）論文 [12]、[13]、[14] 證明在小波域套一個簡單的非線性函數，對廣泛的訊號類別都接近最佳，而且依子頻帶調整門檻在實務上更有幫助。

深度學習改變了什麼、沒改變什麼

- **散射網路**。 Bruna 與 Mallat 的**小波散射轉換**（wavelet scattering transform）[17] 是一個濾波器固定為小波的卷積網路，後接取模（modulus）非線性與平均化。它可被證明對小形變穩定，並成為解釋 CNN 為何有效的參考點。
- **把小波當成降取樣器**。跨步卷積與最大池化在降取樣前沒有做適當的低通濾波，因此會混疊。WaveCNet [19] 在分類網路中以 DWT 層取代它們，提升了對雜訊的穩健性；**MWCNN** [18] 以可逆的 DWT 取代 U-Net 的池化，因此縮小特徵圖時不會遺失資訊。
- **大感受野**。 **WTConv** 層 [20]（ECCV 2024）在多層 DWT 的各子頻帶上做小卷積，因此感受野隨層數快速成長，參數成本卻很小。

沒有改變的部分。 DCT 仍然在 JPEG 裡，DCT 及其整數近似也仍是傳統影像與視訊編解碼器的核心工具。手工調校的小波門檻去雜訊，在品質上已被訓練出來的網路超越，但它仍是不需要訓練資料的實用基準。其風險保證取決於特定收縮結果所假設的訊號與雜訊條件，速度則取決於實作與影像大小。最重要的是，本章的詞彙（基底、框架、濾波器組、多解析度、混疊、稀疏性）正是用來分析卷積網路在做什麼的詞彙。

重點整理

- 線性轉換就是換基底。每個係數都是一個內積，也就是影像與某個基底函數的相關值。
- 正交歸一轉換就是矩陣乘法；可分離的二維情形為 $\mathbf{T} = \mathbf{A}\mathbf{F}\mathbf{A}^{T}$，反轉換用轉置即可，且能量守恆。
- 因為隱含的偶對稱延伸沒有邊界跳躍，DCT 對平滑影像的能量集中優於 DFT 與 WHT，這也是 JPEG 採用它的原因。
- WHT 與斜轉換分別以簡單性（只有 ± 1）或明確的斜坡向量換取部分能量集中能力；哈爾則多了局部性。
- 時頻平面呈現了取捨：傅立葉基底頻率精確、取樣時間精確，小波則依頻率調整磚的形狀。
- 多解析度分析把兩個短濾波器變成 $O(N)$ 的快速小波轉換；在二維中，每一層產生一個近似子頻帶與水平、垂直、對角三個細節子頻帶。
- 小波域的稀疏性帶動了去雜訊（硬／軟門檻）、快速邊緣圖與壓縮（JPEG 2000）。
- 現代網路把小波當成固定特徵擷取器（散射）、可逆降取樣器（MWCNN、WaveCNet），以及便宜的大感受野層（WTConv）。

練習

1. **親手做一個框架**。取平面上角度為 90°、210°、330° 的三個單位向量。分別對 $\mathbf{f} = (1, 0)$ 與 $\mathbf{f} = (0, 1)$ 計算 $\sum_k \langle \mathbf{f}, \mathbf{s}_k \rangle^2$。框架界是多少？要如何由三個係數重建 $\mathbf{f}$？

   提示

兩個總和都等於 3/2；事實上對任何單位向量 $\mathbf{f}$ 總和都是 3/2，所以這是 $A = B = 3/2$ 的緊框架。重建公式為 $\mathbf{f} = \frac{2}{3}\sum_k \langle \mathbf{f}, \mathbf{s}_k\rangle \mathbf{s}_k$。

2. **可分離的成本**。對一張 256×256 影像，計算下列情形所需的乘法次數：(a) 把一般二維線性轉換寫成一個 65536×65536 矩陣；(b) 以稠密 $\mathbf{A}$ 做可分離形式 $\mathbf{AFA}^T$；(c) 三層哈爾 FWT。

   提示

   (a) $N^4 \approx 4.3 \times 10^9$。(b) 兩次各 N^3 的乘法，$2N^3 \approx 3.4 \times 10^7$。(c) 每一層以 2 個係數的濾波器、在半速率下處理目前近似的每一列與每一行；總量是 N^2 乘上一個小常數，約數十萬次。

3. **為什麼要鏡射？**取 $f = [1,2,3,4,5,6,7,8]$（一個斜坡），用 SciPy 計算它的 DFT 與 DCT，分別數一數要多少個係數才能涵蓋 99% 的能量，並用隱含的週期延伸與偶對稱延伸解釋差異。

   提示

   週期延伸會從 8 跳回 1，需要很多 DFT 諧波。偶對稱延伸 $\ldots 2,1,1,2,\ldots,8,8,7,\ldots$ 是連續的，所以 DCT 只需要兩三個係數。

4. **手算哈爾**。對 $[2,2,6,6,3,5,9,9]$ 做兩層哈爾 DWT（平均與差都乘上 $1/\sqrt{2}$）。哪些細節係數是零？為什麼？用 `pywt.wavedec` 驗證你的答案。

   提示

   第一層的配對是 (2,2)、(6,6)、(3,5)、(9,9)，只有 (3,5) 的差不為零。第二層比較相鄰兩對縮放後的和。注意 PyWavelets 的正負號慣例：用它的哈爾濾波器，每個細節係數等於（前 − 後）$/\sqrt{2}$，所以 (3,5) 這一對得到 $-\sqrt{2}$。

5. **實際調整門檻**。修改去雜訊程式碼，改用 (a) 門檻值 $3\hat{\sigma}$，以及 (b) 依子頻帶計算的 BayesShrink 門檻 $\hat{\sigma}^2/\hat{\sigma}_X$，其中每個子頻帶的 $\hat{\sigma}_X = \sqrt{\max(\overline{w^2} - \hat{\sigma}^2, 0)}$。在 $\sigma = 0.05$ 與 $\sigma = 0.2$ 下，和通用門檻值比較 PSNR。

   提示

   512×512 影像的通用門檻值約為 $5\hat{\sigma}$，所以 (a) 會保留更多係數。軟門檻時，(b) 應該是三者中最好的。若 $\hat{\sigma}_X = 0$，就把整個子頻帶設為零。

6. **對平移的敏感度**。把 camera 影像水平平移一個像素，對兩個版本各做一層哈爾 DWT，比較 V 子頻帶的能量。為什麼平移一個像素會讓係數變化這麼大？不抽樣轉換（`pywt.swt2`）又會如何表現？

   提示

   2 倍降取樣讓 DWT 對平移敏感：落在同一個哈爾配對內部的邊緣會產生細節係數，落在兩對之間的同一條邊緣卻不會。平穩（不抽樣）轉換省略降取樣，因此是一個冗餘框架，係數會跟著影像一起平移。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 7. publisher page
2. M. Vetterli and J. Kovačević, *Wavelets and Subband Coding*, Prentice Hall, 1995（作者提供免費版本）. book site
3. N. Ahmed, T. Natarajan and K. R. Rao, "Discrete Cosine Transform," *IEEE Transactions on Computers*, 1974. doi
4. G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, 1992. doi
5. W. K. Pratt, J. Kane and H. C. Andrews, "Hadamard transform image coding," *Proceedings of the IEEE*, 1969. doi
6. W. K. Pratt, W.-H. Chen and L. R. Welch, "Slant transform image coding," *IEEE Transactions on Communications*, 1974. doi
7. S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1989. doi
8. I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, 1988. doi
9. M. Unser and T. Blu, "Wavelet theory demystified," *IEEE Transactions on Signal Processing*, 2003. doi
10. G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt and A. O'Leary, "PyWavelets: A Python package for wavelet analysis," *Journal of Open Source Software*, 2019. doi
11. R. R. Coifman and M. V. Wickerhauser, "Entropy-based algorithms for best basis selection," *IEEE Transactions on Information Theory*, 1992. doi
12. D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, 1994. doi
13. D. L. Donoho, "De-noising by soft-thresholding," *IEEE Transactions on Information Theory*, 1995. doi
14. S. G. Chang, B. Yu and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE Transactions on Image Processing*, 2000. doi
15. A. Skodras, C. Christopoulos and T. Ebrahimi, "The JPEG 2000 still image compression standard," *IEEE Signal Processing Magazine*, 2001. doi
16. R. Rubinstein, A. M. Bruckstein and M. Elad, "Dictionaries for sparse representation modeling," *Proceedings of the IEEE*, 2010. doi
17. J. Bruna and S. Mallat, "Invariant scattering convolution networks," arXiv:1203.1513, 2012. arXiv
18. P. Liu, H. Zhang, K. Zhang, L. Lin and W. Zuo, "Multi-level wavelet-CNN for image restoration," CVPR Workshops (NTIRE), 2018. arXiv
19. Q. Li, L. Shen, S. Guo and Z. Lai, "Wavelet integrated CNNs for noise-robust image classification," CVPR, 2020. arXiv
20. S. E. Finder, R. Amoyal, E. Treister and O. Freifeld, "Wavelet convolutions for large receptive fields," ECCV, 2024. arXiv
21. PyWavelets developers, "Signal extension modes," PyWavelets documentation, accessed 5 November 2026. 文件.

IP 08・影像壓縮與浮水印

影像如何被壓進更少的位元：從熵與霍夫曼碼，到 JPEG、JPEG 2000 與學習式編解碼器，以及如何把隱藏標記寫進影像。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/compression-watermarking/

你將學到

- 壓縮要去除的三種冗餘，以及熵（entropy）如何替無失真編碼設下下限
- 經典無失真編碼器的原理：霍夫曼、Golomb、算術、LZW、游程、符號式與位元平面編碼
- JPEG 如何把影像切成 8×8 的 DCT 區塊並量化，以及低品質時為什麼會出現方塊
- 預測編碼（DPCM、運動補償）與小波編碼（JPEG 2000）如何利用相鄰像素
- 如何把浮水印藏在最低有效位元或 DCT 係數中，以及為什麼一個撐得過 JPEG、另一個撐不過
- 學習式（神經網路）編解碼器與深度學習浮水印在今天的位置

先看全貌

一張未壓縮的 512×512 彩色照片是 786,432 位元組。同一張照片存成 JPEG 可能只有幾十 KB，看起來卻沒什麼差別。壓縮就是找出哪些位元可以預測、哪些細節看不出來，然後不去儲存它們。

> **用最簡單的話說**
>
> 想像你在電話裡跟朋友描述一張天空的照片。你不會說一百萬次「藍色、藍色、藍色……」，而是說「上面全是藍色，左邊有一朵白雲」。這就是壓縮：常見的東西簡短帶過，只把話花在意外的地方。浮水印則像是用隱形墨水在畫上簽名，之後就能證明這張畫是你的。

我們每天看到的影像幾乎都經過壓縮，許多電腦視覺資料集也以 JPEG 儲存，所以模型其實是在帶有壓縮瑕疵的影像上訓練的。浮水印則關係到著作權，也越來越常用來標記生成式模型產生的影像。本章依照教科書 [1] 的順序：先談基礎，再介紹一整套編碼方法，最後談浮水印。

基礎概念

白話版：影像檔裡的位元數通常比它真正承載的資訊多。壓縮就是去掉多餘的位元，而「資訊量」的度量告訴我們最多能壓到哪裡。

壓縮比與冗餘

若一張影像未壓縮時佔 b 位元，壓縮後佔 b' 位元，則**壓縮比**與**相對資料冗餘**為

$$C = \frac{b}{b'}, \qquad R = 1 - \frac{1}{C}.$$

C 是無因次的數字（10 代表「十比一」），R 是原始資料中屬於冗餘的比例。$C = 10$ 時 $R = 0.9$：九成的位元沒有帶來新資訊。

影像中有三種冗餘：

- **編碼冗餘（coding redundancy）**：固定 8 位元的碼對每個灰階都花 8 位元，不論常見或罕見。變長碼可以少浪費一些位元。
- **空間與時間冗餘（spatial and temporal redundancy）**：相鄰像素、相鄰的視訊畫格高度相關，把每個像素獨立編碼就是在重複資訊。
- **不相關資訊（irrelevant information）**：觀看者察覺不到的細節。去掉它是不可逆的（**失真**壓縮，lossy）；只去掉前兩種則是**無失真**壓縮（lossless）。

度量資訊：熵

機率為 $P(a)$ 的符號帶有 $I(a) = -\log_2 P(a)$ 位元的資訊：越罕見的事件越令人意外，資訊越多。對整個符號集合取平均就得到**熵**

$$H = -\sum_{j=1}^{J} P(a_j)\,\log_2 P(a_j) \quad \text{bits/symbol},$$

其中 $a_1, \dots, a_J$ 是所有可能的符號（8 位元影像就是 256 個灰階），$P(a_j)$ 是它們的機率。無雜訊信源編碼定理指出：當符號被逐一、獨立編碼時，沒有任何無失真碼的平均長度能低於 H 位元／符號；好的碼可以逼近 H。

對真實影像我們只能估計。**一階估計**使用正規化直方圖 $p_r(r_k) = n_k / MN$，其中 n_k 是 $M \times N$ 影像中灰階為 r_k 的像素數。圖 8.1 說明這個估計取決於你把什麼當成「符號」。camera 影像的一階熵是 7.23 位元／像素，幾乎等於 8；如果改為編碼每個像素與左鄰像素的差值，熵降到 4.70 位元。差值並沒有比較聰明，它只是去掉了直方圖看不到的空間冗餘。這正是本章後面預測編碼的核心想法。

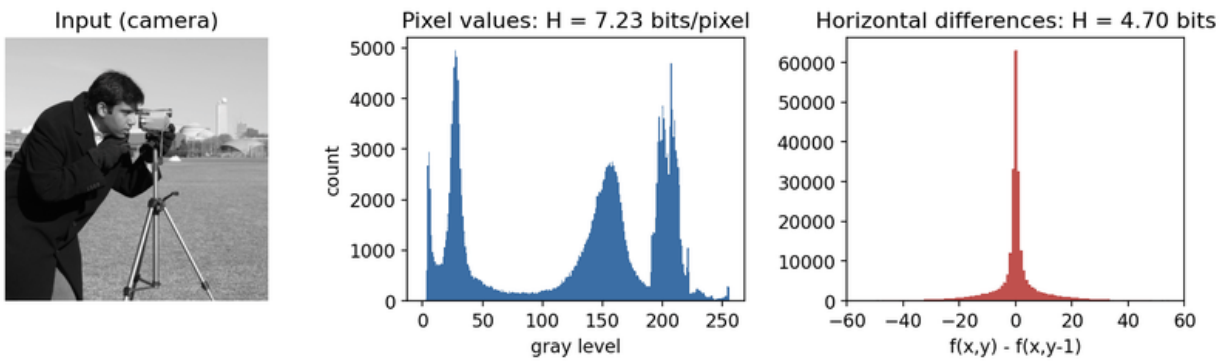


圖 *8.1* — 同一張影像、兩種符號選擇。像素值分布很散（*7.23* 位元）；相鄰差值集中在 *0* 附近（*4.70* 位元）。

```
import numpy as np
from skimage import data

def entropy(x):
    _, counts = np.unique(x, return_counts=True)
    p = counts / counts.sum()
    return -(p * np.log2(p)).sum()

img = data.camera()
print(f"H(pixels) = {entropy(img):.2f} bits")                          # 7.23
print(f"H(diffs)  = {entropy(np.diff(img.astype(int), axis=1)):.2f} bits")  # 4.70
```

保真度準則

有失真時，我們需要衡量損害。設 $f(x,y)$ 為原始影像、$\hat{f}(x,y)$ 為解壓縮後的 $M \times N$ 影像。**均方根誤差**與**峰值訊噪比**為

$$e_{\text{rms}} = \sqrt{\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y)-f(x,y)\right]^2}, \qquad \text{PSNR} = 20\log_{10}\frac{L_{\max}}{e_{\text{rms}}}\ \text{dB},$$

其中 $L_{\max}$ 是像素可能的最大值（8 位元影像為 255）。PSNR 越高，誤差越小；粗略來說，40 dB 左右的差異很難看出，低於 25 dB 則很明顯，但仍取決於影像內容。

這些**客觀**準則容易計算，卻不一定與人眼一致。**主觀**準則請觀察者以量表（從「極佳」到「無法使用」）評分，或兩兩比較。

影像壓縮模型

本章幾乎所有編解碼器都符合同一張方塊圖。**編碼器**有三個階段：

1. **映射器（mapper）**：把影像轉成冗餘較少的形式（差值、轉換係數、游程長度）；
2. **量化器（quantizer）**：依保真度目標丟棄精度（唯一有失真的階段，無失真編解碼器沒有這一步）；
3. **符號編碼器（symbol coder）**：給常見符號較短的碼（霍夫曼、算術、Golomb……）。

解碼器由符號解碼器與反映射器組成。失去的精度不存在「反量化」可以找回。

僅線上版：格式、容器與標準 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/compression-watermarking/notes/格式容器與標準/*

霍夫曼編碼

白話版：最常見的符號給最短的碼，而且任何一個碼都不是另一個碼的開頭，這樣位元流不需要分隔符號也能讀。

霍夫曼演算法（Huffman coding）[8] 在已知符號機率、且逐一編碼的條件下，建出最佳的**前綴碼**（prefix code）。它由下往上建構：

1. 列出所有符號與機率。
2. 把機率最小的兩項合併成新節點，機率為兩者之和。
3. 重複直到只剩一個節點（機率 1）。
4. 從根走到每個葉節點，一邊寫 0、另一邊寫 1，路徑就是碼字。

實作範例

考慮六個符號：$P(a_1) = 0.40$、$P(a_2) = 0.20$、$P(a_3) = 0.15$、$P(a_4) = 0.10$、$P(a_5) = 0.10$、$P(a_6) = 0.05$。合併順序如下：

步驟	合併	新節點
1	a_6（0.05）＋a_4（0.10）	0.15
2	a_5（0.10）＋a_3（0.15）	0.25
3	a_2（0.20）＋節點 0.15	0.35
4	節點 0.25 ＋節點 0.35	0.60
5	a_1（0.40）＋節點 0.60	1.00

從圖 8.2 的樹可得 $a_1 = $ `1`、$a_2 = $ `000`、$a_3 = $ `010`、$a_4 = $ `0010`、$a_5 = $ `011`、$a_6 = $ `0011`。平均碼長為

$$L_{\text{avg}} = \sum_{j} P(a_j)\, l(a_j) = 0.4(1) + 0.2(3) + 0.15(3) + 0.1(4) + 0.1(3) + 0.05(4) = 2.35 \text{ bits/symbol},$$

其中 $l(a_j)$ 是 a_j 的碼長。固定長度碼需要 3 位元，熵則是 $H = 2.284$ 位元，所以霍夫曼碼離下限不到 3%。機率相同時的平手可以任意處理；不同的選擇會改變碼字，但不會改變平均長度。

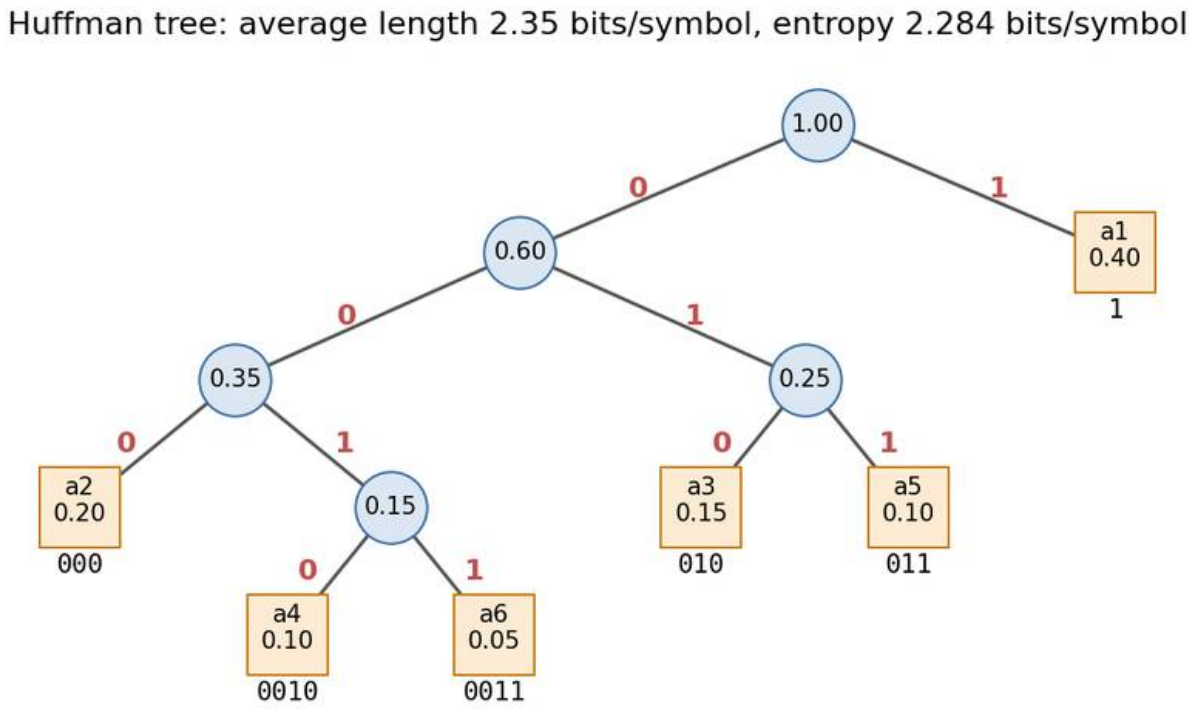


圖 *8.2* — 六符號範例的霍夫曼樹。邊上的數字是位元，碼字印在各葉節點下方。

解碼就是沿著樹往下走：讀位元直到碰到葉節點，輸出符號，再回到根。因為沒有碼字是另一碼字的前綴，這種碼是**即時可解**且**唯一可解**的。JPEG 用霍夫曼表來編碼量化後的係數 [2]。

```
import heapq, itertools

def huffman_codes(probs):
    tick = itertools.count()                        # tie-breaker for equal probabilities
    heap = [(p, next(tick), {s: ""}) for s, p in probs.items()]
    heapq.heapify(heap)
    while len(heap) > 1:
        p1, _, c1 = heapq.heappop(heap)             # least probable
        p2, _, c2 = heapq.heappop(heap)             # second least
        merged = {s: "1" + c for s, c in c1.items()}
        merged |= {s: "0" + c for s, c in c2.items()}
        heapq.heappush(heap, (p1 + p2, next(tick), merged))
    return heap[0][2]

P = {"a1": .40, "a2": .20, "a3": .15, "a4": .10, "a5": .10, "a6": .05}
print(huffman_codes(P))   # a1:'1', a2:'000', a3:'010', a4:'0010', a5:'011', a6:'0011'
```

它的限制是：每個符號至少花 1 個完整位元。若某符號機率是 0.95，它只帶 0.07 位元的資訊，霍夫曼碼卻仍要花 1 位元。下面的算術編碼可以突破這個限制。

Golomb 編碼

白話版：當小數字很常見、大數字很少見時，把數字寫成「有幾整組 m」（用畫記表示）再加「剩下多少」（用一般二進位表示）。

Golomb 碼 [9] 是為近似幾何分布的非負整數 n 設計的，而預測殘差經過映射後正是這種分布。給定參數 $m > 0$，碼 $G_m(n)$ 由兩段串接而成：

1. 商 $q = \lfloor n/m \rfloor$ 的一**元碼**：q 個 1 後接一個 0；
2. 餘數 $r = n \bmod m$ 的**截斷二進位碼**：令 $k = \lceil \log_2 m \rceil$、$c = 2^k - m$，$r < c$ 時用 $k-1$ 位元，其餘則把 $r + c$ 寫成 k 位元。

n	0	1	2	3	4	5	6	7	8
$G_4(n)$	000	001	010	011	1000	1001	1010	1011	11000
$G_3(n)$	00	010	011	100	1010	1011	1100	11010	11011

當 m 是 2 的冪次時，餘數就是 n 的最低 k 個位元，編碼只需要位移與遮罩。預測殘差 e 有正負號，所以要先映射成非負整數，例如 $e \ge 0$ 時 $M(e) = 2e$、$e < 0$ 時 $M(e) = 2|e| - 1$，得到 $0, -1, 1, -2, 2, \ldots$ 交錯排列。JPEG-LS [10] 結合這種映射，以及參數為 2 的冪次、並依局部情境調整的 Golomb 碼。

```
import numpy as np

def golomb(n, m):
    q, r = divmod(n, m)
    unary = "1" * q + "0"
    k = int(np.ceil(np.log2(m)))
    c = 2**k - m                          # how many remainders get k-1 bits
    if r < c:
        tail = format(r, f"0{k-1}b") if k > 1 else ""
    else:
        tail = format(r + c, f"0{k}b")
    return unary + tail

print([golomb(n, 3) for n in range(9)])
```

算術編碼

白話版：不替每個符號各自配碼，而是把整段訊息描述成 0 到 1 之間一個非常精確的數字。

算術編碼（arithmetic coding）[11] 從區間 $[0, 1)$ 開始。每讀一個符號，就把目前的區間依該符號的機率縮小到它所屬的子區間。訊息結束後，最終區間中的任何一個數都能代表這段訊息。若目前區間是 $[\ell, h)$，下一個符號 s 擁有累積範圍 $[F_{\text{lo}}(s), F_{\text{hi}}(s))$，則

$$\ell' = \ell + (h - \ell)\, F_{\text{lo}}(s), \qquad h' = \ell + (h - \ell)\, F_{\text{hi}}(s).$$

最終寬度等於各符號機率的乘積，所需位元數約為 $-\log_2 \prod_i P(s_i) = \sum_i -\log_2 P(s_i)$，正好是訊息的資訊量，沒有「每個符號至少 1 位元」的下限。

實作範例

令 $P(a) = 0.6$、$P(b) = 0.3$、$P(c) = 0.1$，於是 a 擁有 $[0, 0.6)$、b 擁有 $[0.6, 0.9)$、c 擁有 $[0.9, 1)$。編碼訊息 `abac`：

讀入	區間
`a`	$[0, 0.6)$
`b`	$[0.36, 0.54)$
`a`	$[0.36, 0.468)$
`c`	$[0.4572, 0.468)$

寬度為 $0.6 \times 0.3 \times 0.6 \times 0.1 = 0.0108$，即 $-\log_2 0.0108 \approx 6.5$ 位元的資訊。7 位元的二進位小數 $0.0111011_2 = 0.4609375$ 落在區間內，所以 7 個位元就能識別這段訊息（另外需要標示結尾，例如長度或結束符號）。

實際的編碼器使用整數運算並不斷重新正規化，避免區間下溢；也通常使用**自適應**機率模型，邊編碼邊更新，因此不必傳送機率表。JPEG 2000 的係數編碼器最後接的是自適應二元算術編碼器 [14]，「現代觀點」中的學習式編解碼器最後也是用算術編碼。

LZW 編碼

白話版：一邊讀一邊建立詞組字典；詞組再次出現時，只傳它在字典裡的編號。

Lempel–Ziv–Welch（LZW）演算法 [12] 一開始的字典只含所有單一符號（8 位元像素就是 0–255）。它持續延長目前詞組 w，只要「w 加上下一個符號」仍在字典中。一旦不在，就輸出 w 的碼，把延長後的詞組加為新項目（256、257……），再從新符號重新開始。解碼器能從收到的碼重建同一本字典，因此字典不需要傳送。

對一列重複四次 `39 39 126 126` 的 16 個像素，輸出是十個碼：

```
def lzw_encode(seq):
    table = {(v,): v for v in range(256)}
    w, out = (), []
    for x in seq:
        wx = w + (x,)
        if wx in table:
            w = wx
        else:
            out.append(table[w])
            table[wx] = len(table)
            w = (x,)
    if w:
        out.append(table[w])
    return out
```

```
row = [39, 39, 126, 126] * 4
print(lzw_encode(row))   # [39, 39, 126, 126, 256, 258, 260, 259, 257, 126]
```

從 256 起的碼依序代表 `39 39`、`39 126`、`126 126`、`126 39`、`39 39 126`……重複越長，詞組越長，壓縮比越好。LZW 不需要事先知道機率，因此在通用檔案格式中很受歡迎。PNG 使用的是相關的 LZ77 系 deflate 方法 [4]。

游程編碼

白話版：「0000 11 00000 1」變成「四個 0、兩個 1、五個 0、一個 1」。

游程編碼（run-length encoding, RLE）把每一段相同值的連續序列換成（值, 長度）。二值影像只要約定第一段的顏色，就只需要記長度。游程長度本身再用變長碼編碼，因為短游程比長游程常見。

```
import numpy as np

def rle(row):
    starts = np.r_[0, np.flatnonzero(np.diff(row)) + 1]
    lengths = np.diff(np.r_[starts, len(row)])
    return list(zip(row[starts].tolist(), lengths.tolist()))

print(rle(np.array([0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1])))  # [(0, 4), (1, 2), (0, 5), (1,
 ↪  1)]
```

RLE 對文件、線條圖和位元平面很有效，對游程很短的雜訊照片則效果不佳。二維版本會以上一列的顏色轉換位置為參考，編碼目前這列的轉換位置，因此也利用了垂直方向的相關性。JPEG 對量化後大量為零的係數也採用一種游程編碼 [13]。

符號式編碼

白話版：一頁掃描文件大多是同樣幾個字母一再出現。每種字形只存一次，然後只要說「第 7 號字母放在這裡」。

在**符號式**（symbol-based，或稱 token-based）編碼中，經常出現的子影像稱為**符號**，只在**符號字典**中存一次。影像變成一串三元組 (x_i, y_i, t_i)：每次出現的位置，以及它在字典中的編號 t_i。一頁有上千個字元、卻只有約一百種字形的文字，壓縮效果非常好，因為每次出現只花兩個座標和一個編號，這些再以熵編碼處理。

如果允許出現的圖樣與字典項目只是「相似」就算匹配，方法就變成有失真。這能提高壓縮比，但粗心的匹配器可能換錯字元，例如在低解析度掃描中把兩個長得像的數字互換。穩健的系統會把匹配門檻設得保守，或另外編碼圖樣與字典項目之間的殘差，以維持無失真。

位元平面編碼

白話版：把 8 位元影像拆成八張黑白影像，每個位元一張，再用二值方法分別壓縮。

灰階 $a = a_7 2^7 + a_6 2^6 + \cdots + a_0 2^0$ 有八個位元 $a_7, \ldots, a_0$。把每個像素的第 k 位元收集起來，就是**位元平面** k（圖 8.3）。高位元平面看起來像影像的粗略版本，有大片均勻區域；低位元平面看起來像雜訊。每個平面都可以用游程或其他二值方法編碼。

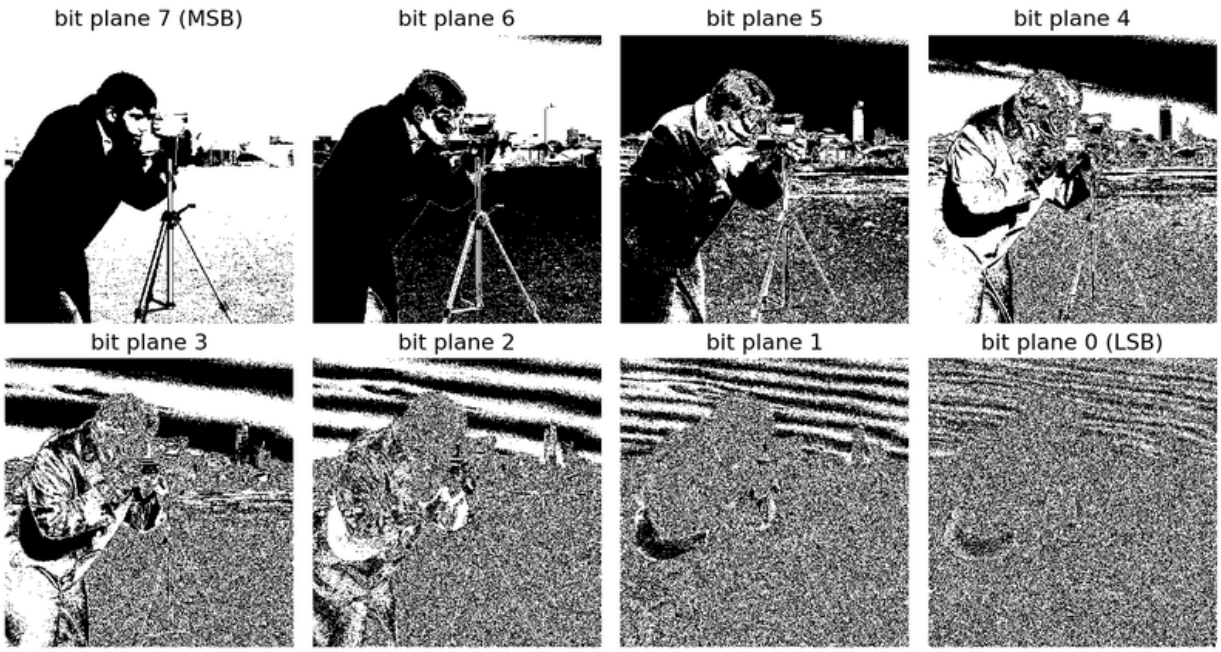


圖 *8.3 — 8* 位元影像的位元平面。結構集中在高位元平面，低位元平面接近隨機。

一般二進位碼的缺點是灰階的微小變化可能翻轉許多位元：127 是 `01111111`，128 是 `10000000`。**格雷碼**（Gray code）讓相鄰的值只差一個位元：

$$g_7 = a_7, \qquad g_i = a_i \oplus a_{i+1}, \quad 0 \le i \le 6,$$

其中 $\oplus$ 是互斥或（XOR）。以格雷碼拆出的平面在平滑區域會有更長的游程。以 camera 影像為例，位元平面 6 在水平相鄰像素間改變值的比例是 6.4%，對應的格雷碼平面只有 3.2%，平均游程約長一倍。在 NumPy 中轉換只要一行：`gray = img ^ (img >> 1)`。

JPEG 2000 也是逐個位元平面、由最高位往下編碼小波係數，這讓它的位元流可以在任何位元率截斷 [14]。

區塊轉換編碼

白話版：把影像切成小磚，每塊用一組標準圖樣（從平滑到細碎）的組合來描述，保留重要的圖樣，丟掉人眼幾乎看不出的細碎圖樣。

區塊轉換編碼器把影像切成不重疊的 $n \times n$ 子影像，對每塊做轉換、量化係數，再做熵編碼。轉換本身不會壓縮；它把大部分能量**集中**到少數係數，讓其他係數可以粗略量化或直接丟掉。

轉換的選擇

對 $n \times n$ 區塊 $g(x,y)$，二維**離散餘弦轉換**（DCT）為

$$T(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x,y) \cos \frac{(2x+1)u\pi}{2n} \cos \frac{(2y+1)v\pi}{2n},$$

其中 $\alpha(0) = \sqrt{1/n}$，$u = 1, \dots, n-1$ 時 $\alpha(u) = \sqrt{2/n}$；u, v 是水平與垂直頻率的索引。Karhunen–Loève 轉換（KLT，見第 7 章）在均方意義下集中能量的效果最好，但它的基底取決於資料，必須另外計算並傳送。DCT 是固定的、有快速演算法，而且對高度相關的影像，它的基底很接近 KLT。在區塊上它也勝過 DFT：DFT 把區塊視為週期函數，左右邊界的落差會產生假的高頻；DCT 隱含的偶對稱延伸則沒有這種落差。

子影像大小

較大的區塊能涵蓋更多相關性、集中能量的效果更好，但計算量更大，誤差也會擴散到更大的範圍。超過約 8×8 或 16×16 後效益很快趨緩。JPEG 固定使用 8×8；許多較新的編解碼器會在多種區塊大小間選擇，平滑區用大區塊，細節處用小區塊。

位元分配

轉換之後，要決定每個係數分到多少位元。

- **區域編碼**（zonal coding）在每個區塊都保留固定區域的係數（通常是變異量最大的低頻）。
- **門檻編碼**（threshold coding）在每個區塊保留超過門檻的係數，保留哪些係數會隨內容改變。

門檻編碼常見的實作是把每個係數除以**正規化（量化）陣列** Z 的對應元素再四捨五入：

$$\hat{T}(u,v) = \operatorname{round}\left[\frac{T(u,v)}{Z(u,v)}\right],$$

$\hat{T}(u,v)$ 就是實際被編碼的整數。解碼器再乘回去，$\hat{T}(u,v)\,Z(u,v)$，然後做反 DCT。Z 在高頻的大數值丟掉細節，在低頻的小數值保留粗略結構。

JPEG baseline

8 位元灰階的 JPEG baseline 編碼器 [2][13] 正是這樣做：

1. 像素值減 128，使其以 0 為中心。
2. 對每個區塊做 8×8 DCT。

3. 用 Z 表量化。標準提供一張亮度範例表；常見的慣例是以品質因子 $q \in [1,100]$ 縮放它：$q < 50$ 時縮放比例 $s = 5000/q$，否則 $s = 200 - 2q$（以百分比計）。
4. DC 係數 $\hat{T}(0,0)$ 以與前一區塊 DC 的差值編碼。
5. 63 個 AC 係數以**之字形**（zigzag）由低頻排到高頻，讓零集中在後段，再以（零的游程, 非零值）配對及霍夫曼表編碼，每個區塊以區塊結束符號收尾。

圖 8.4 追蹤一個暗而平滑的區塊在 $q = 50$ 時經過步驟 2–3 的過程。它的 DC 項很大（−603，來自位準平移），另有少數低頻項。量化後 64 個整數只剩 6 個非零，重建結果卻只差幾個灰階。

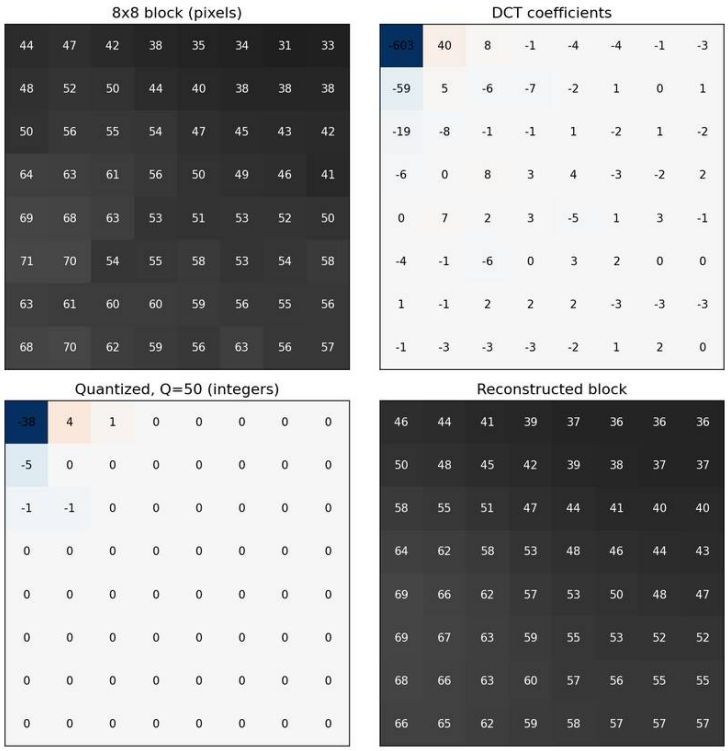


圖 *8.4* — 一個 *8×8* 區塊在品質 *50* 下經過 *JPEG* 流程。量化把 *64* 個係數中的 *58* 個變成零。

圖 8.5 把同樣的流程（不含熵編碼）套用到整張影像、四種品質設定。PSNR 從 $q = 90$ 的 40.3 dB 降到 $q = 5$ 的 26.3 dB，非零係數的比例則從 31.3% 降到 2.4%。低品質時會出現熟悉的瑕疵：**方塊效應**（blocking，各區塊獨立編碼，8×8 的接縫清晰可見）以及邊緣附近的**振鈴**（ringing，形成銳利邊緣所需的高頻被去掉了）。



圖 *8.5* — 不同品質設定下的 *JPEG* 式 *DCT* 量化。$q = 5$ 時方塊效應非常明顯。

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

```
import numpy as np
from scipy.fft import dctn, idctn

Q50 = np.array([[16,11,10,16,24,40,51,61],[12,12,14,19,26,58,60,55],
                [14,13,16,24,40,57,69,56],[14,17,22,29,51,87,80,62],
                [18,22,37,56,68,109,103,77],[24,35,55,64,81,104,113,92],
                [49,64,78,87,103,121,120,101],[72,92,95,98,112,100,103,99]])

def dct_codec(img, Q):
    f = img.astype(float) - 128                       # level shift
    H, W = f.shape
    B = f.reshape(H//8, 8, W//8, 8).swapaxes(1, 2)   # (H/8, W/8, 8, 8) blocks
    q = np.round(dctn(B, axes=(2, 3), norm="ortho") / Q)
    rec = idctn(q * Q, axes=(2, 3), norm="ortho")
    rec = rec.swapaxes(1, 2).reshape(H, W) + 128
    return np.clip(rec.round(), 0, 255).astype(np.uint8), q

rec, q = dct_codec(img, Q50)     # PSNR 32.6 dB, 12.0% nonzero coefficients
```

拖動下方的滑桿，比較一張彩色測試影像的無失真 PNG（465 kB）與品質 4 的真實 JPEG（7.6 kB，約小 60 倍）。注意平坦的背景、臉部和色彩邊緣：方塊、振鈴與顏色暈開（色彩通道以較低解析度儲存，量化也更粗）都看得到。



原圖（*PNG*）



JPEG q=4

太空人測試影像：無失真與重度量化 JPEG 的比較

預測編碼

白話版：用已經送出的像素猜下一個像素，只傳「猜錯多少」。猜得準，數字就小；數字小，就便宜。

無失真預測編碼

編碼器由先前已編碼的像素算出像素 $f(n)$ 的預測值 $\hat{f}(n)$，並編碼**預測誤差**

$$e(n) = f(n) - \hat{f}(n), \qquad \hat{f}(n) = \operatorname{round}\left[\sum_{i=1}^{m} \alpha_i f(n-i)\right],$$

其中 m 是預測器的階數，α_i 是預測係數。解碼器算出同樣的預測值再加回 $e(n)$，所以重建完全精確。圖 8.1 已經展示了效果：最簡單的「等於左鄰像素」預測器就把熵從 7.23 降到 4.70 位元／像素。

PNG 讓每一列從五種濾波器中選擇：不處理、左（Sub）、上（Up）、左與上的平均，以及 Paeth 預測器 [4]。JPEG-LS [10] 使用**中位數邊緣偵測器**（median edge detector）：令 a 為左鄰、b 為上鄰、c 為左上鄰，

$$\hat{f} = \begin{cases} \min(a,b) & c \geq \max(a,b) \\ \max(a,b) & c \leq \min(a,b) \\ a + b - c & \text{otherwise.} \end{cases}$$

前兩種情況在水平或垂直邊緣處選擇位於正確一側的鄰居；第三種則假設局部是平滑平面。

運動補償

視訊多了時間軸。**運動補償**（motion compensation）預測器不從同一畫格的鄰居預測，而是從先前解碼的畫格複製一個區塊，並依**運動向量** (d_x, d_y) 位移；運動向量透過搜尋最佳匹配（例如絕對差值總和最小）求得。編碼器傳送運動向量和殘差，殘差通常再以上一節的轉換編碼處理。只用自身編碼的畫格（intra frame）提供隨機存取的入口；預測畫格則小得多。HEVC（其靜態影像形式存於 HEIC 檔 [7]）與 AV1（AVIF 所用 [6]）都結合了這類畫格間預測與轉換編碼。

有失真預測編碼（DPCM）

若把預測誤差量化，$\dot{e}(n) = Q[e(n)]$，就得到**差分脈衝碼調變**（differential pulse-code modulation, DPCM）。關鍵細節是：編碼器必須以重建值 $\dot{f}(n) = \dot{e}(n) + \hat{f}(n)$ 做預測，也就是解碼器會擁有的那些值。如果編碼器改用原始值預測，量化誤差會在解碼器中累積（漂移，drift）。

```
import numpy as np

def dpcm(row, step):
    """Lossy DPCM with previous-sample prediction and a uniform quantizer."""
    rec = np.empty(len(row)); prev = 128.0; idx = []
    for i, x in enumerate(row.astype(float)):
        e = x - prev                      # prediction error
        k = int(np.round(e / step))       # quantizer index: this is what is sent
        prev = prev + k * step            # reconstruction, identical in the decoder
        rec[i] = prev; idx.append(k)
    return np.array(idx), rec
```

在 camera 影像的一列上以步長 8 執行，量化索引的熵是 1.87 位元／樣本，重建的 PSNR 為 40.8 dB。量化很粗時會出現兩種典型失真：**斜率過載**（slope overload，陡峭邊緣處重建值爬升得不夠快）以及**顆粒雜訊**（granular noise，平坦區域來回抖動）。

僅線上版：最佳預測器與量化器 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/compression-watermarking/notes/最佳預測器與量化器/*

小波編碼

白話版：不把影像切成磚塊，而是拆成一張模糊的小圖，加上好幾層不同尺度的細節。大多數細節值接近零，儲存成本低，而且沒有磚塊接縫。

小波編碼器先做第 7 章的離散小波轉換，再量化係數並做熵編碼。因為小波基底函數彼此重疊、在多個尺度上局部化，小波編碼器沒有區塊邊界，失真表現為模糊與輕微振鈴，而不是方塊。圖 8.6 比較 8×8 DCT 與五層小波轉換，兩者都只保留同樣 2% 的係數（絕對值最大者）。DCT 的結果是一片馬賽克，PSNR 26.8 dB；小波的結果較平滑，達到 28.6 dB。



圖 *8.6* — 同樣的係數預算（2%），不同的轉換。區塊 *DCT* 產生接縫，小波轉換則沒有。

```
import numpy as np, pywt

def wavelet_code(img, keep=0.02, wavelet="bior4.4", level=5):
    c = pywt.wavedec2(img.astype(float), wavelet, level=level, mode="periodization")
    arr, slices = pywt.coeffs_to_array(c)
    t = np.quantile(np.abs(arr), 1 - keep)            # keep the largest 2%
    arr[np.abs(arr) < t] = 0
    c = pywt.array_to_coeffs(arr, slices, output_format="wavedec2")
    return np.clip(pywt.waverec2(c, wavelet, mode="periodization"), 0, 255)
```

這個簡化範例算的是係數數量，不是位元數。真正的編碼器還必須告訴解碼器哪些係數留下來，而聰明的顯著性編碼正是在這裡發揮作用。

小波的選擇

小波決定了能量集中的程度與瑕疵的樣貌。影像編碼器偏好**雙正交**（biorthogonal）小波，因為它們可以是對稱的（邊緣不會位移），而且濾波器短。JPEG 2000 規定了兩種：有失真編碼用不可逆的 9/7 抽頭濾波器組，無失真編碼用可逆的整數 5/3 濾波器組 [14]。上面使用的 PyWavelets `bior4.4` 也有同樣的 9 與 7 抽頭長度。

分解層數

每一層把目前的近似子帶拆成四個一半大小的子帶。層數越多表示法越稀疏，但對常見的影像尺寸，超過約五層後效益遞減，而且量化非常粗的係數會影響大片區域。

量化器設計

小波編碼器通常使用在零附近有加大**死區**（dead zone）的均勻純量量化器：

$$q_b = \operatorname{sign}(c_b) \left\lfloor \frac{|c_b|}{\Delta_b} \right\rfloor,$$

其中 c_b 是子帶 b 中的係數，Δ_b 是該子帶的步長，q_b 是整數索引。小係數會落入零區間；它們可能代表真正的紋理或微弱細節，也可能是雜訊，所以捨棄它們是位元率與失真的取捨。步長依子帶選擇，高頻子帶的誤差比較不明顯，所以步長較大。

JPEG 2000

JPEG 2000 [3][14] 把這些元件組成完整的系統。影像可以切成大塊的 tile，色彩分量先經過分量轉換去相關，每個 tile 做小波轉換與死區量化。每個子帶再切成小的**編碼區塊**（code block），各自獨立地逐位元平面、以自適應算術編碼器編碼。最後，編碼後的片段被組織成**品質層**，同一個檔案只要讀取位元流的前段，就能以較低解析度或較低品質解碼。它也支援無失真編碼，以及以較高品質編碼的感興趣區域（region of interest）。

數位影像浮水印

白話版：浮水印是藏在影像裡、跟著影像一起流傳的額外資訊，例如簽名、擁有者 ID，或「這是模型生成的」標記。

浮水印的種類

- **可見**浮水印是疊加的圖案（半透明商標），會降低畫質，也常可被裁掉或塗掉。**不可見**浮水印的改變低於可見門檻，靠偵測器取回。
- **強健**（robust）浮水印能撐過壓縮、縮放、濾波與裁切（用於著作權與來源追溯）；**脆弱**（fragile）浮水印任何改動都會破壞（用於真偽驗證）；**半脆弱**浮水印能撐過輕度壓縮，但撐不過內容修改。
- **私有**（非盲）偵測需要原始影像；**盲**偵測則不需要。

每種方案都要在**不可察覺性**（影像改變多小）、**強健性**、**容量**（藏多少位元）與**安全性**（沒有金鑰的人能否找到、移除或偽造浮水印）之間取捨。

脆弱範例：最低有效位元

最簡單的不可見浮水印是把每個像素的最低有效位元（least significant bit, LSB）換成浮水印的一個位元：$f_w = 2\lfloor f/2 \rfloor + w$，$w \in \{0,1\}$。每個被改動的 8 位元像素恰好差一個灰階。若被改動的比例為 q，MSE 就是 q，PSNR 為 $10\log_{10}(255^2/q)$ dB。因此約半數像素改動時約為 51.14 dB；最壞情況 $q = 1$ 時為 48.13 dB。若 $q = 0$，兩張影像完全相同，PSNR 為無限大。這些數值直接來自前面的 PSNR 定義，不是所有浮水印都具有的實測保證。

```
def lsb_embed(cover, mark):          # cover: uint8 image, mark: 0/1 array of the same shape
    return (cover & 0xFE) | mark

def lsb_extract(img):
    return img & 1
```

圖 8.7 嵌入了一個二值的「DIP」標誌。從 PNG 取回時完全正確；但只要經過一次品質 95 的 JPEG 壓縮（肉眼看不出差別），取回的位元只有 55% 與浮水印相符，幾乎等於擲銅板。LSB 平面正是影像中像雜訊的部分，也是每個有失真編碼器最先丟掉的部分。這個範例展示的是脆弱性，不是經過認證的完整性：它沒有以金鑰把標記與影像內容綁定，任何人都能替換其中的位元。與預期標記比較可以察覺 JPEG 破壞，但成功取回標記本身，不能證明影像其他內容未曾改動。

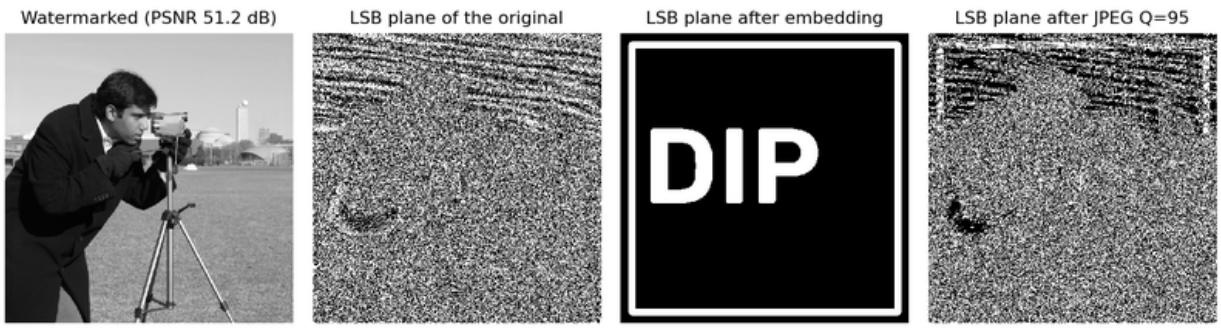


圖 *8.7 — LSB* 浮水印看不出來（*51 dB*）且能完整取回，直到一次輕度 *JPEG* 就把它抹除。

強健範例：DCT 域浮水印

要撐過 JPEG，就把浮水印藏在 JPEG 會保留資訊的地方：中低頻 DCT 係數。下面是我們自己設計的簡單方案。在每個 8×8 區塊中比較兩個中頻係數 $C(2,3)$ 與 $C(3,2)$，JPEG 對它們的量化步長相近。要嵌入位元 1，就讓 $C(2,3)$ 比 $C(3,2)$ 至少大 α；嵌入位元 0 則相反。偵測器只檢查哪個係數較大，因此是盲偵測。

```
import numpy as np
from scipy.fft import dctn, idctn

def blocks(img):
    H, W = img.shape
    return img.astype(float).reshape(H//8, 8, W//8, 8).swapaxes(1, 2).reshape(-1, 8, 8)
```

```
def unblocks(B, H, W):
    return B.reshape(H//8, W//8, 8, 8).swapaxes(1, 2).reshape(H, W)

def dct_embed(img, bits, alpha=12.0):
    C = dctn(blocks(img), axes=(1, 2), norm="ortho")
    for i, b in enumerate(bits):               # one bit per 8x8 block
        x, y = C[i, 2, 3], C[i, 3, 2]
        if (x - y if b else y - x) < alpha:    # enforce the order with margin alpha
            m = (x + y) / 2
            s = alpha / 2 if b else -alpha / 2
            C[i, 2, 3], C[i, 3, 2] = m + s, m - s
    out = unblocks(idctn(C, axes=(1, 2), norm="ortho"), *img.shape)
    return np.clip(out.round(), 0, 255).astype(np.uint8)

def dct_extract(img):
    C = dctn(blocks(img), axes=(1, 2), norm="ortho")
    return (C[:, 2, 3] > C[:, 3, 2]).astype(int)
```

在 512×512 的 camera 影像上，這個方案藏了 4,096 個位元（每個區塊一個）。邊距 α 就是調整強健性的旋鈕：

邊距 α	加浮水印後 PSNR	JPEG q=90 後位元錯誤率	q=75	q=50
12	43.2 dB	0.10%	0.56%	36.6%
30	38.4 dB	0%	0%	0.05%

小邊距幾乎看不出來，但當 JPEG 在這些位置的步長超過邊距時就會失效。大邊距能撐過品質 50，代價是 PSNR 少了約 5 dB。實際系統還會加上錯誤更正碼、用秘密金鑰把每個位元分散到多個區塊，並依局部紋理調整強度，因為人眼在紋理複雜處能容忍較大的改變。

現代觀點

經典流程（轉換、量化、熵編碼）至今仍是所有實際部署的影像編解碼器的基礎。大約 2016 年以後的改變是：每個階段都可以被學習，而同樣的「訓練編碼器與解碼器」想法也重塑了浮水印。

值得一讀的綜述

- **Wallace（1992）**[13]：JPEG 設計最清楚的概述。
- **Skodras、Christopoulos 與 Ebrahimi（2001）**[14]：JPEG 2000 的教學型回顧；它的價值與其說在單純的壓縮效率，不如說在功能（可分級性、感興趣區域編碼、無失真）。
- **Hu、Yang、Ma 與 Liu（2021）**[16]：沿著網路架構、熵模型與位元率控制綜述端到端學習式有失真壓縮。
- **Yang、Mandt 與 Theis（2023）**[17]：寫給機器學習讀者、專書篇幅的神經壓縮入門。
- **Zhong、Das、Alrasheedi 與 Tanvir（2023）**[18]：深度學習影像浮水印的綜述，將方法分成三類。

僅線上版：現代觀點綜述詳讀 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/compression-watermarking/papers/現代觀點綜述詳讀/*

學習式影像壓縮

Ballé 等人 [15] 以學習得到的非線性**分析轉換**（卷積神經網路）取代 DCT，把影像 x 映射到潛在表示 y，量化成 $\hat{y}$ 後再由學習得到的**合成轉換**解碼。整個系統以率失真損失訓練：

$$\mathcal{L} = R + \lambda D = E\left[-\log_2 p(\hat{y})\right] + \lambda\, E\left[d(x, \hat{x})\right],$$

其中 R 是在熵模型 p 下的期望碼長，D 是失真（例如 MSE），λ 決定兩者的取捨。他們的關鍵貢獻是**尺度超先驗**（scale hyperprior）：額外傳送一個小的潛在變數 z 作為旁資訊，用來預測 y 每個元素的尺度，讓熵模型隨每張影像調整。這和 JPEG 每張影像專屬的霍夫曼表、或 JPEG-LS 的情境建模是同一個想法，只是改成端到端學習。訓練時以加性均勻雜訊取代四捨五入，讓梯度能夠傳遞。

沒有改變的部分：量化仍是四捨五入，位元仍由算術編碼器 [11] 產生。學習式編解碼器可以理解為同一張編碼器方塊圖中更好的映射器與更好的機率模型。它們在實務上的障礙是解碼器的運算量，以及熵模型需要在各平台上逐位元一致。

現代標準編解碼器

同一時期，標準化的編解碼器也持續以人工設計的工具進步。AVIF [6] 與 HEIC [7] 沿用 AV1 與 HEVC 視訊編解碼器的畫格內編碼工具，JPEG XL [5] 鎖定攝影與網頁，PNG [4] 仍是無失真圖形的首選。這些格式沒有取代教科書的觀念，而是更彈性地組合預測編碼、區塊轉換與自適應熵編碼。

僅線上版：現代標準編解碼器詳述 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/compression-watermarking/notes/現代標準編解碼器詳述/*

深度學習浮水印

HiDDeN [19] 聯合訓練三個部分：把位元串藏進封面影像的編碼器、取回位元的解碼器，以及中間模擬裁切、模糊與 JPEG 的**雜訊層**（真正的 JPEG 無法微分，因此訓練時使用可微分的近似）。對抗損失讓加浮水印的影像保持自然。強健性是從模擬的攻擊中學來的，而不像前面那樣靠挑選係數來設計，因此它的強健程度取決於模擬得多好。

生成式模型帶來新的用途：在影像誕生時就加上標記。Stable Signature [20] 微調潛在擴散模型的解碼器，讓它生成的每張影像都帶有固定的隱藏簽章；即使經過裁切等編輯，擷取器仍能取回，誤報率低於百萬分之一。它回答的是來源問題（這張圖是不是模型 X 產生的？），而不是某張照片的著作權。移除與偽造攻擊仍是開放問題。

重點整理

- 壓縮去除編碼冗餘、空間／時間冗餘與不相關資訊；只有最後一種是有失真的。

- 熵是獨立符號無失真編碼的下限，而符號的選擇很重要：預測差值的熵遠低於原始像素。
- 霍夫曼碼是逐符號最佳；算術編碼逼近整段訊息的熵；Golomb、LZW、游程、符號式與位元平面編碼各自利用特定結構。
- JPEG ＝位準平移＋ 8×8 DCT ＋量化表＋之字形＋游程／霍夫曼編碼；品質由縮放量化表決定，低品質時出現方塊與振鈴。
- 預測編碼器必須以重建值預測；運動補償預測是視訊編碼的核心。
- 小波編碼（JPEG 2000）沒有區塊接縫，並支援可分級、無失真與感興趣區域編碼。
- LSB 浮水印看不見但很脆弱；DCT 域浮水印只要強度超過量化步長，就能撐過 JPEG。
- 學習式編解碼器與深度浮水印用訓練出的網路取代人工設計的轉換，但核心仍是量化與算術編碼。

練習

1. 某信源有四個符號，機率分別為 0.5、0.25、0.125、0.125。建出霍夫曼碼，計算平均碼長與熵，並解釋為什麼兩者在這裡相等。

   提示

   碼字為 `0`、`10`、`110`、`111`；兩個值都是 1.75 位元。所有機率都是 1/2 的冪次，所以 $-\log_2 P(a_j)$ 是整數，每個碼字都能恰好是那個長度。

2. 預測殘差 $e = 0, -1, 2, -3, 1$ 先用 Golomb 一節的 $M(e)$ 映射，再用 G_2 編碼。寫出位元串並計算位元數。改用 G_4 會比較短嗎？

   提示

   $M(e) = 0, 1, 4, 5, 2$。$m = 2$ 時碼為 `00`、`01`、`1100`、`1101`、`100`（15 位元）；$m = 4$ 時為 `000`、`001`、`1000`、`1001`、`010`（17 位元）。殘差小時，較小的參數勝出。

3. 使用算術編碼範例的機率（$P(a) = 0.6$、$P(b) = 0.3$、$P(c) = 0.1$），解出標記值 0.70 的前三個符號。

   提示

   0.70 落在 $[0.6, 0.9)$，第一個符號是 `b`。重新縮放：$(0.70 - 0.6)/0.3 = 0.333$，落在 a 的範圍。再縮放：$0.333/0.6 = 0.556$，又是 `a`。訊息開頭為 `baa`。

4. 修改 DPCM 程式，讓編碼器以原始的前一像素預測，而不是重建值，但解碼器不變。測量一列的 PSNR 並解釋看到的現象。

   提示

   解碼器把量化誤差加到自己的重建值上，而這個參考值已和編碼器不同。誤差沿著該列累積（漂移）。在 camera 影像第 200 列、步長 8 的情況下，PSNR 從 40.8 dB 掉到約 17 dB。

5. 把 LSB 浮水印改用位元平面 3 而不是位元平面 0。測量加浮水印後的 PSNR，以及經品質 95 與 75 的 JPEG 後位元相符率。你得到什麼、又付出了什麼？

   提示

   每個像素現在最多改變 8 個灰階，PSNR 大約下降 $20\log_{10} 8 \approx 18$ dB（隨機浮水印時約 33 dB）。JPEG 後的相符率在品質 95 時升到約 82%，在品質 75 時卻掉到約 57%：和 DCT 域方案相比，你付出了大量可見度，卻只換到一點強健性。

6. 從量化後留下的基底函數的支撐範圍，解釋為什麼小波編碼器在極低位元率時看起來模糊，而區塊 DCT 編碼器看起來是方塊。

   提示

   留下的 DCT 基底函數被限制在各自的 8×8 區塊內，相鄰區塊各自近似，交界處形成接縫。留下的粗尺度小波函數平滑且與鄰居重疊，所以近似結果連續，只是缺少細節。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 8. publisher page
2. Joint Photographic Experts Group, “JPEG 1 (ISO/IEC 10918-1 | ITU-T Rec. T.81): Digital compression and coding of continuous-tone still images,” JPEG overview page. jpeg.org
3. Joint Photographic Experts Group, “JPEG 2000 (ISO/IEC 15444),” JPEG overview page. jpeg.org
4. W3C, “Portable Network Graphics (PNG) Specification (Third Edition),” W3C Recommendation, 2025. w3.org
5. Joint Photographic Experts Group, “JPEG XL (ISO/IEC 18181),” JPEG overview page. jpeg.org
6. Alliance for Open Media, “AV1 Image File Format (AVIF),” v1.2.0, 2025. aomedia-codec.github.io
7. Nokia Technologies, “High Efficiency Image File Format (HEIF), ISO/IEC 23008-12: technical information.” nokiatech.github.io
8. D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proceedings of the IRE*, vol. 40, pp. 1098–1101, 1952. doi
9. S. W. Golomb, “Run-length encodings,” *IEEE Transactions on Information Theory*, vol. 12, pp. 399–401, 1966. doi
10. M. J. Weinberger, G. Seroussi and G. Sapiro, “The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS,” *IEEE Transactions on Image Processing*, vol. 9, pp. 1309–1324, 2000. doi
11. I. H. Witten, R. M. Neal and J. G. Cleary, “Arithmetic coding for data compression,” *Communications of the ACM*, vol. 30, pp. 520–540, 1987. doi
12. T. A. Welch, “A Technique for High-Performance Data Compression,” *Computer*, vol. 17, pp. 8–19, 1984. doi

13. G. K. Wallace, “The JPEG still picture compression standard,” *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992. doi
14. A. Skodras, C. Christopoulos and T. Ebrahimi, “The JPEG 2000 still image compression standard,” *IEEE Signal Processing Magazine*, vol. 18, pp. 36–58, 2001. doi
15. J. Ballé, D. Minnen, S. Singh, S. J. Hwang and N. Johnston, “Variational image compression with a scale hyperprior,” *ICLR*, 2018. arXiv
16. Y. Hu, W. Yang, Z. Ma and J. Liu, “Learning End-to-End Lossy Image Compression: A Benchmark,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. arXiv
17. Y. Yang, S. Mandt and L. Theis, “An Introduction to Neural Data Compression,” *Foundations and Trends in Computer Graphics and Vision*, vol. 15, no. 2, 2023. arXiv
18. X. Zhong, A. Das, F. Alrasheedi and A. Tanvir, “A Brief Yet In-Depth Survey of Deep Learning-Based Image Watermarking,” *Applied Sciences*, 2023. arXiv
19. J. Zhu, R. Kaplan, J. Johnson and L. Fei-Fei, “HiDDeN: Hiding Data With Deep Networks,” *arXiv:1807.09937*, 2018. arXiv
20. P. Fernandez, G. Couairon, H. Jégou, M. Douze and T. Furon, “The Stable Signature: Rooting Watermarks in Latent Diffusion Models,” *ICCV*, 2023. arXiv

IP 09・形態學影像處理

用小小的結構元素去「探測」形狀：侵蝕、膨脹、開運算、閉運算、擊中擊不中轉換、骨架、形態學重建與灰階形態學，附 scikit-image 程式碼。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/

你將學到

- 如何把二值影像看成像素座標的集合，並用一個小小的**結構元素**（structuring element）去探測它
- 兩個最基本的運算：**侵蝕**（erosion）與**膨脹**（dilation），以及它們組合成的**開運算**（opening）與**閉運算**（closing）
- **擊中擊不中轉換**（hit-or-miss transform）如何找出角點、孤立點這類精確的像素圖樣
- 由這些基本運算組成的經典演算法：邊界、填洞、連通元件、凸包、細化、骨架與修剪
- **形態學重建**（morphological reconstruction）：移除物件，卻不讓留下來的物件變形
- **灰階形態學**：形態學梯度、用頂帽轉換校正不均勻照明，以及粒度測定（granulometry）

先看全貌

前幾章的濾波器大多是線性的：每個輸出像素都是輸入像素的加權和。形態學不一樣，它問的是關於**形狀**的是非題：「這個小形狀放得進這裡嗎？」「它有碰到這裡的東西嗎？」所有答案只靠最小值、最大值、交集和聯集算出來，完全不需要乘法。

> **用最簡單的話說**
>
> 想像桌上有一片形狀歪七扭八的餅乾麵團，還有一個圓形的小餅乾模。把模子在桌上到處滑動。**侵蝕**是標記「模子可以整個壓在麵團上」的位置，所以麵團會變小，細細的部分會消失。**膨脹**是標記「模子有碰到任何麵團」的位置，所以麵團會變大，小縫隙會被補起來。**開運算**只留下真的能壓出餅乾的那些麵團；**閉運算**則把模子戳不進去的小凹陷全部填平。換一個模子形狀（線段、方塊、十字），就是對麵團問另一個問題。

為什麼重要：形態學是清理二值遮罩（不論來自 IP 10 的閾值化，或來自神經網路）的標準工具，也用來量測物件（計數、量大小、找骨架）以及校正不均勻的背景。它的理論基礎**數學形態學**（mathematical morphology）源自 Georges Matheron 與 Jean Serra 對隨機集合與多孔材料幾何的研究 [2]、[3]。Haralick、Sternberg 與 Zhuang 的教學論文把它推廣到整個影像分析社群 [4]，而 Soille 的書至今仍是最實用的標準參考 [5]。本章依照 Gonzalez 與 Woods [1] 的主題順序，程式碼使用 scikit-image [12] 與 SciPy。

預備知識

白話說：二值影像就是一串「亮著」的像素位置；結構元素則是一個標了中心點的小模板，我們拿它在這串位置上滑動。

把影像當成集合

設一張二值影像中，前景像素值為 1、背景像素值為 0。我們用下面的集合描述前景：

$$A = \{(x, y) \in \mathbb{Z}^2 \mid I(x, y) = 1\},$$

其中 I 是影像，(x, y) 是整數像素座標。於是集合運算都有了影像上的意義：**補集** A^c 是背景，**聯集** $A \cup C$ 是逐像素 OR，**交集** $A \cap C$ 是逐像素 AND，**差集** $A - C = A \cap C^c$ 則是把 C 的部分從 A 裡拿掉。

結構元素

結構元素（SE）B 是另一個由位移量組成的小集合，並指定一個**原點**（通常在中心）。把它想成探針。常見的選擇有 3×3 方塊、十字（4 鄰域）、圓盤或線段。結構元素的形狀決定了運算會對哪些特徵起反應：圓盤不偏任何方向，水平線段只對水平結構有反應。scikit-image 把結構元素稱為 **footprint**。實作上結構元素就是一個小小的布林陣列，`True` 的位置就是它的成員；圖示中標成「不在乎」的位置，其實就是不屬於這個集合。

反射與平移

接下來每個定義都會用到兩個集合運算。B 的**反射**（reflection）是

$$\hat{B} = \{w \mid w = -b,\ b \in B\},$$

也就是把每個位移 b 換成 $-b$：結構元素繞原點旋轉 180°。B 以向量 $z = (z_1, z_2)$ 做的**平移**（translation）是

$$(B)_z = \{c \mid c = b + z,\ b \in B\},$$

意思是把結構元素的原點放到像素 z 上。圓盤、方塊這類對稱的結構元素滿足 $\hat{B} = B$，反射看不出差別；不對稱的結構元素就要小心。

```
import numpy as np
from skimage import morphology

B = morphology.disk(2)            # 5x5 disk; origin at the centre (2, 2)
B_hat = B[::-1, ::-1]             # reflection: flip both axes
L = np.array([[1, 1, 0],
              [0, 1, 0],
              [0, 0, 0]], bool)   # an asymmetric element
print(L[::-1, ::-1].astype(int))  # its reflection points the other way
```

侵蝕與膨脹

白話說：侵蝕只在「整個模板都放得進形狀裡」的地方保留像素；膨脹則只要「模板碰到形狀」就把像素點亮。

侵蝕

A 被 B **侵蝕**的結果是

$$A \ominus B = \{z \mid (B)_z \subseteq A\},$$

也就是平移後的結構元素完全落在前景內的所有位置 z。等價寫法是 $A \ominus B = \{z \mid (B)_z \cap A^c = \emptyset\}$：結構元素不能和任何背景像素重疊。侵蝕會讓物件大約縮小一個結構元素半徑，刪掉比結構元素更細的部分，並把被細橋連在一起的物件切開。若 B 包含原點，侵蝕是**反擴張**（anti-extensive）的：$A \ominus B \subseteq A$。

膨脹

A 被 B **膨脹**的結果是

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\},$$

也就是反射後的結構元素至少碰到一個前景像素的所有位置。另一個等價寫法是把 B 的複本放在每個前景像素上再取聯集：$A \oplus B = \bigcup_{a \in A} (B)_a$。膨脹讓物件變大、接起比結構元素窄的縫隙、補上小洞。若 B 包含原點，膨脹是**擴張**（extensive）的：$A \subseteq A \oplus B$。和侵蝕不同，膨脹滿足交換律：$A \oplus B = B \oplus A$。

膨脹公式中的反射，讓膨脹成為真正的 Minkowski 和，也和卷積的「翻轉」一致。對稱結構元素則完全沒差。

對偶性

侵蝕與膨脹在補集與反射之下互為**對偶**（duality）：

$$(A \ominus B)^c = A^c \oplus \hat{B}, \qquad (A \oplus B)^c = A^c \ominus \hat{B}.$$

侵蝕前景，等於用反射後的結構元素膨脹背景，再把結果反過來。證明只要一行：$z \notin A \ominus B$ 恰好表示 $(B)_z$ 碰到 A^c，而這正是 $z \in A^c \oplus \hat{B}$ 的定義（因為 $\hat{\hat{B}} = B$）。寫程式時，必須一致地處理影像邊界，對偶性才會精確成立：對侵蝕而言影像外面要當成前景，對補集的膨脹而言外面才會是背景。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, morphology

A = ~data.horse()                 # True = horse pixels
```

```
B = morphology.disk(3)

eroded  = ndi.binary_erosion(A, structure=B)
dilated = ndi.binary_dilation(A, structure=B)
print(A.sum(), eroded.sum(), dilated.sum())     # erosion shrinks, dilation grows

# Duality check. border_value=1 treats outside pixels as foreground for the
# erosion, which matches background-outside for the dilation of ~A.
lhs = ~ndi.binary_erosion(A, structure=B, border_value=1)
rhs = ndi.binary_dilation(~A, structure=B[::-1, ::-1])
print("duality holds:", np.array_equal(lhs, rhs))   # True
```

開運算與閉運算

白話說：開運算是「先侵蝕、再膨脹回來」，能去掉小斑點和細長部分，大形狀則幾乎不變。閉運算是「先膨脹、再侵蝕回來」，能補上小洞和窄裂縫。

定義

A 以 B 做**開運算**，就是用同一個結構元素先侵蝕再膨脹：

$$A \circ B = (A \ominus B) \oplus B,$$

閉運算則是先膨脹再侵蝕：

$$A \bullet B = (A \oplus B) \ominus B.$$

單獨做侵蝕會丟掉細節，也會讓所有東西縮小；第二步則把倖存部分的大小補回來。這兩者也互為對偶：$(A \bullet B)^c = A^c \circ \hat{B}$、$(A \circ B)^c = A^c \bullet \hat{B}$。對前景做閉運算，就是對背景做開運算。

幾何解釋：滾球

開運算有一個很直觀的圖像：它等於所有能放進 A 的 B 平移複本的聯集，

$$A \circ B = \bigcup \{(B)_z \mid (B)_z \subseteq A\}.$$

想像一顆球（結構元素）在前景裡面滾動，而且必須整顆待在裡面。球滾得到的像素都保留；比球還尖的角、細頸和比球小的斑點都會消失。閉運算則是同樣的圖像換到外面：讓球在背景中滾動，凡是球到不了的背景像素（窄縫、小洞、又深又細的凹口）都變成前景。

性質

開運算與閉運算在嚴格意義上都是**形態學濾波器**（morphological filter）：它們是遞增的，而且是冪等的 [5]。

- **開運算是反擴張的：** $A \circ B \subseteq A$，只會移除像素。

- **閉運算是擴張的：** $A \subseteq A \bullet B$，只會加入像素。
- **遞增性：**若 $C \subseteq D$，則 $C \circ B \subseteq D \circ B$，閉運算亦同。輸入變大，輸出絕不會變小。
- **冪等性：** $(A \circ B) \circ B = A \circ B$，$(A \bullet B) \bullet B = A \bullet B$。同樣的開運算做兩次不會再改變任何東西，因為第一次留下來的部分本來就都是球放得進去的地方。

冪等性正是開、閉運算和侵蝕、膨脹的分水嶺：重複侵蝕會一直縮下去，重複開運算則會停下來。圖 9.1 把四種運算套用在一張帶雜訊的剪影上。開運算去掉背景中的白點；閉運算去掉馬身上的黑點；先開再閉則兩種都去掉，這是非常常用的**形態學雜訊濾波器**。

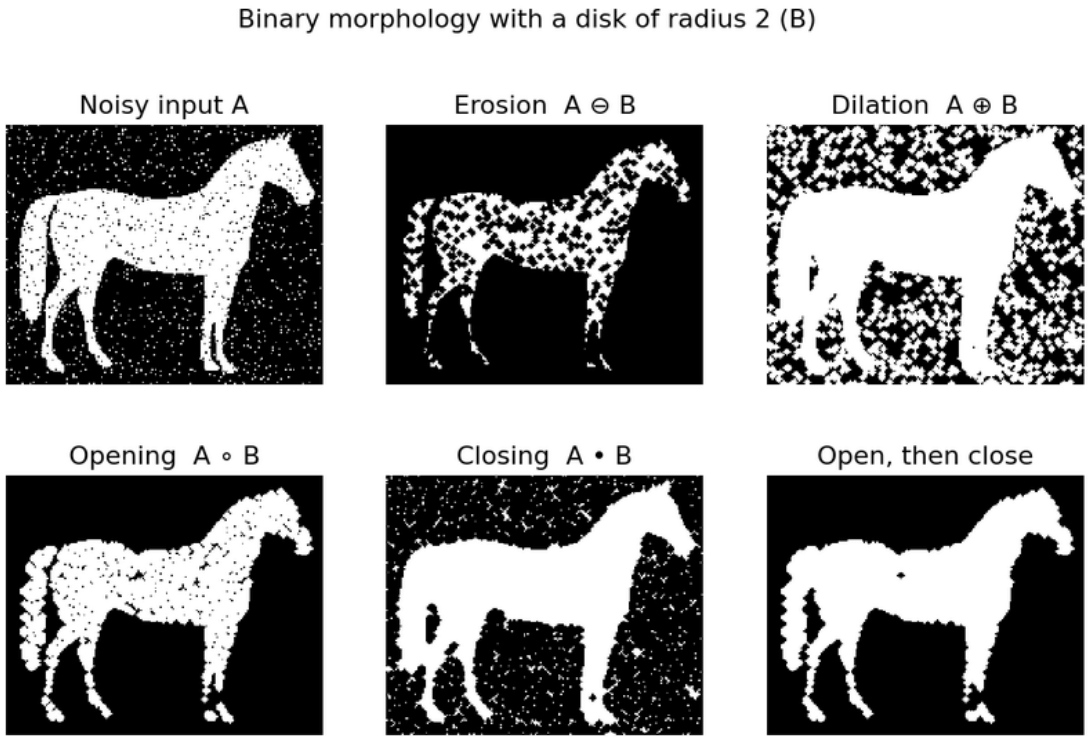


圖 *9.1* — 對翻轉了 *4%* 像素的馬剪影，以半徑 *2* 的圓盤做二值侵蝕、膨脹、開運算與閉運算。先開後閉可以同時去除兩種斑點，並保留輪廓。

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

```
import numpy as np
from skimage import data, morphology

A = ~data.horse()
B = morphology.disk(4)
op = morphology.binary_opening(A, B)
cl = morphology.binary_closing(A, B)
print("opening is anti-extensive:", not (op & ~A).any())
print("closing is extensive:     ", not (A & ~cl).any())
print("opening is idempotent:    ", np.array_equal(morphology.binary_opening(op, B), op))
```

擊中擊不中轉換

白話說：擊中擊不中轉換（HMT）就是二值圖樣的模板比對。你指定哪些像素一定要是前景、哪些一定要是背景，它就只在兩個條件同時成立的位置亮起來。

使用兩個不相交的結構元素：B_1 是必須落在前景的像素，B_2 是必須落在背景的像素。A 的 HMT 為

$$A \circledast B_{1,2} = \{z \mid (B_1)_z \subseteq A \text{ and } (B_2)_z \subseteq A^c\} = (A \ominus B_1) \cap (A^c \ominus B_2).$$

其中 $B_{1,2}$ 代表這一對 (B_1, B_2)；$A \ominus B_1$ 找出前景圖樣吻合的位置，$A^c \ominus B_2$ 找出背景圖樣吻合的位置。既不在 B_1 也不在 B_2 的像素是**不在乎**的位置。習慣上會把這一對畫成一個小網格：1 代表必須是前景、0 代表必須是背景、空白代表不在乎。

幾個實用的模板：

- **孤立點：** B_1 是中心像素，B_2 是它的 8 個鄰居。
- **左上角：** B_1 是中心加上右邊與下面的鄰居，B_2 是上面與左邊的鄰居。
- **單像素線段的端點：**中心與恰好一個鄰居是前景，其餘都是背景。每個方向需要一個模板（共 8 個旋轉）。

圖 9.2 套用了前兩種模板。注意數位圓盤的階梯狀輪廓在像素層級上確實有好幾個「左上角」：HMT 是逐字比對，不是幾何意義上的角。

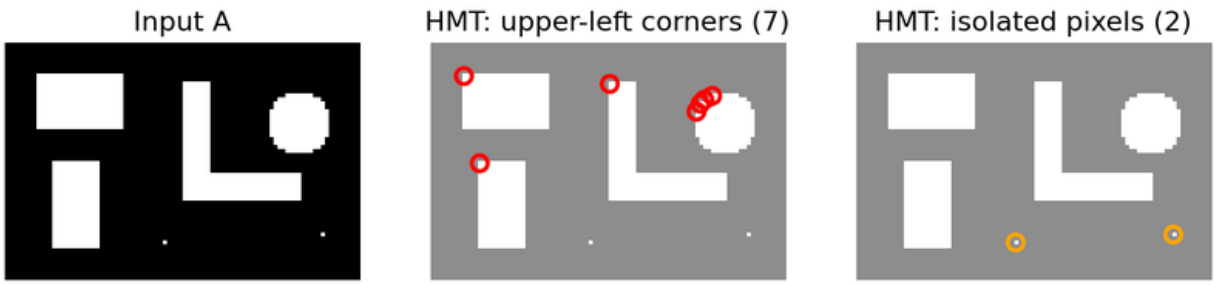


圖 *9.2* — 把擊中擊不中轉換當成精確的圖樣偵測器。中：左上角（包含數位圓盤上的小階梯）。右：孤立的前景像素。

```
import numpy as np
from scipy import ndimage as ndi

A = np.zeros((7, 9), bool)
A[1:5, 1:6] = True                # a rectangle
A[5, 7] = True                    # an isolated pixel
hit  = np.array([[0, 0, 0], [0, 1, 1], [0, 1, 0]], bool)   # must be foreground
miss = np.array([[0, 1, 0], [1, 0, 0], [0, 0, 0]], bool)   # must be background
corners = ndi.binary_hit_or_miss(A, structure1=hit, structure2=miss)
print(np.argwhere(corners))       # [[1 1]]: the upper-left corner only
```

HMT 是下面細化的基本積木；粗化與純形態學的凸包做法（都在「補充資料」頁）也建立在它之上。

一些基本的形態學演算法

白話說：有了侵蝕、膨脹和 HMT，就能組出許多實用工具。大部分都是一個短迴圈：反覆做同一件事，直到沒有任何變化為止。

邊界擷取

一個集合的內邊界，就是侵蝕會拿掉的部分：

$$\beta(A) = A - (A \ominus B),$$

其中 B 取 3×3 方塊（得到 8 連通的邊界）或十字（得到較粗、4 連通的邊界）。B 越大，邊界越粗。圖 9.3 第一格就是馬的 $\beta(A)$。

填洞

洞（hole）是被前景完全包圍的背景區域。只要在洞裡給一個種子像素，就能用膨脹讓種子長大，但每一步都把它限制在背景內：

$$X_k = (X_{k-1} \oplus B) \cap A^c, \qquad k = 1, 2, 3, \ldots$$

其中 X_0 只含種子，B 是 4 連通的十字，和 A^c 取交集讓生長在洞壁停下。當 $X_k = X_{k-1}$ 時停止，$X_k \cup A$ 就是填好的形狀。這種「先膨脹、再裁切」的模式稱為**條件膨脹**（conditional dilation），後面的測地膨脹會再看到它。B 用十字（而不是方塊）很重要：前景牆是 8 連通時，4 連通的背景填充就不會從斜角漏出去。

```
import numpy as np
from scipy import ndimage as ndi

def fill_hole(A, seed):
    """Grow X from a seed inside one hole: X_k = dilate(X_{k-1}) AND not A."""
    B = np.array([[0, 1, 0], [1, 1, 1], [0, 1, 0]], bool)   # 4-connected cross
    X = np.zeros_like(A); X[seed] = True
    while True:
        X_next = ndi.binary_dilation(X, structure=B) & ~A
        if np.array_equal(X_next, X):
            return A | X
        X = X_next

yy, xx = np.mgrid[-12:13, -12:13]
ring = (np.hypot(yy, xx) <= 10) & (np.hypot(yy, xx) > 6)   # boolean ring
print(ring.sum(), fill_hole(ring, (12, 12)).sum())        # 204 -> 317
```

擷取連通元件

把角色對調：要擷取 A 中包含某個種子像素的連通元件，就讓種子膨脹，但限制在前景內：

$$X_k = (X_{k-1} \oplus B) \cap A.$$

B 取 3×3 方塊時得到種子的 8 連通元件；取十字時得到 4 連通元件。對每個尚未標記的種子重複這個步驟，就能標記所有元件。實際的函式庫使用更快的一次或兩次掃描標記演算法，但結果相同：`scipy.ndimage.label(A, structure=np.ones((3, 3)))` 會傳回標記影像與元件數。

凸包

白話說：凸包就像用一條橡皮筋把物件整個圍起來得到的形狀。若一個集合中任兩點之間的線段都留在集合內，它就是**凸**的；**凸包**（convex hull）$C(A)$ 是包含 A 的最小凸集合。它會填平形狀的每個凹陷（圖 9.3 第二格）。形態學可以用擊中擊不中模板把它建出來，但實務上 `skimage.morphology.convex_hull_image` 直接計算前景像素座標的多邊形凸包再點陣化，速度更快，也更接近歐氏凸包。

僅線上版：以擊中擊不中迭代求凸包 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/以擊中擊不中迭代求凸包/*

細化

細化（thinning）有選擇地移除邊界像素，但不把形狀拆開，直到只剩一個像素寬的線。用模板 B 做一次細化是

$$A \otimes B = A - (A \circledast B),$$

也就是刪掉 HMT 比對到的像素。完整的細化會輪流套用一串模板 $\{B^1, B^2, \ldots, B^n\}$（通常是「邊緣像素」圖樣的 8 個旋轉），

$$A \otimes \{B\} = ((\cdots((A \otimes B^1) \otimes B^2)\cdots) \otimes B^n),$$

並重複整個循環，直到沒有像素改變。這些模板的設計保證：刪掉被比對到的像素，不會讓形狀斷開，也不會刪掉線段的端點。Zhang 與 Suen 的平行細化演算法是這個想法廣為使用的高效版本 [6]；`skimage.morphology.skeletonize` 對 2-D 影像預設就用它，`skimage.morphology.thin` 則實作了另一種相關的平行細化。

僅線上版：粗化 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/粗化/*

骨架

白話說：骨架是形狀中央的一條細線，像是形狀的「火柴人」版本。正式的定義使用最大圓盤：若某點是一個放得進 A、而且不被其他放得進的圓盤包含的圓盤中心，它就在**骨架**（skeleton）$S(A)$ 上。這種圓盤至少在兩處碰到邊界。實務上用細化來計算，得到連通、一個像素寬的骨架；或者計算**中軸**（medial axis，由距離轉換算出），它還會在每個骨架點上附帶最大圓盤的半徑，如圖 9.3 最右格。

僅線上版：骨架的形態學公式 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/骨架的形態學公式/*

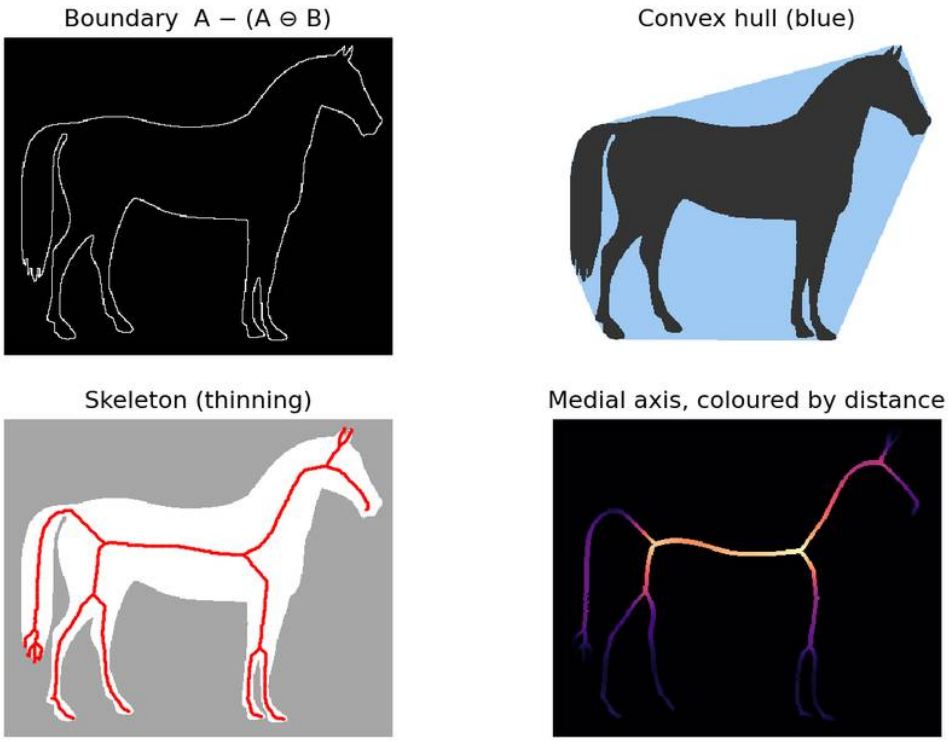


圖 *9.3* — 由侵蝕與 *HMT* 組成的演算法：內邊界、凸包、細化骨架，以及以顏色表示局部半寬的中軸。

修剪

骨架對邊界雜訊很敏感：輪廓上一個小凸起就會長出一根短短的**雜枝**（spur），圖 9.3 中馬尾附近就看得到。**修剪**（pruning）會移除長度短於 L 的雜枝：

1. 把骨架的**端點**（恰有一個 8 鄰居的像素）移除 L 次。所有長度不超過 L 的雜枝都會消失，但真正的分支末端也會少掉 L 個像素。
2. 找出縮短後骨架的端點。
3. 從這些端點做 L 次條件膨脹（限制在原骨架內），把主要分支長回原本長度。
4. 取縮短後的骨架與長回來的末端之聯集。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, morphology

def endpoints(S):
    n = ndi.convolve(S.astype(int), np.ones((3, 3), int), mode="constant") - S
    return S & (n == 1)            # foreground pixels with exactly one neighbour

def prune_tips(S, L):
    """Step 1 of pruning: strip L layers of end points."""
```

```
    S = S.copy()
    for _ in range(L):
        S &= ~endpoints(S)
    return S

skel = morphology.skeletonize(~data.horse())
print(skel.sum(), prune_tips(skel, 10).sum())
```

形態學重建

白話說：重建讓你可以說「把含有這顆種子的物件整個留下來」，而且拿回來的是一模一樣的物件，不是被磨圓的複本。它用兩張影像：**標記**（marker）說明從哪裡開始，**遮罩**（mask）說明最多能長到哪裡。

測地膨脹與測地侵蝕

設 F 為標記、G 為遮罩，且 $F \subseteq G$。F 相對於 G 的**大小為 1 的測地膨脹**（geodesic dilation）是

$$D_G^{(1)}(F) = (F \oplus B) \cap G,$$

其中 B 是小的結構元素（通常是 3×3 方塊）。標記長大一步，但不能離開遮罩。大小為 n 就是重複 n 次：$D_G^{(n)}(F) = D_G^{(1)}(D_G^{(n-1)}(F))$。「測地」指的是沿著留在 G 內的路徑量距離，就像在建築物裡走路，而不是穿牆。

對偶的**大小為 1 的測地侵蝕**是

$$E_G^{(1)}(F) = (F \ominus B) \cup G,$$

此時 $F \supseteq G$：標記會縮小，但不會縮到比遮罩還小。

以膨脹重建與以侵蝕重建

把測地膨脹一直迭代到不再變化，就得到遮罩 G 由標記 F 做的**以膨脹重建**（reconstruction by dilation）：

$$R_G^D(F) = D_G^{(k)}(F) \quad \text{with } k \text{ such that } D_G^{(k)}(F) = D_G^{(k+1)}(F).$$

在二值影像中，$R_G^D(F)$ 恰好是 G 裡所有至少含一個標記像素的連通元件之聯集。同理，**以侵蝕重建** $R_G^E(F)$ 是把測地侵蝕迭代到穩定。Vincent 的論文給出了標準的高效演算法（以佇列為基礎，每個像素只被拜訪有限次，而不是一次次掃過整張影像），以及許多應用 [7]；`skimage.morphology.reconstruction` 就是採用這個方法。

以重建做開運算

一般的開運算會移除小物件，但也會把大物件的角磨圓、把細的部分擦掉。**以重建做開運算**（opening by reconstruction）避開了這個問題：

$$O_R^{(n)}(F) = R_F^D(F \ominus nB),$$

其中 F 是輸入影像，$F \ominus nB$（以 B 侵蝕 n 次）是標記，輸入影像本身就是遮罩。侵蝕決定哪些物件能留下（至少放得進一個結構元素的那些）；重建則把每個倖存者恢復成原本的形狀。比較圖 9.4 的「一般開運算」與「以重建做開運算」：方塊上的細手臂和尖角都完整回來了。**以重建做閉運算**是其對偶，由膨脹與以侵蝕重建組成。

自動填洞

重建也能在不給種子的情況下一次填滿所有洞。洞就是從影像邊框走不到的背景區域。因此建立一個標記：在邊框上取影像的補值，其他地方為零，

$$F(x,y) = \begin{cases} 1 - I(x,y) & \text{if } (x,y) \text{ is on the image border,} \\ 0 & \text{otherwise,} \end{cases}$$

由它重建背景，再取補集：

$$H = [R_{I^c}^D(F)]^c.$$

$R_{I^c}^D(F)$ 是從邊框走得到的背景；其餘的部分 H 就是前景加上它的洞。

清除邊界物件

碰到影像邊界的物件通常不完整，量測時應該排除。用影像在邊框上的值當標記，

$$F(x,y) = \begin{cases} I(x,y) & \text{if } (x,y) \text{ is on the border,} \\ 0 & \text{otherwise,} \end{cases} \qquad X = I - R_I^D(F),$$

重建會找回所有碰到邊框的物件，從影像中減掉它們，就只剩內部的物件。

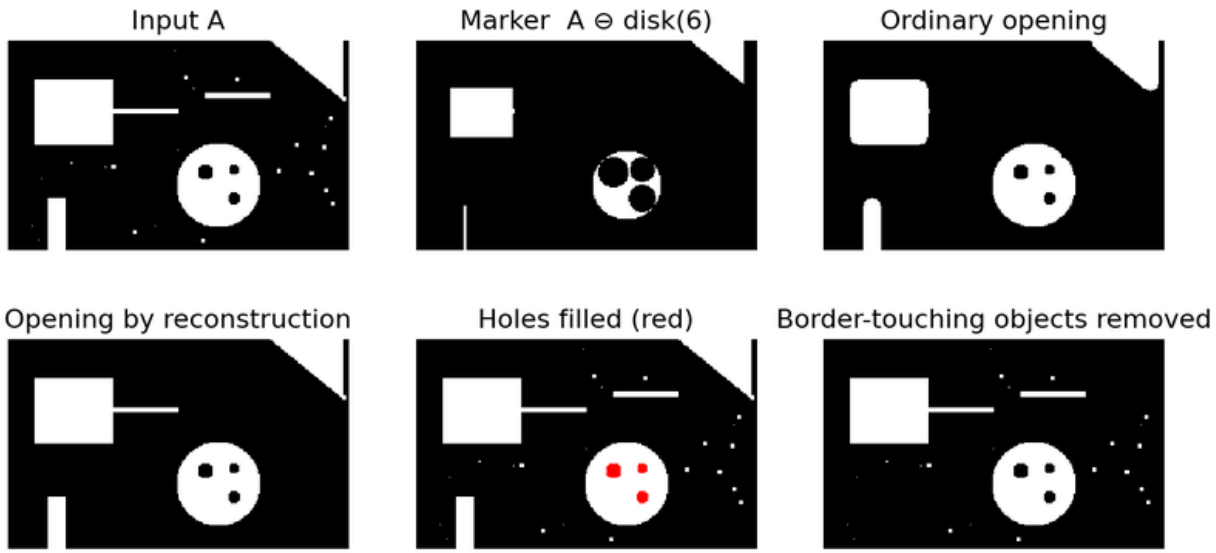


圖 *9.4* — 形態學重建。以半徑 *6* 圓盤侵蝕得到的標記挑出大物件，重建把它們（包括細手臂）完整恢復。下排：自動填洞與清除邊界物件。

```
import numpy as np
from skimage import morphology, segmentation

def open_by_reconstruction(A, footprint):
    marker = morphology.binary_erosion(A, footprint)
    return morphology.reconstruction(marker.astype(np.uint8), A.astype(np.uint8),
                                     method="dilation").astype(bool)

def fill_holes_auto(A):
    """Background reachable from the image frame is background; the rest is hole."""
    Ac = ~A
    marker = np.zeros_like(Ac)
    marker[0, :], marker[-1, :] = Ac[0, :], Ac[-1, :]
    marker[:, 0], marker[:, -1] = Ac[:, 0], Ac[:, -1]
    reached = morphology.reconstruction(marker.astype(np.uint8), Ac.astype(np.uint8),
                                        method="dilation").astype(bool)
    return ~reached

# Library shortcuts: scipy.ndimage.binary_fill_holes(A), segmentation.clear_border(A)
```

二值運算總整理

白話說：本章所有的二值工具，都是由侵蝕與膨脹再加上集合運算（補集、聯集、交集）組成的。最值得記住的是：開、閉運算用來清理，擊中擊不中轉換用來找精確圖樣，重建則用來完整保留整個物件。

僅線上版：二值運算一覽表 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/二值運算一覽表/*

灰階形態學

白話說：在灰階影像裡，「放得進去」變成「局部最小值」，「碰得到」變成「局部最大值」。把影像想成一片地形，亮度就是高度。侵蝕把每個像素降到模板底下最低的地面；膨脹則把它升到最高處。

灰階侵蝕與膨脹

使用**平坦**結構元素 b（一組高度都是零的位移）時，影像 f 的灰階侵蝕與膨脹是

$$[f \ominus b](x,y) = \min_{(s,t)\in b} f(x+s,\, y+t), \qquad [f \oplus b](x,y) = \max_{(s,t)\in b} f(x-s,\, y-t),$$

其中 (s,t) 走遍結構元素的所有位移。膨脹中的負號一樣是反射。用在二值影像上，這兩個式子會完全退化成集合的定義。侵蝕讓影像變暗、亮的特徵縮小、暗的特徵變寬；膨脹則相反。

侵蝕與膨脹也可以使用在每個位移上帶有高度的**非平坦**結構元素，但實務上很少用，因為結果會受影像的亮度尺度影響。

僅線上版：非平坦結構元素 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/非平坦結構元素/*

開運算與閉運算

定義不變：$f \circ b = (f \ominus b) \oplus b$、$f \bullet b = (f \oplus b) \ominus b$。幾何圖像是一顆球從亮度曲面下方往上推：開運算就是球在不超過 f 的前提下能到達的最高曲面。比球窄的亮峰會被削平，比球寬的部分則不受影響。閉運算是球從上方往下壓：比球窄的暗谷會被填平。各種性質（反擴張／擴張、遞增、冪等）只要把 $\subseteq$ 換成 $\leq$ 就照樣成立。

形態學平滑

開運算壓掉小的亮細節，閉運算壓掉小的暗細節，所以用同一個結構元素**先開後閉**，可以同時去除這兩種小結構，同時保留較大區域的銳利邊緣。這就是圖 9.1 雜訊濾波器的灰階版本。相關的**交替序列濾波**（alternating sequential filtering）則用逐漸變大的結構元素輪流做開、閉運算，避免一次用大結構元素造成的瑕疵。

形態學梯度

膨脹減去侵蝕，可以衡量局部對比：

$$g = (f \oplus b) - (f \ominus b).$$

在均勻區域內最大值與最小值幾乎相等，所以 $g \approx 0$；跨過邊緣時兩者相差約一個邊緣高度。和 Sobel 導數不同，g 一定非負，而且（對稱 b 時）不偏任何方向。只取一邊則得到**內梯度** $f - (f \ominus b)$ 或**外梯度** $(f \oplus b) - f$，分別把邊緣響應放在較亮區域的內側或外側。

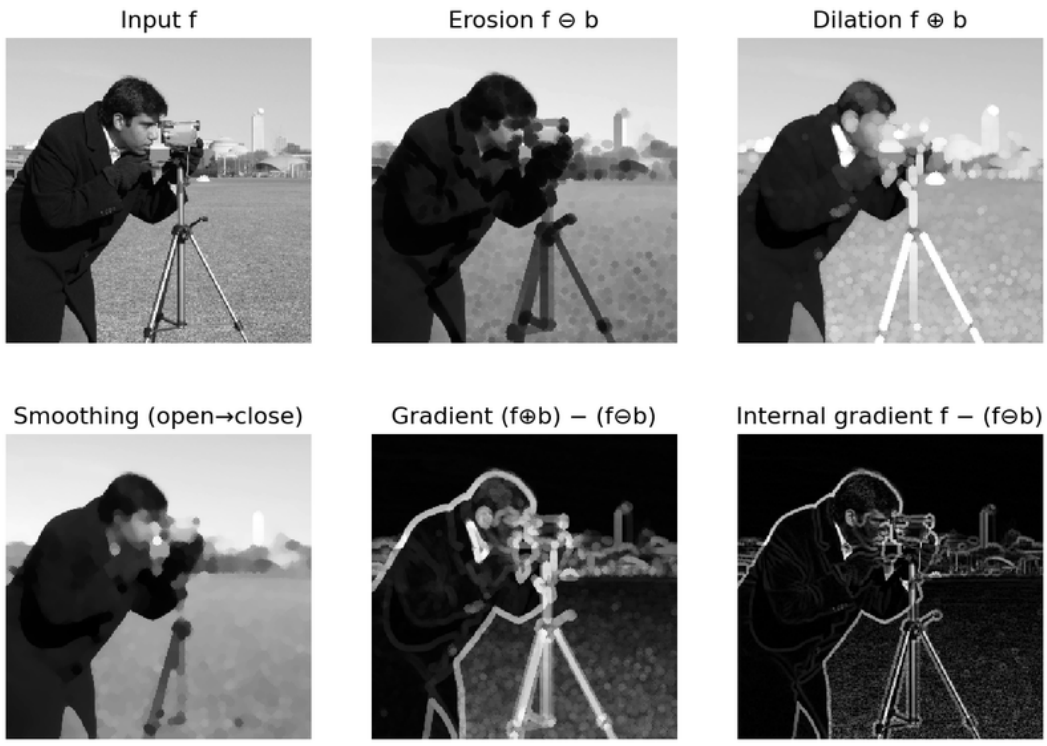


圖 *9.5* — 以半徑 *3* 的平坦圓盤做灰階形態學：侵蝕（局部最小）、膨脹（局部最大）、先開後閉的平滑、形態學梯度，以及內梯度。

```
from skimage import data, morphology, util

f = util.img_as_float(data.camera())
b = morphology.disk(3)                          # flat structuring element
ero  = morphology.erosion(f, b)                 # local minimum
dil  = morphology.dilation(f, b)                # local maximum
grad = dil - ero                                # morphological gradient (>= 0)
smooth = morphology.closing(morphology.opening(f, b), b)
```

頂帽與底帽轉換

用比亮物件更大的結構元素做開運算，會把物件削掉，但保留緩慢變化的背景。因此把影像減去開運算結果，就只剩下物件。這就是**頂帽轉換**（top-hat，又稱 white top-hat）：

$$T_{\text{hat}}(f) = f - (f \circ b).$$

它的對偶是**底帽轉換**（bottom-hat，又稱 black top-hat），用來擷取亮背景上的暗物件：

$$B_{\text{hat}}(f) = (f \bullet b) - f.$$

兩者都非負。最主要的用途是在閾值化之前校正**不均勻照明**。在圖 9.6 中，一張從單側打光的合成顆粒影像讓全域 Otsu 閾值（IP 10）完全失效：亮側的背景比暗側的顆粒還亮。用半徑 8（大於

任何顆粒）的圓盤做頂帽轉換後，背景變平，單一閾值就幾乎完美地分出顆粒。唯一的參數是結構元素大小：它必須大於物件，又要小於背景變化的尺度。

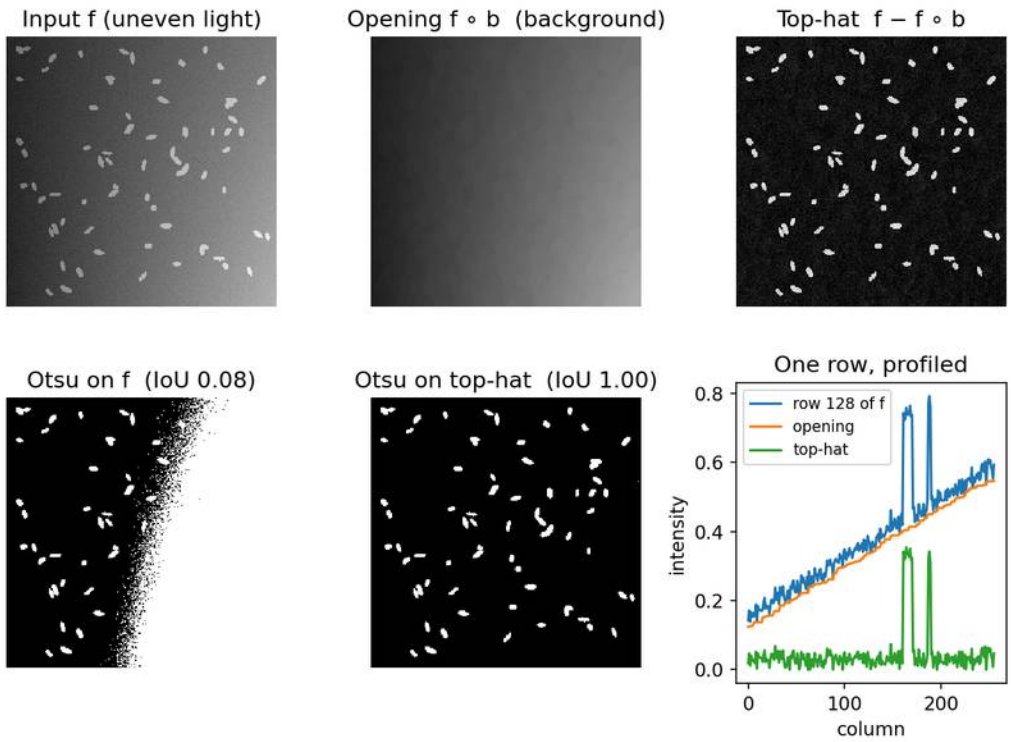
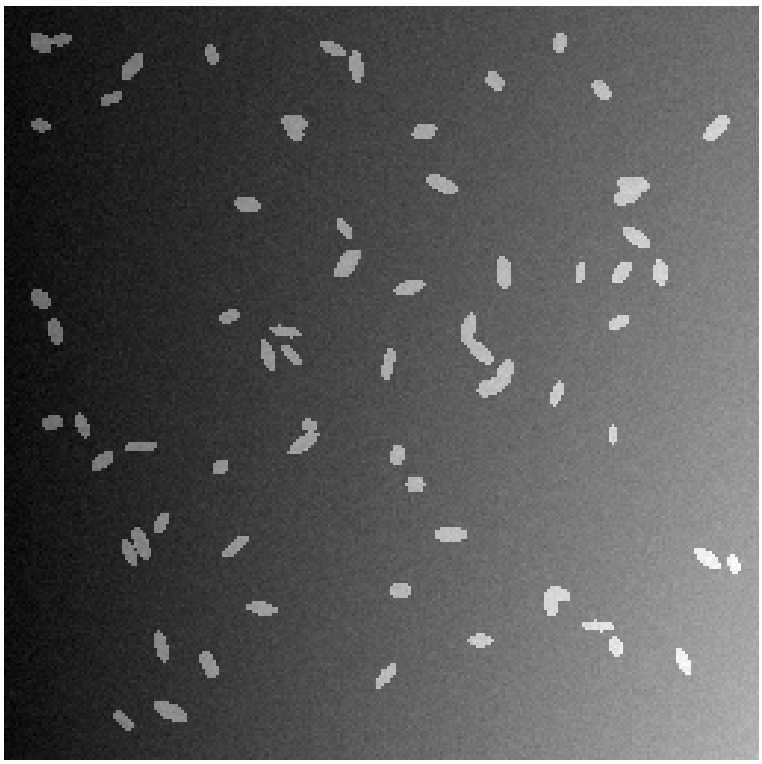


圖 *9.6* — 用頂帽轉換校正不均勻照明。以大於顆粒的圓盤做開運算可估計背景，相減後背景變平。*IoU* 是和已知的顆粒遮罩比較的結果。

拖動滑桿比較輸入與頂帽結果：



輸入



頂帽（圓盤 r=8）

不均勻照明下的顆粒，做白頂帽轉換前後的比較

```
from skimage import data, filters, morphology, util

coins = util.img_as_float(data.coins())
th = morphology.white_tophat(coins, morphology.disk(30))   # bright objects
bh = morphology.black_tophat(coins, morphology.disk(30))   # dark objects
mask = th > filters.threshold_otsu(th)
```

粒度測定

粒度測定（granulometry）不需要分割，就能量出影像中顆粒的大小分布。用越來越大的結構元素 r 對影像做開運算，每次開運算都會移除比 b_r 小的亮顆粒。記錄剩下的總亮度：

$$V(r) = \sum_{x,y} [f \circ b_r](x,y), \qquad r = 0, 1, 2, \dots,$$

其中 b_r 是半徑 r 的圓盤，b_0 是單一像素（所以 $V(0)$ 就是 f 的總和）。$V(r)$ 絕不會增加，因為越大的圓盤能放進的位置越少。它的負離散導數

$$PS(r) = V(r-1) - V(r)$$

是一種大小直方圖，稱為**圖樣頻譜**（pattern spectrum）[8]：在 r 處的值很大，表示有很多影像「質量」位於半徑約為 r 的特徵中。圖 9.7 的場景有兩種顆粒大小，頻譜也對應出現兩個峰。以大小為索引的一族開運算、以及「篩子」的詮釋，可以追溯到 Matheron [2]；Maragos 則把圖樣頻譜發展成多尺度形狀描述子 [8]。

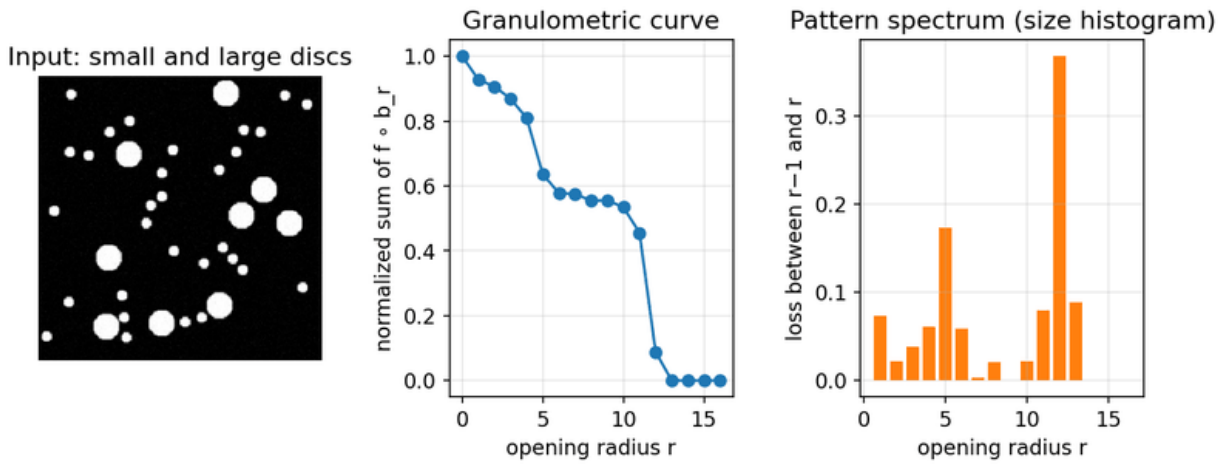


圖 *9.7* — 半徑 *5* 與 *12* 兩種圓盤的粒度測定。粒度曲線下降兩次，圖樣頻譜在兩個半徑處出現峰值。

```
import numpy as np
```

```
from skimage import morphology

def granulometry(f, radii):
    total = f.sum()
    vol = np.array([morphology.opening(f, morphology.disk(r)).sum() / total
                    for r in radii])
    spectrum = -np.diff(vol)                # loss from radius r-1 to r
    return vol, spectrum
```

僅線上版：紋理分割 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/紋理分割/*

僅線上版：灰階重建 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/灰階重建/*

現代觀點

形態學雖然歷史悠久，但遠未結束。有五條研究脈絡值得認識。

理論基礎與教學文獻。 Matheron 的隨機集合理論 [2] 與 Serra 的著作 [3] 建立了整套代數：侵蝕與膨脹是 Minkowski 運算、開運算是「篩子」。Haralick、Sternberg 與 Zhuang 1987 年的教學論文 [4] 至今仍是工程師入門最清楚的文獻之一；Soille 的 *Morphological Image Analysis* [5] 則是實務者查閱測地轉換、重建、分水嶺、粒度測定與高效實作的參考書。

僅線上版：形態學的理論基礎 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/papers/形態學的理論基礎/*

重建與高效演算法。 Vincent 1993 年的論文 [7] 讓重建變得實用。重點在於：天真的「迭代到穩定」迴圈可以換成以佇列為基礎、成本大致隨影像大小線性成長的演算法，於是重建成為填洞、區域極值與 h-dome 的日常工具。

連通運算子與元件樹。若一個濾波器只能移除或合併平坦區（數值相同的連通區域），而絕不產生新的輪廓，它就是**連通**（connected）的；以重建做開運算是最簡單的例子。Salembier 與 Wilkinson 的回顧論文 [10] 介紹了整個家族，其中包括 Breen 與 Jones 的屬性開運算 [9]：依面積或形狀，而不是「結構元素放不放得進」，來決定保留或移除整個元件。高效實作是在影像各閾值元件構成的樹，也就是**最大樹**（max-tree）[11] 上做濾波；scikit-image 提供了 `area_opening`、`area_closing`、`diameter_opening` 與 `max_tree`。

僅線上版：連通運算子與元件樹 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/連通運算子與元件樹/*

大小分布。 Maragos 的圖樣頻譜 [8] 把粒度測定形式化為多尺度形狀描述子，至今仍用於紋理與顆粒大小分析；以屬性為基礎的粒度測定 [9] 則把它推廣到大小以外的形狀屬性。

形態學遇上深度學習。深度學習從兩個方向改變了形態學。

網路內部的形態學。最大池化（max pooling）本來就是平坦灰階膨脹再加上降取樣，所以兩者的連結很自然。有好幾個團隊設計了結構元素可以用梯度下降學習的網路層 [13]、[14]；由於 min 與 max 的梯度很稀疏，後續研究改用平滑、可微的近似 [15]、[16]。這些仍是研究工具：形態學層並不是主流視覺骨幹網路的標準元件。

僅線上版：可學習的形態學層 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/morphology/notes/可學習的形態學層/*

網路周邊的形態學。在分割任務中，形態學一直被大量用作預測遮罩的**後處理**：用開運算去掉偽陽性斑點、用閉運算或填洞修補破碎的物件、`remove_small_objects`、清除邊界物件，以及用骨架量測管狀結構。它也出現在**損失函數**裡：clDice [17] 以反覆的 min 與 max 池化（軟侵蝕與軟膨脹）算出可微的「軟骨架」，再用預測骨架與真實骨架的重疊程度當作損失，鼓勵血管、道路與神經元的分割在拓撲上正確。

沒有改變的是：要清理、量測、校正二值或灰階影像，本章的經典運算子仍然是最快、最可預測的工具，而且它們附帶學習式濾波器無法提供的保證（冪等性、連通性、精確保持形狀）。

重點整理

- 形態學處理的是**形狀**，不是頻率。它只用最小值、最大值、聯集與交集，並由一個結構元素控制；結構元素的大小與形狀就是你提出的問題。
- **侵蝕**保留結構元素放得進去的位置；**膨脹**保留反射後的結構元素碰得到的位置。兩者互為對偶：侵蝕前景等於膨脹背景。
- **開運算**（先侵蝕後膨脹）去除小的亮部；**閉運算**（先膨脹後侵蝕）補上小的暗縫。兩者都是冪等濾波器，先開後閉是很穩健的雜訊清除法。
- **擊中擊不中轉換**比對精確的前景／背景圖樣，是細化、粗化與凸包的基礎。
- **重建**（標記加遮罩，長到穩定為止）能完整保留整個物件：以重建做開運算、自動填洞與清除邊界物件都由它而來。
- 在灰階影像中，侵蝕與膨脹就是局部最小值與最大值。**梯度**、**頂帽**、**底帽**分別用於邊緣、背景校正與小物件擷取；**粒度測定**不需分割就能得到大小分布。
- 今天形態學仍廣泛用於神經網路遮罩的後處理、clDice 這類損失中的可微積木，也是可學習形態學層的研究主題。

練習

1. 一張二值影像中有一個寬 40 像素、高 10 像素的矩形。分別描述它被 (a) 5×5 方塊、(b) 長 15 像素的水平線段、(c) 長 15 像素的垂直線段侵蝕與開運算的結果。哪個結構元素會把矩形完全移除？為什麼？

   提示

   侵蝕保留的是「整個結構元素放得進去」時原點的位置。5×5 方塊可放的位置構成 36×6 的區塊，而開運算會恢復整個矩形，因為矩形本身就是放得進去的方塊之聯集。水平線段也放得進去，所以開運算結果同樣是整個矩形。垂直線段比矩形還高，永遠放不進去：侵蝕與開運算都是空集合。

2. 證明集合的膨脹滿足交換律與結合律：$A \oplus B = B \oplus A$、$(A \oplus B) \oplus C = A \oplus (B \oplus C)$。用第二個等式解釋為什麼用 3×3 方塊膨脹兩次，等於用 5×5 方塊膨脹一次。侵蝕也有同樣的性質嗎？

   提示

把膨脹寫成 Minkowski 和 $\{a+b \mid a \in A, b \in B\}$，兩個等式都直接來自向量加法的交換律與結合律。$3\times 3$ 方塊加上自己就是 5×5 方塊。侵蝕則用 $(A \ominus B) \ominus C = A \ominus (B \oplus C)$，這可由對偶性推得。

3. 設計一組擊中擊不中模板 (B_1, B_2)，偵測一條 8 連通、一個像素寬、往右延伸到盡頭（線從左邊過來）的線段**端點**。要抓到所有端點，總共需要幾個旋轉後的模板？用 `scipy.ndimage.binary_hit_or_miss` 在 `skimage.morphology.skeletonize(~skimage.data.horse())` 上驗證你的答案。

   提示

   中心與左鄰居放在 B_1，其他七個鄰居放在 B_2。線也可能從斜方向過來，所以需要 8 個模板（4 個軸向加 4 個斜向）。把你數到的總數和修剪程式碼中 `endpoints()` 這類「數鄰居」函式的結果比較。

4. 你的分割網路輸出細胞遮罩。許多遮罩有小洞、少數單像素的偽陽性，還有一些細胞被影像邊界切到。用 scikit-image 寫一個五行的後處理流程，並說明步驟順序的理由。

   提示

   一個合理的順序是：先 `remove_small_objects`（或小的開運算），避免斑點被填起來或被合併；接著 `binary_fill_holes`；輪廓參差時再做一次小的閉運算；然後 `clear_border`；最後用 `label` 計數。討論一下如果先做閉運算再去斑點會出什麼問題（相鄰的斑點可能合併成一個「細胞」）。

5. 對灰階影像 f 與平坦結構元素 b，證明頂帽 $T(f) = f - (f \circ b)$ 非負，而且是冪等的：$T(T(f)) = T(f)$。

   提示

   非負性來自 $f \circ b \le f$。冪等性則要證明 $T(f) \circ b = 0$。任取 b 的一個平移 W，令 y 為 W 中 f 最小的點。開運算在 W 上處處至少是 $\min_W f$，又不超過 f，所以在 y 處恰等於 $f(y)$，因此 $T(f)(y) = 0$。於是 b 的每個平移都含有 $T(f)$ 的零點，$T(f)$ 被 b 侵蝕後為零，開運算也是零。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 9. publisher page
2. G. Matheron, *Random Sets and Integral Geometry*, Wiley, 1975. library record
3. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, 1982. library record
4. R. M. Haralick, S. R. Sternberg, and X. Zhuang, “Image Analysis Using Mathematical Morphology,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. PAMI-9, pp. 532–550, 1987. doi
5. P. Soille, *Morphological Image Analysis*, Springer, 2004. doi
6. T. Y. Zhang and C. Y. Suen, “A Fast Parallel Algorithm for Thinning Digital Patterns,” *Communications of the ACM*, vol. 27, pp. 236–239, 1984. doi

7. L. Vincent, “Morphological Grayscale Reconstruction in Image Analysis: Applications and Efficient Algorithms,” *IEEE Trans. Image Processing*, vol. 2, pp. 176–201, 1993. doi
8. P. Maragos, “Pattern Spectrum and Multiscale Shape Representation,” *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 701–716, 1989. doi
9. E. J. Breen and R. Jones, “Attribute Openings, Thinnings, and Granulometries,” *Computer Vision and Image Understanding*, vol. 64, no. 3, pp. 377–389, 1996. doi
10. P. Salembier and M. H. F. Wilkinson, “Connected Operators: A Review of Region-Based Morphological Image Processing Techniques,” *IEEE Signal Processing Magazine*, vol. 26, no. 6, pp. 136–157, 2009. doi
11. E. Carlinet and T. Géraud, “A Comparative Review of Component Tree Computation Algorithms,” *IEEE Trans. Image Processing*, vol. 23, pp. 3885–3895, 2014. doi
12. S. van der Walt et al., “scikit-image: Image Processing in Python,” *PeerJ*, 2014. arXiv
13. R. Mondal, S. Santra, S. S. Mukherjee, and B. Chanda, “Morphological Network: How Far Can We Go with Morphological Neurons?,” BMVC, 2022. arXiv
14. K. Nogueira, J. Chanussot, M. Dalla Mura, and J. A. dos Santos, “An Introduction to Deep Morphological Networks,” arXiv:1906.01751, 2019. arXiv
15. R. Hermary, G. Tochon, É. Puybareau, A. Kirszenberg, and J. Angulo, “Learning Grayscale Mathematical Morphology with Smooth Morphological Layers,” *Journal of Mathematical Imaging and Vision*, vol. 64, pp. 736–753, 2022. doi
16. A. Kirszenberg, G. Tochon, É. Puybareau, and J. Angulo, “Going Beyond p-convolutions to Learn Grayscale Morphological Operators,” arXiv:2102.10038, 2021. arXiv
17. S. Shit et al., “clDice — A Novel Topology-Preserving Loss Function for Tubular Structure Segmentation,” CVPR, 2021. arXiv

IP 10・影像分割（一）：邊緣偵測、閾值化與區域偵測

把影像切成有意義的部分：用梯度、LoG 與 Canny 找邊緣，用 Hough 轉換串接邊緣，用 Otsu 法選閾值，再以成長、分群、圖切割與淹水來找出區域。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/

你將學到

- 影像分割（image segmentation）的正式定義，以及兩大類方法：找**不連續**（邊緣）與找**相似**（區域）。
- 導數如何找出點、線與邊緣，以及 Sobel、Marr–Hildreth（LoG）和 Canny 偵測器的每個步驟。
- 如何把零散的邊緣像素串成邊界，包括用 Hough 轉換。
- Otsu 法如何自動選出閾值，以及單一閾值不夠用時該怎麼辦。
- 區域成長、分裂合併、k-means、SLIC、正規化切割、分水嶺與運動資訊，如何把像素變成區域。

先看全貌

到目前為止，大多數運算都是輸入一張影像、輸出另一張影像。影像分割輸出的則是一張**部件地圖**：每個像素都拿到一個標籤，說明它屬於哪個物體或區域。量測物體、計數細胞、辨識文字，全都仰賴這張地圖正不正確。

> **用最簡單的話說**
>
> 想像一張著色本的圖，黑色輪廓線全部不見了。你還是有兩種辦法分出各個部分。第一種：找顏色突然改變的地方，在那裡畫一條線，這就是**邊緣偵測**。第二種：從某一點開始，只要紙上的顏色看起來一樣，就一直用同一枝蠟筆塗開，這就是**區域成長**。本章就是在講這兩招，以及怎麼讓它們在雜亂的真實照片上也管用。

本章的方法都不需要從標註資料學習。它們不用訓練資料、容易解釋，而且其中的想法（梯度、遲滯、類間變異數、圖切割、淹水）至今仍活在現代深度學習流程裡。

基本概念

白話版：分割就是把影像切成彼此不重疊、合起來涵蓋全部、而且每一塊內部「看起來一樣」的幾塊。

令 R 為整張影像所佔的空間區域。分割是把 R 劃分成 n 個子區域 $R_1, \dots, R_n$，滿足

(a) $\bigcup_{i=1}^{n} R_i = R$, (b) R_i is connected,
(c) $R_i \cap R_j = \emptyset$ for $i \neq j$, (d) $Q(R_i) = \text{TRUE}$, (e) $Q(R_i \cup R_j) = \text{FALSE}$ for adjacent R_i, R_j.

其中 Q 是**邏輯述詞**（logical predicate）：對一組像素回答是或否的測試，例如「所有強度彼此相差不超過 10 個灰階」。也就是說：每個像素都有標籤 (a)、每個區域是連通的一整塊 (b)、沒有像素同時有兩個標籤 (c)、每個區域都通過測試 (d)、相鄰的兩個區域一旦合併就會不通過 (e)。

幾乎所有單色影像的分割演算法，都建立在強度的兩種性質之一 [1]：

- **不連續性**（discontinuity）：邊界就是強度突然改變的地方。點、線與邊緣偵測都靠它。
- **相似性**（similarity）：同一區域的像素共享某種性質（強度、顏色、紋理）。閾值化、區域成長、分群、圖切割與分水嶺都靠它。

邊緣定位精確但常常斷斷續續；區域天生是封閉的，但邊界可能歪掉。

點、線與邊緣偵測

白話版：邊緣就是亮度變化很快的地方，而「變化有多快」正是導數在量的東西。

背景：像素格上的導數

一維數位函數 $f(x)$ 最簡單的近似是

$$\frac{\partial f}{\partial x} \approx f(x+1) - f(x), \qquad \frac{\partial^2 f}{\partial x^2} \approx f(x+1) + f(x-1) - 2f(x).$$

其中 x 是整數像素位置，$f(x)$ 是它的強度。走過一段斜坡時，一階導數在整段斜坡上都不為零，因此響應**很粗**。二階導數只在斜坡兩端不為零、而且正負相反，形成**雙重響應**，中間夾著一個**零交越**（zero crossing）；它對細節與雜訊也敏感得多。後面所有偵測器都從這幾個事實出發。

偵測孤立點

點是一個和周圍不一樣的小斑點。拉普拉斯運算子（Laplacian）

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

可用中心為 -8、八個鄰居都是 1 的 3×3 核（含對角線的版本）實作。令 $Z(x,y)$ 為濾波結果，凡是 $|Z(x,y)| > T$（T 為非負閾值）的位置就標為點。核的係數總和為零，所以平坦區域沒有響應。

偵測線

線是寬度只有一到數個像素的結構。拉普拉斯運算子也會對線起反應，但會產生雙重響應，而且沒有方向性。要找特定角度的線，就用一個沿該角度排著 2、其餘為 -1 的核（水平線的核中間那

一列是 2, 2, 2）。比較水平、+45°、垂直與 −45° 四個核的響應，就能知道每個像素上哪個方向最強。

邊緣模型

真實的邊緣可理想化成三種形狀：

- **階梯邊緣**（step edge）：在一個像素內從一個灰階跳到另一個，多出現在合成影像；
- **斜坡邊緣**（ramp edge）：在數個像素內線性變化，這是光學與取樣造成的模糊在真實影像中的樣子；
- **屋脊邊緣**（roof edge）：先升後降，像一條被模糊過的細線。

在斜坡上，一階導數的大小告訴我們有邊緣，二階導數的正負告訴我們哪一側比較亮，而它的零交越標出斜坡中心。問題是，肉眼幾乎看不見的雜訊就足以毀掉二階導數，所以**先平滑再微分**是所有像樣的邊緣偵測器的一部分。

用梯度做基本邊緣偵測

f 在 (x, y) 的**梯度**（gradient）是向量

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \partial f/\partial x \\ \partial f/\partial y \end{bmatrix}, \qquad M(x,y) = \sqrt{g_x^2 + g_y^2}, \qquad \alpha(x,y) = \operatorname{atan2}(g_y, g_x).$$

g_x、g_y 是兩個方向的偏導數，M 是**梯度大小**（邊緣強度），α 是**梯度方向**（強度上升最快的方向）；邊緣本身與 α 垂直。實務上常用較省計算的 $|g_x| + |g_y|$ 來近似 M。

導數用小型核計算，經典運算子整理於 [1]。**Roberts** 交叉運算子使用 2×2 的對角差分；**Prewitt** 運算子使用列為 $[-1, 0, 1]$ 的 3×3 核；**Sobel** 運算子則把中間一列加權為 2，例如

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix},$$

也就是一個方向做差分、另一個方向乘上 $[1, 2, 1]$ 的平滑。正因為內建平滑，Sobel 通常比 Prewitt 更受青睞。

```
import cv2
import numpy as np
from skimage import data

f = data.camera().astype(np.float32) / 255.0
gx = cv2.Sobel(f, cv2.CV_32F, 1, 0, ksize=3)   # d/dx (columns)
gy = cv2.Sobel(f, cv2.CV_32F, 0, 1, ksize=3)   # d/dy (rows)
M = np.hypot(gx, gy)                           # gradient magnitude
alpha = np.arctan2(gy, gx)                     # gradient direction, radians
edges = M > 0.3 * M.max()                      # crude thresholded edge map
```

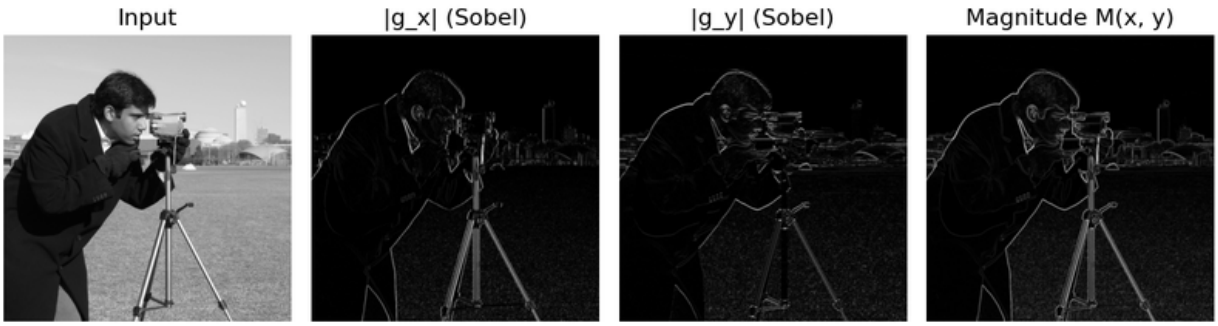


圖 *10.1 — Sobel* 導數。|g_x| 會凸顯垂直結構（例如三腳架的腳），|g_y| 會凸顯水平結構（例如地平線），梯度大小 M 則結合兩者。注意草地的紋理也有響應。

自己動手試試 Sobel 核：換成 y 方向的版本，或把 2 改成 1 變成 Prewitt，看看響應怎麼變。

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

直接對 M 做閾值化得到的邊緣圖並不好：邊緣太粗、弱邊緣消失、紋理造成雜亂。接下來兩個偵測器把平滑與細化直接設計進去。

Marr–Hildreth 邊緣偵測器

Marr 與 Hildreth [2] 主張強度變化發生在許多尺度上，所以偵測器的大小必須可調；而二階導數的零交越正好標出邊緣中心。他們的運算子是**高斯的拉普拉斯**（Laplacian of a Gaussian, LoG）$\nabla^2 G$，其中

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}, \qquad \nabla^2 G(x,y) = \frac{x^2+y^2-2\sigma^2}{\sigma^4}\, e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

σ 是高斯的標準差，決定尺度（σ 越大，只有越粗的邊緣留下來）；因為只在乎零交越，所以省略了正規化常數。由於形狀的關係，$\nabla^2 G$ 又稱為**墨西哥帽**（Mexican hat）。演算法如下：

1. 用 $n \times n$ 的高斯核平滑，n 取不小於 6σ 的最小奇數。
2. 對結果取拉普拉斯（由於線性，步驟 1–2 等於與 $\nabla^2 G$ 做一次卷積）。
3. 標出**零交越**：某一對相對鄰居的正負號不同，且差的絕對值超過閾值的像素。沒有這個閾值，平坦區域的每個小波動都會變成「邊緣」。

高斯差（difference of Gaussians, DoG）$G_{\sigma_1} - G_{\sigma_2}$，取 $\sigma_1/\sigma_2 \approx 1.6$，可以近似 LoG [1]。零交越會形成封閉輪廓，但在紋理處會產生一團「義大利麵」，也會把角落磨圓。

Canny 邊緣偵測器

Canny [3] 把邊緣偵測寫成一個最佳化問題，有三個準則：

1. **低錯誤率**：找到所有真正的邊緣，不產生假邊緣。

2. **定位準確**：偵測到的邊緣要盡量貼近真正的邊緣。
3. **單一響應**：一條真正的邊緣只產生一條偵測結果，而不是雙線。

對白雜訊中的一維階梯邊緣，最佳濾波器可以用高斯的一階導數很好地近似。實用的二維演算法如下：

1. 用標準差為 σ 的高斯**平滑**影像：$f_s = G_\sigma \star f$。
2. **計算** f_s 的梯度大小 M 與角度 α。
3. **非極大值抑制**（non-maximum suppression）。把 α 量化成四個方向；若某像素的 M 小於梯度方向上兩個鄰居中的任一個，就設為零。脊線因此只剩一個像素寬。
4. **雙閾值**。選 T_H 與 T_L，T_H/T_L 約為 2 到 3 [1]。高於 T_H 的是**強**邊緣，介於兩者之間的是**弱**邊緣。
5. **遲滯**（hysteresis）。弱像素只有在透過其他弱像素連到強像素時才保留。

遲滯是關鍵。單一閾值不是把輪廓切斷，就是放進雜訊；雙閾值加上連通性，能保留掛在強邊緣上的微弱輪廓，同時丟掉孤立的弱響應。

```
import cv2
from scipy import ndimage as ndi
from skimage import data, feature

f8 = data.camera()
canny_cv = cv2.Canny(cv2.GaussianBlur(f8, (0, 0), 2), 30, 90)      # uint8 0/255
canny_sk = feature.canny(f8 / 255.0, sigma=2, low_threshold=0.05, high_threshold=0.15)
log = ndi.gaussian_laplace(f8 / 255.0, sigma=3)                    # Marr-Hildreth steps 1-2
```

`cv2.Canny` 沒有平滑參數，所以我們先自己模糊；scikit-image 的版本則直接接受 `sigma`。

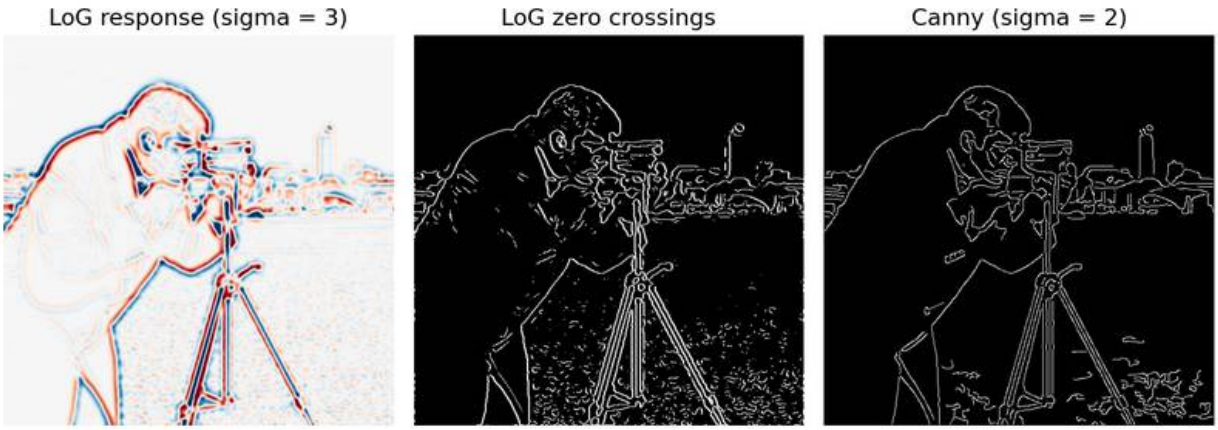


圖 *10.2* — 同一張影像上的 *Marr–Hildreth* 與 *Canny*。*LoG* 響應（左，紅為正、藍為負）在每條邊緣處變號；經過閾值的零交越（中）是封閉的，但在草地上很雜亂。*Canny*（右）給出細而大多連續的邊緣，雜訊少得多。

拖動滑桿，看看 Canny 留下了哪些結構（邊緣以白底黑線顯示）。



輸入



Canny σ=2

攝影師影像與其 Canny 邊緣圖的比較

串接邊緣點

即使是 Canny 也會留下缺口。**邊緣串接**（edge linking）把邊緣像素組成有意義的邊界。

局部處理。最簡單的想法是：把梯度大小與方向都相近的相鄰邊緣像素連起來，讓同一條輪廓上的像素接在一起，無關的像素則保持分開。

僅線上版：局部邊緣串接 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/notes/局部邊緣串接/*

以 Hough 轉換做全域處理。若已知形狀，就讓每個邊緣像素替所有可能通過它的形狀投票。對直線，Duda 與 Hart [4] 提出**法線表示法**

$$\rho = x\cos\theta + y\sin\theta,$$

其中 θ 是直線法線的角度，ρ 是原點到直線的有號距離；和斜率–截距不同，垂直線時它仍然有界。每個點 (x_k, y_k) 在 $\rho\theta$ 平面上變成一條正弦曲線，共線點的曲線會交在同一格。演算法如下：

1. 計算二值邊緣圖（例如用 Canny）。
2. 把 $\rho\theta$ 平面量化成**累加器格**（accumulator cells）$A(\rho, \theta)$，全部設為零。
3. 對每個邊緣像素和每個 θ，算出 ρ 並把 $A(\rho, \theta)$ 加一。
4. 計數大的局部極大值就是直線；也可以檢查投票像素的連續性，把直線切成線段。

```
# continues from the Canny snippet above (uses cv2, np and canny_cv)
import numpy as np

lines = cv2.HoughLines(canny_cv, rho=1, theta=np.pi / 180, threshold=120)
for rho, theta in lines[:5, 0]:
    print(f"rho = {rho:6.1f} px, theta = {np.degrees(theta):5.1f} deg")
```

OpenCV 回傳的 $\theta \in [0°, 180°)$，ρ 可為負，和上面是同一種表示法，只是範圍不同。投票的想法也能延伸到圓與其他參數更多的形狀。

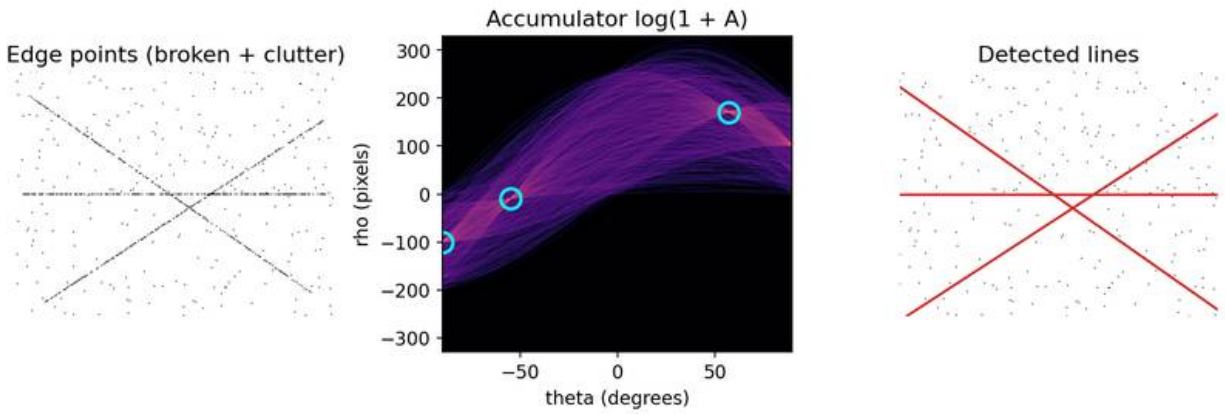


圖 *10.3 — Hough* 轉換。三條缺了 *35%* 像素的直線，外加隨機雜點（左）。每個點沿一條正弦曲線投票；累加器中圈起來的三個峰值（中）是大量曲線交會之處。把峰值換回直線，即使有缺口也能找回三條線（右）。

閾值化

白話版：選一個灰階，比它亮的是「物體」，比它暗的是「背景」。學問全在怎麼選。

基礎

給定強度閾值 T，全域閾值化（thresholding）得到

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{otherwise.} \end{cases}$$

T 對整張影像固定時稱為**全域**閾值；隨位置改變時稱為**可變**閾值（也叫局部或自適應）；用兩個以上閾值則稱為**多重閾值化**。當直方圖的各個峰分得很開時，閾值化效果很好。**雜訊**會把峰變寬，**不均勻的照明與反射**會讓峰在影像各處移動，而很小的物體只會在直方圖上形成一個容易被忽略的小突起。

基本全域閾值化

當峰分得很清楚時，下面這個簡單的疊代規則就很好用：

1. 選一個初始 T，例如平均強度。
2. 把像素分成 G_1（值 $> T$）與 G_2（值 $\le T$）。
3. 計算兩組的平均 m_1、m_2。
4. 令 $T = \tfrac{1}{2}(m_1 + m_2)$。
5. 重複 2–4，直到 T 的變化小於某個 ΔT。

```
def basic_global(img, dT=0.5):
    T = img.mean()
    while True:
        m1, m2 = img[img > T].mean(), img[img <= T].mean()
        T_new = 0.5 * (m1 + m2)
        if abs(T_new - T) < dT:
            return T_new
        T = T_new
```

Otsu 最佳全域閾值

Otsu [5] 選的是讓兩類**可分離性**最大的閾值，以**類間變異數**（between-class variance）衡量。它只需要直方圖，因此很快。

設影像有 L 個灰階，灰階 i 有 n_i 個像素，總像素數為 MN，正規化直方圖為 $p_i = n_i/MN$。閾值 k 把灰階分成 $C_1 = [0,k]$ 與 $C_2 = [k+1, L-1]$。定義

$$P_1(k) = \sum_{i=0}^{k} p_i, \qquad m(k) = \sum_{i=0}^{k} i\,p_i, \qquad m_G = \sum_{i=0}^{L-1} i\,p_i,$$

其中 $P_1(k)$ 是像素落在 C_1 的機率，$m(k)$ 是到灰階 k 為止的累積平均，m_G 是全域平均。於是 $P_2 = 1 - P_1$，兩類的平均為 $m_1 = m/P_1$、$m_2 = (m_G - m)/P_2$。

類間變異數是兩類平均在全域平均周圍的加權散佈。它可以化簡成兩類機率的乘積，乘上兩類平均差距的平方：

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 = P_1 P_2 (m_1 - m_2)^2.$$

兩類平均離得越遠，σ_B^2 越大。再代入 $m_1 = m/P_1$ 與 $m_2 = (m_G - m)/(1 - P_1)$，就得到只需要累積和的形式：

$$\sigma_B^2(k) = \frac{\big(m_G P_1(k) - m(k)\big)^2}{P_1(k)\,\big(1 - P_1(k)\big)}.$$

最佳閾值為 $k^* = \arg\max_k \sigma_B^2(k)$（若有多個並列就取平均）。由於全域變異數 σ_G^2 等於類間與類內變異數之和，而且與 k 無關，所以這同時也**最小化類內變異數**。比值

$$\eta = \frac{\sigma_B^2(k^*)}{\sigma_G^2}, \qquad 0 \le \eta \le 1,$$

衡量**可分離性**：兩個乾淨的峰時接近 1，峰重疊時較低。

僅線上版：推導類間變異數 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/notes/推導類間變異數/*

```
import numpy as np
from skimage import data, filters

def otsu(img):
    p = np.bincount(img.ravel(), minlength=256) / img.size   # normalized histogram
    P1 = np.cumsum(p)                                         # class-1 probability
    m = np.cumsum(np.arange(256) * p)                         # cumulative mean
    mG = m[-1]                                                # global mean
    with np.errstate(divide="ignore", invalid="ignore"):
        sB2 = (mG * P1 - m) ** 2 / (P1 * (1 - P1))
    sB2 = np.nan_to_num(sB2)
    k = int(np.argmax(sB2))
    return k, sB2[k] / np.var(img)                            # threshold, separability

coins = data.coins()
k, eta = otsu(coins)          # k = 107, eta ≈ 0.76
assert k == filters.threshold_otsu(coins)
```

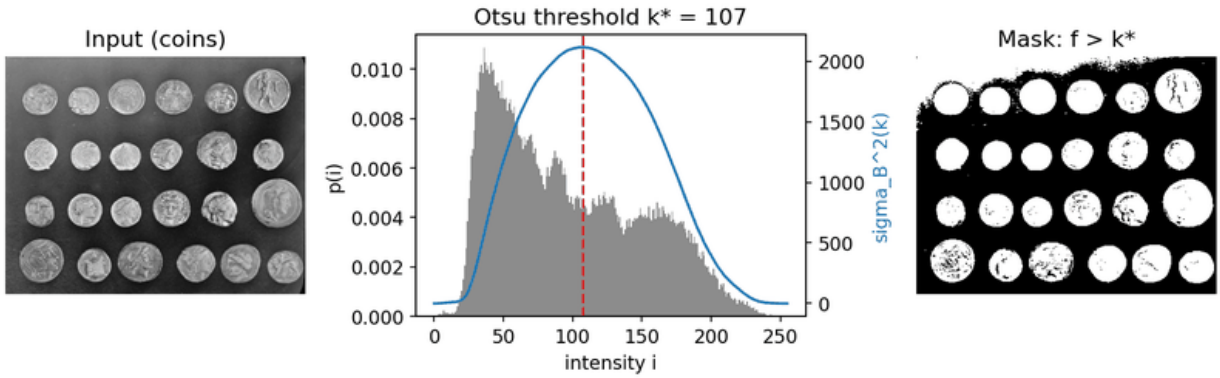


圖 *10.4* — 硬幣影像上的 *Otsu* 法。藍色曲線是 $\sigma_B^2(k)$，其最大值給出 $k^* = 107$（虛線）。遮罩（右）抓到了硬幣，但上緣那條較亮的背景帶也被選進來，提醒我們單一全域閾值無法修正不均勻照明。

用平滑與邊緣改善閾值化

- **先平滑**。雜訊會讓峰變寬直到合併；平滑可以讓峰再變窄，前提是物體比濾波器大得多。
- **只用邊緣附近的像素**。小物體的峰會被背景的峰淹沒。保留邊緣指標（梯度大小或 $|\nabla^2 f|$）很高的像素，例如高於第 99.7 百分位數者，並**只用這些像素**計算直方圖。它們大約一半落在邊界兩側，因此直方圖變得平衡而雙峰。再把得到的閾值套用到整張影像。

僅線上版：多重閾值 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/notes/多重閾值/*

可變閾值化

照明不均時，沒有任何單一 T 管用。三種常見對策：

1. **分塊**。每一塊用自己的 Otsu 值做閾值化；每一塊都必須同時含有兩類。
2. **局部性質**。令 m_{xy}、σ_{xy} 為 (x, y) 周圍視窗的平均與標準差，使用 $T_{xy} = a\,\sigma_{xy} + b\,m_{xy}$（$a, b \geq 0$），或更一般的述詞 $Q(\sigma_{xy}, m_{xy})$。Sauvola 與 Pietikäinen 的文件二值化規則 [7]，$T_{xy} = m_{xy}(1 + k(\sigma_{xy}/R - 1))$，$R$ 為 σ 的動態範圍、k 為小正數，就屬於這一族。
3. **移動平均**。對文字影像以之字形掃描，以 $c \cdot m(k)$ 做閾值，其中 $m(k)$ 是最近 n 個像素的平均，c 略小於 1。

```
from skimage import data, filters

page = data.page()
global_mask = page > filters.threshold_otsu(page)
local_T = filters.threshold_local(page, block_size=35, offset=10)  # Gaussian-weighted local
 ↪  mean - 10
local_mask = page > local_T
sauvola_mask = page > filters.threshold_sauvola(page, window_size=25, k=0.2)
```

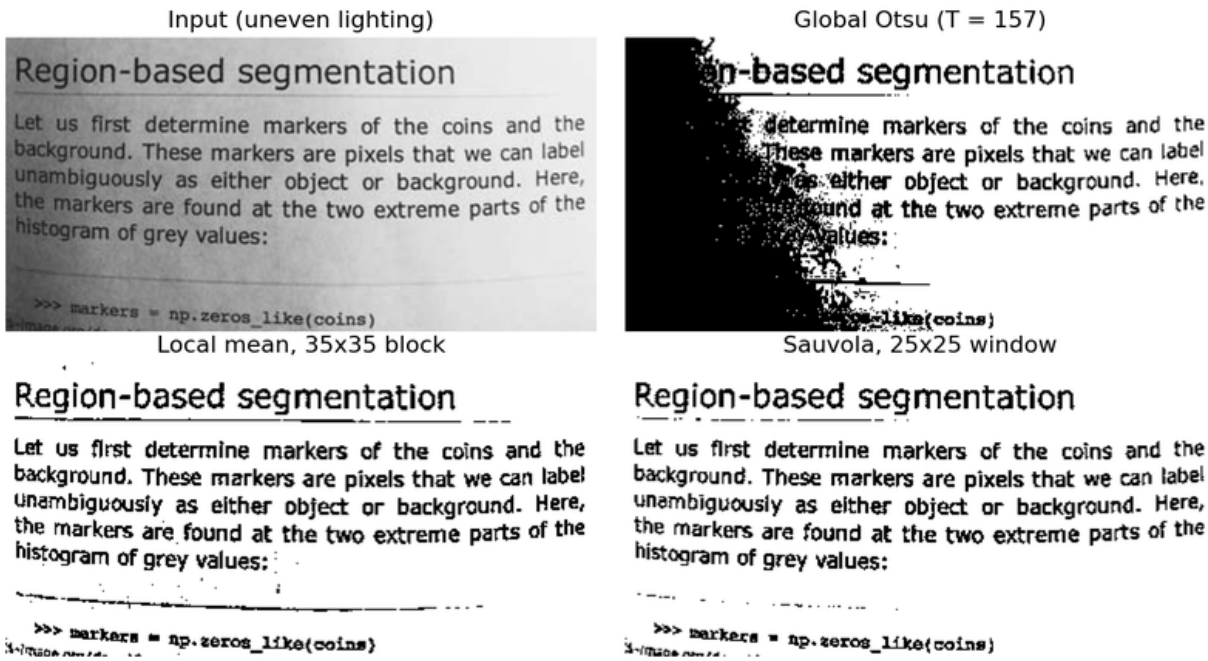


圖 *10.5* — 照明不均文字的可變閾值化。全域 *Otsu* 閾值（右上）把陰影中的左半邊整片塗黑；局部平均閾值（左下）與 *Sauvola* 規則（右下）則在各處都救回了文字。

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

區域成長、分裂與合併

白話版：從幾個「種子」像素出發，不斷加入長得像它們的鄰居；或者從整張影像出發，不斷切成四塊，直到每一塊都均勻為止。

區域成長

區域成長（region growing）把滿足相似性述詞的鄰近像素加入每個**種子**。需要決定的有：**種子**（人工指定、用很保守的閾值選出，或取自直方圖峰值）、**述詞**（例如 $|f(x,y) - f(\text{seed})| \le T$，或對區域統計量的測試）、**連通性**（4 或 8 連通），以及**停止規則**（沒有鄰居通過測試時停止；加上大小或形狀限制會更穩健）。

```
from collections import deque
import numpy as np

def region_grow(img, seed, tol):
    """Grow from `seed` over 8-connected pixels whose value is within `tol` of the seed
    ↪  value."""
```

```
    H, W = img.shape
    ref = float(img[seed])
    mask = np.zeros((H, W), bool)
    mask[seed] = True
    q = deque([seed])
    while q:
        r, c = q.popleft()
        for dr in (-1, 0, 1):
            for dc in (-1, 0, 1):
                rr, cc = r + dr, c + dc
                if 0 <= rr < H and 0 <= cc < W and not mask[rr, cc] \
                        and abs(float(img[rr, cc]) - ref) <= tol:
                    mask[rr, cc] = True
                    q.append((rr, cc))
    return mask
```

`skimage.segmentation.flood(img, seed, tolerance=tol)` 以編譯過的程式碼做同樣的事。

僅線上版：以四元樹做區域分裂與合併 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/notes/以四元樹做區域分裂與合併/*

以分群與超像素做區域分割

白話版：把每個像素當成用幾個數字（顏色、或許加上位置）描述的點，再把靠得近的點聚成一群。

k-means 分群

把每個像素表示成特徵向量 $\mathbf{z} \in \mathbb{R}^n$，例如它的 Lab 顏色（$n = 3$）。**k-means** 尋找 k 個群中心 $\mathbf{m}_1, \dots, \mathbf{m}_k$ 與集合 $C_1, \dots, C_k$，使

$$J = \sum_{i=1}^{k} \sum_{\mathbf{z} \in C_i} \|\mathbf{z} - \mathbf{m}_i\|^2$$

最小，也就是到群平均的總平方距離。求全域最小值是 NP-hard，所以標準演算法交替進行：把每個向量指派給最近的中心，再把每個中心移到其成員的平均。它只會收斂到局部最小值，因此要從多個初始值執行。只用顏色分群會忽略位置，所以同一群可能散布在整張影像上。

超像素與 SLIC

超像素（superpixel）是一小塊緊密、相似且連通的像素。用幾百個超像素取代 10^5 個像素，可以讓後續問題變小，又不會移動重要的邊界。Achanta 等人的 **SLIC** [8] 就是在五維空間 $[l, a, b, x, y]$ 中做 k-means。設有 N 個像素、想要 K 個超像素，格距為 $S = \sqrt{N/K}$。

1. 在格距為 S 的網格上放 K 個中心，每個再移到其 3×3 鄰域中梯度最小的位置。

2. 每個中心**只搜尋周圍** $2S \times 2S$ **的視窗**，依下列距離指派像素

$$D = \sqrt{d_c^2 + \left(\frac{d_s}{S}\right)^2 m^2},$$

其中 d_c、d_s 分別是 Lab 顏色與 (x, y) 上的歐氏距離，m 是**緊緻度**（compactness）：m 大時超像素接近方形，m 小時會緊貼顏色邊緣。3. 把每個中心移到其像素的平均；重複約十次。4. **強制連通性**：把零星碎片併入相鄰的超像素。

局部搜尋讓每次疊代的成本為 $O(N)$，與 K 無關。

```
from skimage import data, segmentation

img = data.astronaut()
sp = segmentation.slic(img, n_segments=300, compactness=10, start_label=1)
```

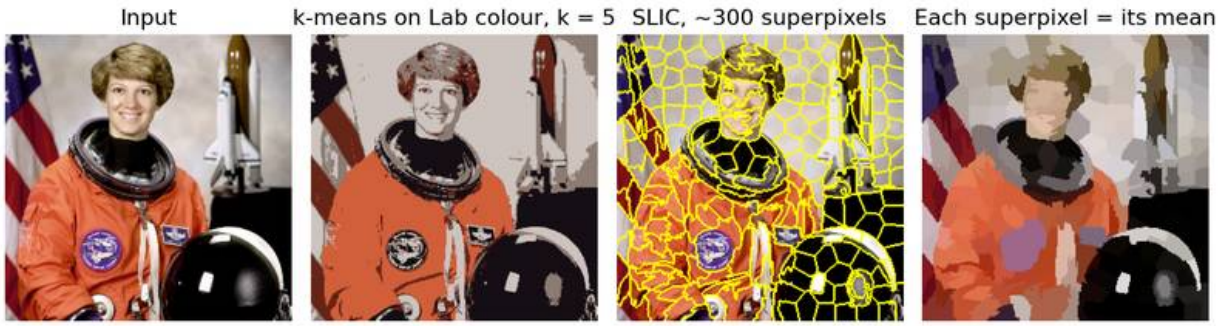


圖 *10.6* — 分群。在 *Lab* 顏色上做 *k-means*（第二張）把影像簡化成五種顏色，但每一群都散布在畫面各處。*SLIC*（第三張，緊緻度 *10*）產生約 *300* 個沿著強邊緣的緊密超像素；把每個超像素塗上其平均色（右），只用極少的單位就保留了可辨識的畫面。

以圖切割做區域分割

白話版：把影像畫成一張網，相鄰像素之間綁著繩子，像的就綁粗繩、不像的就綁細繩。分割就是只剪斷細繩，把網分成幾塊。

把影像視為圖

圖（graph）$G = (V, E)$ 由節點 V 與邊 E 組成。每個像素（或超像素）是一個節點，鄰近節點之間以非負的相似度權重 $w(i, j)$ 相連，例如

$$w(i,j) = \exp\!\left(-\frac{\|\mathbf{F}_i - \mathbf{F}_j\|^2}{\sigma_I^2}\right)\exp\!\left(-\frac{\|\mathbf{X}_i - \mathbf{X}_j\|^2}{\sigma_X^2}\right) \quad \text{if } \|\mathbf{X}_i - \mathbf{X}_j\| \le r, \text{ else } 0,$$

其中 $\mathbf{F}$ 是特徵向量（強度、顏色），$\mathbf{X}$ 是空間位置，σ_I、σ_X 是尺度參數，r 是讓圖保持稀疏的半徑。

最小圖切割

切割（cut）把 V 分成兩個不相交的集合 A、B，並移除兩者之間的邊。其成本為

$$\operatorname{cut}(A,B)=\sum_{u\in A,\,v\in B} w(u,v).$$

最小切割移除的相似度最少。一種常見的形式會加入一個**源點**（source，代表物體）與一個**匯點**（sink，代表背景），並以「多像物體、多像背景」的權重連到每個像素。最小源–匯切割會替每個像素貼上標籤，而且能用最大流演算法有效率地求得；使用者只要在物體與背景上各畫幾筆的互動式工具，就是以此為基礎。

正規化切割

沒有源點與匯點時，單純的最小切割偏好切下單一節點或極小的一群，因為要移除的邊很少。Shi 與 Malik [9] 以**正規化切割**（normalized cut）修正這個問題：

$$\operatorname{Ncut}(A,B)=\frac{\operatorname{cut}(A,B)}{\operatorname{assoc}(A,V)}+\frac{\operatorname{cut}(A,B)}{\operatorname{assoc}(B,V)},\qquad \operatorname{assoc}(A,V)=\sum_{u\in A,\,t\in V} w(u,t),$$

$\operatorname{assoc}(A,V)$ 是 A 到所有節點的總連結。極小的 A 其 assoc 也很小，因此切下它變得很貴。精確最小化是 NP-hard，但鬆弛後可化為特徵值問題；令 W 為權重矩陣、D 為度數 $d_i=\sum_j w(i,j)$ 構成的對角矩陣，

$$(D-W)\,\mathbf{y}=\lambda D\,\mathbf{y}.$$

第二小特徵值對應的特徵向量是一個連續的分割指標；對它做閾值化就能把圖一分為二，遞迴下去就得到更多區塊。實務上會在超像素圖上執行。

```
import numpy as np
from skimage import data, segmentation, graph

img = data.coffee()
sp = segmentation.slic(img, n_segments=400, compactness=30, start_label=1)
rag = graph.rag_mean_color(img, sp, mode="similarity")   # edge weight = exp(-|c_i - c_j|^2 /
 ↪  sigma)
ncut = graph.cut_normalized(sp, rag, rng=0)              # recursive two-way Ncut
print(sp.max(), "superpixels ->", len(np.unique(ncut)), "regions")
```

以形態學分水嶺做分割

白話版：把影像想成一片地形，越亮越高。從每個山谷灌水；兩個山谷的水快要相遇時，就築一道牆。這些牆就是分割結果。

背景：集水盆與水壩

把強度當成高度。每個**區域極小值**（regional minimum）都有一個**集水盆**（catchment basin），也就是水會流到它的所有點；**分水線**把相鄰的集水盆隔開。通常淹的是**梯度大小**而不是原影像：物體內部低、邊界是山脊，因此每個集水盆就是一個物體。

兩股水快要相遇時，演算法會築一道**水壩**：一道一個像素寬、設得比任何強度都高的牆，讓兩股水永遠不會合併。

分水嶺演算法

白話說：讓水位一次升高一個灰階。在每個水位，新淹到的一塊區域只有三種情況：開出新的集水盆（出現了新的極小值）、併入它碰到的那唯一一個集水盆，或同時碰到兩個以上的集水盆，這時就在它們之間築壩。Vincent 與 Soille [10] 提出今日通用的高效實作：先依高度排序像素，再以 FIFO 佇列逐層淹水（「浸沒模擬」，immersion simulation）。

僅線上版：逐步說明淹水演算法 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/notes/逐步說明淹水演算法/*

使用標記

直接用在梯度上，分水嶺會嚴重**過度分割**：雜訊與紋理造成成千上萬個極小值。解法是**標記**（markers），也就是事先確定位於物體內（**內部標記**）或背景中（**外部標記**）的連通成分。淹水只從標記開始，所以區域數就等於標記數。標記可以來自保守的閾值、平滑後的極小值、距離轉換的峰值，或使用者的點擊。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, filters, morphology, segmentation

coins = data.coins()
elevation = filters.sobel(coins.astype(float))
markers = np.zeros_like(coins, dtype=np.int32)
markers[coins < 30] = 1        # sure background
markers[coins > 150] = 2       # sure object
ws = segmentation.watershed(elevation, markers)
coins_mask = morphology.remove_small_objects(ndi.binary_fill_holes(ws == 2), 100)
labels, n = ndi.label(coins_mask)   # n = 24 coins
```

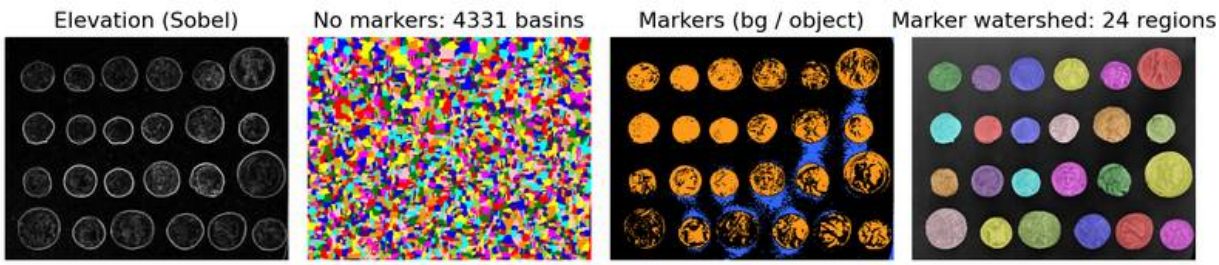


圖 *10.7* — 硬幣影像上的分水嶺。從每個區域極小值淹平滑後的梯度，會得到數千個集水盆（第二張）。用兩個非常保守的閾值產生兩種標記（第三張）後，水正好在硬幣邊緣相遇；填洞並去掉小斑點後，每枚硬幣各成一個區域（右）。

利用運動做分割

白話版：如果攝影機不動，兩張畫面之間有變的東西，就是你在意的東西。把兩張畫面相減，移動的物體就亮起來了。

空間技術：差異影像

給定靜止攝影機拍攝靜態場景的兩張畫面 $f(x,y,t_i)$ 與 $f(x,y,t_j)$，**差異影像**（difference image）為

$$d_{ij}(x,y) = \begin{cases} 1 & \text{if } |f(x,y,t_i) - f(x,y,t_j)| > T \\ 0 & \text{otherwise,} \end{cases}$$

T 取略高於雜訊水準的值。雜訊造成的小點，可以靠丟棄太小的連通成分去除。

累積差異

單一差異影像會把移動物體顯示兩次：它原本在的地方和現在在的地方。**累積差異影像**（accumulative difference image, ADI）把**參考畫面** $R = f(x,y,t_1)$ 與之後每張畫面比較並計數：

$$\begin{aligned} A_k &= A_{k-1} + 1 \ \text{ if } |R - f_k| > T, \\ P_k &= P_{k-1} + 1 \ \text{ if } R - f_k > T, \\ N_k &= N_{k-1} + 1 \ \text{ if } R - f_k < -T, \end{aligned}$$

其他情況維持不變，初始為零（$f_k = f(x,y,t_k)$）。若物體比背景亮，**正 ADI** P 標出物體在參考畫面中的位置；**負 ADI** 朝運動方向增長，增長速度取決於物體速度；**絕對 ADI** A 則兩者都包含。

```
import numpy as np

def accumulative_differences(frames, T):
    R = frames[0].astype(float)
    A = np.zeros(R.shape, int); P = np.zeros(R.shape, int); N = np.zeros(R.shape, int)
    for f in frames[1:]:
        d = R - f.astype(float)
        A += np.abs(d) > T
        P += d > T
        N += d < -T
    return A, P, N
```

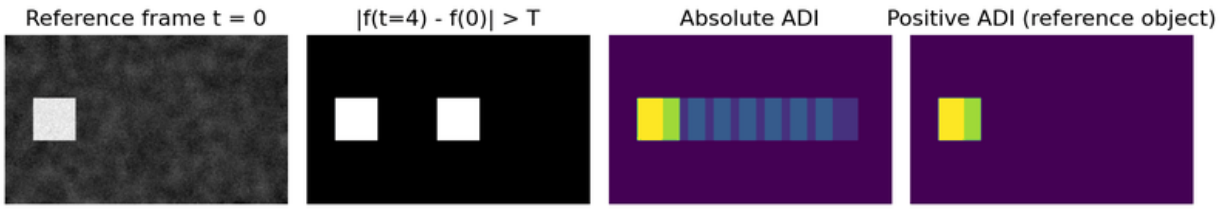


圖 *10.8* — 運動。一個亮方塊在八張畫面中向右移動。單一差異影像（第二張）把方塊顯示了兩次；絕對 *ADI*（第三張）記錄了它到過的每個位置；正 *ADI*（右）只留下它的起點。

僅線上版：建立參考影像 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/notes/建立參考影像/*

現代觀點

以下五篇綜述描繪了這個領域，並說明深度學習改變了什麼、沒改變什麼。

傳統閾值化。 Sezgin 與 Sankur [6] 依所利用的資訊，把閾值化方法分成六族（直方圖形狀、量測空間分群、熵、物體屬性、空間相關、局部灰階曲面），並在非破壞檢測與文件影像上評估其中四十種，挑出在兩個領域都表現穩定的方法。Otsu 法屬於分群這一族。

深度分割。 Minaee 等人 [11] 綜述深度學習的語意分割與實例分割，把模型分成全卷積網路、編碼器–解碼器、多尺度與金字塔模型、遞迴網路、注意力模型與對抗式生成模型，並回顧了資料集與結果。幾個里程碑設計都呼應著傳統想法：

- **FCN** [13] 把分類網路改成全卷積，讓它輸出標籤圖，並融合深層粗略特徵與淺層精細特徵：一種學出來的多尺度分析。
- **U-Net** [14] 加上對稱的擴張路徑與跳接（skip connection），用少量訓練資料就能得到精確邊界，成為生醫影像的預設選擇之一。選讀：*U-Net*：論文導讀 (*https://blog.shyandram.dev/zh-tw/learn/image-processing/segmentation/papers/u-net 論文導讀/*)（見「論文導讀」頁）。論文的主張，以及怎麼讀它的表格。
- **DeepLab** [16] 以空洞卷積（atrous convolution）與空洞空間金字塔池化兼顧解析度與脈絡，再用全連接 CRF 讓邊界更銳利。
- **Mask R-CNN** [15] 在物體偵測器上加一個遮罩分支，能分開同類別而互相接觸的物體，也就是分水嶺中標記所做的工作。

基礎模型與可提示分割。 Segment Anything（SAM）[17] 是可用點、框或遮罩當提示的模型，訓練於 SA-1B：1,100 萬張影像上的十億個以上遮罩。點一下、回傳一個遮罩，這正是種子區域成長與互動式圖切割的介面，只是相似度改由學習而來。Zhou 等人 [12] 回顧了這個「基礎模型時代」的三百多種方法，分為一般任務（語意、實例、全景分割）與可提示任務（互動式、指稱式、少樣本），並說明大型預訓練模型如何帶有分割知識。

邊緣偵測。 Sun 等人 [18] 把傳統邊緣偵測器分成梯度、高斯差、多尺度與結構化學習四類，把深度方法分成編碼器–解碼器、網路重構與多尺度融合三類。HED [19] 是關鍵的早期深度設計：把網路多個深度的側輸出融合成一張邊緣圖，可說是 Marr 與 Hildreth 多尺度論點的學習版後代。該綜述的結論是：學習式偵測器在標準評測上已接近甚至超越人類水準，剩下的開放問題是輕量

化模型、弱監督與可解釋性。

超像素。 Stutz、Hermans 與 Leibe [20] 評測了 28 種演算法，強調要調好參數並**嚴格強制連通性**，提出與超像素數量無關的指標，並測試對雜訊、模糊與仿射轉換的穩健性。

改變了什麼、沒改變什麼。學習出的特徵取代了手工設計的述詞與邊緣強度，在自然影像上尤其明顯。但傳統的骨架還在：網路輸出的是機率圖，仍然要做閾值化；互相接觸的實例仍需要分開，靠的是偵測框、學出來的標記，或在預測的距離圖上跑分水嶺；而在資料稀少、運算受限或行為必須可預測的場合，例如文件掃描與工業檢測，Otsu、Canny 與 Hough 仍然勝出。

重點整理

- 分割把影像劃分成連通、不重疊、各自滿足述詞 Q 的區域；方法不是找**不連續**（邊緣），就是找**相似**（區域）。
- 一階導數給出粗的邊緣響應；二階導數給出雙重響應，邊緣中心是零交越，而且對雜訊非常敏感，所以要先平滑再微分。
- Canny = 高斯平滑 + 梯度 + 非極大值抑制 + 帶**遲滯**的雙閾值；最後一步讓微弱但連通的邊緣得以保留。
- Hough 轉換把找直線變成在 (ρ, θ) 空間投票，對缺口與雜點很穩健。
- Otsu 閾值最大化類間變異數 $\sigma_B^2 = P_1 P_2 (m_1 - m_2)^2$，可由直方圖的累積和算出；照明不均時改用局部閾值。
- 區域方法（成長、分裂合併、k-means、SLIC、正規化切割）依相似性分組；正規化切割避免了單純最小切割偏好小區塊的問題。
- 分水嶺從極小值開始淹梯度影像，在水相遇處築壩；要避免過度分割，**標記**不可或缺。
- 深度網路學會了相似度與邊緣的衡量方式，但閾值化、多尺度融合、種子或提示、實例分離，仍是現代流程的骨架。

練習

1. 一個一維訊號先有五個樣本為 10，接著在四個樣本內線性上升到 50，之後維持 50。寫出它的一階與二階差分，並標出二階差分的零交越相對於斜坡的位置。

   提示

   一階差分在四個斜坡步上都是 10，其他地方為 0。二階差分在斜坡起點為 +10、終點為 −10，中間為 0，所以零交越在斜坡中段。

2. 證明最大化類間變異數 σ_B^2 等價於最小化加權類內變異數 $\sigma_W^2 = P_1 \sigma_1^2 + P_2 \sigma_2^2$。

   提示

   把 $\sum_i (i - m_G)^2 p_i$ 在 k 處拆開，每一類內加減該類平均；交叉項為零，剩下 $\sigma_G^2 = \sigma_W^2 + \sigma_B^2$，而 σ_G^2 與 k 無關。

3. 在 Canny 偵測器中，若設 $T_L = T_H$ 會怎樣？若 $T_L = 0$ 呢？先預測結果，再用 `skimage.feature.canny` 在 `data.camera()` 上驗證。

   提示

   $T_L = T_H$ 等於關掉遲滯：輪廓會斷掉。$T_L = 0$ 會保留每個與強邊緣相連、通過細化的像素，所以邊緣會滲進草地紋理。

4. 一個 Hough 累加器對 640×480 的影像，θ 在 $[-90°, 90°)$ 間以 $1°$ 為一格，ρ 以 1 像素為一格。累加器共有幾格？一張有 20,000 個邊緣像素的邊緣圖會造成多少次加一？

   提示

   $|\rho| \le \sqrt{640^2 + 480^2} = 800$，所以約有 1,601 個 ρ 格與 180 個 θ 格，約 288,000 格。每個邊緣像素對每個 θ 投一票：$20{,}000 \times 180 = 360$ 萬次。

5. 以權重均為 w 的 4 連通像素圖，說明為什麼單純的最小切割傾向切下單一像素。再計算在 N 個像素的影像中切下一個內部像素的 Ncut，說明它接近 1，而沿著短邊界的平衡切割則接近 0。

   提示

   切下一個像素只需成本 $4w$，比任何長邊界都小。對 Ncut 而言，$\mathrm{assoc}(A, V) = 4w$，第一項為 1，第二項極小：Ncut ≈ 1，是很差的分數。

6. 依上文在 `data.coins()` 上執行標記控制分水嶺（背景標記 `coins < 30`、物體標記 `coins > 150`），再把物體閾值改成 120、190 與 230。哪些硬幣會合併或消失？為什麼？

   提示

   在 120 時，物體標記落在上方較亮的背景帶上，硬幣與背景合併成大區域（我們數到 11 個）。在 190 時仍有 24 個，因為每枚硬幣只要一個標記像素就夠；到 230 時，較暗的硬幣失去標記而消失（剩 13 個）。標記決定了結果。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 10. publisher page
2. D. Marr and E. Hildreth, “Theory of edge detection,” *Proceedings of the Royal Society of London. Series B*, vol. 207, no. 1167, pp. 187–217, 1980. doi
3. J. Canny, “A Computational Approach to Edge Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986. doi
4. R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Communications of the ACM*, vol. 15, pp. 11–15, 1972. doi

5. 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
6. M. Sezgin and B. Sankur, “Survey over image thresholding techniques and quantitative performance evaluation,” *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146–165, 2004. doi
7. J. Sauvola and M. Pietikäinen, “Adaptive document image binarization,” *Pattern Recognition*, vol. 33, no. 2, pp. 225–236, 2000. doi
8. R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk, “SLIC Superpixels Compared to State-of-the-Art Superpixel Methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274–2282, 2012. doi
9. J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000. doi
10. L. Vincent and P. Soille, “Watersheds in digital spaces: an efficient algorithm based on immersion simulations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, 1991. doi
11. 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, pp. 3523–3542, 2022. doi ·arXiv
12. 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
13. J. Long, E. Shelhamer and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” arXiv:1411.4038, 2014. arXiv
14. O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, Lecture Notes in Computer Science, vol. 9351, pp. 234–241, Springer, 2015. doi ·arXiv
15. K. He, G. Gkioxari, P. Dollár and R. Girshick, “Mask R-CNN,” arXiv:1703.06870, 2017. arXiv
16. 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
17. A. Kirillov, E. Mintun, N. Ravi et al., “Segment Anything,” arXiv:2304.02643, 2023. arXiv
18. R. Sun, T. Lei, Q. Chen, Z. Wang, X. Du, W. Zhao and A. K. Nandi, “Survey of Image Edge Detection,” *Frontiers in Signal Processing*, 2022. doi
19. S. Xie and Z. Tu, “Holistically-Nested Edge Detection,” arXiv:1504.06375, 2015. arXiv
20. D. Stutz, A. Hermans and B. Leibe, “Superpixels: An Evaluation of the State-of-the-Art,” *Computer Vision and Image Understanding*, vol. 166, pp. 1–27, 2018. doi
21. I. Arganda-Carreras, S. C. Turaga, D. R. Berger et al., “Crowdsourcing the creation of image segmentation algorithms for connectomics,” *Frontiers in Neuroanatomy*, vol. 9, 2015. doi
22. M. Maška, V. Ulman, D. Svoboda et al., “A benchmark for comparison of cell tracking algorithms,” *Bioinformatics*, vol. 30, no. 11, pp. 1609–1617, 2014. doi ·Cell Tracking Challenge

IP 11・影像分割（二）：主動輪廓——蛇模型與水平集

用「會動的曲線」分割物體：參數式蛇模型、梯度向量流、氣球力，以及測地主動輪廓與 Chan–Vese 等水平集方法。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/

你將學到

- 為什麼可以把分割問題寫成「移動一條曲線，直到能量不再下降」，以及這樣做比門檻化和邊緣連接多了什麼好處。
- **蛇模型**（snake，一種顯式、參數式的主動輪廓）如何由內部能量與外部能量定義，以及 Euler–Lagrange 方程式如何把能量變成迭代演算法。
- 外力該怎麼選：影像梯度、邊緣圖、**梯度向量流**（gradient vector flow, GVF）與**氣球力**（balloon force），以及它們各自要解決的問題。
- **水平集方法**（level-set method）如何把曲線隱式表示為函數 ϕ 的零等高線，並在 CFL 限制下以迎風有限差分求解 $\phi_t + F|\nabla\phi| = 0$。
- 速度函數如何編碼以邊緣為主的模型（**測地主動輪廓**）與以區域為主的模型（源自 Mumford–Shah 的 **Chan–Vese**），以及符號距離重新初始化、窄帶法與形態學近似。
- 主動輪廓在今天的位置：成為深度網路裡的輪廓精修模組，以及損失函數。

先看全貌

IP 10 用的是局部規則：這個像素過門檻、那段邊緣連起來、這塊區域往外長。這些規則完全不在乎結果的**形狀**。於是我們可能得到有缺口的邊界、從弱邊緣漏出去的區域，或是一堆零散的孤立像素。

主動輪廓（active contour）換了一個角度。先在物體附近放一條封閉曲線，然後讓它**動起來**。影像推著曲線走（朝向邊緣，或朝向「裡面」與「外面」分得最好的位置），曲線本身的彈性又把它拉住（不喜歡被拉長、也不喜歡被折彎）。當兩邊的力互相抵消，曲線停下來，它最後的位置就是分割結果。輸出永遠是一條封閉、平滑的邊界，而不是一堆邊緣碎片。

> **用最簡單的話說**
>
> 在桌上的玩具外面套一條大橡皮筋，然後讓橡皮筋慢慢縮小。它會往內滑，碰到玩具就停住，緊緊貼著玩具的輪廓。橡皮筋自己想要變短、變圓滑；玩具擋住它，不讓它再縮。蛇模型就是那條橡皮筋，而「玩具」就是圖片裡邊緣很明顯的地方。水平集則是更聰明的橡皮筋：如果裡面有兩個玩具，它還會自己「啪」一聲分成兩條。

本章依照課本 [1] 的編排，依序介紹這兩大家族。**蛇模型**（Kass、Witkin 與 Terzopoulos [2]）把曲線存成一串點，直接移動這些點。**水平集**（Osher 與 Sethian [5]）把曲線隱式地存成某個曲面的

零等高線，改為移動曲面，因此曲線的分裂與合併會自動發生。兩者都是**變分法**（variational）：先寫下一個能量，再對它做梯度下降。這個想法本身，比任何一條特定公式都更長壽，一路延續到了深度學習。

背景：為什麼要讓曲線動起來

白話地說：與其逐一判斷每個像素，不如一次描述整條邊界，然後問「這條邊界有多好？」，再一點一點改進它。

精確地說，主動輪廓方法有三個要素：

1. 封閉曲線 C 在影像定義域 $\Omega \subset \mathbb{R}^2$ 中的**表示法**：顯式（參數化曲線 $v(s)$）或隱式（函數 $\phi(x, y)$ 的零水平集）。
2. 一個**能量** $E(C)$，當 C 是好邊界時能量低。它總是混合了**規則性**項（偏好短而平滑的曲線）與**資料**項（曲線上的邊緣很強，或曲線內外兩區各自很均勻）。
3. 一條**演化規則**，改變 C 讓 E 下降，通常是以人工時間 t 寫成的偏微分方程式（PDE）形式的梯度下降。

和 IP 10 相比，規則性項能補上邊緣的缺口，輸出永遠是可以直接交給 IP 12 形狀描述子的封閉曲線，而先驗知識（使用者點一下的位置、對平滑的偏好、物體內部的灰階模型）也能自然地放進來。代價是能量不是凸函數：梯度下降只會找到**局部**最小值，所以結果取決於起點。

以蛇模型做影像分割

蛇模型是一串在影像上滑動的封閉點鏈。每個點同時感受到兩種拉力：相鄰的點讓鏈子保持短而平滑，影像則把它拖向邊緣。

蛇模型是一條參數曲線

我們把輪廓寫成

$$v(s) = (x(s), y(s)), \qquad s \in [0, 1], \quad v(0) = v(1),$$

其中 s 是沿曲線繞一圈的參數，$x(s)$、$y(s)$ 是影像座標。對 s 的導數記為 $v_s = dv/ds$、$v_{ss} = d^2v/ds^2$。在電腦裡，曲線被取樣成 n 個點 $v_i = (x_i, y_i)$，$i = 0, \dots, n-1$；對封閉蛇模型而言，索引以 n 取模。

能量泛函

白話說：能量替每條曲線打分數。它把曲線被拉長、被彎折的程度加起來，再加上一個在我們希望曲線停留之處較低的影像項；蛇模型要找的是分數最低的曲線。

[2] 的蛇模型能量是

$$E_{\text{snake}}(v) = \int_0^1 \Big[\underbrace{\tfrac{1}{2}\big(\alpha\,|v_s(s)|^2 + \beta\,|v_{ss}(s)|^2\big)}_{E_{\text{int}}} + E_{\text{ext}}(v(s)) \Big]\, ds.$$

- $\alpha \geq 0$ 是**彈性**（張力）權重。$|v_s|^2$ 在曲線被拉長時變大，所以 α 讓蛇模型像一條想縮短的鬆緊帶。
- $\beta \geq 0$ 是**剛性**（彎曲）權重。$|v_{ss}|^2$ 衡量曲線轉彎有多急，所以 β 讓蛇模型像一根不肯被折彎的細金屬棒。
- $E_{\text{ext}}(x, y)$ 是**外部（影像）能量**，一個定義在整張影像上的純量場，在我們希望蛇停下的地方數值**較低**。

對灰階影像 f，常見的外部能量是

$$E_{\text{ext}}(x, y) = -\,w_{\text{edge}}\,|\nabla (G_\sigma * f)(x, y)|^2 \;-\; w_{\text{line}}\,(G_\sigma * f)(x, y),$$

其中 G_σ 是標準差為 σ 的高斯函數，$*$ 是卷積，∇ 是梯度，w_{edge}、w_{line} 是權重。$w_{\text{edge}} > 0$ 時蛇被強邊緣吸引；$w_{\text{line}} > 0$ 時被暗線吸引，$w_{\text{line}} < 0$ 時被亮線吸引。高斯模糊把 E_{ext} 的谷底變寬，讓蛇在幾個像素外就能「感覺」到邊緣；[2] 稱之為尺度空間延續（scale-space continuation）。

從能量到運動：Euler–Lagrange 方程式

要最小化這個能量，我們用變分法：最小化的曲線必須滿足被積函數的 **Euler–Lagrange 方程式**。若 α、β 為常數，它是

$$\alpha\, v_{ss} - \beta\, v_{ssss} - \nabla E_{\text{ext}}(v) = 0.$$

把它讀成力平衡：αv_{ss} 是張力（指向局部曲率中心，讓曲線縮短），$-\beta v_{ssss}$ 是彎曲力（把折角拉直），$F_{\text{ext}} = -\nabla E_{\text{ext}}$ 是影像力。平衡時三者互相抵消。

這條方程式通常無法直接解，所以讓 v 隨人工時間 t 變化，沿著下坡滑動：

$$\gamma\, \frac{\partial v}{\partial t} = \alpha\, v_{ss} - \beta\, v_{ssss} + F_{\text{ext}}(v),$$

其中 $\gamma > 0$ 是黏滯（阻尼）常數。當 $\partial v / \partial t = 0$，就回到 Euler–Lagrange 方程式。

僅線上版：推導蛇模型方程式 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/推導蛇模型方程式/*

離散化與迭代解法

白話說：把曲線取樣成一串點之後，平滑力就變成一個固定的矩陣，所以蛇模型每走一步，每個座標只需要一次矩陣與向量的乘法。

以單位間距取樣 n 個點，中央差分給出

$$v_{ss} \approx v_{i+1} - 2v_i + v_{i-1}, \qquad v_{ssss} \approx v_{i+2} - 4v_{i+1} + 6v_i - 4v_{i-1} + v_{i-2}.$$

把所有 x 座標排成向量 $\mathbf{x} \in \mathbb{R}^n$（$\mathbf{y}$ 同理），則 $\alpha v_{ss} - \beta v_{ssss} = -A\mathbf{x}$，其中 A 是對稱、循環、五對角的 $n \times n$ 矩陣，每一列以對角線為中心為

$$[\ \beta,\ \ -\alpha-4\beta,\ \ 2\alpha+6\beta,\ \ -\alpha-4\beta,\ \ \beta\ ]$$

（在頭尾處循環繞回）。內力對點的位置是線性的，外力則不是。[2] 的經典技巧是**半隱式**步驟：內力部分以新時刻 t 隱式處理，外力部分以舊時刻 $t-1$ 顯式處理：

$$\gamma\,(\mathbf{x}^{t}-\mathbf{x}^{t-1}) = -A\,\mathbf{x}^{t} + \mathbf{f}_x(\mathbf{x}^{t-1},\mathbf{y}^{t-1}) \quad\Longrightarrow\quad \mathbf{x}^{t} = (A+\gamma I)^{-1}(\gamma\,\mathbf{x}^{t-1}+\mathbf{f}_x),$$

$\mathbf{y}$ 則搭配 $\mathbf{f}_y$ 做同樣的事。$\mathbf{f}_x$、$\mathbf{f}_y$ 是 F_{ext} 在目前點位上（以內插）取得的分量，I 是單位矩陣。$(A+\gamma I)$ 固定不變，只需反矩陣或分解一次；之後每次迭代只是一次便宜的矩陣–向量乘法。把剛硬的四階項隱式處理，正是蛇模型能走合理步長而不爆掉的原因。步長是 $1/\gamma$：γ 越大，步伐越小越安全。

```
import numpy as np

def snake_step_matrix(n, alpha, beta, gamma):
    """(A + gamma*I)^-1 for a closed snake with n points (h = 1)."""
    row = np.zeros(n)
    row[[0, 1, 2, -2, -1]] = [2*alpha + 6*beta, -alpha - 4*beta, beta,
                              beta, -alpha - 4*beta]
    A = np.stack([np.roll(row, i) for i in range(n)])
    return np.linalg.inv(A + gamma * np.eye(n))

M = snake_step_matrix(100, alpha=0.05, beta=0.05, gamma=1.0)
# one iteration, given external forces fx, fy sampled at the points:
# x = M @ (gamma * x + fx);  y = M @ (gamma * y + fy)
```

scikit-image 的 `active_contour` 建立相同的內部能量矩陣，也提供上面的邊緣與線條能量，但它並不是上述更新式的完全複本：它自行計算影像梯度，並限制每次迭代每個點最多能移動多遠 [12]：

```
import numpy as np
from skimage import data
from skimage.color import rgb2gray
from skimage.filters import gaussian
from skimage.segmentation import active_contour

img = rgb2gray(data.astronaut())
s = np.linspace(0, 2 * np.pi, 400)
init = np.column_stack([100 + 100 * np.sin(s),   # rows
                        220 + 100 * np.cos(s)])  # columns
snake = active_contour(gaussian(img, sigma=3), init,
                       alpha=0.015, beta=10, gamma=0.001)
print(snake.shape)  # (400, 2): the final (row, col) points
```

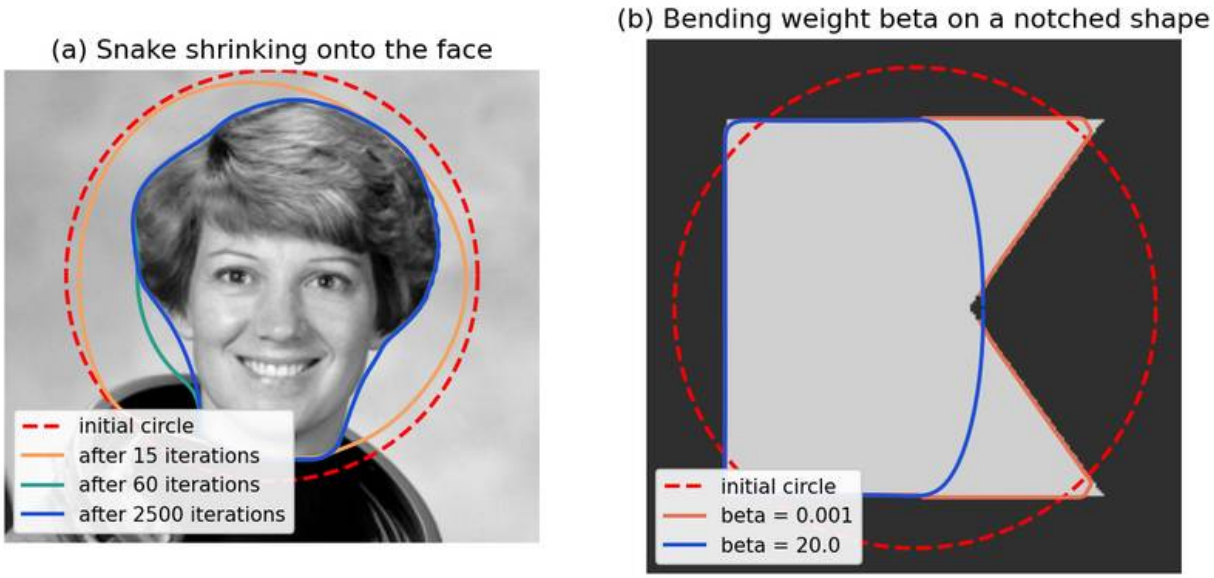


圖 *11.1* — (*a*) 以圓形起始的蛇在張力作用下收縮，直到邊緣力讓它停在頭髮與臉的輪廓上。(*b*) 彎曲權重 β：$\beta = 0.001$ 時蛇會沿著 V 形缺口與尖角走；$\beta = 20$ 時它不肯彎得那麼急，改以一段平滑弧線跨過缺口。

圖 11.1(a) 是蛇模型典型的一生：在平坦背景中只靠張力快速收縮，進入模糊邊緣圖有坡度的範圍後放慢，最後鎖定附近最強的邊緣。(b) 顯示 β 控制的取捨：剛硬的蛇不怕雜訊和小缺口，卻跟不上尖角與凹陷。

外力的選擇

外力決定蛇要找**什麼**。常見的選擇有：

- **邊緣圖的梯度**。先算一張在邊緣上數值大的邊緣圖 $e(x,y) \geq 0$，例如 $e = |\nabla(G_\sigma * f)|$（或其平方），再用 $F_{\text{ext}} = \kappa \nabla e$，$\kappa$ 為權重。這個力沿 e 往上坡，指向邊緣的脊線。
- **線條吸引**。用 $F_{\text{ext}} = \pm\nabla(G_\sigma * f)$ 追蹤暗或亮的曲線狀結構（道路、血管、文字筆畫）。

它們有同一個弱點。邊緣圖的梯度在遠離邊緣處幾乎為零（圖 11.2a）。起點離邊界太遠的蛇，除了自己的張力之外什麼都感覺不到。而在狹窄的凹陷裡，兩側牆面的力都指向側面而互相抵消，沒有任何力把蛇**往下**拉進去。這兩個問題，**捕捉範圍有限**與**難以收斂進凹陷**，正是 GVF 的動機。

> **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

梯度向量流（GVF）

GVF 保留 ∇e 好的部分（在有邊緣的地方準確指向邊緣），再以平滑擴散把其餘區域填滿。Xu 與 Prince [3] 把 GVF 場 $\mathbf{w} = (u, v)$ 定義為：在 $|\nabla e|$ 大的地方貼近 ∇e、其餘地方盡量平滑的向

量場；權重 $\mu > 0$ 平衡這兩個要求，雜訊越多應該越大。這個場以迭代擴散方程式計算：遠離邊緣時，場就只是從邊緣往外擴散，同時把邊緣的方向帶出去。顯式更新同時包含擴散，以及把向量場拉回邊緣梯度的反應項；時間步長必須一起考慮兩者。在單位網格間距下，程式使用 $\Delta t = 0.9/(4\mu + \max|\nabla e|^2)$。其中 0.9 是保守的安全餘裕，不是 GVF 能量本身的參數。

僅線上版：*GVF 能量與擴散方程式 — https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/gvf-能量與擴散方程式/*

```
import numpy as np
from scipy import ndimage as ndi

def gvf(f, mu=0.2, iters=500):
    """GVF on a unit-spaced grid, with a conservative explicit time step."""
    f = np.asarray(f, dtype=float)
    if f.ndim != 2 or min(f.shape) < 2 or not np.isfinite(f).all():
        raise ValueError("f must be a finite 2-D array with both dimensions >= 2")
    if not np.isfinite(mu) or mu <= 0:
        raise ValueError("mu must be finite and positive")
    fy, fx = np.gradient(f)
    mag2 = fx**2 + fy**2
    dt = 0.9 / (4 * mu + mag2.max())  # diffusion AND reaction enter the bound
    u, v = fx.copy(), fy.copy()
    for _ in range(iters):
        u += dt * (mu * ndi.laplace(u, mode="nearest") - (u - fx) * mag2)
        v += dt * (mu * ndi.laplace(v, mode="nearest") - (v - fy) * mag2)
    return u, v

edge_map = np.array([[0., 1.], [1., 0.]])
u, v = gvf(edge_map, mu=0.25, iters=100)
print(np.isfinite(u).all() and np.isfinite(v).all())  # True
print(np.round(u, 3))  # [[ 0.8  0.8], [-0.8 -0.8]]
```

接著蛇模型把 $\kappa\nabla e$ 換成 $F_{\text{ext}} = \kappa\,\mathbf{w}$，演算法其餘部分完全不變。

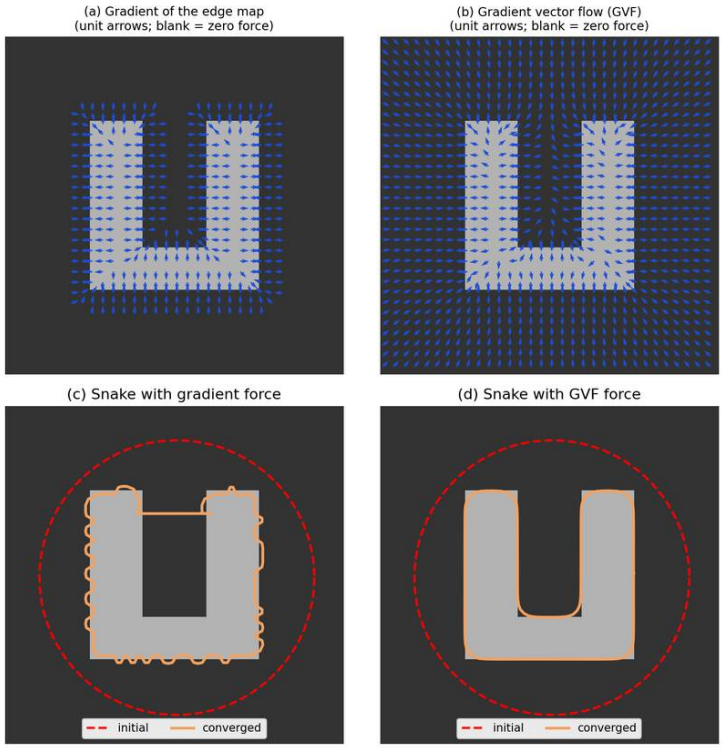


圖 *11.2* — (*a*) 邊緣圖的梯度只在邊緣周圍的細帶內非零。(*b*) *GVF* 把這些向量擴散到整張影像；缺口內的向量向下指。(*c*) 使用梯度力時，從虛線圓出發的蛇卡在缺口開口處（鋸齒狀處是它被邊緣帶勾住、又沒有力量把它撫平的地方）。(*d*) 同樣的蛇、同樣的參數，改用 *GVF* 驅動，便一路沿著凹陷走到底。

圖 11.2 在合成的 U 形上重現了 [3] 的經典測試。改變有兩點：蛇現在從影像任何地方都能感覺到物體（捕捉範圍大），而缺口內擴散後的向量是往下而非往側面指，所以蛇被拉進凹陷。有一個實作細節很重要：蛇進入缺口時長度變長，所以每隔幾十次迭代就要把點**重新取樣**成大致等間距，否則點鏈會太稀疏，貼不住牆面。

氣球力

Cohen [4] 從另一個角度處理捕捉範圍問題。如果把一條小蛇初始化在物體**內部**，張力會讓它在任何邊緣力碰得到它之前就縮成一點。那就加上壓力：

$$F_{\text{ext}}(v) = \kappa \nabla e(v) \;+\; k_1 \,\mathbf{n}(s),$$

其中 $\mathbf{n}(s)$ 是曲線在 $v(s)$ 的單位外法向量，k_1 是壓力。$k_1 > 0$ 時蛇像氣球一樣膨脹；$k_1 < 0$ 時則洩氣收縮。在真正的邊界上，邊緣力必須能壓過壓力（那裡 $\kappa\,|\nabla e| > |k_1|$），但在物體內部微弱、雜亂的梯度處則不行。Cohen 也建議把邊緣力正規化成 $\nabla e / |\nabla e|$，讓一個常數就能控制各處的強度。

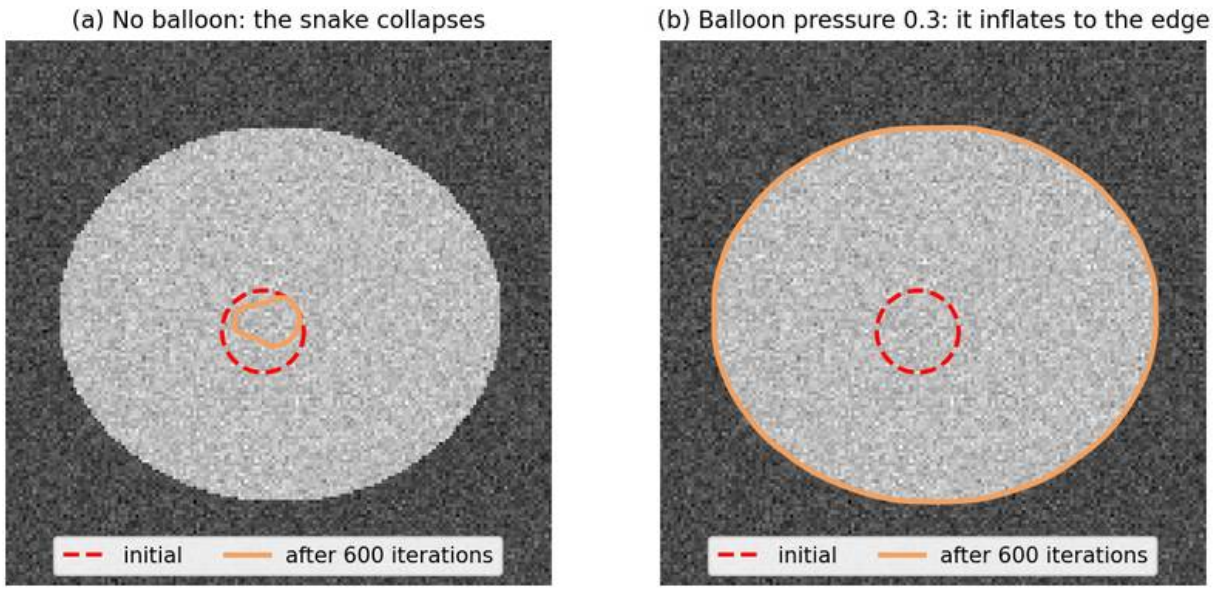


圖 *11.3* — 氣球力。(*a*) 帶雜訊物體內部的一條小蛇，只靠張力和邊緣力時會塌縮，因為附近沒有任何邊緣構得到。(*b*) 加上每次迭代 *0.3* 像素的向外壓力，蛇會一直膨脹，直到真實邊界上的邊緣力與壓力平衡。

氣球力有代價：你必須事先知道該膨脹還是收縮，而壓力太大時蛇會直接衝過弱邊緣。GVF 與氣球力彼此互補，實務上常一起使用。

蛇模型的限制

- **初始化**。能量有許多局部最小值。即使用了 GVF，起點橫跨兩個物體、或落在雜亂強邊緣錯誤一側的蛇，仍會收斂到錯誤答案。
- **參數調整**。α、β、γ、κ、模糊程度 σ、壓力 k_1 與 GVF 的 μ 互相影響。在一張影像上好用的數值，換一張常常就失效。
- **點的管理**。點會沿著曲線漂移，有的地方擠在一起，有的地方散開。實用的蛇模型需要重新取樣，還得偵測曲線是否自我相交。
- **拓撲固定**。蛇是一條封閉點鏈。若不加額外且容易出錯的處理，它圍住兩個物體時無法分裂成兩條（圖 11.6a），也無法和另一條蛇合併。

最後這個限制，正是水平集要拿掉的。

以水平集做分割

不直接追蹤邊界，而是追蹤一片地形，讓它的海岸線就是邊界。把地形抬高或壓低，海岸線就跟著移動。如果水位上升到把兩座山丘分開，一條海岸線就變成兩條，不需要任何特殊程式碼。

隱式表示

白話說：在每個像素存一個高度，曲線就是高度為零的那條「海岸線」。

水平集函數（level-set function）是純量函數 $\phi : \Omega \times [0,\infty) \to \mathbb{R}$，使得時刻 t 的曲線就是它的零水平集：

$$C(t) = \{\, (x,y) \in \Omega \;:\; \phi(x,y,t) = 0 \,\}.$$

本章採用的慣例是：曲線內部 $\phi < 0$，外部 $\phi > 0$。（有些論文，包括 [8]，採相反符號，方程式的正負號也隨之改變。）從 ϕ 可以直接得到兩個幾何量：

$$\mathbf{N} = \frac{\nabla \phi}{|\nabla \phi|}, \qquad \kappa = \nabla \cdot \frac{\nabla \phi}{|\nabla \phi|},$$

其中 $\mathbf{N}$ 是單位**外**法向量，κ 是通過該點之等高線的**曲率**（凸的部分為正；半徑 r 的圓有 $\kappa = 1/r$）。

最方便的 ϕ 是**符號距離函數**（signed distance function）：$|\phi(x,y)|$ 是 (x,y) 到 C 的歐氏距離，正負號則區分內外。它幾乎處處滿足 $|\nabla \phi| = 1$，讓數值計算保持良好。

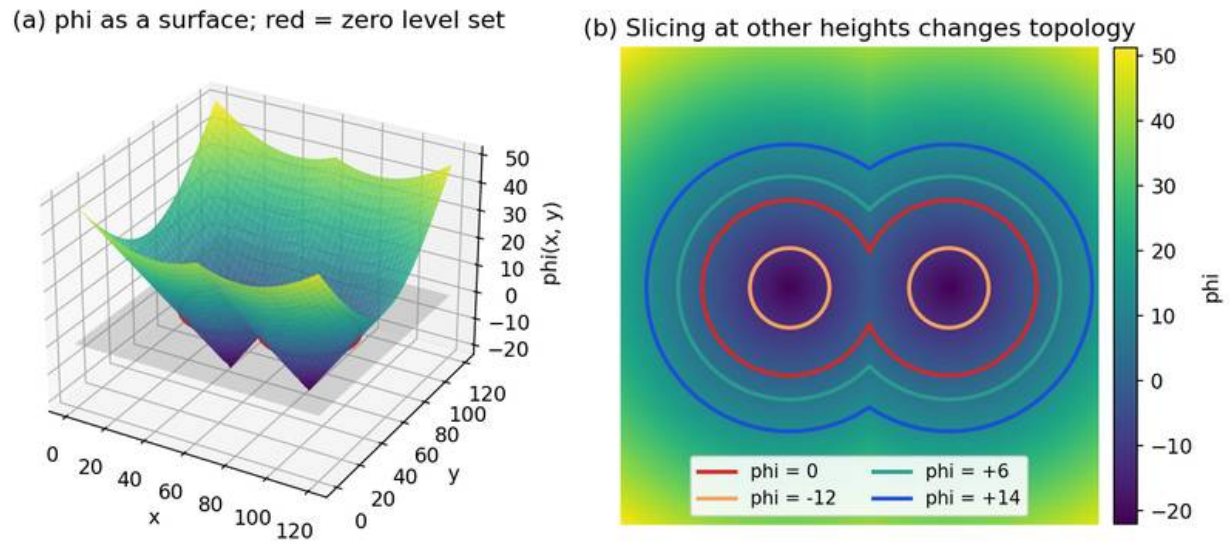


圖 *11.4* — 兩個相切圓盤的水平集函數 φ，以符號距離建構。(*a*) 曲線就是曲面與 $\varphi = 0$ 平面相交之處。(*b*) 在其他高度切同一個曲面，會得到兩條分開的曲線（$\varphi = -12$），或一條合併的曲線（$\varphi = +6$、$+14$）。把 φ 往上或往下移動，就是水平集輪廓不需特殊處理就能分裂與合併的方式。

水平集方程式

假設曲線上每一點沿外法向以速度 F 移動（正值代表向外）：

$$\frac{\partial C}{\partial t} = F\,\mathbf{N}.$$

曲線上的點始終保持零值，所以對所有 t 都有 $\phi(C(t),t) = 0$。以連鎖律微分：

$$\phi_t + \nabla\phi \cdot \frac{\partial C}{\partial t} = 0 \quad\Longrightarrow\quad \phi_t + F\,\nabla\phi \cdot \frac{\nabla\phi}{|\nabla\phi|} = 0 \quad\Longrightarrow\quad \boxed{\phi_t + F\,|\nabla\phi| = 0}$$

這就是 Osher 與 Sethian [5] 的水平集方程式。ϕ_t 是 ϕ 對時間的導數，F 是**速度函數**，可以取決於位置（影像資料）、曲線的幾何（曲率 κ）或全域統計量。方程式是在固定的像素網格上對所有 (x, y) 求解，而不只在曲線上；任何時刻都能取出零等高線得到曲線。方程式完全不在乎零集合有幾塊，所以分裂與合併自然發生。

離散計算：迎風差分與 CFL 條件

白話說：在網格上，波前必須用**來的那一側**的差分來更新，也就是 [5] 格式中的**迎風**（upwind）方向。令 ϕ_{ij}^n 為像素 (i, j) 在第 n 步（步長 Δt，網格間距 $\Delta x = 1$）的值；每一步都減去 $\Delta t\, F_{ij}$ 乘上一個由單側差分組成的 $|\nabla\phi|$ 迎風估計值，下面的 `evolve` 函式就是這樣實作的。若這裡改用普通的中央差分，會產生振盪，尖角也會被抹平甚至發散；迎風格式才能讓波前保持銳利且穩定。

僅線上版：迎風格式的公式 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/迎風格式的公式/*

顯式格式只有在波前每步移動不超過一格時才穩定，這就是 **Courant–Friedrichs–Lewy（CFL）條件**：

$$\Delta t \max_{ij} |F_{ij}| \le c\,\Delta x, \qquad 0 < c \le 1.$$

實務上每一步都依目前的最大速度計算 Δt，下面的程式碼就是這樣做的。

```
import numpy as np
from scipy import ndimage as ndi

def signed_distance(mask):
    """Negative inside, positive outside, ~0 on the boundary."""
    return ndi.distance_transform_edt(~mask) - ndi.distance_transform_edt(mask)

def evolve(phi, F, steps=100, cfl=0.5):
    """Upwind scheme for phi_t + F |grad phi| = 0 (F is an array or scalar)."""
    for _ in range(steps):
        p = np.pad(phi, 1, mode="edge")
        dxm = phi - p[1:-1, :-2]; dxp = p[1:-1, 2:] - phi   # backward / forward
        dym = phi - p[:-2, 1:-1]; dyp = p[2:, 1:-1] - phi
        grad_plus = np.sqrt(np.maximum(dxm, 0)**2 + np.minimum(dxp, 0)**2 +
                            np.maximum(dym, 0)**2 + np.minimum(dyp, 0)**2)
        grad_minus = np.sqrt(np.minimum(dxm, 0)**2 + np.maximum(dxp, 0)**2 +
                             np.minimum(dym, 0)**2 + np.maximum(dyp, 0)**2)
        dt = cfl / (np.abs(F).max() + 1e-12)             # CFL: |F| dt <= dx
        phi = phi - dt * (np.maximum(F, 0) * grad_plus + np.minimum(F, 0) * grad_minus)
    return phi

yy, xx = np.mgrid[:128, :128]
phi = signed_distance((xx - 64)**2 + (yy - 64)**2 < 20**2)
phi = evolve(phi, F=1.0, steps=20)                # grow outward by 20 * 0.5 = 10 px
phi = signed_distance(phi < 0)                     # re-initialize
```

```
print(np.sqrt((phi < 0).sum() / np.pi))            # radius, about 30
```

指定速度函數

所有的影像分析都寫在 F 裡。主流有兩大家族。

以邊緣為主的速度：測地主動輪廓

白話說：讓波前在平坦區域跑得快、在邊緣上變慢，它就會停在邊緣所在之處。

定義一個**邊緣停止函數**，在平坦區域接近 1、在邊緣上接近 0，例如

$$g(x,y) = \frac{1}{1 + |\nabla (G_\sigma * f)(x,y)|^2}.$$

Caselles、Kimmel 與 Sapiro [6] 指出，$\beta = 0$ 的蛇模型與尋找**加權長度**最小的曲線密切相關：

$$L_g(C) = \oint_C g(C(s))\, ds,$$

其中 ds 是弧長。在 g 定義的度量下，邊緣是「便宜」的地方，所以最小化者是一條緊貼邊緣的測地線（最短路徑）。這就是**測地主動輪廓**（geodesic active contour, GAC）。對 L_g 做梯度下降，再加上一個類似 Cohen 氣球的選擇性常數壓力 c，在本章的符號慣例下得到

$$\phi_t = g\,|\nabla \phi|\,(\kappa - c) + \nabla g \cdot \nabla \phi.$$

每一項各有任務。$g\,\kappa\,|\nabla\phi|$ 是曲率流（縮短並平滑曲線），在邊緣附近減速到停止。$-c\,g\,|\nabla\phi|$ 是壓力：$c > 0$ 膨脹、$c < 0$ 收縮，同樣在邊緣上被關掉。$\nabla g \cdot \nabla \phi$ 是平流項，把曲線拉進邊緣谷底（g 最小處），避免它衝過頭。與蛇模型不同，GAC 是**內蘊的**（intrinsic）：它與曲線如何參數化無關；寫成水平集形式後，也能自由分裂與合併。

由於 g 永遠不會真正等於零，強壓力仍會從微弱或模糊的邊緣漏出去。這正是以區域為主之速度函數的動機。

以區域為主的速度：Mumford–Shah 與 Chan–Vese

白話說：不去找邊緣，而是挑一條曲線，把影像分成內外兩部分，並讓兩部分各自盡可能均勻。

Mumford 與 Shah [7] 把分割寫成：同時找出影像的分段平滑近似與一組邊界。他們的能量在三個願望之間取得平衡：貼近影像、除了跨越邊界之外處處平滑，以及邊界要短。這個能量一般而言很難最小化。

僅線上版：*Mumford–Shah 能量 — https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/mumfordshah-能量/*

Chan 與 Vese [8] 把 u 限制成**只有兩個值的分段常數**：C 內為 c_1，C 外為 c_2。能量變成

$$E_{\mathrm{CV}}(c_1, c_2, C) = \mu\,|C| + \lambda_1 \int_{\mathrm{inside}(C)} (f - c_1)^2\, dx\, dy + \lambda_2 \int_{\mathrm{outside}(C)} (f - c_2)^2\, dx\, dy,$$

（原論文還有一個選擇性的面積項 $\nu\,\mathrm{Area}(\mathrm{inside}(C))$，通常設為零）。$\mu \geq 0$ 是平滑權重，$\lambda_1, \lambda_2 > 0$ 衡量各區域被其常數擬合得多好。注意少了什麼：完全沒有梯度。這個模型可以分割邊界模糊、或只由平均亮度改變所定義的物體，這也是 [8] 取名為「沒有邊緣的主動輪廓」（active contours without edges）的原因。

要改寫成水平集形式，使用 Heaviside 函數 $H(z)$（$z \geq 0$ 時為 1，否則為 0）及其導數 Dirac delta $\delta(z)$。內部 $\phi < 0$ 時，內部指示函數是 $H(-\phi)$，長度是 $|C| = \int \delta(\phi)|\nabla\phi|$。固定 ϕ 時，最佳常數就是兩區的平均：

$$c_1 = \frac{\int f\, H(-\phi)}{\int H(-\phi)}, \qquad c_2 = \frac{\int f\, H(\phi)}{\int H(\phi)}.$$

固定 c_1, c_2 時，對 ϕ 做梯度下降得到

$$\phi_t = \delta_\varepsilon(\phi)\left[\,\mu\,\kappa + \lambda_1 (f - c_1)^2 - \lambda_2 (f - c_2)^2\,\right],$$

其中 δ_ε 是寬度為 ε 的平滑化 delta。逐像素來讀：如果某像素的亮度離 c_1 遠、離 c_2 近，括號為正，ϕ 上升，這個像素就移到外面。曲率項讓邊界保持短。演算法交替更新平均值與 ϕ。由於 δ_ε 在零水平集以外也不為零，這個模型還能在離初始曲線很遠的地方**產生新的輪廓**，有助於在 [8] 的範例中偵測內部孔洞與不相連物件。這種行為取決於所選正規化；能量仍然非凸，不同初始曲線可能收斂至不同局部極小值。

僅線上版：*Chan–Vese*：擬合邊界兩側的區域，找出輪廓 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/papers/chanvese 擬合邊界兩側的區域找出輪廓/*

Chan–Vese 只是區域式模型大家族中的一員：任何區域的機率模型都可以取代那兩個常數 [14]。

僅線上版：其他區域模型 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/其他區域模型/*

```
from skimage import data, img_as_float
from skimage.segmentation import chan_vese, morphological_chan_vese

cam = img_as_float(data.camera())
seg, phi, energies = chan_vese(cam, mu=0.25, lambda1=1, lambda2=1,
                               dt=0.5, max_num_iter=200, extended_output=True)
fast = morphological_chan_vese(cam, num_iter=100, init_level_set="checkerboard",
                               smoothing=3)
```

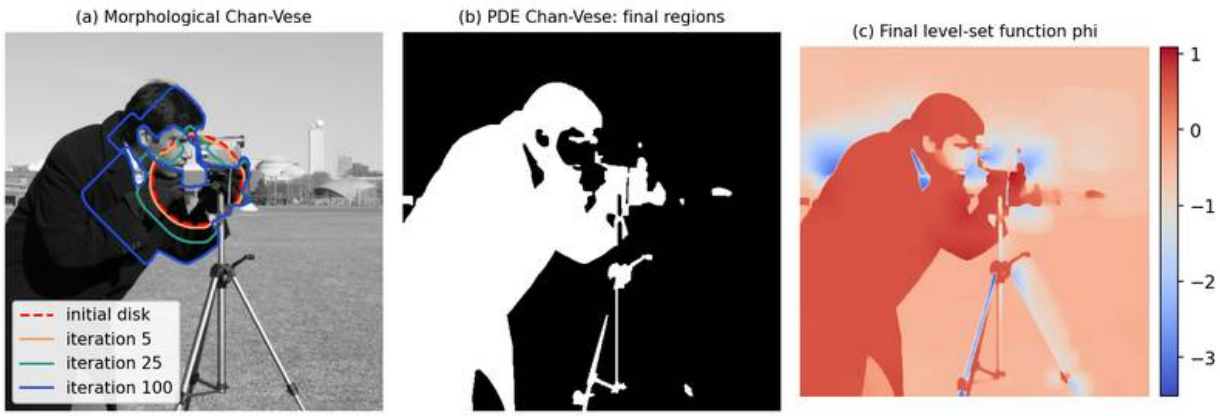


圖 *11.5* — 在攝影師影像上執行 *Chan–Vese*。(*a*) 從臉上一個圓盤出發的形態學近似版本；區域會長進所有「比較接近暗平均、而非亮平均」的地方。(*b*) 從棋盤格出發的 *PDE* 版本：兩相分割把深色人物、相機和三腳架與天空、草地分開，還產生了從未與初始曲線相連的新輪廓（三腳架的腳、遠方建築）。(*c*) 最終的 φ；注意 *scikit-image* 採用相反的符號慣例（內部 $\varphi > 0$）[*12*]。

以符號距離函數初始化與重新初始化

初始的 ϕ 通常是到某條曲線的符號距離：使用者畫的曲線、一個方框、一個圓盤，或許多小圓組成的「棋盤格」，讓演化從每個可能的物體附近開始。PDE 執行時，ϕ 會逐漸偏離距離函數：有些地方變得很陡，有些地方變得很平。陡的地方逼迫時間步長變得極小；平的地方則讓零交越點位置不準、容易受雜訊影響。

經典的補救方法是**重新初始化**（reinitialization）：每隔幾次迭代，在不移動零水平集的前提下，把 ϕ 換成到目前零水平集的符號距離，例如像上面程式那樣用快速歐氏距離轉換重算。Li、Xu、Gui 與 Fox [10] 提出**距離正則化水平集演化**（distance-regularized level-set evolution, DRLSE），在能量中加入一個讓 $|\nabla\phi|$ 保持接近 1 的懲罰項，因此不再需要重新初始化。

僅線上版：重新初始化與 *DRLSE* 的細節 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/重新初始化與-drlse-的細節/*

提升效率

全網格的水平集求解器每一步都更新每個像素，但真正重要的只有零交越處。三種標準的加速方式：

- **窄帶法**（narrow band）。Adalsteinsson 與 Sethian [9] 只在零水平集周圍半寬 w（幾個像素）的帶狀區域內更新 ϕ。波前接近帶的邊緣時，就以新波前為中心重建窄帶（同時重新初始化）。每步的工作量從整張影像面積降到大約曲線長度乘以 w。
- **更大的穩定步長**。半隱式或算子分裂格式可讓剛硬的曲率項使用更大的 Δt，代價是每步要解一個線性系統。
- **形態學近似**。 Márquez-Neila、Baumela 與 Álvarez [11] 以一連串作用在二值水平集上的二值形態學算子（膨脹、侵蝕與平滑算子）取代 PDE。結果速度快、沒有時間步長、不需要重新初

始化，也不會數值不穩定。

scikit-image 依據 [11] 提供 `morphological_geodesic_active_contour` 與 `morphological_chan_vese`，另外也有以 PDE 為基礎的 `chan_vese` [12]。請記住，形態學版本是 PDE 的**近似**：它們在像素網格上移動，「smoothing」參數是算子套用的次數而不是權重 μ，結果與連續模型相近，但並不相同。

僅線上版：形態學近似如何運作 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/形態學近似如何運作/*

```
from skimage import data, img_as_float
from skimage.segmentation import (inverse_gaussian_gradient, disk_level_set,
                                  morphological_geodesic_active_contour)

coins = img_as_float(data.coins())
g = inverse_gaussian_gradient(coins, alpha=100, sigma=2)   # ~0 on edges, ~1 elsewhere
init = disk_level_set(coins.shape, radius=150)
mask = morphological_geodesic_active_contour(g, num_iter=250, init_level_set=init,
                                             smoothing=1, balloon=-1, threshold=0.69)
```

這裡 `inverse_gaussian_gradient` 建立邊緣停止函數 g，`balloon=-1` 讓初始圓盤收縮（相當於 $c < 0$），`threshold` 決定 g 的哪些值算是能擋住氣球的邊緣。

實務比較：蛇模型與水平集

兩個家族最小化的能量很相似，差別主要在表示法，而表示法決定了什麼事情容易做。

簡單說：蛇模型的未知數少（n 個點）、每次迭代便宜、容易互動編輯，點還能帶標籤，但拓撲固定，點會漂移，也可能交叉。水平集每個像素（或窄帶內每個像素）有一個未知數，但能自動分裂與合併、不可能自我相交，也能直接推廣到 3D。兩者典型的失敗也不同：蛇模型會卡在雜亂背景或凹陷處；邊緣式水平集會從弱邊緣漏出去；區域式水平集則會把外觀相似的區域合併。

僅線上版：蛇模型與水平集：逐項比較 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/蛇模型與水平集逐項比較/*

Xu、Yezzi 與 Prince 研究過兩種表述之間的關係 [3]；對許多能量而言，選擇哪一種是實作上的考量，而不是原則上能不能建模。經驗法則：

- 需要一條拓撲已知的邊界、要互動、要快、或需要點的對應關係時（在影片各影格間追蹤輪廓、擬合嘴唇外形、描繪建物輪廓），用**蛇模型**。
- 物體數量未知、物體可能相接或分裂、邊界無法以梯度清楚定義（區域式模型），或在 3D 中工作時，用**水平集**。

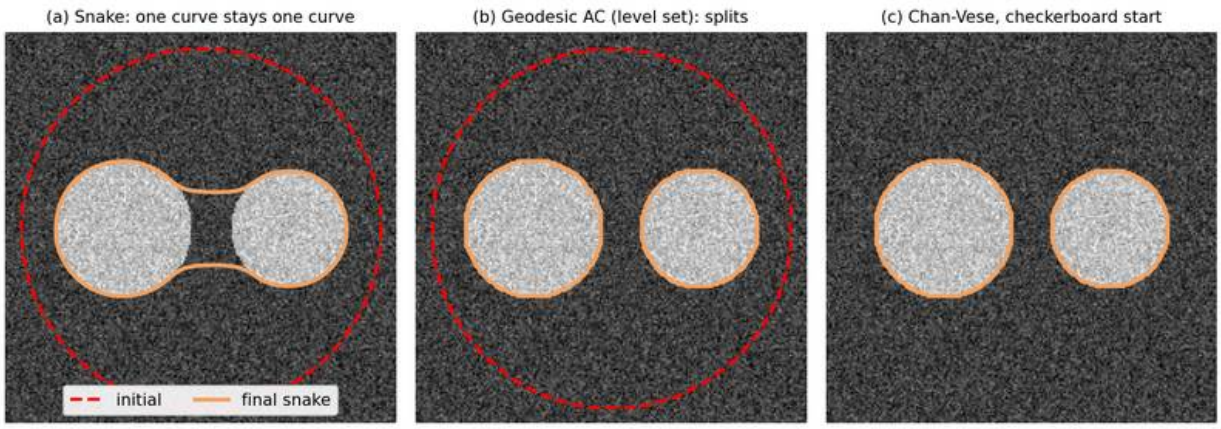


圖 *11.6* — 實務上的拓撲問題。(*a*) 圍住兩個圓盤起始的蛇只能收縮成一條曲線，包住兩者並橫跨中間的空隙。(*b*) 從同一個初始圓出發的測地主動輪廓（形態學水平集版本），兩半在中間相遇時就分裂成兩條曲線。(*c*) 從棋盤格出發的 *Chan–Vese* 根本不需要有意義的初始曲線。

現代觀點

主動輪廓已經不是分割影像的預設方法，深度網路才是。但本章的想法，包括結合邊界長度與區域擬合的能量、反覆變形的顯式輪廓，以及把水平集當作形狀表示，仍是深度分割中活躍的成分。以下六篇論文構成一張不錯的地圖。

醫學影像中的可變形模型。 McInerney 與 Terzopoulos [13] 寫下了經典的早期綜述。它的主要結論至今仍成立：當影像本身含糊不清、需要模型提供平滑性與先驗知識時，可變形模型最有用。

統計式區域水平集。 Cremers、Rousson 與 Deriche [14] 從同一個貝氏框架推導出以區域為主的水平集方法：每個區域都有一個機率模型（針對亮度、色彩、紋理或運動）。他們主張區域式能量比邊緣式蛇模型對雜訊較不敏感、局部最小值也較少。

近期的分類。 Chen、Ge、Wang 與 Chen [15] 把主動輪廓模型分為邊緣式、區域式與混合式，並與深度學習分割比較。傳統模型不需要訓練資料、能給出平滑的封閉邊界，但對初始化和參數敏感；在有標註資料的情況下，深度模型更準確；兩者是互補的。

深度分割中的形狀限制。 Bohlender、Oksuz 與 Mukhopadhyay [16] 綜述了如何把形狀知識（包括主動輪廓與符號距離圖）注入深度醫學影像分割，以修補逐像素分類器產生的破碎區域與拓撲錯誤。

僅線上版：四篇綜述的詳細內容 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/papers/四篇綜述的詳細內容/*

端對端學習的顯式輪廓。 Peng 等人的 Deep Snake [17] 保留了蛇模型的表示法，也就是一個反覆變形的封閉多邊形，但改用學習而來的更新：在頂點位置取樣 CNN 特徵，再沿多邊形做環狀卷積（circular convolution），預測每個頂點朝邊界的位移；作者報告它在實例分割上可達即時速度。環狀卷積扮演的正是經典蛇模型中五對角矩陣 A 的角色：沿著封閉曲線把每個頂點與它的鄰居耦合起來。

把主動輪廓能量當作損失函數。有些方法不在測試時執行輪廓演化，而是把能量放進訓練損

失：帶有長度項與區域項、類似 Chan–Vese 的監督式損失 [18]；作用在網路 softmax 輸出上、不需要標註，因此也支援非監督與半監督訓練的 Mumford–Shah 風格損失 [19]；以及**邊界損失**（boundary loss）[20]，它是一個以真值符號距離圖加權的區域積分，專為微小病灶這類極度不平衡的問題設計。

僅線上版：三種主動輪廓損失的細節 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/active-contours/notes/三種主動輪廓損失的細節/*

深度學習改變了什麼、沒改變什麼：

- **改變了：資料項**。手工設計的力（∇e、GVF、g、區域平均）被學習到的特徵取代，這是準確度提升的主因。
- **沒改變：規則性先驗**。邊界長度、曲率懲罰、符號距離表示與區域均勻性，以損失項和輸出表示的形式重新出現，因為沒有它們，逐像素分類器仍會產生參差不齊、破碎的邊界。
- **沒改變：顯式與隱式的取捨**。以輪廓為基礎的網路繼承了蛇模型固定的拓撲（每個實例一個多邊形）與速度；以遮罩或距離圖為基礎的網路則繼承了水平集的彈性。
- **本身仍然有用**。沒有標註資料時，用 scikit-image 跑 Chan–Vese 或 GAC 只要幾行程式、不需訓練，很適合當基準、產生偽標籤，或作為清理網路遮罩的後處理步驟。

重點整理

- 主動輪廓透過**在曲線上最小化能量**來分割：規則性項（長度、彎曲）加上資料項（邊緣或區域擬合）。對能量做梯度下降，就是一條人工時間中的 PDE。
- **蛇模型**是顯式的點鏈。它的 Euler–Lagrange 方程式 $\alpha v_{ss} - \beta v_{ssss} + F_{\text{ext}} = 0$ 以半隱式步驟 $\mathbf{x}^t = (A + \gamma I)^{-1}(\gamma \mathbf{x}^{t-1} + \mathbf{f}_x)$ 求解。
- 單純的梯度力**捕捉範圍小**，也進不了**凹陷**。GVF 把邊緣向量擴散到整張影像；氣球力讓曲線膨脹或收縮。
- **水平集**把曲線存成 $\{\phi = 0\}$，在 **CFL** 步長限制下以**迎風差分**演化 $\phi_t + F|\nabla\phi| = 0$，拓撲改變自動發生。
- 模型寫在速度 F 裡：**測地主動輪廓**最小化以邊緣加權的長度；**Chan–Vese** 擬合兩個區域平均，不需要邊緣。
- 讓 ϕ 保持接近**符號距離**（重新初始化或 DRLSE），並用**窄帶**或**形態學**近似節省計算；後者是近似 PDE，而不是精確求解。
- 在深度學習中，同樣的想法以**學習式輪廓變形**（Deep Snake）以及由長度、區域與符號距離項構成的**損失函數**延續下來。

練習

1. 只有張力。一條 $\beta = 0$、沒有外力的封閉蛇，初始為半徑 R 的圓，以 n 個等距點取樣。利用離散內力 $\alpha(v_{i+1} - 2v_i + v_{i-1})$，證明圓仍然是圓，並求出步長 τ 的一次顯式更新後半徑如何改變。

提示

對角度間隔 $\theta = 2\pi/n$ 的正多邊形（從圓心量起），$v_{i+1} + v_{i-1} = 2\cos\theta\, v_i$。內力為 $-2\alpha(1-\cos\theta)\, v_i$，指向圓心，所以 $R \mapsto R\,[1 - 2\alpha\tau(1-\cos\theta)]$。圓以等比方式縮小；注意縮小速率與 n 有關，這也是蛇模型參數會與取樣綁在一起的原因之一。

2. 凹陷測試。用上面的 `gvf` 函數建立圖 11.2 的 U 形，改變 GVF 權重 μ（0.05、0.1、0.2）與 GVF 迭代次數。哪個參數決定場能從邊緣延伸多遠？哪個決定它有多平滑？

提示

擴散長度大致隨 $\sqrt{\mu n \Delta t}$ 成長，其中 n 是迭代次數。較大的 μ 也改變平滑與資料擬合的平衡。程式依擴散和反應項調整 Δt，所以 $\mu > 0.25$ 不會自動導致不穩定；改變 μ 時，應比較相同演化時間，或使用固定的收斂容許值。

3. 正負號與方向。在本章慣例（內部 $\phi < 0$）下，當 $F = -1$ 時，圓在 $\phi_t + F|\nabla\phi| = 0$ 下會怎樣？$F = -\kappa$ 時呢？分別寫出半徑 $r(t)$。

提示

$F = -1$：圓以單位速度縮小，$r(t) = r_0 - t$。$F = -\kappa = -1/r$：曲線縮短流，$\dot r = -1/r$，所以 $r(t) = \sqrt{r_0^2 - 2t}$，圓在 $t = r_0^2/2$ 時消失。

4. 實際體會 CFL。用 `evolve` 程式碼片段，在同一個圓上分別以 `cfl=0.5`、`1.0`、`2.0` 執行。量測最後的半徑並觀察零等高線。哪裡壞掉了？為什麼窄帶法無法移除這個限制？

提示

超過 1 時，波前每一步試圖跨越不只一格；迎風資訊取自錯誤的位置，輪廓變得鋸齒狀，甚至數值發散。窄帶法減少的是更新的像素數，而不是每一步的大小。

5. Chan–Vese 失效的時候。建立一張合成影像：左半部是由黑到白的平滑漸層；右半部是灰色背景上的一個灰色圓盤，兩者平均值相同但變異數不同（只在圓盤內加強烈雜訊）。執行 `chan_vese`，從能量的角度解釋結果，並提出一個能成功的區域模型修改。

提示

Chan–Vese 只比較平均值，所以（平均相同的）圓盤是看不見的，而漸層會從中間被切開。若像 [14] 的統計框架那樣，以各自擁有平均值**和**變異數的高斯分布描述每個區域，圓盤就能被分開。

6. 把能量當作損失。改用軟遮罩 $m(x,y) \in [0,1]$（屬於內部的機率）而非水平集來寫 Chan–Vese 能量，並以總變差 $\int|\nabla m|$ 取代長度項。證明固定 m 時最佳的 c_1, c_2 是加權平均，並說明為什麼這個函數可以在沒有標註的情況下訓練網路。

提示

$E = \mu\int|\nabla m| + \lambda_1\int m\,(f-c_1)^2 + \lambda_2\int(1-m)(f-c_2)^2$。令 $\partial E/\partial c_1 = 0$ 得 $c_1 = \int mf / \int m$。E 中沒有任何地方用到真值，只用到影像本身，所以不需標註就能對產生 m 的網路求梯度；可與 [19] 對照。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 11. publisher page
2. M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: Active contour models,” *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988. doi:10.1007/BF00133570
3. C. Xu and J. L. Prince, “Snakes, shapes, and gradient vector flow,” *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 359–369, 1998; see also C. Xu, A. Yezzi Jr., and J. L. Prince, “On the relationship between parametric and geometric active contours,” *Proc. 34th Asilomar Conf. on Signals, Systems, and Computers*, 2000. GVF project page with both papers
4. L. D. Cohen, “On active contour models and balloons,” *CVGIP: Image Understanding*, vol. 53, no. 2, pp. 211–218, 1991. author PDF
5. S. Osher and J. A. Sethian, “Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations,” *Journal of Computational Physics*, vol. 79, pp. 12–49, 1988. doi:10.1016/0021-9991(88)90002-2
6. V. Caselles, R. Kimmel, and G. Sapiro, “Geodesic active contours,” *International Journal of Computer Vision*, vol. 22, no. 1, pp. 61–79, 1997. PDF
7. D. Mumford and J. Shah, “Optimal approximations by piecewise smooth functions and associated variational problems,” *Communications on Pure and Applied Mathematics*, vol. 42, no. 5, pp. 577–685, 1989. doi:10.1002/cpa.3160420503
8. T. F. Chan and L. A. Vese, “Active contours without edges,” *IEEE Transactions on Image Processing*, vol. 10, no. 2, pp. 266–277, 2001. doi:10.1109/83.902291
9. D. Adalsteinsson and J. A. Sethian, “A fast level set method for propagating interfaces,” *Journal of Computational Physics*, vol. 118, pp. 269–277, 1995. summary page
10. C. Li, C. Xu, C. Gui, and M. D. Fox, “Distance regularized level set evolution and its application to image segmentation,” *IEEE Transactions on Image Processing*, vol. 19, no. 12, pp. 3243–3254, 2010. doi:10.1109/TIP.2010.2069690
11. P. Márquez-Neila, L. Baumela, and L. Álvarez, “A morphological approach to curvature-based evolution of curves and surfaces,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014. doi:10.1109/TPAMI.2013.106
12. scikit-image developers, “skimage.segmentation” API reference (`active_contour`, `chan_vese`, `morphological_chan_vese`, `morphological_geodesic_active_contour`). docs
13. T. McInerney and D. Terzopoulos, “Deformable models in medical image analysis: A survey,” *Medical Image Analysis*, vol. 1, no. 2, 1996. author PDF
14. D. Cremers, M. Rousson, and R. Deriche, “A review of statistical approaches to level set segmentation: Integrating color, texture, motion and shape,” *International Journal of Computer Vision*, vol. 72, no. 2, pp. 195–215, 2007. doi:10.1007/s11263-006-8711-1
15. Y. Chen, P. Ge, G. Wang, and H. Chen, “An overview of intelligent image segmentation using active contour models,” *Intelligence & Robotics*, vol. 3, no. 1, pp. 23–55, 2023. open access
16. S. Bohlender, I. Oksuz, and A. Mukhopadhyay, “A survey on shape-constraint deep learning for medical image segmentation,” *IEEE Reviews in Biomedical Engineering*, vol. 16, pp. 225–240, 2023. arXiv:2101.07721
17. S. Peng, W. Jiang, H. Pi, X. Li, H. Bao, and X. Zhou, “Deep snake for real-time instance segmentation,” *CVPR*, 2020. arXiv:2001.01629

18. X. Chen, B. M. Williams, S. R. Vallabhaneni, G. Czanner, R. Williams, and Y. Zheng, “Learning active contour models for medical image segmentation,” *CVPR*, 2019. CVF open access
19. B. Kim and J. C. Ye, “Mumford–Shah loss functional for image segmentation with deep learning,” *IEEE Transactions on Image Processing*, vol. 29, pp. 1856–1866, 2020. doi ·arXiv:1904.02872
20. H. Kervadec, J. Bouchtiba, C. Desrosiers, E. Granger, J. Dolz, and I. Ben Ayed, “Boundary loss for highly unbalanced segmentation,” *Medical Image Analysis*, vol. 67, 2021 (first presented at MIDL 2019). arXiv:1812.07032

IP 12・特徵擷取

把區域、邊界與整張影像轉成一小串數字，讓它們在物體平移、旋轉、縮放或亮度改變時保持不變：從鏈碼、GLCM 紋理到 Harris、MSER 與 SIFT。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/

你將學到

- 「特徵」（feature）與「描述子」（descriptor）的差別，以及描述子最常追求的四種不變性：平移、旋轉、尺度與光照。
- 如何把分割出來的邊界變成可以量測的東西：鏈碼、多邊形、標記圖（signature）與骨架，再進一步算出形狀數與傅立葉描述子。
- 如何用大小、形狀、拓撲（尤拉數）、紋理（直方圖動差、共生矩陣、頻譜）與動差不變量描述整個區域。
- 主成分分析（PCA）如何把物體轉正，並把大量量測壓縮成少數幾個數字。
- Harris–Stephens 角點偵測器、最大穩定極值區域（MSER）與 SIFT 如何找出並描述可以在兩張照片之間配對的興趣點。
- 近二十年的綜述論文如何評價這些手工設計的特徵，以及學習式特徵（SuperPoint、LoFTR、LightGlue、DINOv2）在哪些地方取代了它們。

先看全貌

分割（IP 10、IP 11）告訴我們物體「在哪裡」。但程式沒辦法像人一樣「看」一個區域，它需要一小串數字來說明這個區域「長什麼樣子」，才能比較、排序、搜尋和辨識。產生這串數字的過程，就是特徵擷取。

> **用最簡單的話說**
>
> 想像你在電話裡跟朋友形容一個玩具。你沒辦法傳照片，所以你說：「它是圓的，大概跟柳丁一樣大，中間有一個洞，表面摸起來凹凸不平。」朋友就能從一整箱玩具裡挑出正確的那一個，就算它倒過來放、放得比較遠，或房間比較暗也沒關係。你說的每一句話都是一個**特徵**。好的特徵，就是不管玩具怎麼轉、怎麼移、燈光怎麼變，都依然成立的那些描述。

本章依照 Gonzalez 與 Woods 教科書的順序 [1]：先談邊界，再談區域，接著是處理大量量測的統計工具（PCA），最後是完全不需要分割的特徵：角點、穩定區域與 SIFT 關鍵點。最後這一類把傳統影像處理接上了現代電腦視覺的任務，例如全景接圖、三維重建與視覺定位。

背景：特徵、描述子與不變性

白話版。特徵是你在影像中找到的有趣東西（一個角、一團斑點、一個區域）。描述子是你用來描述它的一串數字。我們希望那些「無關緊要的變化」發生時，描述子不要跟著變。

精確版。依照 [1]，我們把工作拆成兩步：

- **特徵偵測**（feature detection）：找出特徵**在哪裡**——一條邊界、一個區域，或一個帶有尺度與方向的點。
- **特徵描述**（feature description）：替每個偵測到的特徵指定一個向量 $\mathbf{d} \in \mathbb{R}^n$。

若描述子 $\mathbf{d}(\cdot)$ 對一族轉換 $\mathcal{T}$ 滿足

$$\mathbf{d}(T(f)) = \mathbf{d}(f) \quad \text{for all } T \in \mathcal{T},$$

就稱它對 $\mathcal{T}$ **不變**（invariant），其中 f 是影像（或區域），T 是施加在它上面的轉換。若偵測到的量會以可預期的方式**跟著**轉換一起變（例如關鍵點的位置隨物體移動，物體放大兩倍時偵測到的尺度也變成兩倍），就稱它是**共變的**（covariant）。偵測器應該共變，描述子應該不變。

我們最在意的轉換是：

變化	對影像的影響	常見的解法
平移	座標整體位移	減去質心，或改用差值
旋轉	座標旋轉	使用標準方向、取絕對值，或使用與旋轉無關的量
尺度	座標乘上倍率	依大小正規化，或在各尺度上搜尋
光照	強度改變，大致為 $f \mapsto af + b$	用梯度消去 b，再正規化消去 a

不變性與**鑑別力**之間永遠有取捨。對一切都不變的描述子，其實什麼也描述不了：「6」轉 180° 就變成「9」。挑選特徵，其實就是在決定哪些差異對你的任務是重要的。

邊界前處理

白話版。在量測邊界之前，得先把邊界像素按順序排好；通常還要簡化它，免得雜訊和像素造成的鋸齒主導了結果。

邊界追蹤

分割出的區域是一堆像素的集合，但大多數邊界描述子需要的是**有順序**、封閉的點列 $(x_0, y_0), \dots, (x_{K-1}, y_{K-1})$。邊界追蹤演算法從一個固定的起始像素出發，沿著區域邊緣一個像素一個像素地繞行，直到回到起點，就能產生這串點 [1]。OpenCV 的 `cv2.findContours` 和 scikit-image 的 `measure.find_contours` 都能直接給你這串點。

僅線上版：*Moore 邊界追蹤法* — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/moore-邊界追蹤法/*

鏈碼

鏈碼（chain code）把點列換成**方向**序列。從一個邊界像素走到下一個，每一步都是 4 或 8 個方向之一，從正東開始逆時針編號為 0–3 或 0–7 [1]。鏈碼很精簡（8 方向時每步 3 位元），而且本來就與平移無關。

剩下兩個問題：鏈碼取決於**起點**，也取決於**旋轉**。解法都很簡單：

- **起點**。把碼視為環狀序列，選擇能組成最小整數的那個循環位移。
- **旋轉（45° 的倍數）**。改用**一階差分**（first difference）：相鄰兩個元素之間逆時針轉了幾個方向，$d_k = (c_k - c_{k-1}) \bmod 8$，其中 c_k 是第 k 個碼。物體轉 90° 時每個 c_k 都加 2，而每個 d_k 不變。

直接在原始像素格點上算出的鏈碼又長又雜，所以實務上會先把邊界重新取樣到較粗的格點上，讓每一節跨過好幾個像素（圖 12.1 左）。

```
import numpy as np, cv2
from skimage import data

mask = (data.horse() == 0).astype(np.uint8)          # True inside the horse
cnts, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
pts = max(cnts, key=len)[:, 0, :]                    # (x, y), 8-connected boundary

# Freeman 8-direction code: 0 = east, counted counter-clockwise (y points down)
DIRS = {(1, 0): 0, (1, -1): 1, (0, -1): 2, (-1, -1): 3,
        (-1, 0): 4, (-1, 1): 5, (0, 1): 6, (1, 1): 7}
steps = np.diff(np.vstack([pts, pts[:1]]), axis=0)
code = np.array([DIRS[tuple(s)] for s in steps])

diff = (code - np.roll(code, 1)) % 8                 # first difference: rotation invariant
shifts = [tuple(np.roll(diff, -i)) for i in range(len(diff))]
shape_number = min(shifts)                           # start-point invariant
print(len(code), code[:12], diff[:12])
```

僅線上版：斜率鏈碼 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/斜率鏈碼/*

多邊形近似

用少數幾個頂點的多邊形，就能抓住邊界的精髓。

最小周長多邊形（minimum-perimeter polygon, MPP）。想像一條橡皮筋在蓋住邊界的一條方格帶子裡收縮，它最後停留的最短封閉路徑就是 MPP [1]。方格大小控制細節多寡：方格越大，多邊形越粗略。

僅線上版：最小周長多邊形的細節 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/最小周長多邊形的細節/*

合併法（merging）。沿著邊界前進，只要目前這段的最小平方直線擬合誤差低於門檻 T，就繼續把點加進來；誤差超過 T 時就開新的一段。方法簡單，但頂點不一定落在真正的轉角上。

分裂法（splitting）。把距離最遠的兩個邊界點連成一條弦，找出離這條弦最遠的邊界點；若距離超過 T，就把它設為頂點，問題一分為二，再遞迴處理。分裂法較容易把頂點放在真正的反曲點上。`skimage.measure.approximate_polygon` 實作的就是這種遞迴分裂的概念；圖 12.1 第二格顯示兩種容許誤差的結果。

標記圖

標記圖（signature）把二維邊界降成一維函數。最簡單的是質心到邊界的距離，表示成角度的函數 $r(\theta)$，或正規化弧長的函數 $r(s)$ [1]。圓形得到常數；正方形得到四個一模一樣的凸起。標記圖天生與平移無關；旋轉變成函數的**循環位移**，尺度變成乘法，所以只要選一個標準起點（例如最遠點），再除以最大值或標準差即可正規化。$r(\theta)$ 只有在星形區域才是函數；像圖 12.1 的馬，從質心射出的一條射線可能穿過邊界不只一次，所以那裡改用 $r(s)$。

骨架與中軸

白話版。在一片草地的整圈邊緣同時點火，火線從四面八方往內燒，不同方向的火線相遇的地方就構成**骨架**。

精確版。區域的**中軸**（medial axis）是區域中那些在邊界上有**不只一個**最近點的點所組成的集合 [1]。實務上用**細化**（thinning，見 IP 09）求骨架；骨架對邊界上的小突起非常敏感，通常需要先平滑，或事後修剪短分支。

僅線上版：中軸轉換 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/中軸轉換/*

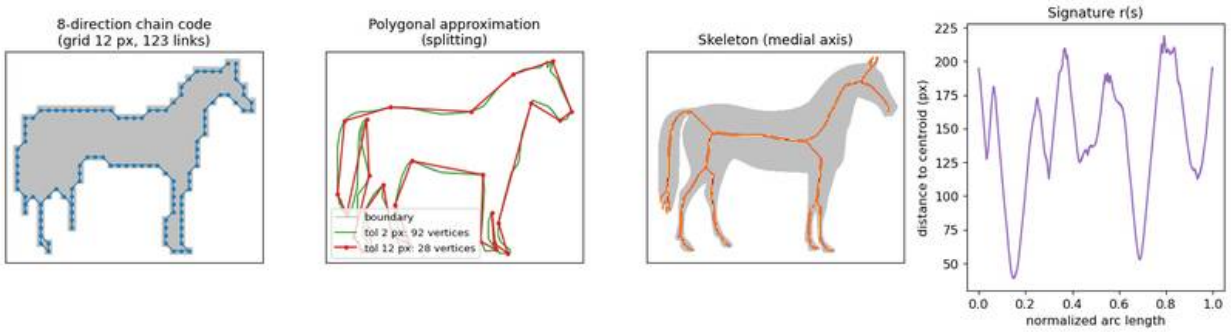


圖 *12.1* — 同一匹馬剪影（*skimage.data.horse*）的邊界表示。由左至右：*12* 像素格點上的 *8* 方向鏈碼；容許誤差 *2* 與 *12* 像素的遞迴分裂多邊形近似；細化得到的骨架；以及標記圖 $r(s)$，即到質心的距離對正規化弧長作圖（幾個深谷就是馬腿）。

邊界特徵描述子

白話版。邊界一旦變成有序點列，就可以量它：有多長、有多寬、有多彎，以及把它寫成一堆波的總和時長什麼樣子。

基本描述子

- **長度**。邊界上的像素數是粗略估計。若用 8 連通鏈碼，水平與垂直步長算 1，斜向步長算 $\sqrt{2}$。
- **直徑**。$\mathrm{Diam}(B) = \max_{i,j} D(p_i, p_j)$，其中 p_i, p_j 是邊界點，D 是距離（通常是歐氏距離）。連接這兩個最遠點的線段稱為**長軸**；**短軸**與之垂直，長度取到能讓以兩軸為邊的方框剛好包住邊界。這個方框稱為**基本矩形**（basic rectangle）。
- **離心率**。長軸長度與短軸長度的比值。（後面的區域離心率改用二階動差定義。）
- **曲率**。沿邊界的斜率變化率。數位邊界上的曲率很雜，所以通常改算某點兩側擬合線段的斜率差。曲率變號處就是反曲點；（沿行進方向）曲率為正表示凸的部分，為負表示凹的部分。

形狀數

邊界的**形狀數**（shape number）是其 4 方向鏈碼的一階差分，再循環位移成最小整數 [1]。它的長度稱為形狀的**階數**（order）n。比較兩個形狀之前，先讓格點對齊各自的方向；接著以**相似度**（degree of similarity）比較，也就是兩者形狀數仍然相同的最大階數。

僅線上版：計算指定階數的形狀數 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/計算指定階數的形狀數/*

傅立葉描述子

白話版。沿著邊界走一圈，把每個點寫成一個複數。這串數每走一圈就重複一次，所以可以拆成許多波。前幾個慢的波給出大致輪廓，快的波負責補細節。

精確版。設邊界有 K 個點 (x_k, y_k)，$k = 0, \dots, K-1$，把每個點寫成

$$s(k) = x(k) + j\,y(k).$$

對 $s(k)$ 做離散傅立葉轉換，就得到**傅立葉描述子**

$$a(u) = \sum_{k=0}^{K-1} s(k)\, e^{-j2\pi uk/K}, \qquad u = 0, 1, \dots, K-1,$$

反轉換則可以重建邊界。如果只保留其中 P 個係數（正負頻率中最低的那些），其餘設為零，就得到平滑、但保留整體形狀的近似 $\hat{s}(k)$（圖 12.2）。這就是為什麼少數幾個傅立葉描述子就能精簡地表示一個形狀。

這些描述子對基本幾何變化的反應很單純 [1]：

邊界的變化	對 $a(u)$ 的影響
平移 $\Delta_{xy} = \Delta_x + j\Delta_y$	只有 $a(0)$ 改變：變成 $a(0) + K\Delta_{xy}$
旋轉 θ	每個係數乘上 $e^{j\theta}$
縮放 α	每個係數乘上 α
起點移動 k_0	$a(u)$ 乘上 $e^{-j2\pi k_0 u/K}$

因此可以組出不變的描述子：丟掉 $a(0)$（平移）、除以 $|a(1)|$（尺度）、只保留大小 $|a(u)|$（旋轉與起點）。只留大小的代價是丟掉了相位，而相位也帶有一部分形狀資訊。這種正規化要求第一諧波不為零；對稱輪廓或改變走訪方向，都可能使它消失。程式會拒絕數值上過小的第一諧波，避免直接相除。比較各自描出的形狀時，也必須統一取樣方式與走訪方向；下面的變換實驗保留同一組取樣點。

```
import numpy as np
from skimage import data, measure

mask = data.horse() == 0
c = max(measure.find_contours(mask.astype(float), 0.5), key=len)
z = c[:, 1] + 1j * c[:, 0]                          # boundary as complex numbers x + jy

def fourier_descriptor(z, n=16):
    a = np.fft.fft(z)
    a[0] = 0                                        # drop DC   -> translation invariance
    scale = np.abs(a[1])
    if scale <= 1e-12 * np.linalg.norm(a):
        raise ValueError("first harmonic is too small for this normalization")
    a = a / scale                                   # divide by |a(1)| -> scale invariance
    return np.abs(np.r_[a[1:n + 1], a[-n:]])     # magnitudes -> rotation and start-point
     ↪  invariance

f1 = fourier_descriptor(z)
f2 = fourier_descriptor(0.5 * np.exp(1j * 0.7) * np.roll(z, 300) + (40 + 10j))
print(np.abs(f1 - f2).max())                        # ~1e-16: same descriptor
```

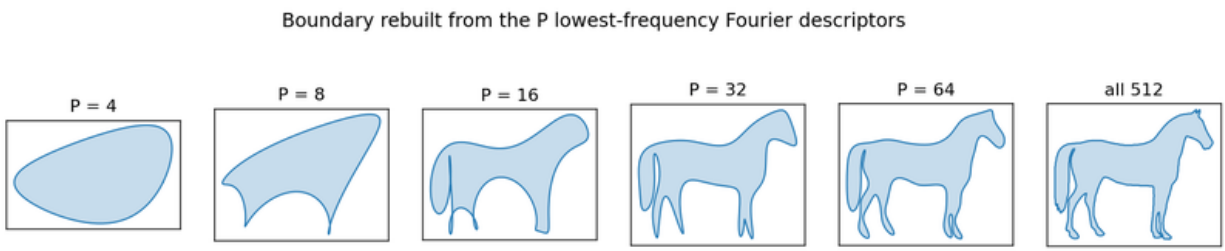


圖 *12.2* — 將馬的邊界重新取樣成 *512* 點，再用最低頻的 P 個傅立葉描述子重建。$P = 4$ 時只是一團橢圓；到 $P = 32$ 時腿、脖子和頭都已經認得出來；其餘係數主要在補小尺度的細節。

邊界的統計動差

白話說：把一段邊界或一個標記圖當成一維曲線，用幾個統計動差（例如分散程度與不對稱性）來概括它 [1]。動差計算便宜、物理意義清楚，而且（若在旋轉正規化後的標記圖上計算）對旋轉不敏感。

僅線上版：一維邊界函數的動差 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/一維邊界函數的動差/*

區域特徵描述子

白話版。除了描邊，也可以看整塊區域：多大、多圓、有幾個洞，以及內部看起來如何（平滑、粗糙、有條紋）。

基本描述子

- **面積** A：區域中的像素數。
- **周長** P：邊界的長度。
- **緊緻度**（compactness）：P^2/A。它沒有單位，所以對尺度不變（在數位化誤差範圍內也對平移與旋轉不變）。圓盤使它最小。
- **圓度**（circularity）：$R_c = 4\pi A/P^2$。圓盤為 1，越細長或越不規則就越小，等於緊緻度的倒數再乘上常數。
- **等效直徑**：$d_e = 2\sqrt{A/\pi}$，即同面積圓盤的直徑。
- **離心率**（用動差定義）：設 $\lambda_1 \geq \lambda_2$ 為像素座標共變異數矩陣的特徵值，$e = \sqrt{1 - \lambda_2/\lambda_1}$。圓盤為 0，線段趨近 1。（scikit-image 採用此定義。）

```
import numpy as np
from skimage import data, filters, measure, morphology, segmentation

img = data.coins()
bw = img > filters.threshold_otsu(img)
bw = morphology.binary_closing(bw, morphology.disk(2))
bw = segmentation.clear_border(morphology.remove_small_objects(bw, 200))
for r in measure.regionprops(measure.label(bw))[:4]:
    circ = 4 * np.pi * r.area / r.perimeter ** 2  # 1 for a perfect disk
    print(f"area={r.area:5.0f}  circularity={circ:.2f}  "
          f"eccentricity={r.eccentricity:.2f}  euler={r.euler_number}  "
          f"hu1={r.moments_hu[0]:.4f}")
```

在 coins 影像上，分割良好的硬幣圓度約 0.9、離心率約 0.3–0.4（數位周長會略為高估，所以即使是完美的數位圓盤，得分也會比 1 低一點）。

拓撲描述子

拓撲研究的是在「橡皮膜」變形下保持不變的性質：可以拉伸、彎折，但不能撕開或黏合。面積和周長不是拓撲性質；連通塊的數目和洞的數目才是。**尤拉數**（Euler number）定義為

$$E = C - H,$$

其中 C 是連通成分數，H 是洞數 [1]。字母「A」的 $E = 0$（一個成分、一個洞）；「B」的 $E = -1$。注意 E 會受前景與背景所用的連通定義（4 或 8 連通）影響。

僅線上版：多邊形網路的尤拉公式 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/多邊形網路的尤拉公式/*

紋理

白話版。紋理是表面在眼睛裡的「觸感」：平滑、粗糙、規則。我們用像素值的統計，以及「像素值彼此如何相鄰」的統計來描述它。

直方圖的統計動差。設 z 為代表強度的隨機變數，$p(z_i)$ 為正規化直方圖，$i = 0, \dots, L-1$，平均值 $m = \sum_i z_i p(z_i)$。則 [1]：

$$\mu_n(z) = \sum_{i=0}^{L-1} (z_i - m)^n \, p(z_i) \qquad (n\text{-th central moment})$$

$$R(z) = 1 - \frac{1}{1 + \sigma^2(z)} \qquad (\text{relative smoothness; } \sigma^2 = \mu_2)$$

$$U(z) = \sum_{i=0}^{L-1} p^2(z_i) \qquad (\text{uniformity})$$

$$e(z) = -\sum_{i=0}^{L-1} p(z_i) \log_2 p(z_i) \qquad (\text{entropy})$$

R 對常數區域為 0，變異數越大越接近 1（先把 σ^2 正規化到 $[0,1]$，例如除以 $(L-1)^2$）。μ_3 量的是直方圖的偏斜。這些量完全不管像素**在哪裡**，所以棋盤格和把它打亂後的影像會得到一模一樣的值。

共生矩陣。為了描述空間排列，Haralick、Shanmugam 與 Dinstein 提出了灰階共生矩陣（gray-level co-occurrence matrix, GLCM）[3]。先固定一個**位置運算子** Q，例如「右邊一個像素」。G 是 $L \times L$ 矩陣，元素 g_{ij} 計算「灰階為 i 的像素，在 Q 指定的位置上出現灰階為 j 的像素」的次數。令 $n = \sum_{i,j} g_{ij}$，

$$p_{ij} = g_{ij} / n,$$

就是該像素對出現的機率估計。由 p_{ij} 可以算出幾個好用的描述子（m_r, m_c 與 σ_r, σ_c 分別是列、行邊際分布的平均值與標準差）：

$$\text{Contrast} = \sum_{i,j} (i-j)^2 \, p_{ij} \qquad \text{Homogeneity} = \sum_{i,j} \frac{p_{ij}}{1 + |i-j|}$$

$$\text{Uniformity (energy)} = \sum_{i,j} p_{ij}^2 \qquad \text{Entropy} = -\sum_{i,j} p_{ij} \log_2 p_{ij}$$

$$\text{Correlation} = \sum_{i,j} \frac{(i - m_r)(j - m_c) \, p_{ij}}{\sigma_r \sigma_c}$$

（scikit-image 的 `graycoprops` 所回報的 “energy” 是均勻度總和的平方根。）質量集中在對角線附近，表示鄰居的值相近（平滑紋理：均質性高、對比低）；質量遠離對角線，表示變化劇烈（對比高）。實務上會先量化到少數幾階（8–32 階），對多個角度與距離計算 G，再對角度取平均，以得到近似的旋轉不變性。

```
import numpy as np
from skimage import data
```

```
from skimage.feature import graycomatrix, graycoprops

for name in ["brick", "grass", "gravel"]:
    q = (getattr(data, name)()[:256, :256] // 32).astype(np.uint8)   # 8 gray levels
    P = graycomatrix(q, distances=[1], angles=[0, np.pi/4, np.pi/2, 3*np.pi/4],
                     levels=8, symmetric=True, normed=True)
    feats = {p: graycoprops(P, p).mean() for p in ["contrast", "homogeneity", "energy",
     ↪  "correlation"]}
    print(name, {k: round(float(v), 3) for k, v in feats.items()})
```

```
brick  {'contrast': 0.269, 'homogeneity': 0.883, 'energy': 0.579, 'correlation': 0.817}
grass  {'contrast': 0.995, 'homogeneity': 0.717, 'energy': 0.288, 'correlation': 0.664}
gravel {'contrast': 0.669, 'homogeneity': 0.772, 'energy': 0.323, 'correlation': 0.78}
```

磚牆以大片平坦的磚面為主，GLCM 集中在少數幾個對角線格子：能量高、對比低。草地每根草葉都不同，GLCM 散得很開：對比最高、能量最低。光是四個數字，就足以把三種紋理分開（圖 12.3）。

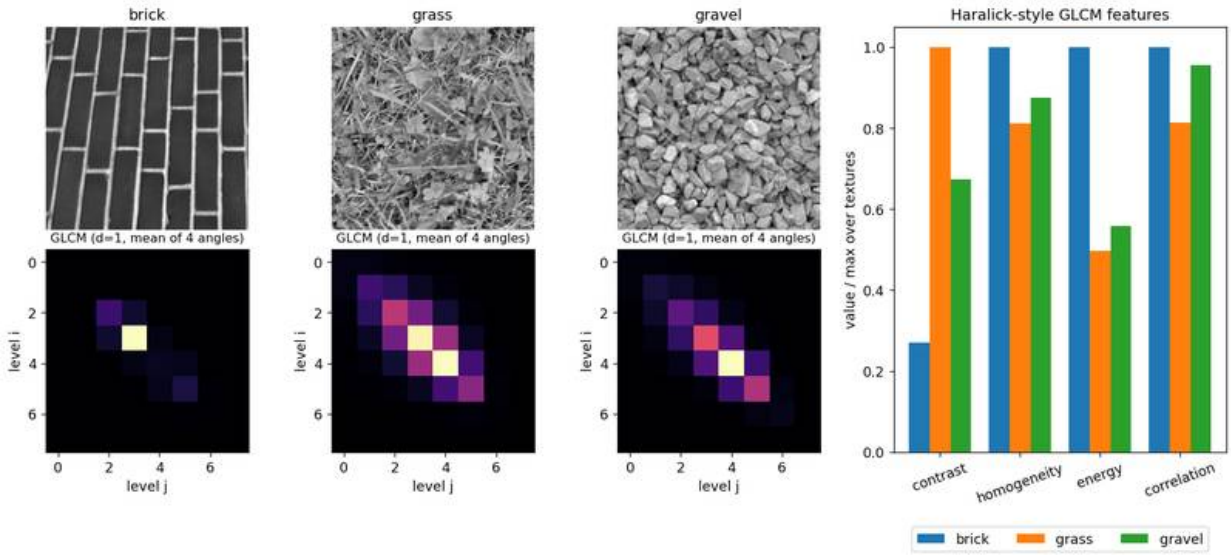


圖 *12.3* — 三種 *skimage.data* 紋理的共生矩陣特徵。上排：*256* × *256* 的區塊。下排：距離 *1*、四個角度平均後的 *GLCM*（越亮表示該像素對越常出現）。右：四個 *Haralick* 式特徵，各自除以三種紋理中的最大值，以便畫在同一個座標軸上。

頻譜紋理。週期性或有方向性的紋理，會在傅立葉頻譜上產生強烈而集中的峰值。把頻譜寫成極座標 $S(r, \theta)$，r 是到原點的距離（頻率），θ 是方向。兩個一維函數可以摘要它 [1]：

$$S(r) = \sum_{\theta=0}^{\pi} S_\theta(r), \qquad S(\theta) = \sum_{r=1}^{R_0} S_r(\theta),$$

其中 $S_\theta(r)$ 是角度 θ 那條射線上的頻譜，$S_r(\theta)$ 是半徑 r 的半圓上的頻譜，R_0 是考慮的最大半徑。$S(r)$ 的峰值揭露圖樣的週期（例如磚塊的間距）；$S(\theta)$ 的峰值揭露主要方向。只需要半圈，因為實數影像的頻譜對原點對稱。

動差不變量

白話說：動差概括區域的「質量」如何分布在中心周圍；把它們適當組合，就能在區域被平移、縮放或旋轉時保持不變。

對大小為 $M \times N$ 的二維影像（或區域指示函數）$f(x,y)$，$(p+q)$ 階動差為

$$m_{pq} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} x^p y^q f(x,y),$$

中心動差為

$$\mu_{pq} = \sum_{x} \sum_{y} (x - \bar{x})^p (y - \bar{y})^q f(x,y), \qquad \bar{x} = \frac{m_{10}}{m_{00}},\ \bar{y} = \frac{m_{01}}{m_{00}},$$

其中 $(\bar{x}, \bar{y})$ 是質心。中心動差對平移不變。**正規化中心動差**

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \qquad \gamma = \frac{p+q}{2} + 1,$$

則再加上尺度不變。Hu [2] 從二階與三階的 η_{pq} 推導出七個對平移、尺度**以及旋轉**都不變的組合。前兩個是

$$\phi_1 = \eta_{20} + \eta_{02}, \qquad \phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2.$$

ϕ_1 是對質心的正規化「轉動慣量」：質量離中心越遠就越大。ϕ_2 量的是細長程度。第七個不變量在鏡射下會變號，因此可以分辨形狀與它的鏡像。由於高階動差會放大雜訊，而且數值跨越好幾個數量級，一般會比較 $\operatorname{sign}(\phi_i)\log|\phi_i|$。在 scikit-image 中，`regionprops(...).moments_hu` 會傳回全部七個；OpenCV 則是 `cv2.HuMoments(cv2.moments(mask))`。

以主成分作為特徵描述子

白話版。量測一個物體很多項時，有些量測其實在重複講同一件事。PCA 會找出一組新的座標軸，從「最有資訊」排到「最沒資訊」，讓我們只留前幾個、丟掉其他。對二維形狀來說，第一個軸就是形狀最長的那個方向。

精確版。設 $\mathbf{x}_1, \dots, \mathbf{x}_K$ 是 n 維向量，例如多光譜影像中每個像素的 n 個波段值，或某區域中每個像素的 (x,y) 座標。它們的平均與共變異數為

$$\mathbf{m}_x = \frac{1}{K}\sum_{k=1}^{K} \mathbf{x}_k, \qquad \mathbf{C}_x = \frac{1}{K-1}\sum_{k=1}^{K} (\mathbf{x}_k - \mathbf{m}_x)(\mathbf{x}_k - \mathbf{m}_x)^{\mathsf{T}}.$$

$\mathbf{C}_x$ 是實對稱矩陣，因此有 n 個正交單位特徵向量 $\mathbf{e}_i$，對應特徵值 $\lambda_1 \ge \lambda_2 \ge \dots \ge \lambda_n \ge 0$。把特徵向量當成矩陣 $\mathbf{A}$ 的各列。**Hotelling 轉換**（也稱離散 Karhunen–Loève 轉換，或 PCA）為

$$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m}_x).$$

新向量的平均為零，共變異數是**對角**矩陣 $\mathbf{C}_y = \mathbf{A}\mathbf{C}_x\mathbf{A}^{\mathsf{T}} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$：$\mathbf{y}$ 的各分量互不相關，第 i 軸上的變異數為 λ_i。若只保留前 k 列 $\mathbf{A}_k$，重建 $\hat{\mathbf{x}} = \mathbf{A}_k^{\mathsf{T}}\mathbf{y} + \mathbf{m}_x$ 的均方誤差為

$$e_{\text{ms}} = \sum_{j=k+1}^{n} \lambda_j,$$

也就是被丟掉的特徵值總和。在這個意義下，沒有其他投影到 k 維的線性方法做得更好。

在影像處理中的三種用途 [1]：

1. **壓縮多光譜影像**。有 n 個對齊好的波段時，每個像素是一個 n 維向量。大部分變異通常集中在前兩、三個主成分影像，因此可以用它們代替全部 n 個波段來顯示或分類。
2. **正規化物體姿態**。把區域的像素座標當作向量，$\mathbf{y} = \mathbf{A}(\mathbf{x} - \mathbf{m}_x)$ 會把質心移到原點，並把區域轉到長軸與第一座標軸對齊（圖 12.4）。之後算出的描述子就與平移、旋轉無關；再除以 $\sqrt{\lambda_1}$ 就連尺度也無關。要注意：特徵向量的正負號是任意的，轉正後的物體可能被翻轉；可以用「三階動差較重的一側朝右」之類的規則決定。
3. **特徵影像**（eigen-images）。把同一類物體的大量同尺寸影像各自攤平成向量，前幾個特徵向量重新排回影像形狀，就是涵蓋該組影像大部分變化的「特徵影像」。任何新影像都可以用少數幾個座標 $\mathbf{y}$ 描述，成為精簡的辨識特徵。

```
import numpy as np
from skimage import data, transform

mask = transform.rotate((data.horse() == 0).astype(float), 35, resize=True) > 0.5
X = np.argwhere(mask)[:, ::-1].astype(float)     # (x, y) of every object pixel
m = X.mean(axis=0)
C = np.cov((X - m).T)                            # 2x2 covariance matrix
evals, evecs = np.linalg.eigh(C)                 # ascending eigenvalues
A = evecs[:, ::-1].T                             # rows = eigenvectors, largest first
Y = (X - m) @ A.T                                # Hotelling transform y = A(x - m)
print(np.round(np.cov(Y.T), 3))                  # diagonal: the axes are decorrelated
```

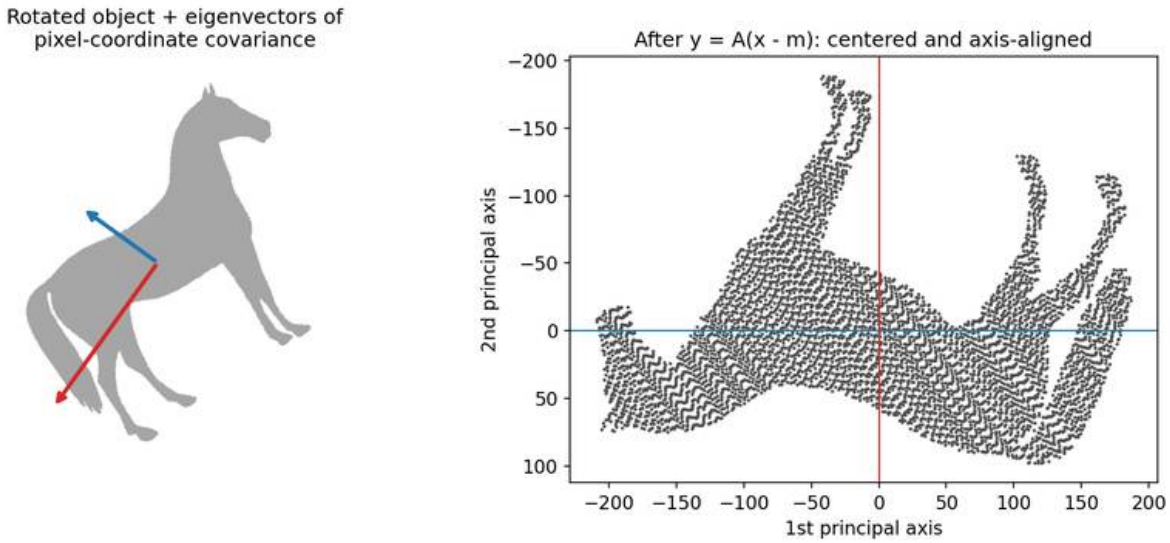


圖 *12.4* — 用 *PCA* 正規化姿態。左：旋轉 *35°* 的馬，箭頭是座標共變異數的特徵向量（長度 =

$2\sqrt{\lambda}$）。右：經 $y = A(x - m)$ 轉換後的像素座標。物體已置中，最長的方向落在水平軸上；它看起來是翻過來的，因為每個特徵向量的正負號是任意的。

整張影像的特徵

白話版。前面都需要先分割出物體。但很多任務（全景接圖、追蹤、三維重建）需要的是**地標**：能在任何照片中自動找到，並且能在同一場景的另一張照片中再找到。角點和穩定的斑塊就是好地標。

Harris–Stephens 角點偵測器

白話版。透過一個小窗口看影像，然後把窗口稍微挪一下。在平坦區域，什麼都沒變；在邊緣上，沿著邊緣挪不會變，橫越邊緣才會變；在角點上，往**任何**方向挪，看到的都會變。Harris 與 Stephens 把這個想法寫成公式 [4]。

精確版。窗口 w 位移 (u, v) 所造成的變化為

$$E(u,v) = \sum_{x,y} w(x,y)\,[f(x+u, y+v) - f(x,y)]^2 \approx [u \;\; v]\,\mathbf{M}\begin{bmatrix} u \\ v \end{bmatrix},$$

這裡用了一階泰勒展開，**結構張量**（structure tensor，又稱二階動差矩陣）為

$$\mathbf{M} = \sum_{x,y} w(x,y)\begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix},$$

其中 f_x, f_y 是影像的偏導數，w 是窗函數（通常為高斯）。$\mathbf{M}$ 的特徵值 λ_1, λ_2 說明一切：兩個都小是平坦區，一大一小是邊緣，兩個都大是角點。1988 年時逐像素算特徵值太昂貴，所以 Harris 與 Stephens 改用**響應函數**

$$R = \det(\mathbf{M}) - k\,\operatorname{tr}(\mathbf{M})^2 = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2,$$

k 是一個小常數（常見值約 0.04–0.06）。R 在角點為大的正值，在邊緣為負值，在平坦區接近零。角點就是 R 在非極大值抑制後、超過門檻的局部極大值。

性質：$\mathbf{M}$ 由導數組成，所以 R 不受亮度加減的影響；它的特徵值與旋轉無關，所以偵測器對旋轉共變。但它**不**對尺度共變：在小窗口中看到的角點，換成大窗口可能就像一條邊緣。補上尺度的正是 SIFT。

```
import numpy as np
from scipy import ndimage as ndi
from skimage import data, feature, util

def harris(img, sigma_d=1.0, sigma_i=1.5, k=0.05):
    Ix = ndi.gaussian_filter(img, sigma_d, order=(0, 1))   # derivative along x (columns)
    Iy = ndi.gaussian_filter(img, sigma_d, order=(1, 0))   # derivative along y (rows)
    Sxx = ndi.gaussian_filter(Ix * Ix, sigma_i)            # entries of the structure tensor
     ↪  M
    Syy = ndi.gaussian_filter(Iy * Iy, sigma_i)
```

```
    Sxy = ndi.gaussian_filter(Ix * Iy, sigma_i)
    det, tr = Sxx * Syy - Sxy ** 2, Sxx + Syy
    return det - k * tr ** 2                                  # R > 0 corner, R < 0 edge, |R|
     ↪  small flat

img = util.img_as_float(data.camera())
R = harris(img)
corners = feature.corner_peaks(R, min_distance=7, threshold_rel=0.02)   # (row, col) list
```

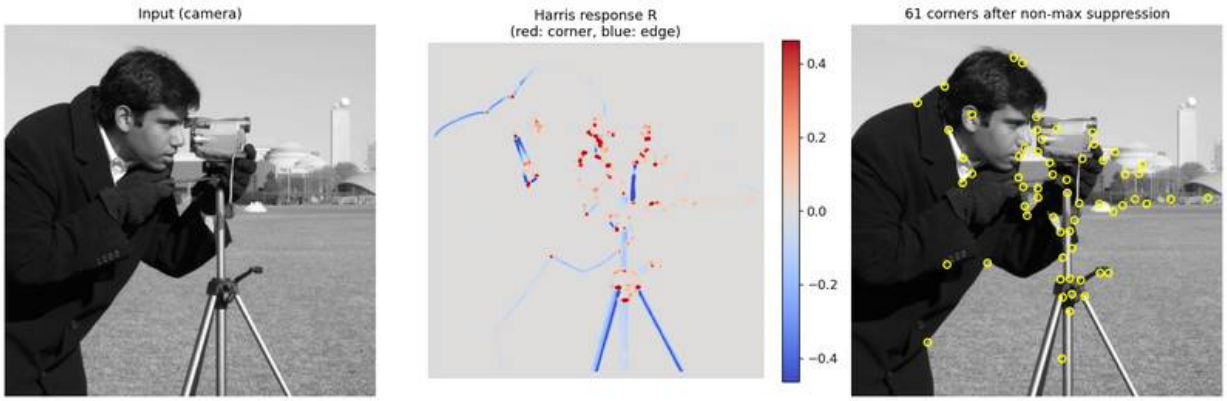


圖 *12.5 — skimage.data.camera* 上的 *Harris–Stephens* 角點（$k = 0.05$，積分 $\sigma = 1.5$）。中：響應 R；強烈的正值（紅）集中在相機機身與腳架關節，而腳架的腳屬於邊緣，呈負值（藍）。右：非極大值抑制後 R 的局部極大值。

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

最大穩定極值區域（MSER）

白話版。想像把一片灰階地形慢慢灌水。白紙上的黑字會變成一個個小湖。水位上升時，大部分湖泊會迅速擴大或彼此合併，但字母形成的湖，在很長一段水位範圍內都維持同樣形狀，因為墨水比周圍的紙暗得多。水位變化很大、形狀卻幾乎不變的區域，就是**最大穩定**的區域。Matas、Chum、Urban 與 Pajdla 為寬基線立體視覺提出了這個方法 [5]。

精確版。在每個灰階 t 對影像做門檻化。$\{f \le t\}$（亮區域則為 $\{f \ge t\}$）的一個連通成分稱為**極值區域** Q_t：區域內每個像素都比它外圍邊界上的每個像素暗（或亮）。隨著 t 增加，成分逐漸長大、合併，形成一棵樹。對每個區域定義穩定度

$$q(t) = \frac{|Q_{t+\Delta} \setminus Q_{t-\Delta}|}{|Q_t|},$$

其中 $|\cdot|$ 是面積，Δ 是灰階步長。$q(t)$ 出現局部最小值時，該區域就是**最大穩定**的。把像素依強度排序後，用 union-find 結構就能在接近線性的時間內建出整棵樹。

為什麼有效：這個定義只用到強度的**大小順序**，所以 MSER 對任何單調的亮度變化都不變。在連續的幾何變換下，連通區域仍映射成連通區域，所以 MSER 對仿射變換共變（常見做法是對每個

區域擬合橢圓，再正規化成圓形後才描述）。MSER 最適合邊界清楚的均勻斑塊，例如文字、標誌、窗戶；在模糊或紋理繁雜的場景中表現較差。

```
import cv2
from skimage import data

img = data.page()                                        # uint8 grayscale
mser = cv2.MSER_create(delta=5, min_area=10, max_area=800, max_variation=0.5)
regions, boxes = mser.detectRegions(img)                 # pixel lists + bounding boxes
print(len(regions), "regions")
```

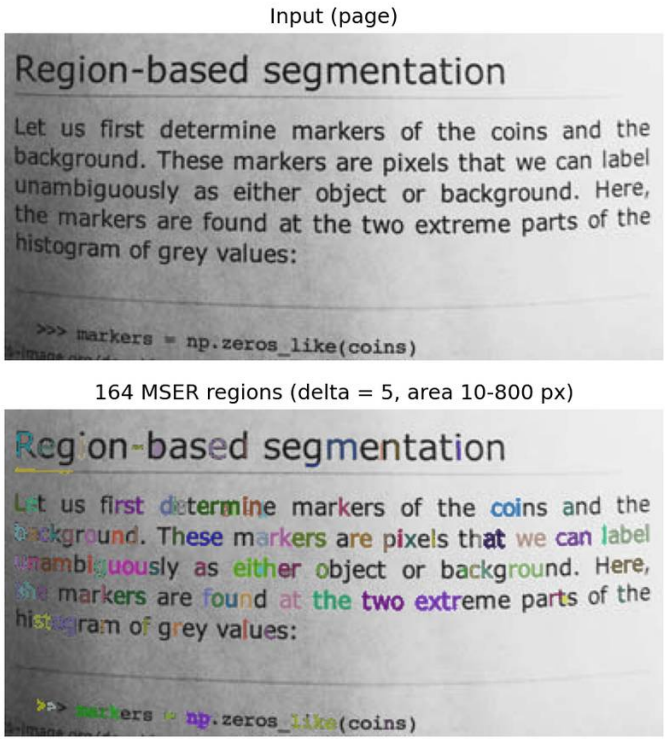


圖 *12.6* — 光照不均的 *skimage.data.page* 上的 *MSER*。每個上色的區域都是在 ± 5 個灰階內保持穩定的極值區域，大多是單一字母或黏在一起的字母群；由於穩定度只取決於強度順序，左下角較暗的部分並不造成困擾。

尺度不變特徵轉換（SIFT）

白話版。 Lowe 提出的 SIFT [6] 對每個地標回答三個問題：它**在哪裡**、它**多大**、它**朝哪個方向**？接著在這個地標自己的大小與方向下，寫下周圍梯度的摘要。這讓摘要較不受尺度、平面內旋轉與對比變化影響；劇烈視角或照明改變仍可能使它改變。

僅線上版：*SIFT*：先讓局部特徵可重複偵測，再進行比對 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/papers/sift 先讓局部特徵可重複偵測再進行比對/*

尺度空間

為了找出各種大小的特徵，SIFT 先建立**尺度空間**（scale space）：用寬度越來越大的高斯核模糊影像，

$$L(x, y, \sigma) = G(x, y, \sigma) \star f(x, y), \qquad G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2},$$

其中 $\star$ 表示卷積，σ 是尺度。尺度分成好幾個**八度**（octave）；每個八度讓 σ 加倍，並切成 s 個間隔，所以相鄰尺度相差 $k = 2^{1/s}$ 倍。Lowe [6] 使用 $s = 3$ 與基礎尺度 $\sigma_0 = 1.6$。每完成一個八度就把影像降取樣一半，以節省計算。

以高斯差偵測關鍵點

白話說：把兩張模糊程度不同的影像相減，會凸顯某個特定大小的斑點；關鍵點就是這個響應最強的位置。

把相鄰的模糊影像相減：

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma).$$

這個**高斯差**（difference of Gaussians, DoG）非常接近尺度正規化的高斯拉普拉斯 $\sigma^2\nabla^2 G$，只差常數倍 $(k-1)$。它對大小與 σ 相符的斑塊反應最強（圖 12.7）。候選關鍵點是 D 值比全部 26 個鄰居都大或都小的像素：同一張 DoG 影像中的 8 個，加上上下兩個尺度各 9 個。因此每個關鍵點同時帶有位置**和**特徵尺度。

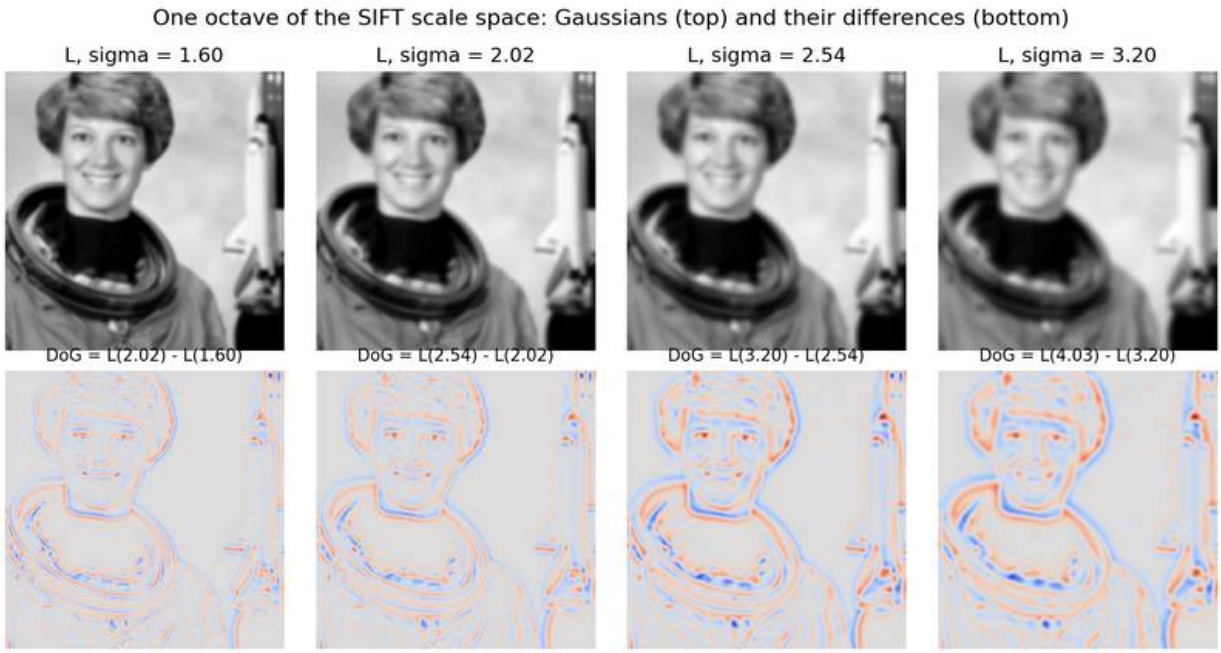


圖 *12.7* — 在 *skimage.data.astronaut* 的局部上建立的一個 *SIFT* 式八度（$\sigma_0 = 1.6$，每個八度三個間隔）。下排為 *DoG* 影像：細結構（髮絲、眼睛）在小 σ 有反應，大結構（頭盔環、火箭箭身）在

大 σ 有反應。關鍵點是同時在空間與相鄰 *DoG* 影像間取極值的點。

僅線上版：關鍵點定位 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/關鍵點定位/*

方向指定

白話說：替每個關鍵點指定一個方向，也就是它周圍最主要的梯度方向，之後的量測都相對於這個方向進行。

在關鍵點附近、以它的尺度在 L 上計算梯度大小與方向，建立 36 格（每格 10°）的方向直方圖；每個樣本以梯度大小加權，並乘上 σ 為關鍵點尺度 1.5 倍的高斯窗。最高峰就是關鍵點的方向；其他超過最高峰 80% 的峰值，會產生同位置但方向不同的額外關鍵點 [6]。此後所有量測都相對於這個方向進行，因而得到旋轉不變性。

關鍵點描述子

白話說：在關鍵點自己的大小與方向下，用一小格一小格的方向直方圖來概括它周圍的梯度。

在關鍵點周圍取一個窗口，依其方向旋轉、依其尺度決定大小，再切成 4×4 個格子。每個格子累積一個 8 格的梯度方向直方圖，以梯度大小與以關鍵點為中心的高斯加權，並用三線性內插，避免小小的位移造成數值在格子之間突然跳動。這樣得到 $4 \times 4 \times 8 = 128$ 個數字 [6]。最後：

1. 把向量正規化成單位長度（抵消 $f \mapsto af$ 的對比變化；梯度本身已抵消 $+b$）；
2. 把每個元素截在 0.2 以下（限制少數超大梯度的影響，例如反光等非線性光照）；
3. 再正規化一次。

配對

兩張影像的描述子用歐氏距離比較。只看最近鄰並不可靠：很多關鍵點根本沒有真正的對應點。Lowe 的**比值測試**（ratio test）只在

$$\frac{\|\mathbf{d} - \mathbf{d}_1\|}{\|\mathbf{d} - \mathbf{d}_2\|} < \tau$$

時保留配對，其中 $\mathbf{d}_1, \mathbf{d}_2$ 是另一張影像中最近與次近的描述子，τ 是門檻（Lowe 建議 0.8）[6]。有鑑別力的配對會比第二名近得多。大型資料庫會改用近似最近鄰搜尋，而留下來的配對通常還要經過幾何驗證，例如用 RANSAC（random sample consensus，隨機取樣一致）[21] 擬合相似、仿射或投影轉換，只保留內點（inlier）。「補充資料 (https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/ransac 在有離群值時擬合模型/)」頁介紹了 RANSAC。

```
import numpy as np
from skimage import data, feature, transform
from skimage.color import rgb2gray
from skimage.measure import ransac
```

```
img1 = rgb2gray(data.astronaut())
tf = transform.AffineTransform(scale=0.7, rotation=np.deg2rad(25), translation=(140, -60))
img2 = 0.8 * transform.warp(img1, tf.inverse) + 0.1        # second "view"

sift = feature.SIFT()
sift.detect_and_extract(img1); k1, d1 = sift.keypoints, sift.descriptors
sift.detect_and_extract(img2); k2, d2 = sift.keypoints, sift.descriptors

matches = feature.match_descriptors(d1, d2, max_ratio=0.6, cross_check=True)  # ratio test
src, dst = k1[matches[:, 0]][:, ::-1], k2[matches[:, 1]][:, ::-1]           # (row, col) ->
 ↪  (x, y)
model, inliers = ransac((src, dst), transform.SimilarityTransform,
                        min_samples=3, residual_threshold=2, max_trials=500)
print(len(matches), "matches,", inliers.sum(), "inliers")
print("recovered scale %.3f, rotation %.1f deg" % (model.scale, np.degrees(model.rotation)))
```

在這組合成影像上，程式找到 484 個配對，其中 483 個與真正的轉換一致，並還原出尺度 0.700、旋轉 25.0°。真實照片有視角變化、遮擋和重複圖樣，內點比例會低得多，所以幾何驗證絕對不能省。

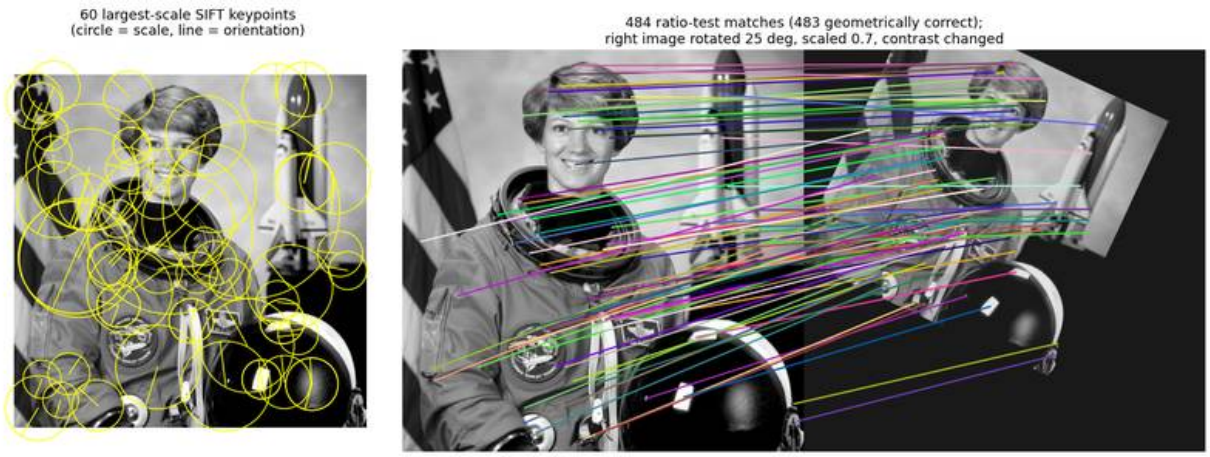


圖 *12.8 — skimage.data.astronaut* 上的 *SIFT*。左：尺度最大的 *60* 個關鍵點（圓半徑 ∝ 尺度，線段 = 方向）。右：與旋轉 *25°*、縮放 *0.7* 並改變對比的複本之間的部分比值測試配對；*484* 個配對中有 *483* 個與真實轉換的誤差在 *3* 像素以內。

> **去哪裡找 SIFT**
>
> scikit-image 在 0.19 版加入了 SIFT 偵測器與描述子（即上面用到的 `skimage.feature.SIFT`）。在 OpenCV 中，先呼叫 `cv2.SIFT_create()`，再用 `detectAndCompute(gray, None)`。

現代觀點

綜述論文怎麼說

形狀描述子。 Zhang 與 Lu [9] 把形狀描述子分成以輪廓為主或以區域為主、全域式或結構式；本章的邊界與區域描述子都能放進這個表格，而且沒有一種描述子適用於所有應用。

局部特徵偵測器。 Tuytelaars 與 Mikolajczyk [10] 以好的局部特徵應具備的性質來組織偵測器，其中最重要的是可重複性，可透過不變性或強健性達成；有些性質彼此競爭，該怎麼取捨要看應用。SIFT 重速度的後繼者正是這種取捨的例子：SURF [7] 在積分影像上使用方框濾波器，ORB [8] 則結合 FAST 角點測試與二元描述子。

紋理。 Liu 等人 [11] 回顧了約二十年的紋理表示法，從以詞袋編碼的手工局部描述子，一路到以 CNN 為基礎的表示法；共生統計 [3] 位在這段歷史的源頭。

影像配對：從手工到深度。 Ma 等人 [12] 指出學習已經進入配對流程的每一個階段。Jin 等人 [13] 以最終相機姿態的準確度評分，發現只要設定調好，像 SIFT 這樣的傳統流程仍可能勝過被視為最先進的方法。

僅線上版：現代觀點綜述導讀 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/papers/現代觀點綜述導讀/*

深度學習如何改變特徵擷取

學習式關鍵點與描述子。 SuperPoint [14] 訓練一個網路同時輸出關鍵點熱度圖與稠密描述子，並透過**單應性調適**（homographic adaptation）自我監督；DISK [16] 則用強化學習端到端地訓練偵測與描述。它們用學來的函數取代 Harris 與 SIFT 的公式，但保留了相同的概念：稀疏、可重複的點，加上描述子。

學習式配對。 SuperGlue [15] 與它更快的後繼者 LightGlue [18] 讓兩張影像的關鍵點彼此注意，取代比值測試，並解一個也能回答「沒有配對」的指派問題；LoFTR [17] 拿掉偵測器，以 Transformer 在稠密特徵圖上配對，對低紋理區域特別有幫助。

通用深度特徵。像 DINOv2 [19] 這樣的自我監督基礎模型所產生的區塊特徵，不需微調就能用於許多任務，而 RoMa [20] 等配對方法就建立在它之上：SIFT 核心那個手工設計的描述子，已經被一個在極大量影像上訓練出的網路取代。

僅線上版：學習式特徵的細節 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/feature-extraction/notes/學習式特徵的細節/*

沒有改變的部分。本章的**問題拆解方式**依然存在。現代流程仍然是偵測、描述、配對、再做幾何驗證，而且通常仍使用 RANSAC 類的強健估計器 [21]。尺度與旋轉仍然必須處理，不是內建（SIFT 的尺度空間、ORB 的方向），就是透過資料擴增讓模型學會。不變性依然是設計選擇：過度不變的描述子會失去鑑別力，無論它是手工設計還是學出來的。在以量測為主的領域（材料、顯微影像、品質檢測），面積、圓度、尤拉數與 GLCM 統計等區域描述子仍然很受歡迎，因為每個數字都有人可以檢查的物理意義。

重點整理

- **偵測器**找出特徵（點、區域、邊界）；**描述子**把每個特徵轉成向量。偵測器應對幾何變化共變，描述子應對任務中無關緊要的變化不變。
- 邊界要先排序（邊界追蹤）、再簡化（鏈碼、最小周長多邊形、合併與分裂、標記圖、骨架），之後才拿來量測。
- **傅立葉描述子**用少數係數壓縮封閉輪廓；丟掉 $a(0)$、除以 $|a(1)|$、只取大小，就得到平移、尺度、旋轉與起點不變性。
- 區域描述子從大小與形狀（面積、圓度、離心率）、拓撲（尤拉數 $E = C - H$）、紋理（直方圖動差、**GLCM** 特徵、頻譜）一路到 **Hu 動差不變量**。
- **PCA**（Hotelling 轉換）讓量測去相關、把物體轉到主軸方向，並提供精簡的特徵影像描述子；丟掉分量造成的誤差等於被丟掉的特徵值總和。
- **Harris** 由結構張量找角點；**MSER** 找在多個門檻下都穩定的區域；**SIFT** 加上尺度空間、方向與正規化的 128 維梯度直方圖，再以比值測試配對。
- 學習式偵測器、配對器與基礎模型特徵如今是多數配對研究的焦點，但仔細調整的傳統流程仍是強勁的基準線，而「偵測—描述—配對—驗證」的架構並未改變。

練習

1. **手算鏈碼**。寫出一個與座標軸對齊、寬 3 節、高 2 節的長方形邊界的 8 方向鏈碼，從左下角出發、逆時針行走。算出它的一階差分與形狀數（最小循環位移）。再把長方形轉 90°，說明形狀數不變。

   提示

   從左下角逆時針走（0 = 東、2 = 北）：0 0 0 2 2 4 4 4 6 6。一階差分的每個元素是相對前一個方向逆時針轉了幾格：2 0 0 2 0 2 0 0 2 0（開頭的 2 來自首尾相接的 6 → 0）。最小循環位移為 0 0 2 0 2 0 0 2 0 2。轉 90° 後每個碼都加 2（mod 8），差分不變。

2. **實測傅立葉描述子的不變性**。使用本章的 `fourier_descriptor` 函數。(a) 用數值說明它無法分辨形狀與其鏡像。(b) 從轉換效果表解釋原因，並提出一個能分辨鏡像的修改。

   提示

   用 `mask[:, ::-1]` 翻轉遮罩，並用 `find_contours` 追蹤兩條輪廓；兩組描述子的差距約在 10^{-4} 以內。追蹤器總是以相同的旋轉方向走過每條邊界，因此鏡像邊界是 $s'(k) = \overline{s(-k)}$（共軛且反向），最多差一個位移。它的 DFT 是 $\overline{a(u)}$：大小相同、相位相反，所以只看大小看不出鏡射。改為保留部分相位：複數 $a(u)\,a(2-u)/a(1)^2$（索引取模 K）不受平移、旋轉、尺度與起點影響，但在鏡射下會變成共軛，因此它虛部的正負號就能區分形狀與其鏡像。

3. **格點上的圓度**。用 `skimage.draw.disk` 與 `regionprops` 計算半徑 5、10、20、50 像素的數位圓盤之 $4\pi A/P^2$。為什麼結果不完全等於 1？它如何隨半徑變化？也試試 `perimeter_crofton`。

提示

數位周長是階梯狀的，不同估計方法用不同方式修正。小圓盤的相對誤差較大。兩種周長估計都列出來，其中一種應該明顯更接近真正的 $2\pi r$。

4. **紋理的方向**。做一張週期 8 像素的水平條紋合成紋理，加一點雜訊。計算 0°、45°、90°、135° 四個角度、位移 1 像素的 GLCM 對比，以及角度頻譜標記 $S(\theta)$。兩者各自哪個角度最突出？為什麼兩種方法的結論一致？

提示

沿著條紋（0°）移動時強度幾乎不變，對比低；橫越條紋（90°）才會變。頻譜中，水平條紋的能量落在垂直頻率軸上。注意慣例：scikit-image 的角度 0 代表水平方向的像素位移。

5. **以特徵值理解 PCA**。對半軸 $a = 40$、$b = 10$ 的實心橢圓，預測座標共變異數特徵值的比 λ_1/λ_2，再用程式驗證。用你的答案說明 λ_1, λ_2 與 `regionprops` 回報的離心率之間的關係。

提示

均勻實心橢圓沿長度為 a 的半軸方向，變異數為 $a^2/4$。所以 $\lambda_1/\lambda_2 = (a/b)^2 = 16$，而 $e = \sqrt{1 - \lambda_2/\lambda_1} = \sqrt{1 - b^2/a^2} \approx 0.968$。

6. **比值測試的取捨**。使用 SIFT 配對程式，把 `max_ratio` 從 0.5 調到 0.95，畫出 (a) 配對數量與 (b) RANSAC 內點比例。接著加上 0.6 倍縮放、30° 旋轉與高斯雜訊（$\sigma = 0.05$）再做一次。你會把門檻設在哪裡？為什麼答案取決於雜訊？

提示

寬鬆的比值會收進更多配對，但錯的也更多；嚴格的比值只留下少數、多半正確的配對。RANSAC 能容忍大量離群值，但需要最低數量的正確配對，所以最佳門檻落在兩個極端之間，並隨影像品質移動。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 12. publisher page
2. M.-K. Hu, “Visual pattern recognition by moment invariants,” *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179–187, 1962. doi
3. R. M. Haralick, K. Shanmugam, and I. Dinstein, “Textural features for image classification,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, 1973. doi
4. C. Harris and M. Stephens, “A combined corner and edge detector,” in *Proc. Alvey Vision Conference*, pp. 23.1–23.6, 1988. doi
5. J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image and Vision Computing*, vol. 22, no. 10, pp. 761–767, 2004. doi

6. D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. doi
7. H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (SURF),” *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008. doi
8. E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” in *Proc. IEEE International Conference on Computer Vision (ICCV)*, pp. 2564–2571, 2011. doi
9. D. Zhang and G. Lu, “Review of shape representation and description techniques,” *Pattern Recognition*, vol. 37, no. 1, pp. 1–19, 2004. doi
10. T. Tuytelaars and K. Mikolajczyk, “Local invariant feature detectors: A survey,” *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008. doi
11. L. Liu, J. Chen, P. Fieguth, G. Zhao, R. Chellappa, and M. Pietikäinen, “From BoW to CNN: Two decades of texture representation for texture classification,” *International Journal of Computer Vision*, vol. 127, pp. 74–109, 2019. arXiv
12. J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, “Image matching from handcrafted to deep features: A survey,” *International Journal of Computer Vision*, vol. 129, pp. 23–79, 2021. doi
13. Y. Jin, D. Mishkin, A. Mishchuk, J. Matas, P. Fua, K. M. Yi, and E. Trulls, “Image matching across wide baselines: From paper to practice,” *International Journal of Computer Vision*, 2021. doi ·arXiv
14. D. DeTone, T. Malisiewicz, and A. Rabinovich, “SuperPoint: Self-supervised interest point detection and description,” arXiv:1712.07629, 2017. arXiv
15. P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, “SuperGlue: Learning feature matching with graph neural networks,” arXiv:1911.11763, 2019. arXiv
16. M. J. Tyszkiewicz, P. Fua, and E. Trulls, “DISK: Learning local features with policy gradient,” arXiv:2006.13566, 2020. arXiv
17. J. Sun, Z. Shen, Y. Wang, H. Bao, and X. Zhou, “LoFTR: Detector-free local feature matching with Transformers,” arXiv:2104.00680, 2021. arXiv
18. P. Lindenberger, P.-E. Sarlin, and M. Pollefeys, “LightGlue: Local feature matching at light speed,” arXiv:2306.13643, 2023. arXiv
19. M. Oquab, T. Darcet, T. Moutakanni, et al., “DINOv2: Learning robust visual features without supervision,” arXiv:2304.07193, 2023. arXiv
20. J. Edstedt, Q. Sun, G. Bökman, M. Wadenbäck, and M. Felsberg, “RoMa: Robust dense feature matching,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. arXiv
21. 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
22. scikit-image contributors, “skimage.measure.ransac,” *scikit-image 0.26 documentation (API reference)*. docs

IP 13・影像圖樣分類

從最小距離與貝氏分類器，到感知器、反向傳播與卷積神經網路：電腦如何判斷一張影像或一個區域是什麼。

網站版本: https://blog.shyandram.dev/zh-tw/learn/image-processing/pattern-classification/

你將學到

- 圖樣、圖樣向量與圖樣類別如何把「這張影像裡是什麼？」變成一個精確的決策問題
- 原型比對（prototype matching）怎麼運作：最小距離分類器、以相關進行的樣板比對、SIFT 特徵比對，以及結構描述的比對
- 為什麼貝氏分類器是最佳的，以及當每個類別都是高斯分布時它會變成什麼
- 感知器如何學習、多層網路如何用反向傳播訓練，以及卷積神經網路（CNN）如何自己學出特徵
- 訓練的實務面：過擬合、資料擴增與正則化
- 影像分類如何從 AlexNet 走到 Vision Transformer、CLIP 與 DINOv2，以及為什麼基準測試的數字值得懷疑

先看全貌

> **用最簡單的話說**
>
> 想像餐桌上有一堆水果，旁邊放了三個碗，分別寫著「蘋果」、「萊姆」、「柳丁」。你拿起每顆水果，看看顏色、摸摸大小，然後丟進它最像的那個碗。這就是圖樣分類。顏色和大小是**特徵**，碗是**類別**，你腦中那條規則（「紅色又大顆就是蘋果」）就是**分類器**。這一章講的就是怎麼建立那條規則：拿每種水果的「標準樣子」來比、用機率來算，或是讓一堆小小的加法機器組成的網路，從大量例子中自己學會規則。

第 10 到 12 章把影像切成區域，再用數字描述每個區域：面積、矩、傅立葉描述子、紋理統計量、關鍵點描述子。這一章替整條流程收尾：拿這些描述，替每一個指定一個**標籤**，例如「硬幣」、「數字 7」、「水體」、「腫瘤」、「貓」。幾乎所有實用的視覺系統最後都要做這樣的決定，所以這裡正是影像處理與機器學習交會的地方。

本章的順序既是歷史，也是觀念的推進 [1]。先是直接使用**原型**（每個類別的典型成員）的分類器；接著是**統計上最佳**的分類器，它需要知道各類別的機率分布；最後是**類神經網路**，它直接從資料學出決策規則。卷積形式的網路連特徵也一併學習，取代了 IP 12 大部分的人工特徵設計。若想先溫習監督式學習、成本函數與梯度下降，可以參考 ML 01・機器學習入門。

背景

白話版。分類器就是一個函數：看一份「描述」，回答「這東西屬於哪一組？」。我們用一批已知答案的例子來建立它。

圖樣（pattern）是代表某個物體或區域的一組描述子。**圖樣類別**（pattern class）是一群具有某種共同性質的圖樣。我們把 W 個類別記為 $\omega_1, \omega_2, \dots, \omega_W$。機器的**圖樣辨識**，就是自動把每個圖樣指派到它的類別，而且錯誤越少越好。

工作分成兩個階段：

1. **表示（representation）**。決定要量什麼：像素值、區域描述子、關鍵點描述子，或學出來的特徵。IP 12 處理的就是這一階段。
2. **決策（decision）**。有了量測值之後選出類別。本章處理這一階段。

方法通常以**監督式學習**訓練：手上有一組標籤已知的**訓練集**，用來設定分類器的參數；再用分類器從沒看過的**測試集**評估表現。第三份資料**驗證集**用來挑選設定，例如網路大小或何時停止訓練。把這幾份資料混用，是結果過度樂觀最常見的原因。

理解任何分類器的好方法是**決策函數**（decision function）。替每個類別 ω_j 定義一個函數 $d_j(\mathbf{x})$，再把 $\mathbf{x}$ 指派給函數值最大的類別：

$$\mathbf{x} \in \omega_i \quad \text{if} \quad d_i(\mathbf{x}) > d_j(\mathbf{x}) \quad \text{for all } j \neq i.$$

其中 $\mathbf{x}$ 是圖樣，d_j 是類別 ω_j 的純量分數。類別 i 與 j 之間的**決策邊界**，就是 $d_i(\mathbf{x}) - d_j(\mathbf{x}) = 0$ 的集合。本章從最簡單到深度網路的每一種分類器，都只是建立這些函數的不同方式。

圖樣與圖樣類別

白話版。描述物體的方式有兩種：一串數字（向量），或描述各部件怎麼連在一起（字串或樹）。

圖樣向量

最常見的表示法是 n 個量測值組成的行向量：

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

其中每個 x_k 是一個描述子，例如區域面積、色相、某個矩不變量，或某個紋理濾波器的響應。一個向量就是 n 維**特徵空間**中的一個點。好的特徵會讓同類別的圖樣彼此靠近、不同類別的圖樣彼此遠離；特徵若不好，再厲害的分類器也救不回來。

這裡有兩個實務重點。第一，單位不同的特徵需要**正規化**，否則數值範圍最大的那個特徵會主宰所有距離。以水果為例，色相落在 $[0, 1]$，直徑卻以公分計，所以比較距離前要先把每個特徵標準化成平均 0、變異數 1。第二，整張影像本身也可以是圖樣向量：把 $M \times N$ 影像攤平成 MN 維

向量。類神經網路就是這樣「看」影像的，這也是它們需要大量資料的原因，因為這個空間非常巨大。

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

僅線上版：結構圖樣：字串與樹 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/pattern-classification/notes/結構圖樣字串與樹/*

以原型比對進行圖樣分類

白話版。每個類別留一個典型範例。要分類新東西時，看它最像哪一個典型範例。

最小距離分類器

最簡單的原型是每個類別的**平均向量**，由該類別的訓練圖樣算出：

$$\mathbf{m}_j = \frac{1}{n_j} \sum_{\mathbf{x} \in \omega_j} \mathbf{x}, \qquad j = 1, 2, \ldots, W,$$

其中 n_j 是類別 ω_j 的訓練圖樣數。新圖樣被指派到歐氏距離 $D_j(\mathbf{x}) = \|\mathbf{x} - \mathbf{m}_j\|$ 最小的類別。

對 j 最小化 $D_j^2 = \mathbf{x}^\top \mathbf{x} - 2\mathbf{x}^\top \mathbf{m}_j + \mathbf{m}_j^\top \mathbf{m}_j$，等同於最大化

$$d_j(\mathbf{x}) = \mathbf{x}^\top \mathbf{m}_j - \tfrac{1}{2}\, \mathbf{m}_j^\top \mathbf{m}_j,$$

因為 $\mathbf{x}^\top \mathbf{x}$ 與 j 無關。這個 d_j 對 $\mathbf{x}$ 是線性的。類別 i 與 j 之間的邊界為

$$d_i(\mathbf{x}) - d_j(\mathbf{x}) = \mathbf{x}^\top (\mathbf{m}_i - \mathbf{m}_j) - \tfrac{1}{2} (\mathbf{m}_i - \mathbf{m}_j)^\top (\mathbf{m}_i + \mathbf{m}_j) = 0,$$

也就是連接 $\mathbf{m}_i$ 與 $\mathbf{m}_j$ 線段的**垂直平分線**：在 2 維是一條直線，3 維是一個平面，一般情況下是超平面。圖 13.1 是它在水果資料上的結果。

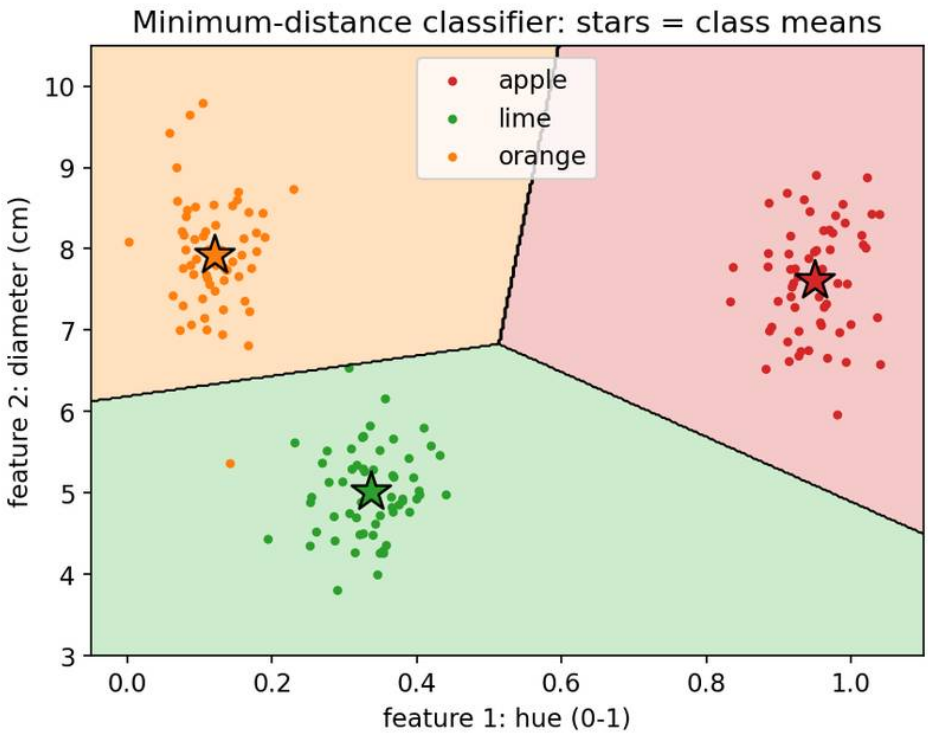


圖 *13.1* — 以最小距離分類合成的水果資料（色相、直徑）。星號是各類別平均；邊界是兩兩平均之間的垂直平分線（在標準化後的特徵上計算）。

```
import numpy as np

def fit_min_distance(X, y):
    """X: (n, d) pattern vectors, y: (n,) labels -> class list and mean vectors."""
    classes = np.unique(y)
    M = np.stack([X[y == c].mean(axis=0) for c in classes])
    return classes, M

def predict_min_distance(X, classes, M):
    d2 = ((X[:, None, :] - M[None, :, :]) ** 2).sum(axis=-1)  # (n, W) squared distances
    return classes[d2.argmin(axis=1)]

rng = np.random.default_rng(0)
X = np.r_[rng.normal([0, 0], 1, (100, 2)), rng.normal([4, 3], 1, (100, 2))]
y = np.repeat([0, 1], 100)
classes, M = fit_min_distance(X, y)
print("training accuracy:", (predict_min_distance(X, classes, M) == y).mean())
```

當各類別是緊密、大致圓形的群集，而且平均之間的距離遠大於群集本身的散布時，這個分類器表現很好。若某個類別是細長的、彎曲的，或由好幾群組成，它就會失敗，因為一個平均代表不了整個類別。兩個簡單的延伸可以改善：每個類別保留**多個原型**，或乾脆保留**所有**訓練圖樣，取最近的那一個（最近鄰規則）。

以相關進行比對（樣板比對）

當原型是一張小影像而不是向量時，就把它拿去跟大影像的每個位置比較，這是原型比對的 2 維版本。原始相關 $\sum wf$ 會偏好亮的區域，因此改用**正規化相關係數**（normalized correlation coefficient）：

$$\gamma(x,y) = \frac{\sum_{s,t}\left[w(s,t)-\bar{w}\right]\left[f(x+s,y+t)-\bar{f}_{xy}\right]}{\sqrt{\sum_{s,t}\left[w(s,t)-\bar{w}\right]^2 \sum_{s,t}\left[f(x+s,y+t)-\bar{f}_{xy}\right]^2}},$$

其中 w 是樣板，$\bar{w}$ 是它的平均，f 是影像，$\bar{f}_{xy}$ 是樣板左上角位於 (x,y) 時其下方影像區塊的平均；總和範圍是樣板的支撐區。因為先減去平均再除以標準化項，γ 落在 $[-1,1]$，而且區塊亮度被縮放或平移時不會改變。$\gamma = 1$ 表示在亮度與對比之外完全吻合。注意這是相關而非卷積：樣板不翻轉（見 ML 03・卷積、影像濾波與 CNN）。

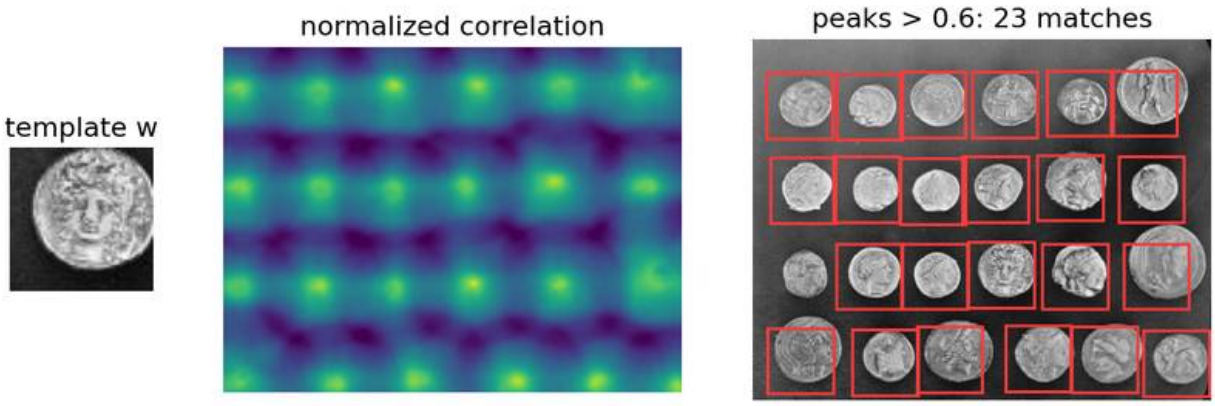


圖 *13.2* — 在 *skimage* 的 *coins* 影像上做樣板比對。左：當作樣板的一枚硬幣。中：相關圖，影像越像樣板的位置越亮。右：大於 *0.6* 的區域極大值。

```
import cv2
import numpy as np
from skimage import data

img = data.coins().astype(np.float32)
w = img[170:225, 180:235]                              # one coin as the template
R = cv2.matchTemplate(img, w, cv2.TM_CCOEFF_NORMED)  # values in [-1, 1]
y0, x0 = np.unravel_index(R.argmax(), R.shape)
print("best match at", (int(x0), int(y0)), "score", round(float(R.max()), 3))
```

圖 13.2 也顯示了這個方法的極限。樣板幾乎找到了每一枚硬幣，因為每枚硬幣都是暗背景上大小相近的亮圓盤；在這個解析度下，它分不出不同的硬幣圖案。相關對旋轉與縮放也很敏感：樣板轉 30° 或放大 30%，峰值可能就不明顯了。常見的補救方式是用一組旋轉、縮放過的樣板去搜尋，或改為比對不變特徵而非原始像素。

比對 SIFT 特徵

IP 12 的尺度不變特徵轉換（scale-invariant feature transform，SIFT）[3] 正好提供這類不變特徵。每個關鍵點都有位置、尺度、方向和一個 128 維描述子。要判斷某個原型物體是否出現在場景影像中：

1. 分別從原型與場景擷取關鍵點與描述子。
2. 對原型的每個描述子，以歐氏距離找出場景中最近與第二近的描述子。
3. 只有當最近距離明顯小於第二近距離時才接受，例如 $d_1/d_2 < 0.8$。這個**比值檢定**（ratio test）[3] 能排除重複紋理中的模稜兩可配對。
4. 檢查被接受的配對是否符合同一個幾何轉換（相似、仿射或投影轉換）；不符合的就是離群值。

若通過幾何一致性檢查的配對夠多，就判定「物體存在」。這仍然是原型比對：原型是一組描述子，而「距離」是一致配對的數量。它能容忍旋轉、尺度變化、部分遮蔽和中等程度的視角改變，這些都是樣板比對做不到的。

僅線上版：比對結構原型 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/pattern-classification/notes/比對結構原型/*

最佳（貝氏）統計分類器

白話版。如果我們知道每個類別出現得有多頻繁、它的量測值通常長什麼樣子，就存在一條最好的規則：選出「在已量到的值之下，機率最高」的類別。平均而言，沒有任何規則犯的錯會比它更少。

推導

白話說：替每一種可能的錯誤訂一個代價，然後對每個圖樣選出期望代價最小的類別。

令 $p(\omega_k \mid \mathbf{x})$ 為圖樣 $\mathbf{x}$ 來自類別 ω_k 的機率，L_{kj} 為把真正屬於 ω_k 的圖樣判為 ω_j 時的**損失**。選擇類別 ω_j 的期望損失稱為**條件風險**（conditional risk）：

$$r_j(\mathbf{x}) = \sum_{k=1}^{W} L_{kj}\, p(\omega_k \mid \mathbf{x}).$$

利用貝氏定理 $p(\omega_k \mid \mathbf{x}) = p(\mathbf{x} \mid \omega_k)\, P(\omega_k)/p(\mathbf{x})$，可以改用**類別條件密度** $p(\mathbf{x} \mid \omega_k)$（類別 ω_k 的圖樣長什麼樣子）與**事前機率** $P(\omega_k)$（這個類別有多常見）來表示。$1/p(\mathbf{x})$ 對每個 j 都一樣，比較風險時可以省略。

對每個 $\mathbf{x}$ 都選出 $r_j(\mathbf{x})$ 最小之 j 的分類器，會使整體平均損失最小，這就是它被稱為**最佳**的原因：在已知真實分布的前提下，沒有任何規則能做得更好。在常見的 **0–1 損失**下（$k = j$ 時 $L_{kj} = 0$，否則為 1），每種錯誤代價相同，

$$r_j(\mathbf{x}) \propto p(\mathbf{x}) - p(\mathbf{x} \mid \omega_j) P(\omega_j).$$

最小化它等於最大化決策函數

$$d_j(\mathbf{x}) = p(\mathbf{x} \mid \omega_j)\, P(\omega_j).$$

換句話說：選出事後機率最大的類別。問題在於我們從來不知道真實的密度，只能從訓練資料估計，而分類器的好壞取決於這些估計的好壞。

高斯圖樣類別的貝氏分類器

白話說：如果每個類別的量測值形成一團鐘形的雲，最佳規則就會變成一條由各團雲的中心與散布寫成的明確公式。

假設每個類別都是平均為 $\mathbf{m}_j$、共變異數矩陣為 $\mathbf{C}_j$ 的多變量高斯分布：

$$p(\mathbf{x} \mid \omega_j) = \frac{1}{(2\pi)^{n/2} |\mathbf{C}_j|^{1/2}} \exp\!\Big[-\tfrac{1}{2} (\mathbf{x}-\mathbf{m}_j)^{\top} \mathbf{C}_j^{-1} (\mathbf{x}-\mathbf{m}_j) \Big],$$

其中 n 是 $\mathbf{x}$ 的維度，$|\mathbf{C}_j|$ 是 $\mathbf{C}_j$ 的行列式。因為對數是遞增函數，可以改為最大化 $\ln d_j$，並丟掉常數 $\frac{n}{2}\ln 2\pi$：

$$d_j(\mathbf{x}) = \ln P(\omega_j) - \tfrac{1}{2}\ln|\mathbf{C}_j| - \tfrac{1}{2}(\mathbf{x}-\mathbf{m}_j)^{\top}\mathbf{C}_j^{-1}(\mathbf{x}-\mathbf{m}_j).$$

最後一項是**馬氏距離**（Mahalanobis distance）平方的一半；這種距離會把空間拉伸，使每個類別的散布看起來是圓的。這個 d_j 對 $\mathbf{x}$ 是二次的，所以類別之間的邊界是二次曲面：在 2 維是橢圓、拋物線或雙曲線。兩個特例把我們帶回前面的小節：

- **共變異數相同**（所有 j 皆 $\mathbf{C}_j = \mathbf{C}$）。二次項 $\mathbf{x}^{\top}\mathbf{C}^{-1}\mathbf{x}$ 對所有類別都相同而抵消，剩下線性函數 $d_j(\mathbf{x}) = \ln P(\omega_j) + \mathbf{x}^{\top}\mathbf{C}^{-1}\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^{\top}\mathbf{C}^{-1}\mathbf{m}_j$。邊界變成超平面。
- **單位共變異數且事前機率相等**（$\mathbf{C} = \mathbf{I}$，$P(\omega_j) = 1/W$）。函數化簡為 $\mathbf{x}^{\top}\mathbf{m}_j - \frac{1}{2}\mathbf{m}_j^{\top}\mathbf{m}_j$，正是最小距離分類器。所以最小距離分類器只在「類別是圓形、散布相同、出現機率相等的高斯分布」時才是貝氏最佳。

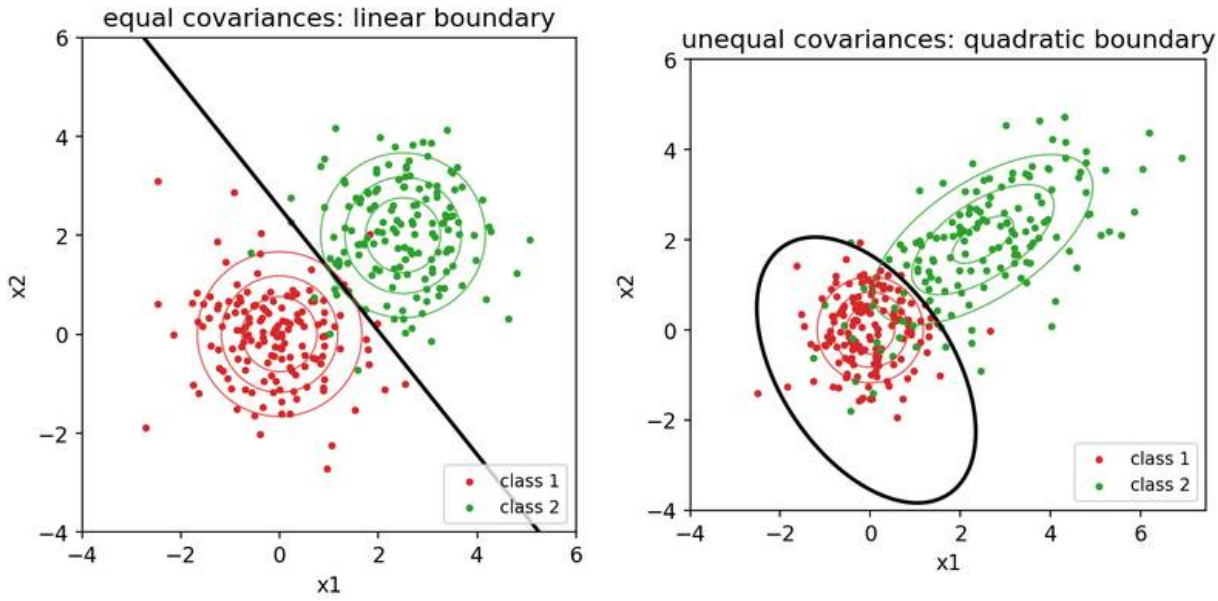


圖 *13.3* — 兩個高斯類別的貝氏決策邊界（粗黑線）。細等高線是各類別的密度。左：共變異數相同，邊界是直線。右：共變異數不同，邊界是包住較集中類別的二次曲線。

從樣本估計 $\mathbf{m}_j$ 與 $\mathbf{C}_j$ 很容易：樣本平均與樣本共變異數。若訓練集中各類別的比例反映真實情況，事前機率就可以用這些比例。

```
import numpy as np

class GaussianBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.m = [X[y == c].mean(0) for c in self.classes]
        self.C = [np.cov(X[y == c].T) for c in self.classes]
        self.logP = [np.log((y == c).mean()) for c in self.classes]
        return self

    def decision(self, X):
        scores = []
        for m, C, lp in zip(self.m, self.C, self.logP):
            d = X - m
            maha = np.einsum("ni,ij,nj->n", d, np.linalg.inv(C), d)
            scores.append(lp - 0.5 * np.linalg.slogdet(C)[1] - 0.5 * maha)
        return np.stack(scores, axis=1)            # (n, W): one d_j(x) per class

    def predict(self, X):
        return self.classes[self.decision(X).argmax(1)]
```

每個類別的共變異數參數數量是 $n(n+1)/2$。4 個特徵時每類 10 個數字，很容易估計；1,000 個特徵時約有 50 萬個，除非資料極多，否則估計會很不可靠。這是高斯貝氏分類器最適合用在「短而精挑細選」的特徵向量上的原因之一。

應用：多光譜影像的像素分類

遙測衛星會在多個光譜波段記錄地面上的每個位置。於是每個像素自然就是一個圖樣向量 $\mathbf{x} = (x_{\text{blue}}, x_{\text{green}}, x_{\text{red}}, x_{\text{NIR}})^\top$，NIR 代表近紅外光。健康的植被在近紅外光反射強、在紅光反射弱；水體在近紅外光很暗；建成區在各波段都中等偏亮。

我們合成了一張 128×128 的場景，含三個類別（水體、植被、建成區），替每個類別設定四個波段的平均反射率，並在每個波段加上標準差 0.06 的獨立高斯雜訊。每類只隨機挑 50 個像素當訓練資料，估計 $\mathbf{m}_j$ 與 $\mathbf{C}_j$，再用高斯貝氏規則分類所有像素。

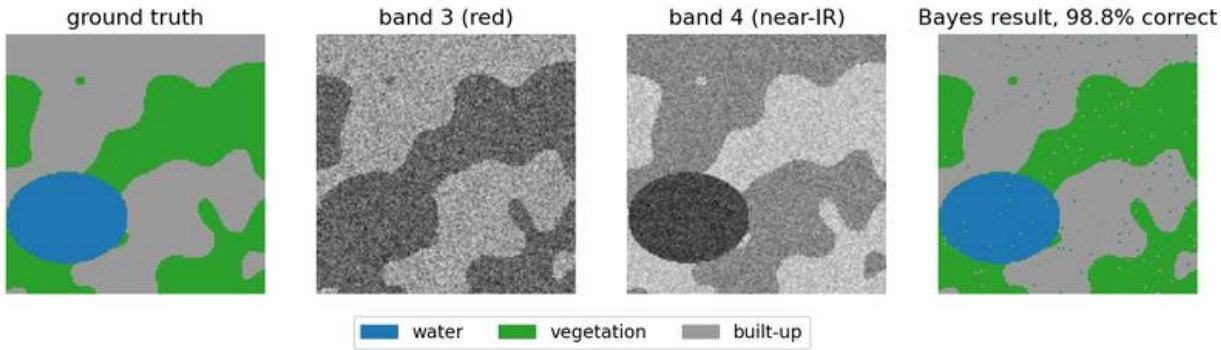


圖 *13.4* — 以貝氏規則分類合成的四波段場景。單一波段雜訊多又模稜兩可（紅光幾乎分不開水體與植被），但四維圖樣向量能把類別分得很好：只用 *150* 個訓練像素，就有 *98.8%* 的像素被正確分類。

這裡有兩個教訓。第一，單一波段不夠：在紅光波段中，水體與植被的值很接近，是波段的**組合**把它們分開的。第二，剩下的錯誤都是零散的單一像素。這裡的貝氏規則把每個像素獨立處理，忽略了鄰居。加入空間脈絡，例如平滑決策分數，或使用能看到一整個鄰域的分類器，就能去掉大部分這類錯誤。這個觀察自然地把我們帶到卷積網路。

類神經網路與深度學習

白話版。類神經網路是一疊非常簡單的單元。每個單元把輸入乘上權重加總，再決定要「發射」多強。每犯一次錯就微調一下權重，網路就能自己學會決策規則，不需要我們寫出機率分布。

背景

貝氏分類器需要密度估計，而在高維空間中這很難。另一條路是**直接從訓練資料學出決策函數**。類神經網路用許多簡單、可調整、分層排列的運算單元做到這一點。這個想法可追溯到 1950 年代末的感知器 [4]；1986 年反向傳播廣為人知後，多層網路才變得實用 [5]；2012 年深度卷積網路贏得 ImageNet 競賽後，它在影像領域成為主流 [7]。關於感知器、MLP、激活函數與訓練技巧的精簡整理，可參考 ML 02・類神經網路。

感知器

感知器（perceptron）計算輸入的加權和加上偏差，然後輸出正負號：

$$d(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0 = \sum_{k=1}^{n} w_k x_k + w_0, \qquad \text{class} = \begin{cases} \omega_1 & d(\mathbf{x}) > 0 \\ \omega_2 & d(\mathbf{x}) \le 0 \end{cases}$$

其中 $\mathbf{w} = (w_1, \dots, w_n)^\top$ 是**權重**，w_0 是**偏差**（bias）。邊界 $d(\mathbf{x}) = 0$ 是一個超平面。為了簡化符號，在每個圖樣後面補一個常數 1，$\mathbf{y} = (x_1, \dots, x_n, 1)^\top$，並把偏差併入權重 $\mathbf{w} = (w_1, \dots, w_n, w_0)^\top$，於是 $d = \mathbf{w}^\top \mathbf{y}$。

學習規則。把標籤編碼成：ω_1 為 $t = +1$、ω_2 為 $t = -1$。一次呈現一個訓練圖樣，在第 k 步：

$$\mathbf{w}(k+1) = \begin{cases} \mathbf{w}(k) + \alpha\, t\, \mathbf{y}(k) & \text{if } t\, \mathbf{w}(k)^\top \mathbf{y}(k) \le 0 \text{ (misclassified)} \\ \mathbf{w}(k) & \text{otherwise,} \end{cases}$$

其中 $\alpha > 0$ 是**學習率**。每次修正都會移動超平面，把出錯的圖樣推向正確的一側：更新後 $t\, \mathbf{w}^\top \mathbf{y}$ 會增加 $\alpha \|\mathbf{y}\|^2$。若兩個類別**線性可分**，這個程序保證在有限次修正後停止，並得到一個分隔超平面；若不可分，它永遠不會穩定下來。這時改用梯度下降去最小化一個平滑的誤差，例如 $\mathbf{w}^\top \mathbf{y}$ 與 t 的差的平方（最小均方法，又稱 delta 規則）。這個想法直接通往反向傳播。

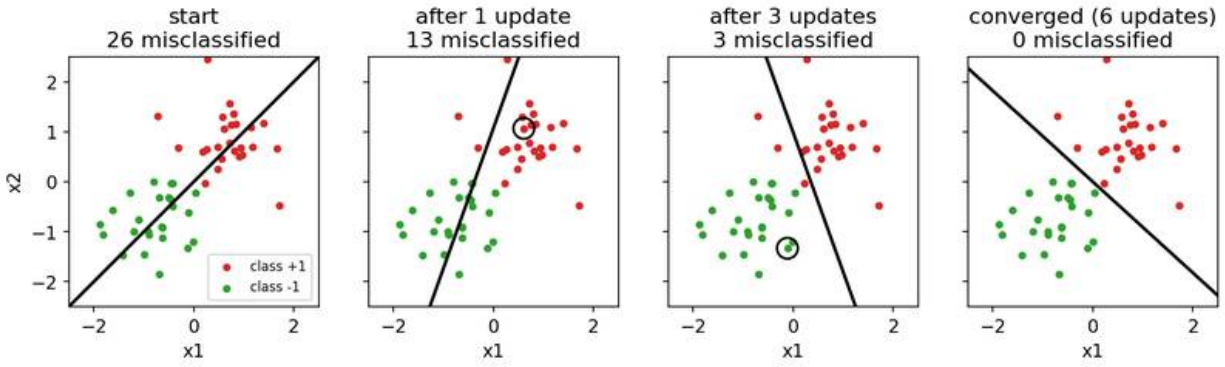


圖 *13.5* — 在兩個可分類別上訓練感知器。圈起來的點是觸發該次更新的錯誤分類圖樣。修正六次後，直線已能分開兩類，學習停止。

```
import numpy as np

def train_perceptron(X, y, alpha=0.5, epochs=100):
    """X: (n, d), y in {-1, +1}. Returns augmented weights (w_1..w_d, w_0)."""
    Xa = np.c_[X, np.ones(len(X))]                # append the constant input 1
    w = np.zeros(Xa.shape[1])
    for _ in range(epochs):
        mistakes = 0
        for xi, yi in zip(Xa, y):
            if yi * (w @ xi) <= 0:                # wrong side (or on the line)
                w += alpha * yi * xi
                mistakes += 1
        if mistakes == 0:                         # a full clean pass: converged
```

```
            break
    return w
```

單一感知器只能畫出一個超平面。它解不了 XOR：兩個類別位於正方形的對角。要畫出彎曲或分成多段的邊界，就需要更多層。

多層前饋網路

多層前饋網路（multilayer feedforward network，又稱多層感知器 MLP）把多層單元疊起來。第 ℓ 層接收第 $\ell-1$ 層的輸出，預測時沒有任何訊號往回流。與感知器不同的是，每個單元會把加權和送進一個平滑、可微分的**激活函數** h，例如 sigmoid $h(z)=1/(1+e^{-z})$、雙曲正切，或線性整流單元 $\mathrm{ReLU}(z)=\max(0,z)$。平滑性讓以梯度為基礎的訓練成為可能；非線性則讓深度有意義，因為一疊純線性的層會塌縮成單一線性層。

只要有兩個隱藏層和足夠的單元，網路幾乎能形成任何形狀的決策區域。實務上的問題不是網路**能不能**表示某個邊界，而是訓練能不能從手邊的資料中**找到**它。

前向傳遞

白話說：每一層把輸入乘上權重矩陣、加上偏差，再套用激活函數；最後一層把分數轉成各類別的機率。

把層編號為 $\ell=1,\dots,L$，第 0 層是輸入 $\mathbf{a}^{(0)}=\mathbf{x}$。每一層：

$$\mathbf{z}^{(\ell)}=\mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)}+\mathbf{b}^{(\ell)},\qquad \mathbf{a}^{(\ell)}=h(\mathbf{z}^{(\ell)}),$$

其中 $\mathbf{W}^{(\ell)}$ 是權重矩陣（第 i 列是進入單元 i 的權重），$\mathbf{b}^{(\ell)}$ 是偏差向量，$\mathbf{z}^{(\ell)}$ 是**淨輸入**，$\mathbf{a}^{(\ell)}$ 是**激活值**，h 逐元素作用。對 W 類問題，輸出層有 W 個單元，其淨輸入以 **softmax** 函數轉成機率：

$$p_j=\frac{e^{z_j^{(L)}}}{\sum_{i=1}^{W} e^{z_i^{(L)}}},\qquad j=1,\dots,W,$$

預測類別就是 p_j 最大的那一個。注意輸出層就是一組決策函數 $d_j(\mathbf{x})=z_j^{(L)}$，和「背景」一節完全相同。隱藏層算出的特徵，讓這些簡單的線性決策函數就足夠了。

反向傳播

訓練就是調整所有權重，降低衡量輸出錯得多離譜的**損失** E。分類問題的標準選擇是**交叉熵** $E=-\sum_j t_j \ln p_j$，其中 $\mathbf{t}$ 是 one-hot 標籤（正確類別為 1，其餘為 0）。對 sigmoid 輸出，經典的替代選擇是平方誤差 $E=\frac{1}{2}\|\mathbf{a}^{(L)}-\mathbf{t}\|^2$。我們需要每一層的 $\partial E/\partial \mathbf{W}^{(\ell)}$，反向傳播用一次由後往前的連鎖律就把它們全部算出來 [5]。

白話說：反向傳播把輸出端的誤差一層一層往回送，而每個權重的梯度就是「到達它輸出端的誤差」乘以「它輸入端的激活值」。定義第 ℓ 層的**誤差訊號**為 $\boldsymbol{\delta}^{(\ell)}=\partial E/\partial \mathbf{z}^{(\ell)}$。softmax 搭配交叉熵

時，輸出層的訊號就是 $\delta^{(L)} = \mathbf{p} - \mathbf{t}$。前面每一層的訊號，是把下一層的訊號經由轉置的權重傳回來，再乘上激活函數的導數；$\mathbf{W}^{(\ell)}$ 的梯度是 $\delta^{(\ell)}\mathbf{a}^{(\ell-1)\top}$，最後以學習率 α 做梯度下降更新每個權重。反向傳遞的計算量和前向傳遞差不多，這正是擁有數百萬個權重的網路能被訓練的原因。感知器學習規則 [4] 是另一種依分類錯誤更新的規則，不是對硬階梯函數直接做一般反向傳播：階梯函數的導數在門檻之外為零，在門檻處沒有定義。可微分的激活函數，才讓 [5] 的連鎖律提供有用的梯度。

僅線上版：逐步說明反向傳播 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/pattern-classification/notes/逐步說明反向傳播/*

訓練

實務上，我們在一小批隨機挑選的訓練圖樣（**mini-batch**）上平均梯度、更新，然後重複，這就是**隨機梯度下降**（SGD）。走過整個訓練集一次稱為一個 **epoch**。訓練會持續許多個 epoch，同時觀察驗證集上的損失。學習率是最重要的設定：太大，損失會震盪甚至發散；太小，訓練慢如蝸牛。權重必須以小的隨機數初始化而不是全零，否則同一層的所有單元會算出同樣的東西、收到同樣的更新，永遠分不開。

下面的 NumPy 程式訓練一個兩層網路解 XOR，也就是單一感知器解不了的問題：

```
import numpy as np

rng = np.random.default_rng(0)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], float)
y = np.array([[0], [1], [1], [0]], float)         # XOR: not linearly separable

W1, b1 = rng.normal(0, 1, (2, 8)), np.zeros(8)    # input -> 8 hidden units
W2, b2 = rng.normal(0, 1, (8, 1)), np.zeros(1)    # hidden -> 1 output
sig = lambda z: 1 / (1 + np.exp(-z))
alpha = 1.0

for step in range(5000):
    # forward pass
    a1 = np.tanh(X @ W1 + b1)
    out = sig(a1 @ W2 + b2)
    # backward pass (cross-entropy loss with a sigmoid output)
    d2 = (out - y) / len(X)                        # delta at the output layer
    d1 = (d2 @ W2.T) * (1 - a1 ** 2)               # delta at the hidden layer
    W2 -= alpha * a1.T @ d2;  b2 -= alpha * d2.sum(0)
    W1 -= alpha * X.T @ d1;   b1 -= alpha * d1.sum(0)

print(out.round(3).ravel())                        # close to [0, 1, 1, 0]
```

（這裡每一列是一個圖樣，所以矩陣乘法相對於上面的式子是轉置的。）

深度卷積神經網路

白話版。我們不告訴網路該量哪些特徵，而是讓它把可學習的小濾波器在影像上滑動，就像 IP 03 的濾波器一樣，並自己學會哪些濾波器有助於分類。結果前幾層會找出邊緣和斑點，後面幾層再把它們組合成部件與整個物體。

為什麼全連接網路處理影像很吃力

把一張 20×20 影像送進一個有 2,048 個隱藏單元的全連接層，這一層就有 $400 \times 2048 = 819{,}200$ 個權重。而且這一層忽略了「相鄰像素彼此相關」以及「往右移兩個像素的 3 還是 3」這兩件事。CNN 把這兩點直接建進結構裡 [6]。

卷積層

卷積層對輸入套用 F 個小核（例如 5×5）。若有 C 個輸入通道 a_c，輸出通道 f 在像素 (x, y) 的淨輸入為

$$z_f(x,y) = \sum_{c=1}^{C} \sum_{s,t} w_{f,c}(s,t)\, a_c(x+s, y+t) + b_f,$$

再接一個激活函數，通常是 ReLU：$a_f(x,y) = \max(0, z_f(x,y))$。這個總和和 IP 03 一樣是空間相關，但核的值 $w_{f,c}$ 是**學出來的**。每個輸出通道是一張**特徵圖**（feature map）：在核所對應的圖樣出現的地方，值就大。有兩個想法大幅減少參數量：

- **局部連接**。每個輸出只取決於一個小鄰域（它的**感受野**，receptive field）。
- **權重共享**。同一個核用在每一個位置，所以在某處學到的特徵，在任何地方都能偵測到。

在單通道影像上用 8 個 5×5 核，只需要 $8 \times 25 + 8 = 208$ 個參數，相比之下前面是 819,200 個。

池化層

池化層（pooling layer）縮小每張特徵圖，通常是取每個不重疊 2×2 區塊的最大值（max-pooling）或平均值。它讓寬和高減半，使響應能容忍小幅位移，也擴大了後續層的感受野。卷積與池化堆疊數次後，得到的特徵會對更大、更抽象的結構產生響應。

全連接層與整條流程

最後一個池化層之後，特徵圖被**攤平**成一個向量，經過一或多個全連接層，最後接 softmax。圖 13.6 是我們訓練的小網路。

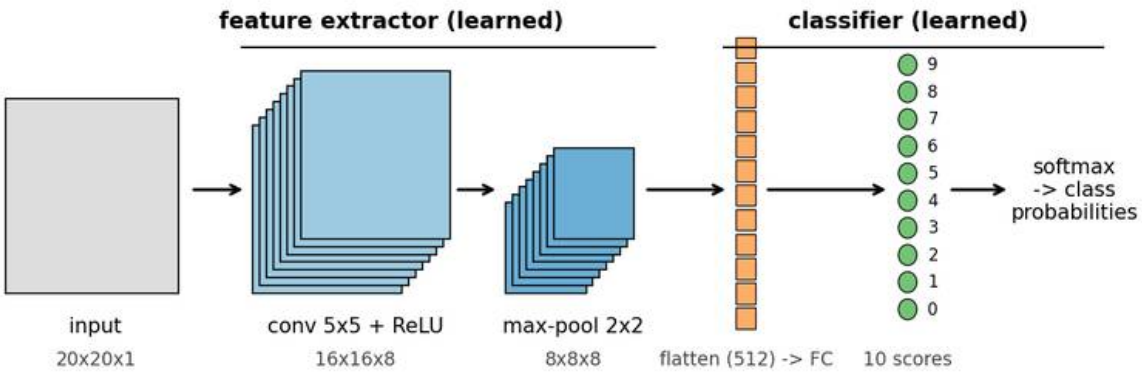


圖 *13.6* — 本章使用的小型 *CNN*。卷積與池化層構成學出來的特徵擷取器；全連接層是作用在這些特徵上的、學出來的線性分類器。

這是本章最關鍵的觀念。傳統流程是**人工設計的特徵 → 分類器**；CNN 則是**學出來的特徵 → 分類器**，兩部分一起用反向傳播訓練，最小化同一個損失。最後的全連接 softmax 層只是一組線性決策函數，和最小距離分類器、等共變異數貝氏分類器中遇到的是同一種東西。改變的是：這些線性函數所作用的空間本身是學出來的。

訓練 CNN

反向傳播完全照用，只是局部導數不同。共享的核權重會從所有用到它的位置收集梯度；最大池化只把梯度傳給原本是最大值的那個輸入；ReLU 只在輸入為正的地方讓梯度通過。

僅線上版：卷積、池化與 *ReLU* 的梯度 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/pattern-classification/notes/卷積池化與-relu-的梯度/*

以下用 SciPy 寫出一個「卷積、ReLU、池化」階段的前向傳遞：

```
import numpy as np
from scipy.signal import correlate2d
from skimage import data, transform

img = transform.resize(data.camera(), (64, 64), anti_aliasing=True)
k = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], float)    # one 3x3 "filter"
fmap = np.maximum(correlate2d(img, k, mode="valid") + 0.0, 0)  # conv + bias + ReLU
pooled = fmap[:62, :62].reshape(31, 2, 31, 2).max(axis=(1, 3))  # 2x2 max-pooling
print(img.shape, "->", fmap.shape, "->", pooled.shape)
```

實作範例：數字辨識

CNN 的經典試驗場是手寫數字辨識，LeNet 類型的網路就是在 MNIST 數字資料上發展出來的 [6]。為了讓一切自給自足、可重現，我們自行產生**類 MNIST** 的 20×20 數字：每個 0–9 的數字以六種 OpenCV 字型之一繪出，大小、筆畫粗細、位移與旋轉（$\pm 15°$）都是隨機的，最後加上雜訊（圖 13.7）。

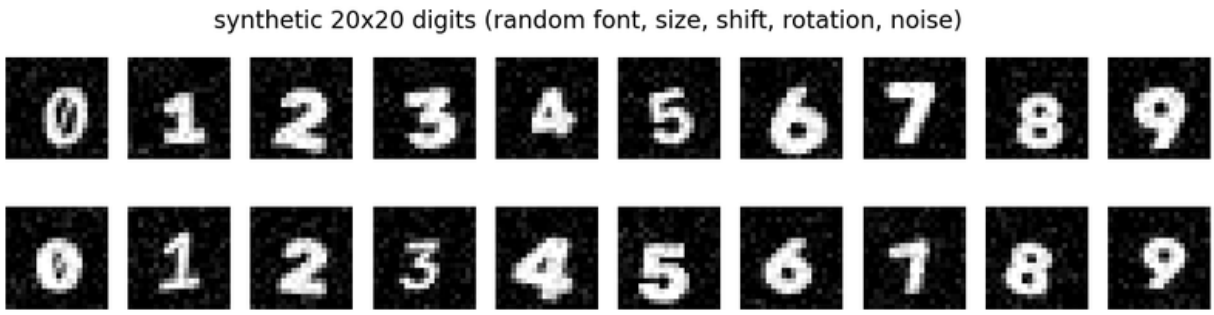


圖 *13.7* — 合成的訓練數字。字型、大小、位置與角度的變化，迫使分類器學習形狀，而不是背下像素。

圖 13.6 的網路（八個 5×5 濾波器、ReLU、2×2 最大池化、接到 10 個輸出的全連接層、softmax 搭配交叉熵）在 `scripts/figures/dip_ch13.py` 中用大約 80 行 NumPy 寫成。我們以 mini-batch SGD（batch 大小 32、學習率 0.05、少量權重衰減）在 3,000 張數字上訓練 12 個 epoch，在另外 1,000 張驗證數字上達到 **98.5%** 準確率。圖 13.8 呈現它學到了什麼。

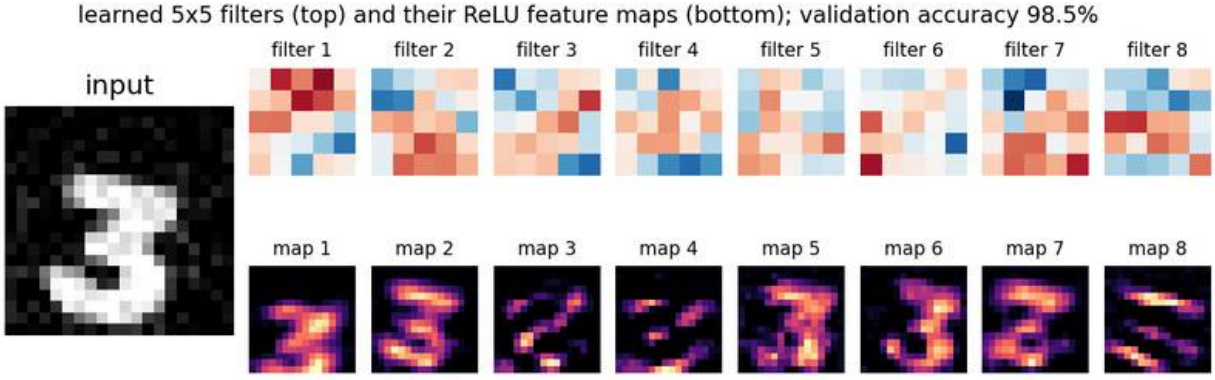


圖 *13.8* — 訓練後 *CNN* 的內部。上：八個學出來的 *5×5* 核（紅為正、藍為負）。下：它們對輸入數字的 *ReLU* 特徵圖。有幾個核像是有方向性的邊緣或筆畫偵測器，和 *IP 10* 人工設計的梯度濾波器很相似。

沒有人叫網路去偵測邊緣。有方向性、像邊緣的核之所以出現，是因為它們對分辨數字有用。在自然影像上訓練的大型網路，第一層也有相同的現象，越深的層則出現越複雜的圖樣。

一些實作上的細節

白話版。把作業背得滾瓜爛熟的網路，考試還是可能考砸。訓練的實務技巧，大多是在讓它學會一般規則，而不是記住特定例子。

過擬合

參數多、訓練例子少的網路，可以靠死背把訓練誤差壓到零，但對新資料的誤差仍然很高。訓練表現與驗證表現之間的這個落差就是**過擬合**（overfitting）。圖 13.9（左）清楚呈現了這一點：只用 150 張數字訓練時，CNN 的訓練準確率達到 100%，驗證準確率卻只有 86.2%。

基本的防線：

- **更多資料**。沒有什麼比它更可靠。資料擴增（見下）是取得更多資料的便宜方法。
- **驗證集與提早停止（early stopping）**。追蹤驗證準確率，保留最佳 epoch 的權重。
- **更小或結構更好的模型**。與全連接層相比，CNN 的權重共享本身就是很強的正則化。

資料擴增

資料擴增（data augmentation）以保留標籤的轉換產生新的訓練例子：小幅位移、旋轉、縮放、翻轉（只在合理時使用：翻轉的貓還是貓，但翻轉的 b 會變成 d）、裁切、亮度與對比變化，以及加入雜訊 [18]。每個 epoch 看到的每張影像都略有不同，因此更難死背，而對這些轉換的不變性則從資料中學到。



圖 *13.9* — 同一個 *CNN* 只用 *150* 張合成數字訓練 *60* 個 *epoch*。左：沒有擴增時，訓練與驗證之間出現很大的落差（過擬合）。右：隨機位移、旋轉與縮放讓兩條曲線保持接近，最終驗證準確率從 *86.2%* 提升到 *91.3%*。

正則化

正則化（regularization）加入對較簡單解的偏好：

- **權重衰減**（weight decay，L_2 正則化）在損失中加上 $\frac{\lambda}{2}\sum w^2$，等於在每個權重的梯度上加 λw，把權重往零收縮。我們的訓練腳本使用 $\lambda = 10^{-4}$。

- **Dropout** 在每個訓練步驟隨機把一部分單元設為零，讓任何單元都無法依賴特定的夥伴。測試時使用所有單元，並適當縮放權重。效果類似於把許多「變瘦」的網路做集成平均 [17]。

其他實務細節同樣重要：輸入正規化（平均 0、變異數 1）、以「依輸入數量縮放的變異數」初始化權重（讓激活值在層與層之間既不爆炸也不消失）、訓練過程中逐步降低學習率，以及永遠只在開發期間沒被用來做任何決定的測試集上報告最終結果。

現代觀點

教科書這一章結束的地方，正是現代影像分類開始的地方。以下六篇綜述與回顧論文，勾勒出這個領域此後的走向。

深度學習之前的統計圖樣辨識。 Jain、Duin 與 Mao 的回顧 [2] 描繪了傳統領域的全貌，從表示與特徵選擇，到分類器、分類器結合與錯誤率估計。它的主要結論是：特徵表示與評估流程比分類器的選擇更重要。這個結論經得起時間考驗：深度學習把「表示」這一部分自動化了。

CNN 時代。 Rawat 與 Wang [11] 回顧了到 2017 年為止用於影像分類的 CNN。這條脈絡從 LeCun 等人以梯度訓練、用於文件辨識的網路 [6] 開始，經過 2012 年贏得 ImageNet 競賽 [10] 的 AlexNet [7]，再到 VGG [8] 與 ResNet [9]；ResNet 的恆等捷徑讓非常深的網路變得可以訓練。這些架構都仍看得出圖 13.6 的影子。

視覺中的 Transformer。 Vision Transformer（ViT）[12] 把影像小塊當成 token，以自注意力（self-attention）取代卷積；Khan 等人 [13] 綜述了它的眾多變體，而 ConvNeXt [14] 顯示現代化的純 CNN 又能再度匹敵。實務上的教訓是：架構家族的差異，不如訓練資料、規模與訓練配方來得重要。

基礎模型特徵與零樣本分類。 CLIP [15] 同時訓練一個影像編碼器與一個文字編碼器，因此不需要標註影像，只要把每張影像與 `a photo of a {class}` 這類提示文字比較，就能建出分類器：這是一個**最小距離分類器**，只是原型來自語言。DINOv2 [16] 以自監督方式學習通用特徵，在它凍結的特徵上接一個線性層或最近鄰規則，就能分類得很好。最昂貴的特徵擷取器只訓練一次；上面的分類器往往正是本章的簡單規則之一。

資料擴增。 Shorten 與 Khoshgoftaar 的綜述 [18] 把擴增方法分成基本影像操作與學習式方法。它的結論和圖 13.9 一致：在標註資料有限時，擴增是降低過擬合最便宜也最有效的方法之一。

評估陷阱與資料集偏差。 Torralba 與 Efros [19] 顯示分類器能分辨一張影像來自哪個資料集，Liu 與 He [20] 則發現，對現代網路與網路規模的資料集，這一點依然成立。在某個基準上的準確率，描述的是那個基準；只要可能，就用與訓練資料獨立蒐集的資料來測試。

僅線上版：現代觀點論文導讀 — *https://blog.shyandram.dev/zh-tw/learn/image-processing/pattern-classification/papers/現代觀點論文導讀/*

改變了什麼，沒改變什麼。深度學習取代了人工特徵，也讓分類器本身幾乎成了附屬品：一個線性 softmax 層。但它沒有改變決策理論的基礎。softmax 輸出被訓練來估計事後機率，所以網路是在逼近貝氏規則；交叉熵就是負對數概似。原型比對以「嵌入空間中的最近鄰搜尋」形式延續，用於零樣本分類與檢索。在物體外觀固定、訓練資料稀少的受控工業檢測中，以正規化相關做樣板比對仍然是對的工具。理解傳統方法，才知道現代方法到底在最佳化什麼，以及什麼時候簡單的方法就夠了。

重點整理

- 分類透過決策函數 $d_j(\mathbf{x})$ 替圖樣指定標籤；兩個決策函數相等之處就是邊界。
- 最小距離分類以類別平均為原型，邊界是垂直平分線。正規化相關與 SIFT 比對則是針對影像與關鍵點的原型比對。
- 貝氏分類器使平均損失最小。對高斯類別它是二次的；共變異數相同時是線性的；單位共變異數且事前機率相等時就化簡為最小距離。
- 感知器靠修正錯誤學出分隔超平面。使用平滑激活函數的多層網路能學出非線性邊界，反向傳播用一次反向傳遞算出所有梯度。
- CNN 是學出來的特徵擷取器（卷積、非線性、池化）加上學出來的線性分類器。權重共享讓它有效率，也能容忍位移。
- 過擬合是最主要的實務風險。使用驗證資料、資料擴增、權重衰減與 dropout，並且不要碰測試集。
- 現代系統常把大型預訓練特徵擷取器（CLIP、DINOv2）與非常簡單的分類器搭配。基準準確率反映了資料集偏差，所以要用獨立蒐集的資料測試。

練習

1. **手算平分線**。兩個類別平均為 $\mathbf{m}_1 = (1, 2)^\top$ 與 $\mathbf{m}_2 = (5, 4)^\top$。把最小距離邊界 $d_1(\mathbf{x}) - d_2(\mathbf{x}) = 0$ 寫成 x_1, x_2 的方程式，並驗證兩個平均的中點落在邊界上。

   提示

   $\mathbf{m}_1 - \mathbf{m}_2 = (-4, -2)^\top$，且 $\frac{1}{2}(\mathbf{m}_1 - \mathbf{m}_2)^\top(\mathbf{m}_1 + \mathbf{m}_2) = \frac{1}{2}(-4 \cdot 6 - 2 \cdot 6) = -18$。邊界為 $-4x_1 - 2x_2 + 18 = 0$，即 $2x_1 + x_2 = 9$。中點 $(3, 3)$ 代入得 $6 + 3 = 9$。✓

2. **事前機率的影響**。在一維問題中，類別 ω_1 是平均 0 的高斯分布，類別 ω_2 是平均 4 的高斯分布，變異數都是 1。求 $P(\omega_1) = P(\omega_2)$ 時與 $P(\omega_1) = 0.9$ 時的貝氏門檻值。門檻往哪個方向移動？為什麼這樣合理？

   提示

   令 $\ln P_1 - \frac{1}{2}x^2 = \ln P_2 - \frac{1}{2}(x - 4)^2$，得 $x = 2 + \frac{1}{4}\ln(P_1/P_2)$。事前機率相等時 $x = 2$；$P_1 = 0.9$ 時 $x = 2 + \frac{1}{4}\ln 9 \approx 2.55$。門檻往較少見的類別移動，於是更多模稜兩可的圖樣被判給常見的類別。

3. **相關的不變性**。證明若把影像區塊的每個像素換成 $af + b$（$a > 0$），正規化相關係數 γ 不變。若 $a < 0$ 會怎樣？接著用 `cv2.matchTemplate` 在 `skimage.data.camera()` 上驗證。

   提示

   減去平均就消去了 b。a 在分子出現一次，在分母則以 $\sqrt{a^2} = |a|$ 出現，所以 $a > 0$ 時抵消。$a < 0$（相片負片）時 γ 變號：完美吻合變成 -1。

4. **停不下來的感知器**。產生兩團互相重疊的高斯點雲，用本章的感知器規則跑 100 個 epoch，畫出每個 epoch 的錯誤數。接著改用梯度下降最小化平方誤差 $\frac{1}{2}\sum(\mathbf{w}^\top \mathbf{y} - t)^2$，比較兩條學習曲線。

   提示

   感知器的錯誤數會一直起伏，因為即使沒有任何直線能完美分開，每一次錯誤仍會移動直線。最小平方規則則會收斂到一條固定的折衷直線，因為它的損失平滑且為凸函數。

5. **計算參數量**。某網路的輸入是 64×64 灰階影像。比較 (a) 有 1,024 個隱藏單元的全連接層與 (b) 有 32 個 3×3 核的卷積層的權重數量（含偏差）。不補邊時，(b) 會產生多少個輸出？

   提示

   (a) $4096 \times 1024 + 1024 = 4{,}195{,}328$。(b) $32 \times 9 + 32 = 320$ 個參數，產生 32 張 62×62 的特徵圖，也就是 123,008 個輸出。參數少、輸出多，這就是權重共享。

6. **有害的擴增**。使用 `scripts/figures/dip_ch13.py` 中的數字產生器，在擴增中加入隨機 180° 旋轉。重新訓練後，檢查哪兩個數字之間的混淆增加最多，並解釋原因。

   提示

   6 轉 180° 看起來像 9（0、1、8 則看起來和自己一樣）。這個擴增不再保留標籤，等於教網路對同一個形狀給出互相矛盾的標籤。可以預期 6 與 9 的混淆會大幅上升。

參考文獻

1. R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018, Ch. 13. publisher page
2. A. K. Jain, R. P. W. Duin, and J. Mao, “Statistical Pattern Recognition: A Review,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000. doi:10.1109/34.824819
3. D. G. Lowe, “Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, 2004. doi:10.1023/B:VISI.0000029664.99615.94
4. F. Rosenblatt, “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain,” *Psychological Review*, vol. 65, 1958. doi:10.1037/h0042519
5. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning Representations by Back-Propagating Errors,” *Nature*, vol. 323, 1986. doi:10.1038/323533a0
6. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proceedings of the IEEE*, vol. 86, 1998. doi:10.1109/5.726791
7. 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

8. K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” arXiv:1409.1556, 2014. arXiv
9. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” arXiv: 1512.03385, 2015. arXiv
10. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, et al., “ImageNet Large Scale Visual Recognition Challenge,” arXiv:1409.0575, 2014. arXiv
11. W. Rawat and Z. Wang, “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review,” *Neural Computation*, vol. 29, no. 9, 2017. doi:10.1162/neco_a_00990
12. A. Dosovitskiy, L. Beyer, A. Kolesnikov, et al., “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” *ICLR*, 2021. arXiv
13. 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
14. Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A ConvNet for the 2020s,” *CVPR*, 2022. arXiv
15. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, et al., “Learning Transferable Visual Models From Natural Language Supervision,” arXiv:2103.00020, 2021. arXiv
16. M. Oquab, T. Darcet, T. Moutakanni, et al., “DINOv2: Learning Robust Visual Features without Supervision,” arXiv:2304.07193, 2023. arXiv
17. 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
18. C. Shorten and T. M. Khoshgoftaar, “A Survey on Image Data Augmentation for Deep Learning,” *Journal of Big Data*, vol. 6, 2019. doi:10.1186/s40537-019-0197-0
19. A. Torralba and A. A. Efros, “Unbiased Look at Dataset Bias,” *CVPR*, 2011. paper page
20. Z. Liu and K. He, “A Decade’s Battle on Dataset Bias: Are We There Yet?,” *ICLR*, 2025. arXiv