1141　計算機視覺　54013

HW2 Edge Detection and Morphology

S1254040　資工三　羅章弘

民國一一四年十一月十九日星期三

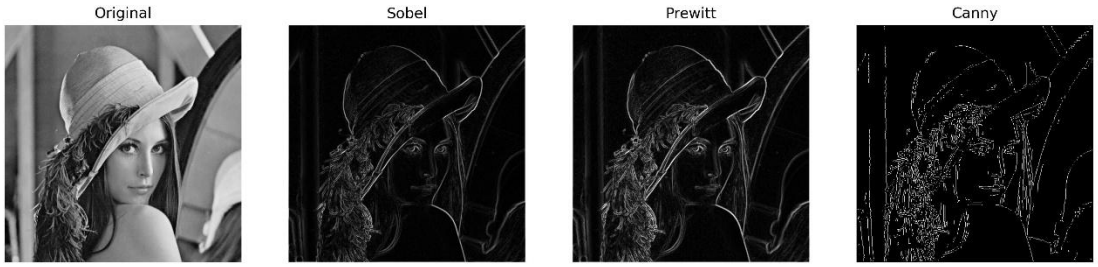
目錄

Environment

- Python 版本：3.12.7
- numpy 版本：2.2.6
- opencv-python 版本：4.12.0.88
- matplotlib：3.10.7

Edge detection

各邊緣偵測(Sobel、Prewitt、Canny)實作結果與原圖比較



Sobel

Sobel Filter 大尺寸卷積核之推導與實作方法

參考來源

https://stackoverflow.com/questions/9567882/sobel-filter-kernel-of-large-size

一、前言

Sobel Filter 為影像處理中常用的邊緣偵測方法，其經典形式為 3×3 卷積核，分別對應水平梯度 G_x 與垂直梯度 G_y。然而在某些應用中（如降低噪聲、增加平滑性），使用較大尺寸的 Sobel kernel（如 5×5、7×7）將能提供更穩定的梯度估計。

於此整理任意尺寸 Sobel kernel 的推導與建構方式。

二、經典 3×3 Sobel Filter

標準的 Sobel kernel 來源於以鄰域像素為基礎的梯度估計。水平與垂直方向之經典卷積核如下：

(1)水平方向 G_x

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

(2)垂直方向 G_y

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

這兩個 kernel 的設計理念包含：

1. 距離越遠、貢獻越小
2. 投影方向越符合梯度方向、權重越大

此概念可推廣至更大尺寸的 kernel。

三、任意尺寸 Sobel Kernel 的推導原理

1. 座標表示

假設 kernel 大小為 $N \times N$，中心位置座標為 $(0,0)$。

對於 kernel 中每個位置 (i,j)，其對梯度的貢獻可基於：

- 與中心距離 $d = \sqrt{i^2 + j^2}$
- 在 x、y 方向上的投影量（即 i、j）

2. 通用公式

任意位置的 Sobel kernel 權重可由下列原則推導：

(1)水平方向 G_x

$$G_x(i,j) = \frac{i}{i^2 + j^2}$$

(2)垂直方向 G_y

$$G_y(i,j) = \frac{j}{i^2 + j^2}$$

註：中心點 $(i,j) = (0,0)$ 因分母為 0，需設為 0。

此方法代表「距離反比權重」，並依方向給予投影比重。

這是一種自然、連續化的梯度估計方式，可延伸生成任意尺寸(5×5、7×7、9×9 …)的 Sobel kernel。

四、5×5 與 7×7 Sobel Kernel 之示例

1. 5×5 Sobel G_x 浮點版

$$\begin{bmatrix} -\frac{2}{8} & -\frac{1}{5} & 0 & \frac{1}{5} & \frac{2}{8} \\ -\frac{2}{5} & -\frac{1}{2} & 0 & \frac{1}{2} & \frac{2}{5} \\ -\frac{2}{4} & -1 & 0 & 1 & \frac{2}{4} \\ -\frac{2}{5} & -\frac{1}{2} & 0 & \frac{1}{2} & \frac{2}{5} \\ -\frac{2}{8} & -\frac{1}{5} & 0 & \frac{1}{5} & \frac{2}{8} \end{bmatrix}$$

2. 5×5 Sobel G_x 整數版（浮點版乘上 20）

$$\begin{bmatrix} -5 & -4 & 0 & 4 & 5 \\ -8 & -10 & 0 & 10 & 8 \\ -10 & -20 & 0 & 20 & 10 \\ -8 & -10 & 0 & 10 & 8 \\ -5 & -4 & 0 & 4 & 5 \end{bmatrix}$$

3. 7×7 Sobel G_x 浮點版

（依同原理建立：距離反比）

$$\begin{bmatrix} -\frac{3}{18} & -\frac{2}{13} & -\frac{1}{10} & 0 & \frac{1}{10} & \frac{2}{13} & \frac{3}{18} \\ -\frac{3}{13} & -\frac{2}{8} & -\frac{1}{5} & 0 & \frac{1}{5} & \frac{2}{8} & \frac{3}{13} \\ -\frac{3}{10} & -\frac{2}{5} & -\frac{1}{2} & 0 & \frac{1}{2} & \frac{2}{5} & \frac{3}{10} \\ -\frac{3}{9} & -\frac{2}{4} & -1 & 0 & 1 & \frac{2}{4} & \frac{3}{9} \\ -\frac{3}{10} & -\frac{2}{5} & -\frac{1}{2} & 0 & \frac{1}{2} & \frac{2}{5} & \frac{3}{10} \\ -\frac{3}{13} & -\frac{2}{8} & -\frac{1}{5} & 0 & \frac{1}{5} & \frac{2}{8} & \frac{3}{13} \\ -\frac{3}{18} & -\frac{2}{13} & -\frac{1}{10} & 0 & \frac{1}{10} & \frac{2}{13} & \frac{3}{18} \end{bmatrix}$$

大尺寸整數 kernel 因為比例縮放會造成誤差，實務上建議保持浮點格式。

註：Edge_detection\Sobel.py 程式碼最下面可以驗證出

五、與高斯導數（Gaussian Derivative）的比較

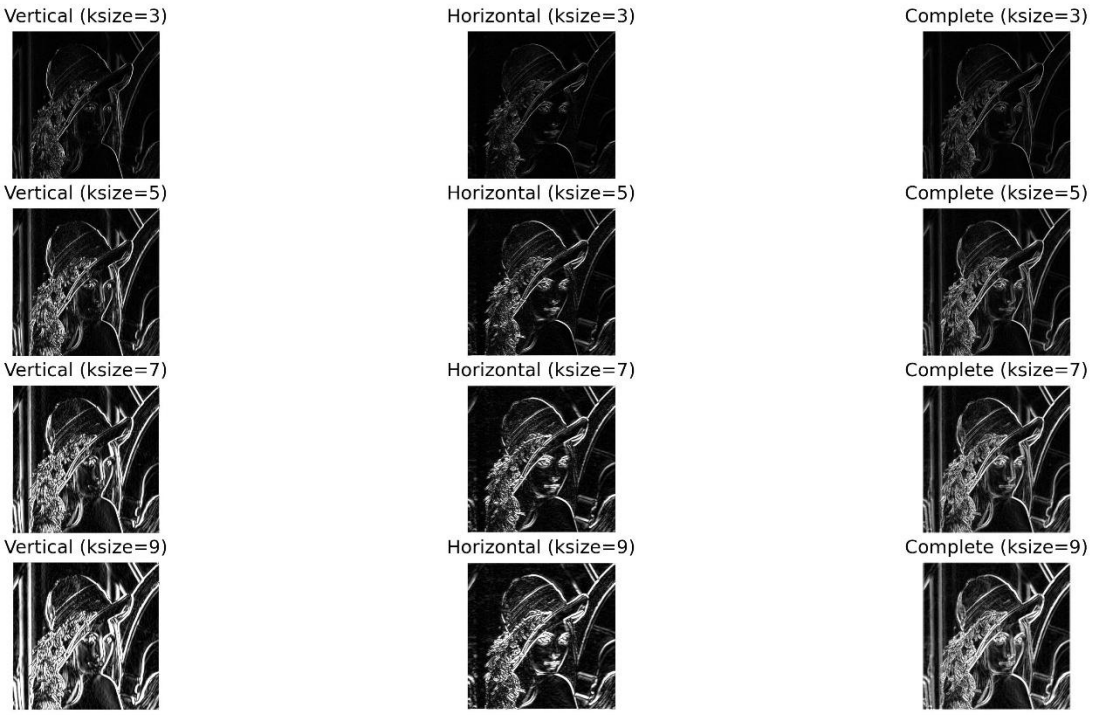
若目的是更平滑、更抗噪的梯度估計，使用 Gaussian derivative kernel（高斯導數）會比擴大 Sobel kernel 更理想。

理由：

- Gaussian 適用於連續化梯度推估
- 可自然實現多尺度特性（scale space）
- 頻域特性更優
- 噪聲抑制能力較強

因此，大尺寸 Sobel kernel 在某些情況下僅為折衷式解法。

實作結果



Sobel 不同 kernel size（3、5、7、9）結果簡單描述

整體來看，kernel size 越大，邊緣越平滑、線條越粗、細節越被抑制。

以下依三種結果（Vertical、Horizontal、Complete）說明：

1. Vertical（垂直邊緣）結果
   - ksize=3：邊緣最銳利、細節保留多，噪聲也較明顯。
   - ksize=5：垂直邊緣變得稍微粗一些，細節開始變平滑。
   - ksize=7：邊緣更粗，細小雜訊被平滑掉。
   - ksize=9：平滑化最強，垂直邊緣變得最粗、最柔和。

2. Horizontal（水平邊緣）結果
   - 與垂直方向類似，kernel 越大 -> 邊緣越平滑、線條越粗。
   - ksize=3：保留最多細節。
   - ksize=9：的結果則更強調主結構，細節較少。

3. Complete（結合 X 與 Y）結果

在 Complete 圖中，影像的水平邊緣(absX)與垂直邊緣(absY)被合併成一張完整的邊緣圖。

程式碼：dst = cv.addWeighted(absX, 0.5, absY, 0.5, 0)

把水平邊緣與垂直邊緣平均結合，兩者各佔 50%權重，形成同時包含 X 與 Y 方向梯度的邊緣影像。

- ksize=3：具有最清晰與細緻的輪廓，邊緣銳利。
- ksize=5：整體邊緣變得自然平滑。
- ksize=7：細節明顯減少，較大範圍的輪廓更突出。
- ksize=9：最平滑、最粗糙的邊緣特徵，只強調主要結構。

總結

kernel 越大：邊緣越平滑、越粗；細節越少、噪聲越少。

kernel 越小：邊緣越銳利、細節越多，也較容易帶入噪聲。

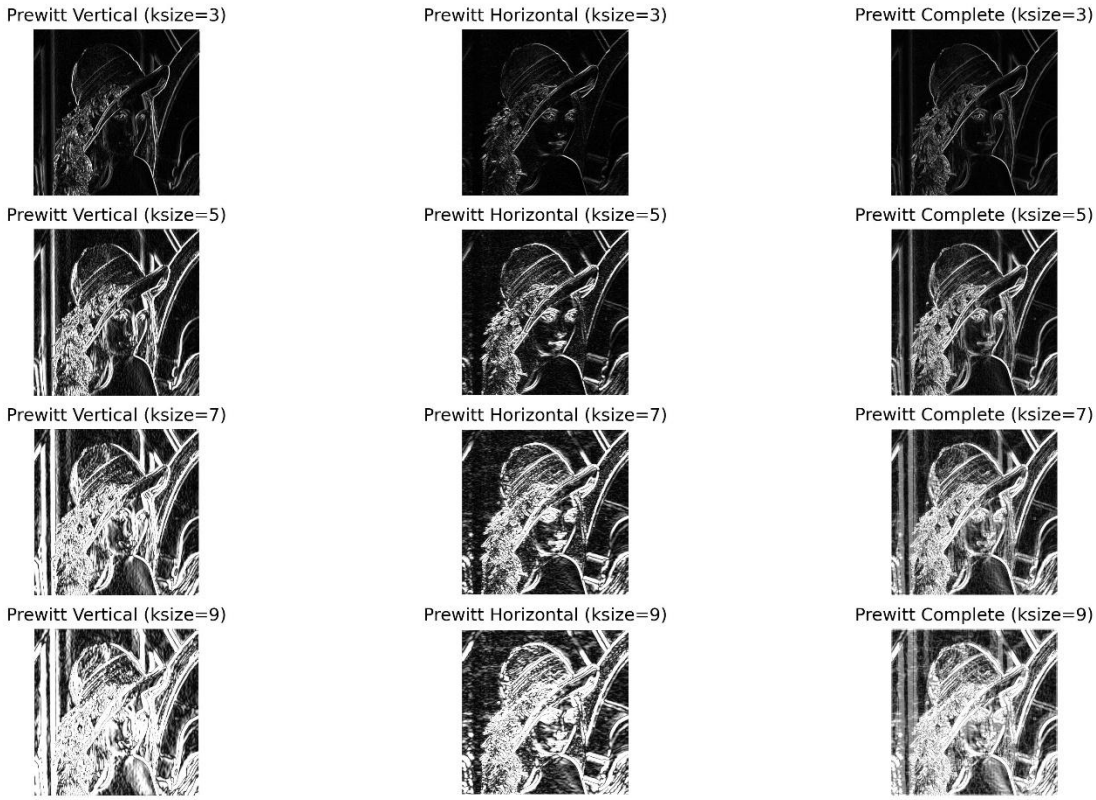
Prewitt

實作方式皆與 Sobel 相同，只差在 kernel 內容。Prewitt kernel 內容非常簡單，以 5x5 水平方向 G_x 為例：

$$\begin{bmatrix} -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \end{bmatrix}$$

左邊皆為-1，中間為 0，右邊皆為 1，可依此邏輯擴充。

實作結果



Canny

實作步驟

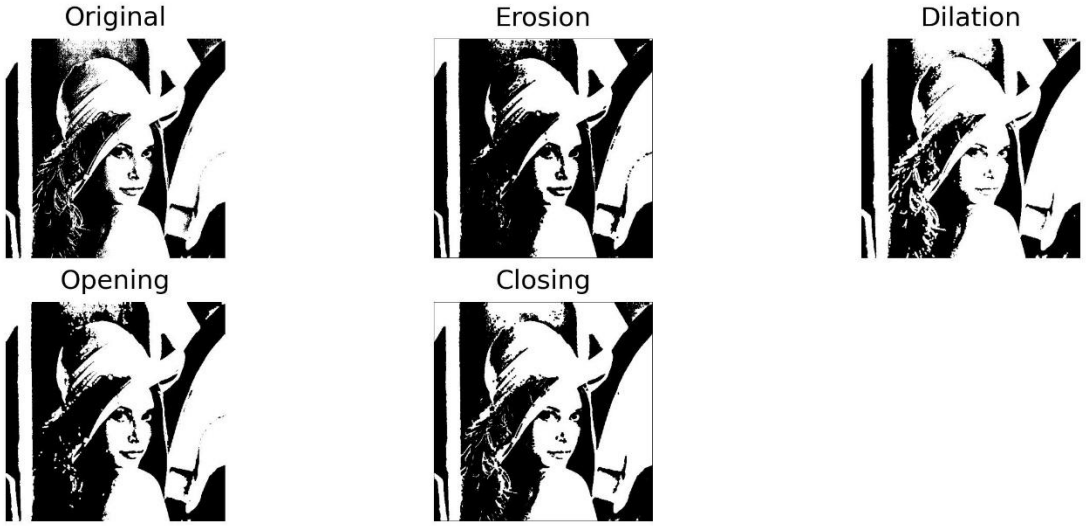
1. 高斯濾波平滑影像（Gaussian Blur）
   先去除影像噪聲，避免後面把雜點誤判成邊緣。
2. Sobel 計算梯度（Gradient by Sobel）
   利用 Sobel 求出水平方向與垂直方向的梯度，得到邊緣強度與方向。
   沿用前面已寫的 Sobel。
3. 非極大值抑制（NMS, Non-Maximum Suppression）
   只保留「真正最尖銳、最可能是邊緣」的像素，其他方向上的較弱值全部壓低成 0，使邊緣變得細而銳利。
4. 雙閾值 + 遲滯追蹤（Double Threshold + Hysteresis）
   - 強邊緣（大於高閾值）直接保留
   - 弱邊緣（介於兩個閾值間）只有在連到強邊緣時才保留
   - 其他全部捨棄

最終得到乾淨、連續、可靠的邊緣圖。

實作結果

Morphology

各形態學操作(erosion、dilation、opening、closing)實作結果與原圖比較



皆以 3x3 正方形作為結構元素

Erosion

主要程式碼：

```
# Perform erosion
for i in range(h):
    for j in range(w):
        region = padded[i:i+kh, j:j+kw]
        if np.all(region[kernel == 1] == 255):  # 結構元素區域全部為白色
            result[i, j] = 255
```

實作結果

Dilation

主要程式碼：

```
# Perform dilation
for i in range(h):
    for j in range(w):
        region = padded[i:i+kh, j:j+kw]
        if np.any(region[kernel == 1] == 255):  # 只要結構元素區域有白色
            result[i, j] = 255
```

實作結果

Opening

主要程式碼：

導入前面做好的 Erosion 與 Dilation

```python
# 兼容相對匯入與直接執行
try:
    from .Erosion import erosion
    from .Dilation import dilation
except Exception:
    # 當直接以 python Closing.py 執行（__main__），相對匯入會失敗
    # 此處回退到同資料夾的絕對匯入
    from Erosion import erosion
    from Dilation import dilation

# Opening function
# 先侵蝕，再膨脹
def opening(binary_img, kernel=None):
    eroded = erosion(binary_img, kernel)
    opened = dilation(eroded, kernel)
    return opened
```

實作結果

Closing

主要程式碼：

導入前面做好的 Erosion 與 Dilation，導入程式碼同 Opening

```
# Closing function
def closing(binary_img, kernel=None):
    # 先膨脹，再侵蝕
    dilated = dilation(binary_img, kernel)
    closed = erosion(dilated, kernel)
    return closed
```

實作結果