S1154007 賴宥瑋 計算機視覺作業 2

Table of contents

一、Edge detection

1-1. Sobel

做法:

1. 定義 sobel 核(gx 和 gy)
2. 分別將兩個核跟圖片做捲積(convolve2d)計算
3. 最後取平方和開根號作為最後的輸出圖片

```
# 定義 Sobel 卷積核
sobel_gx = np.array([[-1, -2, -1],
                     [ 0,  0,  0],
                     [ 1,  2,  1]])

sobel_gy = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]])
```

```
def convolve2d(image, kernel): # 手動卷積
    height, width = image.shape # 照片大小
    k_height, k_width = kernel.shape # kernel 大小
    pad_height, pad_width = k_height // 2, k_width // 2 # padding用，上下(kh // 2)，左右(kw // 2)
    # padding，使用reflect
    padded_img = np.pad(image, ((pad_height, pad_height), (pad_width, pad_width)), mode='reflect')
    result = np.zeros_like(image, dtype=float) # 輸出照片初始化為0

    # 卷積操作
    for x in range(height):
        for y in range(width):
            result[x, y] = np.sum(padded_img[x:x + k_height, y:y + k_width] * kernel)

    return result
```

```
# 計算 Sobel 水平方向與垂直方向的梯度
sobel_x_grad = convolve2d(image, sobel_gx)
sobel_y_grad = convolve2d(image, sobel_gy)

# 計算 Sobel 梯度幅值
sobel_grad = np.sqrt(sobel_x_grad**2 + sobel_y_grad**2)
```

1-2. Prewitt

做法:

1. 定義 Prewitt 核(gx 和 gy)
2. 分別將兩個核跟圖片做捲積(convolve2d)計算:跟 Sobel 一樣
3. 最後取平方和開根號作為最後的輸出圖片

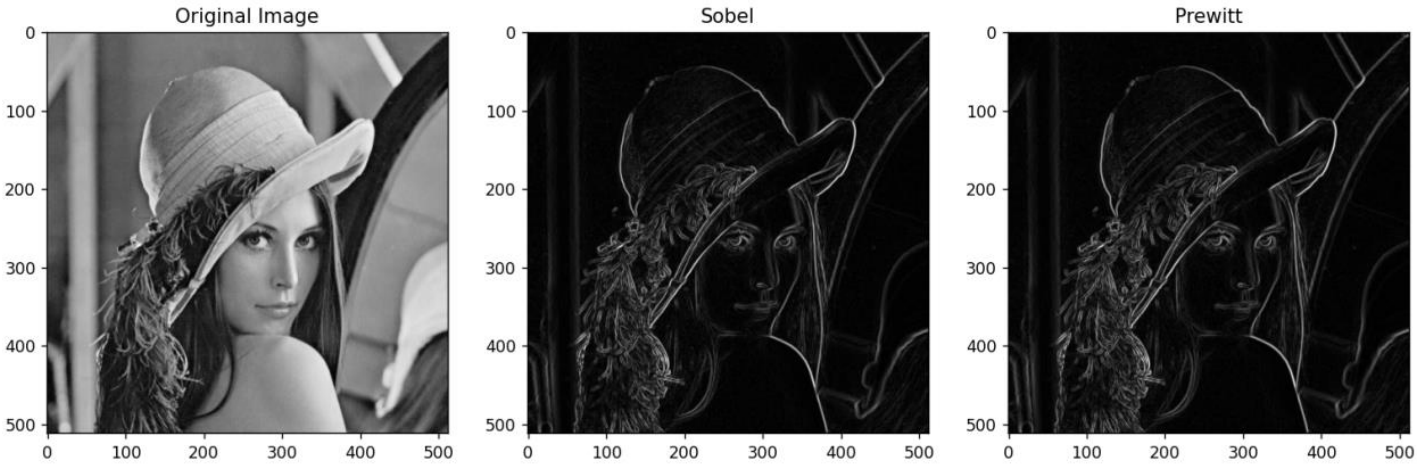
```
# 定義 Prewitt 卷積核
prewitt_gx = np.array([[-1, -1, -1],
                       [ 0,  0,  0],
                       [ 1,  1,  1]])

prewitt_gy = np.array([[-1, 0, 1],
                       [-1, 0, 1],
                       [-1, 0, 1]])
```

```
# 計算 Prewitt 水平方向與垂直方向的梯度
prewitt_x_grad = convolve2d(image, prewitt_gx)
prewitt_y_grad = convolve2d(image, prewitt_gy)

# 計算 Prewitt 梯度幅值
prewitt_grad = np.sqrt(prewitt_x_grad**2 + prewitt_y_grad**2)
```

Result



1-3. Canny

做法:

1. 使用高斯濾波:產生網格->計算值 $fs(x,\ y) = G(x,\ y) \star f(x,\ y)$

```
def gaussian_filter_2d(shape, sigma):  # 高斯濾波(2D)數值計算
    m, n = [(ss - 1) // 2 for ss in shape]  # m, n為產生網格用
    x, y = np.ogrid[-m:m + 1, -n:n + 1]  # 產生網格座標
    h = np.exp(-(x * x + y * y) / (2. * sigma * sigma))  # 公式計算
    h = h / (2. * np.pi * sigma * sigma)  # 公式計算
    return h / h.sum()  # Normalization

def gaussian_filter(image, sigma) : # 將圖片作用於高斯filter
    kernel_size = int(6 * sigma + 1)
    if(kernel_size % 2 != 1):
        kernel_size += 1 # 確保大小為奇數

    kernel = gaussian_filter_2d((kernel_size, kernel_size), sigma)  # 使用高斯濾波器
    smooth = convolve2d(image, kernel)  # 使用手動卷積
    return smooth                          也是用之前在 Sobel 定義的
```

2. 計算 magnitude 和角度(α):利用 Sobel

```
def gradient_magnitude_and_angle(image): # 計算magnitude和angle
    # 計算梯度x和y方向的梯度
    gx = convolve2d(image, np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]))  # x方向梯度
    gy = convolve2d(image, np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]))  # y方向梯度
    magnitude = np.sqrt(gx**2 + gy**2) # magnitude 計算
    angle = np.arctan2(gy, gx) * (180 / np.pi) # 將弧度轉成角度
    angle[angle < 0] += 180  # 將角度轉換為0到180度
    return magnitude, angle  # 將magnitude和angle回傳
```

3. Preserve edge points : Non-maxima suppression

```
def non_maxima_suppression(magnitude, angle): # 非最大值抑制
    height, width = magnitude.shape # 圖片大小
    suppressed = np.zeros_like(magnitude) # 初始值為0

    # 從(1,1)開始到(254,254)
    for x in range(1, height - 1):
        for y in range(1, width - 1):
            direction = angle[x, y] # 夾角，與x軸
            if direction < 22.5 or direction >= 157.5: # 垂直方向
                neighbors = [magnitude[x - 1, y], magnitude[x + 1, y]] # 選上下
            elif 22.5 <= direction < 67.5: # -45度角方向
                neighbors = [magnitude[x - 1, y - 1], magnitude[x + 1, y + 1]] # 選左上右下
            elif 67.5 <= direction < 112.5: # 水平方向
                neighbors = [magnitude[x, y - 1], magnitude[x, y + 1]] # 選左右
            else: # 45度角方向
                neighbors = [magnitude[x + 1, y - 1], magnitude[x - 1, y + 1]] # 選左下右上

            if magnitude[x, y] >= max(neighbors): # 如果magnitude > 兩個neighbors
                suppressed[x, y] = magnitude[x, y] # 保留，否則suppression(初始值本來就是0)

    return suppressed
```

4. Edge determination

```python
def edge_determination(suppressed, low_threshold, high_threshold): # 邊緣確定
    height, width = suppressed.shape
    strong_edges = (suppressed >= high_threshold) # 強邊緣(存座標)
    weak_edges = (suppressed >= low_threshold) & (suppressed < high_threshold) # 弱邊緣 - 強邊緣 (存座標)

    #初始化0
    final_edges = np.zeros_like(suppressed)

    # 檢查所有點確認是否為強邊緣
    for x in range(1, height - 1):
        for y in range(1, width - 1):
            #如果為強邊緣
            if strong_edges[x, y]:
                final_edges[x, y] = 255  # 強邊緣 = 255
                # 檢查周圍的弱邊緣(8-connectivity)
                for j in range(-1, 2):
                    for i in range(-1, 2):
                        if weak_edges[x + j, y + i]:
                            final_edges[x + j, y + i] = 255  # 連接的弱邊緣 = 255
```

5. 套用步驟 1~4 形成 canny edge detection

```python
def canny_edge_detection(image): # 所有步驟

    # 高斯平滑(step 1 : smoothing use Guassian)
    smoothed = gaussian_filter(image, sigma=1.4) # sigma = 1.4

    # 計算梯度值和角度(step 2 : Compute gradient magnitude and angle)
    magnitude, angle = gradient_magnitude_and_angle(smoothed)

    # 非最大值抑制(step 3 : Preserve edge points)
    suppressed = non_maxima_suppression(magnitude, angle)

    # 弱邊緣 強邊緣設定
    low_threshold = 30
    high_threshold = 60

    # 邊緣確認(step 4 : Edge determination)
    canny_img = edge_determination(suppressed, low_threshold, high_threshold)

    return canny_img
```

Result

Original Image

Canny

二、Morphology

2-1. Dilation

做法：從左上角開始，若有碰到(sum>0)，反白

```python
def dilation(image, kernel):
    h, w = image.shape # 圖片大小 h*w
    kernel_height, kernel_width = kernel.shape # kernel大小
    pad_height, pad_width = kernel_height // 2, kernel_width // 2 # padding值為(kernel size // 2)

    # 在上下左右填上0(若kernel size = 3，則在上下左右各延伸一像素) padding值為0(因為填充不應該影響原先的像素進行dilation)
    padded_image = np.pad(image, ((pad_height, pad_height), (pad_width, pad_width)), mode='constant', constant_values=0)

    #初始化output_image(全黑背景)
    output_image = np.zeros_like(image)

    # 遍歷整張圖
    for i in range(h):
        for j in range(w):
            # 以該點當最左上角，產生一個大小為kernel_size的window size
            window = padded_image[i:i + kernel_height, j:j + kernel_width]
            if (window * kernel).sum() > 0: # 當有碰到
                output_image[i, j] = 1 # 反白

    return output_image
```

```python
# kernel
kernel = np.ones((5,5), np.uint8)
```

Kernel:在 erosion opening closing 都使用此核

[[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1],
[1, 1, 1, 1, 1]]

Result

2-2. Erosion

做法: 從左上角開始，只要沒有全部碰到(sum != 5*5)，變黑

```python
def erosion(image, kernel):
    h, w = image.shape # 圖片大小 h*w
    kernel_height, kernel_width = kernel.shape # kernel大小
    pad_height, pad_width = kernel_height // 2, kernel_width // 2 # padding值為(kernel size // 2)

    # 在上下左右填上0(若kernel size = 3，則在上下左右各延伸一像素) padding值為1(因為填充不應該影響原先的像素進行erosion)
    padded_image = np.pad(image, ((pad_height, pad_height), (pad_width, pad_width)), mode='constant', constant_values=1)

    #初始化output_image(全白背景)
    output_image = np.ones_like(image)

    # 遍歷整張圖
    for i in range(h):
        for j in range(w):
            # 以該點當最左上角，產生一個大小為kernel_size的window size
            window = padded_image[i:i + kernel_height, j:j + kernel_width]
            if (window * kernel).sum() != kernel.sum(): # 當沒有全部碰到
                output_image[i, j] = 0 # 變黑

    return output_image
```

Result

2-3. Opening

做法：$A \circ B = A \ominus B \oplus B$ (都是利用先前的 dilation 和 erosion 函式)

```
def opening(image, kernel):
    eroded = erosion(image, kernel)        # 先侵蝕
    opened = dilation(eroded, kernel)      # 再膨脹
    return opened
```

2-4. Closing

做法：$A \bullet B = A \oplus B \ominus B$ (都是利用先前的 dilation 和 erosion 函式)

```
def closing(image, kernel):
    dilated = dilation(image, kernel)      # 先膨脹
    closed = erosion(dilated, kernel)      # 再侵蝕
    return closed
```

Result