Image Processing

Programming Assignment 2

311551096 錢承

1. Introduction / Objectives

實作 Canny edge detector，此演算法中包含許多不同的過程，每一個過程都是一個獨立的演算法，包含降噪處理所需要的 Gaussian Filter、計算梯度的 Sobel Operator、尋找梯度變化最大點的 Non-Maximum Suppression、偵測潛在邊緣的 Double threshold 以及判斷是否為邊界的 Hysteresis。

2. A review of the methods you have used (be concise)

本次實作主要完成以下五種方法 Gaussian Filter、Sobel Operator、Non-Maximum Suppression、Double threshold 和 Hysteresis。

a. Gaussian Filter

使用 gaussian Filter 作為 kernel，期望可以藉由模糊圖像來將雜訊濾除，如下圖所示。

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

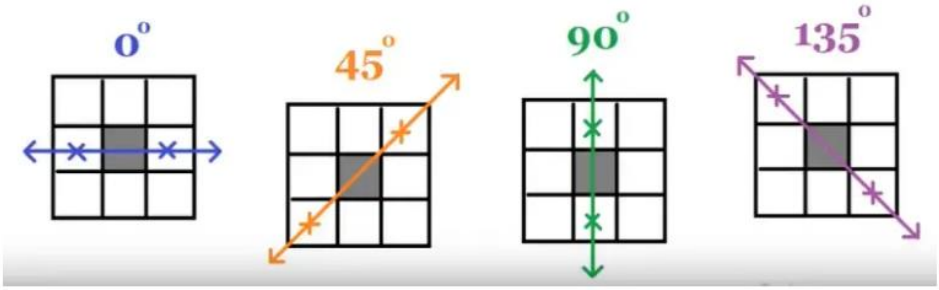
b. Sobel Operator

使用 Sobel Filter 作為 kernel，可以計算圖片的 gradient，如下圖所示。

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

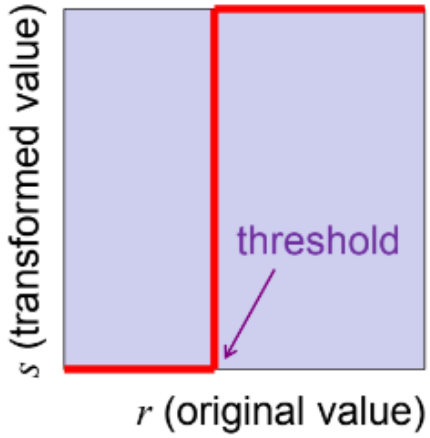
c. Non-Maximum Suppression

尋找四個方向中，梯度變化最大的點，其餘的兩個點的梯度給 0。



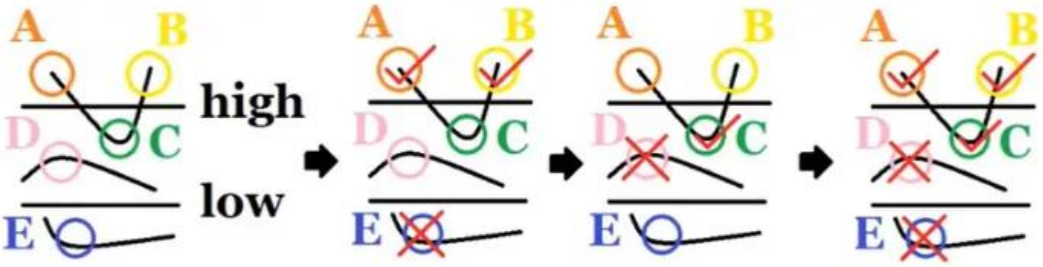
d. Double Thresholding

Thresholding 採用類似於 Power-Law Transformations 的方法，不過函數的 output 只有兩個值。而 Canny edge detector 中使用的 Double Thresholding，將圖片分為三個值，低於 low threshold 的位置全部給 0，介於 low threshold 和 high threshold 直接的位置給定為 weak pixel，而高於 high threshold 的位置全部給定為 strong pixel。



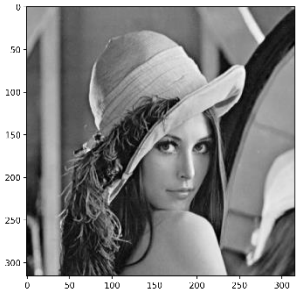
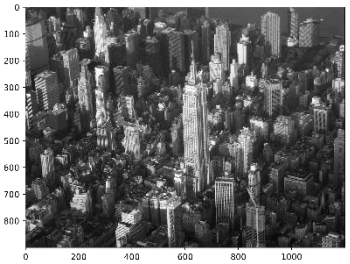
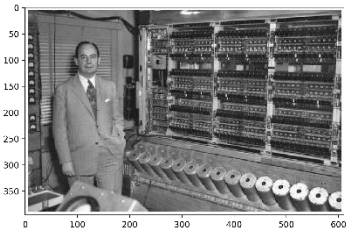
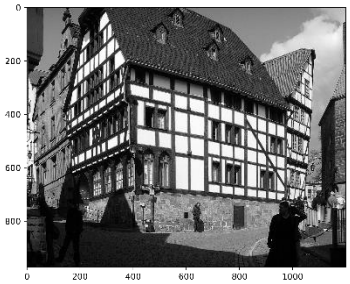
e. Hysteresis

weak pixel 周圍若有 strong pixel 時，weak pixel 將轉變為 strong pixel，反之則直接給 0。



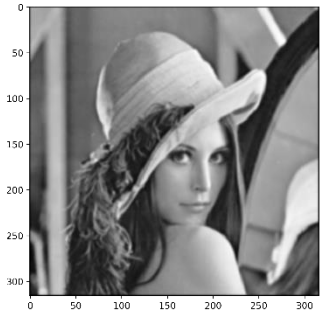
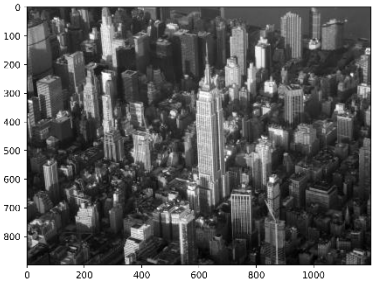
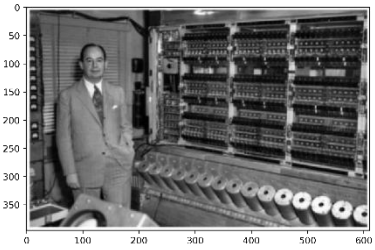
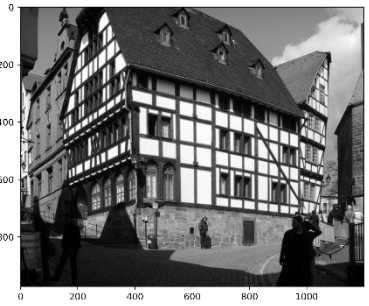
3. An explanation of the experiments you have done, and the results.

a. Original images

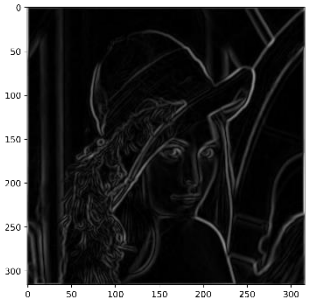
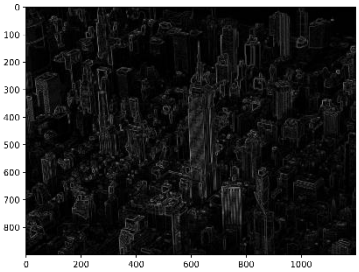
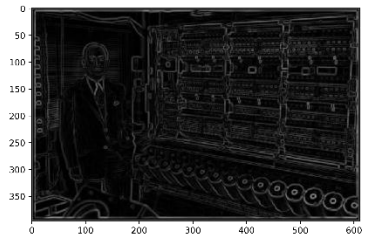
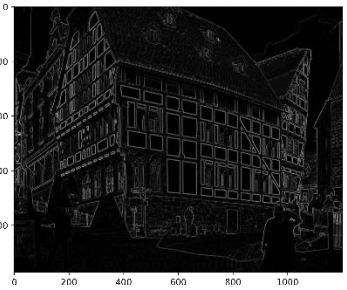
b. Gaussian Filter

經過 Gaussian Filter 後可以觀察到，圖片變得模糊，但是大部分細微的雜訊也同時被去除。

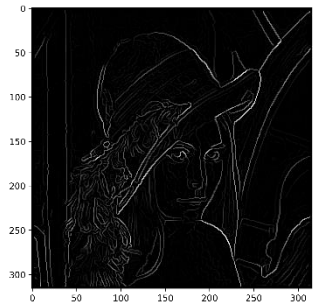
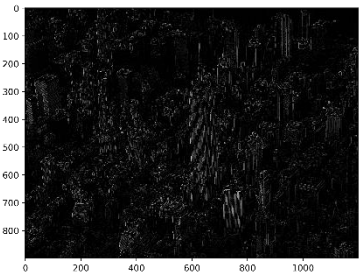
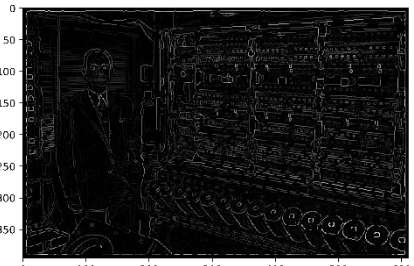
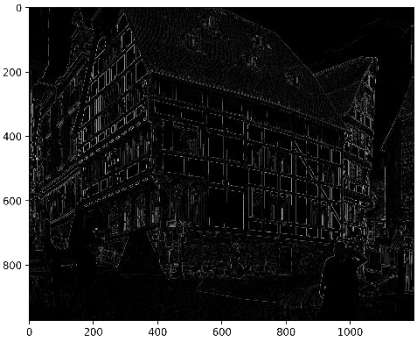
c. Sobel Operator

此時可以看到，透過 Sobel Operator 後計算出來的梯度，已經可以初步看到邊緣了。

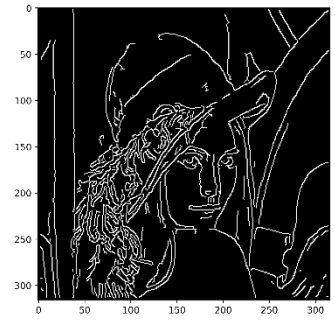
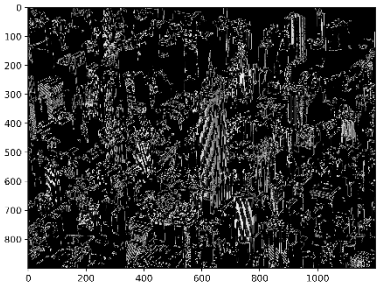
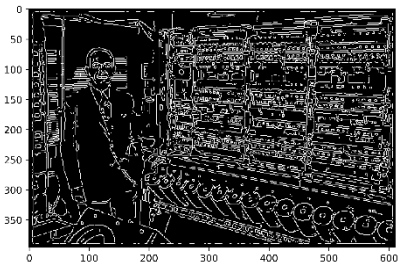
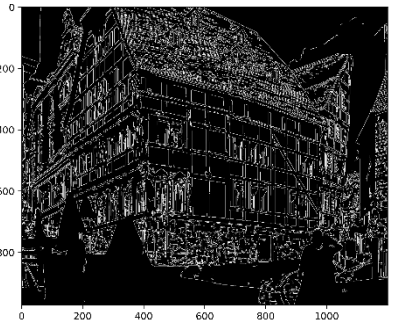
d. Non-Maximum Suppression

此步驟之後，有效的選出梯度大的部分，從圖片中可以觀察到，邊緣的地方變得更加銳利了。

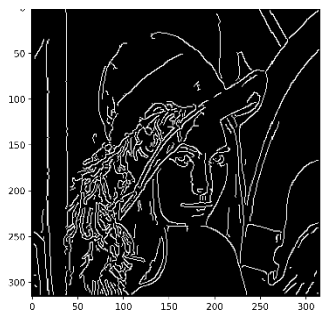
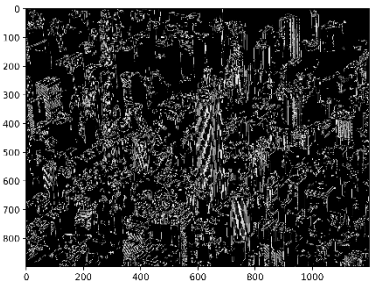
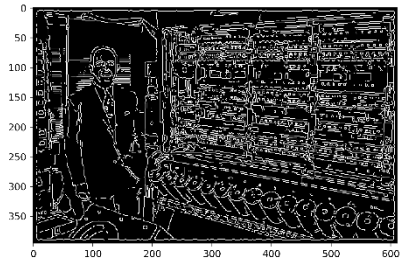
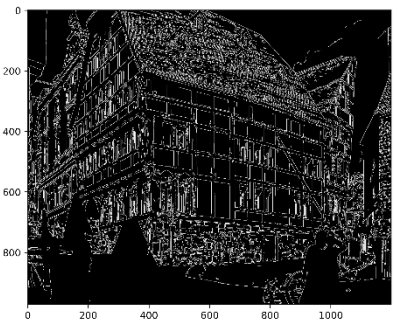
e. Double Thresholding

依照設定的兩個 Thresholding 後，將圖片分為三個值，分別為 0、weak pixel、strong pixel，而由圖片可以觀察到絕大部分為 strong pixel。

f. Hysteresis

最後看 weak pixel 附近有沒有存在 strong pixel，若有就直接變成 strong pixel，否則直接變 0。但是由第四點可以知道，絕大部分為 strong pixel，因此此步驟和前一步的結果大致相同。

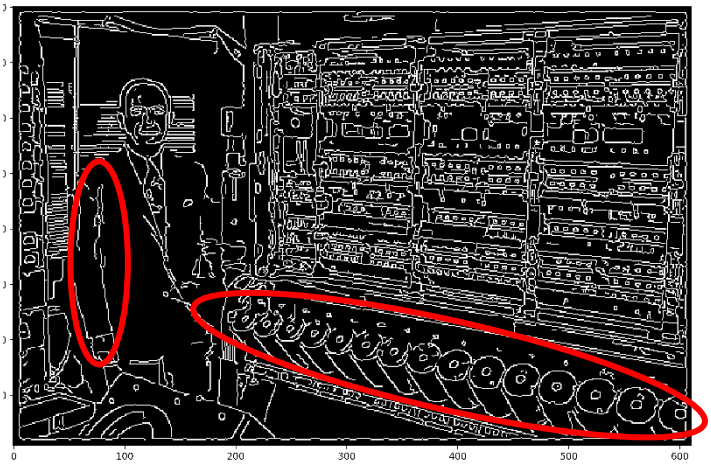





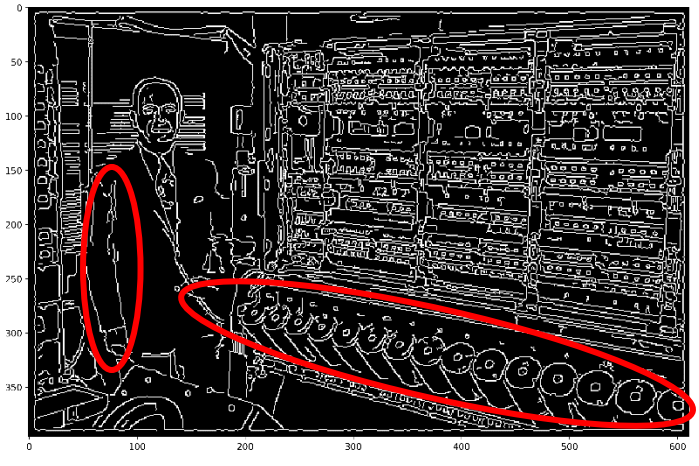
4. Discussions

我利用 Rong W, Li Z, Zhang W, et al.在“An improved CANNY edge detection algorithm”中提出的 kernel(右)計算 gradient，並且與我原先使用的方法(左)進行比較，可以發現在紅圈處，邊緣有更好的被偵測出來。

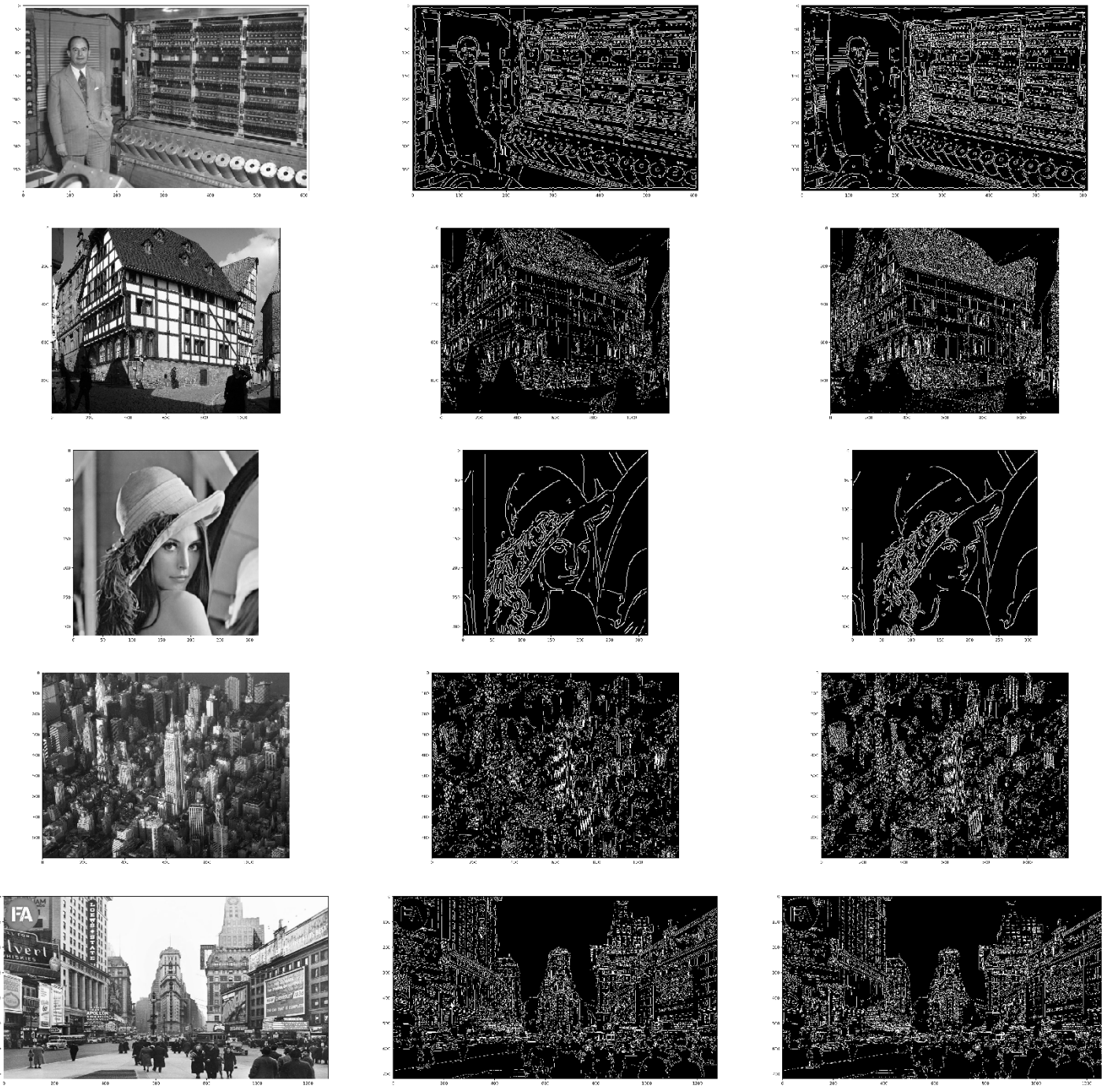
Rong W, Li Z, Zhang W, et al.提出的 kernel:

$$G_x = \begin{pmatrix} -\frac{\sqrt{2}}{4} & 0 & \frac{\sqrt{2}}{4} \\ -1 & 0 & 1 \\ -\frac{\sqrt{2}}{4} & 0 & \frac{\sqrt{2}}{4} \end{pmatrix} \qquad G_y = \begin{pmatrix} \frac{\sqrt{2}}{4} & 1 & \frac{\sqrt{2}}{4} \\ 0 & 0 & 0 \\ -\frac{\sqrt{2}}{4} & -1 & -\frac{\sqrt{2}}{4} \end{pmatrix}$$

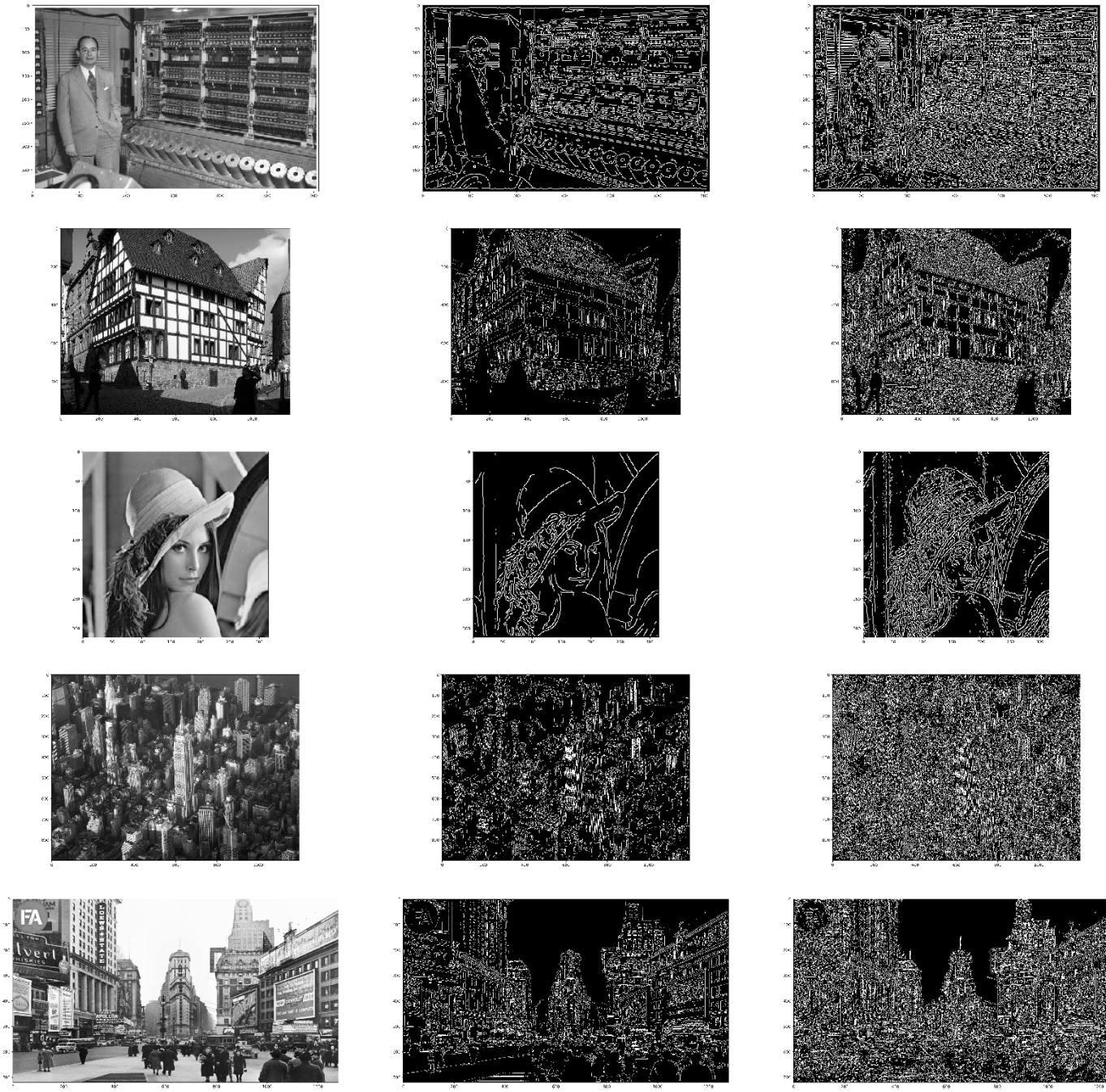




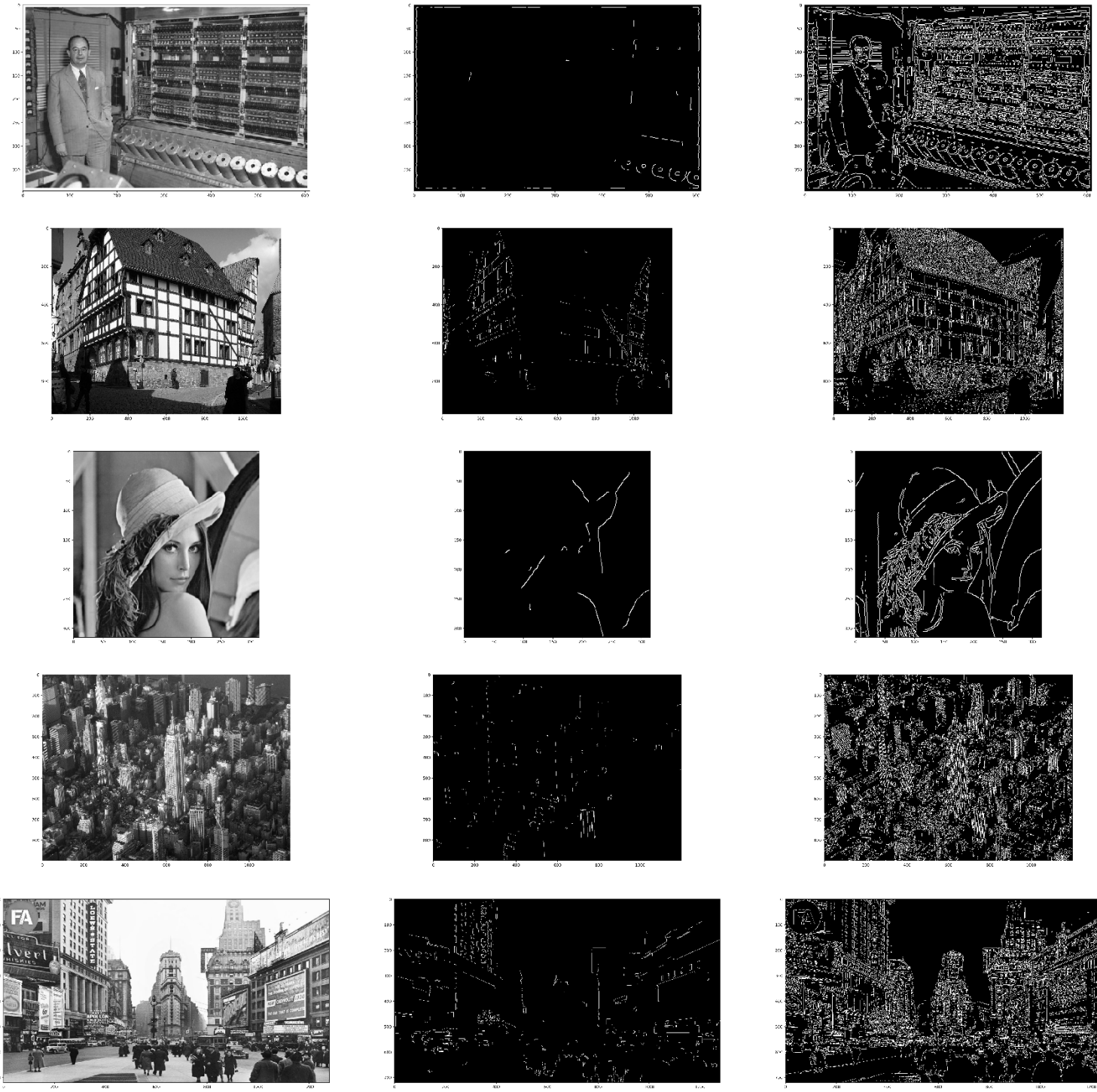
下圖為原圖(左)、我實作的 Canny edge detector(中)以及的 openCV 提供的 Canny edge detector function(右)比較，可以觀察到兩者都可以達成很好的效果。其中，我實作的 Canny 兩個 threshold 分別為 0.10 和 0.15，而 openCV 的 Canny 兩個 threshold 分別為 0.5 和 0.8。



下圖為原圖(左)、我實作的 Canny edge detector(中)以及的 openCV 提供的 Canny edge detector function(右)比較，我實作的和 openCV 的 Canny 兩個 threshold 皆為 0.12 和 0.15。可以觀察到此 threshold 對兩者的實作都來說太小了，導致許多不是邊緣的地方被認定為邊緣。



下圖為原圖(左)、我實作的 Canny edge detector(中)以及的 openCV 提供的 Canny edge detector function(右)比較，我實作的和 openCV 的 Canny 兩個 threshold 皆為 0.4 和 0.7。可以觀察到此 threshold 對我的實作都來說太大了，導致許多邊緣邊緣被濾掉。但是對於 openCV 的實作來說適當，邊緣的地方有被保留，而其他地方有適當的濾除



5. Program Code

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def dataloader(dir_name = 'data'):
    imgs = []
    for filename in os.listdir(dir_name):
        if os.path.isfile(dir_name + '/' + filename):
            img = cv2.imread(dir_name + '/' + filename, cv2.IMREAD_GRAYSCALE)
            imgs.append(img)
    return imgs


def visualize_part(imgs, name = 'img'):
    plt.figure(figsize=(15, 20))
    for i, img in enumerate(imgs):
        plt.subplot(int(len(imgs)//2+1), 2, int(i+1//2+1))
        plt.imshow(img, 'gray')

    plt.savefig(name + '.png', dpi=500)


def visualize(imgs, images_original, format = 'gray', lowthreshold = 0.4,
highthreshold = 0.7, name = 'img'):
    plt.figure(figsize=(50, 50))

    for i, img in enumerate(zip(images_original, imgs)):
        plt_idx = (i+1)*3 - 2
        plt.subplot(len(imgs), 3, plt_idx)
        plt.imshow(img[0], format)

        plt_idx = (i+1)*3 - 1
        plt.subplot(len(imgs), 3, plt_idx)
        plt.imshow(img[1], format)

        plt_idx = (i+1)*3
        plt.subplot(len(imgs), 3, plt_idx)
        plt.imshow(cv2.Canny(img[0], lowthreshold * 255, highthreshold *
255), format)

    plt.savefig(name + '.png', dpi=500)
```

```
class cannyEdgeDetector:
    def __init__(self, lowthreshold=0.12, highthreshold=0.15):
        self.weak_pixel     = 100
        self.strong_pixel   = 255
        self.lowThreshold   = lowthreshold
        self.highThreshold  = highthreshold
        self.imgs_final     = []
        self.smoothedImg    = []
        self.gradient       = []
        self.theta          = []
        self.nonMaxImg      = []
        self.thresholdImg   = []


    def conv(self, image, kernel, stride = 1, padding = 1):
        image = np.pad(image, [(padding, padding), (padding, padding)],
mode='constant', constant_values=0)

        kernel_height, kernel_width = kernel.shape
        padded_height, padded_width = image.shape

        output_height = (padded_height - kernel_height) // stride + 1
        output_width = (padded_width - kernel_width) // stride + 1

        new_image = np.zeros((output_height,
output_width)).astype(np.float32)
        for y in range(0, output_height):
            for x in range(0, output_width):
                new_image[y][x] = np.sum(image[y * stride:y * stride +
kernel_height, x * stride:x * stride + kernel_width] *
kernel).astype(np.float32)

        return new_image


    def gaussian_filters(self, img):

        kernel = np.array(
            [[ 1,  4,  7,  4,  1],
             [ 4, 16, 26, 16,  4],
             [ 7, 26, 41, 26,  7],
             [ 4, 16, 26, 16,  4],
             [ 1,  4,  7,  4,  1] ]
        ) /273.
```

```
        return self.conv(img, kernel, padding = 2)


    def sobel_filters(self, img):

        kernel_x = np.array([[-1, 0, 1],
                             [-2, 0, 2],
                             [-1, 0, 1]])
        gradient_x = self.conv(img, kernel_x)

        kernel_y = np.array([[ 1, 2, 1],
                             [ 0, 0, 0],
                             [-1,-2,-1]])
        gradient_y = self.conv(img, kernel_y)

        gradient = np.hypot(gradient_x, gradient_y)
        gradient = gradient / gradient.max() * 255
        theta = np.arctan2(gradient_y, gradient_x)
        return gradient, theta


    def non_max_suppression(self, img, angle):
        x, y    = img.shape
        output = np.zeros((x,y))
        angle   = angle * 180. / np.pi
        angle[angle < 0] += 180


        for i in range(1,x-1):
            for j in range(1,y-1):
                front = 255
                back  = 255

                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                    front = img[i, j+1]
                    back  = img[i, j-1]

                elif (22.5 <= angle[i,j] < 67.5):
                    front = img[i+1, j-1]
                    back  = img[i-1, j+1]

                elif (67.5 <= angle[i,j] < 112.5):
                    front = img[i+1, j]
                    back  = img[i-1, j]
```

```
                elif (112.5 <= angle[i,j] < 157.5):
                    front = img[i-1, j-1]
                    back  = img[i+1, j+1]


                if (img[i,j] >= front) and (img[i,j] >= back):
                    output[i,j] = img[i,j]
                else:
                    output[i,j] = 0

        return output


    def threshold(self, img):

        highThreshold = img.max() * self.highThreshold
        lowThreshold = highThreshold * self.lowThreshold

        strong_i, strong_j = np.where(img >= highThreshold)
        weak_i  , weak_j   = np.where((img <= highThreshold) & (img >=
lowThreshold))

        img  = np.zeros(img.shape)
        img[strong_i, strong_j] = self.strong_pixel
        img[weak_i, weak_j]     = self.weak_pixel

        return img


    def hysteresis(self, img):

        M, N   = img.shape
        weak   = self.weak_pixel
        strong = self.strong_pixel

        for i in range(1, M-1):
            for j in range(1, N-1):
                if (img[i,j] == weak):
                    if (   (img[i+1, j-1] == strong) or (img[i+1, j] ==
strong) or (img[i+1, j+1] == strong)
                        or (img[i  , j-1] ==
strong)           or                (img[i  , j+1] == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] ==
strong) or (img[i-1, j+1] == strong)):
                        img[i, j] = strong
```

```
                    else:
                        img[i, j] = 0

        return img


    def detect(self, imgs):
        for img in imgs:
            self.smoothedImg.append(self.gaussian_filters(img))
            gradient, theta  = self.sobel_filters(self.smoothedImg[-1])
            self.gradient.append(gradient)
            self.theta.append(theta)
            self.nonMaxImg.append(self.non_max_suppression(self.gradient[-1],
self.theta[-1]))
            self.thresholdImg.append(self.threshold(self.nonMaxImg[-1]))
            self.imgs_final.append(self.hysteresis(self.thresholdImg[-1]))

        return self.imgs_final


if __name__ == '__main__':
    images_original = dataloader()
    detector = cannyEdgeDetector(lowthreshold = 0.10, highthreshold = 0.15)
    imgs_final = detector.detect(images_original)
    visualize(imgs_final, images_original, lowthreshold = 0.5, highthreshold
= 0.8)

    visualize_part(detector.smoothedImg, 'smoothedImg')
    visualize_part(detector.nonMaxImg, 'nonMaxImg')
    visualize_part(detector.thresholdImg, 'thresholdImg')
    visualize_part(detector.imgs_final, 'imgs_final')
```