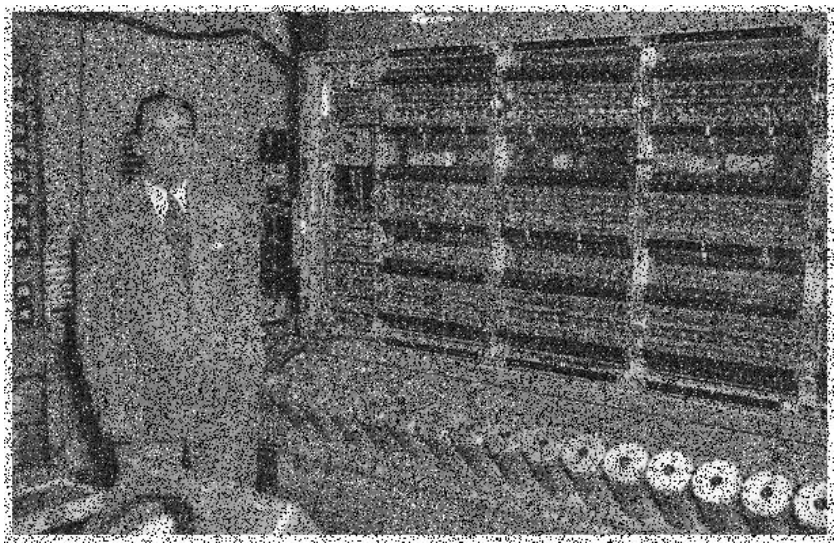
Image Processing

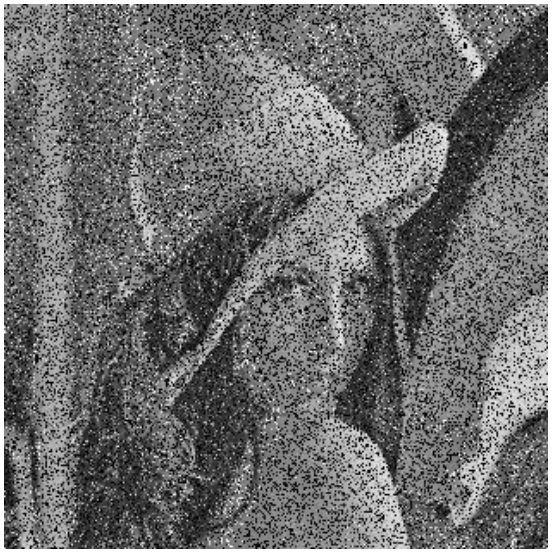
Programming Assignment 1

311551096 錢承

1. Introduction / Objectives

希望藉由 Power-Law Transformations、Histogram Equalization、Median Filters、Thresholding、Filters of Image Gradients、Gaussian Blur 等方法將下方四張具有高斯雜訊的彩色/黑白圖片進行強化或復原處理。我使用 python 實作 Convolution、Histogram、Median、Thresholding 等計算。





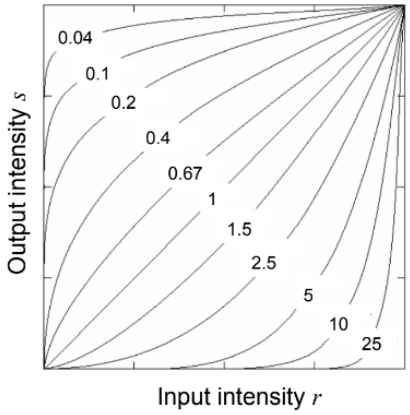
2. A review of the methods you have used (be concise)

本次實驗主要嘗試以下五種方法 Power-Law Transformations、Histogram Equalization、Median Filters、Thresholding、Filters of Image Gradients。

a. Power-Law Transformations:

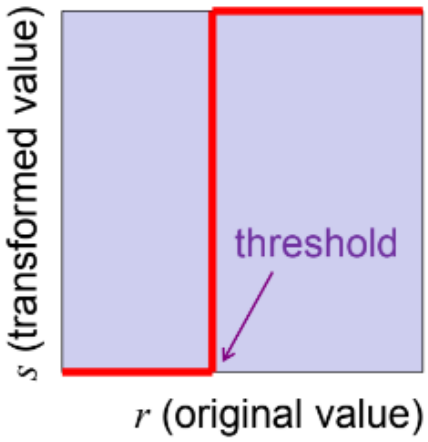
利用以下函數對每一個 pixel 進行處理，而我 c 和 γ 分別採用 1 和 1.5。

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



b. Thresholding:

採用類似於 Power-Law Transformations 的方法，不過函數的 output 只有兩個值。



c. Median Filters:

將 Mask 遮到的數據先做由小到大的排序，接著對此數列取中值，再來取代中間數據，計算方式如下。

$$g(x,y) = \sum_{i=1}^{n} w_i v_i(x,y)$$

$$w_i = \begin{cases} 1 & \text{if } i = (n+1)/2 \\ 0 & \text{otherwise} \end{cases} \quad \text{assuming odd } n$$

d. Histogram Equalization:

計算圖片中每一個 pixel 值的分佈，再來調整其分布狀況，最理想的結果就是平均分佈，計算方式如下圖。

Now let us consider the discrete case (value range $0 \sim L-1$):

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

Transformation function:

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

e. Filters of Image Gradients:

使用 Prewitt Filter 作為 kernel，期望可以偵測邊緣，如下圖所示。

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

f. Gaussian Blur

使用 gaussian Filter 作為 kernel，期望可以模糊圖像，如下圖所示。

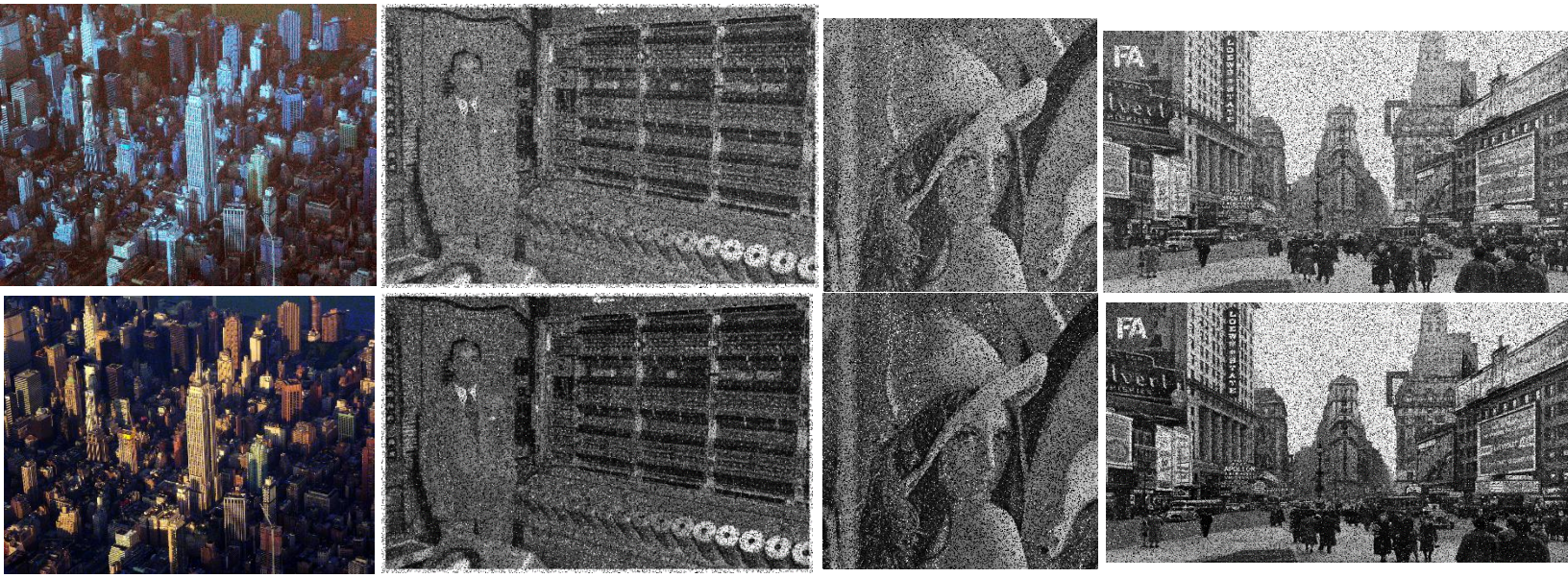
$\frac{1}{273}$

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

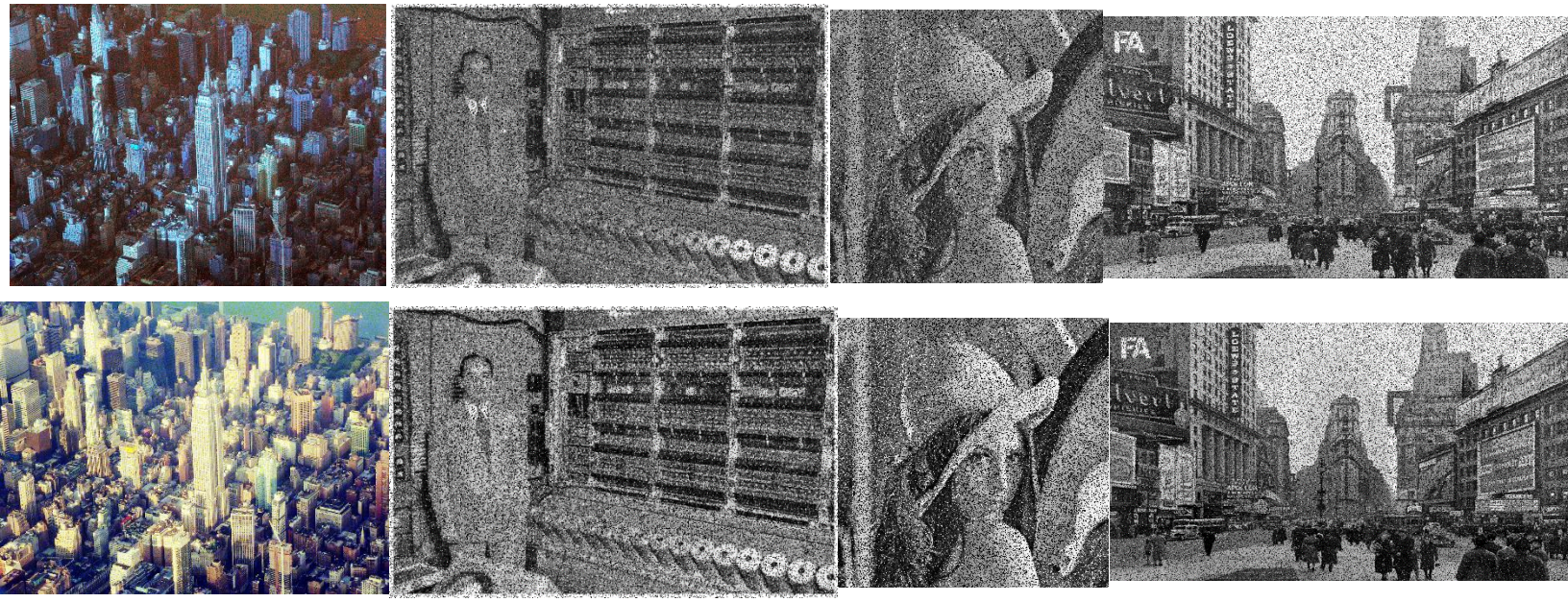
3. An explanation of the experiments you have done, and the results.

每張圖片接經過三個階段，分別為 Power-Law Transformations、Histogram Equalization 和 Median Filters。

我先對圖片進行 Power-Law Transformations，c 和 γ 分別採用 1 和 1.5，在彩色的圖片上，與原圖(上)相比已經有顯著的改善，可是依舊有雜訊存在。但是黑白的圖片，沒有顯著的改善。

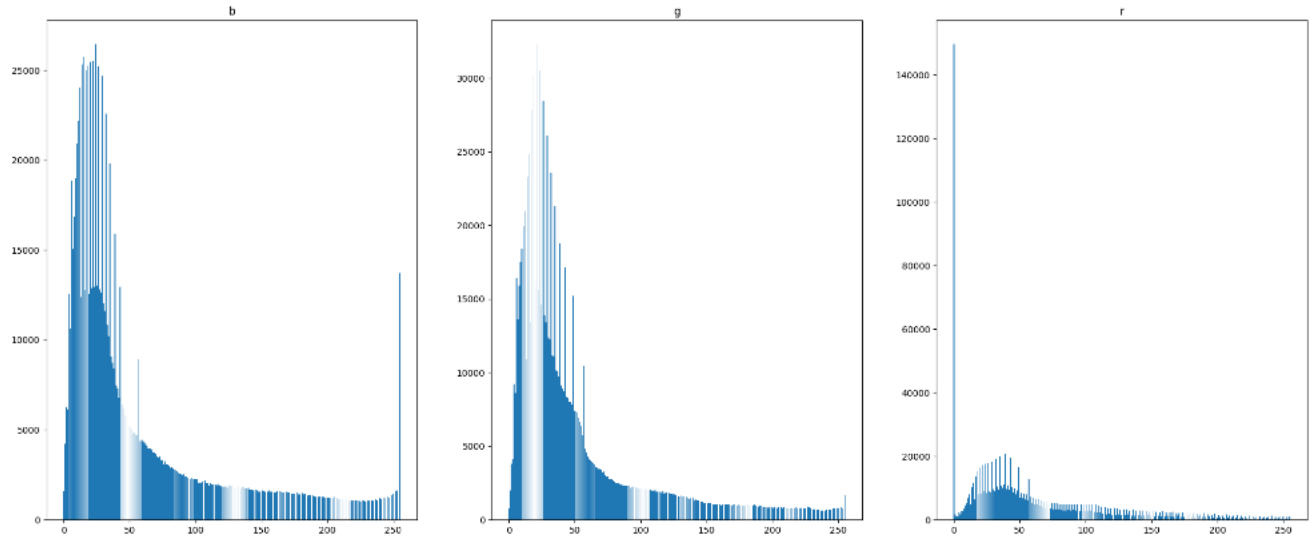


再來我對圖片進行 Histogram Equalization，可以觀察到各個圖片都可以顯示了更多細節。但是雜訊的部分在黑白圖片沒有改善，原圖(上)相比如下。

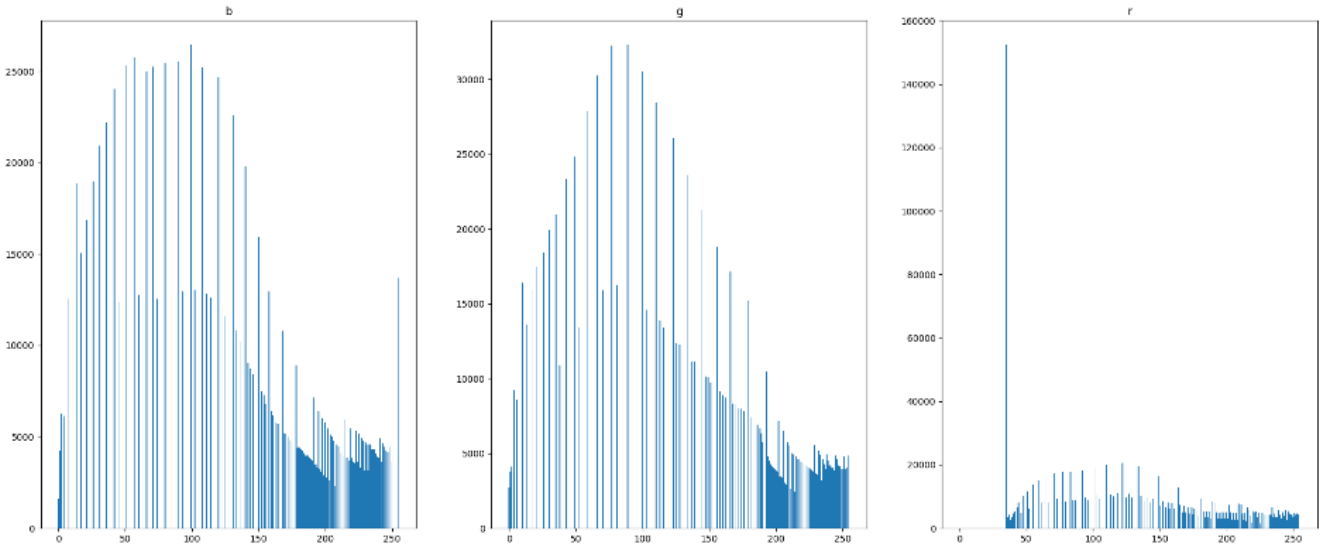


以彩色圖片的 Histogram 為例，可以觀察到在進行 Histogram Equalization 之後，RGB 的分布變得更加的均勻。

Histogram Equalization 之前:

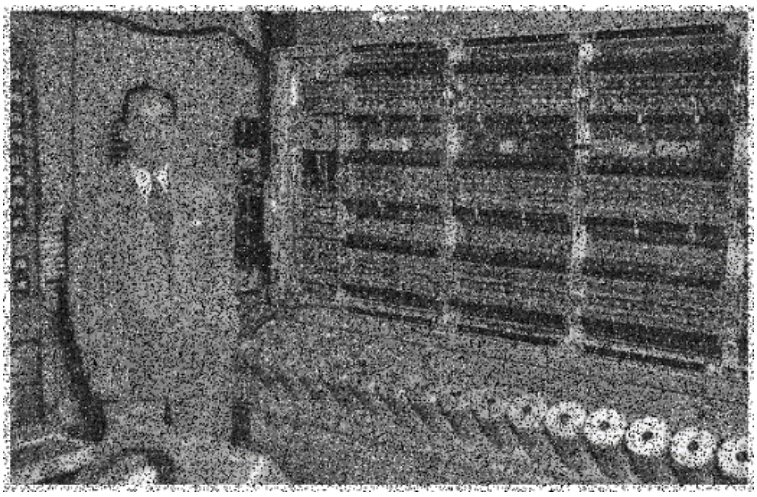
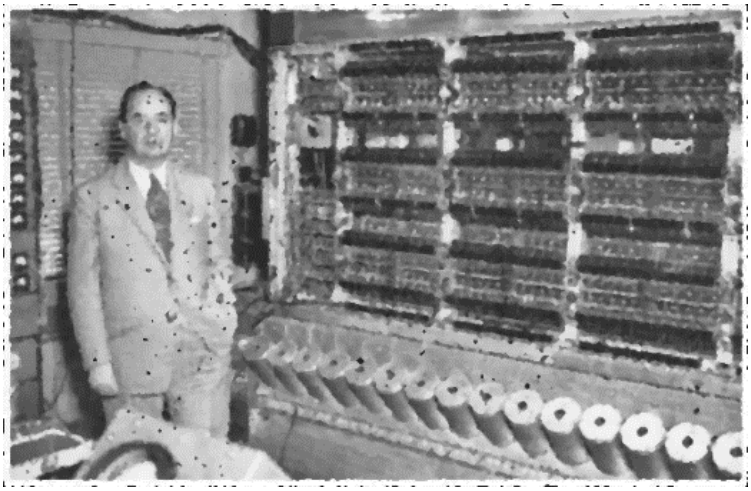


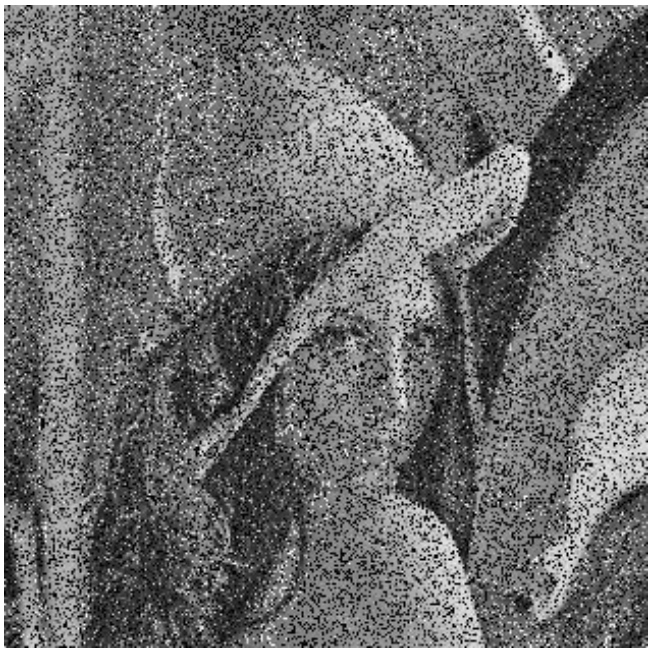
Histogram Equalization 之後:



最後進行三次 Median Filters，由於極端值被剔除，所以在黑白的圖片可以觀察到，雜訊的部分有十分顯著的改善，但是在天空的部分，不均勻的部分反而被強化了。其中重複執行三次的原因是由於第一次執行還是有可能選取到極端值，因此佛重複執行可以確保剔除到最多的極端值。下方為原圖(左)和最終結果(右)比較。



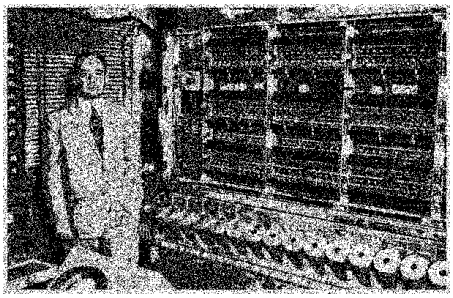



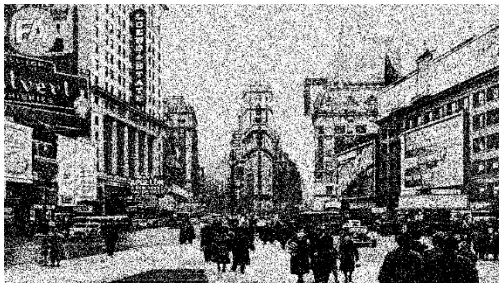


4. Discussions

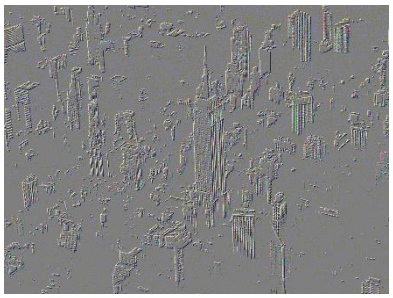
除了上方實作的三種方法證實對圖片前強化有效之外，我還有嘗試另外三種方法，但是無法以任何方式(相加、相減等)強化圖片。

第一種方法為 threshold(下圖)，可以強化明的/顏色的差異，可是同時也強化了雜訊，因此若將此結果繼續用其他方法強化，效果不增反減。

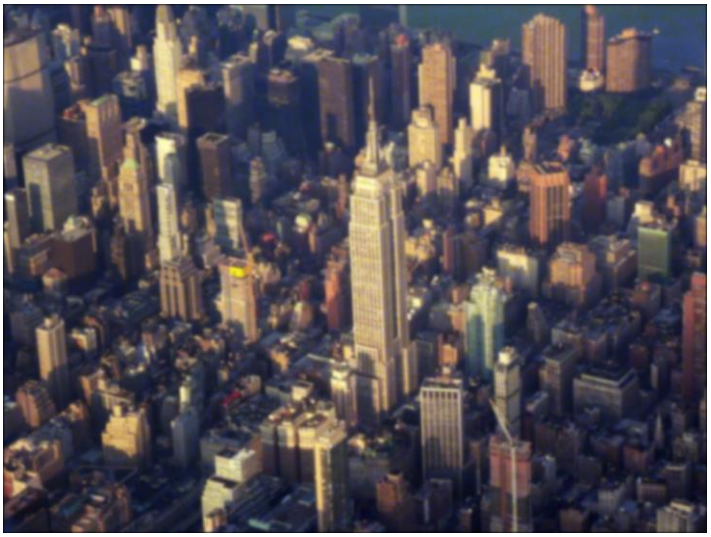



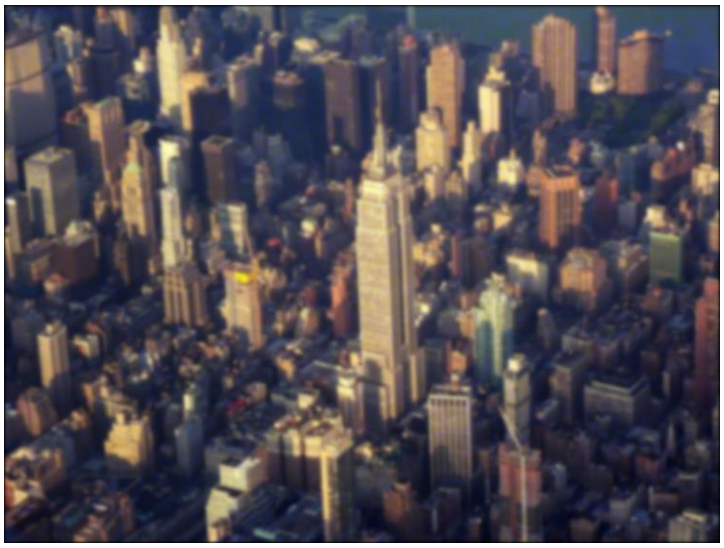


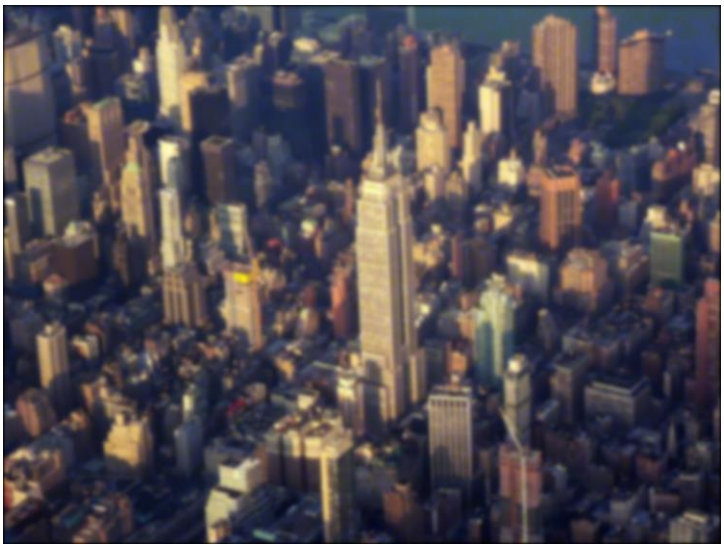
第二種方法為 Filters of Image Gradients (下圖)，雖然可以偵測邊緣，但是當此結果加入原圖後，不但邊緣被強化，非邊緣的部分也被強化了，導致圖片效果降。




第三種方法為 Gaussian Blur，此實驗的 kernel 為 5x5，下圖為進行不同次數的 Gaussian Blur，分別為 1、3、5、7、9、10(左至右, 上到下)，有成功將圖片模糊，但是模糊後不利於之後的強化，因此沒有放入主要的圖片處理中。





5. Program Code

Usage: python processing.py –-name [image name] {-g: Read image as gray}

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

def save_img(args, img, name = 'processing'):

    if(args.g):
        plt.imsave('test/'+ args.name + '_' + name +'.png', img , cmap='gray')
    else:
        plt.imsave('test/'+ args.name + '_' + name +'.png', img )


def to01(img):
    img = (img + np.abs(img.min()))
    img /= img.max()
    return img

def conv(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 get_histogram(image, bins = 256):

    histogram = np.zeros(bins)
    for v in range(256):
        histogram[v] +=  (image == v).sum()
```

```
    return histogram


def median(image, kernel_size = 3, stride = 1, padding = 1):
    image = np.pad(image, [(padding, padding), (padding, padding)],
mode='constant', constant_values=0)

    kernel_height, kernel_width = kernel_size, kernel_size
    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):
            tmp = (image[y * stride:y * stride + kernel_height, x * stride:x *
stride + kernel_width]).reshape(-1,1)
            tmp.sort(axis = 0)
            new_image[y][x] = tmp[(kernel_size*kernel_size-
1)//2].astype(np.float32)

    return new_image

def his_equl(args, img):

    if args.g:
        his = get_histogram(img)
        p = his/his.sum()
        cdf = np.cumsum(p)

        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                img[i,j] = 255 * cdf[ img[i,j]]

        save_img(args, img , 'his_equl' )

        plt.bar(np.arange(len(his)), his)
        plt.savefig('test/' + args.name + '_his_equl_before.png')

        plt.clf()
        plt.bar(np.arange(len(his)),get_histogram(img))
        plt.savefig('test/' + args.name + '_his_equl_after.png')

    else:
```

```
        his = { 'b' : get_histogram(img[:,:,0]),
                'g' : get_histogram(img[:,:,1]),
                'r' : get_histogram(img[:,:,2]) }

        cdf = { 'b' : np.cumsum(his['b']/his['b'].sum()),
                'g' : np.cumsum(his['g']/his['g'].sum()),
                'r' : np.cumsum(his['r']/his['r'].sum()) }

        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                count = 0
                for c in his.keys():
                    img[i,j,count] = 255 * cdf[c][img[i,j,count]]
                    count+=1

        save_img(args, img , 'his_equl' )

        plt.figure(figsize=(25,10))
        count = 1
        for c in his.keys():
            plt.subplot(1,3,count)
            plt.title(c)
            plt.bar(np.arange(256), his[c])
            count+=1

        plt.savefig('test/' + args.name + '_his_equl_before.png')


        plt.clf()
        his = { 'b' : get_histogram(img[:,:,0]),
                'g' : get_histogram(img[:,:,1]),
                'r' : get_histogram(img[:,:,2]) }
        count = 1
        for c in his.keys():
            plt.subplot(1,3,count)
            plt.title(c)
            plt.bar(np.arange(256), his[c])
            count+=1

        plt.savefig('test/' + args.name + '_his_equl_after.png')

    return img

def power_transformation(arg, img, gamma = 1.5, c = 1.):
    img = img.astype('float64') / 255.0
```

```
    img = c * np.power(img, gamma)
    save_img(arg, img, 'power_transformation' )

    return img

def gradients(arg, img):

    kernel_h = np.array(
        [[ -1, 0, 1],
         [ -1, 0, 1],
         [ -1, 0, 1]]
    )
    kernel_w = np.array(
        [[ -1, -1, -1],
         [  0,  0,  0],
         [  1,  1,  1]]
    )

    if arg.g:
        w = conv(img, kernel_w)
        h = conv(img, kernel_w)
        img = w + h
    else:
        w = []
        h = []
        for i in range(3):
            w.append(conv(img[:,:,i], kernel_w))
            h.append(conv(img[:,:,i], kernel_h))
        w = np.stack((w[0], w[1], w[2]), axis=2)
        h = np.stack((h[0], h[1], h[2]), axis=2)
        img = w + h
        img = to01(img)

    save_img(arg, img, 'gradients' )

    return img


def median_filter(arg, img, kernel = 3):
    if arg.g:
        img = median(img, kernel)
        save_img(arg, img, 'median_filter' )
        return img
    else:
        output = []
```

```
        for i in range(3):
            output.append(median(img[:,:,i], kernel))

        img = np.stack((output[0], output[1], output[2]), axis=2)
        save_img(arg, img/255, 'median_filter' )
        return img


def threshold(arg, img, threshold ):
    idx = img > threshold
    img[idx] = 255
    idx = img <= threshold
    img[idx] = 0

    save_img(arg, img/255, 'threshold' )
    return img

def gaussian_blur(arg, img, it):
    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.

    if arg.g:
        img = conv(img, kernel, padding = 2)
        save_img(arg, img, 'gaussian_blur_' + str(it) )
    else:
        img[:,:,0] = conv(img[:,:,0], kernel, padding = 2)
        img[:,:,1] = conv(img[:,:,1], kernel, padding = 2)
        img[:,:,2] = conv(img[:,:,2], kernel, padding = 2)

        save_img(arg, img/img.max(), 'gaussian_blur_' + str(it) )

    return img

if __name__ == '__main__':

    parser = argparse.ArgumentParser(add_help=True)

    parser.add_argument('--name', default='computer', type=str)
    parser.add_argument('-g', action="store_true")
    args = parser.parse_args()
```

```
#loading data
if(args.g):
        img = cv2.imread('data/' + args.name + '.png', cv2.IMREAD_GRAYSCALE)
else:
        img = cv2.imread('data/' + args.name + '.png')


test = img.copy()

img = power_transformation(args, img)
img = his_equl(args, (img*255).astype('uint8'))
for i in range(3):
    img = median_filter(args, img)

threshold(args, test, (test.max()+test.min())/2)
gradients(args, test)

tmp = img.copy()
for i in range(1,11):
    tmp = gaussian_blur(args, test, i)
```