彩色影像處理

作業 1：空間濾波

M10915045 施信宏

此次作業三個題目皆有實作，故會全部放上。

Include function:

```
# 引入所需要的函式
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
```

Load image:

```
origin_img = cv2.imread('C:\\Users\\User\\Desktop\\NTUST\\image_processing\\HW1\\camaraMan.png',0)
print(origin_img.shape)
```

```
In [8]: runcell(1, 'C:/Users/User/Desktop/NTUST/image_processing/HW1/Hw1-1.py')
(480, 640)
```

Matplot 畫圖:

```
#%% #畫圖
fig = plt.figure(figsize=(20,20))
ax = fig.add_subplot(3, 1, 1)
imgplot = plt.imshow(origin_img,'gray')
ax.set_title('origin')
ax.axis('off')
ax = fig.add_subplot(3, 1, 2)
imgplot = plt.imshow(relief,'gray')
ax.set_title('gx + 0.5')
ax.axis('off')
ax = fig.add_subplot(3, 1, 3)
imgplot = plt.imshow(sobel,'gray')
ax.set_title('after sobel')
ax.axis('off')
```

畫圖 code 在每個部份只有改變數及名稱。

三個作業皆用上述的 code 讀取、畫圖，後面不再貼上。

第一題:

先將 img 除上 255 ，使其介於[0 , 1]

```
#%%
origin_img = origin_img / 255
```

Sobel filter:

```
#%%
# 0 for horizontal derivative
# 1 for vertical derivative
#vertical= np.array([[-1,-2,-1],[0,0,0],[1,2,1]]) Gy
#horizontal = np.array([[-1,0,1],[-2,0,2],[-1,0,1]]) Gx
def sobel_filter(img , axis):
    if axis == 0:
        kernel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
        res = cv2.filter2D(img,-1,kernel)
    elif axis == 1:
        kernel =  np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
        res = cv2.filter2D(img,-1,kernel)
    return res
```

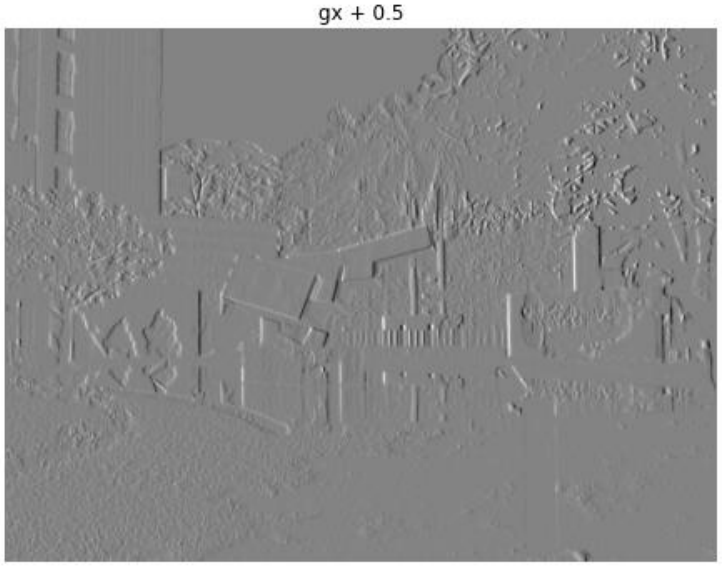
執行 sobel，得到 gx ,gy 最後取絕對值做 bitor

```
#%%
gx = sobel_filter(origin_img,0) # horizontal
gy = sobel_filter(origin_img,1) # vertical
relief = gx + 0.5
np.absolute(gx) # get abs
np.absolute(gy)
# set threshold = 0.25
_ ,thresgx = cv2.threshold(gx,0.25,1,cv2.THRESH_BINARY)
_ ,thresgy = cv2.threshold(gy,0.25,1,cv2.THRESH_BINARY)
# result mix gx and gy
sobel = cv2.bitwise_or(thresgx,thresgy)
```

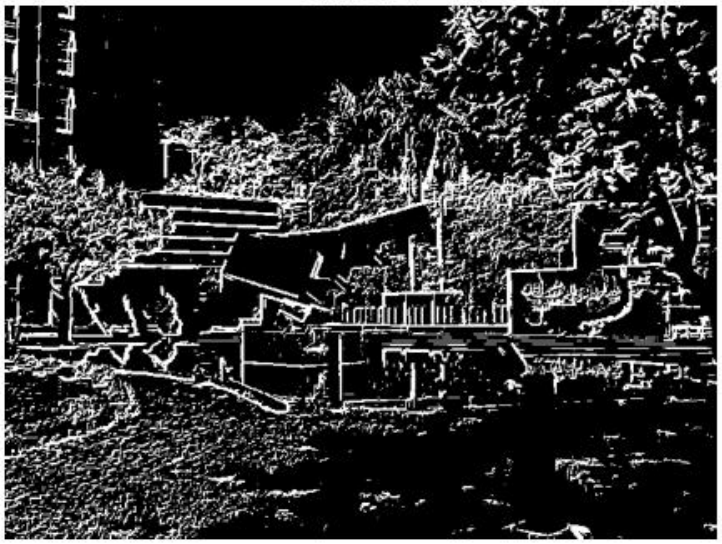
Result:

origin

gx + 0.5



after sobel



第二題:

先實作出 mean_filter，這裡規定要使用雙重迴圈，所以要考慮捲積問題。但此處不用考慮 padding,

Strides 這兩項變數。

捲積後大小公式：(W — F + 1) / S

W = input shape F = filter size S = 1

所以運算時要先使用另一個 matrix，並且大小為 (input shape + filter_size -1) * (input shape + filter_size -1)，再將 input image 放入，剩餘地方補上 0，即 zero-padding，最後得到結果即為原先 input 的大小，才能避免捲積後大小不同問題。

```python
def mean_filter(img,kernel_size):
    #定義kernel 此處使用 mean filter
    kernel = np.ones([kernel_size , kernel_size]) / (kernel_size * kernel_size)
    print(kernel)
    # 先 padding 因為 convolution後size會縮小
    convolve_img = np.zeros([img.shape[0] + kernel_size - 1, img.shape[1] + kernel_size - 1])
    convolve_img[:img.shape[0],:img.shape[1]] = img # 將原圖放入要進行捲積的array
    print(convolve_img.shape)
    res_img = np.zeros([img.shape[0],img.shape[1]])
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            conv_arr =  convolve_img[row : row + kernel_size , col : col + kernel_size]
            res_img[row][col] = np.sum(conv_arr * kernel)
    print(res_img.shape)
    return res_img
```

實作 UAM:

此處使用 3*3 mean filter，再按照講義公式。

```python
#%%
mean_img = mean_filter(origin_img,3)
# USM = 0.8 * (a-b) + a
UM_img = 0.8 * (origin_img - mean_img ) + origin_img
```

Result:

在此圖並不明顯，但使用 camaraMan 時能明顯看出。

origin

after mean_filter

Unsharp Masking

第三題:

將原圖加上噪點:

```
noise_num = int(origin_img.shape[0] * origin_img.shape[1] * 0.15) # 總pixel的15%
noise_img = origin_img.copy()

for loop in range(noise_num):
    row = int(np.random.uniform(0, origin_img.shape[0])) # random choose
    col = int(np.random.uniform(0, origin_img.shape[1]))
    noise_img[row,col] = 255 # 將選到的位置設為255
```

Median_filter:

Convolution 問題上題已討論。

```
#new_height = new_width = (W – F + 1) / S
# 有padding : floor[(W - F + 2 * P + 1) / S] + 1

def median_filter(img,kernel_size):
    # 先 padding 因為 convolution後size會縮小
    convolve_img = np.zeros([img.shape[0] + kernel_size - 1, img.shape[1] + kernel_size - 1])
    # 因kernel 為 3*3
    convolve_img[:img.shape[0],:img.shape[1]] = img
    print(convolve_img.shape)
    res_img = np.zeros([img.shape[0],img.shape[1]])
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            res_img[row][col] = np.median(convolve_img[row:row + kernel_size,col:col + kernel_size])

    return res_img
```

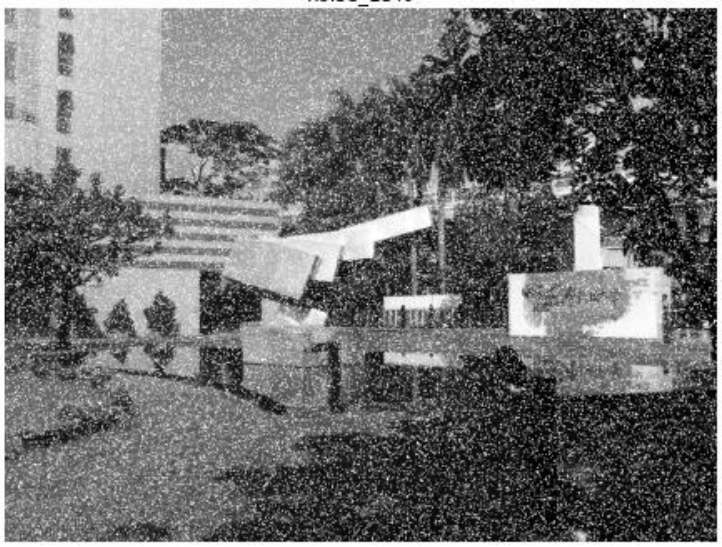
將有噪點的圖放入 median_filter:

```
filter_img = noise_img.copy()
res_img = median_filter(filter_img,3)
print(res_img.shape)
```

Result:

origin

noise_15%



median_filter