HW2_影像增強 Image Enhancement

一、 影像增強-銳化

銳化是一種常用的影像增強技術，可以增強影像中的細節與邊緣，銳化可以通過對影像進行二階微分(laplacian_mask)取得銳化結果，laplacian_mask 專門將細節放大，但細節會包含雜訊，簡單來說，若使用 laplacian_mask 來將圖片銳化是沒問題的，但雜訊也會被放大，因此若我們只希望將細節銳化即可，不要影響雜訊的話，可以再加上一階微分(sobel_filter)，將 sobel_filter 當作遮罩配合卷積運算的話就能解決雜訊也被放大的問題以此來提升影像的銳利度，既達到銳化邊緣，又不放大雜訊。

二、流程圖

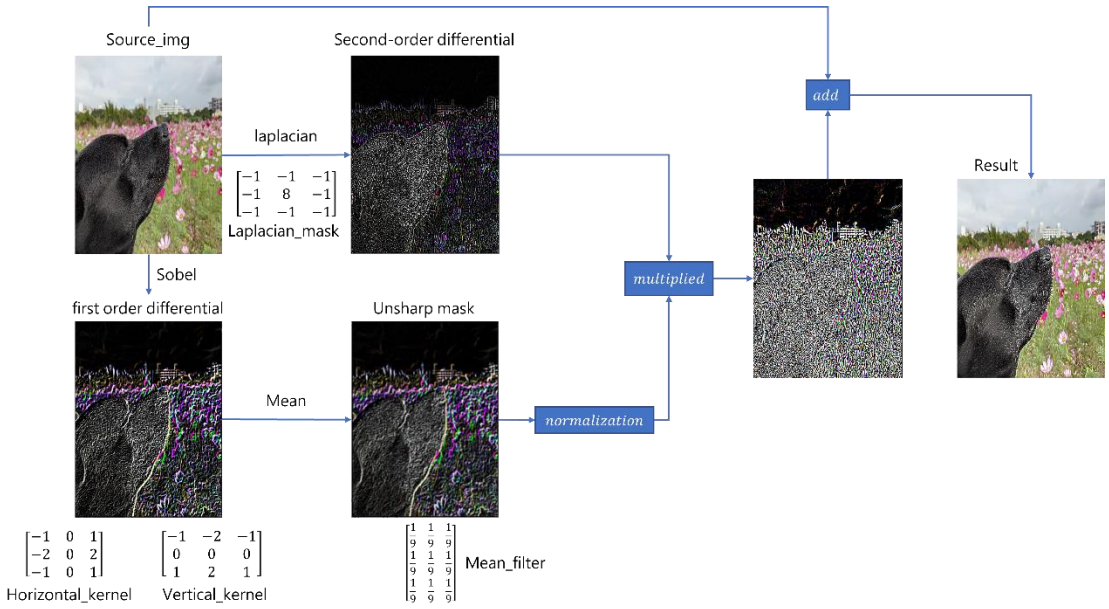

圖 一　　流程圖

1. 對原影像做 Sobel Operator，得到一階微分過後的影像
2. 將一階微分過後的影像做平均濾波器(Mean Filter)，做完後將結果正規化至 0~1 之間，得到我們的 Unsharp mask
3. 對原影像做 Laplace operator，得到二階微分過後的影像，即是從原影像過濾出的細節
4. 將一階微分經過平均濾波器及正規化後的結果與二階微分的結果做相乘的動作，得到實際需要增強的數值；即是將一階微分經過平均濾波器及正規化後的結果當作 mask 把二階微分的結果非影像邊緣的數值過濾掉
5. 將實際需要增強的數值以及原影像相加，即完成了銳化邊緣且不放大雜訊的銳化結果。

name	Kernel
Sobel_filter (Horizontal_kernel)	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
Sobel_filter (Vertical_kernel)	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Laplacian_mask	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
Mean_filter	$\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$

圖 二　　kernel

三、程式實作

首先使用 cv2.imread 讀取圖片，取名為 img，並使用 resize 將圖片縮小成原本的 0.1，以此增加運算速度。

```
img = cv2.imread('test3.jpg')
img = cv2.resize(img, (int(img.shape[0]*0.1), int(img.shape[1]*0.1)))
```

圖 三　讀取圖片並 resize

第一階段，先將 img 傳入 sobel_operator 函數做計算，sobel_operator 函數內會創造兩個 kernel，一個是 Horizontal_kernel→$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$，此 kernel 專門計算水平的邊緣，另一個則是專門計算垂直邊緣的 kernel：Vertical_kernel →$\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix}$，將這兩個 kernel 創造出來後，各自傳入 Convolution 函數計算。

```
sobel_img = sobel_operator(img.copy()) #一階微分後結果
```

圖 四　將圖片傳入 sobel_operator 函數

```
def sobel_operator(img):
    #定義計算水平和垂直邊緣的kernel
    horizontal_kernel = np.array([
                        [-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]],np.float32)

    vertical_kernel = np.array([
                        [-1, -2, -1],
                        [0, 0, 0],
                        [1, 2, 1]],np.float32)
    #計算水平和垂直邊緣
    horizontal_edge , vertical_edge = Convolution(img,horizontal_kernel),Convolution(img,vertical_kernel)
```

圖 五　創建 kernel 並傳入 Convolution

Convolution 函數會先讀取圖片的高度及高度、kernel 的 row 及 col 以及計算 kernel 中間的位置為 kernel_center_x 及 kernel_center_y，之後使用 np.zeros 創建一個與傳進來的圖片大小一樣的全黑圖片。之後跑四層迴圈，第

一個迴圈從 kernel_center_x 開始，一直到圖片的高度-kernel_center_x，第二層迴圈從 kernel_center_y 開始至圖片的寬度-kernel_center_y，此時建立一個計算總合的變數 conv_sum，之後兩層迴圈即是將 kernel 的 row 及 kernel 的 col 跑一輪，迴圈內會計算每一個像素點做 Convolution 後的值並加總給 conv_sum，待 kernel 跑完後，會先判斷 conv_sum 內三個值是否為超過 255，若超過則改值為 255，三個值的原因是因為圖片為彩色的，有三個通道。計算出三個通道的值後，將其像素值傳回去剛建立之全黑圖片的該座標點，全部跑完後即為 Convolution 後的圖片。

```
def Convolution(img , kernel):
    img_height = img.shape[0]
    img_width = img.shape[1]
    kernel_rows = kernel.shape[0]
    kernel_cols = kernel.shape[1]

    kernel_center_x = kernel_rows // 2
    kernel_center_y = kernel_cols // 2
    result = np.zeros((img_height, img_width,3), dtype=img.dtype)

    for i_h in range(kernel_center_x,img_height-kernel_center_x):
        for i_w in range(kernel_center_y,img_width-kernel_center_y):
            conv_sum = 0
            for k_h in range(kernel_rows):
                for k_w in range(kernel_cols):
                    #計算kernel在img上的對應位置
                    i_k_h = i_h + k_h - kernel_center_x
                    i_k_w = i_w + k_w - kernel_center_y
                    conv_sum += (img[i_k_h][i_k_w]) * (kernel[k_h][k_w])
            for i in range(len(conv_sum)):
                conv_sum[i] = min(255,max(0,conv_sum[i]))
            result[i_h][i_w] = conv_sum
    return result
```

圖 六　Convolution 函數內容

待水平及垂直邊緣計算結束後，我們會先建立一個與原圖大小相同的全為 0 的圖片，之後跑三個迴圈，前兩個迴圈負責跑圖片的長寬，第三個迴圈負責

彩色通道，之後我們將水平計算過後的圖片及垂直計算過後的圖片的每個像素值做絕對值後相加，若超過 255 則使其像素值為 255，並將該像素值回傳給我們剛剛所建立的全黑圖片，待迴圈跑完後即能取得第一張一階微分的圖片。

```
#輸出圖片
sobel_img = np.zeros((img.shape[0],img.shape[1],3),dtype='uint8')

#計算水平和垂直相加後輸出的結果
for i in range(horizontal_edge.shape[0]):
    for j in range(vertical_edge.shape[1]):
        for k in range(3):
            result = abs(int(horizontal_edge[i][j][k])) + abs(int(vertical_edge[i][j][k]))
            sobel_img[i][j][k] = result if result <256 else 255
return sobel_img
```

圖 七　　製作一階微分過後的結果圖片

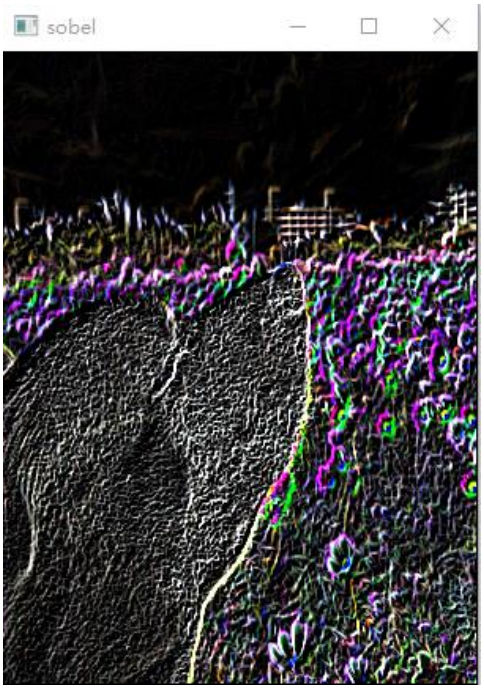

圖 八　　一階微分過後的結果圖

一階微分的圖片完成後，即能透過 Mean_filter 將圖片做模糊化的動作，首先建立一個 Mean_filter，再將 Mean_filter 及一階微分過後的圖片傳入 Convolution 函數內做計算，得出的結果即為模糊圖片。

```
blur_img = blur(sobel_img.copy())
```

圖 九　　將一階微分過後的圖片傳入 blur 函數

```
def blur(img):
    result = np.zeros((img.shape[0],img.shape[1],3),dtype=np.float32)
    Mean_filter = np.array([
                [1/9, 1/9, 1/9],
                [1/9, 1/9, 1/9],
                [1/9, 1/9, 1/9]],np.float32)
    blur = Convolution(img, Mean_filter)
    result = output_img(img,blur)
    return result
```

圖 十　　將傳入的圖片做模糊化

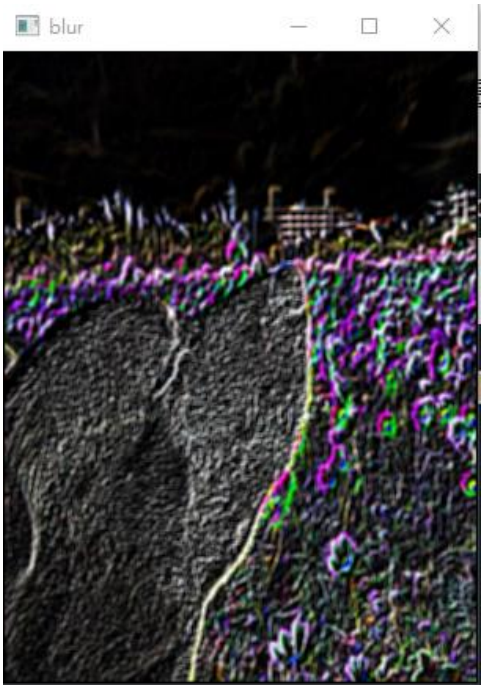

圖 十一　模糊化圖片

得到模糊化的一階微分圖片後，即可將該圖片傳入 blur_normalize 函數內，將此圖片所有的像素值正規化至 0~1 之間，正規化的算式為

$$\frac{像素值-圖片最小像素值}{圖片最大像素值-圖片最小像素值}$$

```
blur_normalize = blur_normalize(blur_img)
```

圖 十二　將模糊圖片傳入 blur_normalize 函數

```
def blur_normalize(img):
    img_min = np.amin(img)
    img_max = np.amax(img)
    normalize = np.zeros((img.shape[0],img.shape[1],3),dtype=np.float32)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            for k in range(3):
                normalize[i][j][k] = float((img[i][j][k]-img_min) / (img_max-img_min))
    return normalize
```

圖 十三　圖片做正規化

一階微分模糊後做正規化的圖片完成後，即可再使用原圖片做二階微分，使用 laplacian_mask 再加上 Convolution 即可完成。

```
laplacian_img = laplacian(img.copy())
```

圖 十四　原圖片傳入 laplacian 函數

```
def laplacian(img):
    result = np.zeros((img.shape[0],img.shape[1],3),dtype='uint8')
    laplacian_mask =np.array([
        [-1,-1,-1],
        [-1,8,-1],
        [-1,-1,-1]],np.float32)
    laplacian = Convolution(img,laplacian_mask)
    result = output_img(img,laplacian)
    return result
```

圖 十五　laplacian 函數內容

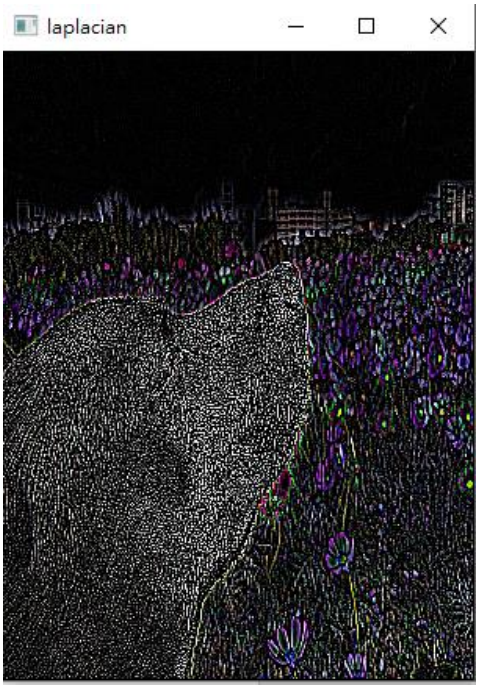

圖 十六　二階微分結果

計算出一階微分模糊後正規化的圖片以及二階微分結果的圖片過後，即可

將此兩張圖片傳入 multiple 函數將兩張圖片相乘，一樣是使用三個迴圈，前兩個迴圈為圖片的長與寬，第三個迴圈為通道數 3。

```
multiple_img= multiple(blur_normalize,laplacian_img)
```

圖 十七　將兩張圖片傳入 multiple 函數

```
def multiple(blur_nor,img2):
    result = np.zeros((img2.shape[0],img2.shape[1],3),dtype=np.float32)
    for i in range(img2.shape[0]):
        for j in range(img2.shape[1]):
            for k in range(3):
                result[i][j][k] = float(blur_nor[i][j][k] * img2[i][j][k])
    return result
```

圖 十八　將兩張圖片的像素值做相乘

圖 十九　相乘後結果

得到相乘的結果後，即可將此結果與原圖傳入 add 函數內做相加，若相加後像素值>255 則將此像素值改為 255，即能取得最終的銳化結果。

```
result = add(img.copy(),multiple_img)
```

圖 二十　將圖片傳入 add 函數

```python
def add(org_img , mul_img):
    result = np.zeros((org_img.shape[0],org_img.shape[1],3),dtype = org_img.dtype)
    for i in range(org_img.shape[0]):
        for j in range(org_img.shape[1]):
            for k in range(3):
                result[i][j][k] = org_img[i][j][k]+mul_img[i][j][k] if org_img[i][j][k]+mul_img[i][j][k] < 256 else 255
    return result
```

圖 二十一 add 函數內容

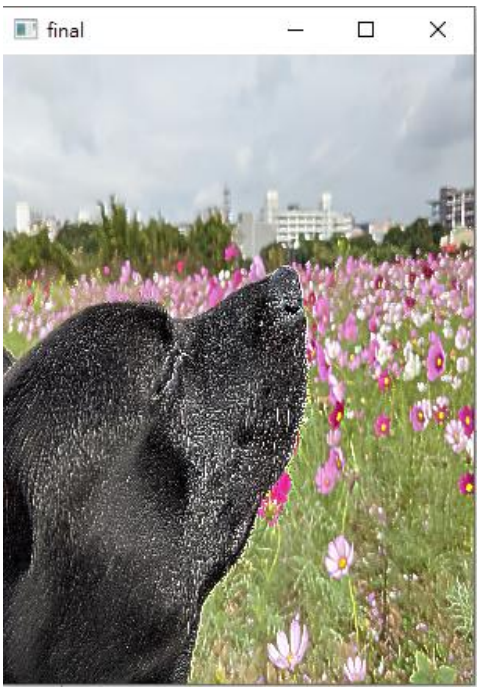

圖 二十二 銳化結果

完整程式碼如下：

```python
import cv2
import numpy as np

def output_img(img,conv):
    result = np.zeros((img.shape[0],img.shape[1],3),dtype=conv.dtype)
    for i in range(conv.shape[0]):
        for j in range(conv.shape[1]):
            for k in range(3):
                result[i][j][k] = int(conv[i][j][k]) if int(conv[i][j][k])<256 else 255
    return result

def Convolution(img , kernel):
    img_height = img.shape[0]
    img_width = img.shape[1]
    kernel_rows = kernel.shape[0]
    kernel_cols = kernel.shape[1]

    kernel_center_x = kernel_rows // 2
    kernel_center_y = kernel_cols // 2
    result = np.zeros((img_height, img_width,3), dtype=img.dtype)

    for i_h in range(kernel_center_x,img_height-kernel_center_x):
        for i_w in range(kernel_center_y,img_width-kernel_center_y):
            conv_sum = 0
            for k_h in range(kernel_rows):
                for k_w in range(kernel_cols):
                    #計算kernel在img上的對應位置
                    i_k_h = i_h + k_h - kernel_center_x
                    i_k_w = i_w + k_w - kernel_center_y
                    conv_sum += (img[i_k_h][i_k_w]) * (kernel[k_h][k_w])
            for i in range(len(conv_sum)):
                conv_sum[i] = min(255,max(0,conv_sum[i]))
            result[i_h][i_w] = conv_sum
    return result

def sobel_operator(img):
    #定義計算水平和垂直邊緣的kernel
    horizontal_kernel = np.array([
                        [-1, 0, 1],
                        [-2, 0, 2],
                        [-1, 0, 1]],np.float32)

    vertical_kernel = np.array([
                        [-1, -2, -1],
                        [0, 0, 0],
                        [1, 2, 1]],np.float32)
    #計算水平和垂直邊緣
    horizontal_edge , vertical_edge = Convolution(img,horizontal_kernel),Convolution(img,vertical_kernel)

    #輸出圖片
    sobel_img = np.zeros((img.shape[0],img.shape[1],3),dtype='uint8')

    #計算水平和垂直相加後輸出的結果
    for i in range(horizontal_edge.shape[0]):
        for j in range(vertical_edge.shape[1]):
            for k in range(3):
                result = abs(int(horizontal_edge[i][j][k])) + abs(int(vertical_edge[i][j][k]))
                sobel_img[i][j][k] = result if result <256 else 255
    return sobel_img
```

```python
def blur(img):
    result = np.zeros((img.shape[0],img.shape[1],3),dtype=np.float32)
    Mean_filter = np.array([
              [1/9, 1/9, 1/9],
              [1/9, 1/9, 1/9],
              [1/9, 1/9, 1/9]],np.float32)
    blur = Convolution(img, Mean_filter)
    result = output_img(img,blur)
    return result

def laplacian(img):
    result = np.zeros((img.shape[0],img.shape[1],3),dtype='uint8')
    laplacian_mask =np.array([
        [-1,-1,-1],
        [-1,8,-1],
        [-1,-1,-1]],np.float32)
    laplacian = Convolution(img,laplacian_mask)
    result = output_img(img,laplacian)
    return result

def blur_normalize(img):
    img_min = np.amin(img)
    img_max = np.amax(img)
    normalize = np.zeros((img.shape[0],img.shape[1],3),dtype=np.float32)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            for k in range(3):
                normalize[i][j][k] = float((img[i][j][k]-img_min) / (img_max-img_min))
    return normalize

def multiple(blur_nor,img2):
    result = np.zeros((img2.shape[0],img2.shape[1],3),dtype=np.float32)
    for i in range(img2.shape[0]):
        for j in range(img2.shape[1]):
            for k in range(3):
                result[i][j][k] = float(blur_nor[i][j][k] * img2[i][j][k])
    return result

def add(org_img , mul_img):
    result = np.zeros((org_img.shape[0],org_img.shape[1],3),dtype = org_img.dtype)
    for i in range(org_img.shape[0]):
        for j in range(org_img.shape[1]):
            for k in range(3):
                result[i][j][k] = org_img[i][j][k]+mul_img[i][j][k] if org_img[i][j][k]+mul_img[i][j][k] < 256 else 255
    return result
img = cv2.imread('test3.jpg')
img = cv2.resize(img, (int(img.shape[0]*0.1), int(img.shape[1]*0.1)))
sobel_img = sobel_operator(img.copy()) #一階微分後結果
blur_img = blur(sobel_img.copy())
blur_normalize = blur_normalize(blur_img)
laplacian_img = laplacian(img.copy())
multiple_img= multiple(blur_normalize,laplacian_img)
result = add(img.copy(),multiple_img)

cv2.imshow('img',img)
cv2.imshow('sobel',sobel_img)
cv2.imshow('blur',blur_img)
cv2.imshow('laplacian',laplacian_img)
cv2.imshow('mul',multiple_img)
cv2.imshow('final',result)
cv2.waitKey(0)
```

圖 二十三　　最終程式碼

四、心得

此次作業其實不難，就是 Convolution 需要想一下該如何實作，以及 DataType 該如何設置，剩下的就是超過 255 該如何處理，以及搜尋一下一階微分和二階微分的 kernel 長什麼樣子即可！