Color Image Processing　HW1　何柏昇　M11107309

第一題 影像邊緣偵測

1. 讀取附件的 8-bit 灰階影像。
2. 顯示輸入影像。
3. 將影像轉換成 double 格式，數值範圍在[0 1]之間。
4. 用雙層迴圈由左而右，由上而下讀取以(x, y)為中心的 3*3 影像區域。
5. 將 3*3 影像區域點對點乘上 Sobel 濾鏡數值矩陣後，將數值總和存入輸出影像的(x, y)位置。
6. 將濾波後的影像加上 0.5，呈現浮雕影像。
7. 分別將濾波後的影像開絕對值，再二值化(門檻值自訂)，用 bitor(bitwise or)或直接相加，產生輪廓影像。
8. 轉成 8bit，儲存影像檔。

Sobel 濾鏡

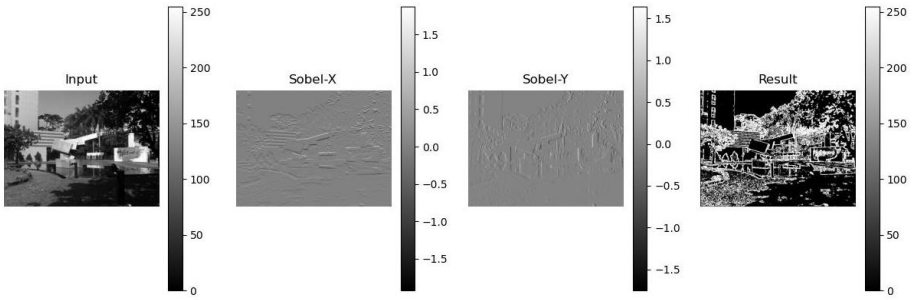
水平濾波

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

垂直濾波

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

Result



Code

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import os

# Read image
img = cv.imread("ntust_gray.jpg",0)
# Show information of image
print(img.dtype)
print(img.shape)

# Turn to double type, and range in [0,1]
img = img.astype(float) / 255

# Sobel kernel
Gx = np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])
Gy = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
```

首先使用 OpenCV 來讀取影像，於函式中輸入影像位置，以及 color type：0 為 gray、1 為 RGB。

接著把影像由 int 轉為 double 型態，並且縮放到[0 1]，之後計算時會有比較精準的結果。

最後利用 Numpy 建立 Sobel 運算子，為水平與垂直偵測兩種，Mask 尺寸為 3*3。

```
# Convolution
rows, colums=img.shape
tmpx = np.zeros(img.shape)
tmpy = np.zeros(img.shape)
for row in range(rows):
    for colum in range(colums):
        if ((row-1 > 0)&(colum-1 > 0)&(row+1 < rows)&(colum+1 < colums)):
            tmpx[row, colum] = np.sum(np.multiply(img[row-1:row+2,colum-
1:colum+2],Gx))*0.5
            tmpy[row, colum] = np.sum(np.multiply(img[row-1:row+2,colum-
1:colum+2],Gy))*0.5
```

利用兩層迴圈將每個 pixel 掃過，進行卷積運算，由於使用的 kernel size 為 3*3，當遇到處理影像最外圍會無法計算，最後再利用 if 判斷邊緣像素。

為了加速運算過程，採取 Numpy 的 multiply，將兩個矩陣進行元素相乘，最後透過 sum 算出矩陣內的元素總和，並將結果呈上 0.5 進行 scaling。

```
# Absolute
result = abs(tmpx) + abs(tmpy)
# Binarization
thresh = np.mean(result)
maxval = 255
result = (result > thresh) * maxval
```

將經過 Sobel 運算後的結果，先取絕對值後，再進行二值化計算。取代使用雙層迴圈，改用回傳布林值的方式，就可加速計算過程，並將結果 scaling 到[0 255]。

```
# Create folder to save image
if not os.path.exists('images'):
    os.makedirs('images')

# Show all images
imgs = [img*maxval, tmpx, tmpy, result]
titles = ['Input', 'Sobel-X', 'Sobel-Y', 'Result']
fig = plt.figure()
fig.set_figwidth(15)
for i in range(4):
    # Save image
    #cv.imwrite('images/'+titles[i]+'.jpg', imgs[i])
    # Plot image
    plt.subplot(1,4,i+1)
    plt.imshow(imgs[i], 'gray')
    plt.title(titles[i])
    plt.axis('off')
    plt.colorbar()
print("Save the image of result to /images/ \n")
plt.savefig('images/All-Result-Sobel.jpg')
plt.show()
```

最後以 pyplot 顯示結果，為了顯示方便，採用 subplot 的形式，將計算過程的每張圖一起顯示，並加入 color bar，方便觀察數值變化，並將結果存入 images 資料夾中。

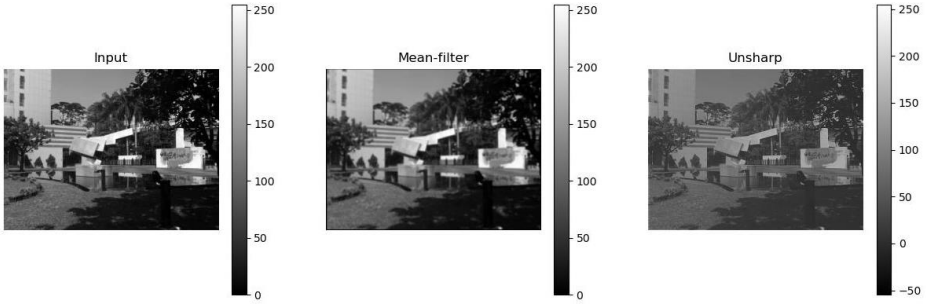
第二題 Unsharp Masking(USM)影像銳化

1. 輸入影像模糊參數（例如均值濾波的濾鏡尺寸 n）。
2. 讀取附件的 8-bit 灰階影像。
3. 顯示輸入影像。
4. 將影像轉換成 double 格式，數值範圍在[0 1]之間。
5. 用雙層迴圈對 n*n 濾鏡（均值濾鏡或高斯濾鏡）做影像模糊化，獲得模糊影像。
6. 利用原圖與模糊影像的差異，加上原圖，獲得銳利影像。

n*n 均值濾波器

$$\frac{1}{n} * \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

Result



Code

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
import os

# Read image
img = cv.imread("ntust_gray.jpg",0)
# Show information of image
print(img.dtype)
print(img.shape)

# Turn to double type, and range in [0,1]
img = img.astype(float) / 255
```

首先使用 OpenCV 來讀取影像，於函式中輸入影像位置，以及 color type：0 為 gray、1 為 RGB。

接著把影像由 int 轉為 double 型態，並且縮放到[0 1]，之後計算時會有比較精準的結果。

```
# Mean filter
def get_input(s):
    while True:
        try:
            n = int(input('Enter %s: ' % s))
        except ValueError:
            print('Error: Invalid Input.')
        if n%2 == 0 :
            raise ValueError('n=%d is a even value!' % n)
        return n
# Enter the n of filter
n = get_input('n of filter (odd)')
filter = np.ones([n,n])/(n**2)
```

建立一個 n*n 的 Mean Filter，由於 kernel size 與卷積的特性，這邊要求輸入 n 必須為奇數，並在輸入過程中檢查是否符合輸入要求。

在得到 n 之後，利用 Numpy 提供的 ones 快速建立一個 n*n 的矩陣，元素皆為 1，並將矩陣除以 n*n，就可以得到一個 Mean Filter。

```
# Convolution
rows, colums=img.shape
tmp = np.zeros(img.shape)
k=int((n-1)/2)
for row in range(rows):
    for colum in range(colums):
        if ((row-k > 0)&(colum-k > 0)&(row+k < rows)&(colum+k < colums)):
            tmp[row, colum] = np.sum(np.multiply(img[row-k:row+k+1,colum-
k:colum+k+1],filter))

# Unsharp Masking 0.8*(a-b)+a
result = 0.8*(img-tmp)+img
result = result*255/np.max(result)
```

利用兩層迴圈將每個 pixel 掃過，進行卷積運算，由於使用的 kernel size 為不一定，所以必須修改 if 的判斷。由於 n 為奇數，所以可改寫為$n = 2 * k + 1$，透過 k 就能知道濾鏡是幾層，再透過 if 就可以處理邊緣像素。

為了加速運算過程，在矩陣的元素相乘的步驟，採取 Numpy 的 multiply，將兩個矩陣進行元素相乘，最後透過 sum 算出矩陣內的元素總和。並將結果與原圖相減，得到圖像的輪廓差異，並加回原圖，即可銳化影像。

```
# Create folder
if not os.path.exists('images'):
    os.makedirs('images')

# Show all images
imgs = [img*255, tmp*255, result]
titles = ['Input', 'Mean-filter', 'Unsharp']
fig = plt.figure()
fig.set_figwidth(15)
for i in range(3):
    # Plot image
    plt.subplot(1,3,i+1)
    plt.imshow(imgs[i], 'gray')
    plt.title(titles[i])
    plt.axis('off')
    plt.colorbar()
print("Save the image of result to /images/ \n")
plt.savefig('images/All-Result-Unsharp.jpg')
plt.show()
```

最後以 pyplot 顯示結果，為了顯示方便，採用 subplot 的形式，將計算過程的每張圖一起顯示，並加入 color bar，方便觀察數值變化，並將結果存入 images 資料夾中。