國立中興大學

一一一學年度第一學期

數位影像處理

第二次平時作業

班級：資工碩士一年級
學號：7110056210
學生：丁吾心
授課教師：吳俊霖 教授

目錄

影像銳化 Sharpening

1.1 題目說明

影像銳化的動作是指在原始影像中，將我們想增強的一些特徵增強，更凸顯該區域的細節，讓模糊的影像更清晰。

首先對原圖做一階微分，因數位影像為不連續的區間，所以一階微分使用前項減後項的方式，可以取得影像中的邊界，接著對一階微分結果用平均濾波器去雜訊(模糊)，再正規化到 0~1 的區間。

第二步是將原影像做二階微分，此步驟可以取得影像中的細節，但可能會有雜訊也夾雜在裡面。所以接著乘上一階微分的影像，可以做出想強化的部分。

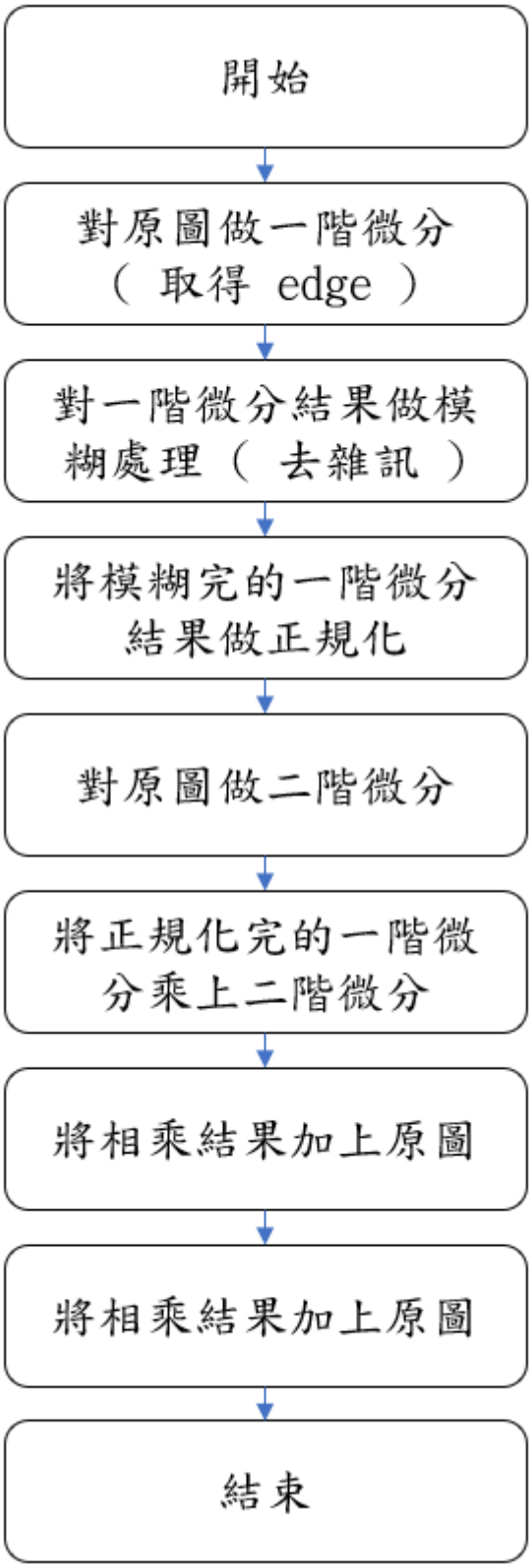
最後將第二步產生的圖與原圖相加即可增強銳化的部分。

外流講義示意圖


3. (0)一階微分(找Edge)
4. (3)模糊後結果(去雜訊)
5. 將(4)正規化到0.0~1.0，乘上(2)
將(5)加上原始影像(0)（無雜訊了）
將(4)正規化到0.0~1.0
乘上(1)，加上(0)
[和課本步驟不一樣]

外流講義示意圖

1.2 演算法流程


開始
對原圖做一階微分
（ 取得 edge ）
對一階微分結果做模
糊處理（ 去雜訊 ）
將模糊完的一階微分
結果做正規化
對原圖做二階微分
將正規化完的一階微
分乘上二階微分
將相乘結果加上原圖
將相乘結果加上原圖
結束

1.3 詳細流程與說明

1.3.1 整體流程

```
sobel_img = Sobel( ori_img )                          # 一階微分
s_img = convolution( sobel_img, m_mask )              # 模糊（去雜訊）
La_img = convolution( ori_img, l_mask )               # 二階微分
n_img = s_img / ( np.amax( s_img ) - np.amin( s_img ) ) # 正規化 0 ~ 1 之間
multi_img = np.multiply( n_img, La_img )              # 一階微分模糊完乘上二階微分
final_img = ori_img + multi_img                       # 最後再加上原圖
```

1.3.2 一階微分

```
def Sobel( ori_img ):
    ret_sobel = np.zeros( ( ori_img.shape[ 0 ], ori_img.shape[ 1 ], 3 ), np.float64 )
    sobel_x = np.array( [ [ -1, -2, -1 ], [ 0, 0, 0 ], [ 1, 2, 1 ] ] )
    sobel_y = np.array( [ [ -1, 0, 1 ], [ -2, 0, 2 ], [ -1, 0, 1 ] ] )
    for i in range( 1, ori_img.shape[ 0 ] - 1 ):
        for j in range( 1, ori_img.shape[ 1 ] - 1 ):
            part_img = ori_img[i-1:i+2, j-1:j+2, :]
            for c in range( 3 ):
                value = abs( np.sum( part_img[ :, :, c] * sobel_x ) ) + abs( np.sum( part_img[ :, :, c] * sobel_y ) )
                value = Boundary( value )
                ret_sobel[ i, j, c ] = value
    return  ret_sobel
```

$$|(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$$

1.3.3 去雜訊以及二階微分

```
def convolution( ori_img, mask ):
    ret_img = np.zeros( ( ori_img.shape[ 0 ], ori_img.shape[ 1 ], 3 ), np.float64 )
    for i in range( 1, ori_img.shape[ 0 ] - 1 ):
        for j in range( 1, ori_img.shape[ 1 ] - 1 ):
            part_img = ori_img[i-1:i+2, j-1:j+2, :]
            for c in range( 3 ):
                value = np.sum( part_img[ :, :, c] * mask )
                value = Boundary( value )
                ret_img[ i, j, c ] = value
    return ret_img
```

```
l_mask = np.array([ [ -1, -1, -1 ], [ -1, 8, -1 ], [ -1, -1, -1 ] ])
m_mask = np.array([ [ 1, 1, 1 ], [ 1, 1, 1 ], [ 1, 1, 1 ] ] )
```

將不同的 mask 代入 function 裡面分別做去雜訊以及二階微分

1.3.4 結果

最後將一階微分模糊完且正規化的結果與二階微分相乘，再加上原圖。

```
multi_img = np.multiply( n_img, La_img )              # 一階微分模糊完乘上二
final_img = ori_img + multi_img                       # 最後再加上原圖
```

1.4 結果圖

原圖

一階微分

8

一階微分去雜訊

二階微分

一階二階微分相乘

原圖

結果圖

可以觀察到頭髮以及睫毛部分有明顯的銳化效果

1.5 程式碼

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

def Boundary( value ):   # 讓值在 0 ~ 255
    ret_value = value.copy()
    if value < 0:
        ret_value = 0
    elif value > 255:
        ret_value = 255
    return ret_value

def Sobel( ori_img ):
    ret_sobel = np.zeros( ( ori_img.shape[ 0 ], ori_img.shape[ 1 ], 3 ), np.float64 )
    sobel_x = np.array( [ [ -1, -2, -1 ], [ 0, 0, 0 ], [ 1, 2, 1 ] ] )
    sobel_y = np.array( [ [ -1, 0, 1 ], [ -2, 0, 2 ], [ -1, 0, 1 ] ] )
    for i in range( 1, ori_img.shape[ 0 ] - 1 ):
        for j in range( 1, ori_img.shape[ 1 ] - 1 ):
            part_img = ori_img[i-1:i+2, j-1:j+2, :]
            for c in range( 3 ):
                value = abs( np.sum( part_img[ :, :, c] * sobel_x ) ) +
abs( np.sum( part_img[ :, :, c] * sobel_y ) )
                value = Boundary( value )
                ret_sobel[ i, j, c ] = value
    return   ret_sobel

def convolution( ori_img, mask ):
    ret_img = np.zeros( ( ori_img.shape[ 0 ], ori_img.shape[ 1 ], 3 ), np.float64 )
    for i in range( 1, ori_img.shape[ 0 ] - 1 ):
        for j in range( 1, ori_img.shape[ 1 ] - 1 ):
            part_img = ori_img[i-1:i+2, j-1:j+2, :]
            for c in range( 3 ):
                value = np.sum( part_img[ :, :, c] * mask )
                value = Boundary( value )
                ret_img[ i, j, c ] = value
```

```
    return ret_img

def main():
    test_name = 'test.jpg'
    ori_img = cv2.imread( test_name )
    ori_img = cv2.copyMakeBorder( ori_img, 1, 1, 1, 1, cv2.BORDER_CONSTANT,
value = 0 ) # Padding
    gray_img = cv2.cvtColor( ori_img, cv2.COLOR_BGR2GRAY )
    l_mask = np.array([ [ -1, -1, -1 ], [ -1, 8, -1 ], [ -1, -1, -1 ] ] )
    m_mask = np.array([ [ 1, 1, 1 ], [ 1, 1, 1 ], [ 1, 1, 1 ] ] )
    m_mask = m_mask / 9
    cv2.imshow( 'Ori', ori_img )

    sobel_img = Sobel( ori_img )                                    # 一階微分
    s_img = convolution( sobel_img, m_mask )                        # 模糊 (去雜訊)
    La_img = convolution( ori_img, l_mask )                         # 二階微分
    n_img = s_img / ( np.amax( s_img ) - np.amin( s_img ) ) # 正規化 0 ~ 1 之間
    multi_img = np.multiply( n_img, La_img )                        # 一階微分模糊
完乘上二階微分
    final_img = ori_img + multi_img                                 # 最後再加上原
圖

    cv2.imshow( 'Final', final_img.astype( np.uint8 ) )
    cv2.imwrite( 'Sobel.jpg', sobel_img )
    cv2.imwrite( 'Smooth.jpg', s_img )
    cv2.imwrite( 'Lapalcian.jpg', La_img )
    cv2.imwrite( 'Multi.jpg', multi_img )
    cv2.imwrite( 'Final.jpg', final_img )
    cv2.waitKey()
    cv2.destroyAllWindows()

main()
```

1.6 心得分享

在過程中有試過只將灰階拿出來做，但銳化效果沒有三通到都做來的明顯，所以最後用三通道的結果來呈現作業。

發現了 opencv 中 cv2.imshow()，顯示圖片會有些問題，還沒找到問題所在，最後以寫檔的方式將圖片輸出。