

# Hw2\_Image Sharpening

學號:610415145 姓名:許晉偉

Date due:2022/11/18

Date handed in:2022/11/18

# ■ Technical description

## spatial

### (1) Laplacian operator

```
img = cv2.imread('blurry_moon.tif',0)
#img = cv2.imread('skeleton_orig.bmp',0)
H,W = img.shape[:]
img2= np.zeros((H, W), np.uint8)
k=np.array([[ -1, -1, -1],[ -1, 9, -1],[ -1, -1, -1]])
```

使用 opencv 函式庫

進行基本圖片操作

使用 numpy 函式庫進

行矩陣建立

k 變數為參考上課講義使用簡化後的 kernel

這樣卷積後就可以少掉與原圖相加的步驟

img2 變數用於存放結果圖

-1	-1	-1
-1	9	-1
-1	-1	-1

```
def convolution(x,y,img):
    result=0
    for i in range(0,3):
        for j in range(0,3):
            result=result+img[x+i,y+j]*k[i][j]
    return result

for i in range(0,H-2):
    for j in range(0,W-2):
        temp=convolution(i,j,img)
        if(temp)<0 :
            temp=0
        if(temp)>255 :
            temp=255
        img2[i][j]=temp
```

接者利用定義的卷積函式進行卷積後

進行 Relu 避免卷積後的值超過

(0~255)這個範圍造成結果錯誤

所以我用 temp 這個變數判斷後再存

入結果圖

```
cv2.imwrite('blurry_moon_Lap_spatial.jpg', img2)
cv2.imshow('result',img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

接者利用 cv2 圖函式庫顯示出原圖和銳化過的圖

## (2)Unsharpmasking

```
img = cv2.imread('blurry_moon.tif',0)
#img = cv2.imread('skeleton_orig.bmp',0)
H,W = img.shape[:]
Unsharp_Masking= np.zeros((H, W), np.uint8)
padding=np.pad(img, ((1, 1), (1, 1)), 'constant', constant_values=0)
Laplacian_img=np.zeros((H, W), np.uint8)
k=np.array([[ -1, -1, -1],[ -1, 9, -1],[ -1, -1, -1]])
```

Padding 這個變數是因為 Unsharpmasking 需要利用原圖減去 Laplacian 後的圖,兩者維度須一致所以我在 convelotion 前先把他填充

Unsharp masking consists of generating a sharp image by subtracting from an image a blurred version of itself, i.e.,

$$f_{hp}(x,y) = f(x,y) - f_{lp}(x,y). \quad (4.4-14)$$

創建好需要的變數後接者這邊套用講義的公式,這邊的濾波我選擇用均值濾波

```
def meanfilter(x,y,img):
    result=0
    for i in range(0,3):
        for j in range(0,3):
            result=result+img[x+i][y+j]*k[i][j]
    result=result//9
    return result

for i in range(0,H-2):
    for j in range(0,W-2):
        temp=meanfilter(i,j,padding)
        if(temp)<0 :
            temp=0
        if(temp)>255 :
            temp=255
        Laplacian_img[i][j]=temp
Mask=(img/255)-Laplacian_img
```

1. 程式載入一張原始影像，若為"彩色影像"則將它轉成"灰階影像"，得〈Result 1〉
2. 對〈Result 1〉做 Sobel Operator，會得到一張一階維分的〈Result 2〉
3. 對〈Result 1〉做 Laplace Operator，會得到一張二階維分的〈Result 3〉
4. 對〈Result 2〉做 算術平均濾波(Mean Filter)，將做完的結果正規化[0,1]得到〈Result 4〉
5. 將〈Result 3〉與〈Result 4〉做相乘的動作，得到我們真正要 Enhance 的數值〈Result 5〉
6. 用〈Result 5〉對 原始影像(Source Image) 做 Enhancement

做好後與講義不同的是我參考網路上的教學將原圖進行正規化並與濾波圖相減

後會獲得 Mask=>非影像邊緣的數值

```

for i in range(0,H):
    for j in range(0,W):
        temp=(img[i][j])-(Mask[i][j])
        if(temp)<0 :
            temp=0
        if abs(temp)>255 :
            temp=255
        Unsharp_Masking[i][j]=np.round(temp)

```

此時我再將原圖減掉 Mask 讓原圖中非影像邊緣的數值給去掉便完成銳化

```

result = cv2.hconcat([img,Unsharp_Masking])
cv2.imshow('result',Mask)
cv2.waitKey(0)
cv2.imshow('result',result)
cv2.imwrite('Unsharpmasking_saptial.jpg', Unsharp_Masking)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

接者將結果圖印出

### (3) high-boost filtering

High-boost filtering generalizes this by multiplying  $f(x, y)$  by a constant  $A \geq 1$ :

$$f_{hb}(x, y) = Af(x, y) - f_{lp}(x, y) \quad (4.4-15)$$

high-boost filtering 與 Unsharpmasking 整體的步驟幾乎一樣

```

A=3
for i in range(0,H):
    for j in range(0,W):
        temp=img[i][j]-(A*Mask[i][j])
        if(temp)<0 :
            temp=0
        if abs(temp)>255 :
            temp=255
        highboost_filteringimg[i][j]=np.round(temp)

```

差別在於用了一個常數將 Mask 放大,也就是將非影像邊緣的數值(雜訊)給放大,

再讓我消去它後便達成更強的銳化效果,這邊我選的係數是 3

## frequency

```
import cv2
import numpy as np
import math
from matplotlib import pyplot as plt
from scipy.fftpack import fft,ifft
```

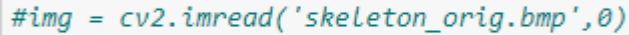
使用 `opencv` 用於圖片的基本讀寫與顯示

`numpy` 用於進行陣列操作

`math` 用於影像處理的公式計算

`matplotlib` 用於圖表的顯示

`scipy.fftpack` 用於傅立葉轉換

```
img = cv2.imread('blurry_moon.tif',0)
img = cv2.imread('skeleton_orig.bmp',0)
H,W = img.shape[:]
```

讀取圖片

```
fft2 = np.fft.fft2(img)
lap_fft2 = np.fft.fft2(img)

#Centralization
shift2center = np.fft.fftshift(fft2)
shift2center[int((H/2)-1) : int((H/2)+1), int((W/2)-1) : int((W/2)+1)] = 0
```

將原圖做二維傅立葉轉換,並將得到的頻譜進行中心化

即將低頻訊號集中至中心

```
#Laplacian sharpening
for i in range(H):
    for j in range(W):
        lap_fft2[i][j] = -4*(math.pi**2)*abs((i-H/2)**2 + (j-W/2)**2)*shift2center[i][j]

#Inverse Centralization
center2shift = np.fft.ifftshift(lap_fft2)

#Inverse Fourier transform
ifft2 = np.fft.ifft2(center2shift)
```

$$\nabla^2 f(x, y) \Leftrightarrow -4\pi^2 \left[ (u - M/2)^2 + (v - N/2)^2 \right] F(u, v).$$

將中心化後的頻譜做拉普拉斯轉換，並根據上方的公式進行運算，也在計算中

改變濾波器中心,接著將做完拉普拉斯的頻譜進行逆中心化, 將逆中心化的頻譜

進行傅立葉逆轉

並且針對三種不同的影像處理技術進行不同的處理

### (1) Laplacian operator

```
lap_img = np.abs(iff2)/np.max(np.abs(iff2))  
result_img = lap_img + (img/255)
```

以右方的公式將原圖與特徵圖相加得到結果

$$g(x, y) = \begin{cases} f(x, y) - \nabla^2 f(x, y) \\ f(x, y) + \nabla^2 f(x, y) \end{cases}$$

### (2) Unsharpmasking

```
#normalization & unsharp masking  
lap_img = np.abs(iff2)/np.max(np.abs(iff2))  
result_img = (img/255) - lap_img
```

套用公式將原圖減掉特徵圖得到結果

$$f_{hp}(x, y) = f(x, y) - f_{lp}(x, y)$$

### (3) high-boost filtering

```
A = 2  
lap_img = np.abs(iff2)/np.max(np.abs(iff2))  
result_img = A*(img/255) - lap_img
```

套用公式將原圖乘上常數(2)減掉特徵圖得到結果

$$f_{hb}(x, y) = Af(x, y) - f_{lp}(x, y)$$

```
cv2.imshow('input image', img)  
cv2.imshow('result', result_img)  
result_img=result_img*255  
cv2.imwrite('blurry_moon_Lap_frequency.jpg', result_img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

最後都透過 CV2函式庫將結果顯示出來

# Experimental results

## 1. Laplacian operator

### Spatial

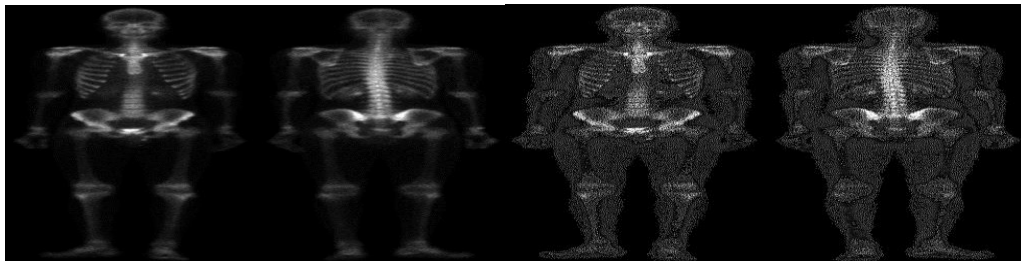
原圖

結果

'blurry\_moon.tif'



'skeleton\_orig.bmp'



### Frequency

'blurry\_moon.tif'

原圖

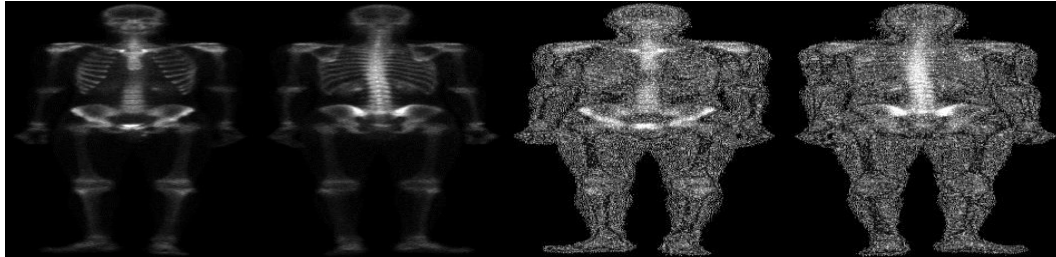
結果



原圖

結果

'skeleton\_orig.bmp'



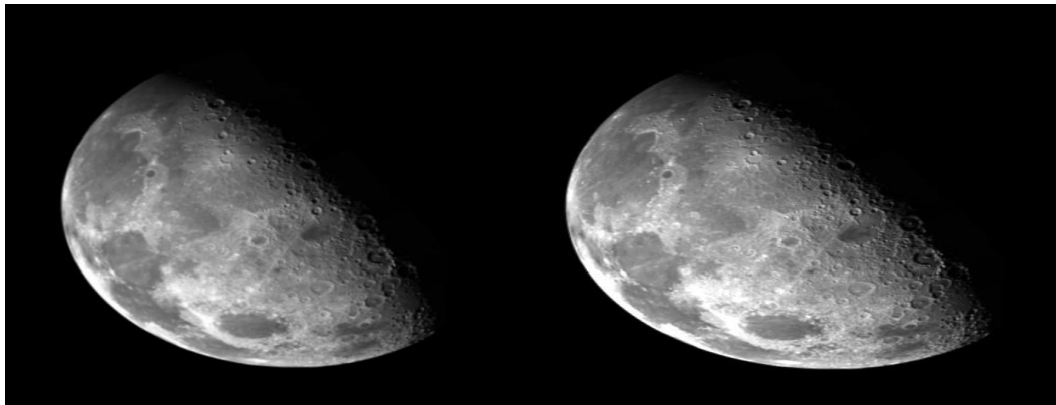
(2) Unsharpmasking

**Spatial**

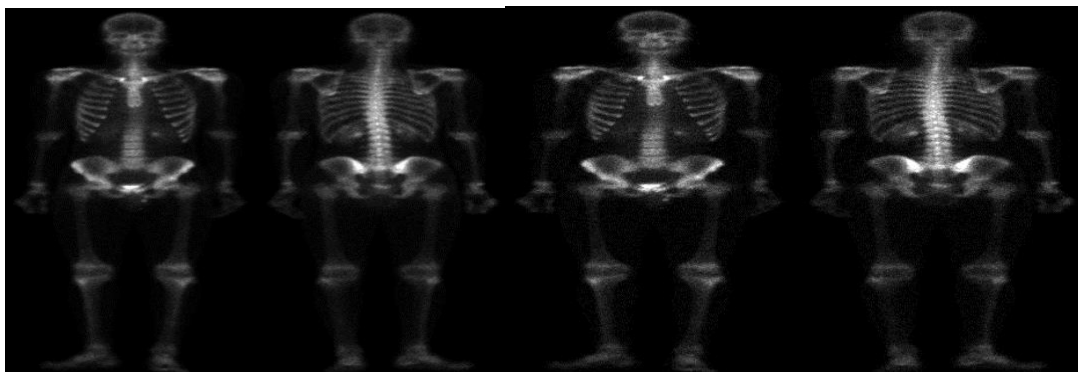
原圖

結果

'blurry\_moon.tif'



'skeleton\_orig.bmp'





## Frequency

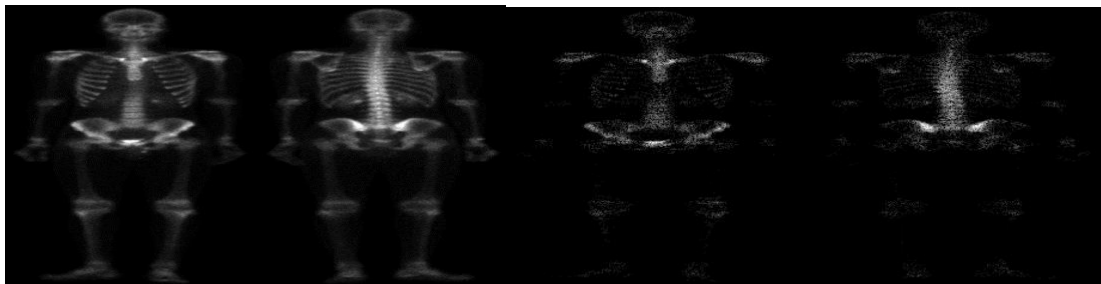
'blurry\_moon.tif'

原圖

結果



'skeleton\_orig.bmp'



(3) high-boost filtering

## Spatial

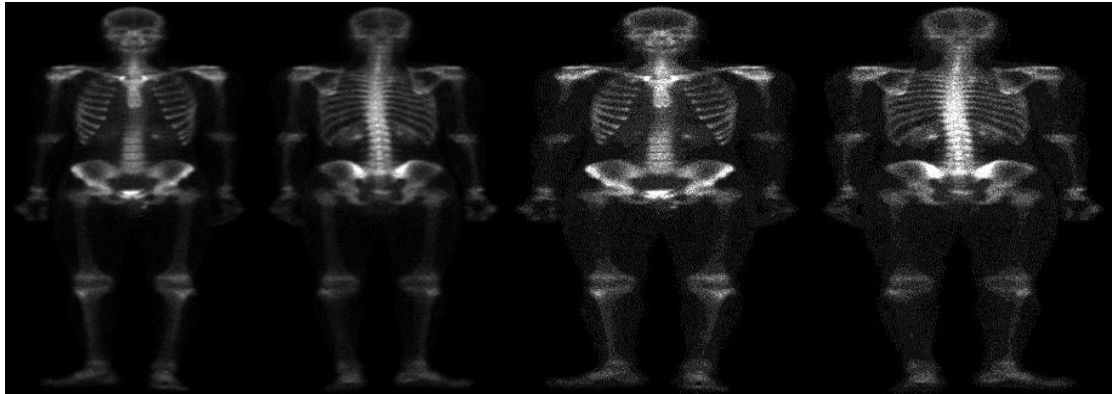
原圖

結果

'blurry\_moon.tif'



'skeleton\_orig.bmp'

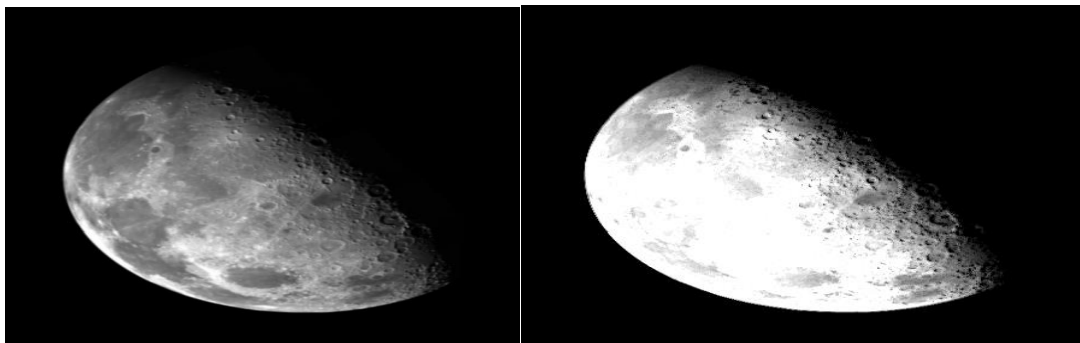


Frequency

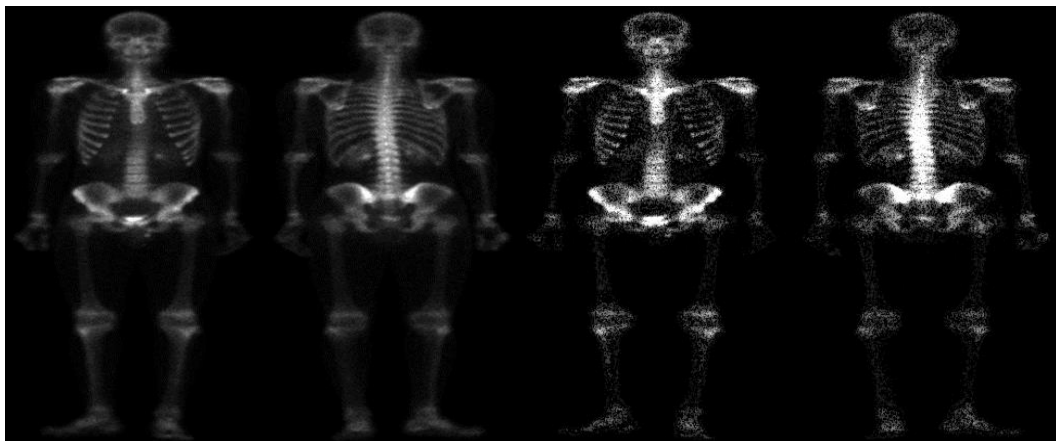
'blurry\_moon.tif'

原圖

結果



'skeleton\_orig.bmp'



# ■ Discussions

## (1) Laplacian operator

空間域的部份上次就做過了,這次加上了頻域,我覺得相比較起來空間域比較直觀好理解多,我覺得實作頻域非常困難,中間各種轉換出來的圖對我來說非常不直觀,看不出來自己的寫到底有沒有問題,所以這次花了很多的時間再弄頻域

## (2) Unsharpmasking

一開始我參考網路上的用高斯模糊去作但是我發現 `sigma` 挑選很困難做出來效果也不好,反而換成了比較簡單的均值濾波後效果好很多,但實作出來結果還是怪怪的,直到發現有人用正規化後參考他的做法,終於把結果用出來

## (3) high-boost filtering

其實一開始看講義不太懂他跟 Unsharpmasking 的差別後來看了各種資料發現原來 Unsharpmasking 就是 high-boost filtering 的一種做起來其實差不多很快就用好了

# ■ References and Appendix

<https://jennaweng0621.pixnet.net/blog/post/404319692-%5Bpython-%2B-opencv%5D-%E5%82%85%E7%AB%8B%E8%91%89%E8%BD%89%E6%8F%9B-%28fourier-transform%29>

[https://blog.csdn.net/full\\_speed\\_turbo/article/details/72916814](https://blog.csdn.net/full_speed_turbo/article/details/72916814)

<https://jason-chen-1992.weebly.com/home/-unsharp-masking>