**Sharpening**

4110056030 資工三 鄭詠謙

* **Code**

import cv2

import numpy as np

# 定義卷積核

masks = np.array([

    [1, 1, 1,

     1, 1, 1,

     1, 1, 1],

    [-1, -2, -1,

      0,  0,  0,

      1,  2,  1],

    [-1, 0, 1,

     -2, 0, 2,

     -1, 0, 1],

    # [0, -1, 0,

    #  -1,  4, -1,

    #  0, -1, 0]

    #  ,

    [-1, -1, -1,

     -1,  8, -1,

     -1, -1, -1]

]).reshape(4, 3, 3)

source\_image\_path = "300px-Lenna.jpg"

def convolution(src, mask):

    dst = np.zeros\_like(src)

    for i in range(1, src.shape[0] - 1):

        for j in range(1, src.shape[1] - 1):

            region = src[i-1:i+2, j-1:j+2]

            convo = np.sum(region \* mask)

            dst[i, j] = np.clip(convo, 0, 255)

    return dst

# 讀取影像

src\_img = cv2.imread(source\_image\_path)

src\_temp = src\_img

if src\_img is None:

    print("Error! Can not open the Image file!")

else:

    cv2.imshow("Source Image", src\_img)

cv2.namedWindow("First-order differential", cv2.WINDOW\_AUTOSIZE)

cv2.namedWindow("Mean Filter of first-order", cv2.WINDOW\_AUTOSIZE)

cv2.namedWindow("Second-order differential", cv2.WINDOW\_AUTOSIZE)

cv2.namedWindow("Unsharp Masking", cv2.WINDOW\_AUTOSIZE)

# 轉換成灰階影像

src\_gray = cv2.cvtColor(src\_img, cv2.COLOR\_BGR2GRAY)

cv2.imshow("Gray level", src\_gray)

# 初始化結果矩陣

diff\_1 = np.zeros\_like(src\_gray)

diff\_2 = np.zeros\_like(src\_gray)

mean\_f = np.zeros\_like(src\_gray)

# 進行卷積操作

diff\_1 = convolution(src\_gray, masks[1]) # Sobel y

diff\_2 = convolution(src\_gray, masks[2]) # Sobel x

# 計算Sobel邊緣檢測結果

sobel = np.clip(diff\_1 + diff\_2, 0, 255)

diff\_1 = sobel

diff\_2 = convolution(src\_gray, masks[3]) # Laplacian

mean\_f = convolution(diff\_1, masks[0])  # Mean filter on Sobel result -> Unsharp mask

# Unsharp masking

for i in range(1, diff\_2.shape[0] - 1):

    for j in range(1, diff\_2.shape[1] - 1):

        src\_img[i, j] = np.clip(src\_img[i, j] + diff\_2[i, j] \* (mean\_f[i, j] / 255), 0, 255)

# 灰度圖像與Sobel結果相加

gray\_plus\_sobel = cv2.add(src\_gray, diff\_1)

# 灰度圖像與laplace結果相加

gray\_plus\_laplace = cv2.add(src\_gray, diff\_2)

# 灰度圖像與正規化後結果相加

gray\_plus\_std = cv2.add(src\_gray, mean\_f)

# 算 PSNR 值

psnr\_value\_sobel = cv2.PSNR(src\_gray, gray\_plus\_sobel)

print("sobel PSNR:", psnr\_value\_sobel)

psnr\_value\_laplace = cv2.PSNR(src\_gray, gray\_plus\_laplace)

print("laplace PSNR:", psnr\_value\_laplace)

psnr\_value\_sharp = cv2.PSNR(src\_gray, gray\_plus\_std)

print("std PSNR:", psnr\_value\_sharp)

# 顯示結果

cv2.imshow("First-order differential", diff\_1)

cv2.imshow("sobel", gray\_plus\_sobel)

cv2.imshow("Mean Filter of first-order", mean\_f)

cv2.imshow("std", gray\_plus\_std)

cv2.imshow("Second-order differential", diff\_2)

cv2.imshow("laplace", gray\_plus\_laplace)

cv2.imshow("Unsharp Masking", src\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

* **Result**
  + 一階微分(sobel):

mask 有兩個 => dy 跟 dx

[-1, -2, -1,

 0, 0, 0,

 1, 2, 1],

[-1, 0, 1,

-2, 0, 2,

-1, 0, 1]

 => 

原圖 一階微分結果

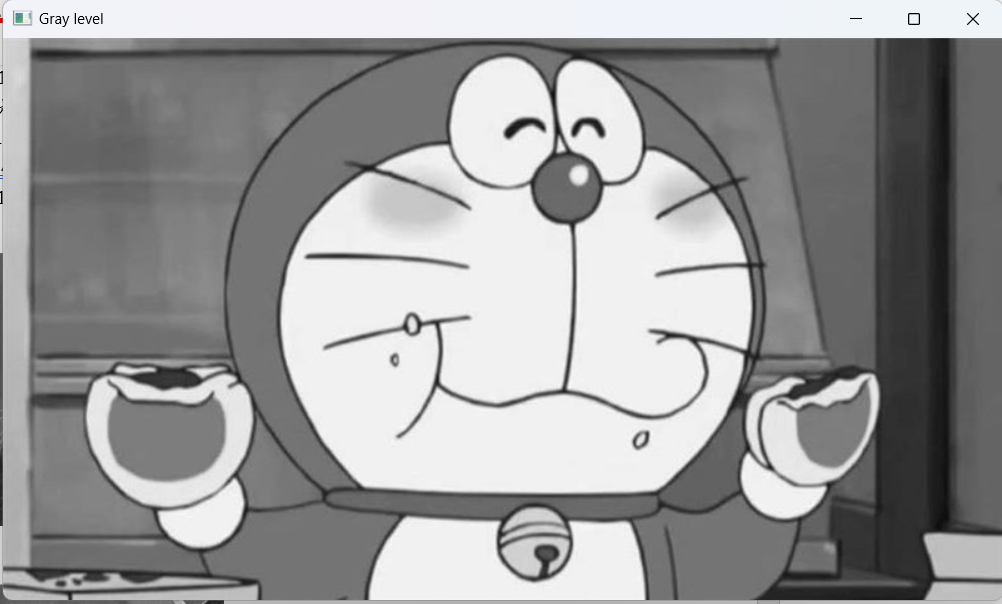
* 

與原圖相加後

PSNR: 27.94

另外找的圖

原圖:



一階微分:



* + 二階微分(laplace):

有兩種mask:

程式中使用

[-1, -1, -1,

  -1,  8, -1,

  -1, -1, -1]

這個是原版:

 [0, -1, 0,

  -1,  4, -1,

  0, -1, 0]

 => 

原圖 二階微分結果

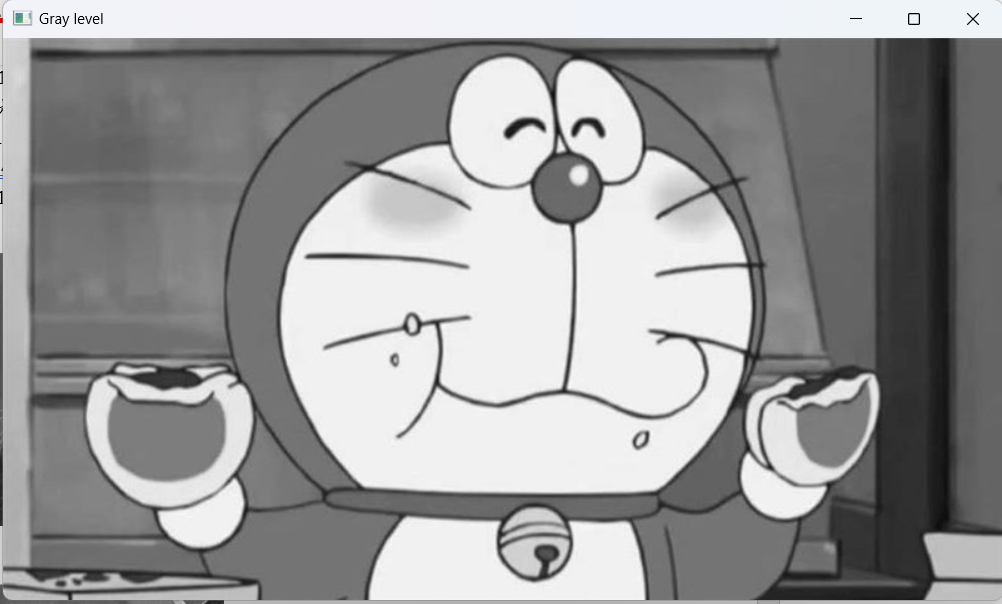
* 

與原圖相加後

PSNR: 27.92

另外的圖

原圖:



二階微分:



* + 取正規化

對 Sobel 邊緣偵測結果套用均值濾波器以平滑影像，

算術平均濾波(Mean Filter)，而後正規化[0,1]

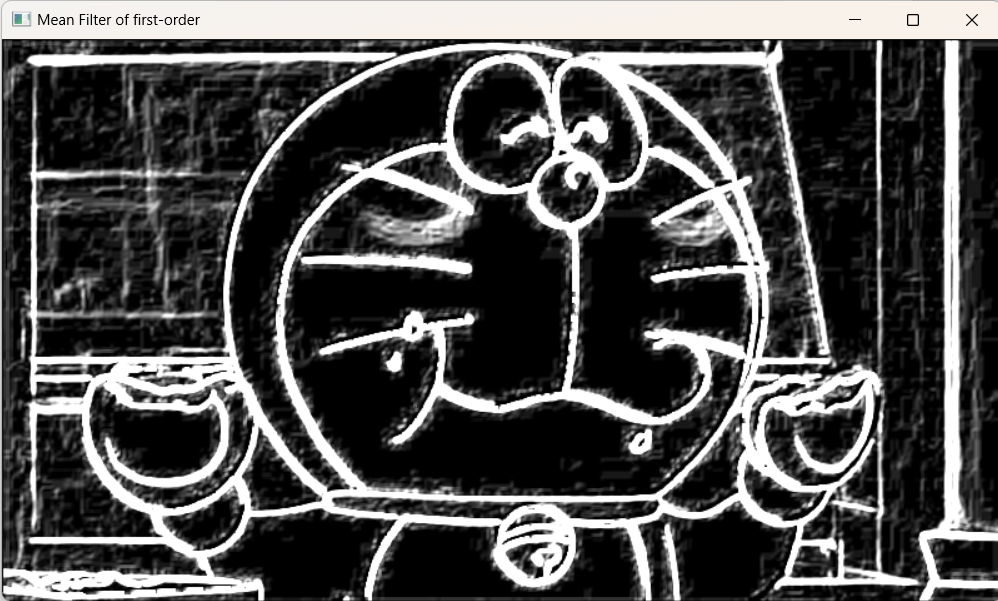
 => 

非影像邊緣的數值 加上原始

放大過後的雜訊

另外的圖

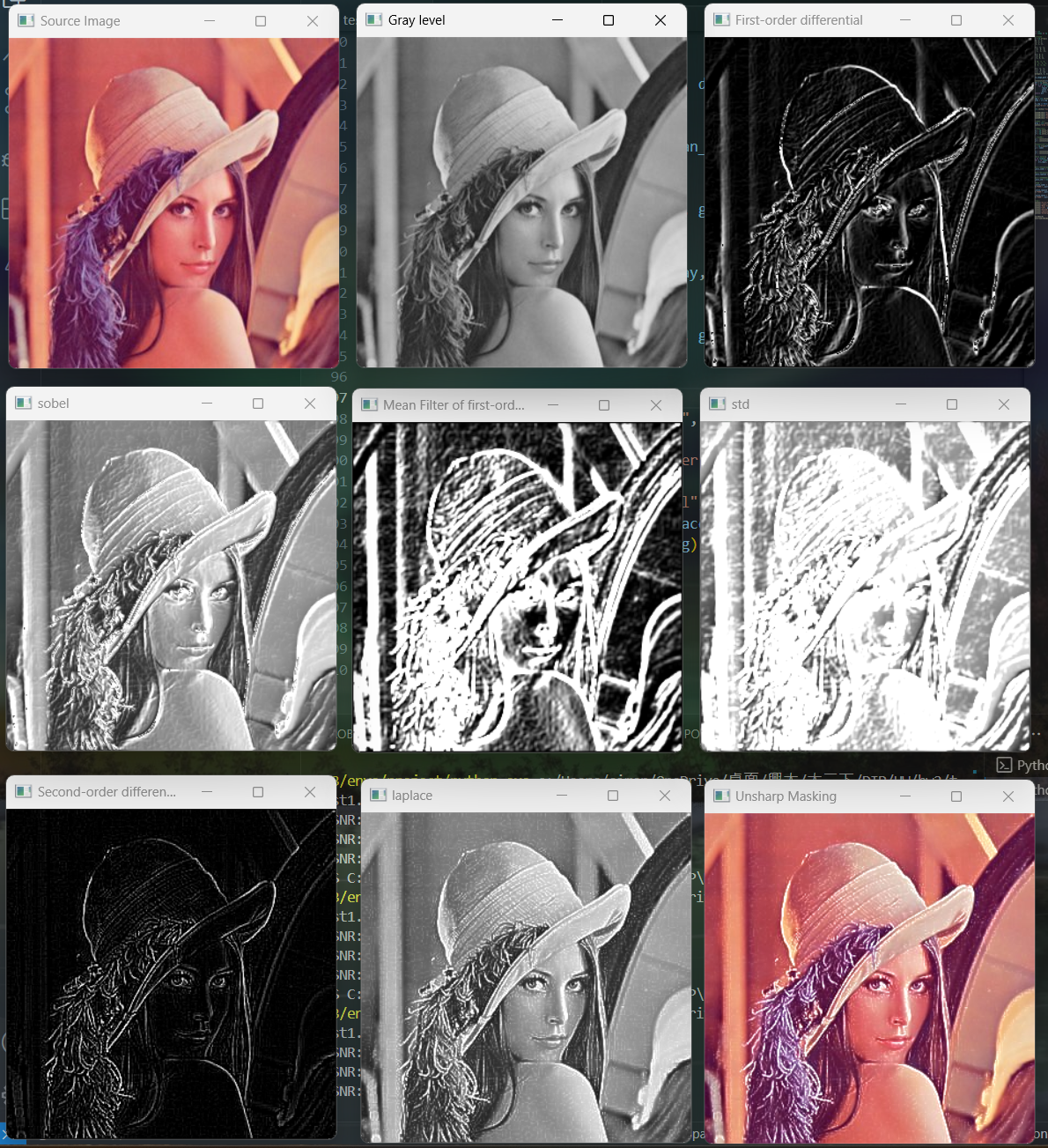
正規化:



* **Discussion**
  + **Sharpening過程**

**設計一個Mask 罩在影像的邊緣上 ，針對邊緣的部分做sharpening，其餘的平坦區略過，避免把平坦區的雜訊放大。**

雜訊在Frequency domain 分佈也是落在高頻區，那麼連雜訊也放大。為了要避免這個問題，並做到影像銳化的效果，使用**Unsharp Masking**。



* + 計算PSNR

分成兩種方法Sobel & Laplace，

PSNR 是比較不同影像處理技術品質的指標。

****

* + **邊緣偵測**：使用sobel和laplace濾波器可以有效的凸顯影像中的邊緣。sobel濾波器基於一階導數來偵測邊緣，這使得對於雜訊變得敏感，而laplace濾波器使用二階導數，更明顯的邊緣偵測，但可能會放大雜訊。
  + **影像增強**：非銳化遮罩透過銳化邊緣來增強影像，這可以提高清晰度，但如果應用不佳，可能會產生artifacts。