

## 作業一

這次作業是要在至少四張圖像用上影像處理的技術去改善圖或提高圖像的質量。

這次作業用上的技術有：

### 1. Histogram Equalization直方圖均衡化:

先算出輸入圖像的像素在0~255中各有幾個，然後做累加，最後根據累加的結果將對應的像素按乘上255，使圖像色彩平均分佈

### 2. Mean Filter均值濾波:

均值濾波是圖像平滑化其中一種技術，主要是用來過濾噪點。意指將輸入圖像的每個像素值取周圍 $N \times M$ 個像素值的均值，從而去除圖中的噪點。

### 3. Median Filter中位數濾波:

中位數濾波是另一種圖像平滑化的技術，主要也是用來過濾噪點。意指將輸入圖像的每個像素值取周圍 $N \times M$ 個像素值中位數，從而去除圖中的噪點。

### 4. Sharpen Filter-1銳化濾波-1:

銳化濾波是將圖像的邊緣和變化部份突顯出來的，此作業做了兩個銳化濾波，第一個是用拉普拉斯算子 $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ 並在中心點再加上1來突顯邊緣變化。

### 5. Sharpen Filter-2銳化濾波-2:

第二個方法是將原圖像減去一個均值濾波，得出鄰邊緣的圖像，然後用原圖加上邊緣圖像，藉此來強化邊緣。

### 6. Boundary Extraction邊緣提取

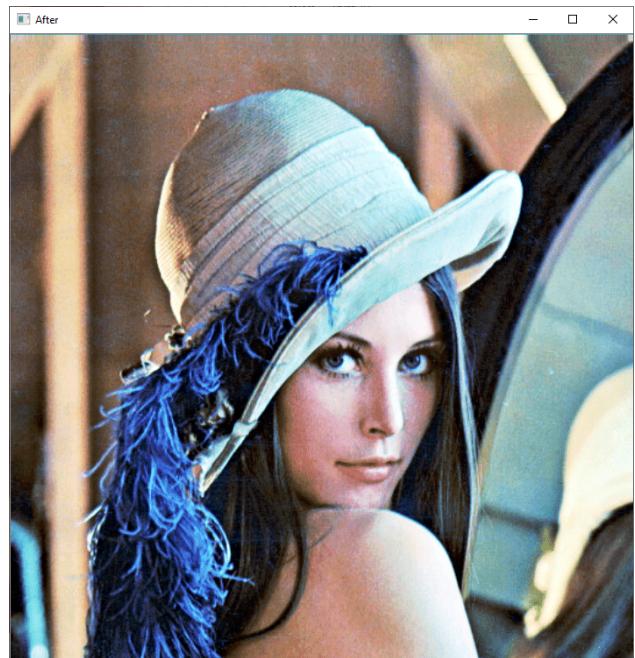
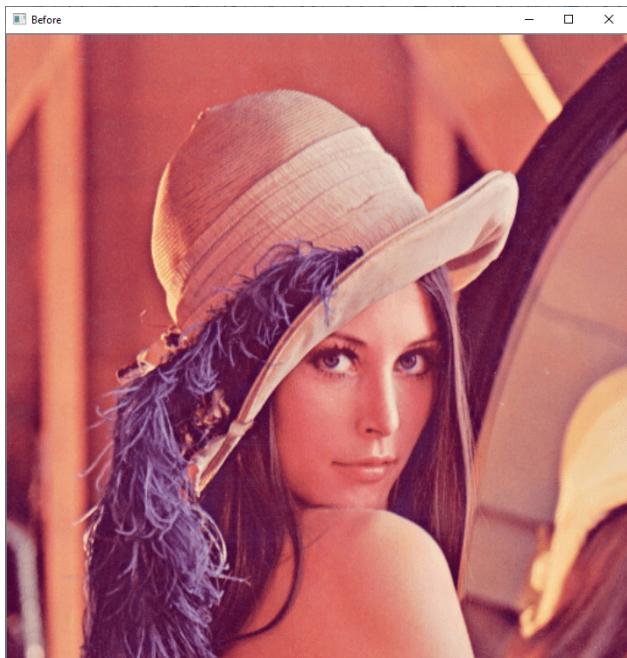
邊緣提取，用擴張濾波將原圖像擴大，然後減去原圖像得出圖像的邊界。

第一部份：

用名氣女孩Lenna的圖片做Histogram Equalization讓她變得更漂亮。

圖1-1: 原圖

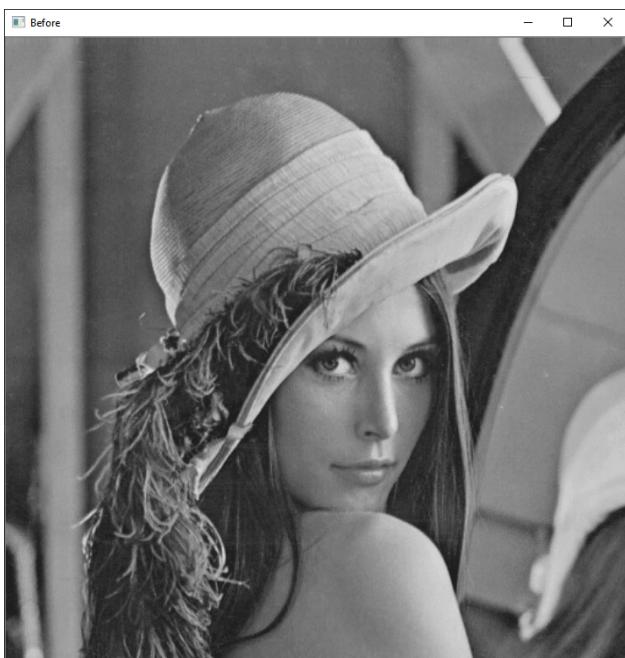
圖1-2: 直方圖均衡化



可以看到圖1-1的原圖比較偏向紅色，不能明顯地看出圖像的顏色差異，我用了Histogram來處理後，明顯看到顏色的不同，圖像亦邊得較為明亮。彩色圖片可能不夠明顯，所以我將圖片灰化處理後再做一次對比，結果如下（圖1-3、1-4），可以看得出來在灰階化後的圖像明顯的變化，更容易看來每個灰階的差異，圖像亦變得更立體了？

圖1-3: 原圖（灰階）

圖1-4: 直方圖均衡化



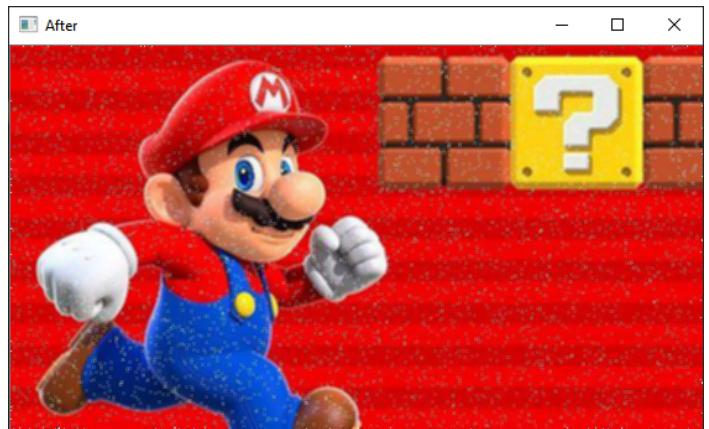
第二部份：

用經典遊戲瑪利奧圖片加上噪點，來進行去噪的試驗。

圖2-1-1 原圖（有噪點）



圖2-1-2 Mean Filter均值濾波

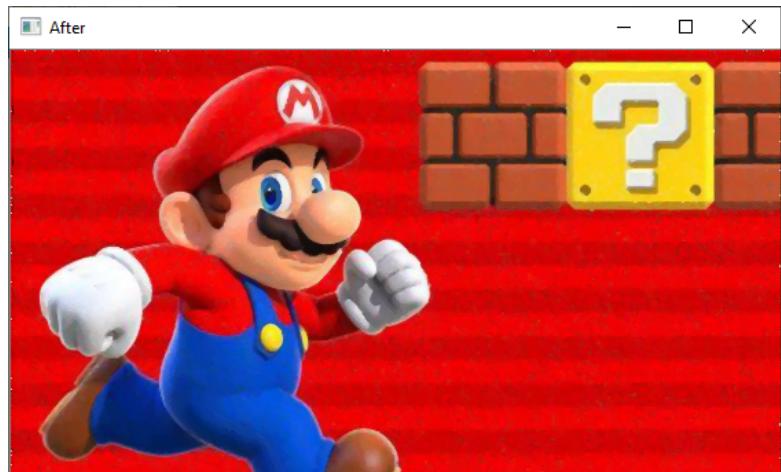


可以看到均值濾波對噪點的作用沒想像中好，而圖片也因為處理後變得模糊。下面我採用 Median Filter中位數濾波來再進行一次處理。

圖2-2-1 原圖（有噪點）



圖2-2-2 Median Filter中位數濾波



可以看到中位數濾波比均值濾波要得出更好的結果，在去噪同時亦保持原有的清晰度。

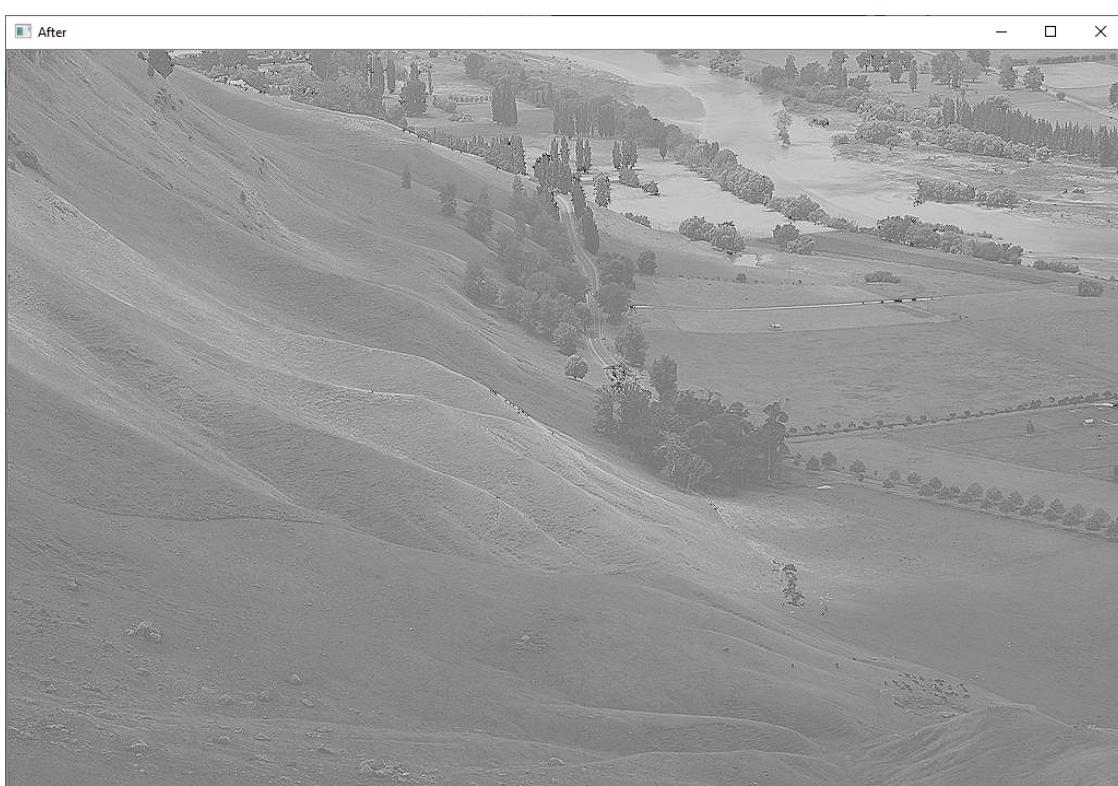
第三部份：

第三張圖片為灰階的草原圖，我將會用Sharpen Filter-1銳化濾波-1來對圖像做處理

圖3-1-1 原圖



圖3-1-2 Sharpen Filter-1銳化濾波-1後

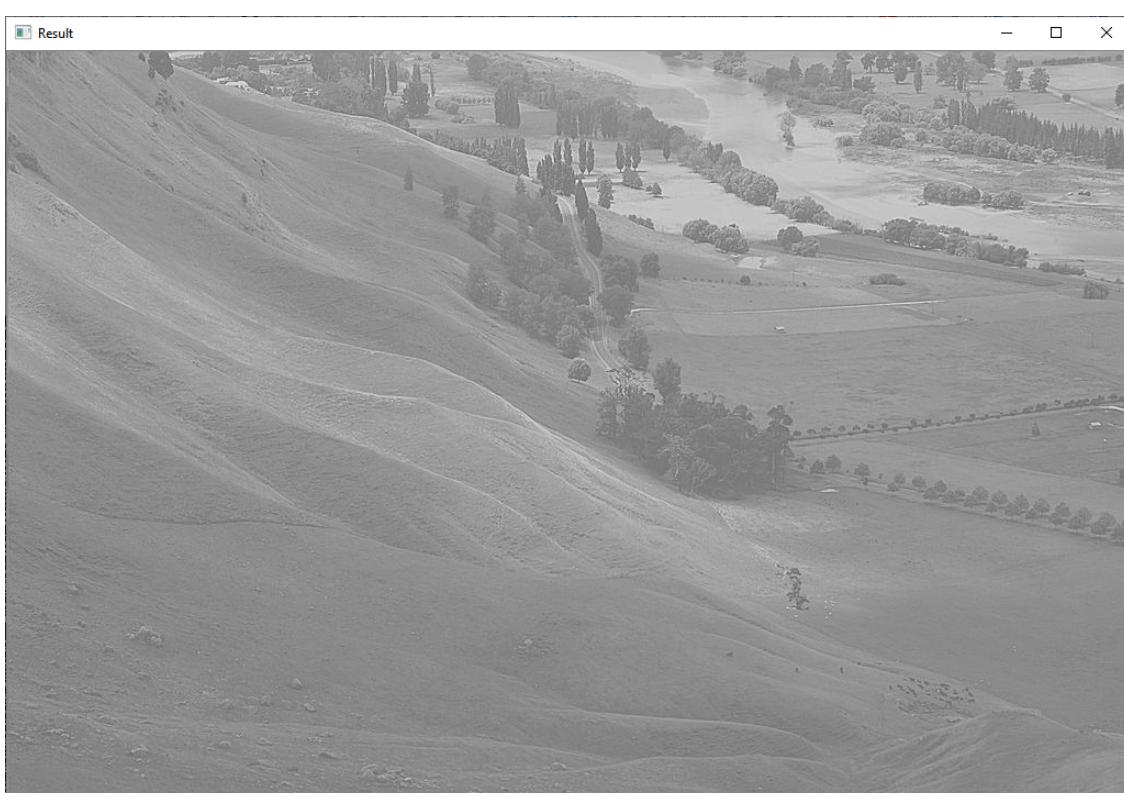


像這種細微的改變圖像太小不容易看清，所以我特意把圖片拉大，在經過銳化後的圖片，邊線上的增強明顯比原圖更容易看得清楚，但是我發現這有些地方會變得奇怪，所以網上找了另一種方式來做銳化，效果如下。

圖3-2-1原圖



圖3-2-2Sharpen Filter-2銳化濾波-2

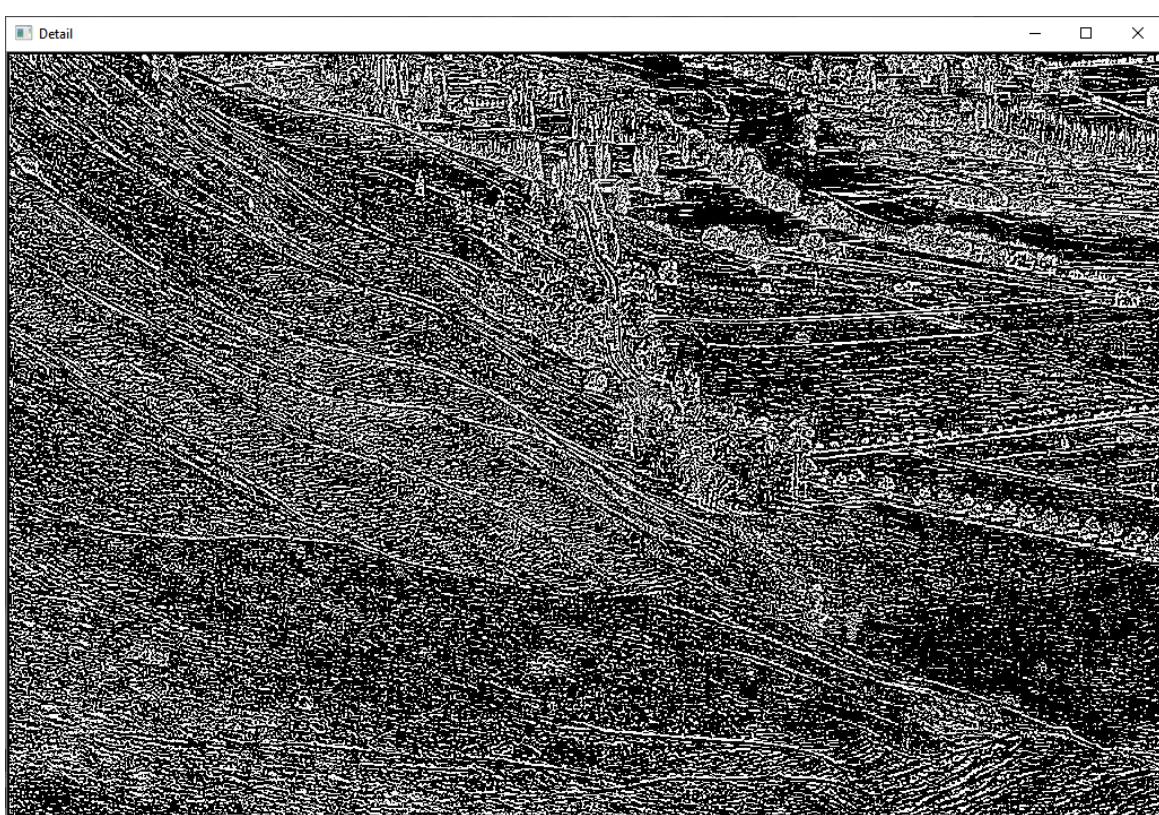


這個方法有如開頭的說明是先用均值濾波將影像變模糊，達到類似擴張的效果，然後再用原圖減去均值濾波圖，從而顯現所有邊界的，最後再加上原圖來突出圖片的邊界，均值濾波和邊界效果圖如下。

圖3-2-3 均值濾波後的圖片



圖3-2-4 邊緣圖像



可以看到雖然圖片好像很花很像噪點圖，但仍然能夠看到邊緣輪廓，加上原圖後效果就更為明顯，亦不會看到奇怪的點，在我看來，第一個銳化濾波中間樹林路上的黑點有些奇怪，我研究了許久沒能發現原因，最後google了另一個方法去做了另一個銳化濾波。

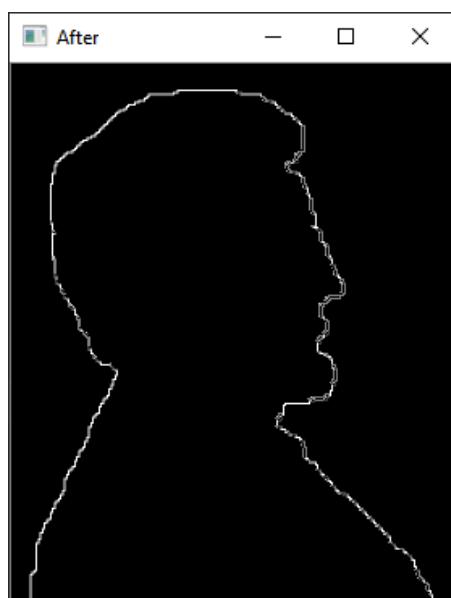
第四部份：

這部份較為簡單，主要是想用擴張濾波來勾勒出圖片中人像的輪廓，因為這種圖片不好找，就直接在老師的講義中截取了來做素材。

圖4-1



圖4-2



這個部份就相對簡單了，把原圖素做了擴張後再將減去原圖，就可以成功勾勒出人像的輪廓了，感覺和銳化濾波2有點相似

總結：

1. Histogram Equalization 是一個很好用的工具，它能把有色差和陰暗的圖片均衡化從而把圖片正常化，觀感上亦能突顯出圖片的對比度等令圖片更立體。
2. 中位數比均值更能保持圖片原有的清晰度，但其實兩個濾波都是依靠周圍像素的值來達到去噪的果。
3. 銳化濾波-1的拉普拉斯算子不知何解有某些地方會出現奇怪的「噪點」，暫時還未想出有什麼端倪，可能是因為原圖像在被些點上有噪點，而噪點的邊界被突顯出來，所以就發生這樣的情況，但我也試過先用中位數濾波去噪再做銳化，其結果是不堪入目，也就沒有放上來，暫時沒能想到什麼好辦法。最後，就用了網上的另一個方法做銳化，效果還不錯。
4. 勾勒輪廓看來只適用於一些特殊圖片，對較為複雜的圖像就不容易成功。不過不一樣的技術適用的範圍本來也不盡相同，基本上以上的技術亦只能針對不同情況使用。像第三部份發現的奇怪地方可能是要先找到噪點，再用局部去噪來處理，然後才銳化。不過這部份可能要再試作一下。

以下為本次作業的coding:

### 第一部份Histogram Equalization

```

import numpy as np
import cv2
import copy

def CountPixel(original_image, ArrayB, ArrayG, ArrayR, imageH, imageW):
    B=0
    G=1
    R=2
    for i in range(imageH): #use i as index of image to determine which pixels
    need to be filtered
        for j in range(imageW): #use j as index of image to determine which
    pixels need to be filtered
            ArrayB[original_image[i][j][B]] += 1 #count the corresponding pixel
    number in the image from 0 to 255
            ArrayG[original_image[i][j][G]] += 1
            ArrayR[original_image[i][j][R]] += 1
    return ArrayB, ArrayG, ArrayR

def Accumulate(Array):
    AccumArray = [0]*256
    j = 0
    for i in range(256):
        AccumArray[i] = Array[i] + j #accumulate the appear pixel
        j = AccumArray[i]
    return AccumArray

def ImageHeq(original_image, ACPAB, ACPAG, ACPAR, imageH, imageW):
    img_tmp = original_image
    R = 2
    G = 1
    B = 0
    for i in range(imageH):
        for j in range(imageW):
            img_tmp[i][j][B] = ACPAB[img_tmp[i][j][B]] * 255 #do the histogram
    equalization
            img_tmp[i][j][G] = ACPAG[img_tmp[i][j][G]] * 255
            img_tmp[i][j][R] = ACPAR[img_tmp[i][j][R]] * 255
    return img_tmp

image = cv2.imread('lenna_rgb.png')
image_temp = copy.deepcopy(image)
arr = np.array(image)
imageSize = arr.shape

H = imageSize[0]
W = imageSize[1]
IRed = [0]*256 #initial the RGB value for counting the corresponding number of
image
IGreen = [0]*256
IBlue = [0]*256
AcPAR = [0]*256
AcPAG = [0]*256
AcPAB = [0]*256

CountPixel(image_temp, IBlue, IGreen, IRed, H, W)

EveryPAR = np.array(IRed) / (H*W)

```

```

EveryPAG = np.array(IGreen) / (H*W)
EveryPAB = np.array(IBlue) / (H*W)

AcPAR = Accumulate(EveryPAR)    #accumulate the number of appear pixel for each
channel
AcPAG = Accumulate(EveryPAG)
AcPAB = Accumulate(EveryPAB)

image_temp = ImageHeq(image_temp, AcPAB, AcPAG, AcPAR, imageSize[0],
imageSize[1])  #do the histogram equalization
print('end')
cv2.imshow('Before', image)
cv2.imshow('After', image_temp)
cv2.waitKey(0)

```

第二部份 Median Filter中位數濾波: (Mean Filter均值濾波會結合Sharpen Filter-2銳化濾波-2一並列出)

```

import numpy as np
import cv2
import copy

def medFilter(img, imgS, fils):
    temp = []
    imgTempf = copy.deepcopy(img)
    for k in range(imgS[2]):  # use k as index of image to determine which
pixels need to be filtered
        for i in range(imgS[0]):  # use i as index of image to determine which
pixels need to be filtered
            for j in range(imgS[1]):
                imgTempf = conv(imgTempf, imgS, fils, temp, i, j, k)
    return imgTempf

def conv(imgOriginal, imgSS, filSS, tempp, ii, jj, kk):
    imgNew = copy.deepcopy(imgOriginal)
    for l in range(filSS[0]):  # if 3x3 kernel, l = 0, 1, 2
        if ii + l < imgSS[0]:
            for m in range(filSS[1]):
                if jj + m < imgSS[1]:
                    tempp.append(imgOriginal[ii + l][jj + m][kk])  # save pixel
value of image
        else:
            break
    else:
        break
    temppSort = sorted(tempp)  # sort the pixel
    num = int(len(temppSort) / 2)  # to find the middle index of pixel
    if (ii + 1 < imgSS[0]) and (jj + 1 < imgSS[1]):
        imgNew[ii + 1][jj + 1][kk] = temppSort[num]  # put the middle index of
pixel value to the core of filter location of image
    tempp.clear()
    temppSort.clear()
    return imgNew

image = cv2.imread('mario.jpg')
imgTemp = copy.deepcopy(image)
arr = np.array(imgTemp)
imgSize = arr.shape

fNum = 3

```

```

kernel = []
for i in range(fNum):
    kernel.append([1] * fNum)

fil = np.array(kernel)
filSize = fil.shape
print('kernel: ', kernel)
print('filterSize: ', filSize)

imgTemp2 = medFilter(imgTemp, imgSize, filSize)

print('imageSize: ', imgSize)
print('end')
cv2.imshow('Before', image)
cv2.imshow('After', imgTemp2)
cv2.waitKey(0)

```

### 第三部份 Sharpen Filter-1銳化濾波-1:

```

import numpy as np
import cv2
import copy

def sharpenFilter(img, imgS, ker, fils):
    imgTempf = copy.deepcopy(img)
    imgAfter = copy.deepcopy(img)
    for i in range(imgS[0]): #use i as index of image to determine which pixels
    need to be filtered
        for j in range(imgS[1]):
            imgAfter = conv(imgTempf, imgAfter, imgS, ker, fils, i, j)

    return imgAfter

def conv(imgOriginal, imgNew, imgSS, ker, filSS, ii, jj): #imgOriginal - Original
    Image, imgNew - New Image, ker - filter
    valFromFilter = [] #imgSS - imageSize, filSS - Filter Size, ii - X index,
jj - Y index
    for l in range(filSS[0]): #do the dot product by x direction
        if ii + filSS[0] < imgSS[0]: #check the location whether will exceed
the image size
            for m in range(filSS[1]):
                if jj + filSS[1] < imgSS[1]:
                    valFromFilter.append(imgOriginal[ii + l][jj + m] * ker[l][m])
#Dot product of filter and image
        else:
            break
    else:
        break

    valSum = sum(valFromFilter) #sum of all dot product
    if ii + filSS[0] < imgSS[0] and jj + filSS[1] < imgSS[1]:
        imgNew[ii + 1][jj + 1] = valSum #assign sum value to image by the
location of filter center
    valFromFilter.clear()
    return imgNew

image = cv2.imread('lenna_rgb.png')
imgTemp = copy.deepcopy(image)
imgGray = cv2.cvtColor(imgTemp, cv2.COLOR_BGR2GRAY)
arr = np.array(imgGray)
imgSize = arr.shape

```

```

kernel = [[0, -1, 0], [-1, 5, -1], [0, -1, 0]]

fil = np.array(kernel)
filSize = fil.shape
print('filter: ', kernel)
print('filterSize: ', filSize)

imgTemp2 = sharpenFilter(imgGray, imgSize, kernel, filSize)

print('imSize: ', imgSize)
print('end')

cv2.imshow('Before', imgGray)
cv2.imshow('After', imgTemp2)
cv2.waitKey(0)

```

### 第三部份 Sharpen Filter-2銳化濾波-2:

```

import numpy as np
import cv2
import copy

def smoothFilter(img, imgs, fils):
    temp = []
    imgTempf = copy.deepcopy(img)
    for i in range(imgs[0]):
        for j in range(imgs[1]):
            imgTempf = conv(img, imgTempf, imgs, fils, temp, i, j)

    return imgTempf

def conv(imgTempff, imgNew, imgSS, filSS, tempp, ii, jj):
    for l in range(filSS[0]):
        if ii + filSS[0] < imgSS[0]:
            for m in range(filSS[1]):
                if jj + filSS[1] < imgSS[1]:
                    tempp.append(imgTempff[ii + l][jj + m])
                else:
                    break
            else:
                break

    temppSum = sum(tempp)
    cenPointx = int(filSS[0] / 2)
    cenPointy = int(filSS[1] / 2)
    num = int(temppSum / (filSS[0] * filSS[1]))
    if (ii + filSS[0] < imgSS[0]) and (jj + filSS[1] < imgSS[1]):
        # print('imgTempf[' , ii + 1, ',' , jj + 1, ']: ', imgTempff[ii + 1][jj + 1], 'temppSum: ', temppSum)
        imgNew[ii + cenPointx][jj + cenPointy] = num
    tempp.clear()
    temppSum = 0
    return imgNew

image = cv2.imread('lenna_rgb.png')
imgGray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
imgTemp = copy.deepcopy(imgGray)
imgTempResult = copy.deepcopy(imgGray)
arr = np.array(imgTemp)
imgSize = arr.shape

```

```

# print(imgTemp)
print('imSize: ', imgSize)

fNum = 5
kernel = []
for i in range(fNum):
    kernel.append([1] * fNum)

fil = np.array(kernel)
filSize = fil.shape
print('kernel: ', kernel)
print('imSize: ', filSize)

imgTemp2 = smoothFilter(imgTemp, imgSize, filSize)
for i in range(imgSize[0]):
    for j in range(imgSize[1]):
        imgTemp[i][j] = int(imgGray[i][j]) - int(imgTemp2[i][j])
        if imgTemp[i][j] > 255:
            imgTemp[i][j] = 255
        elif imgTemp[i][j] < 0:
            imgTemp[i][j] = 0
        imgTempResult[i][j] = int(imgGray[i][j]) + int(imgTemp[i][j])
        if imgTempResult[i][j] > 255:
            imgTempResult[i][j] = 255
        elif imgTempResult[i][j] < 0:
            imgTempResult[i][j] = 0

print('imSize: ', imgSize)
print('end')

cv2.imshow('Before', imgGray)
cv2.imshow('Average', imgTemp2)
cv2.imshow('Detail', imgTemp)
cv2.imshow('Result', imgTempResult)
cv2.waitKey(0)

```

#### 第四部份 Boundary Extraction 邊緣提取

```

import numpy as np
import cv2
import copy

def convWithAll(img, imgs, fils):
    imgTempf = copy.deepcopy(img)
    imgAfterConv = copy.deepcopy(img)
    for i in range(imgs[0]): #use i as index of image to determine which pixels
    need to be filtered
        for j in range(imgs[1]):
            imgAfterConv = conv(imgTempf, imgAfterConv, imgs, fils, i, j)

    for i in range(imgs[0]):
        for j in range(imgs[1]):
            imgTempf[i][j] = int(imgAfterConv[i][j]) - int(imgTempf[i][j])

    return imgTempf

def conv(imgOriginal, imgNew, imgSS, filSS, ii, jj): #imgOriginal - Original
Image, imgNew - New Image, ker - filter
    xCenter = int(filSS[0]/2) # if 3x3 kernel, xCenter = 1 from 0, 1, 2
    yCenter = int(filSS[1]/2)

    for l in range(filSS[0]): # if 3x3 kernel, l = 0, 1, 2

```

```
if ii + filSS[0] < imgSS[0]:
    for m in range(filSS[1]): # m = 0, 1, 2
        if jj + filSS[1] < imgSS[1]:
            if imgOriginal[ii + xCenter][jj + yCenter] > 0:
                imgNew[ii + 1][jj + m] = imgOriginal[ii + xCenter][jj +
yCenter]
        else:
            break
    else:
        break
return imgNew

image = cv2.imread('pic4.png')
imgGray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
imgTemp = copy.deepcopy(imgGray)
arr = np.array(imgTemp)
imgSize = arr.shape

print('imSize: ', imgSize)

fNum = 3
kernel = []
for i in range(fNum):
    kernel.append([1] * fNum)

fil = np.array(kernel)
filSize = fil.shape
print('kernel: ', kernel)
print('filSize: ', filSize)

imgTemp2 = convWithAll(imgTemp, imgSize, filSize)

print('imSize: ', imgSize)
print('end')

cv2.imshow('Before', imgGray)
cv2.imshow('After', imgTemp2)

cv2.waitKey(0)
```