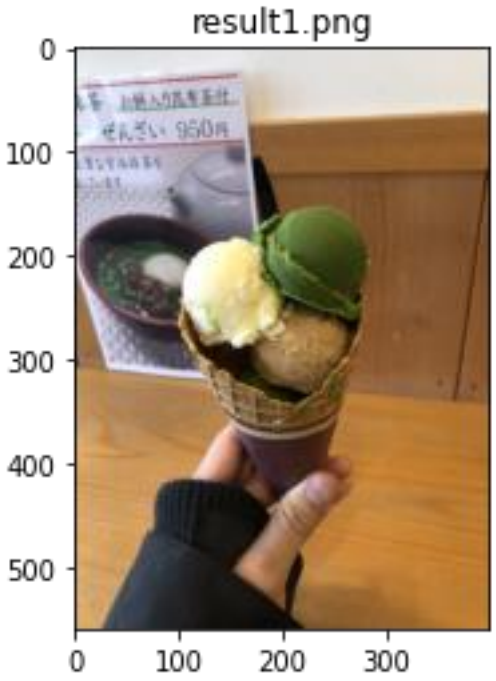
Homework number: DIP Homework Assignment #1

姓名: 方郁婷
學號: R10922196
Email: r10922196@ntu.edu.tw

Problem 0: WARM-UP

(a) (5 pt) Perform horizontal flipping on sample1.png and output the result as result1.png.



Motivation and Approach:

藉由 column 的交換來達到水平翻轉

```
height, width, channel = sample1.shape
result1 = sample1
# swap by column
for j in range(width//2):
    result1[:, [j, width-j-1]] = result1[:, [width-j-1, j]]
```

Discussion of results:

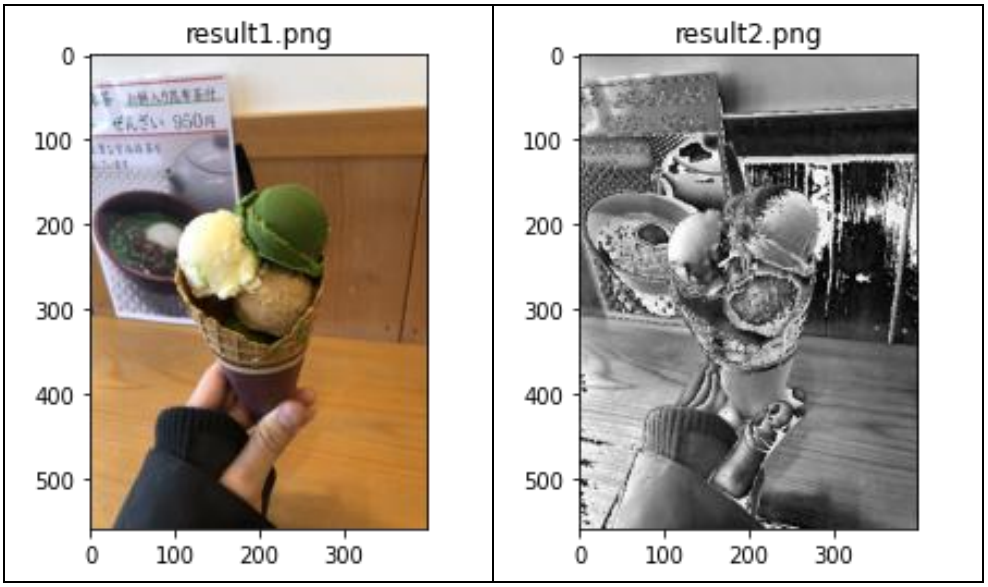
成功水平翻轉

(b) (5 pt) Please convert result1 to a grayscale image named result2.png.

Motivation and Approach:

- 比較直覺的作法，直接除以三

```
result2 = (result1[:,:,0]+result1[:,:,1]+result1[:,:,2])/3
```

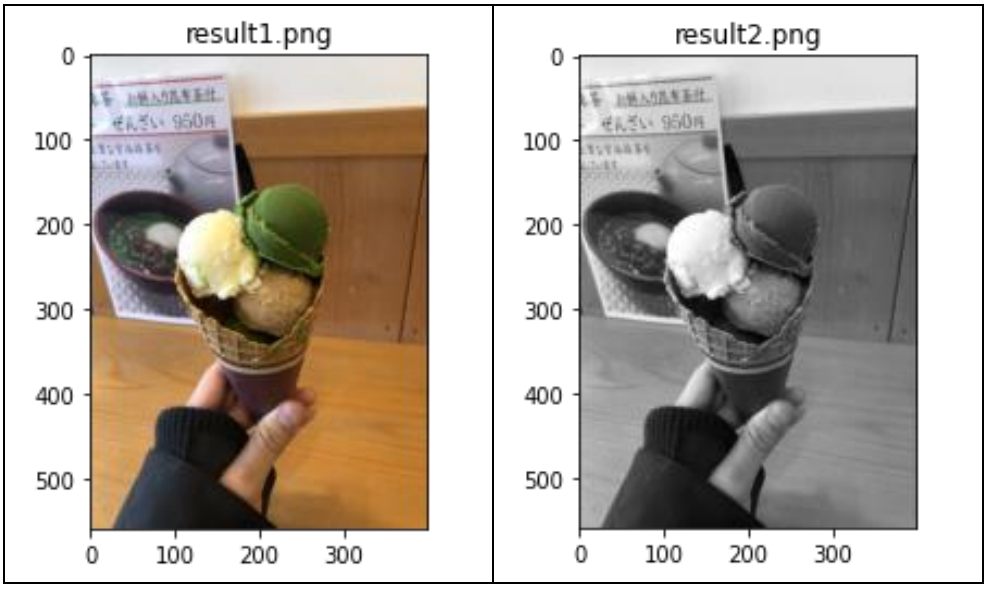


Discussion of results: 變得有點像鬼片特效

- 實際上人眼對綠色的亮度感最大，對藍色最小，所以以下才是最合適的比例

  Gray = 0.299 * Red + 0.587 * Green + 0.114 * Blue

```
result2 = 0.299*result1[:,:,0]+0.587*result1[:,:,1]+0.114*result1[:,:,2]
```



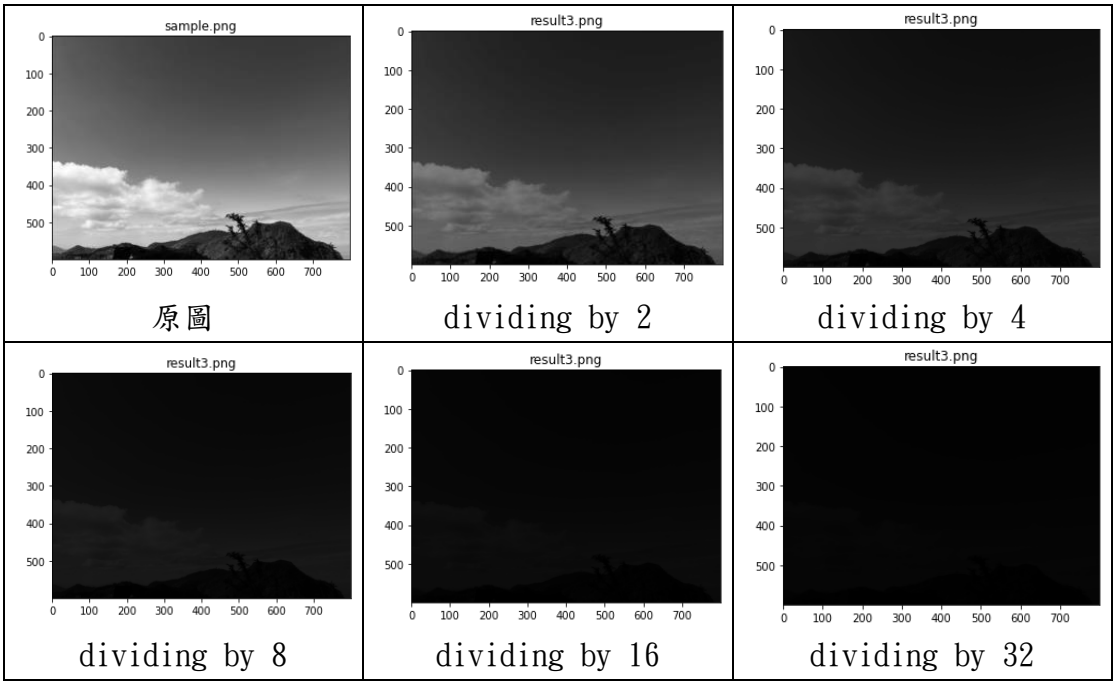
變得比較自然一點

Problem 1: IMAGE ENHANCEMENT

(a)(10 pt) Decrease the brightness of sample2 by dividing the intensity values by 2 and output the result as result3.png.

- 除完後要記得四捨五入取整數

```
result3 = sample2
result3 = result3/2
result3 = np.around(result3)
```



原圖 | dividing by 2 | dividing by 4

dividing by 8 | dividing by 16 | dividing by 32

圖片因為 value 變小而變暗了

(b) (10 pt) Increase the brightness of result3 by multiplying the intensity values by 3 and output the result as result4.png.

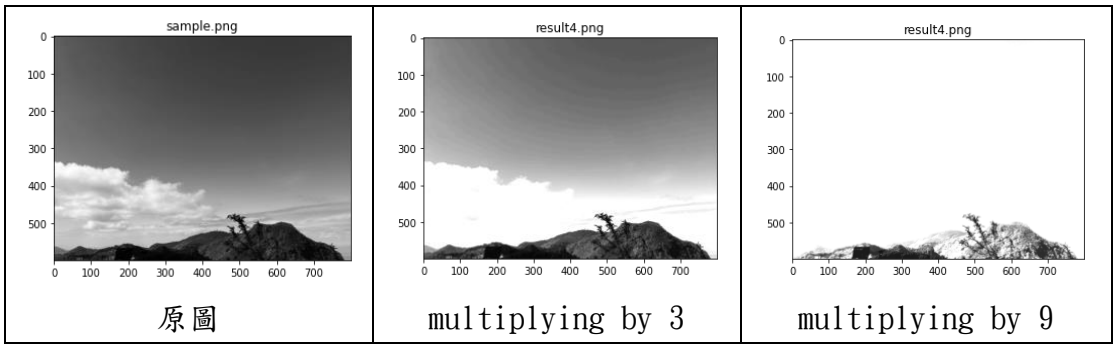
- 因為裡面有些值為 0，所以先加 1，不然乘了依舊還是 0

```
result3[result3 == 0]
```

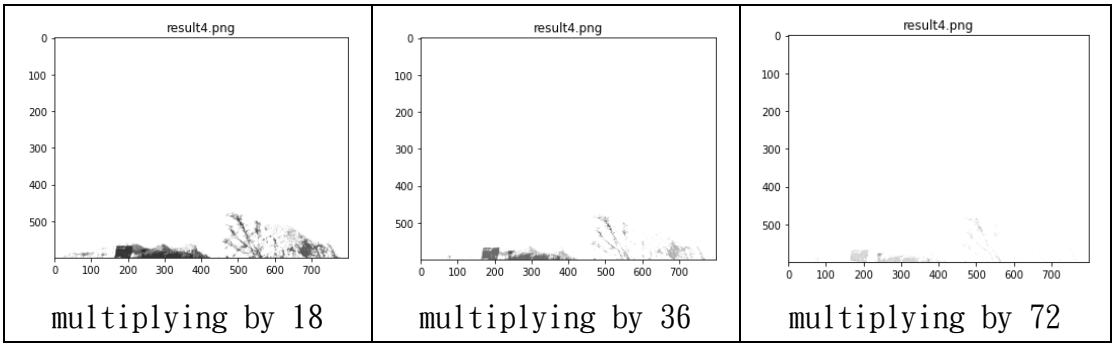
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

- 乘法要轉成小數以免溢位，四捨五入後要將超過 255 的數值 clip 掉

```
result4 = result3+1
result4 = result4*3.0
result4 = np.around(result4)
result4 = np.clip(result4,0,255)
```



原圖 | multiplying by 3 | multiplying by 9



multiplying by 18	multiplying by 36	multiplying by 72

圖片因為 value 變大而變亮了

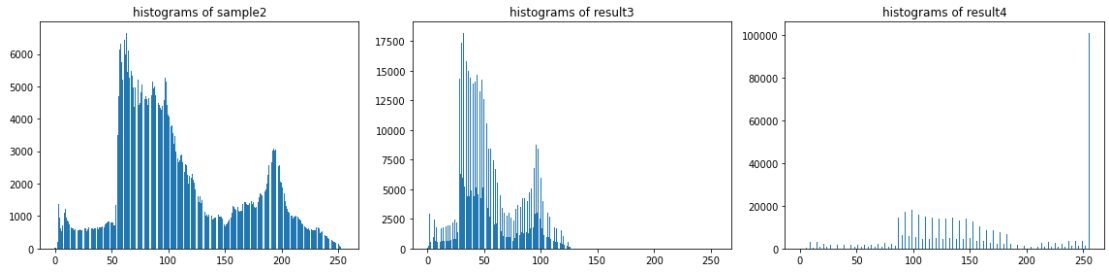
result4 在 multiplying by 3 時，雲的部分超過了 255，導致雲的細節消失了

(b)(10 pt) Plot the histograms of sample2, result3 and result4. What can you observe from these three histograms?

藉由 np.histogram 去計算[0, 255] 之間每個 value 存在的 pixel 個數
因為 bin 多了一個維度所以之後要去掉

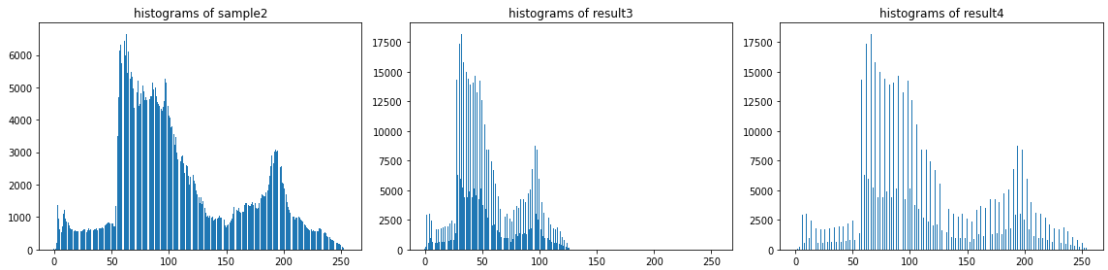
```
hist,bins = np.histogram(sample2,bins=256,range=(0,256))
print(hist.shape)
print(bins.shape) #多了一個維度
bins = bins[:-1]
# bins
```

```
(256,)
(257,)
```



- 原本的圖，每個 value 所擁有的個數都還算蠻多的
- Result3 經由除法後，value 變小，所以個數的值都往左集中，圖片變得比較黑、比較暗
- Result3 經由乘法後，value 變大，所以個數的值都往右集中(尤其是 value 為 255 的個數)，圖片變得比較白、比較亮

將 sample2 除 2 再乘 2 之後(reault4)



無法完全將 result4 復原為 smaple2，形狀是差不多的，但 result4 變得比較離散(因為有經過四捨五入和加 1 導致數據有些微改變　　)

(d) (10 pt) Perform global histogram equalization on result3 and result4, and output the results as result5.png and result6.png, respectively. Please compare these two resultant images and plot their histograms.

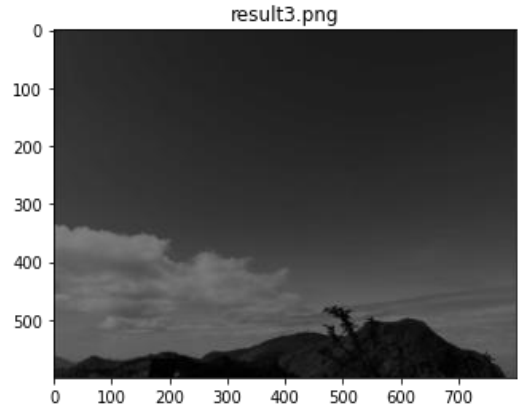
- 轉換函數

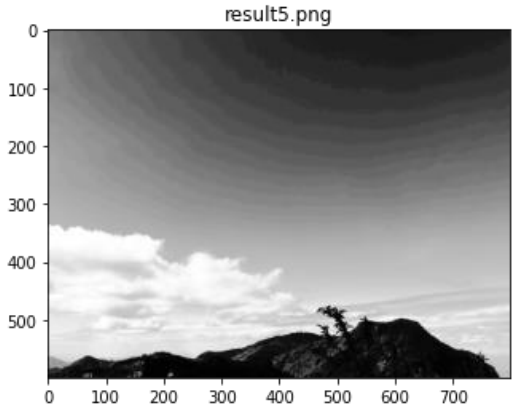
```
def globalHE(img):
    hist,bins = np.histogram(img.ravel(),256,[0,255])
    pdf = hist/img.size # 出現次數/總像素點 = 機率 = pdf
    cdf = pdf.cumsum() # 將機率利用cumsum()累加 = cdf
    equ_value = np.around(cdf * 255).astype('uint8') #將cdf乘以255(max value) ，再四捨五入取整數
    result = equ_value[img] #將原本的value數值transfer到對應的數值
    return result
```

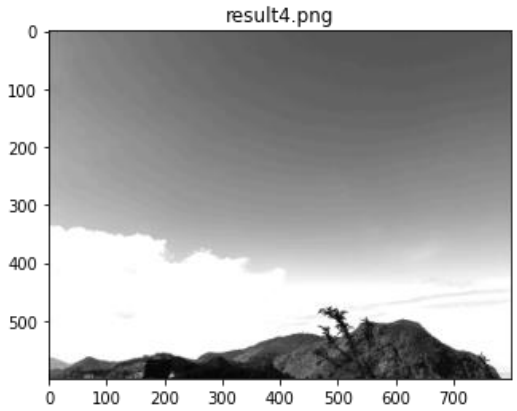
- 轉換後的結果

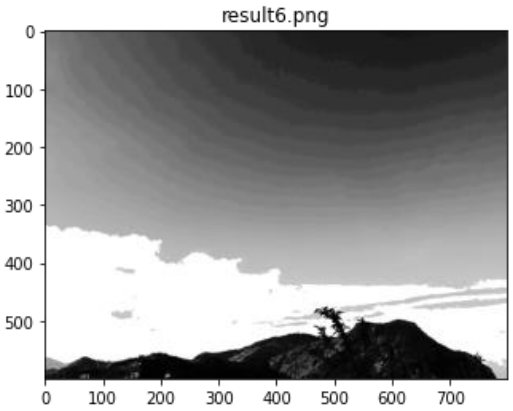
  對比度增強了(亮變更亮，暗變更暗)

- resultant images



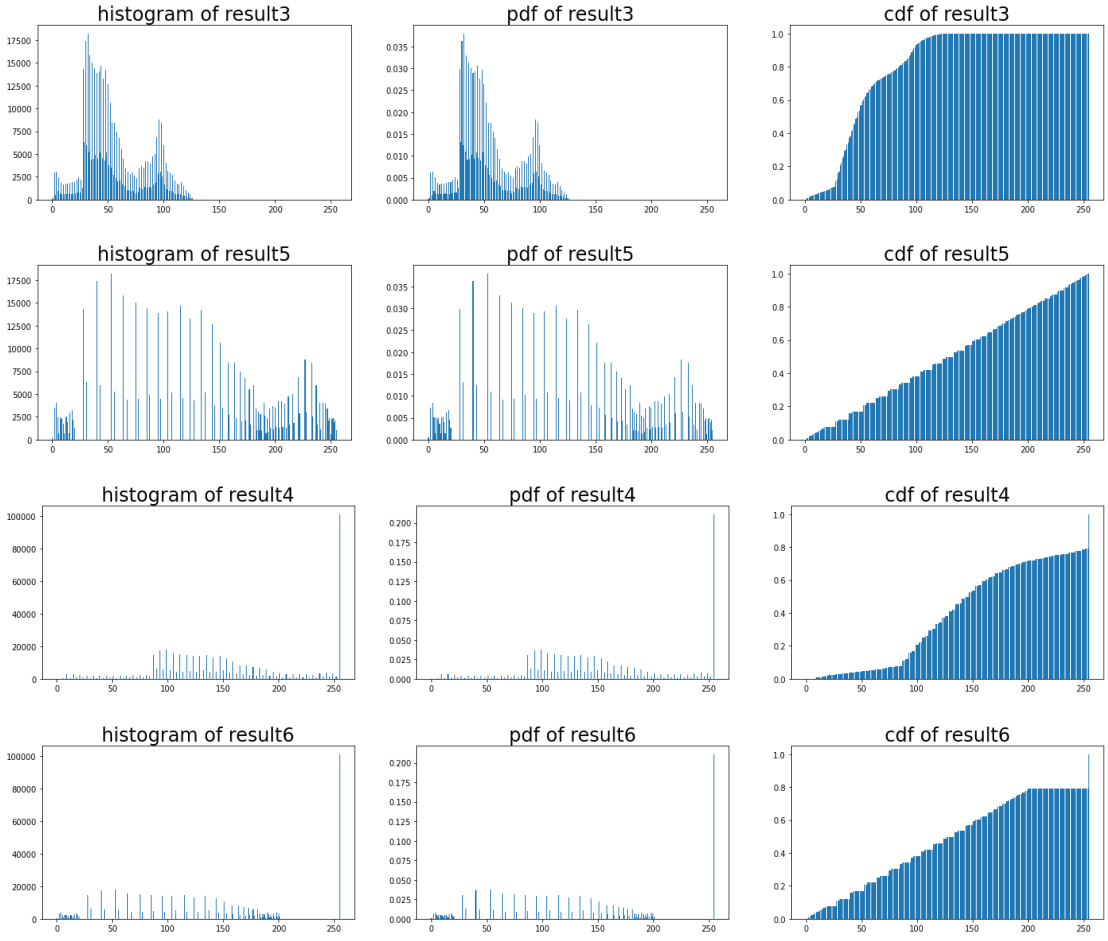






根據 result6，雖然對比增強了，但山的部分細節反而變得較不明顯

- histograms



- global histogram equalization 的原理

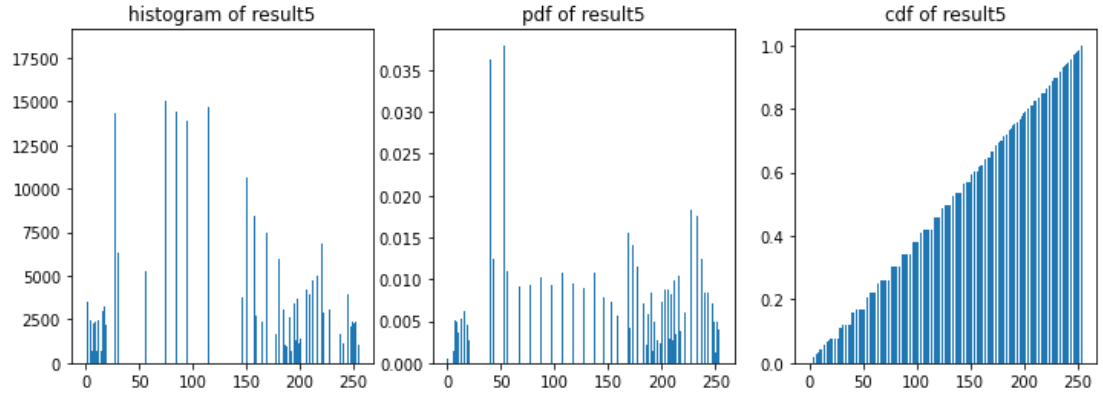
一開始想很久使用 CDF 再乘以 max value 的原理是什麼，看了轉後後的結果圖終於弄懂了，global histogram equalization 的目的是要把 CDF 的形狀排成整齊的三角形，所以會將每條 CDF 放到合適的位置，使得對邊(CDF 的值)與鄰邊(CDF * max value)是一個固定的比例，這樣排出來的 CDF 會近似成一個漂亮

的直角三角形。

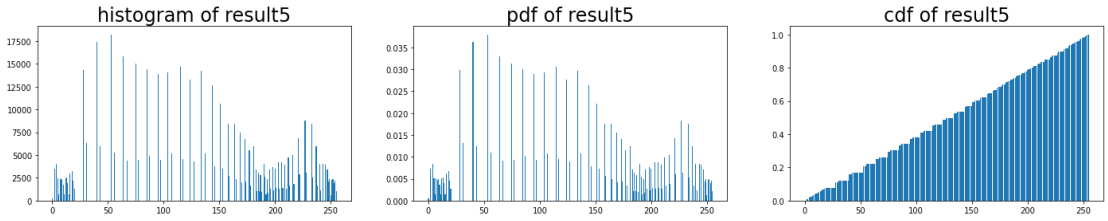
- 畫圖的 bug:

Histogram 和 PDF 的圖一直長得不一樣(照理來說形狀應該要一樣)，我檢查很多次，算出來的數值都沒錯，最後發現是因為圖片太擠所以柱狀圖被壓縮了(有些不會顯示出來)，將圖設大一點後就正常了。

plt.figure(figsize=(12,4))



plt.figure(figsize=(24,4))

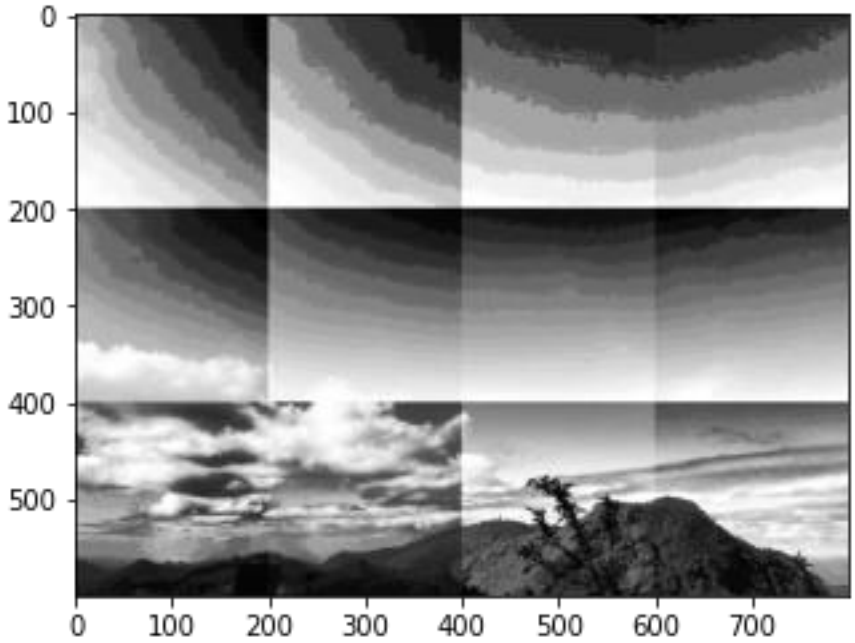


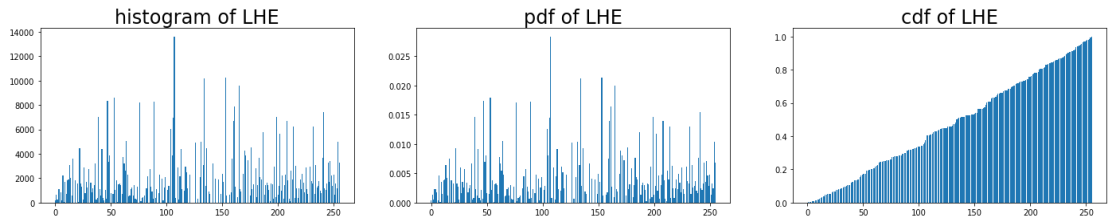
(e) (10 pt) Perform local histogram equalization on result3 and result4, and output the result as result7.png and result8.png, respectively. Please compare these two resultant images and plot their histograms.

作法一: 直接切成一塊一塊做 HE

```
for i in range(0,img.shape[0]-kernel_size,kernel_size):
    for j in range(0,img.shape[1]-kernel_size, kernel_size):
        #提取圖片的區塊(kernel_size x kernel_size)
        kernel = img[i:i+kernel_size,j:j+kernel_size]
        # HE
        hist,bins = np.histogram(kernel,256,[0,255])
        pdf = hist/kernel.size
        cdf = pdf.cumsum()
        plt.bar(bins[:-1],cdf)
        equ_value = np.around(cdf * 255).astype('uint8')

        kernel = equu_value[kernel]
        ime_ahe[i:i+kernel_size,j:j+kernel_size] = kernel
```

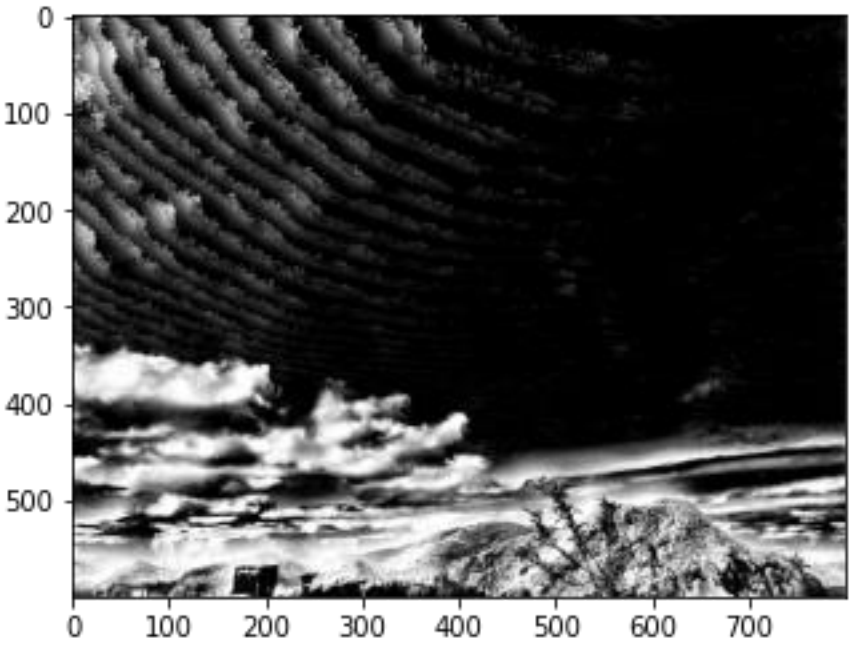
- 結果就是圖片會很明顯的出現一塊一塊的邊界(因為會個別把每個區塊的白放到最白，黑放到最黑)



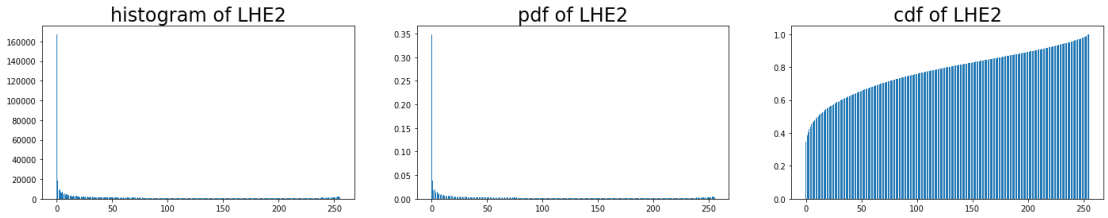


另一種方法

```python
# 遍歷圖片的每個pixels
for i in range(0,img.shape[0]-kernel_size):
    for j in range(0,img.shape[1]-kernel_size):
        #提取圖片的區塊(kernel_size x kernel_size)
        kernel = img[i:i+kernel_size,j:j+kernel_size]
        #由小到大排序
        kernel_flat = np.sort(kernel.flatten())
        #找到目前的pixel在這個區塊排第幾名
        rank = np.where(kernel_flat == img[i,j])[0][0]
        #排第幾名就獲取相對應排名的亮度
        ime_ahe[i,j] = int( max_value * ( rank / kernel_size_squared ) )
```

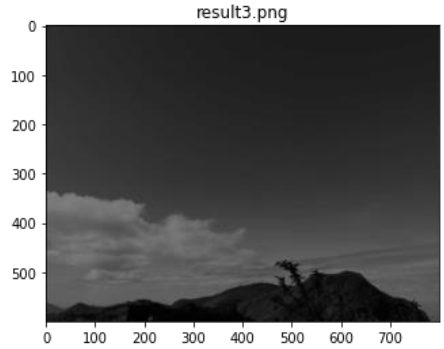


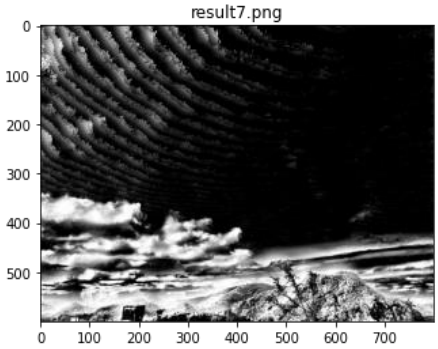
- 結果是天空會有漸層，有點像是抽象水墨畫，有一種獨特的風格

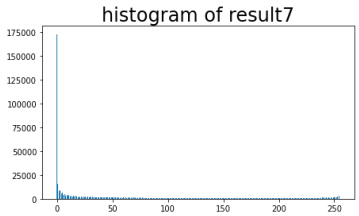


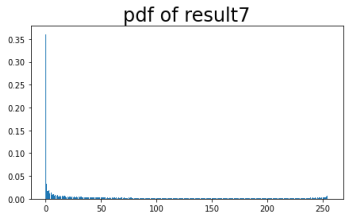
最後使用第二種方法，因為不會有格線，第一種方法如果不要有格線，可以像第二種方法一樣一次只移一個 pixel，但會花超級久的時間(第二種只需 10 秒)，所以最後還是選擇用第二種。

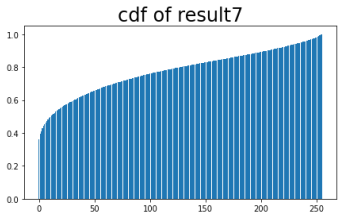
- 比較圖

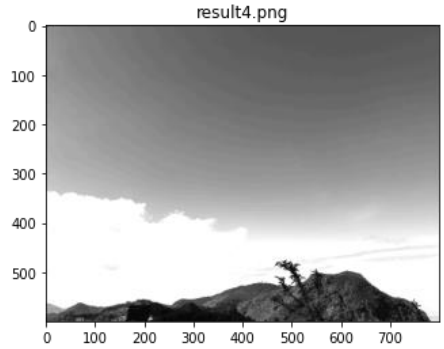


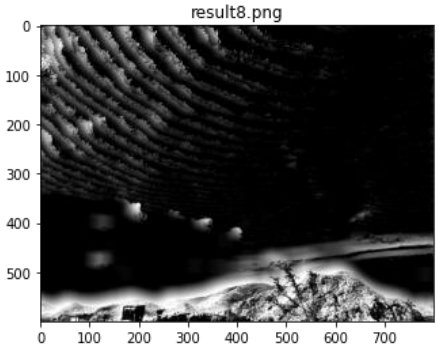


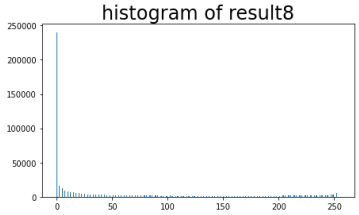


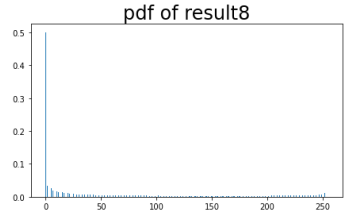


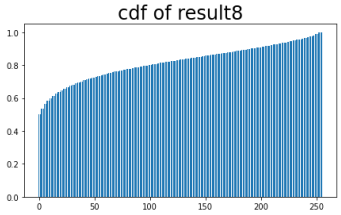








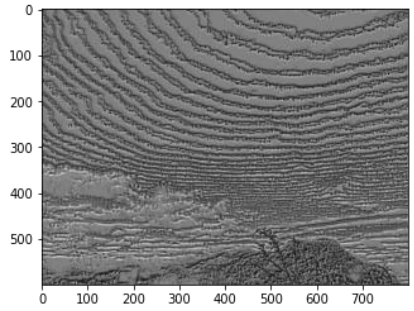




總結:

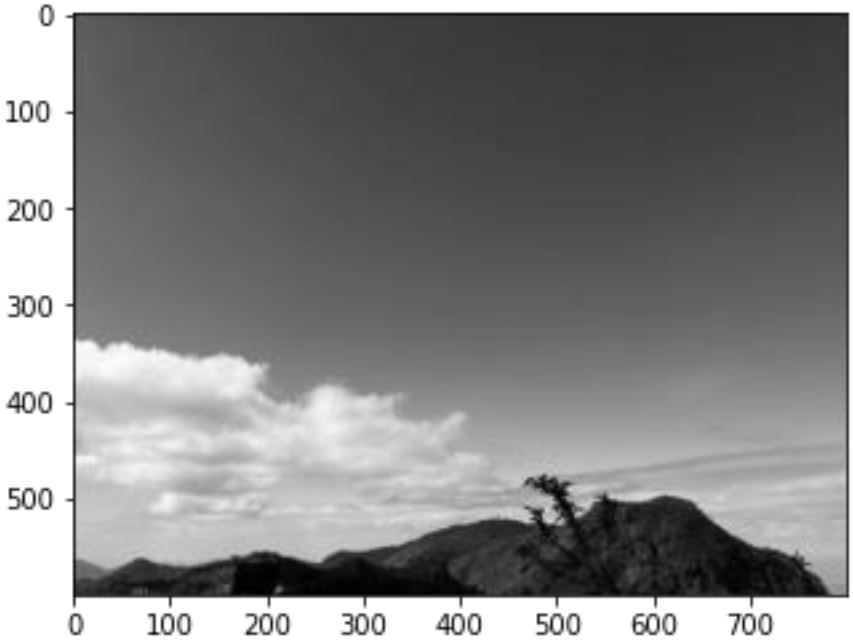
用 global HE 會有過白或過黑的情況，使用 local HE 則比較不會這樣，但以這個例子來看還是有過黑的情況(hustogram 的 0 的個數特別多)，而且天空漸層的線變得特別明顯，但山的紋路是變得比較清晰的。

- 補充: 這是看到討論區上傳的演算法實作出來的結果



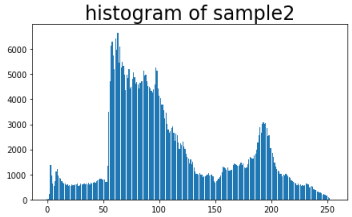
kernel 設 11，大概跑了 56 秒

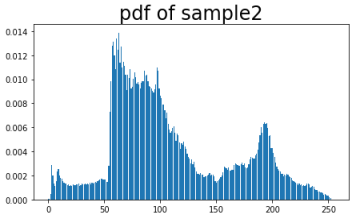
(f) (10 pt) Design a transfer function to enhance sample2 and output the result as result9.png. Try your best to obtain the most appealing result by adjusting the parameters. Show the parameters, the best resultant image and its corresponding histogram. Provide some discussions on the result as well.

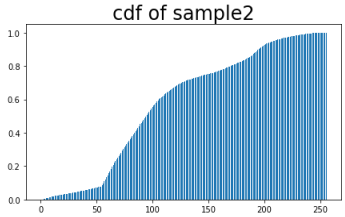


sample2

觀察:





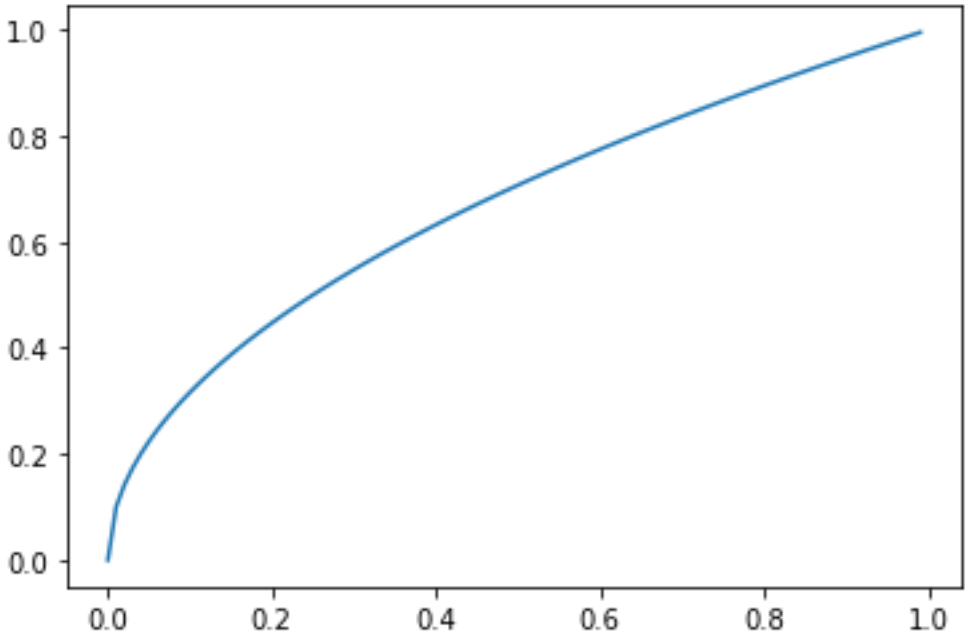


以 histogram 來說，整體是偏暗的，因此我使用了 ppt 裡提到的 Power-Law 來提升暗區的細節

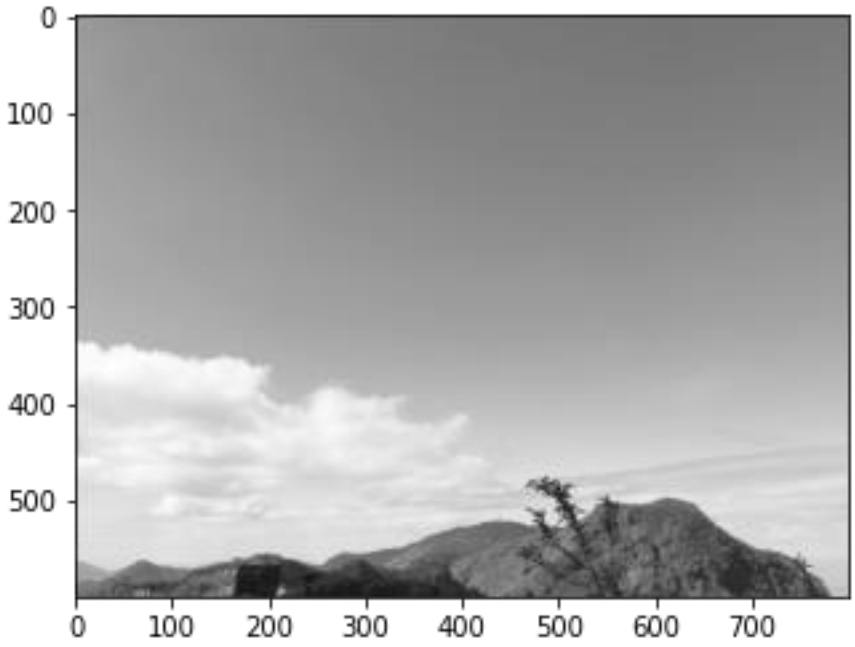
- **Power-Law**

$$G(j,k)=\left[F(j,k)\right]^{p} \quad 0\le F(j,k)\le 1$$

下圖是 p=0.5 時的轉換函數



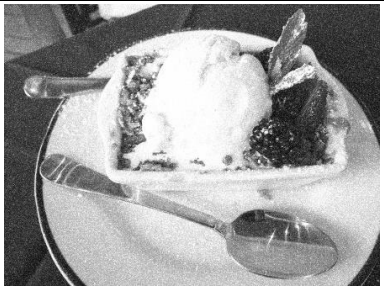
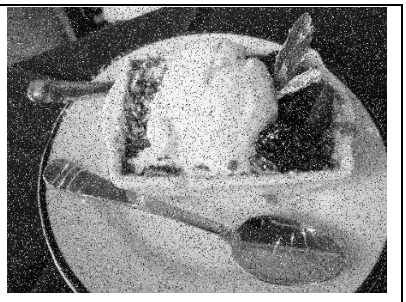
當 p<1 時，數值越小，提升的幅度就越大，因此暗區的數值轉換後會差異會變得比較大，反之，亮區的變化會變得比較小，經過幾次調整後，我覺得 p=0.5 是最好看的圖，因此我將 p 設為 0.5



result9

Problem 2: NOISE REMOVAL

(a) (20 pt) Design proper filters to remove noise on sample4.png and sample5.png. Output the clean images as result10.png and result11.png, respectively. Write down details of your noise removal process in the report, including the filters and parameters you use. Please also provide some discussions about the reason why those filters and parameters are chosen.

		
Sample3	Sample4	Sample5

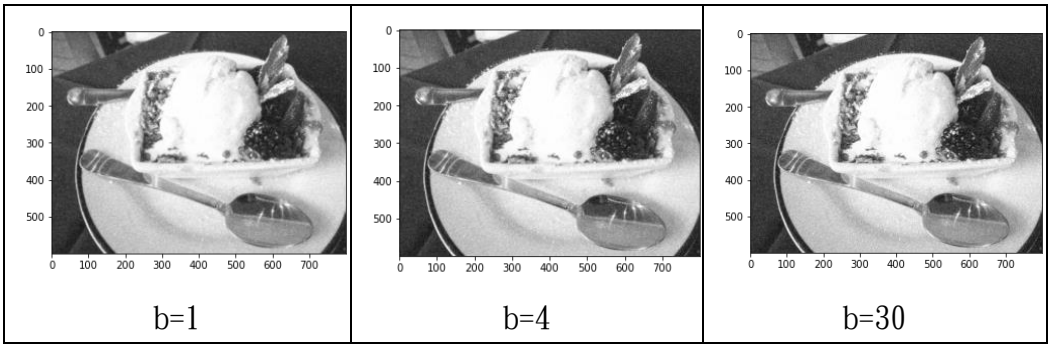
- sample4.jpg 的 noise 是 uniform noise，所以使用 low-pass-filter 進行去噪
  low-pass-filter 的原理是藉由平均模糊掉比較突出的 pixel(噪點)

General form

$$H = \frac{1}{(b+2)^2}\begin{bmatrix} 1 & b & 1 \\ b & b^2 & b \\ 1 & b & 1 \end{bmatrix}$$

其中 b 為一個可自己決定的變數

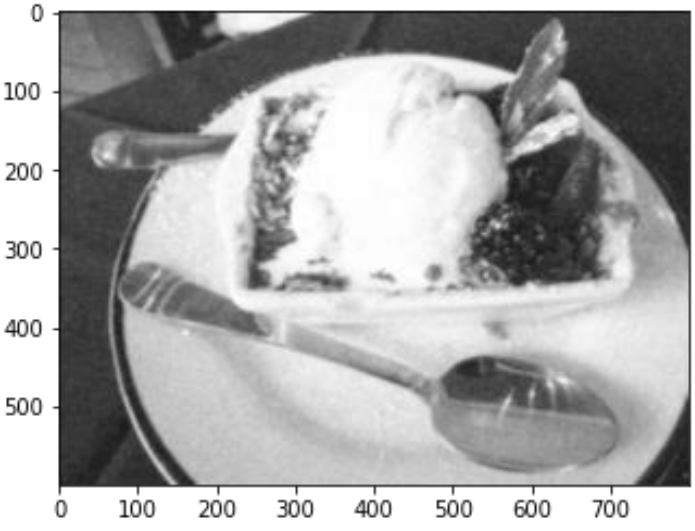
```
import math
b = 4
kernel = (np.array([[1, b, 1] ,[b, math.pow(b, 2), b] ,[1, b, 1]])) / math.pow(b + 2, 2)
kernel_size = kernel.shape[0]
# Padding
result10 = np.lib.pad(sample4, kernel.shape, 'reflect')
height, width = result10.shape

for i in range(kernel_size // 2, height - kernel_size // 2):
    for j in range(kernel_size // 2, width - kernel_size // 2):
        #將位置對應的pixel與kernel相乘後取sum
        p = (result10[i-1:i+2,j-1:j+2] * kernel).sum()
        result10[i,j] = p

# 將padding切掉
result10 = result10[kernel_size:height-kernel_size,kernel_size:width-kernel_size]
```



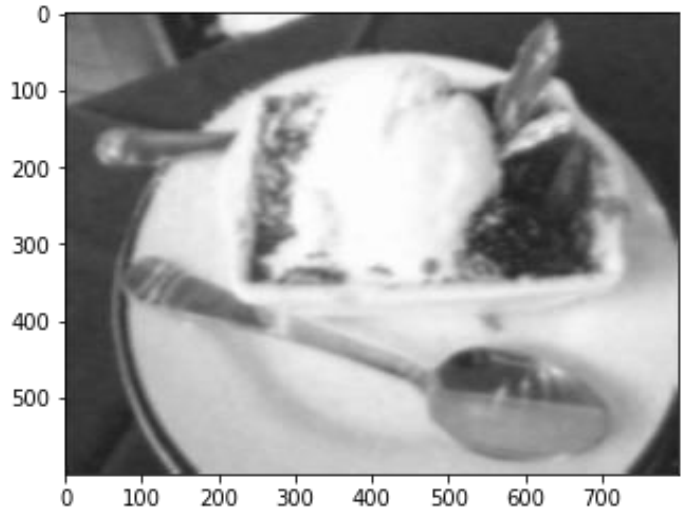
b=1 | b=4 | b=30

以結果來看 b 好像影響不大

將 b 設為 1 擴大 kernel 的結果



kernel_size = 5

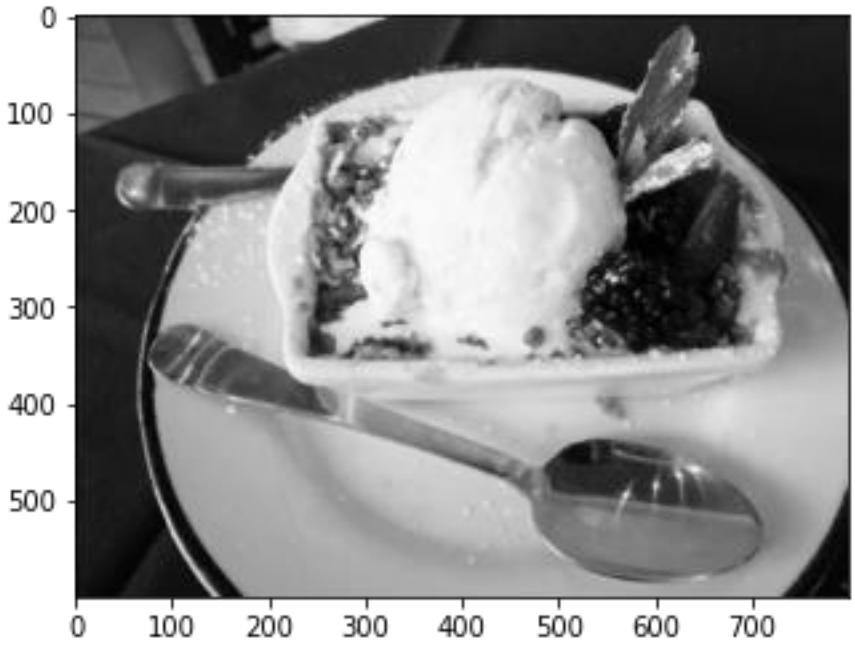


kernel_size = 11

隨著 kernel_size 變大，雖然雜訊變得比較不明顯，但圖變得比較模糊

- sample7.jpg 的 noise 是 椒鹽雜訊，所以使用 median-filter 進行去噪
因為椒鹽雜訊都是極值，如果使用平均，對周圍的 piexl 影響較大，所以不適合使用 low-pass-filter，也因為椒鹽雜訊都是極值，選中位數會選到極值的機率相當低，因此取中位數可以有效消除

```
kernel_size = 3
# Padding
result11 = np.lib.pad(sample5, (kernel_size,kernel_size), 'reflect')
height, width = result11.shape
gap = kernel_size // 2
for i in range(gap, height - gap):
    for j in range(gap, width - gap):
        #取kernel的中位數
        result11[i,j] = np.median(result11[i-gap:i+gap+1,j-gap:j+gap+1])

# 將padding切掉
result11 = result11[kernel_size:height-kernel_size,kernel_size:width-kernel_size]
plt.imshow(result11, cmap='gray',vmin=0, vmax=255)
```



看起來效果不錯

(b) (10 pt) In noise removal problem, PSNR is a good way to present the quality of your recovered image. Please compute PSNR values of result10.png and result11.png respectively, and provide some discussions.

- 使用 ppt 上的公式計算
  - **Peak signal-to-noise ratio (PSNR)**
    - **Mean squared error (MSE)**

$$MSE = \frac{1}{w*h}\sum_{j}\sum_{k}[F(j,k)-F'(j,k)]^2$$

    - **The PSNR is defined as**

$$PSNR = 10\times\log_{10}\left(\frac{255^2}{MSE}\right)$$

```python
def psnr(img1, img2):
    mse = np.mean((img1 - img2)**2)
    return 10 * math.log(math.pow(255, 2) / mse, 10)
```

PSNR of result10 = 27.677670250256202

PSNR of result11 = 35.9919614270816

總結:

Result10 的分數比 Result11 的分數還低，表示影像品質更差。

uniform noise 感覺比椒鹽雜訊更難清除，因為 uniform noise 比起椒鹽雜訊較不突出，要辨識此 pixel 是否為噪點時會比較困難，使用 low-pass-filter 進行平均時幾乎對所有的 pixel 都會有影響，與 sample3 會有較大的差異。