2024 Spring DIP HW1 Report

> DIP Homework Assignment #1
> Name: 陳德維
> ID: b10902138
> email: b10902138@ntu.edu.tw

Problem 0

(a) Perform vertical flipping on sample1.png and output the result as result1.png.

- 原理
  - 透過 `np.flipud(img)` 將各個row上下翻轉即可做到vertical flipping
- Result:

	
sample1	**result1**

(b) Transforming a color image to grayscale is a fundamental task in digital image processing and computer vision. Please generate a grayscale version of result1.png and save it as result2.png.

- 原理
  - 參考 https://en.wikipedia.org/wiki/Luma_(video) 中，**CCIR 601** (i.e. most digital standard definition formats) 的公式 $Y'_{601} = 0.299R' + 0.587G' + 0.114B'$
  - 因為人眼對於不同顏色的光的敏感度不一樣，因此使用加權的方式來計算相對亮度
- 實現方式

```python
def rgb2gray(img):
    return np.dot(img[...,:3], [0.1140, 0.5870, 0.2989])
```

因為是透過cv2讀入是存成BGR的形式，所以要用`[0.1140, 0.5870, 0.2989]`

- Result

	
result1	**result2**

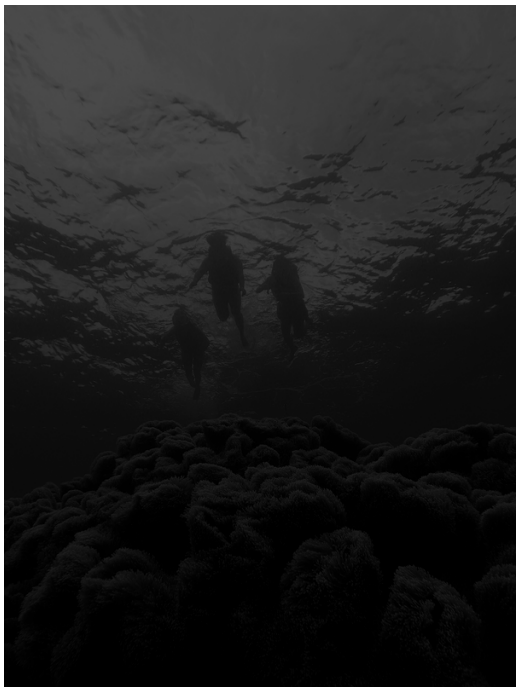
Problem 1

(a) Decrease the brightness of sample2.png by dividing the intensity values by 3 and output the result as result3.png.

- 原理
  - 對每一個pixel的value除以3，並clip進[0, 255]的範圍
- 實現方式

```
img = np.clip(img.astype(np.uint16) // 3, 0, 255).astype(np.uint8)
```

- Result

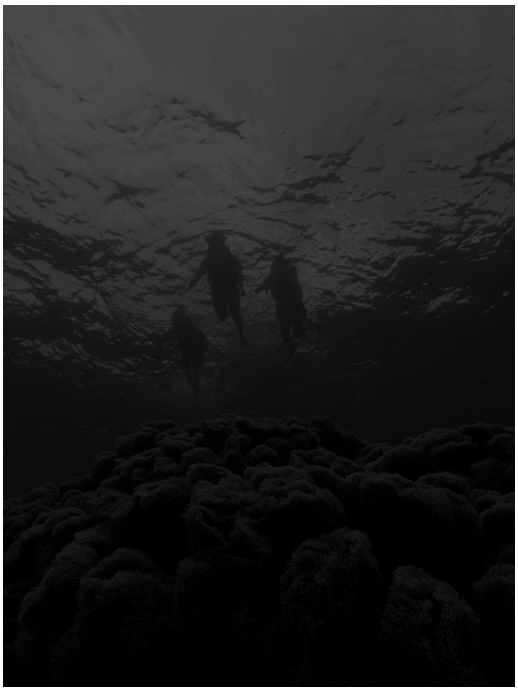
	
sample2	**result3**

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

- 原理
  - 對每一個pixel的value乘以3，再並clip進[0, 255]的範圍
- 實現方式

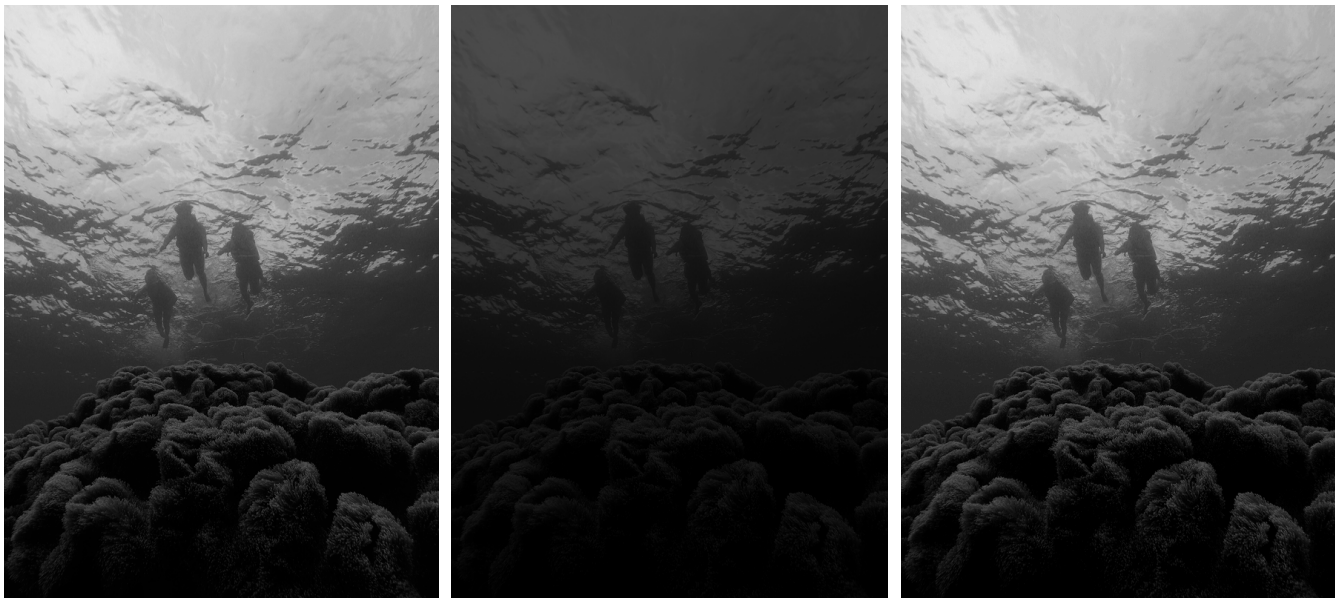
```
img = np.clip(img.astype(np.uint16) * 3, 0, 255).astype(np.uint8)
```

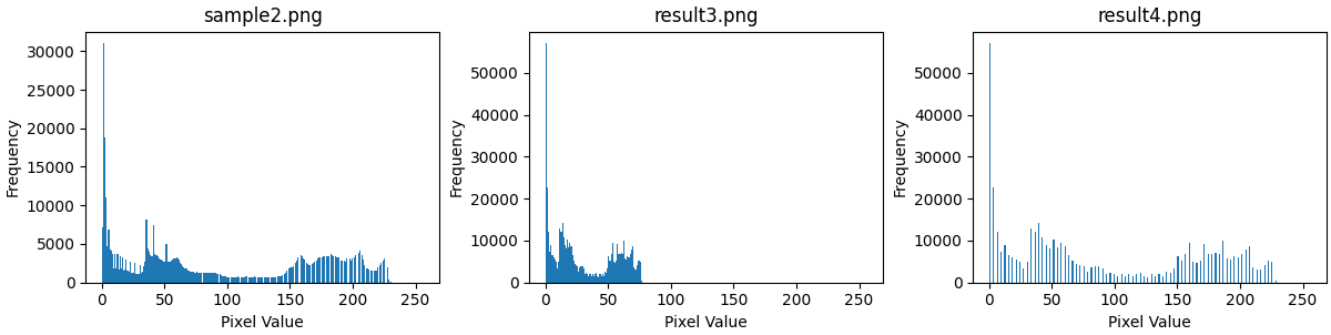
- Result

	
result3	**result4**

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

- Image & Histogram





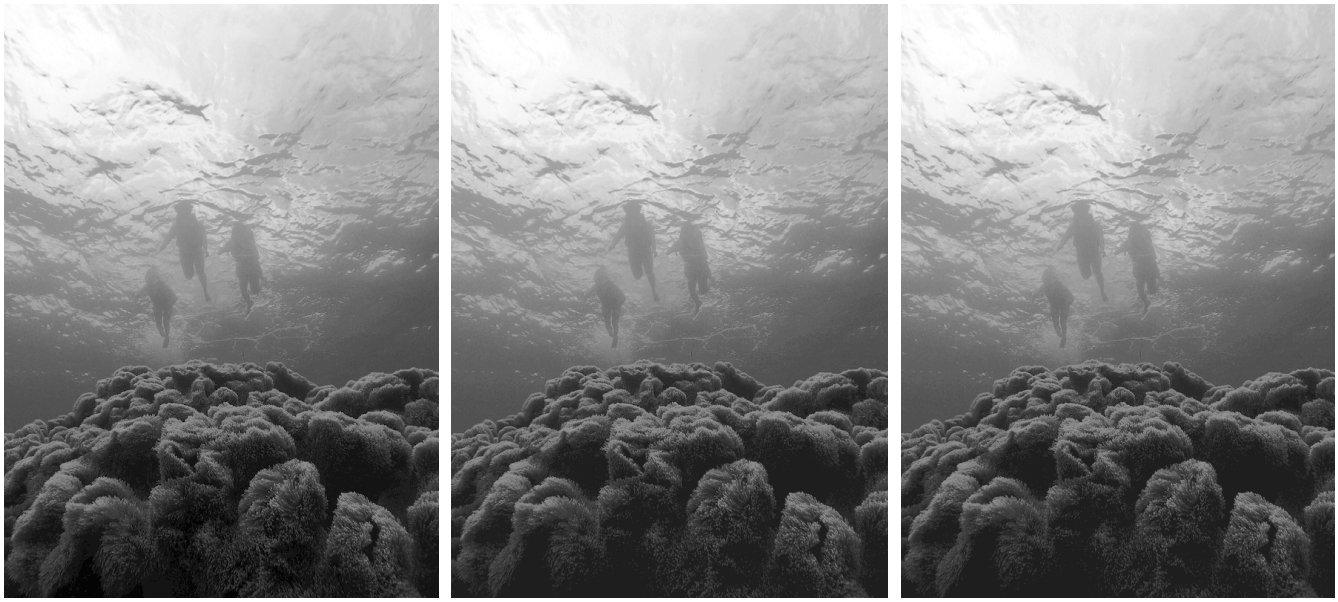
- 觀察
  - `result3.png` 是將 `sample2.png` 的像素value除以3，可以想成將像素分佈向左方壓縮，的確可以從上方直方圖中看出
  - `result3.png` 像素value為0的數量有將進60000，是因為我們做的是整數除法，因此原本value為0, 1, 2的像素都會變成0
  - `result4.png` 是將 `result3.png` 的像素value乘以3，可以想像成將像素分佈向右方拉長，可以看到像素的分布與 `sample2.png` 有相似的趨勢，但是我們可以看到 `resul4.png` 的分佈更**離散**了點，是因為原本value為 $3k$, $3k+1$, $3k+2$ 的像素經過整數除法3後再放大三遍都會變成 $3k$，因此有這樣的狀況。

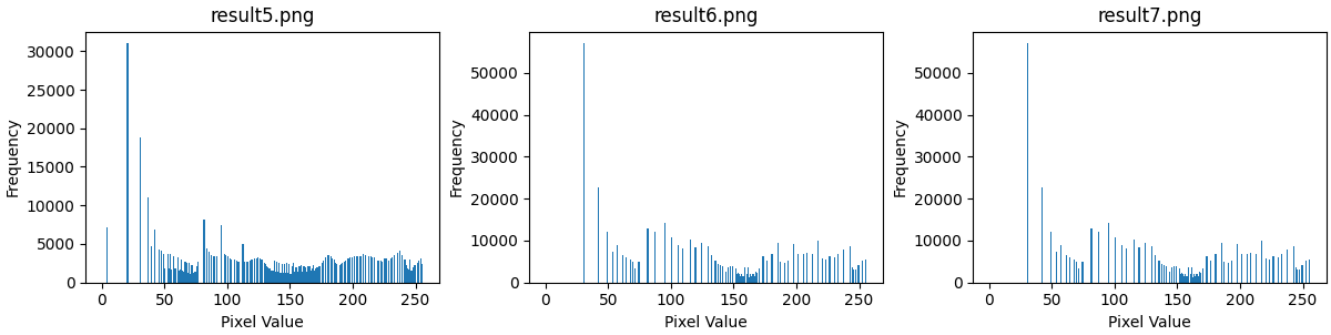
(d) Perform global histogram equalization on sample2.png, result3.png and result4.png, and output the results as result5.png, result6.png and result7.png, respectively. Please compare these three resultant images and plot their histograms.

- 實現方式 (Global histogram equalization)

```python
def global_histogram_equalization(img):
        hist, bin = np.histogram(img.flatten(), bins=256, range=[0, 256])
        pdf = hist/img.size
        cdf = np.cumsum(pdf)
        tranlate_table = np.round(cdf*255).astype(np.uint8)
        return tranlate_table[img]
```

- Image & Histogram





- 討論
  - 可以看出三者大致上很相似，但是 `result6.png` 和 `result7.png` 稍微模糊一點，彷彿有一層薄霧蓋在上面。推測原因是經過上題所描述放大縮小pixel value導致的失真，進而使得可以被map的pixel value種類減少，導致整體直方圖分佈較為離散。
  - 從直方圖上來看，GHE做的是希望能夠將cdf擺放的與uniform distrubtion時一樣，然而他並不能真的改變一張圖原本pixel的分佈狀況，因此我們可以看到pixel value的整體分佈趨勢仍與原本的差不多，而pixel value的確有稍微被更平均的map到$[0, 255]$
  - 與原圖 `sample2.png` 相比，圖片下方區域有明顯的提亮，而上方區域也不會過量，效果挺不錯

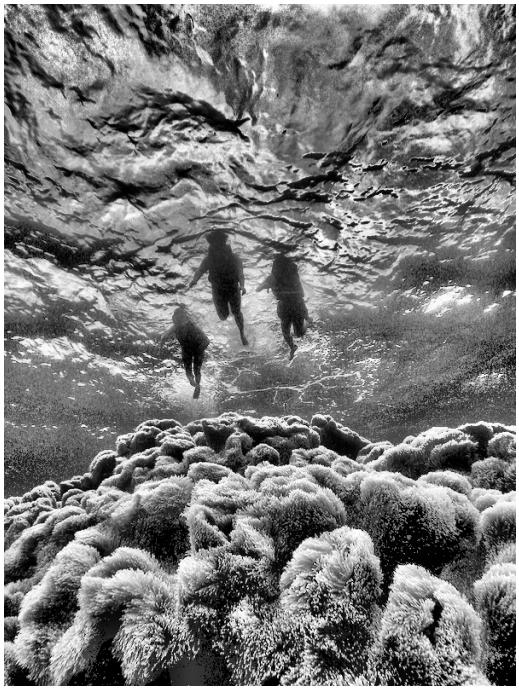
(e) Perform local histogram equalization on sample2.png. Output the results as result8.png and plot its histograms. Compare and discuss the results of global and local histogram equalization.

- 原理 (Local Histogram Equalization)
  - 對於每一個pixel，我們對他進行一個masking的動作，並算出這個pixel在以它為中心周遭所有元素之中他的pixel value排名(rank)，並用這個rank與這個mask裡的pixel個數的比值來內插進$[0, 255]$，當作他的最終pixel value

- 預期會產生的問題
  - 對於一大片相近顏色的區域，如果這個區域比mask還大，那麼會導致這片區域裡的pixel被map到$[0, 255]$之間，導致嚴重失真
- 實現方式

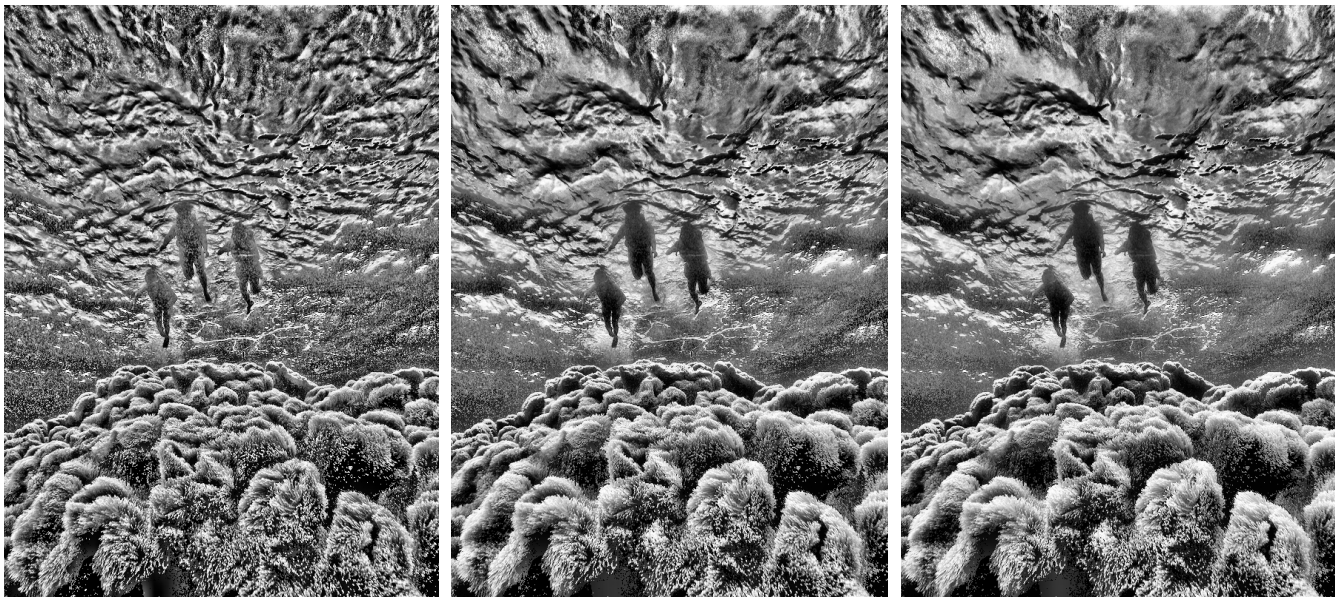
```
def local_histogram_equalization(img, mask_size=31):
    assert mask_size%2==1, "Mask size should be odd"
    h, w = img.shape
    block_size = (mask_size-1)//2
    padded_img = np.lib.pad(img, block_size, 'reflect')
    res = np.zeros(img.shape, dtype=np.uint8)

    # todo: O(n^2) -> O(n)?
    for i in range(h):
        for j in range(w):
            trans_x = i + block_size
            trans_y = j + block_size
            block = padded_img[trans_x-block_size:trans_x+block_size+1,trans_y-
block_size:trans_y+block_size+1]
            ranking = np.sort(block.flatten())
            rank = np.where(ranking == img[i,j])[0][0]
            res[i, j] = int(255 * (rank / (2*block_size+1)**2))
    return res
```

- Result (Best Effort)

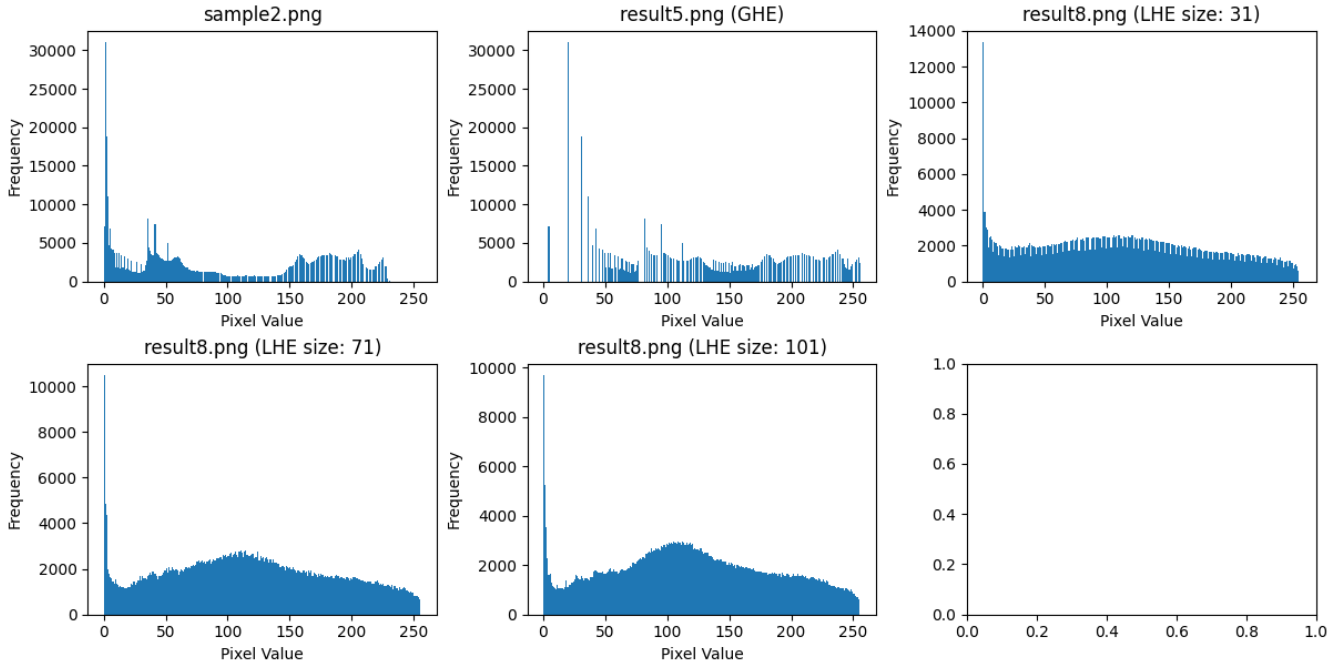
	
sample2	**result8 (Mask_size=101)**

```
![[result8.png | result8.png  (Mask_size=101) | 300]]
```

- Comparison



Mask size=31(left), Mask size=71(mid), Mask size=101(right)

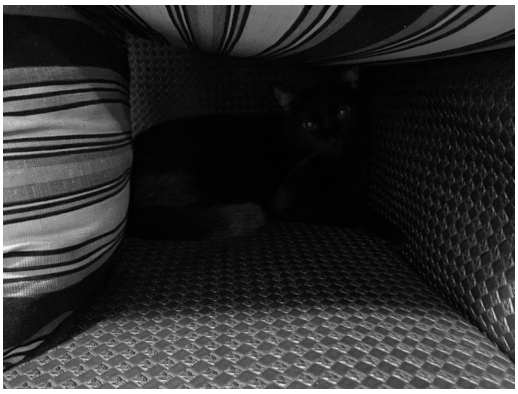
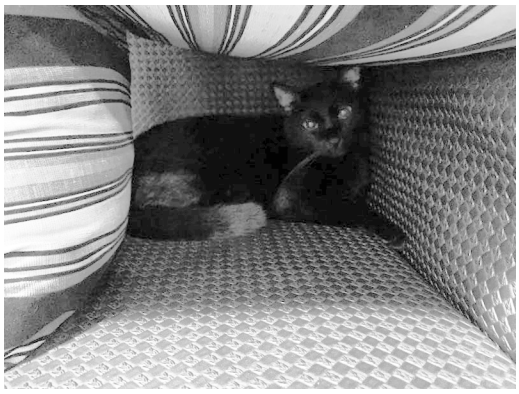


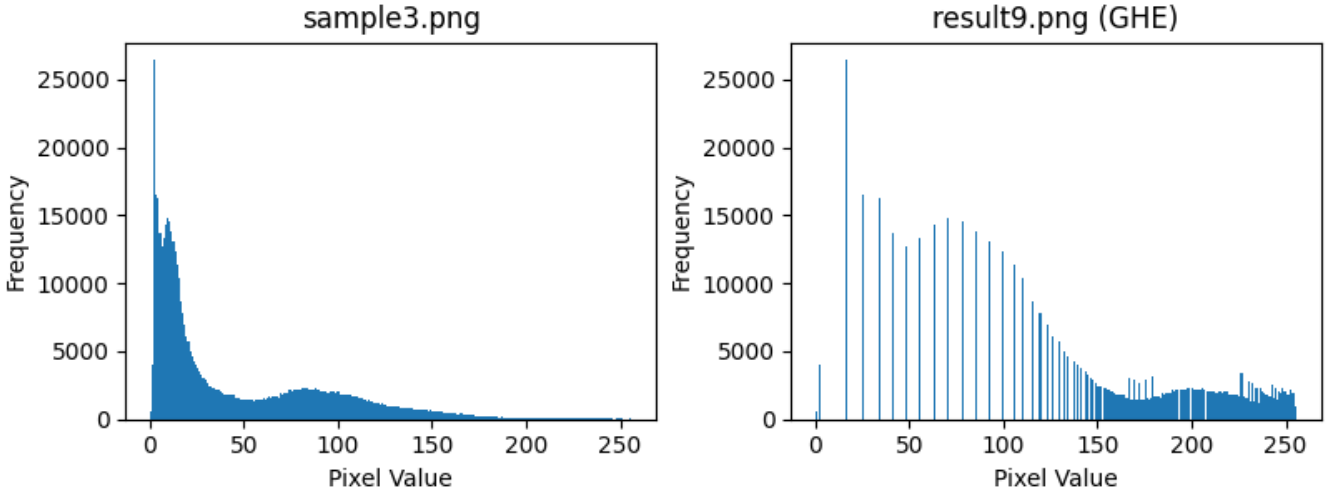
- 可以看到上方提到預期的問題發生了，由於大片海洋顏色相近，因此被處理的很像腦袋的皺褶...，但是我們可以透過提高mask size的大小，可以看到隨著mask size變大，失真的問題有稍微減緩了
- 直方圖的部分仍然與GHE的問題一樣，透過內插的方法仍會受到原本的分佈影響，並不能真正的做到uniform distribution，從實驗結果來看mask size越小越平均
- 實作方式目前我採用$O(n^2)$的方式，導致要跑很久，應該是可以優化成$O(n)$，或是改用CLAHE (Contrast Limited AHE)

(f) TA's cat is playing hide and seek and seems to be missing! Please design a transfer function to enhance sample3.png 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.

- Intuition
  - 將 `sample3` 的直方圖畫出後觀察，可看到大部分的pixel value集中在暗部，少部分集中在中間，因此可以推測若能將分佈往右拉(調亮)，但又不把所有的暗部都調亮，就可以將這隻貓與背景分開了
  - 最簡單的方法直接將照片直接乘以3，但是這樣暗部也會跟著變亮，效果不會很好

    - 因此，我最後選用GHE，既能將分佈往亮部拉，又能維持整個分佈的趨勢
- Result (Global Histogram Equalization)

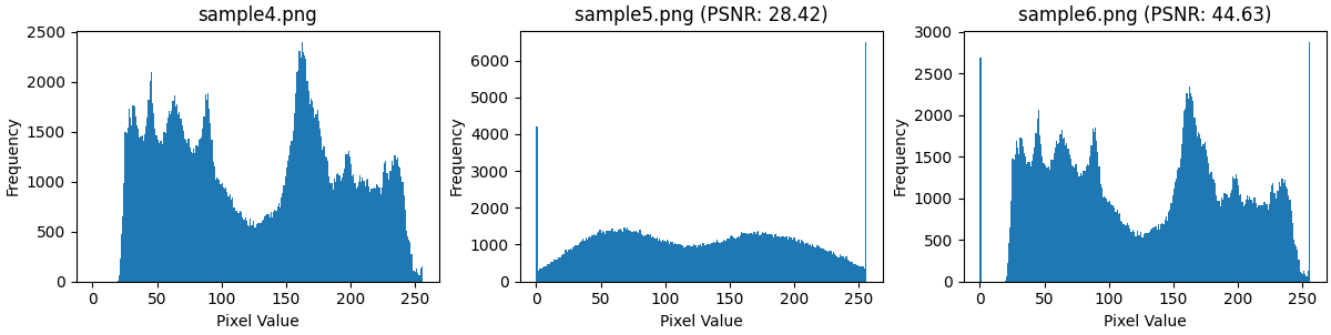
	
sample3	**result9**



Problem 2

(a) Design different filters to remove the noise in sample5.png and sample6.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 used, along with discussions on their selection rationale.

- Sample Image Histogram



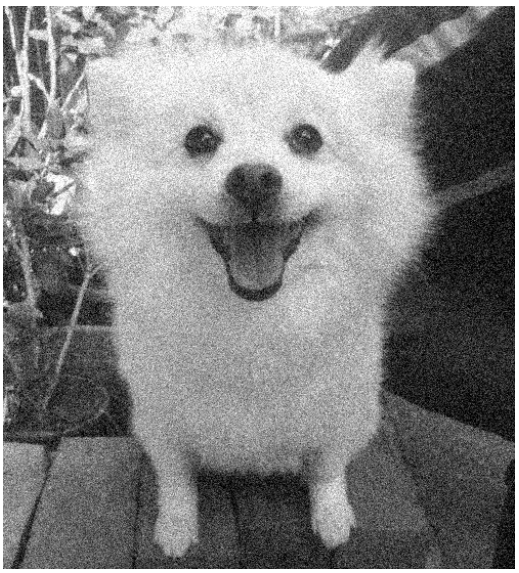
- `sample5.png`
    - 觀察
        - 可以從直方圖觀察出這張照片的noise應該是Gaussian Noise

- 應對策略
  - 使用 low-pass filter 做 spacial filtering
- 實現方式

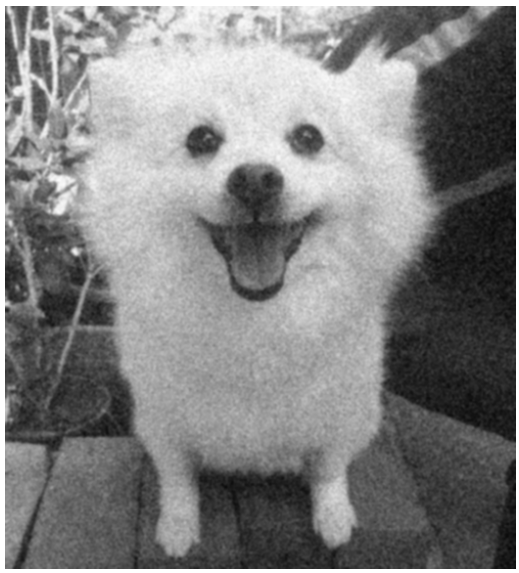
```
def spacial_filtering(img, filter):
    mask_size, _ = filter.shape
    assert mask_size==_, "Filter should be square matrix"
    assert mask_size%2==1, "Filter size should be odd"

    h, w = img.shape
    result_img = np.zeros((h, w), dtype=np.uint8)
    padded_img = pad_image(img, mask_size=mask_size)

    for i in range(h):
        for j in range(w):
            result_img[i,j] = np.round(np.sum(filter * padded_img[i:i+mask_size,
j:j+mask_size]))
    return result_img
```

- Result (Best Effort)
  - 使用 5x5 的 low-pass filter

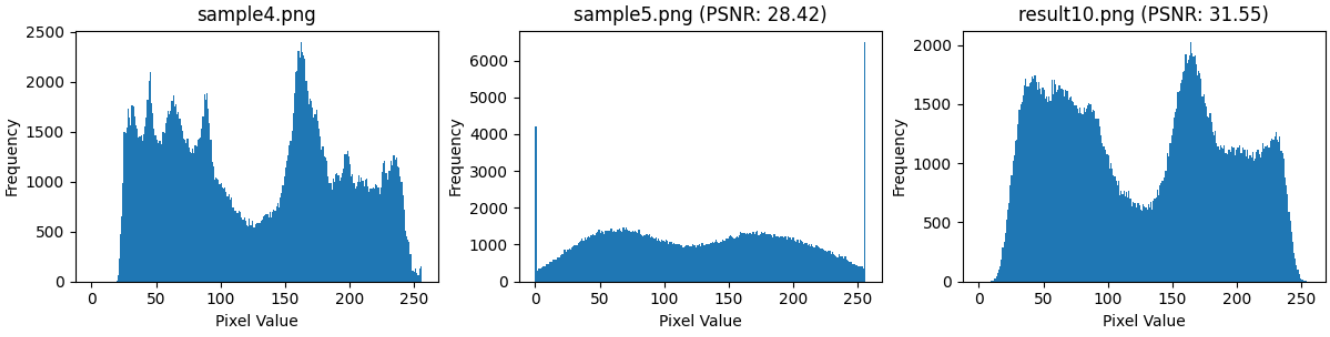
$$\frac{1}{256}\begin{bmatrix}1, & 4, & 6, & 4, & 1\\ 4, & 16, & 24, & 16, & 4\\ 6, & 24, & 36, & 24, & 6\\ 4, & 16, & 24, & 16, & 4\\ 1, & .4, & 6, & 4, & 1\end{bmatrix}$$



sample5

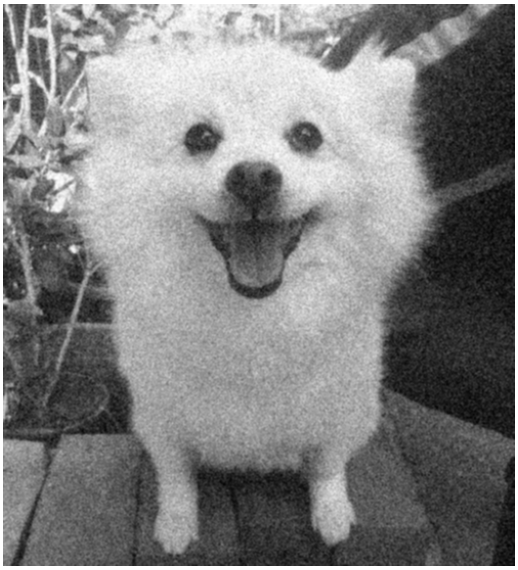


result10



- 觀察
  - 可以從直方圖觀察到不僅分佈圖的趨勢與ground-truth接近許多，且黑白的outlier消失了
  - 仍然感覺有許多noise, `PSNR` 也僅有31 (雖然老師上課說31就算合格了)
- 比較
  - Using 3x3 general form low-pass filter, result in more noise.

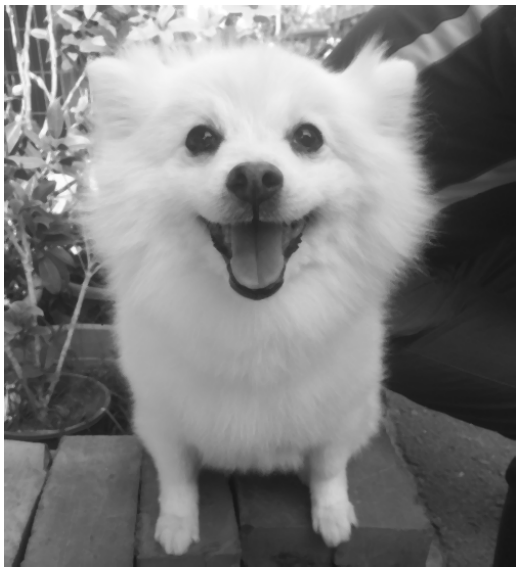
$$\frac{1}{16}\begin{bmatrix}1, & 2, & 1\\ 2, & 4, & 2\\ 1, & 2, & 1\end{bmatrix}$$

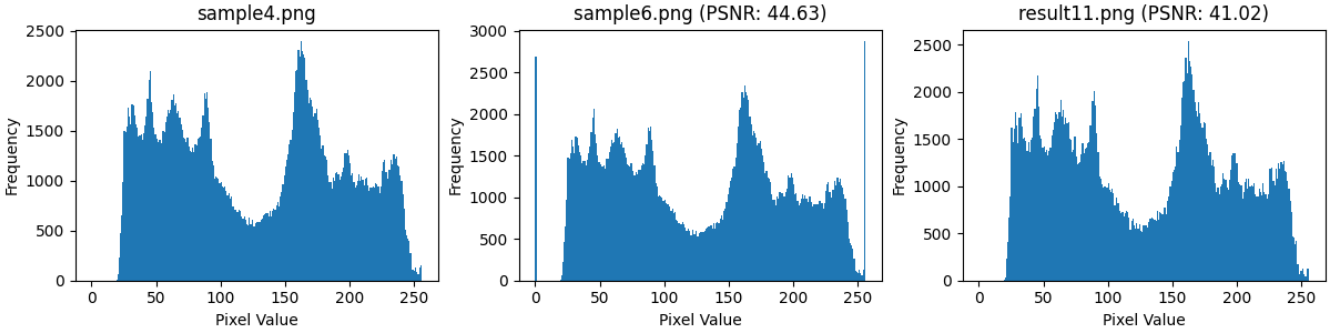
	
result10	**3x3 general form low-pass filter**

- `sample6.png`
  - 觀察
    - 可以從直方圖中觀察到大致分佈走向仍存在，僅有黑白的outlier特別突出，因此可以判斷出這張照片的noise應該是 Salt & Pepper Noise
  - 應對策略
    - 對每個pixel做 Median filtering
  - 實現方式

```
def median_filtering(img, kernel_size: int):
    assert kernel_size%2==1, "Kernel size should be odd"
    h, w = img.shape
    result_img = np.zeros((h, w), dtype=np.uint8)
    padded_img = pad_image(img, mask_size=kernel_size)
    for i in range(h):
        for j in range(w):
            result_img[i, j] = np.round(np.median(padded_img[i:i+kernel_size,
j:j+kernel_size]))
    return result_img
```

- Result (Best Effort with kernel size 3)

	
sample6	**result11**



- 觀察
  - 可以從直方圖觀察到黑白的outlier消失了，整體分佈也維持的很好
  - 雖然照片看起來清楚很多，但是 `PSNR` 卻降低了。推測原因是因為對每個pixel取median後，雖然outlier會被消滅掉，但是也會影響到原本正常的pixel，且 `PSNR` 僅考慮MSE，因此這種稍微失真的狀況會被考慮進去，因此 `PSNR` 是有可能變低的。
  - 解決方法：我們可以設一個threshold，大於這個threshold再當成是outlier

(b) In noise removal problems, PSNR is a widely used metric to present the quality of the recovered image. Please compute PSNR values of result10.png and result11.png, respectively, and provide some discussions.

- PSNR 公式

$$PSNR = 0 * log_{10}(\frac{255^2}{MSE}),\ \ MSE = \frac{1}{w*h}\sum_j\sum_k[F(j,k) - F^{'}(j,k)]^2$$

- 實現方式

```
def calc_psnr(ground_truth, img):
    return 10 * np.log10(255**2 / np.mean(np.square(img-ground_truth)))
```

- Result

```
sample5  PSNR: 28.420855108965377
result10 PSNR: 31.54730788751785
sample6  PSNR: 44.632682657685365
result11 PSNR: 41.022472270576955
```

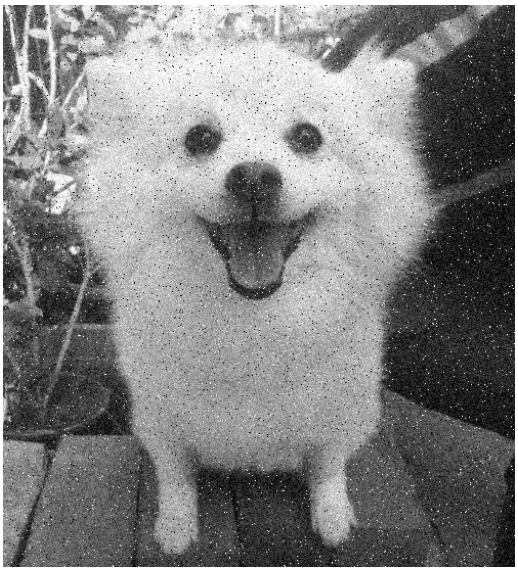
- 討論
  - `PSNR` 越高表示Noise越少
  - `sample5` 和 `result10` 的結果符合我們的預期，`PSNR` 提高
  - `sample6` 和 `result11` 的結果上一題已經討論過了，是因為對每個pixel取median後，雖然outlier會被消滅掉，但是也會影響到原本正常的pixel，且 `PSNR` 僅考慮MSE，因此這種稍微失真的狀況會被考慮進去，因此 `PSNR` 變低。

(c) Design an algorithm to remove the noise from sample7.png. Output the clean image as result12.png. Additionally, calculate the PSNR value of the resultant image. Provide the details of your noise removal process in the report, and provide some discussions regarding to the result.

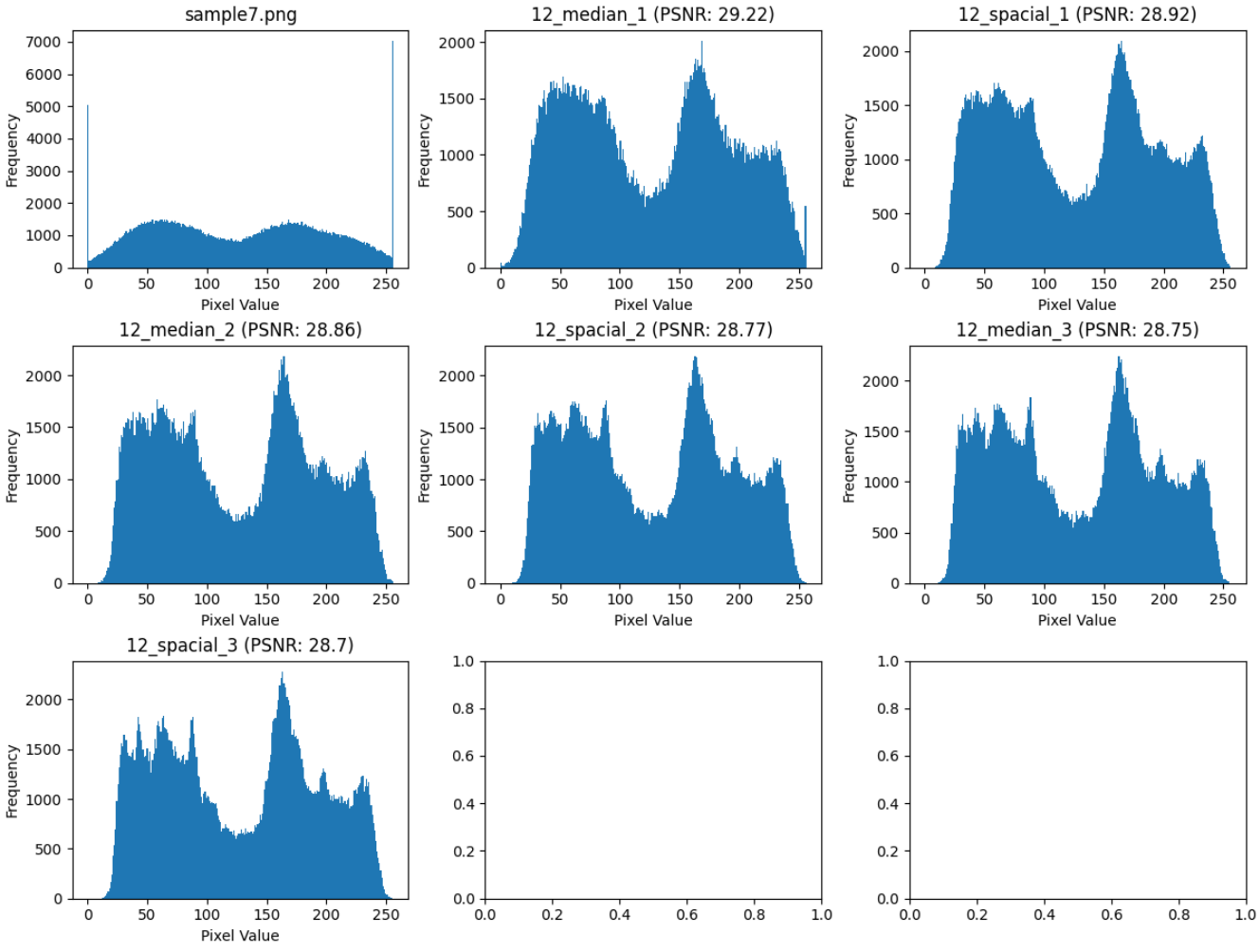
- 觀察
  - 直接看這張照片的noise應該是Gaussian Noise + Salt & Pepper Noise 兩者都有
- 應對策略
  - 先使用Median Filtering來處理 Salt & Pepper Noise，再使用 low-pass filter 做 `spacial filtering` 來處理 Gaussian Noise
  - 重複以上動作數次
- Parameter:
  - `Total Iterations`: 3
  - `Median Filtering`: Kernel size = 3
  - `Spacial filtering`: with 5x5 的 low-pass filter

$$\frac{1}{256}\begin{bmatrix}1, & 4, & 6, & 4, & 1\\ 4, & 16, & 24, & 16, & 4\\ 6, & 24, & 36, & 24, & 6\\ 4, & 16, & 24, & 16, & 4\\ 1, & .4, & 6, & 4, & 1\end{bmatrix}$$

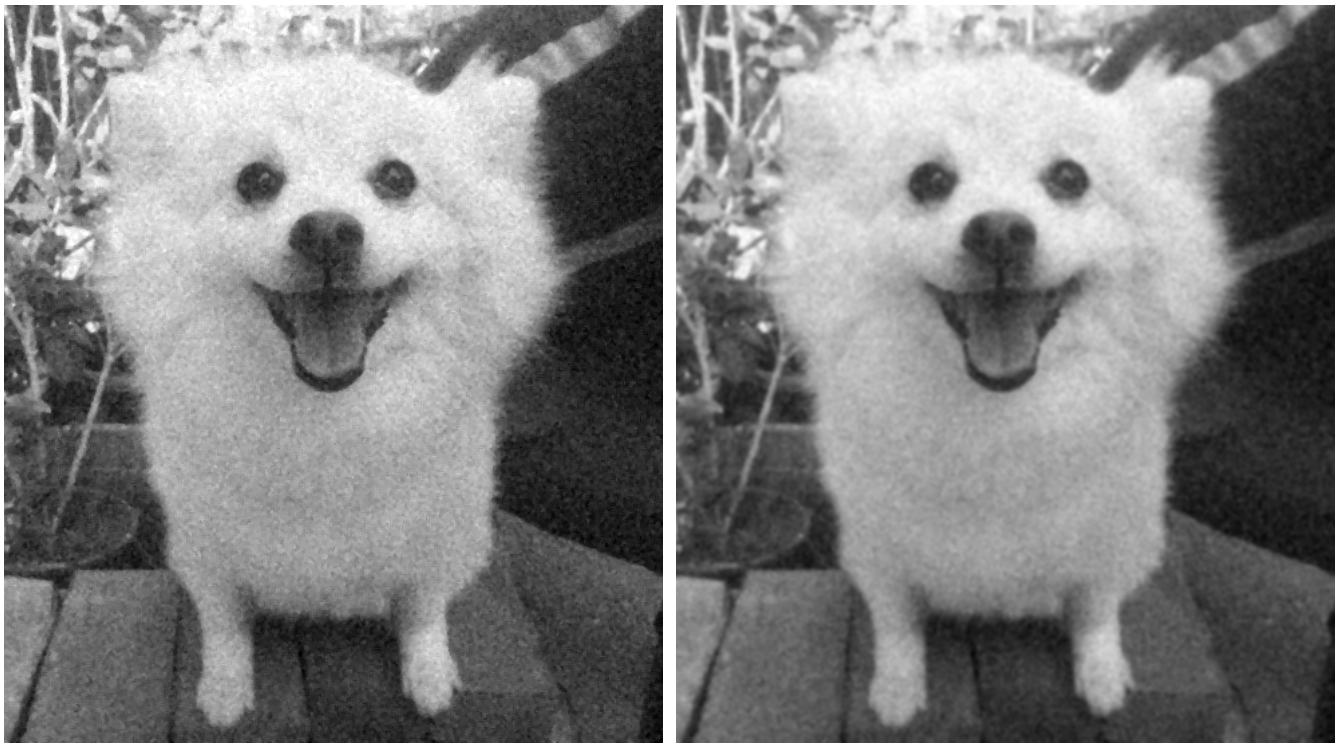
- Result
  - `PSNR` = 28.70

	
sample7	**result12**

Each Iteration

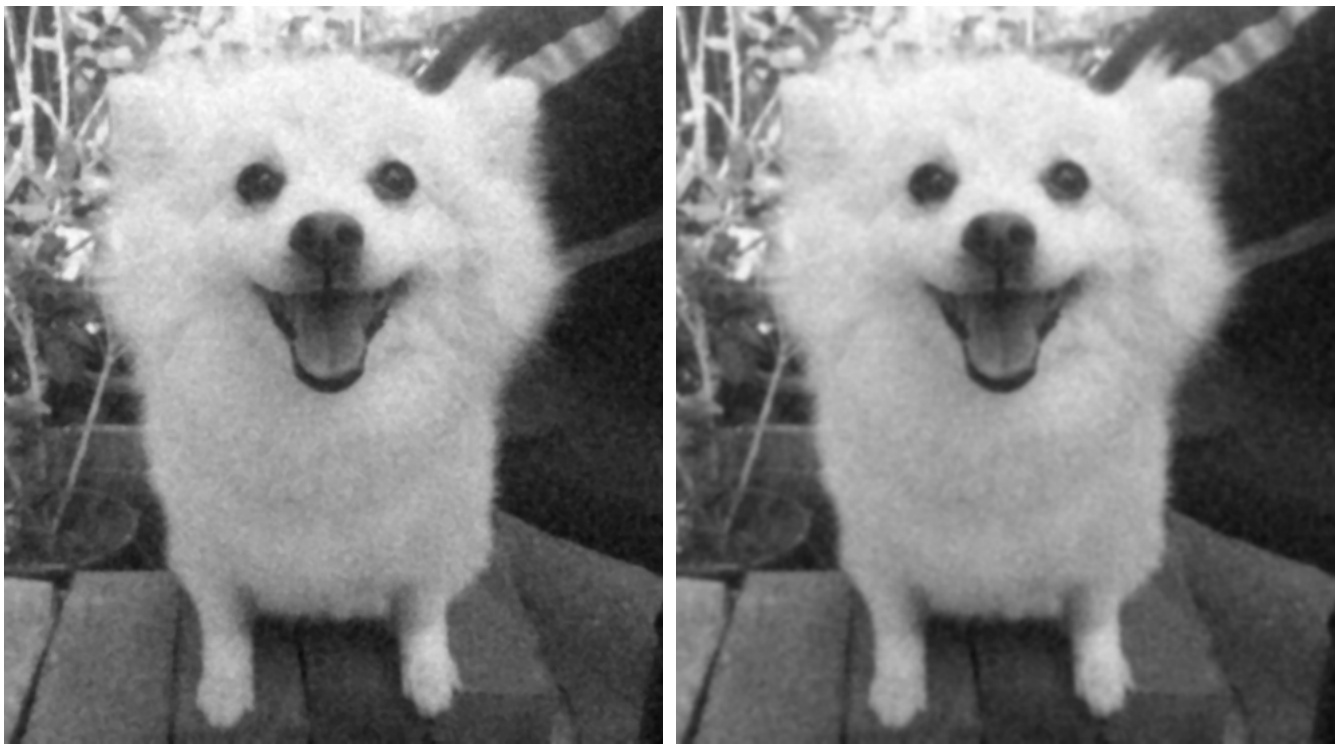


- `Iteration 1`



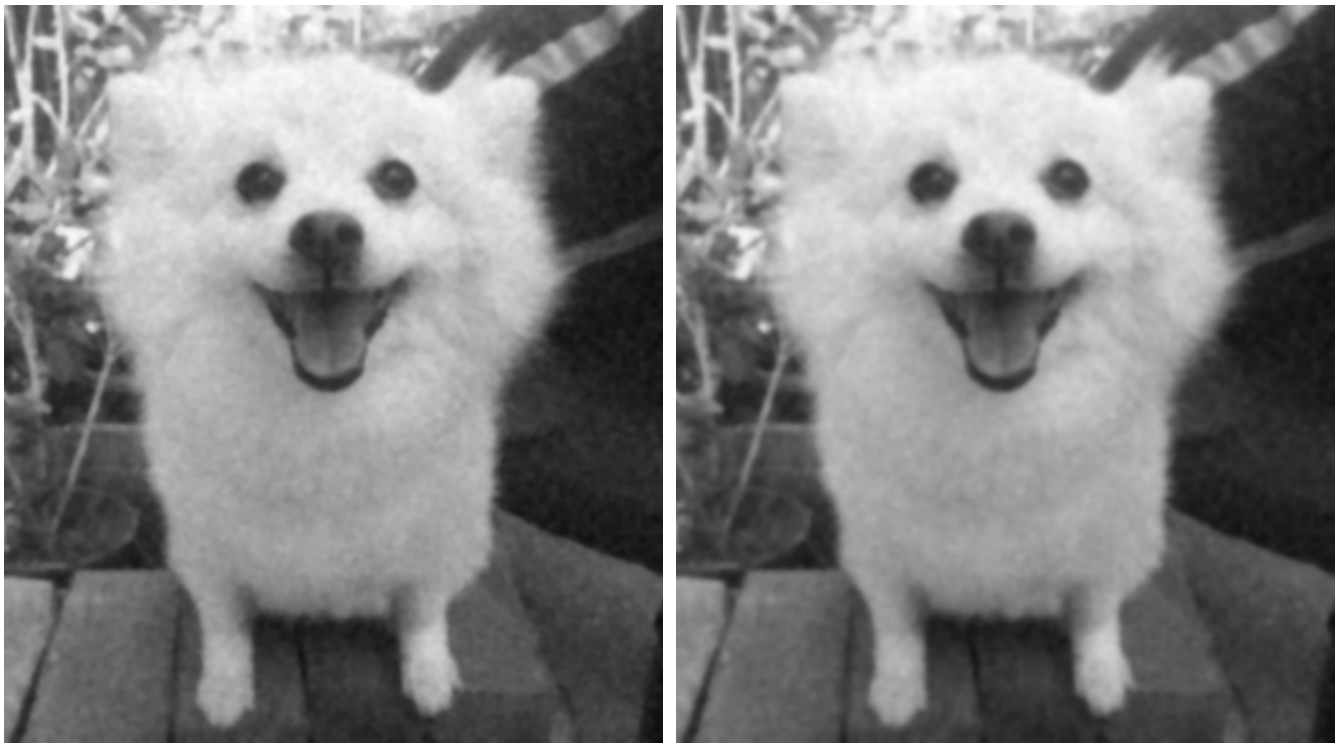
median filtering(left) / spacial filtering(right)

- `Iteration 2`



median filtering(left) / spacial filtering(right)

- `Iteration 3`



median filtering(left) / spacial filtering(right)

- 討論
  - 可以發現iterate多次得效果還不錯，先將outlier拔掉後再blur可以降低blur的誤差，使得照片更接近原本的狀況