HW2

工科所碩一 劉宜珊

1.Histogram equalization

- global histogram equalization

[算法]：把原本集中在某區塊的機率函數平均分布在整張影像的所有亮度上面，使影像更平滑，並將整個影像拉開產生明顯對比。

$$C(r_k) = \sum_{i=0}^{k} P(r_i) = \sum_{i=0}^{k} \frac{n_i}{n}$$

PDF：P(r)表示此像素值出現的機率

eg. 一張 10*10pixels 圖片裡面像素值 50 出現 3 次

P(50) = 3/(10*10) = 0.03

CDF：C(r)表示把小於等於 r 的 P(r)相加

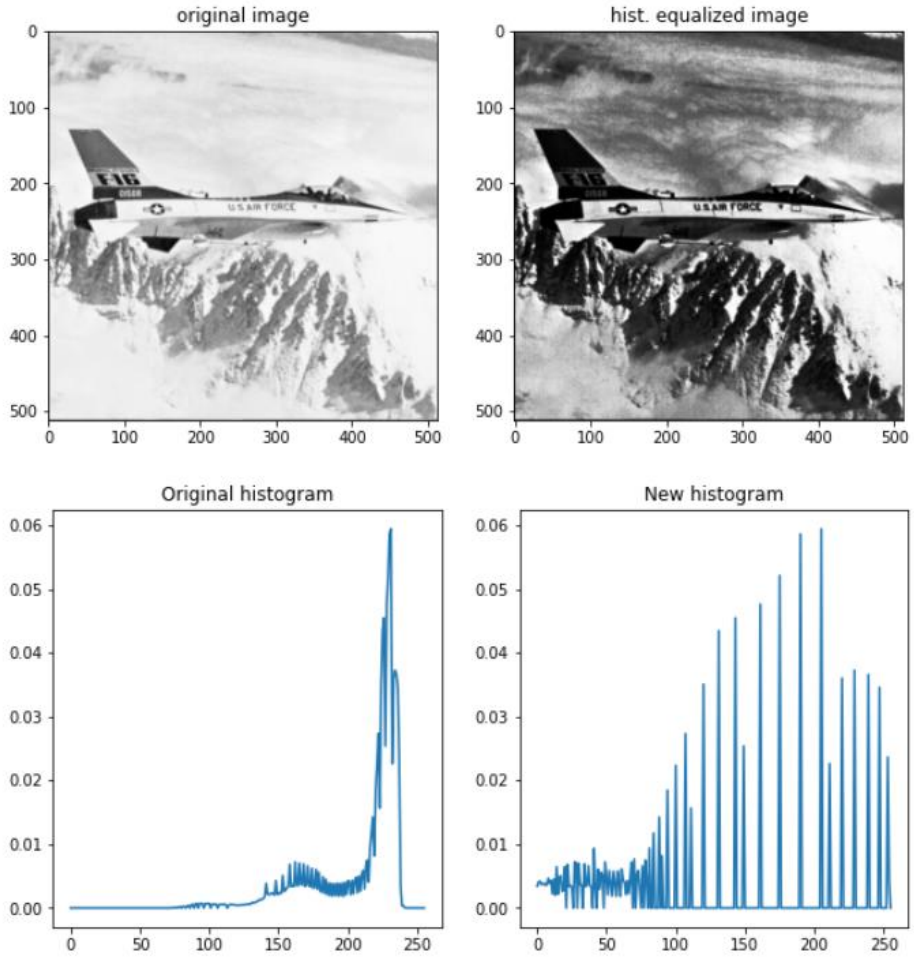
e.g C(3) = P(0)+P(1)+P(2)+P(3)

$$\mathrm{H}(v) = (int)\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

最後照著上面 H(v)的公式，就可以求出轉換後的值。

[分析]：可以看出圖片的對比度有較明顯，像雲朵皺褶的部分也看得更清楚，但在亮度的部分似乎沒有比原圖還亮。

[實現] :



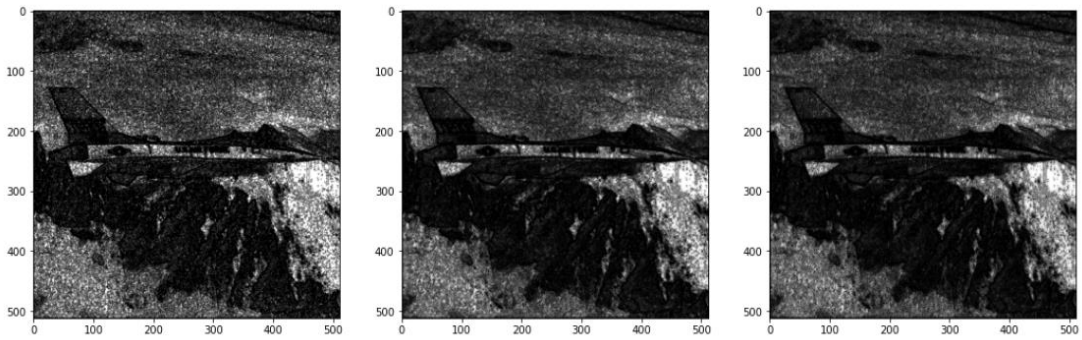
- local histogram equalization

  [算法] : 與 global histogram equalization 方法略同，只是 global 是針對整張圖片去統計像素機率，而 local 是針對指定的 kernel size 區塊去執行而已。

  [分析] : 執行後發現 kernel size 越大，圖像細節越不清晰，我想是因為較小的 kernel 能夠處理到圖像中越小的細節，越小的區塊對比拉開後，整張圖片對比拉開的區塊也就愈明顯，因此若是想局步調整某細節，我會選擇較小的 kernel size 去調整。

[實現] :

3 x 3 **5 x 5** **7 x 7**



- Histogram matching

[算法] : 將圖像 A 作為輸入圖像，圖像 B 作為目標圖像。計算 A 和 B 的 histogram 和 equalized histogram. 然後再將 A 圖像的每個 pixel 的 equalized histogram 值分別對應到 B equalized histogram 值

0	2	1	3	4
1	3	4	3	3
0	1	3	1	4
3	1	4	2	0
0	4	2	4	4

input: Image A

Pixel Value	Histogram	Equalized Histogram
0	4	4
1	5	9
2	3	12
3	6	18
4	7	25

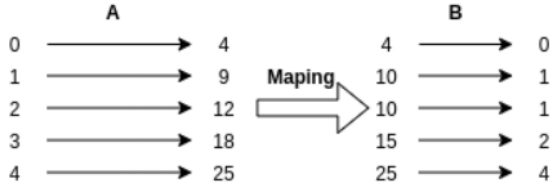
Equalized Histogram of A

2	1	2	1	0
3	3	2	4	4
1	3	2	4	4
0	0	3	2	1
1	3	1	4	0

Target: Image B

Pixel Value	Histogram	Equalized Histogram
0	4	4
1	6	10
2	5	15
3	5	20
4	5	25

Equalized Histogram of B

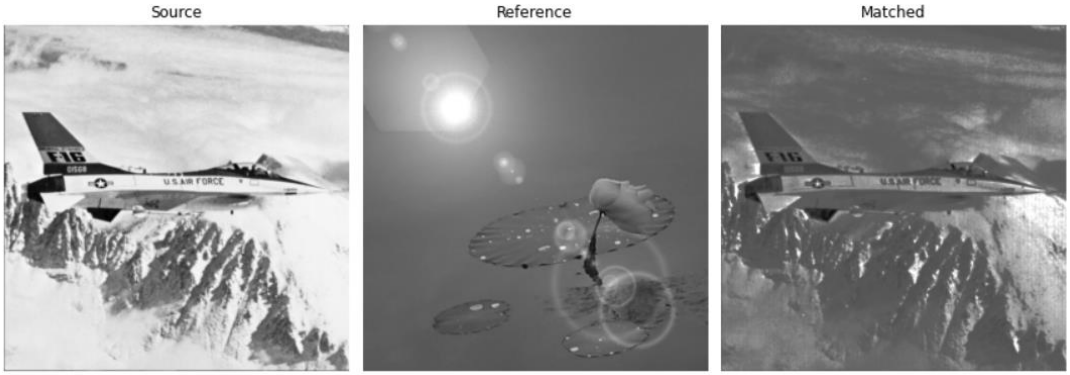


0	1	1	2	4
1	2	4	2	2
0	1	2	1	4
2	1	4	1	0
0	4	1	4	4

Modified: Image A

[分析]：通過變換後，Source 圖像和 F16 和 Reference 圖像 flower 的亮度變化規律有較接近一些

[實現]：



2. Convolution

- Gaussian Filter：

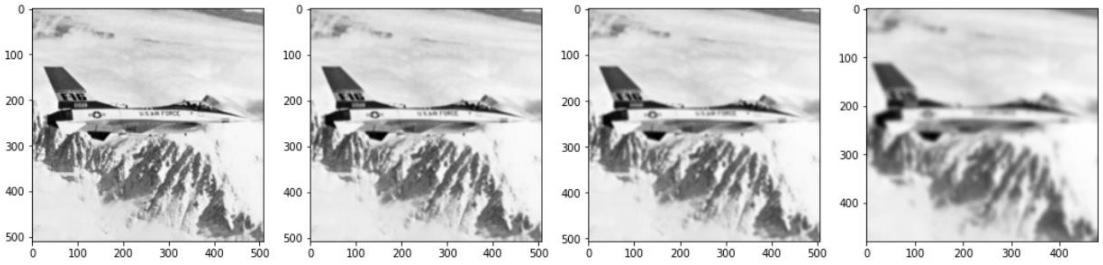
  [算法]：中心點的加權值加強，遠離中心點加權值減小，並運用以下二維高斯函式去計算每個點的權重。

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/2\sigma^2}$$

  [分析]：kernel size 越大，圖像會越模糊，而隨著 sigma 的增大，整個高斯函式的尖峰逐漸減小，整體也變的更加平緩，對影象的平滑效果越來越明顯。下面實現的部分 sigma 設定為 4。

[實現] :

3 x 3　　5 x 5　　7 x 7　　33 x 33



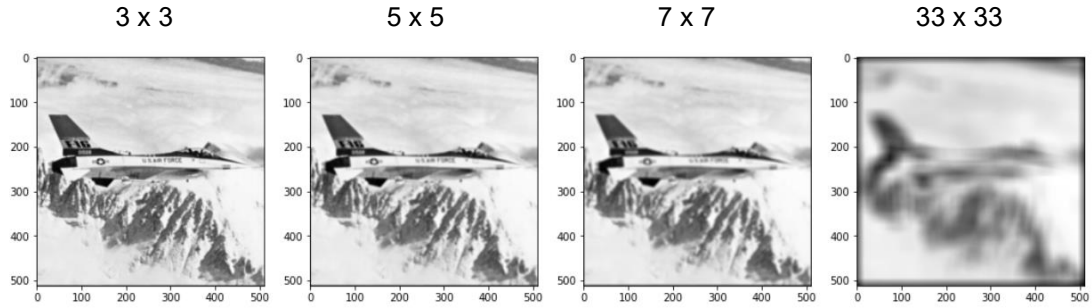
- Averaging Filter :

  [算法] : 選擇一個總 pixel 數為奇數，K x K 大小的方形 kernel，每個 kernel 所在的最中央那個點，把它調整為 kernel 內所有其它點的平均值。以 3*3 的 kernel 為例，透過「Spatial Convolution」對整張影像進行處理，取得遮罩之像素矩陣後加總除以總數(計算平均值)。

$$\frac{1}{9}\times\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

  [分析] : kernel size 越大，圖像會越模糊，當然也較能去噪，而模糊的程度似乎比上面 Gaussian Filter sigma 設定為 4 還再模糊一點。

[實現] :



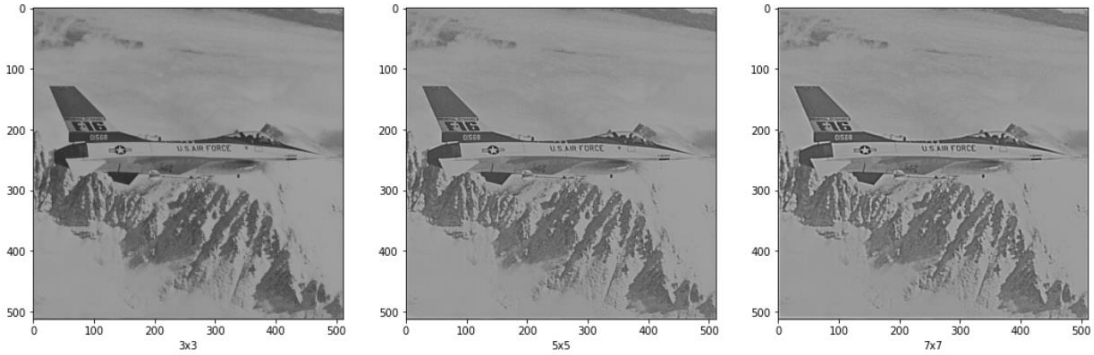
- Unsharp mask filter

[算法] :原始圖像減去前面 3x3、5x5、7x7 kernel size 的高斯模糊後得到的圖像(sigma=4)，再乘上一個 amount 值，在這裡 amount 設定為 2，算出結果後再加回原始圖像，得到 Unsharp mask filter 圖像。

公式=> Image = image + (image - gaussimage) * amount

[分析] :若減去的高斯模糊圖像 kernel size 越大，所得到的圖像邊緣將會越清晰，而在乘上的 amount 數值若越大，圖像也將越清晰。

[實現] :



- **Laplacian Filter**

[算法]：一種空間二階導數的運算子，對於影像中快速變化的區域(包含 edge)具有很大的強化作用，用以下公式對圖像進行運算:

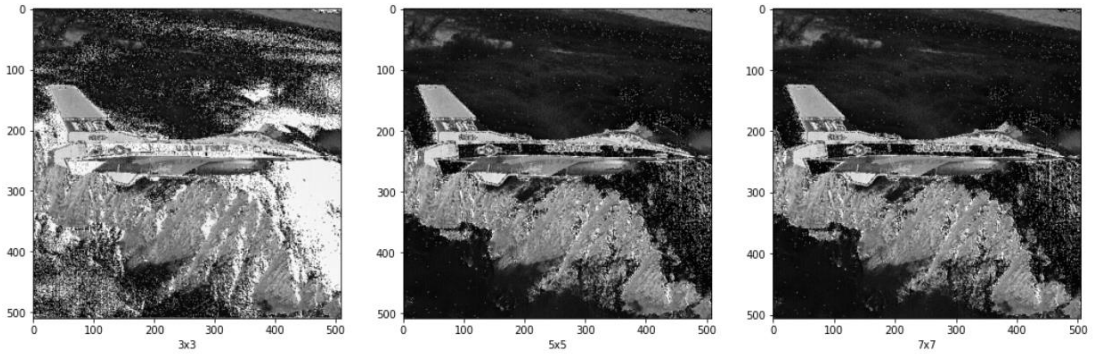
$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

首先先建好這個公式的 function，再創建此公式的 filter，之後再用這個 filter 對整張圖進行卷積運算。

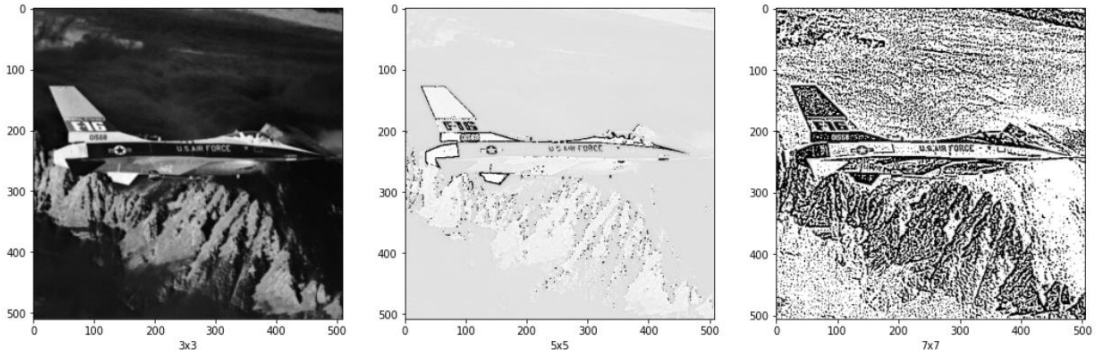
[分析]：在這裡 sigma 用 0.5 代入，用大於等於 1 的值代入的話，再不同 kernel size 間似乎會有黑白反轉的情況發生，這個原因還會再深入去探究。而 kernel 越大，圖像邊緣的部分也越明顯、清晰。

[實現]：

Sigma = 0.5



Sigma = 1



- **Sobel Filter**

  [算法] :利用局部差分尋找邊緣，計算所得到的是一個梯度的近似值。

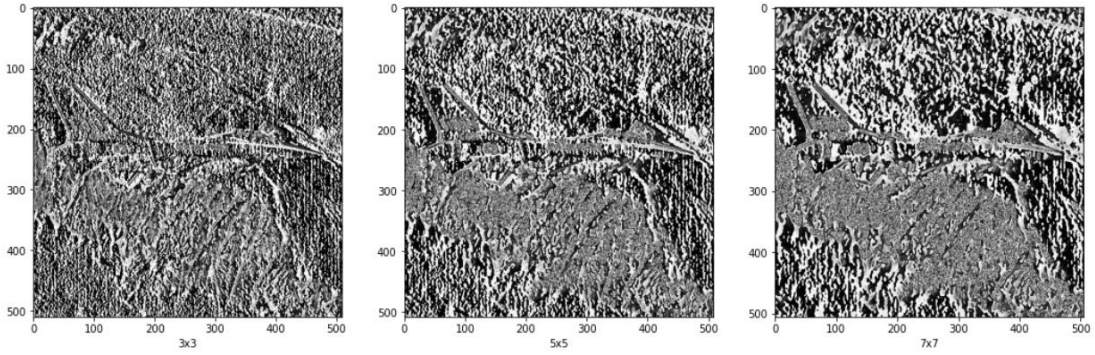
  以下為水平和垂直方向的偏導數計算 filter(以 3x3 為例)

$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

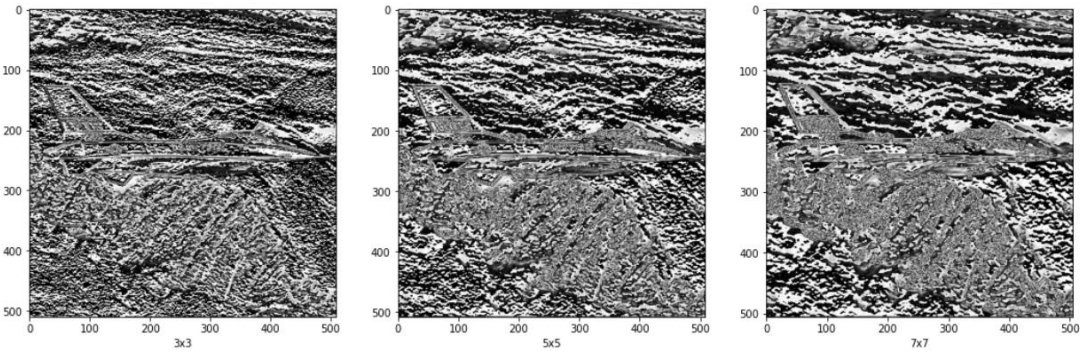
  [分析] :圖像 kernel size 越大，所得到的圖像邊緣將會越清晰且明顯，分別針對 x 方向和 y 方向進行 sobel 運算。只計算 x 方向的邊緣時，可以取得水平方向的邊緣；只計算 y 方向的邊緣時，可以取得垂直方向的邊緣

  [實現] :

positive x



positive Y



3. Convolution-2

[分析] :

Kernel-1 :

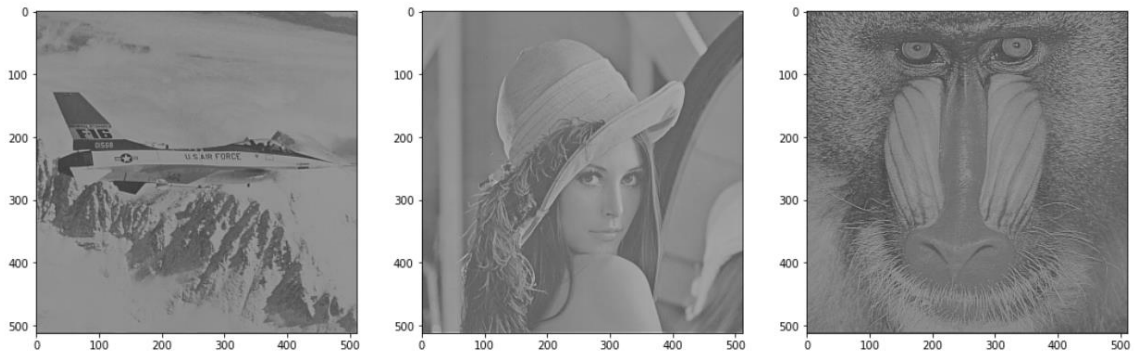
很明顯可以發現對比度下降、整張圖片的亮度也暗了許多，且圖像的深度也較深。這樣的特徵會讓要萃取某些部位的特徵時會較為困難，因對比度低，每個部份的顏色都較相似，細節也較不明顯。

Kernel-2 :

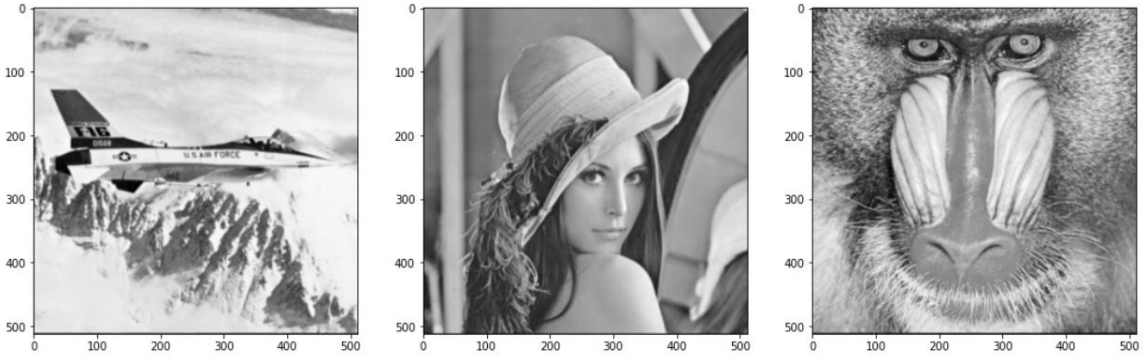
對比度有變稍微高一點，這樣能讓圖像較暗處能夠更清楚顯現較多的細節，顏色較深的部分和顏色較淺的部分也分明許多，能夠使得圖像整體更為立體。

[實現] :

Kernel - 1



Kernel - 2



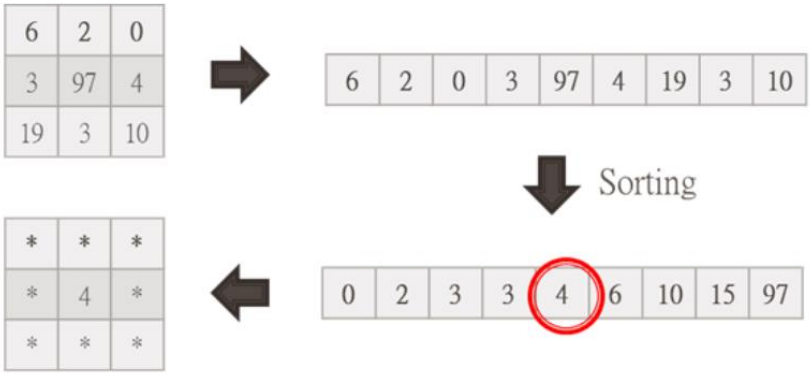
4. Image denoising

[算法] :

在非線性濾波（Order-Statistic Filter）中，選擇了 Medium filter 去做，而此濾波會取目前像素點及其周圍鄰近像素點(奇數個像素點)的像素值，將這些項素質排序，然後將位於中間位置的像素值作為目前像素的像素值。

[算法示意圖] :

以 kernel size 3x3 為例

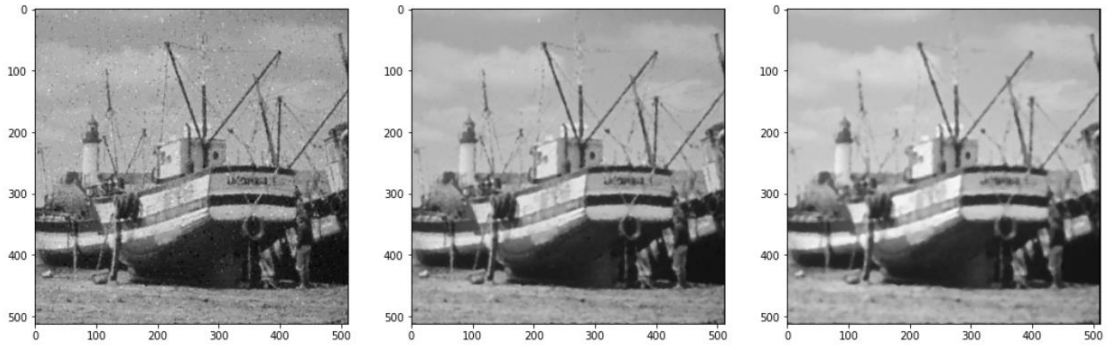


[分析] :

分別用 kerenel size 3x3、5x5、7x7 去試，發現 kernel size 越大去噪效果越好，但較沒辦法如較小的 kernel size 一樣保存圖像中較細節的部份，因此在消除噪聲和維持圖像細節是存在矛盾的，很明顯能看到 kernel size 3x3 的影像是最清晰的，雖然雜點仍十分明顯。

[實現] :

3 x 3　　5 x 5　　7 x 7

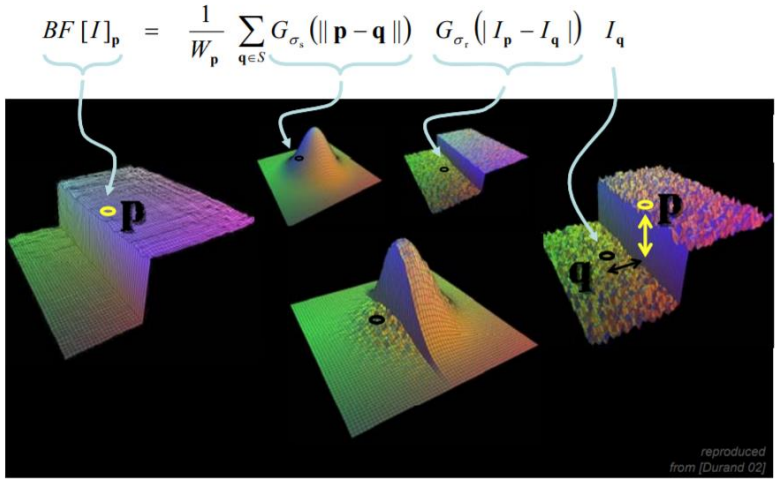


5. Bilateral filter

[算法] :

採用加權平均的方法，用周邊像素亮度值的加權平均代表某個像素的強度。較特別的地方在於：雙邊濾波的權重不僅考慮了像素的歐氏距離（如普通的高斯低通濾波，只考慮了位置對中心像素的影響），還考慮了像素範圍域中的輻射差異（例如卷積核中像素與中心像素之間相似程度、顏色強度，深度距離等），在計算中心像素的時候同時考慮這兩個權重。

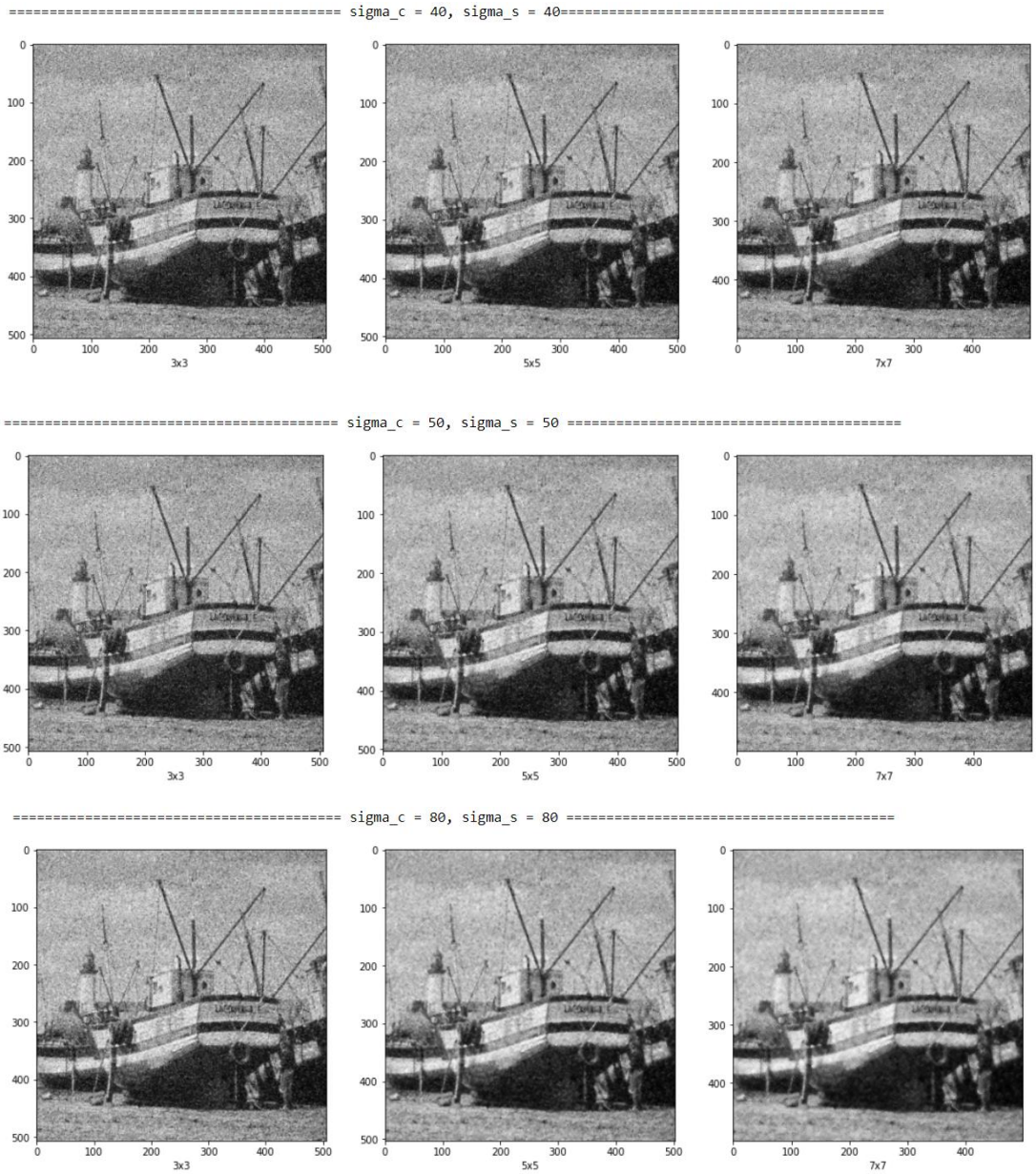
[算法示意圖] :



[分析] :

分別用 kerenel size 3x3、5x5、7x7，並用 sigma_c = 40 和 sigma_ s= 40，sigma_c = 50 和 sigma_ s= 50，sigma_c = 80 和 sigma_ s= 80，去進行效果的比較。發現 kernel size 越大，或是 sigma_c、sigma_ s 值越大，去躁的效果會越好，但模糊只是執行的時間會較久，尤其 kernel size 越大時。

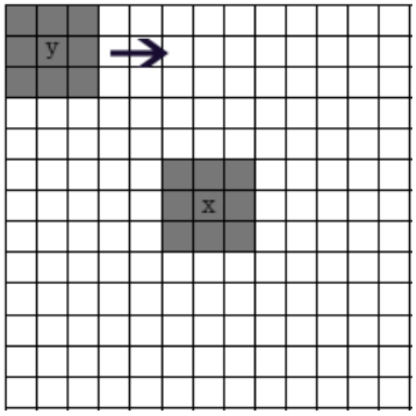
[實現] :



6. Non Local Means (NLM) filter

[算法] : NLM 算法是根據圖像中像素點與其周圍像素點的灰度值相似度進行加權求均值，因此在一定程度上消除噪聲，同時保留更多的有效信息。此算法需要在整個影象範圍內判斷畫素間的相似度，也就是每處理一個像素點時，都要計算它與影象中所有像素點間的相似度。但是考慮到效率問題，再用程式實現時，會設定兩個固定大小的視窗：搜尋視窗和鄰域視窗。鄰域視窗在搜尋視窗中滑動，根據鄰域間的相似性確定像素的權值。

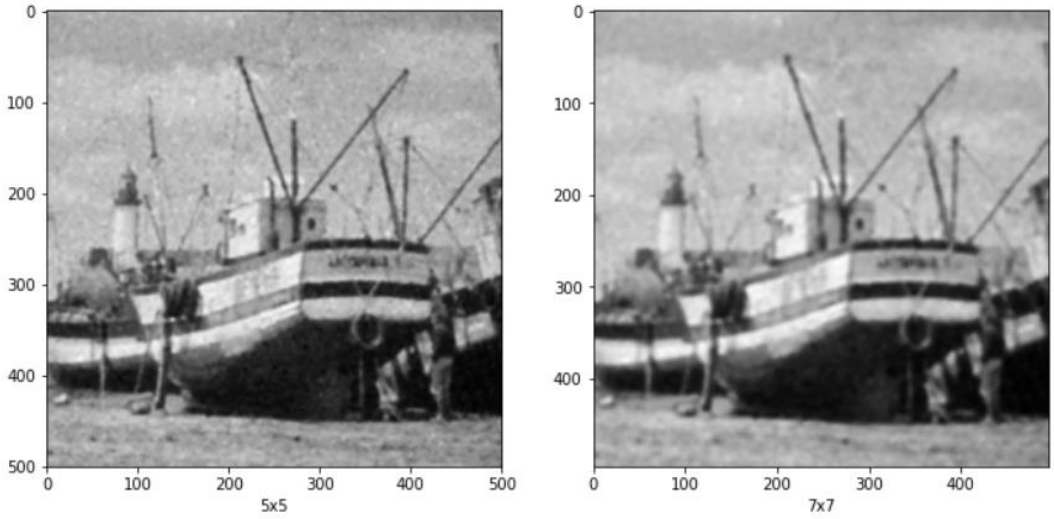
[算法示意圖] :



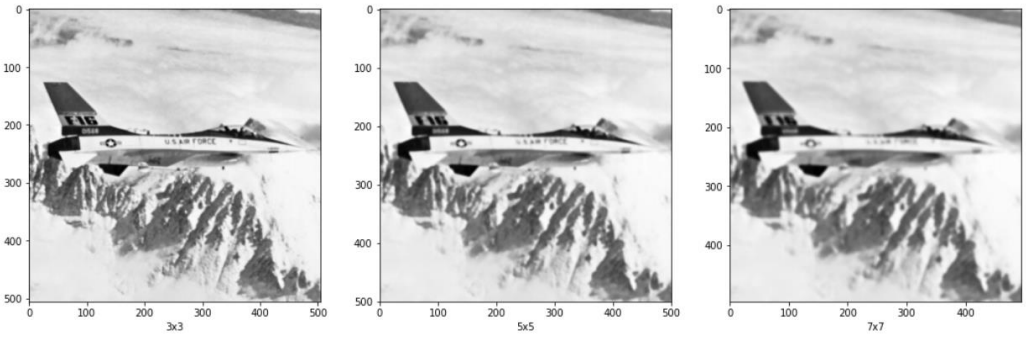
[分析] : 從 Noisy.raw 這張圖來看，5x5 和 7x7 的比起來，圖像較清晰，但雜點較多，我想是因為在真的實行 NLM 時，前半段會先進行高斯模糊，而 Kernel size 越大，圖像將會越模糊，從而導致之後實行 NLM 之後也較模糊，但也因為 7x7 的圖像較模糊，所以雜點也被模糊掉，因此相對 5x5 的圖像而言，雜點較不明顯。因此可以做出以下結論，越大的高斯函式變化越平緩，去噪水平越高，但同時也會導致影象越模糊。越小，邊緣細節成分保持得越多，但會殘留過多的雜點。

[實現] :

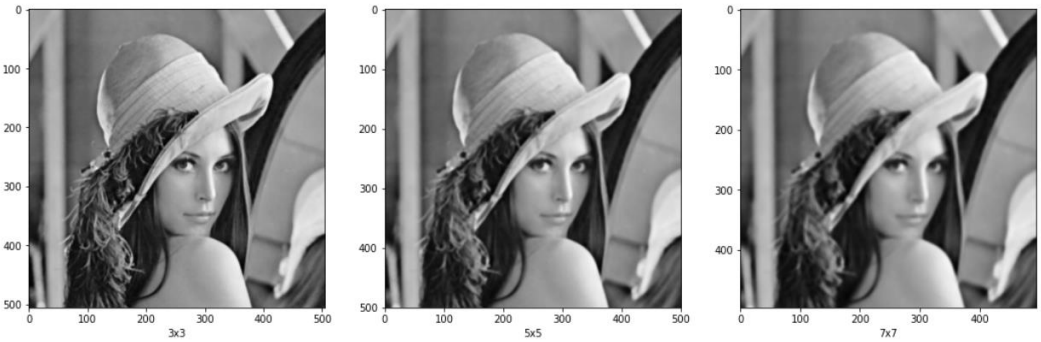
Noisy :



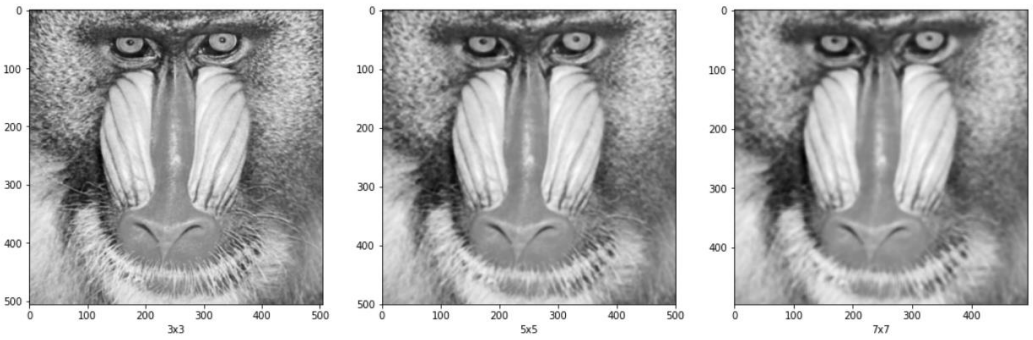
F16



Lena



Baboon



7. Performance and efficiency improvement

[算法] :

因 NLM 演算法耗時較長，所以用積分影像技術對演算法進行加速，為了降低空間複雜度，將偏移量作為最外層迴圈，即每次只需要在一個偏移方向上求取積分影像，並對該影像進行處理，而不需要一次性求取出所有積分影像

先建一個關於畫素差值的積分影象：

$$S_t(x) = \sum_{z1 \le x1, z1 \le x2} s_t(z), x = (x1, x2)$$

其中

$$s_t(x) = ||v(x) - v(x+t)||^2$$

這樣在計算 V(x)和 V(y) (y = x+t) 間的距離時，就可以在常量時間內完成 ：

這樣，整個演算法複雜度將降為 O(ND^2)

[實現] :

照著 paper 的演算法打過一次但沒有顯示出來，麻煩改作業的助教手下留情

QQ 小的盡力了。

input : V, ds, Ds, h
output: Vd
Function *st(t; z) Compute $s_t(z)$ following (13)*
    Dist2 ← 0
    for $c = 0$ *to* $Nc - 1$ **do**
        Dist2 ← Dist2 + $(\texttt{Vsym}[c](z) - \texttt{Vsym}[c](z+t))^2$
    return *(Dist2)*
**** INITIALIZATION ****
(N1, N2) ← image size
Nc ← number of color channels
Vsym ← symmetrized noisy image V with border Ds+ds
Vd ← all planes filled with 0
SW ← array filled with 0 *Sum of Weights image*
**** MAIN LOOP *** shift vector* $t = (t1, t2)$
for $t = (t1, t2) = (-Ds, -Ds)$ *to* $(+Ds, +Ds)$ **do**
    — Step 1 — Compute the integral image St, t being fixed, following (15)
    St(0,0) ← 0
    for $x1 = 1$ *to* $N1 - 1$ **do**
        St(x1,0)=st(t;(x1,0))+St(x1-1,0)
    for $x2 = 1$ *to* $N2 - 1$ **do**
        St(0,x2)=st(t;(0,x2))+St(0,x2-1)
    for $x = (x1, x2) = (1, 1)$ *to* $(N1 - 1, N2 - 1)$ **do**
        St(x)=st(t;x)+St(x1-1,x2)+St(x1,x2-1)-St(x1-1,x2-1)
    — Step 2 — Compute weight and estimate for patches $V(x), V(y)$ *with* $y = x + t$
    for $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
        $y \leftarrow x + t$
        Compute distance between the two patches using integral images, see (16)
        Dist2 ← $St(x + (ds, ds)) + St(x - (ds, ds)) - St(x + (ds, -ds)) - St(x + (-ds, +ds))$
        Dist2 ← Dist2 $/d^2$
        Compute unnormalized weight $w(x, y)$ *following (12)*
        $W(x, y) = e^{-\text{Dist2}/(Nc \times h^2)}$
        $SW(x) \leftarrow SW(x) + W(x, y)$ *Compute sum of weights, for subsequent normalization*
        Compute estimate as weighted average
        for $c = 0$ *to* $Nc - 1$ **do**
            $\texttt{Vd}[c](x) \leftarrow \texttt{Vd}[c](x) + W(x, y) \times \texttt{Vsym}[c](y)$

**** FINAL LOOP *** Compute final estimate at pixel* $x = (x_1, x_2)$
for $x = (x1, x2) = (0, 0)$ *to* $(N1 - 1, N2 - 1)$ **do**
    for $c = 0$ *to* $Nc - 1$ **do**
        $\texttt{Vd}[c](x) \leftarrow \min(\max(\texttt{Vd}[c](x)/SW(x), 0), 255)$