# DIPCV - Assignment III

工科所碩一 劉宜珊

## 1. Fourier Transform

### I. Based on DFT, please draw the frequency response without shitting, where the lower-frequency will be located at four corners.

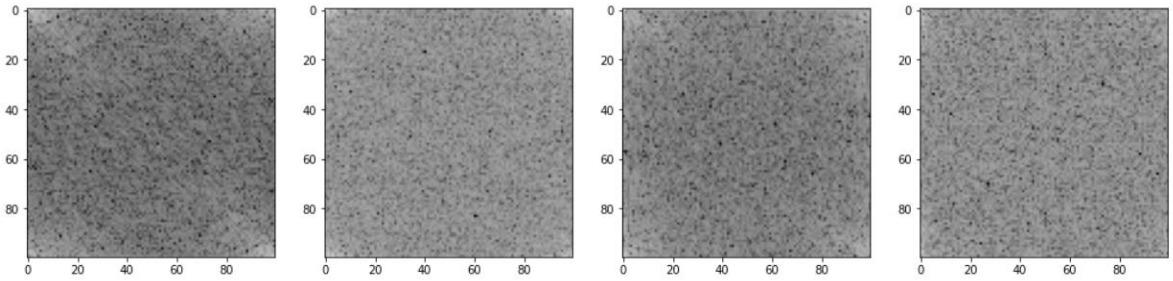
【作法】

$$\tilde{I}(u,v) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} I(i,j)\, e^{-\sqrt{-1}\frac{2\pi}{N}(ui+vj)}$$

建一個 forward_transform( ) functon 去計算 DFT 公式，因原圖像太大會跑不出來，所以用 Nearest Neighbor 將原圖 512 X 512 縮小到 100 X 100，再將 100 X 100 的圖片當 input 值放到 function 裡做 DFT

【實現】

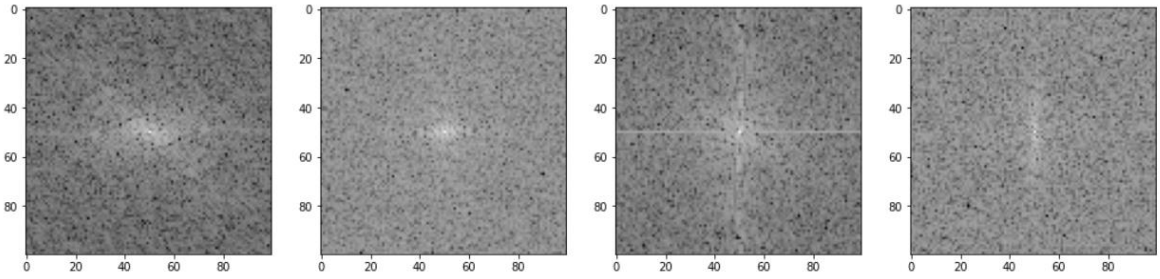
**Lena** **Baboon** **F16** **Noisy**



### ii. Please show the centralized result based on (i).

【實現】

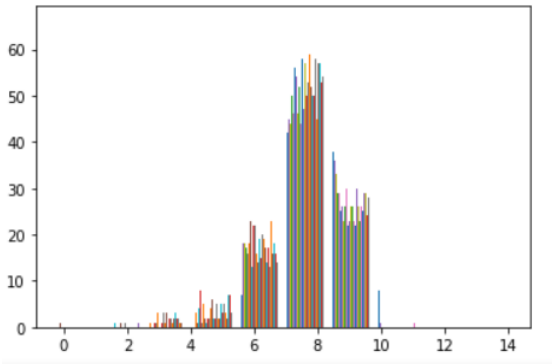
**Lena** **Baboon** **F16** **Noisy**



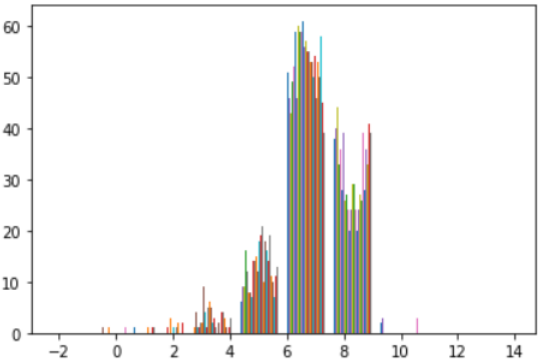
**iii. histogram of the spectrum**

【實現】

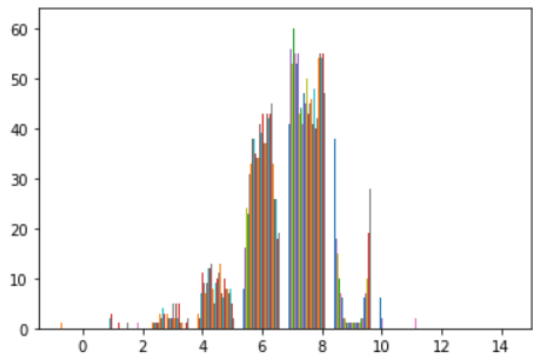
Noisy




Baboon




F16




【分析】

- Noise.raw 和 Baboon.raw 在大約 frequency20-40(x 軸 8-10)之間分布的比例比 F16 還高 ； 在低頻段的部分大約 frequency10-20(x 軸 4-5)之間分布比例 F16 比 Noise.raw 和 Baboon.raw 還高，而將影像讀出後，也可以很明顯的發現 F16 淺色部分比 Noise.raw 和 Baboon.raw 還多，由此可知為何會如此分布。

## 2. Low-pass filter

- **Apply a random Noise**

【作法】

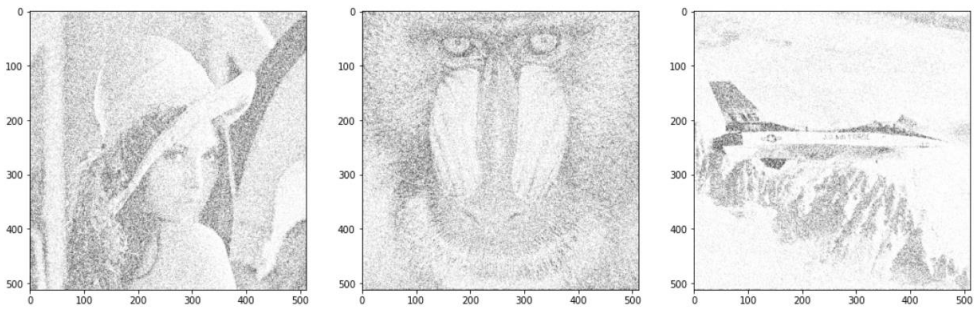
- Uniform noise :
  建一個 uniform_noise(img) function，在函式中產生最低值為 0，最高值為 1 的 Uniform noise filter，再將 noise 加到原始的圖片上生成有雜訊的圖片 。
- Gaussian noise :
  建一個 gaussian_noise(img, mean=0, sigma=0.1) function，

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

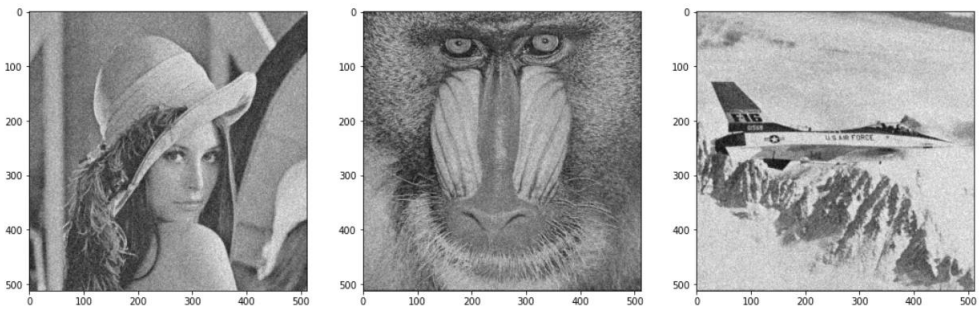
  在這裡設定 mean=0, sigma=0.1
  並以上面的高斯公式亂數去製作雜點 filter，再將這個 filter 加到原始的圖片上生成有雜訊的圖片 。

【實作】

Uniform noise :



Gaussian noise



- **ideal low-pass**

【作法】

1) 建一個 ideal_low_pass_filter(cutoff) 函式去計算出 ideal low pass filter，而要 cutoff 的 frequency 在此設定為 30、80、200，再分別以這三個不同的 cutoff 值去搭配 gaussian noise 和 uniform noise

公式：

$$H(u,v)=\begin{cases}1 & \text{if } D(u,v)\le D_0\\ 0 & \text{if } D(u,v)>D_0\end{cases}$$

$D_0$ : cutoff 的 frequency

D : frequency domain 中的點（u，v）與 frequency rectangle 中心之間的距離

$$D(u,v)=\left[(u-P/2)^2+(v-Q/2)^2\right]^{1/2}$$

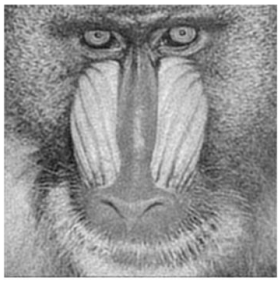
2) 得出 lowpass filter 後，再與輸入的圖片做卷積，用所建的函式 Filtering(img, mask)

【實作】

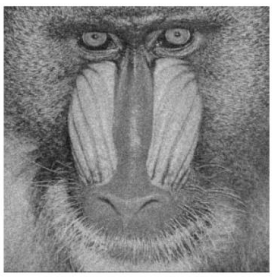
gaussian noise + ideal low-pass cutoff 30



gaussian noise + ideal low-pass cutoff 80

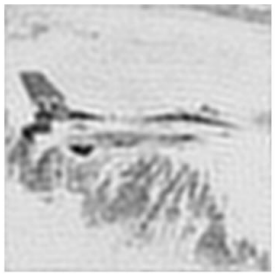




gaussian noise + ideal low-pass cutoff 200


uniform noise + ideal low-pass cutoff 30





uniform noise + ideal low-pass cutoff 80



uniform noise + ideal low-pass cutoff 200



- **Gaussian low-pass**

【作法】

1) 建一個 gaussian_low_pass_filter(cutoff) 函式去計算出 gaussian low pass filter，而要 cutoff 的 frequency 在此設定為 30、80、200，再分別以這三個不同的 cutoff 值去搭配 gaussian noise 和 uniform noise

公式 ：

$D_0$ : cutoff 的 frequency

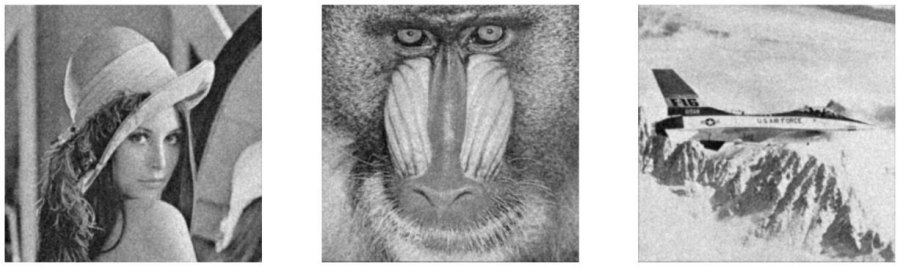
$H(u,v) = e^{-D^2(u,v)/2\sigma^2}$ letting $\sigma = D_0$ → $H(u,v) = e^{-D^2(u,v)/2D_0^2}$

2) 得出 gaussian lowpass filter 後，再與輸入的圖片做卷積，用所建的函式 Filtering(img, mask)

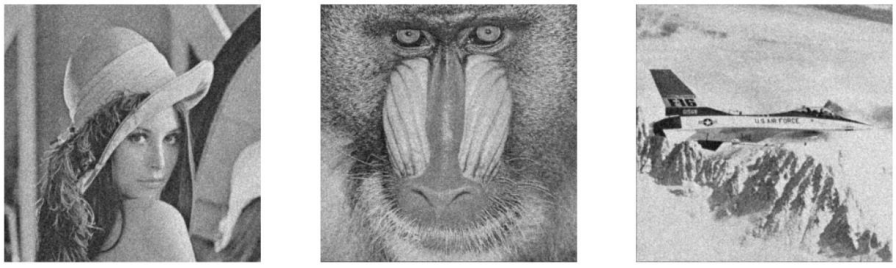
【實作】

gaussian noise + gaussian low-pass cutoff 30

gaussian noise + gaussian low-pass cutoff 80



gaussian noise + gaussian low-pass cutoff 200

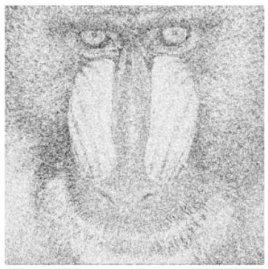


uniform noise + gaussian low-pass cutoff 30

uniform noise + gaussian low-pass cutoff 80



uniform noise + gaussian low-pass cutoff 200


## 3. high-pass filter

- **ideal high-pass filters**

【作法】

1) 用 1 去減掉 ideal low-pass 後所得到的 filter，因前面已經有建好 ideal low-pass 函式，所以直接 recall。在此 cutoff 設定為 15、80、200

   公式 ：

$$H_{HP}(u,v) = 1 - H_{LP}(u,v)$$

   $H_{LP}(u, v)$ : 為 ideal low-pass filter

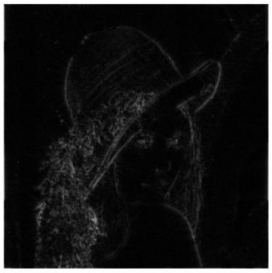
2) 得到 high-pass filter 後，再與輸入的圖片做卷積，用所建的函式 Filtering (img, mask)，將會得到影像的邊緣和高頻值。

【實作】

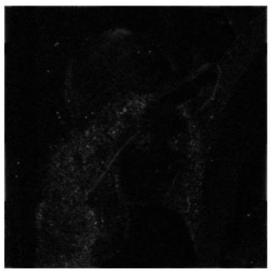
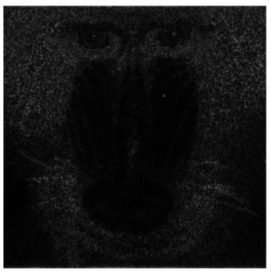
ideal high-pass cutoff 15



ideal high-pass cutoff 80



ideal high-pass cutoff 200


【分析】

可以發現 Cutoff 值設定越大，影像的邊緣輪廓會越不明顯，甚至 cutoff 值為 200 時幾乎整張影像都是黑的，我想是因為 frequency 越大，需要越深越明顯的邊緣輪廓才能通過 filter，因此 cutoff 越大，能通過 filter 的邊緣輪廓越少，自然整個影像的邊緣輪廓也就越不明顯或直接消失。

- **butterworth high-pass filters**

【作法】

1) 建一個 butterworth_high_pass_filter(cutoff, order) 函式去計算出 butterworth high pass filter，而要 cutoff 的 frequency 在此 cutoff 皆設定為 5，order 設定為 2、11、20
公式 ：

$$H(u,v)=\frac{1}{1+\left[D_0 / D(u,v)\right]^{2n}}$$

2) 得到 butterworth high-pass filter 後，再與輸入的圖片做卷積，用所建的函式 Filtering (img, mask)。

【實作】

Butterworth high-pass cutoff 5/ order2

Butterworth high-pass cutoff 5/ order11

Butterworth high-pass cutoff 5/ order20

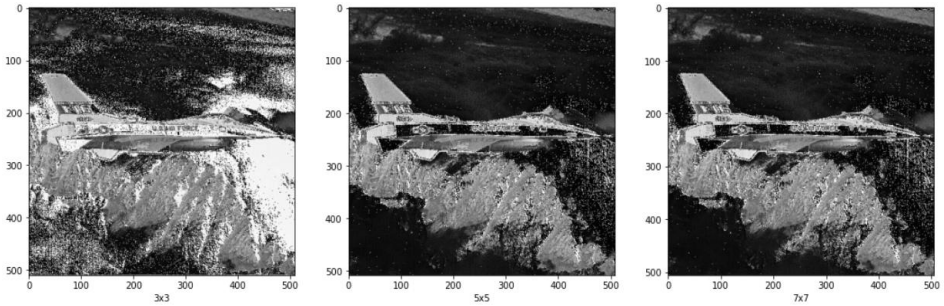
【分析】

前面已比較過不同 cutoff 的情況，而在此 cutoff 都設定在 5 去比較不同 order 的情況，可以發現 order 越大，影像的紋理線條更明顯，由紅色圈出來的地方可以到比較明顯的差異。

- **與 Laplacian 做比較**

Butterworth high-pass cutoff 5/ order2

Laplacian




【分析】

以 F16 這張圖為例，可以看出在影像中物體的邊緣 frequency high-pass 保留得比 Laplacian 好，且邊緣線條也較完整請連續；但在紋理的部分 Laplacian 保留的較細緻，影像細節也保留的較好。不過，如果在做影像辨識時，注重的為整個影像中物體邊緣的話，用 frequency high-pass 會比較好，若用 Laplacian，因其紋理被保留得不錯，會成為雜訊影響到影像中物體邊緣的偵測，有可能錯將紋理當成邊緣而影響辨識的準確率。

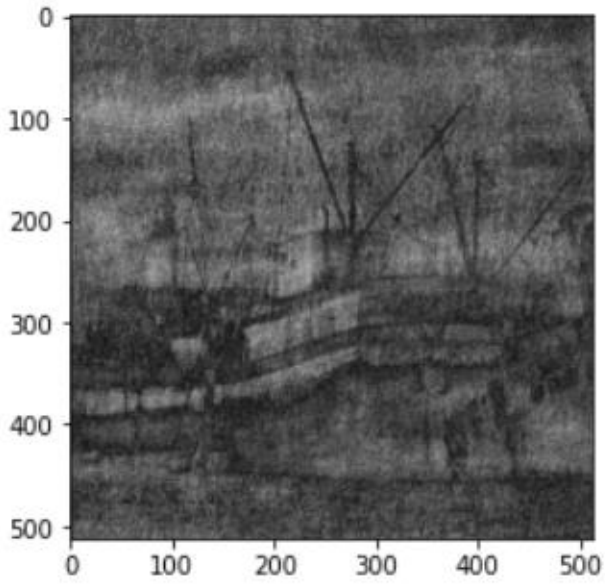
## 4. Image denoising

### i. Inverse filter

【作法】

- 建一個 inv_filter(img_bgr) 函式去計算 Inverse filter
  *變數 g 表示
  g(x, y)： 指所觀察到的有雜訊的圖片，在此指要輸入去處理的影像 Noisy.raw
  *變數 h 表示
  h(x, y)： 為高斯分布的雜點
- 將得到的 g、h 做 FT transformation 得到 G、H，得到 G、H 後再做正規化後得到 H_norm、G_norm
- 將設去除雜點後為 F_temp = G_norm/H_norm ，再將 F_temp 正規化得到 F_norm 並乘上 G.max()，將影像重新縮放到原始比例得到 F_hat
- 將 F_hat 做 FT transform 並取絕對值後，就可以得到 Inverse filter，再用所建的函式 Filtering (img, mask)與輸入的圖片做卷積

【實作】



**ii. Wiener filter**

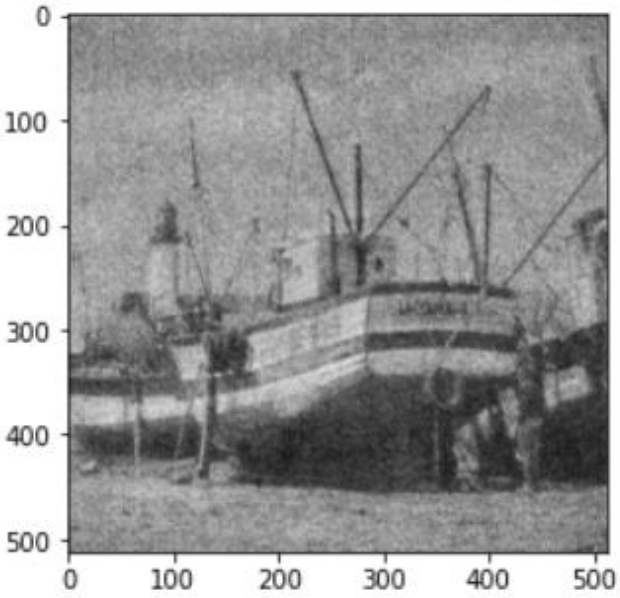
【作法】

- 建一個 weiner_filter(img_bgr, K_small) 函式去計算 weiner filter
  *變數 g 表示
  g(x, y)： 指所觀察到的有雜訊的圖片，在此指要輸入去處理的影像 Noisy.raw
  *變數 h 表示
  h(x, y)： 為高斯分布的雜點
- 將得到的 g、h 做 FT transformation 得到 G、H，得到 G、H 後再做正規化後得到 H_norm、G_norm
- 以此公式創建 H_weiner，並正規化得到 H_norm

$$W(u,v) = \frac{H^*(u,v)}{|H(u,v)|^2 + K(u,v)}$$

- 假設去雜點後的影像為 F_temp，F_temp = G_norm/H_norm ，再將 F_temp 正規化得到 F_norm 並乘上 G.max()，將影像重新縮放到原始比例得到 F_hat
- 將 F_hat 做 FT transform 並取絕對值後，就可以得到 weiner filter，再用所建的函式 Filtering (img, mask)與輸入的圖片做卷積

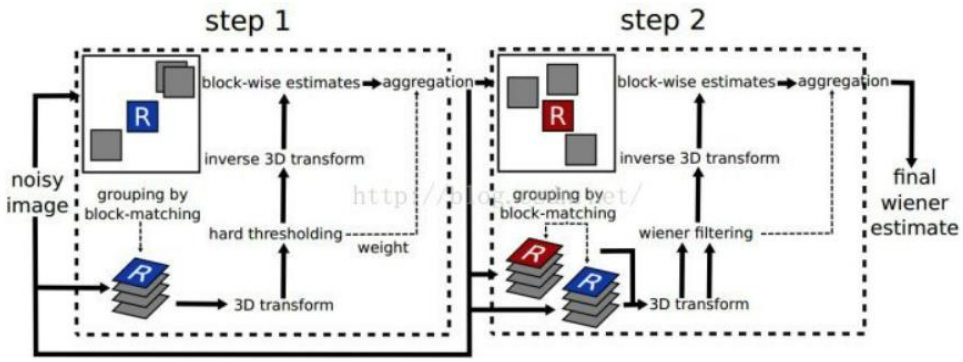
【實作】



**iii. BM3D [1]**

**【作法】**

以以下這張圖去分為 step1 和 step2 去實現



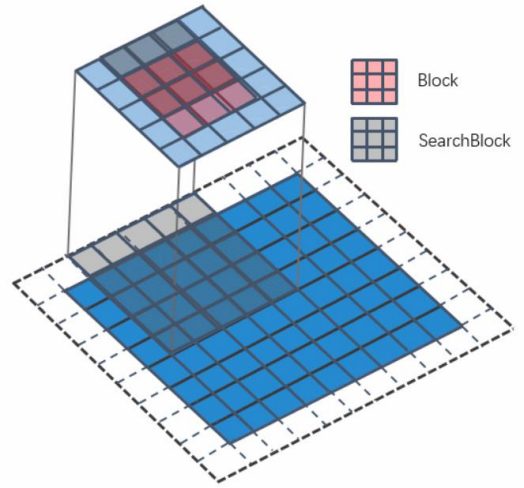

- 先建 Initialization、SearchWindow、dct2D、idct2D、PreDCT、ComputePSNR 函式去預處理 step1、step2 步驟中會用到的方法

**Step1:**

**S1.1---grouping**

建 Step1_Grouping 函式去實現以下 ：

1) 與 Non-Local Means 相似，先尋找出所有的 block，並且做初步的 2D 變換。(方便後面的 Grouping)
2) 在 SearchWindows 中，Grouping 所有的 Block。(由於邊界原因，不同的 Group，Group 的大小不同)。




3) 計算 block 之間的 L2 distance，並根據 distance 的大小順序進行排列。

**S1.2--- collaborative filtering**

建 Step1_3DFiltering 函式去實現以下 ：

1) 在每一個 Group 中，都有了相似的 Block 集合，然後再進行 3D 的線性變換。但是由於 3D 線性變換較為複雜。一般使用一次 2D 的變換以及第三維度的 1D 變換進行代替
2) 在前面 S1.1. (1)中，已經對所有的 block 進行了 2D 的 DCT 變換。所以直接對每一個 Group 進行第三維的 1D 變換即可。到此，已經完成了一次完整的 3D 變換。
3) Hard Threshold 閾值化處理 ： 對 3D 變換後的 Group 進行一次硬閾值的控制，將不符合條件的 3D 轉換值直接變為 0。
4) 對 Threshold 後的 Group 進行像素值為 0 的統計，根據 0 的個數，計算出權重 weight。
5) 將所有的 Group 進行 3D 逆轉換。這裡同樣使用一次 2D 逆轉換和一次 1D 逆轉換進行代替。
6) 然後將 Group 中的所有圖片按照權重值 aggregate 到圖像中。
7) 得到 Step1 的結果圖像 image_basic

**S1.3---aggregation**

建 Step1_Aggregation 函式，並根據前面 S1.2. 4) 計算出的權重和一個自定義的 kaiser Windows 進行疊加。

**Step2:**

**S2.1--- grouping**

1) 保留 Step1 中的噪聲圖像 Group
2) 將 Step1 的結果 image_basic 進行 grouping

**S2.2--- collaborative filtering**

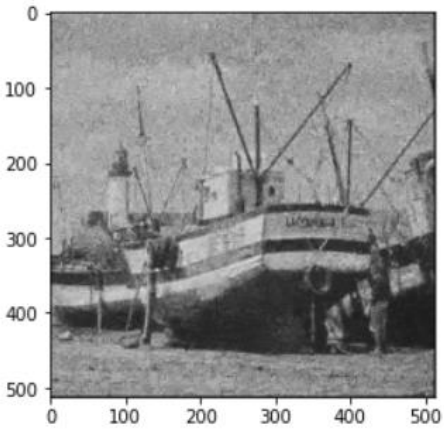
1) 與 Step 1 不同，Step 2 不適用 Hard Threshold，而是採用 Wiener Filter。這一步的目的為利用 Wiener Filter 的特性，在第一步降噪後的圖像的基礎上，結合原始圖像，恢復圖像的一些細節信息。

**S2.3---aggregation**

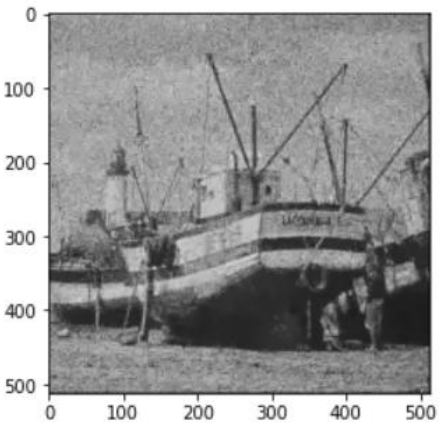
1) 結合 Kaiser Window 進行 aggregate

【實現】

BM3D Step1



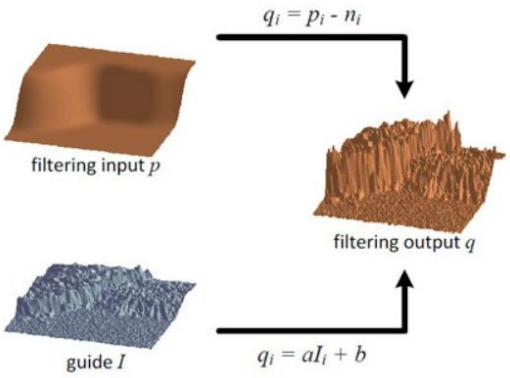

BM3D Step2




**iv. Guided Filter [2]**

【作法】

建一個 gf_color(I, p, r, eps, s=None)和一個 gf_gray(I, p, r, eps, s=None) 函式，其中 P 為輸入圖，而 I 為引導圖，將之合併做濾波，在此輸入圖和引導圖皆為同一張圖，是意圖如下:




Guided Filter 假設輸出圖 q 與 guide I 之間有局部線性的關係，而輸出圖 q 同時也等同輸入圖 p 去除雜訊 n 後的結果。因此可以得到一組聯立方程式，考慮無約束的影像還原方法，把雜訊與其他變數分別置於等號兩邊，目標最小化 $n^2$ 並施加正則化，得到 a 與 b 在 window k 中的最小平方式：

$$E(a_k, b_k) = \sum_{i \in w_k} [(a_k I_i + b_k - p_i)^2 + \varepsilon a_k^2]$$

其解為 ：

$$a_k = \frac{\frac{1}{|w|}\sum_{i \in w_k} I_i p_i - \mu_i \bar{p_k}}{\sigma_k^2 + \varepsilon}$$

$$b_k = \bar{p_k} - a_k \mu_k$$

其中，$a_k$的分子為 I 與 p 對應座標 window 的共變異數，分母為 I 影像中各座標 windows 的變異數。可以進一步把這個結果流程化如下， $f_{mean}$ 為半徑 r ，實際 window 大小為 (2r + 1) * (2r + 1)的 Box filter。

**Algorithm 1. Guided Filter.**

**Input:** filtering input image $p$, guidance image $I$, radius $r$, regularization $\epsilon$

**Output:** filtering output $q$.

1: $\text{mean}_I = f_{\text{mean}}(I)$
$\text{mean}_p = f_{\text{mean}}(p)$
$\text{corr}_I = f_{\text{mean}}(I. * I)$
$\text{corr}_{Ip} = f_{\text{mean}}(I. * p)$

2: $\text{var}_I = \text{corr}_I - \text{mean}_I. * \text{mean}_I$
$\text{cov}_{Ip} = \text{corr}_{Ip} - \text{mean}_I. * \text{mean}_p$

3: $a = \text{cov}_{Ip}./(\text{var}_I + \epsilon)$
$b = \text{mean}_p - a. * \text{mean}_I$

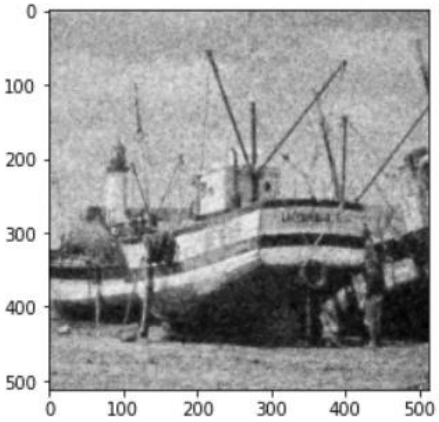
4: $\text{mean}_a = f_{\text{mean}}(a)$
$\text{mean}_b = f_{\text{mean}}(b)$

5: $q = \text{mean}_a. * I + \text{mean}_b$

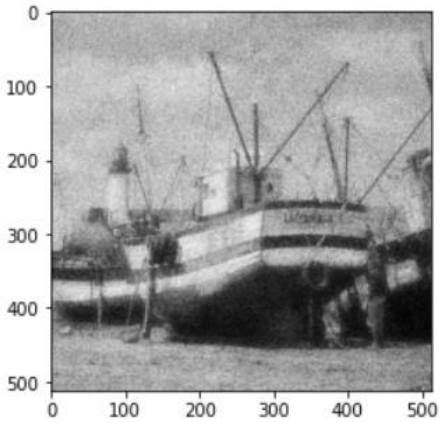
/* $f_{\text{mean}}$ is a mean filter with a wide variety of O($N$) time methods. */

**【實作】**

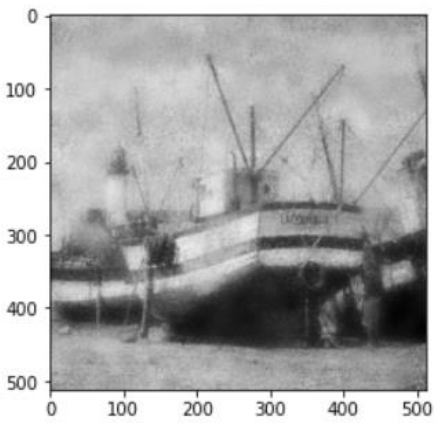
r = 2 eps = 0.05



r = 8 eps = 0.05



r = 8 eps = 4



【分析】

發現 window 半徑 r 和正則項 esp，分別影響輸出的模糊範圍及模糊程度，r、esp 愈大，模糊的程度也愈大，不過即使在很大程度的模糊中，也仍然能在圖片中看出 Guided Filter 對船身細節的保留。

## 5. DCT as the image restoration domain

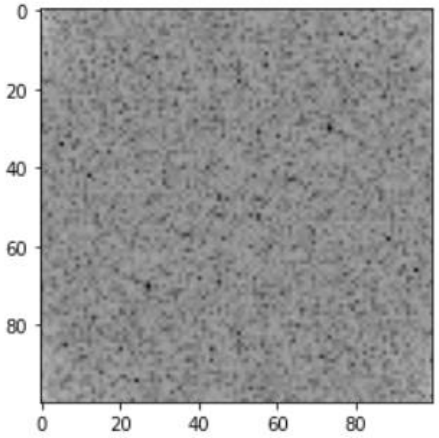
【作法】

$$\tilde{I}_{real}(u, v) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(i, j) \cos\left[\frac{2\pi}{N}(ui + vj)\right]$$

建一個 discrete_cosine_tranform functon 去計算 DCT 公式，因原圖像太大會跑不出來，所以用 Nearest Neighbor 將原圖 512 X 512 縮小到 100 X 100，再將 100 X 100 的圖片當 input 值放到 function 裡做 DCT

【實作】

**Frequency**



**套 DCT 的 Noisy.raw**