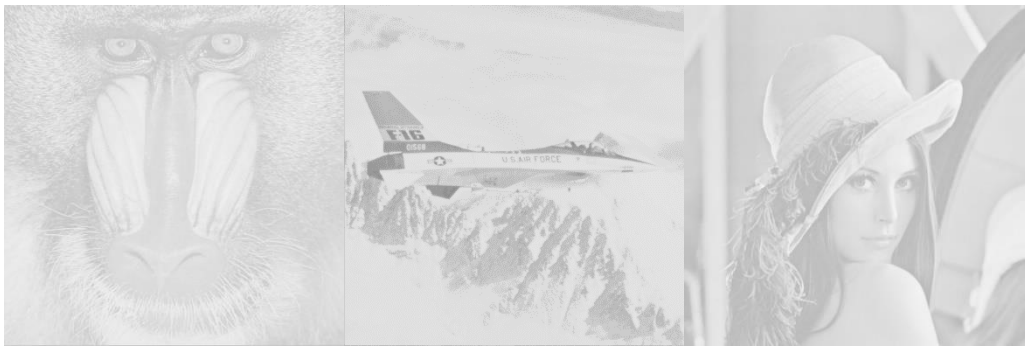
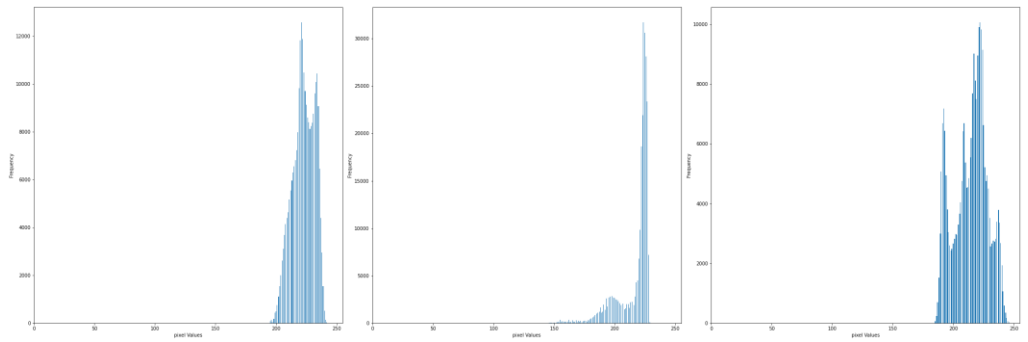
HW2　RE6091020 汪玄同

1. Histogram equalization

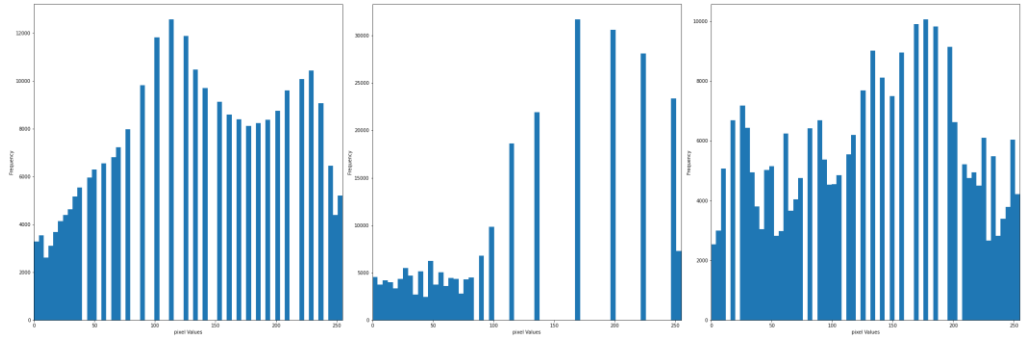
(1)下圖為三張測試用原始圖片

下圖為測試圖片經過調亮之後的結果以及他們的 histogram

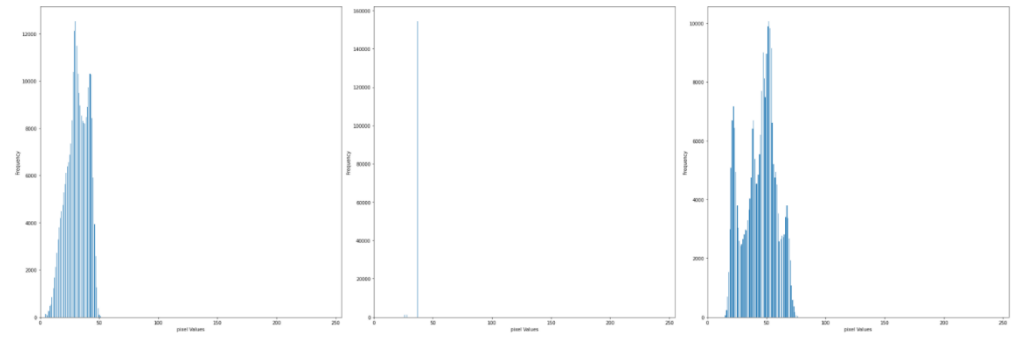




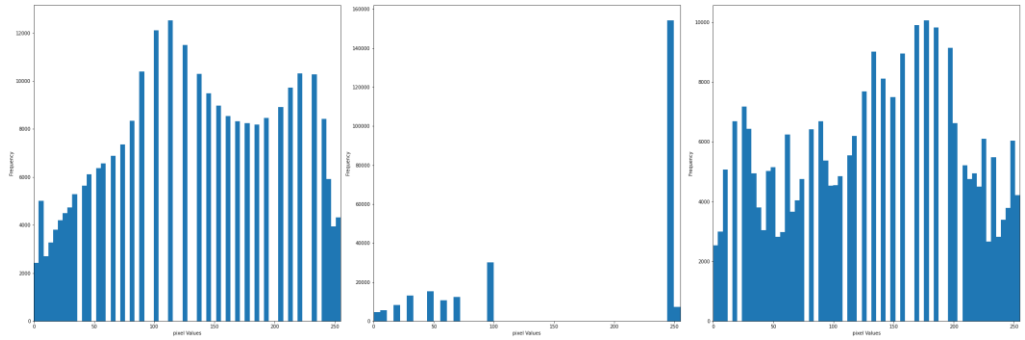
下圖為經過 histogram equalization 轉換後的結果圖片以及他們的 histogram



下圖為測試圖片經過調暗之後的結果以及他們的 histogram

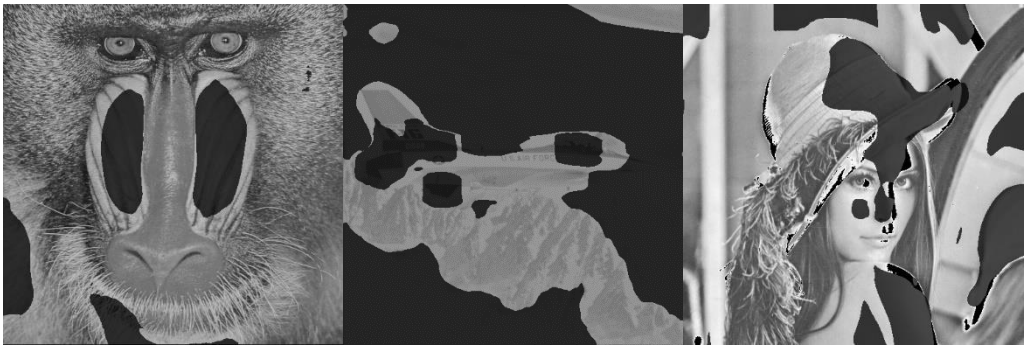


下圖為經過 histogram equalization 轉換後的結果圖片以及他們的 histogram



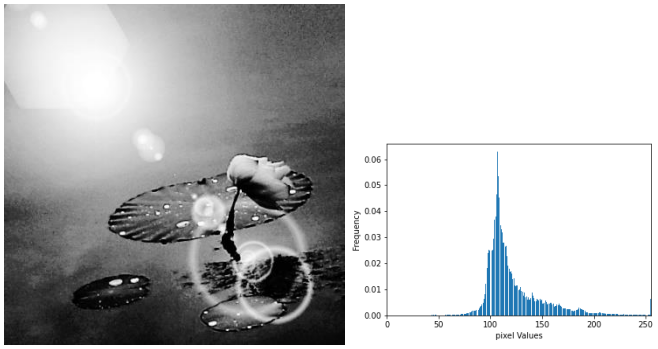
(2)使用 local histogram 將整體很暗的圖片中有亮度的地方抓出來
底下為原圖

下圖為經過 local histogram 轉換後的圖片，參數設定為 kernel 大小為 51×51，$k_0 = 1.2, k_1 = 0.1, k_2 = 1.5, E = 3$，執行時間約為 15 秒

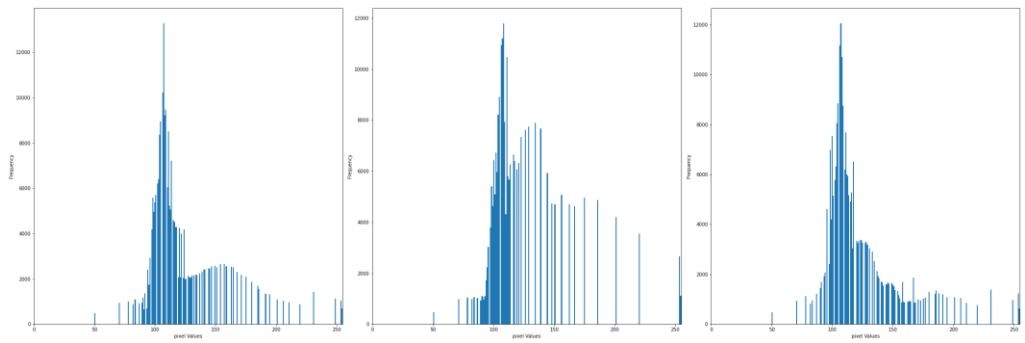


(2)Histogram matching 下圖為三張測試圖片以及要將三張圖片轉換的目標 histogram

轉換的目標圖片以及 histogram



下圖為轉換後的結果圖以及 histogram



2. Convolution

原圖

(1) Gaussian filter

Kernel size 3

Kernel size 5

Kernel size 7

Kernel size 33

(2) Averaging filter

Kernel size 3

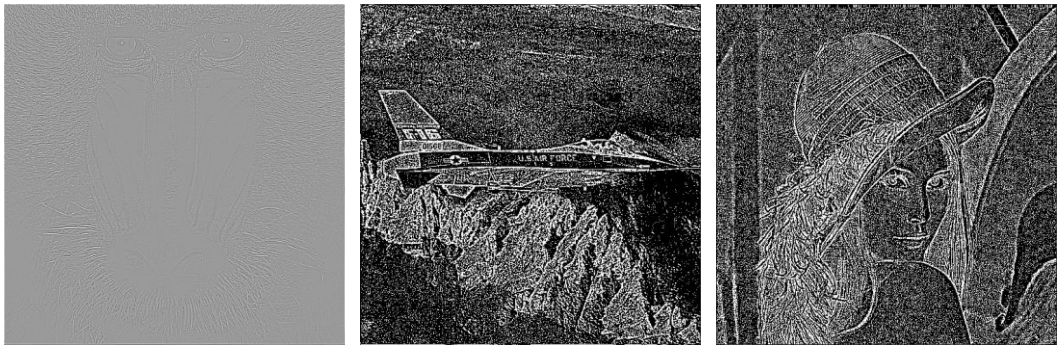
Kernel size 5

Kernel size 7

Kernel size 33

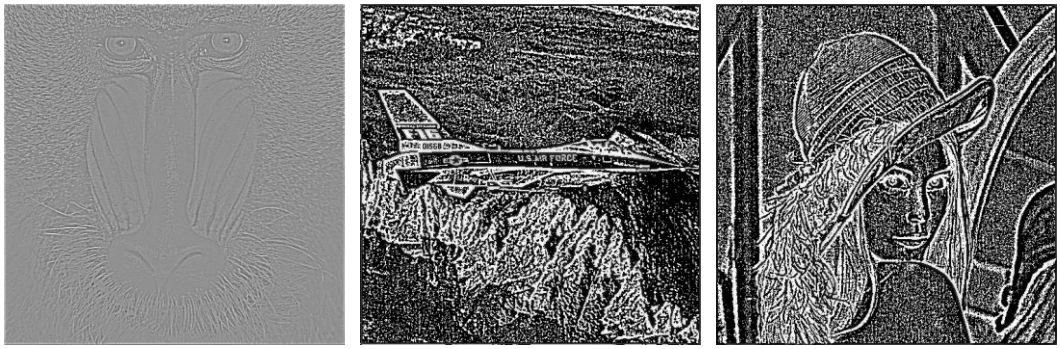
(3) Unsharp mask filter

Kernel size 3



Kernel size 5

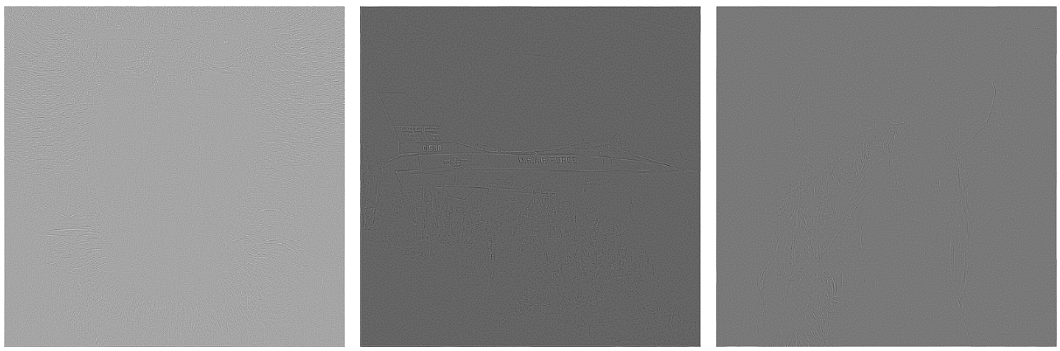
Kernel size 7



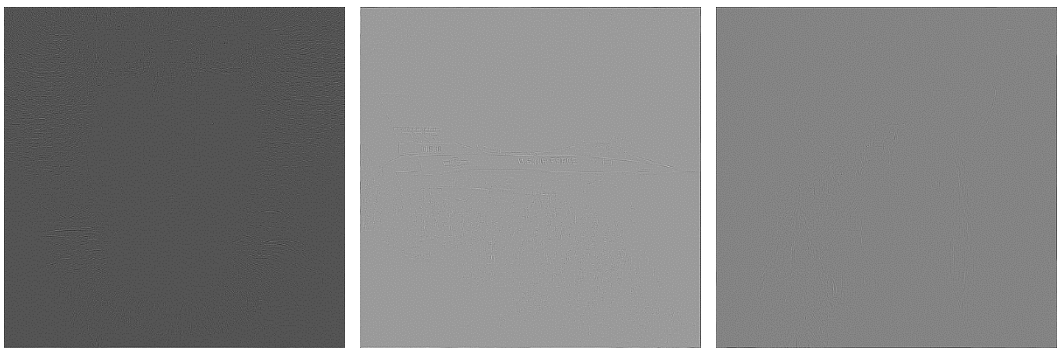
(4) Laplacian filter

Kernel size 3

Kernel size 5

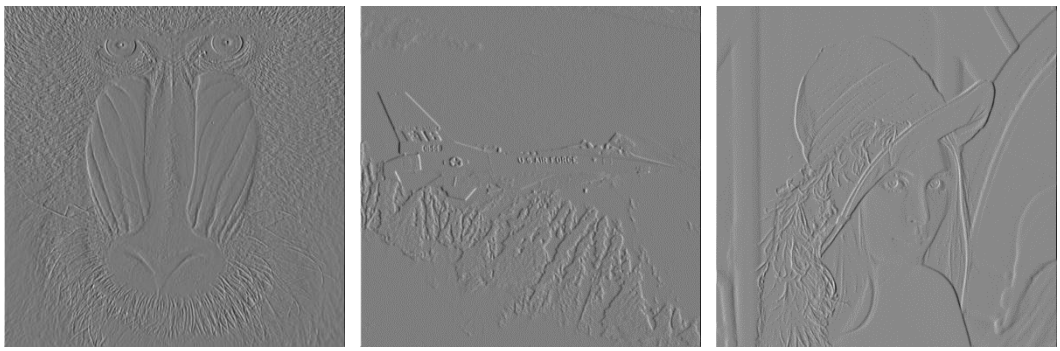


Kernel size 7

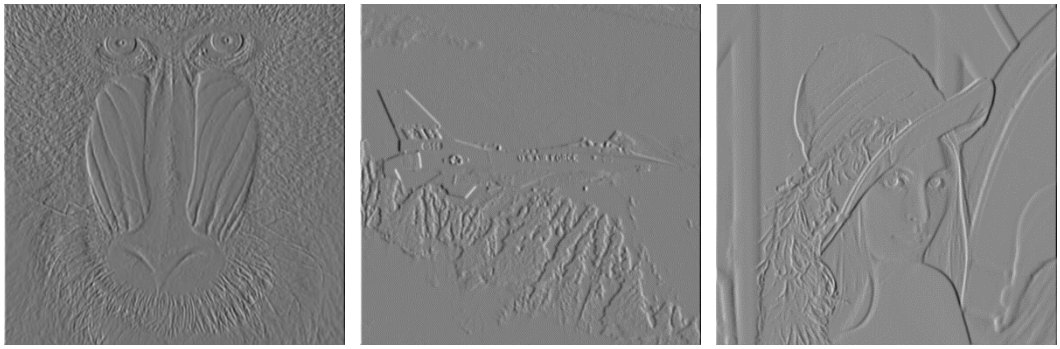


(5) Sobel filter 方向為水平(得到方向水平的邊緣)

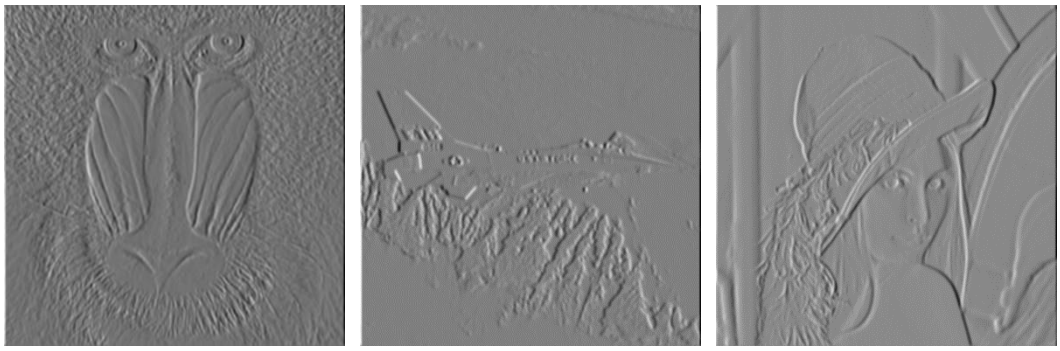
Kernel size 3



Kernel size 5

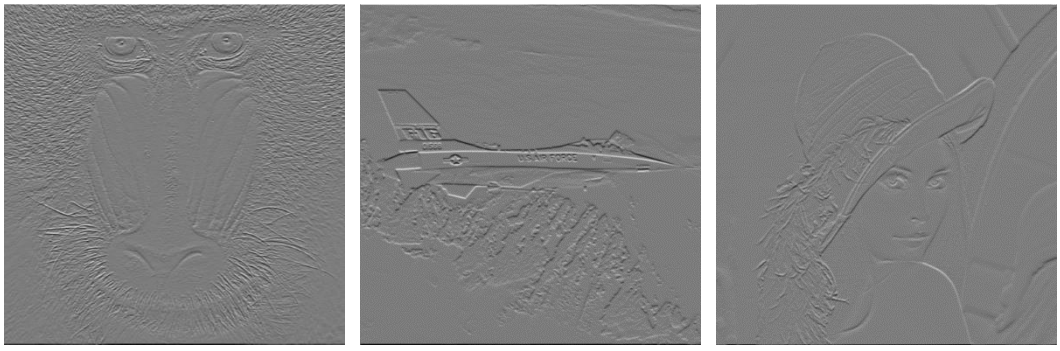


Kernel size 7

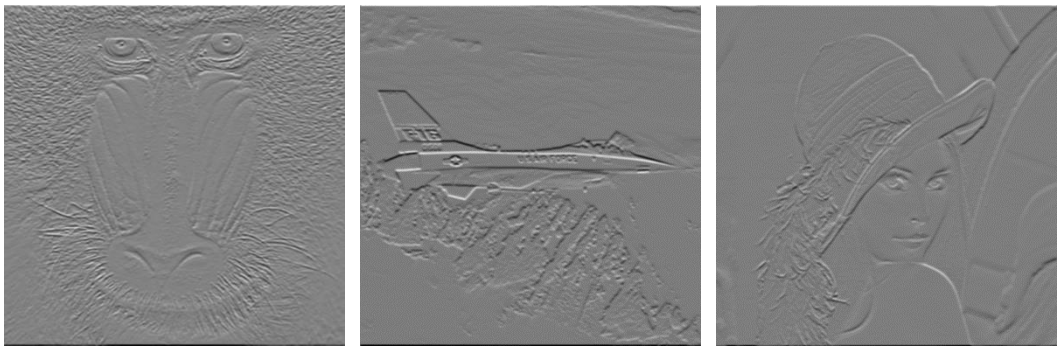


(6) Sobel filter 方向為垂直(得到方向垂直的邊緣)

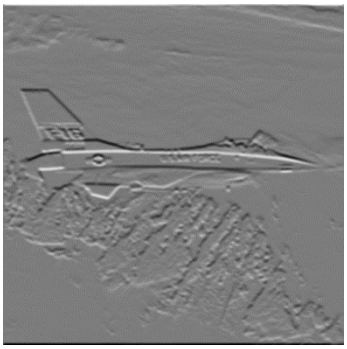
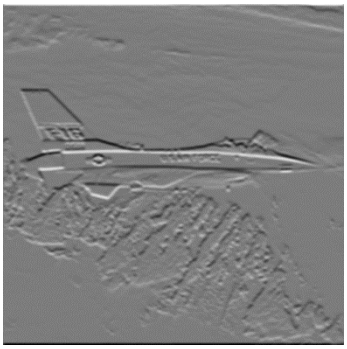
Kernel size 3



Kernel size 5

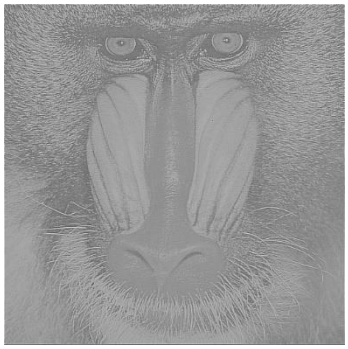


Kernel size 7

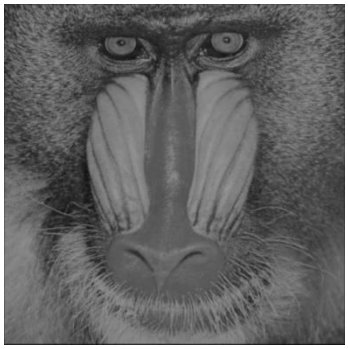
3. Convolution-2

(1) Kernel-1



這個 kernel 把圖片中比較平滑的部分變的更加平滑，使圖片視覺上變得更加清晰，並使邊緣變的更加明顯，但是他 kernel 中的數值和為 2 比 1 高，所以會使原本的圖片亮度變高，改變圖片原本亮度。

(2) Kernel-2



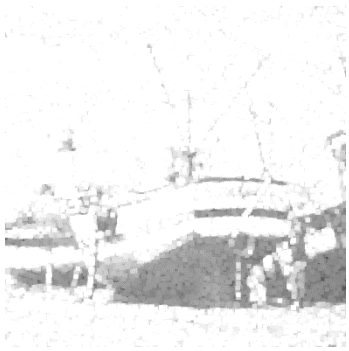
這個 kernel 使原本的圖片變稍微模糊，他的 kernel 數值和為$\frac{19}{25}$，比 1 低所以容易使圖片亮度變低，如果把左上和右上的值改完 4 則可以降低這個問題。

4. Image denoising

原圖

下圖從左而右分別為使用 minimum, median, maximum 進行降躁的圖





從圖可看出 median 的效果最好，另兩者會太暗或太亮

5. Bilateral filter


左圖為原圖，有圖為使用參數 kernel size=7, $\sigma_c = 16, \sigma_s = 8$得到的圖

6. NLM filter


左圖為原圖，右圖為使用參數 Big window size is 11, Small window size is 5,$\sigma = 0.5$得到的圖

7. NLM filter improvement

使用的改良方法參考文章

SINGLE-IMAGE DERAINING USING AN ADAPTIVE NONLOCAL MEANS FILTER

Jin-Hwan Kim , Chul Leey, Jae-Young Simz, and Chang-Su Kim (2013)

文中目標為將有雨滴的照片修成沒有雨滴的照片，文中將 NLM 修改成專門清除雨滴的演算法，方法為先計算圖片的一階和二階微分，得到一個矩陣

$$\begin{bmatrix} g_x^2(\mathbf{q}) & g_x(\mathbf{q})g_y(\mathbf{q}) \\ g_x(\mathbf{q})g_y(\mathbf{q}) & g_y^2(\mathbf{q}) \end{bmatrix}$$

其中q為圖片中的某一個 pixel 位置，q為一個二維向量，$g_x^2(\mathrm{q})$為位置q對x軸方向的二階微分，$g_y^2(\mathrm{q})$為位置q對y軸方向的二階微分，$g_x(q)g_y(\mathrm{q})$為q的x軸和y軸方向的一階微分相乘。

文中對這個矩陣做奇異值分解，用分解後的矩陣計算某個點是否判斷為雨滴，判斷方式為使用分解後的矩陣計算點q以及附近的其他點是否會組成雨滴的形狀，例如雨滴的形狀會是垂直的橢圓形，若是的話則將點q標記為 1 否的話標記為 0，下圖為文中原圖和標記後的圖。


由於資料中的圖躁聲是很多很小的點，無法使用上述方式得到哪些像素是躁聲，所以改用計算一個像素q旁邊像素的平均，若像素q的值較旁邊像素平均值太大或太小，則認定為躁聲，下圖左圖為原圖，有圖白色部分為判斷為躁聲的像素。

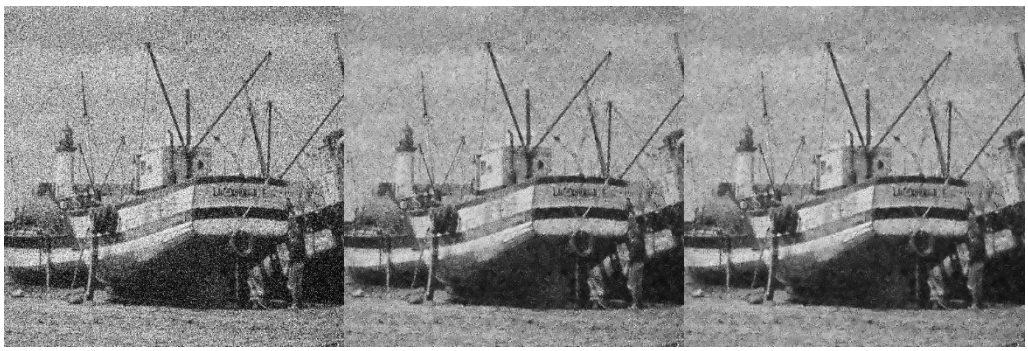

得到判斷像素是否為躁聲的圖後假設這個圖的矩陣為M且$M(q)$為 1 代表判斷q為躁聲 0 則不是躁聲。
假設要得到像素q經過 NLM 計算後的值，p為要與 q 做比較的像素則

$$\begin{aligned}\tilde{\mathbf{B}}_{\mathbf{p}} &= \mathbf{B}_{\mathbf{p}} \otimes (\mathbf{1}-\mathbf{R}_{\mathbf{p}}) \otimes (\mathbf{1}-\mathbf{R}_{\mathbf{q}}) \\ \tilde{\mathbf{B}}_{\mathbf{q}} &= \mathbf{B}_{\mathbf{q}} \otimes (\mathbf{1}-\mathbf{R}_{\mathbf{p}}) \otimes (\mathbf{1}-\mathbf{R}_{\mathbf{q}})\end{aligned}$$

其中B_p, B_q為像素 p, q 附近的點，R_p, R_q則代表想素 p, q 在矩陣 M 附近的點。最後 NLM 式子為

$$\hat{I}(\mathbf{p}) = \frac{\sum_{\mathbf{q}} \exp\left(-\frac{\|\tilde{\mathbf{B}}_{\mathbf{p}}-\tilde{\mathbf{B}}_{\mathbf{q}}\|^2}{\sigma^2 N_{\mathbf{p},\mathbf{q}}}\right)(1-\mathcal{M}(\mathbf{q}))\, I(\mathbf{q})}{\sum_{\mathbf{q}} \exp\left(-\frac{\|\tilde{\mathbf{B}}_{\mathbf{p}}-\tilde{\mathbf{B}}_{\mathbf{q}}\|^2}{\sigma^2 N_{\mathbf{p},\mathbf{q}}}\right)(1-\mathcal{M}(\mathbf{q}))}$$

下圖左圖為原圖，中間為原本 NLM，右圖為改進後的結果



程式碼網址: https://github.com/ss9636970/img_filters