影像處理與電腦視覺 作業二

統計所 李權恩 R26094022

本次作業主要以 Python 完成，最後再以 tkinter 套件將程式包裝成一個使用者介面以及 pyinstaller 包裝成.exe 檔。

相關程式碼在此：

https://github.com/Quan-En/DIPCV/tree/main/assignment2

.exe 執行檔下載位置：

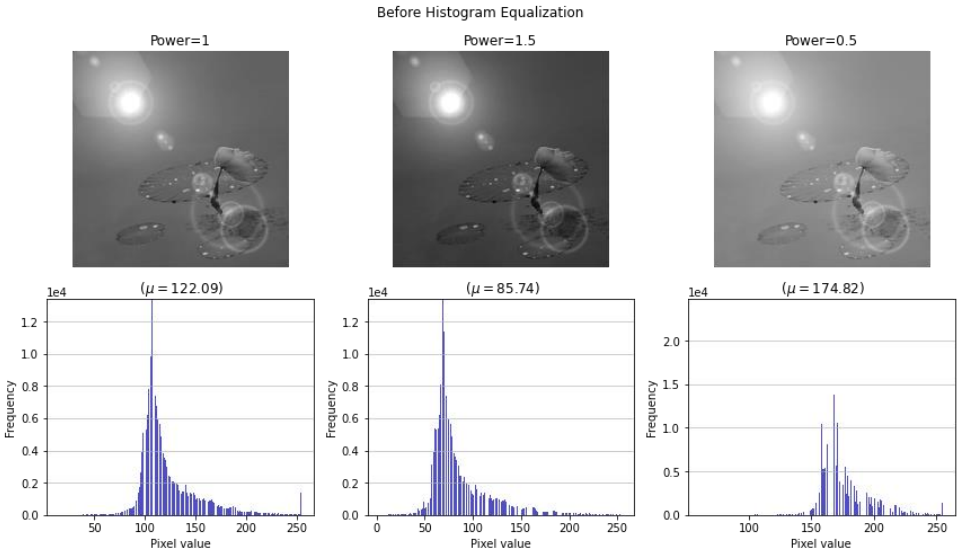
https://github.com/Quan-En/DIPCV/blob/main/assignment2/dist/gui_viewer.exe

Problem

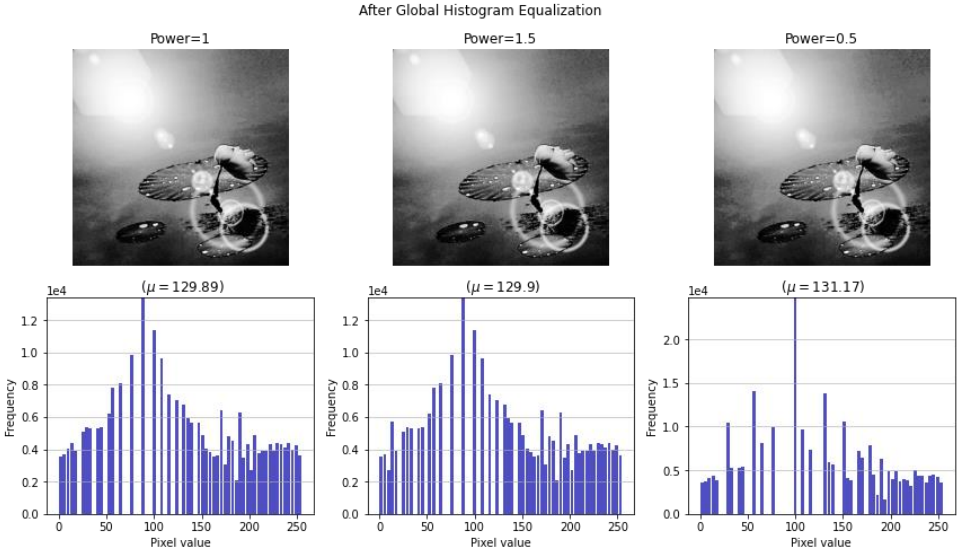
1. Histogram Equalization: 在進行 Histogram equalization 之前先使用伽瑪校正(Gamma correction)將圖片的亮度進行調整，在此處使用兩種不同的設定來增加/降低圖片的亮度，設定分別為$\gamma_1 = 1.5, \gamma_2 = 0.5$。下方為轉換後的圖片：

i. 利用調整過後的圖片進行 Histogram equalization，下方將以 flower.raw 為例：

下圖為 flower.raw 在未進行 global histogram equalization 的視覺化效果與其亮度值的直方圖，藉此可以得知其亮度值的分布有一明顯高峰，而根據伽瑪校正的不同係數而有不同的集中趨勢位置。

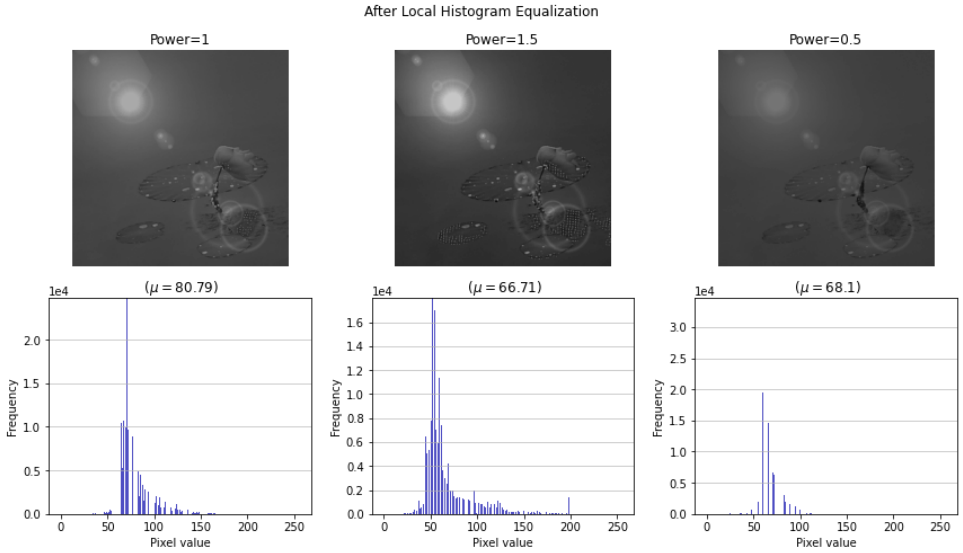


以視覺化的效果來說，上方的亮度值分布並不理想，因此使用 histogram equalization 來轉換亮度值得分布來更好的呈現亮暗對比。Histogram equalization 是機率論中，機率積分轉換(Probability integral transform)的應用，考慮不同圖片的亮度值分布為一隨機變數，利用其累積分布函數將隨機變數進行轉換後皆為一均勻分配，至此便可使用上述的轉換過程重新調整亮度值，下圖為視覺化轉換後的圖片結果以及其亮度值的直方圖：

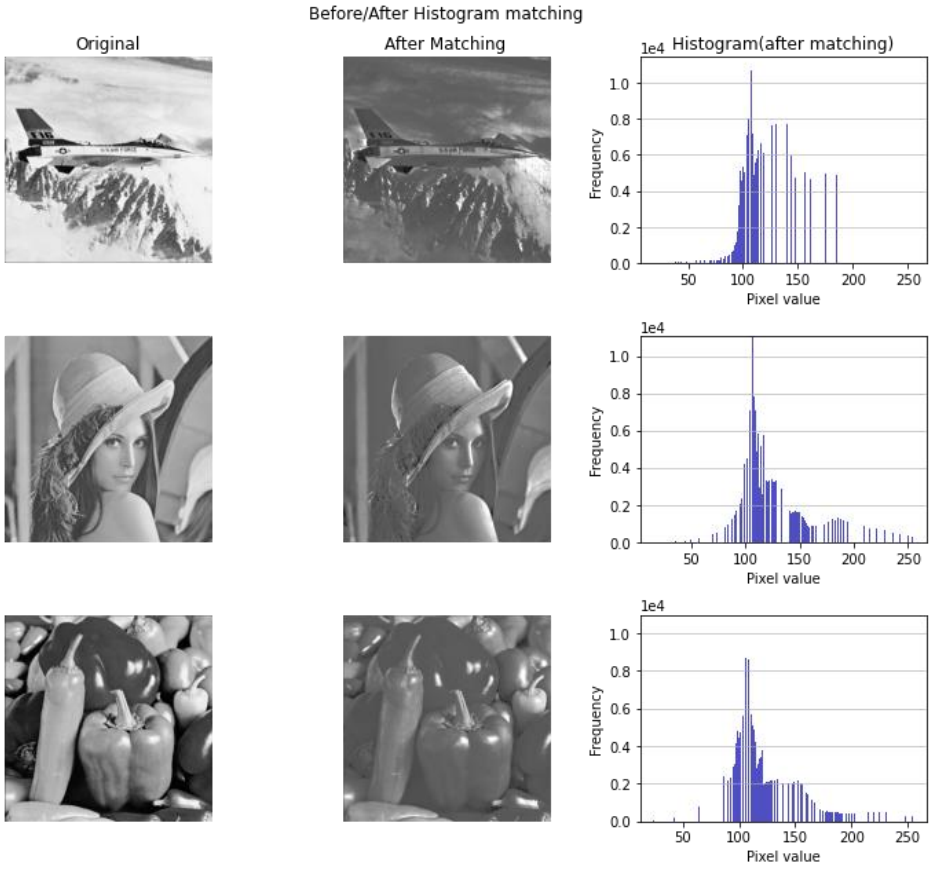


ii. 使用 local histogram equalization 對這三幅圖像進行增強，最後保留 windows size=10 的結果，下圖以 flower.raw 作為範例：

此外在進行局部增強之後為了確保值域仍在 0 至 255 之間，需先將增強後的亮度值單位化到 0 至 1 再乘上 255。



iii. 參考 flower.raw 進行 Histogram matching: 以下展示將三張圖片各自映射至參考圖後的結果，並展示其 histogram matching 後的直方圖。

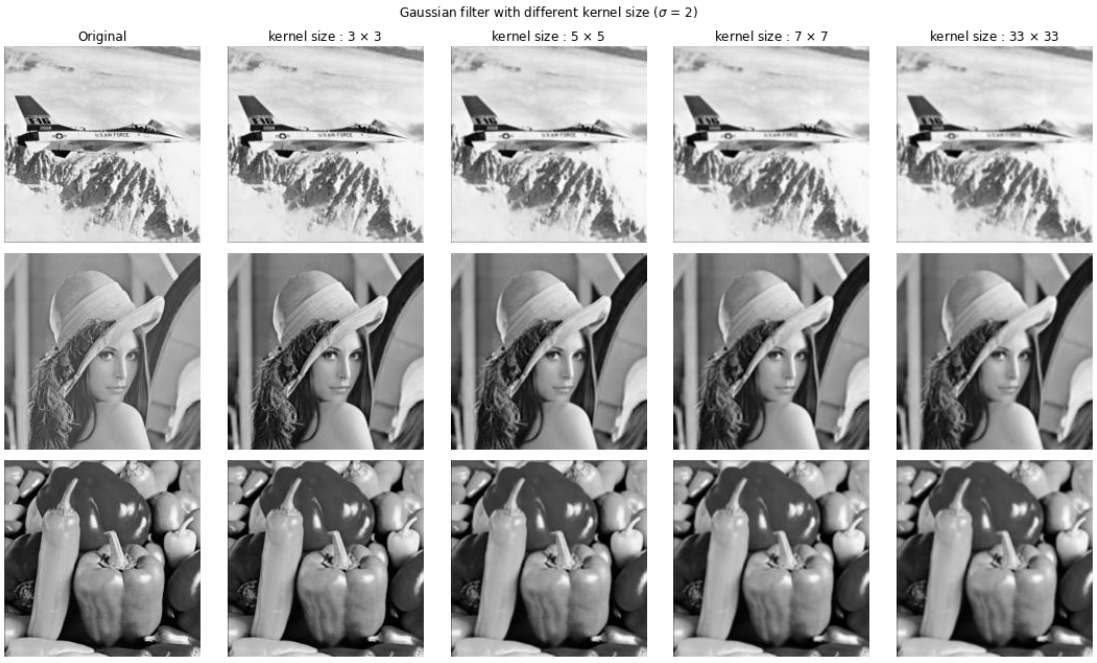


2. Convolution:

在利用各式各樣不同 kernel 來進行 Convolution 前，需根據 kernel size 的大小來對圖片進行邊緣填充(padding)，而填充方式也有不同的作法，本篇主要採用鏡像填充(reflection padding)，目的是希望圖片的邊緣在 convolution 後不會因為 kernel size 越來越大時而變成全黑。

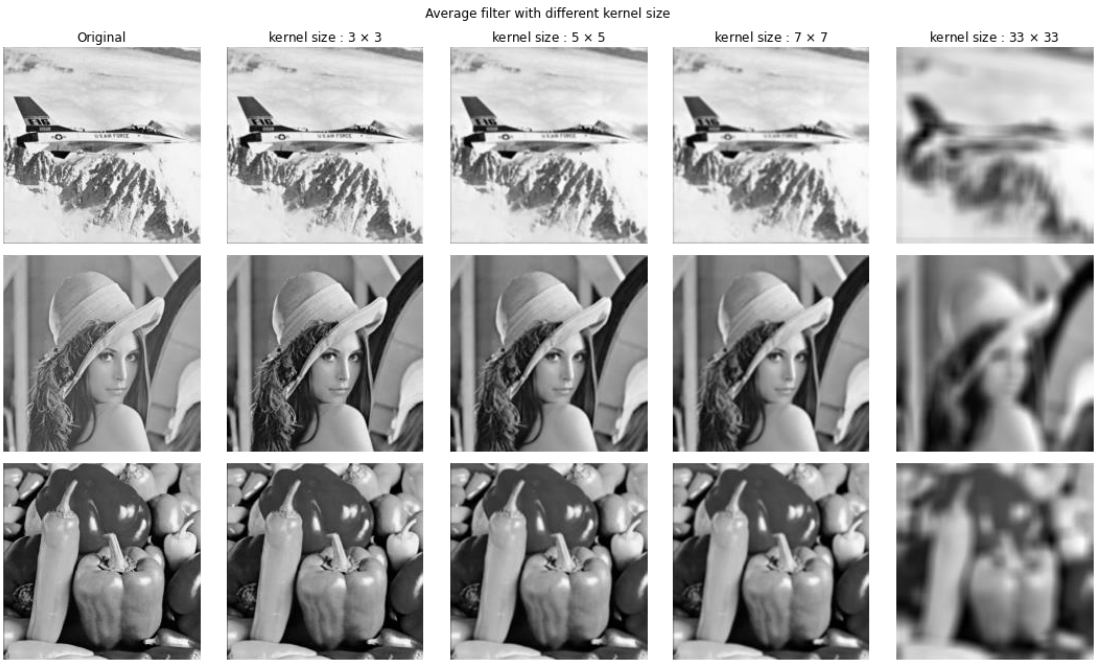
i. Gaussian filter:

Gaussian filter 主要是根據 Gaussian distribution 的 density function 來決定 kernel 每個位置的權重，越接近中心權重越大，下圖為 3 種不同圖片利用 Gaussian filter 的結果，最左行代表原圖，剩餘行數由左至右分別代表不同的 kernel size($3 \times 3, 5 \times 5, 7 \times 7, 33 \times 33$)。而 Gaussian filter 除了 kernel size 外須額外提供 σ 來計算 kernel 的權重，在本次實作中考慮 $\sigma = 2$。最後隨著 kernel size 加大，模糊的效果也愈強。



ii. Averaging filter:

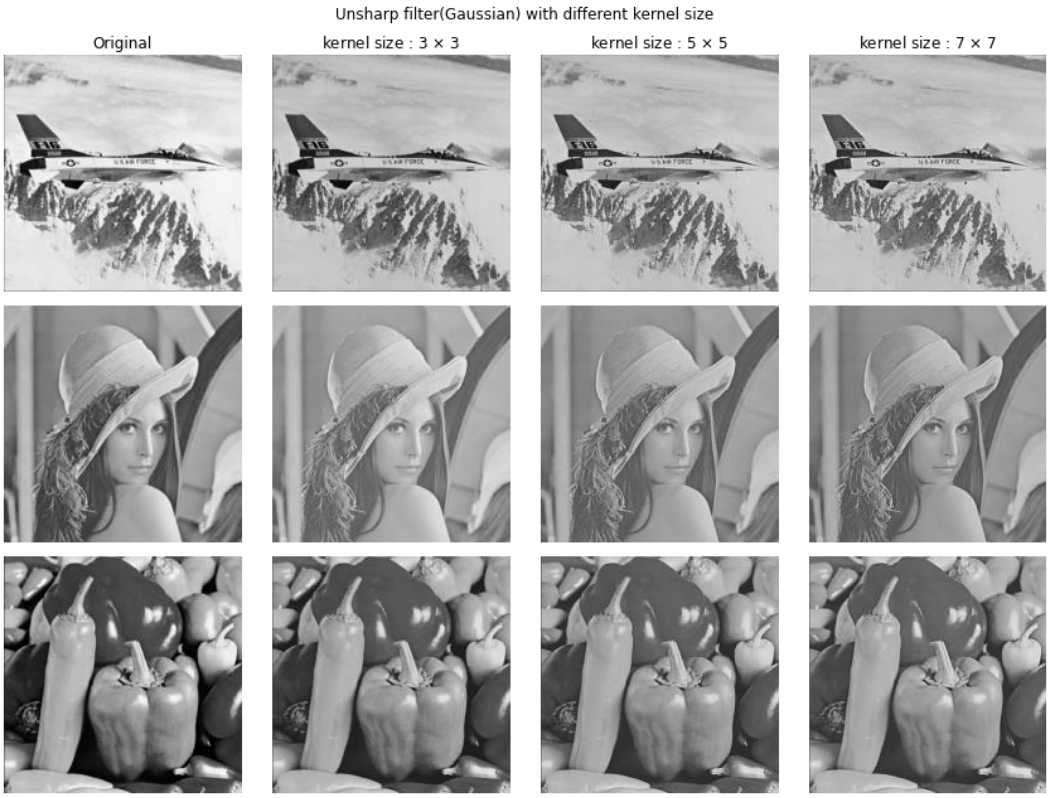
Averaging filter 主要使用平均數的方式來進行 convolution，因此在 kernel 中每個位置的權重皆相同，而 kernel size 的大小也影響著權重的大小。下圖為 3 種不同圖片利用 Averaging filter 的結果，最左行代表原圖，剩餘行數由左至右分別代表不同的 kernel size($3 \times 3, 5 \times 5, 7 \times 7, 33 \times 33$)。與 Gaussian filter 相似，皆有將圖片糊化的效果，但不同的是，隨著 kernel size 的提高，Averaging filter 的糊化效果比 Gaussian filter 更強。



iii. Unsharp mask filter:

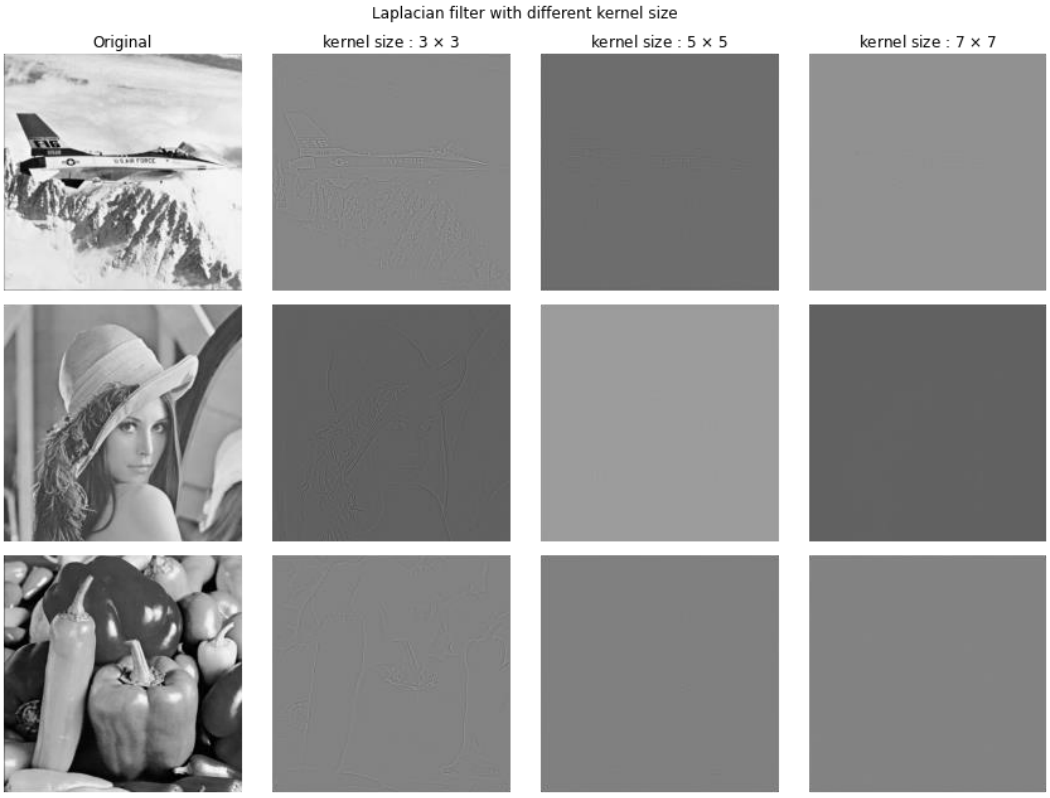
Unsharp mask filter 主要強化圖像當中的邊緣，為了能夠凸顯邊緣，可以考慮比較原始圖片與糊化圖片之間的差異，再將這份差異值乘上一指定倍數加回原始圖片，藉此達到強化邊緣的效果。

下圖為不同圖片根據不同 kernel size 進行 convolution 的結果，糊化圖片的取得方式皆是使用 Gaussian filter，$\sigma = 0.5$，原始圖片與糊化圖片的差異值會乘上0.5後加回原始圖片。



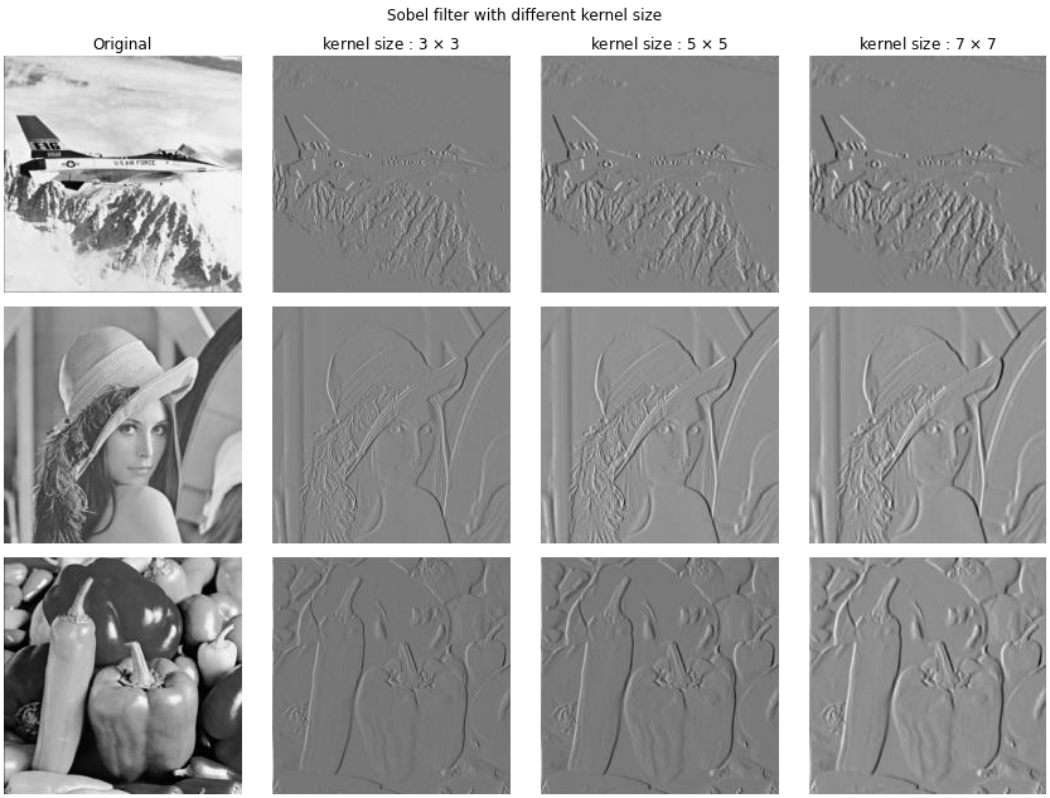
iv. Laplacian filter:

Laplacian filter 是一種邊緣檢測器，用於計算圖像的二階導數，測量一階導數變化的速率，藉此判斷相鄰像素值的變化是來自邊緣還是連續進展。Laplacian filter 的 kernel 通常包含交叉模式中的負值，以數組為中心。corner 是 0 或正值。 中心值可以是負值也可以是正值。下圖為使用不同 kernel size 的 Laplacian filter 所產生的結果。



v. Sobel filter:

Sobel filter 與 Laplacian filter 相同，主要用於邊緣檢測，kernel 的取得方式為橫向及縱向的亮度差分近似值，下圖為使用不同 kernel size 的 Sobel filter 所產生的結果。

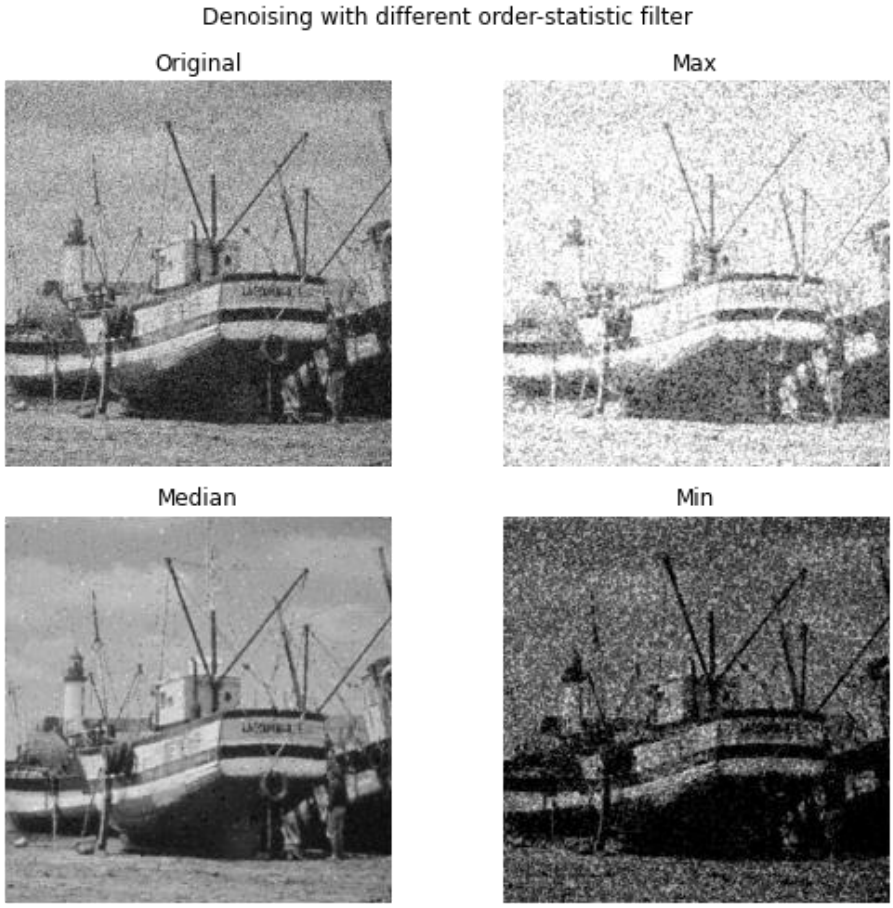


3. Convolution-2：根據給定的兩種不同特殊 kernel 來討論性質與潛在問題並視覺化結果。

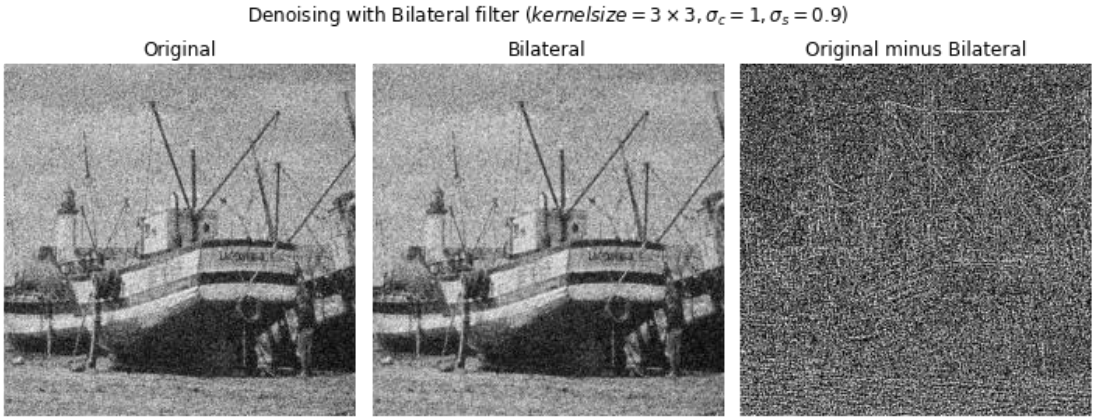
   在實作過程中，考慮到值域可能在 convolution 後超過 255 或小於 0，因此對於整張圖片經過 convolution 後有在進行調整值域的動作，先單位化到[0,1]之間再乘上 255，藉此重新調整至 0 到 255。而在第一個 kernel 上它主要放大了每個 pixel 的亮度值，因此在 convolution 且調整值域過後整張圖片會看起來亮暗不明顯；而在第二個 kernel 上如果不進行值域調整，圖片會有變暗的效果，因為 kernel 內部所有權重的總和小於 1，此外也會有糊化的效果。

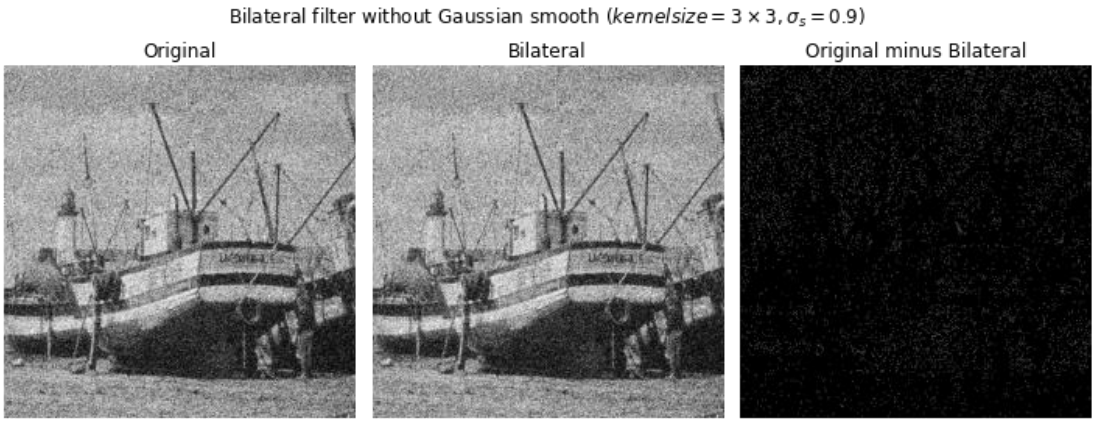

Special kernel
Original
kernel-1
kernel-2

4. Denoising：利用 Noisy.raw 進行降噪任務。以順序統計量的濾波器進行，並視覺化降噪後的結果。在圖像當中被視作噪音的亮度值常常會是與周遭像素的亮度值相差太遠的極端值，因此我們可以考慮使用中位數作為 filter 來過濾噪音。下圖為使用最大值、中位數、最小值三種不同順序統計量進行處理，可以從右下角利用中位數處理的圖像中看出不錯的降噪效果。



5. Bilateral filter：Bilateral filter 同時根據兩種 pixel 之間的距離以及像素之間的差異來進行 convolution，藉此可以在平滑化的過程中保留邊緣，但潛在問題在於此 filter 的權重是 pixel by pixel，無法較泛化的加速程式的運算。下方兩張圖分別代表有/無使用 Gaussian 平滑 kernel 的 Bilateral filter。

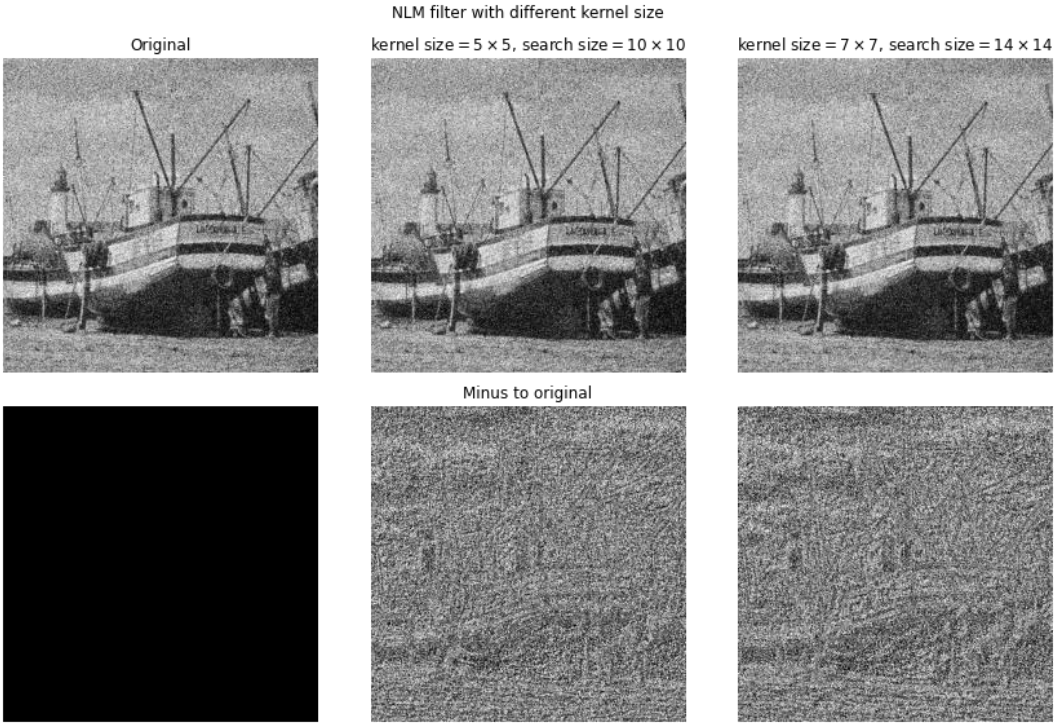




6. Non Local Means (NLM) filter：

實現簡單的 NLM 以對 Noisy.raw 進行降噪，內核大小分別為 5×5、7×7。

下圖為實驗後的視覺化結果。在下圖中的第二列是將第一列的圖與原始 Noisy.raw 相減來觀察降躁的結果，在不同 kernel size 的圖中除了黑白雜訊外都還有一點原始圖片的邊緣。



7. 性能和效率提升：請嘗試以任何方式加速 NLM 過濾器