HOMEWORK REPORT

By Beltus Nkwawir Wiysobunri -- 704181021

Requirements for Running the Code

This program was written using Ubuntu 18.04.3 LTS operation system and the programming language I used was python version 3.6. In order to successfully run the code written for this homework, the following requirements may be mandatory.

- Create a virtual python environment and ensure it is activated
- Install python 3.6.
- Install jupyter notebook in your environment
- Install the opencv, matplotlibrary, numpy , scipy libraries which were the cardinal backbone behind the success of this project.
- Link to github https://github.com/Beltus/Advanced-Computer-Vision-Course

Link to Code on Github

https://github.com/Beltus/Advanced-Computer-Vision-Course

Step 1: Convert Color Image to Gray Scale.

The original image was converted to a grayscale image by first separating it into its various channels (R,G,B). Then computing individual channel average values. Finally, combining these average values of the R, G, and R channels to get the resulting grayscale image.

Figure 1 shows the R, G and B channel images. Figure 2 shows the resulting grayscale image.

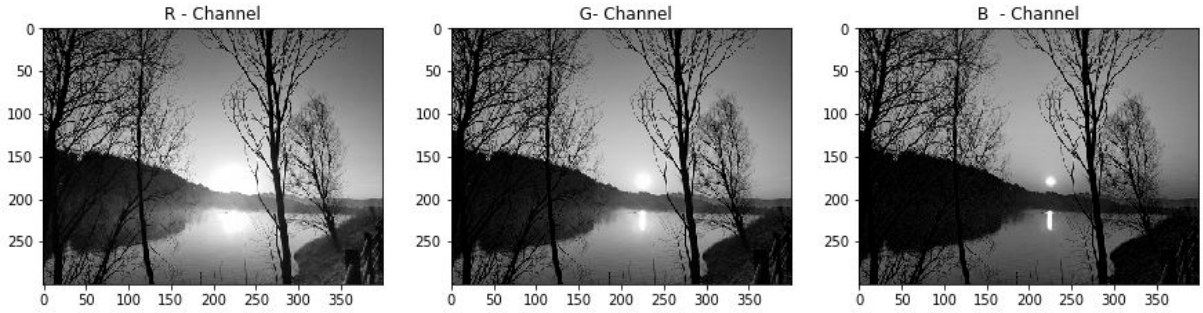

Figure1: Red, Green and Blue Channel images

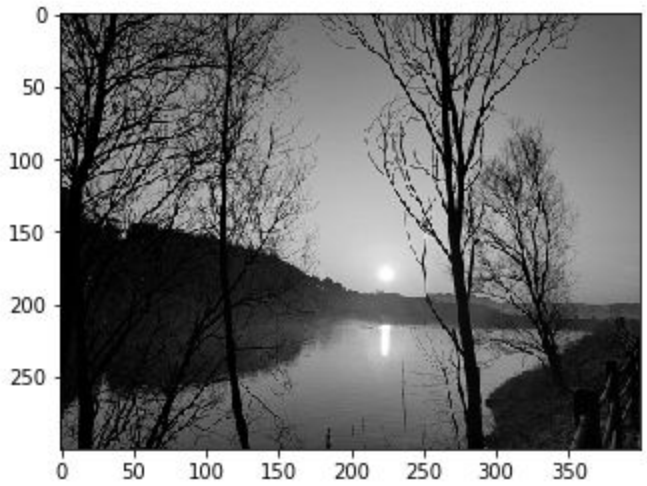

Figure 2: Grayscale image

Step 2: Compute Histogram from Grayscale Image

Matplotlibrary histogram function was helpful here to compute the histogram of the grayscale image. Figure 3 below shows the results:

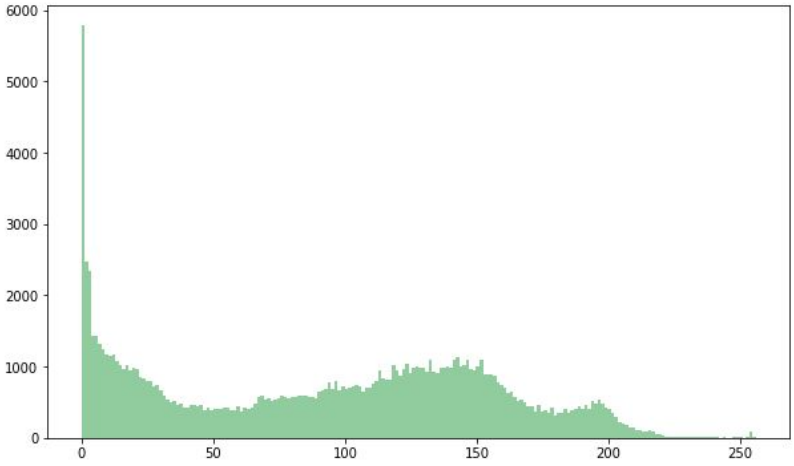

Figure3: Histogram

Step 3: Converting Image from Grayscale to Binary.

Based on the histogram plot above, and after some experimentation with a series of threshold values, the best results were obtained with a threshold value of 50. Figure 4 shows the output.

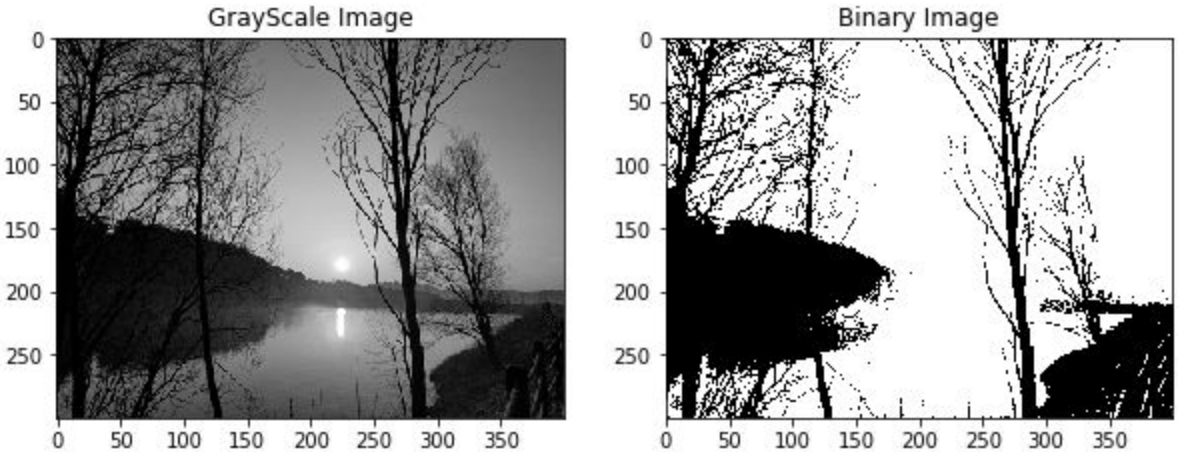

Figure 4: Grayscale to Binary image

Step 4: Adding Zero-Mean Gaussian Noise to Respective Channels.

I implemented a function for adding a zero-mean Gaussian noise for varying standard deviations. The values for standard deviation tested were 1 , 5 , 10 , 20.

Figure 5 shows R, G, and B channel images after adding Gaussian noise with a standard deviation of 20.

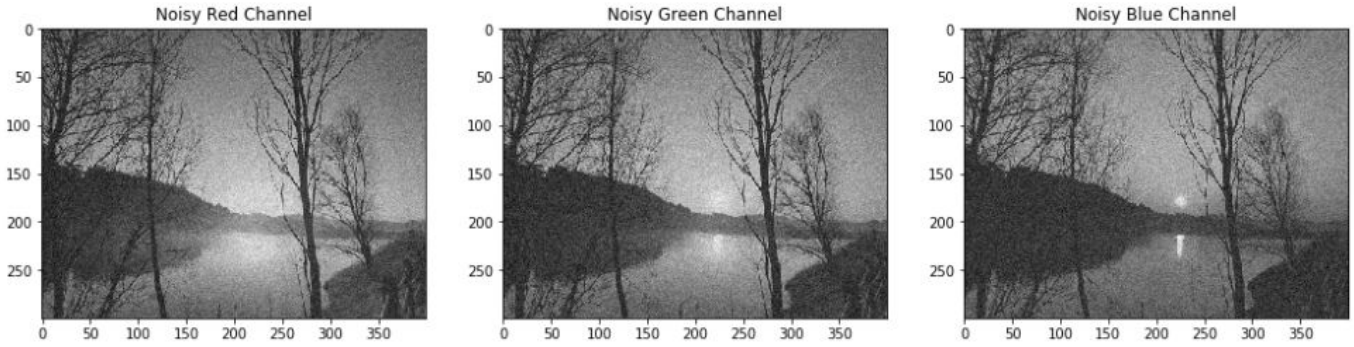

Figure 5: Gaussian noise with a standard deviation of 20 on R, G, and B channels.

Step 5: Grayscale Images from Average values of R, G, B channels For Varying Gaussian Noise.

In this step, for each value of the standard deviation, I computed the corresponding Gaussian noisy image for the R, G, and B channels. By taking the average of the respective channels and then combining the results, I obtained the grayscale average noisy image. Figure 6 below shows the results obtained.

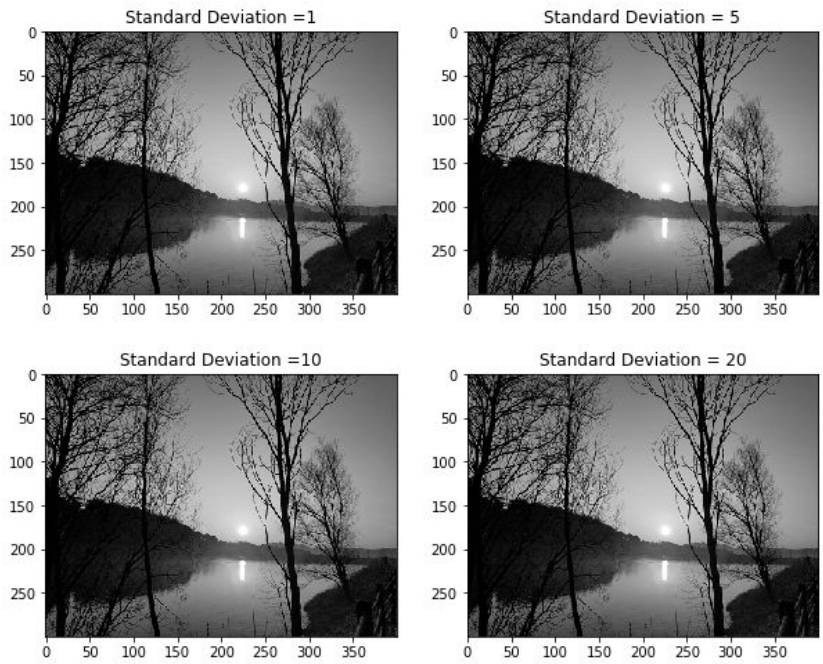

Figure 6: Grayscale images from average RGB channels with varying Gaussian noise levels

Step 6: Low-Pass Filtering of Images

a) Using Mean Filter.

In order to filter the images, I used the mean filter with 2 different kernels shown below:

$$h_1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad h_2 = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 7a shows the results obtained using the 3 x 3 h1 kernel. Figure 7b shows the results obtained using the 5 x 5 h2 kernel.

By visually inspecting these images, we can see that, in Figure7a, the images appear t**o be smoother (more blur)** than the images in Figure 7b. This is because, the smaller the kernel size, the greater is the blurring effect.

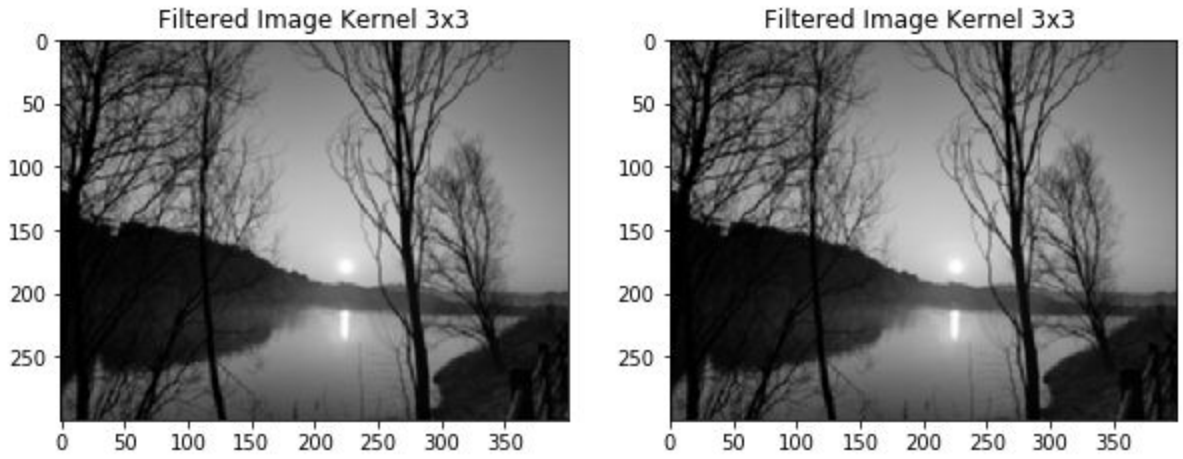

Figure 7a: Images Filtered with using a mean filter of kernel size 3 x3

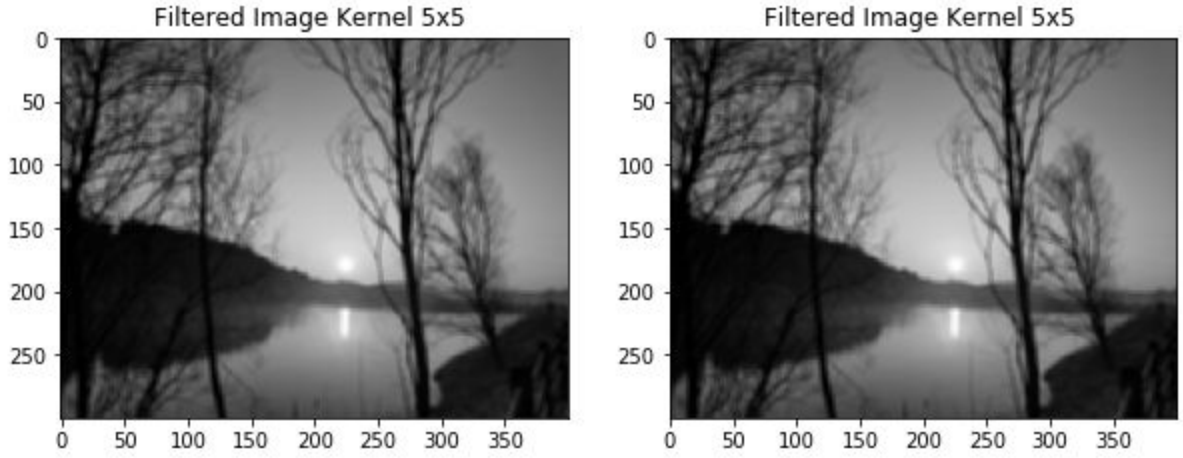

Figure 7b: Images Filtered using a mean filter of kernel size 5 x5

b) Using Gaussian Filter.

After applying the 3 x 3 Gaussian kernel to the averaged grayscale images the results can be seen in Figure 7c below.

$$h = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

After some experimentation with kernel size, I observed that using a **larger kernel size decreased the smoothing(blurring) effect.** The converse is true

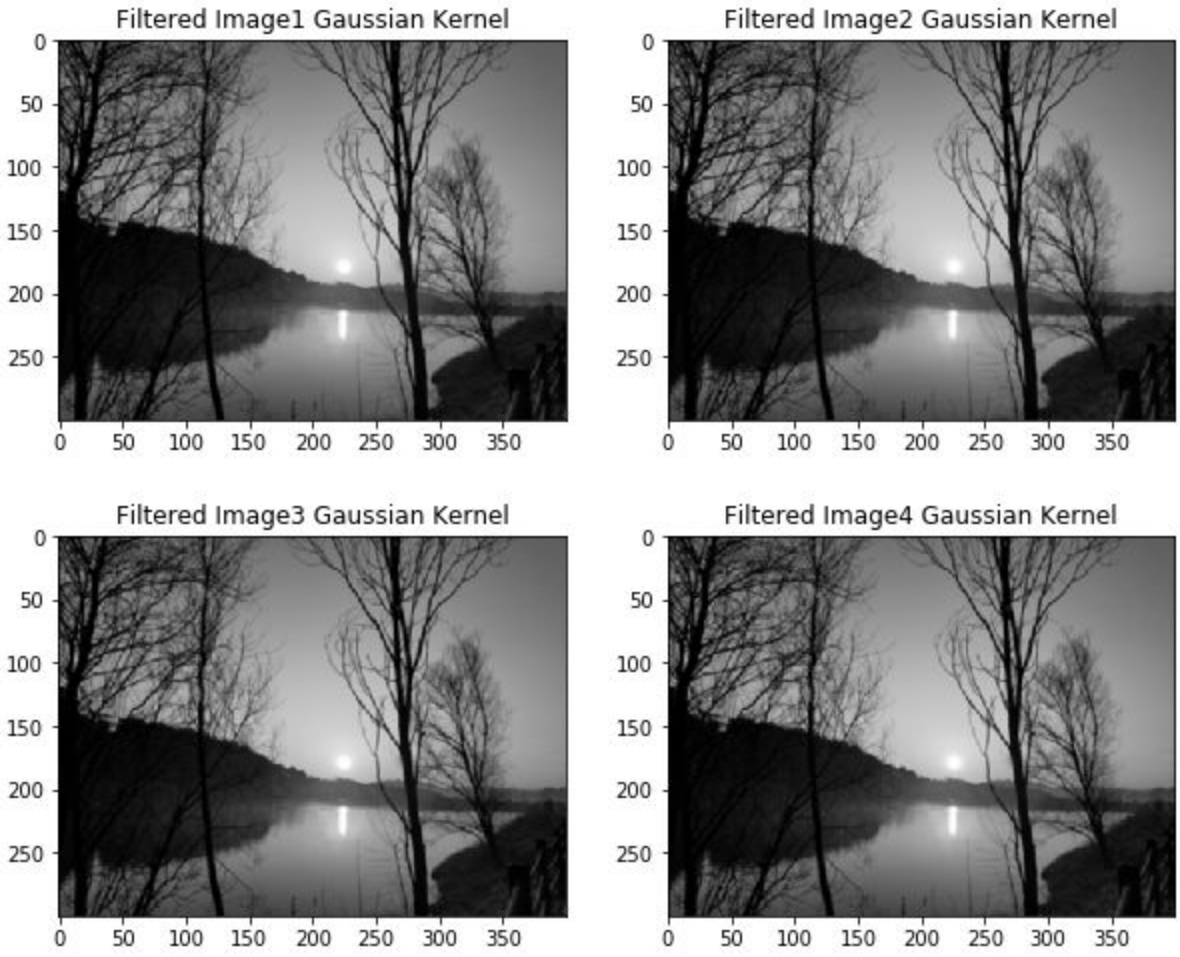

Figure 7c: Images Filtered with using a mean filter of kernel size 3 x3

Step 7: High-Pass Filtering of Images

a) Using Laplacian Filter.

Using the Laplacian high-pass filter below, the four averaged grayscale images where filtered and the results are shown in Figure 8a.

$$h_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

High-pass filtering on the images allows the high-frequency content while filtering out the low-frequency content. It can be observed from the gray images that, high-frequency components are low, resulting in an almost complete loss of image content.

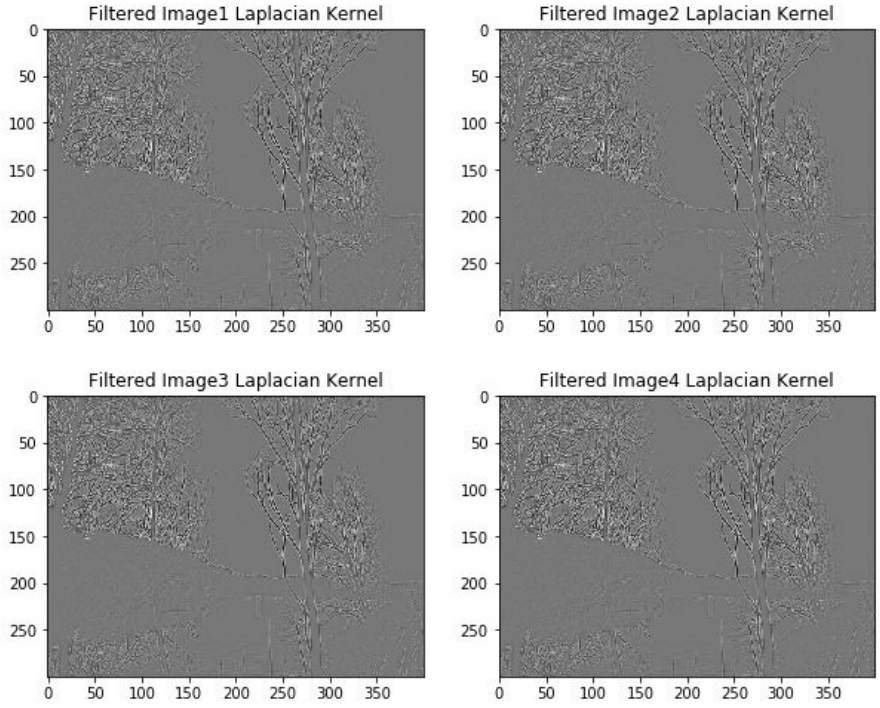

Figure 8a: Images Filtered with using an h1 laplacian filter of kernel size 3 x3

Using another variant of the Laplacian filter shown below. I obtained almost similar results like the images shown in Figure 8a.

However, by visual inspection, the h2 filter results are little bit better compared to results obtained using the h1 filter. The final high-pass filtered image results are presented in Figure 8b

$$h_2 = \begin{bmatrix} 0.17 & 0.67 & 0.17 \\ 0.67 & -3.33 & 0.67 \\ 0.17 & 0.67 & 0.17 \end{bmatrix}$$

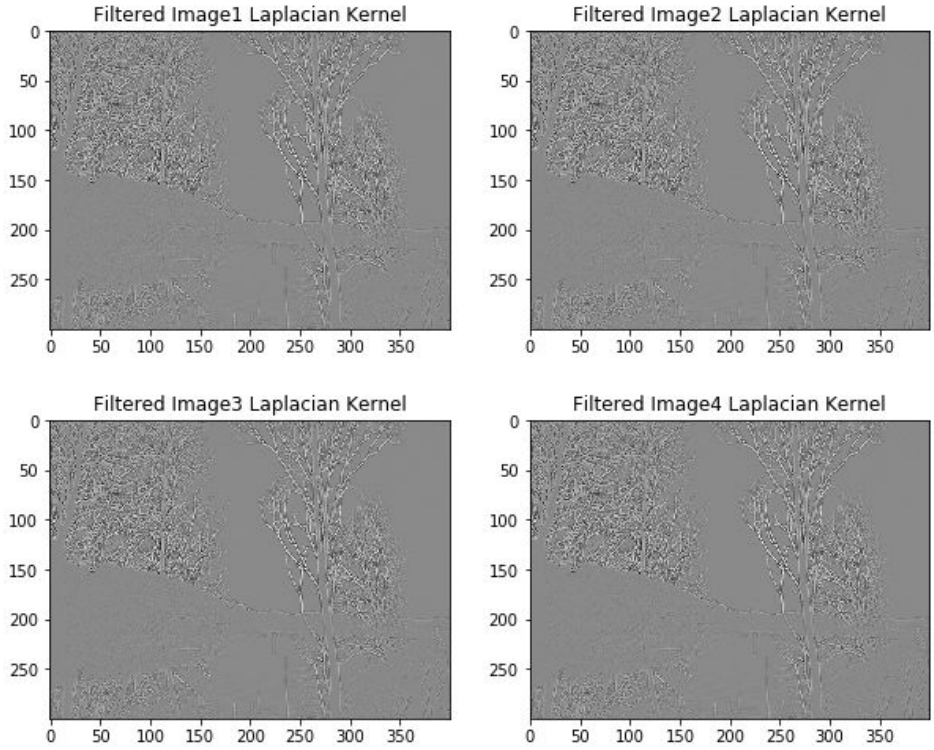

Figure 8b: Images Filtered with using h2 laplacian filter of kernel size 3 x3

b) Using High-Boost Filter

Using the high-boost filter shown below, I obtained different results for different values of the constant A.

$$h_{HB} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9A-1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

For A = 1.1, the results are shown in Figure 8c and for A = 1.5, the results are shown in figure 8d.

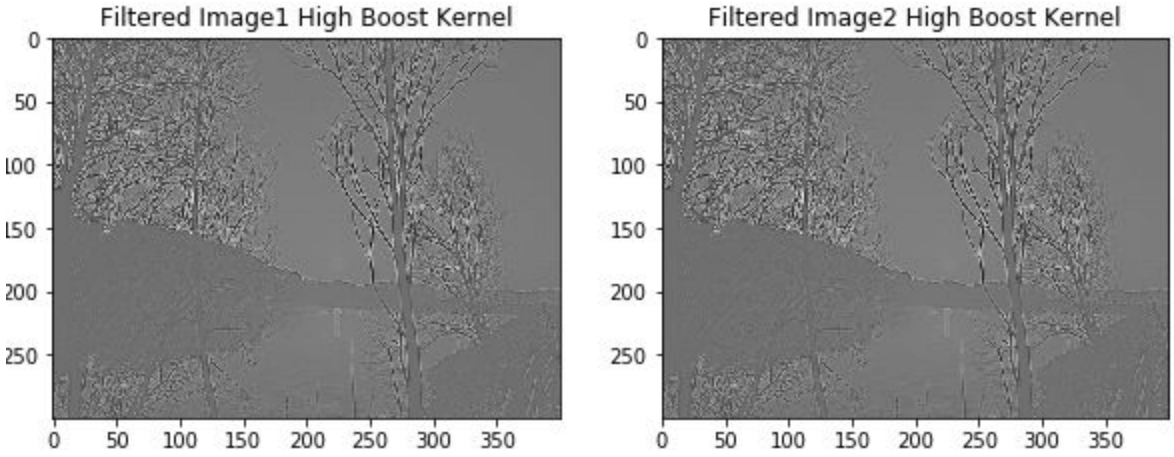

Figure 8c: Images Filtered with using high-boost high-pass filter with A = 1.1

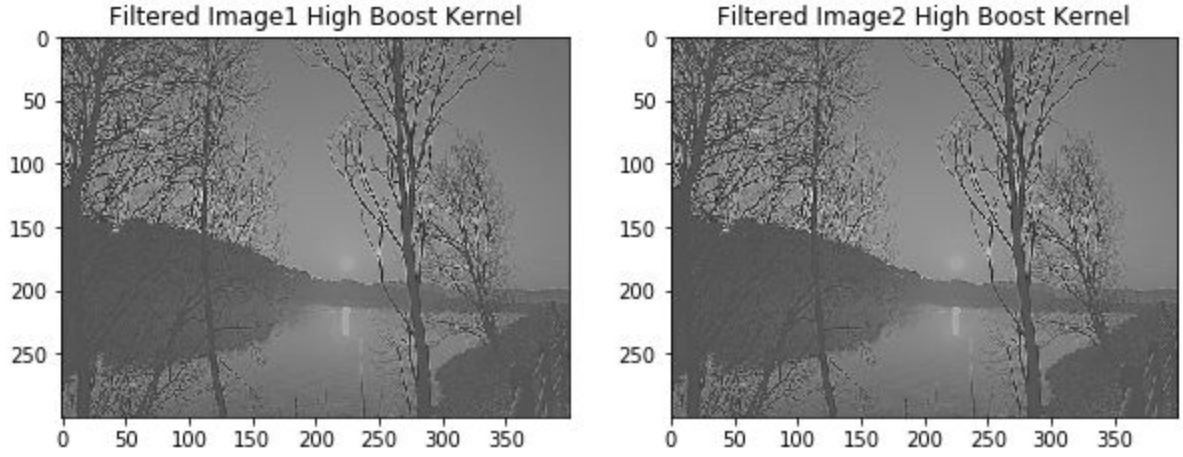

Figure 8d: Images Filtered with using high-boost high-pass filter with A = 1.5

It can be concluded that the smaller the constant A, the sharper the image and converse is true.

Comments On Figure 1

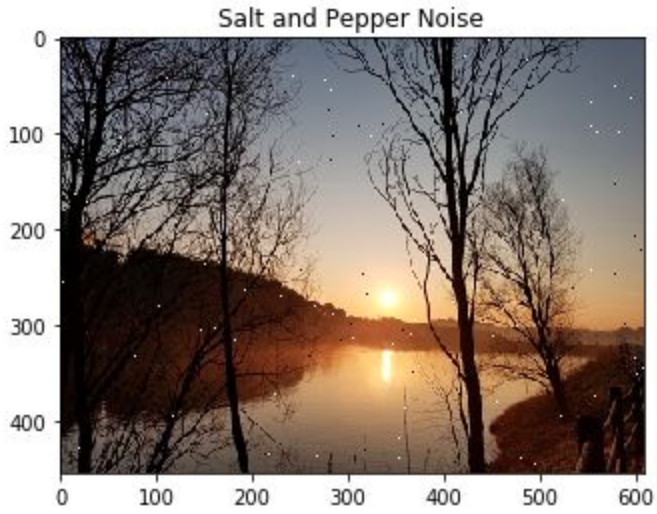

Figure: Noisy Image

It can be seen clearly that the image contains some random noise. This type of noise is called the **Salt and Pepper noise.**
In order to de-noise the image, I used a **median filter** which is a non-linear filter. The results after applying the median filter can be seen in the figure below: The noise is completely removed.

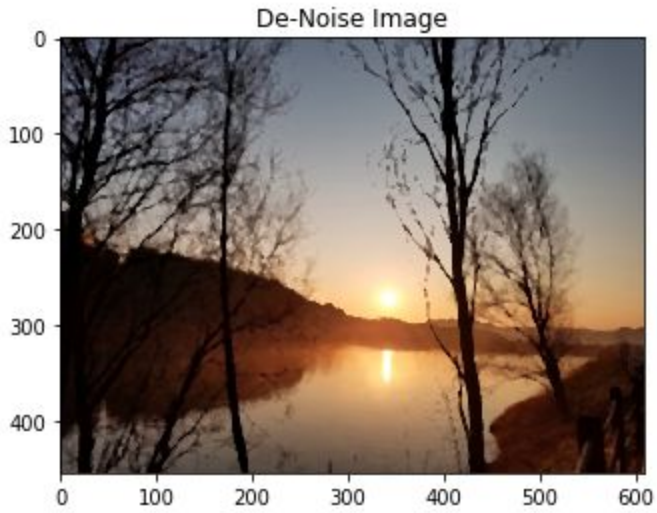

Figure: Denoised Image

Link to Code on Github Repository

https://github.com/Beltus/Advanced-Computer-Vision-Course