**ENGN 2605 Image Understanding**

**Lab 02**

**Instructor: Prof. Benjamin Kimia**

**Name: Kuan-Min Lee**

**Student ID: 140481859**

**Part I: 2D Convolution**

**Problem 1. Implement a 2D convolution**

For the first question, the following function needs to be implemented into MATLAB code:

(y: the convolution outcome matrix, x: original matrix, h: filter matrix

i,j: index for original matrix for convolution, m,n; index for storing convolution outcome)

Besides, from the description of the manual, zero padding is also needed in order to maintain the size of the convolution outcome to be the same as the input.

Consider all the above factors, the below flow chart demonstrates the working process of the code implemented:

Input Image

Zero Padding

Convolution

Figure 1: Flow Chart of Code Mechanism

And the following pseudocodes demonstrate the idea of zero padding and convolution from the my\_conv function from lab02.m file:

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Algorithm 1: Zero Padding\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Input Data:** image, filter

**Calculate:**

**Create:**

Zero Padded Image matrix with size of:

{Length: length of image+2\*Width of Padding, Width: width of image+2\*Width of Padding,

Channel: number of channels from image}

Zero Padding temp (empty variable for storing)

**Looping from** 1 to Width of Padding\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Looping from** 1 toChannel of image\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Create:**

1. Zero Padding for Vertical Direction with size of:

{Length: length of image + 2\*(index of padding - 1), Width: 1}

1. Zero Padding for Horizontal Direction with size of:

{Length: 1, Width: width of image + 2\*(index of padding)}

**If this is the first padding\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Else**

**End if\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**End Loop: Channel\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**End Loop: Width of Padding\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Return** Zero Padding Image

**End of Algorithm\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Algorithm 2: Convolution\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Input Data:** image, filter

**Create:**

Output Image matrix with size of:

{Length: length of image, Width: width of image, Channel: number of channels from image}

**Looping from** 1 to length of image\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Looping from** 1 to width of image**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Looping from** 1 to channel of image\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Conduct multiplication of:

Image [index of length to index of length + length of filter,

index of width to index of width + length of filter,

index of channel]

with filter

1. Store the multiplication value at the location:

Output Image [index of length,

index of width,

index of channel]

**End Loop: Channel\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**End Loop: Width of Image\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**End Loop: Length of Image\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Return** Output Image matrix

**End of Algorithm\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

To validate the function implemented, the function is compared with the built-in imfilter() function. The validation of the above codes are tested with a sample matrix generated by using the built-in function magic(), and a filter generated with built-in ones() function. To test it, both the algorithm and imfilter() are fed with the same sample matrix and filter, and then the outcomes from each of these will be judged whether they are identical by using built-in isequal() function from MATLAB. If the outcomes are identical, the isequal() function will return the value of 1, meaning it’s TRUE.

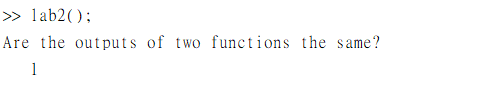


Figure 2: Testing Message from MATLAB Command Window

**Problem 2 Play with Filters**

**Smoothing Filters:**

In the following section, seven different filters are being tested: 3\*3 Box Filter, 3\*3 Weighted Box Filter, Horizontal Filter, Vertical Filter, 3\*3 Circular Shaped Filter, 5\*5 Box Filter, and 5\*5 Circular Shaped Filter. To test them out, the following images are fed to the function to conduct the experiment.



Figure 3: Tested Images

The following are the results generated using the my\_conv function implemented from the previous section:



Figure 4: Outcomes from 3\*3 Box Filter



Figure 5: Outcomes from 3\*3 Weighted Box Filter



Figure 6: Outcomes from 3\*3 Horizontal Filter



Figure 7: Outcomes from 3\*3 Vertical Filter



Figure 8: Outcomes from 3\*3 Circular Shaped Filter



Figure 9: Outcomes from 5\*5 Box Filter



Figure 10: Outcomes from 5\*5 Circular Shaped Filter

In general, the main function of each of these filters is trying to smooth out the content of the image. In other word, by implementing these filters, a less contrast image is expected.

For 3\*3 box filter, the filter elevates the brightness of the overall images and smooth out some contrast from the original image. As for 3\*3 weighted box filter, it contains more contrast than the ordinary 3\*3 box filter, since it emphasizes more on the color of the central pixel. From the outcome of 3\*3 horizontal filter, the contrast in the horizontal direction is more obvious. This can be viewed in the shadow region from the first figure and road region from the second figure. The outcome of 3\*3 vertical filter is the opposite, showing more contrast in the vertical region. Especially in the region of hat from the first figure, and the building region from the second figure. For 3\*3 circular shaped filter, it possesses the most contrast out of all 3\*3 filters overall. As for 5\*5 box filter, it tends smooth out the image even more by making the image more blur. It shows up some improvement in the 5\*5 circular shaped filter. However, compared to 3\*3 filters, 5\*5 circular shaped filter is still more blur.

In term of image visibility, the 3\*3 circular shaped filter performs the best. However, in term of smoothing out the image, the best might be 5\*5 box filter, since it wipes out the most details out of any of these.

Compared box filter, it can be observed that the larger the window size is, the more blur it will be in the outcome, since the filter takes more pixel into account and store it into the outcome matrix. Compared the weighted box filter, it can be seen that the outcome from weighted box filter contains more details and contrast, due to the fact that it put more emphasis on the central pixel instead of averaging all the pixel in once.

**Enhancement Filters:**

In the following section, the same filters with two different sizes are tested with the following images:

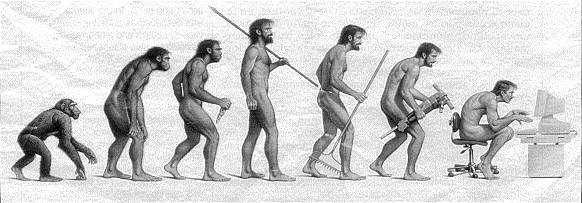


Figure 11: Tested Images for Enhancement Filters

The following shows the outcomes of 4\*4 and 8\*8 Laplacian filters respectively:

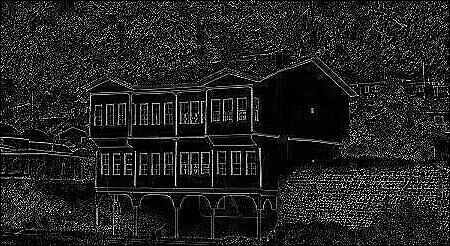
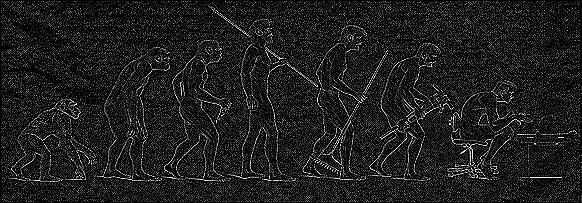


Figure 12: Outcomes for 4\*4 Laplacian Filters

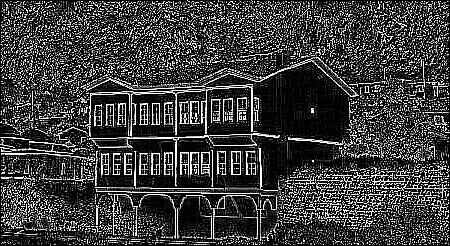
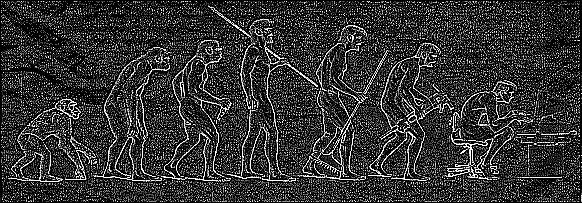


Figure 13: Outcomes for 8\*8 Laplacian Filters

In general, both filters do the job of enhancing the color contrast in the original image. The difference between the two is how detail it pertains for their outcomes in term of their window sizes.

For this part, the 8\*8 Laplacian filter performs better since it preserves more details in the image.

**Edge Filters:**

In this section, four different filters from two different edge detection algorithms are compared. The tested edge detection algorithms are: Prewitt and Sobel algorithm. For testing, the same data image from enhancement filters sections are being used.

The following shows the results from each of the filter:

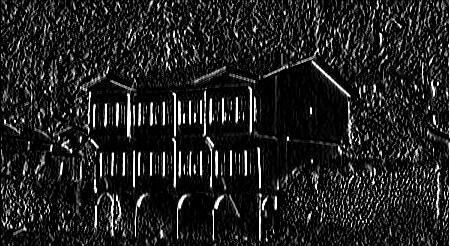
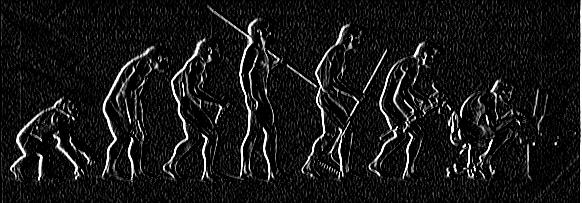


Figure 14: Outcomes for Prewitt X-direction Filter

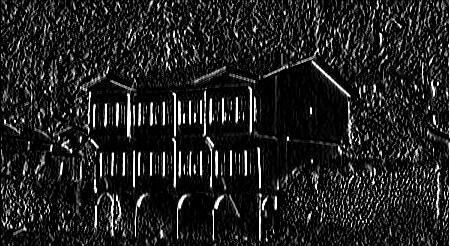
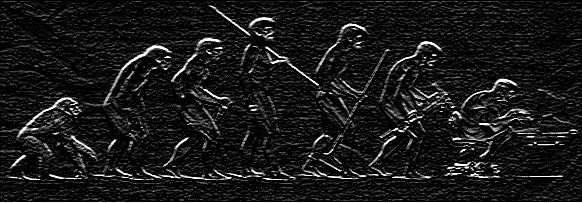


Figure 15: Outcomes for Prewitt Y-direction Filter

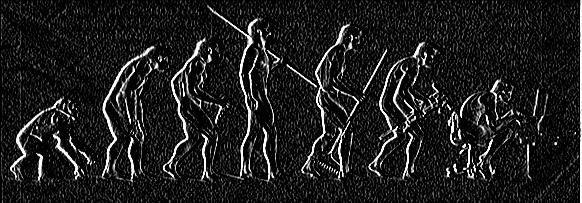


Figure 16: Outcomes for Sobel X-direction Filter

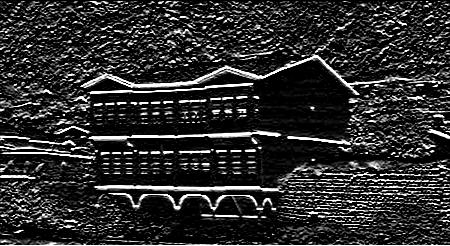


Figure 17: Outcomes for Sobel Y-direction Filter

Out of these outcomes, it can be viewed that Sobel edge detection algorithm performs better from Prewitt, since it contains a less sharp margin in the convolution kernel.

The weight put in the central pixel can enhance the edge portion of the image. Therefore, for this case, Sobel edge detection outperform Prewitt with relatively obvious results.

Compared with enhancement filter, the function of enhancement filter is to enhance the color contrast from the image, and the main purpose of edge detection filter is to mark out the edge region. From the results, it can be observed that the edge detection wipes out some details from the results from enhancement filter.

**Problem 3 Gaussian Smoothing**

In this portion, three different gaussian kernels with three different sizes of windows are tested. For each size, three different standard deviations are inserted to observe the effects coming from different standard deviations. For tested images, the same set of images from smooth filter section of problem 2 are utilized.

The following figures are the outcomes from the correlated values of window sizes and standard deviations:



Figure 18: Outcomes for Window Size 3\*3 and Sigma of 0.5



Figure 19: Outcomes for Window Size 3\*3 and Sigma of 1



Figure 20: Outcomes for Window Size 3\*3 and Sigma of 2



Figure 21: Outcomes for Window Size 5\*5 and Sigma of 0.5



Figure 22: Outcomes for Window Size 5\*5 and Sigma of 1



Figure 23: Outcomes for Window Size 5\*5 and Sigma of 2



Figure 24: Outcomes for Window Size 7\*7 and Sigma of 0.5



Figure 25: Outcomes for Window Size 7\*7 and Sigma of 1



Figure 26: Outcomes for Window Size 7\*7 and Sigma of 2

In term of window size, the larger the window size, the less details it can preserve in the outcomes. Therefore, as the window sizes increase, the scenarios from each image become more blur. As for the effects from sigma (standard deviation), it can also be viewed that the larger the sigma is, the more contrast the image will be. It makes sense since that the larger standard deviation also means the larger color difference.

In contrast to all the filters from the previous section, gaussian filter not only can enhance the contrast of the image, but also can preserve enough details and suppress noise from the environment. From some of the figures from the previous, the existence of noise can be observed from some dots in the images.

**Problem 4 Non-Linear Filter**

In the following section, a tested image is utilized to test out the effect of three different filters: maximum filter, minimum filter, and median filter.

The tested image is shown below, which contains the existence of salt-and-pepper noise:



Figure 27: Tested Image for this Section

As for the tested filters, each of them contains the sizes of 3\*3, 5\*5, and 7\*7 respectively. The following figures are the results generated with the correlated filter types and window sizes:

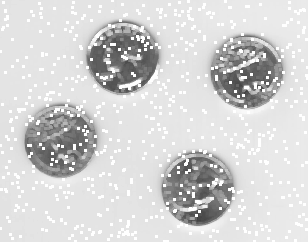


Figure 28: Maximum Filter with Size of 3\*3

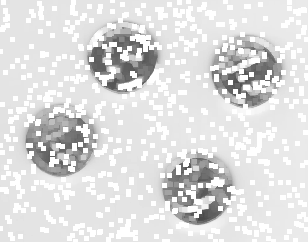


Figure 29: Maximum Filter with Size of 5\*5

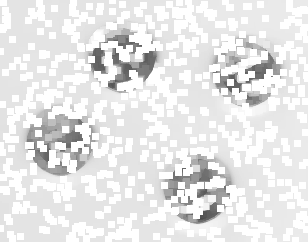


Figure 30: Maximum Filter with Size of 7\*7

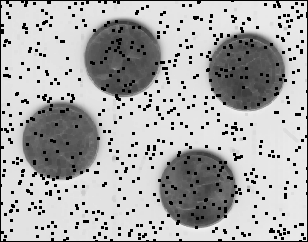


Figure 31: Minimum Filter with Size of 3\*3

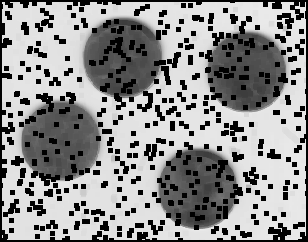


Figure 32: Minimum Filter with Size of 5\*5

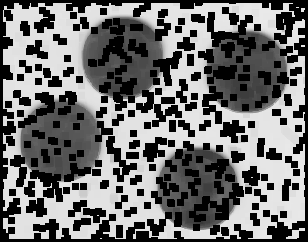


Figure 33: Minimum Filter with Size of 7\*7



Figure 34: Median Filter with Size of 3\*3



Figure 35: Median Filter with Size of 5\*5

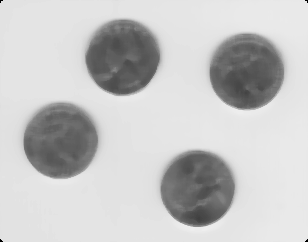


Figure 36: Median Filter with Size of 7\*7

From the above figures, it can be viewed that either maximum filter or minimum filter has the trend of enlarging the salt-and-pepper noise from background. Except for median filter, which suppresses the noise the most. Out of these settings, the median filter with size of 3\*3 performs the best.

**Problem 5 Salt and Pepper vs White Noise**

In the below section, there are two set of testing being set up. The first set contains implementing gaussian filter and box filter onto the image from problem 4 section.

The below figures are the results from the 5\*5 gaussian and 5\*5 box respectively:



Figure 37: Results of 5\*5 Gaussian Filter

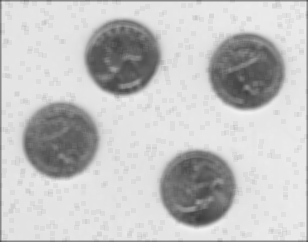


Figure 38: Results of 5\*5 Box Filter

The second section involves in using maximum, minimum, and median filters implemented in the problem 4 section to the traffic image from problem 2 section. The following are the results from correlated filters:



Figure 39: Results of 3\*3 Maximum Filter



Figure 40: Results of 3\*3 Minimum Filter



Figure 41: Results of 3\*3 Median Filter

For the first part of testing salt-and pepper noise, from the visibilities of the images, the gaussian filter performs the better since it suppresses the noises while at the same preserves more details compared to box filter.

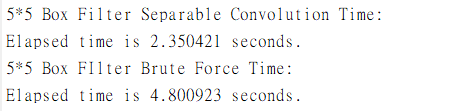
As for the second part of testing white noise, from the quality of the images, the median filter performs the best since the objects from the images have the best visibilities in that outcome.

**Problem 6 Separable Convolution**

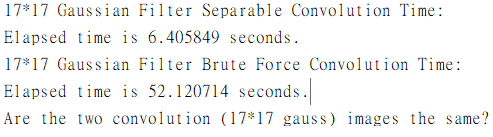
In this section, the separable convolution is discussed and implemented. To start with separable convolution, a separable filter means the filter can be expressed as the outer product of two vector arrays.

For the filters from the problem2, for smoothing filters, all box filters are separable filters and all circular shaped filters are inseparable. For Laplacian filters, all of them are inseparable. As for edge detection filters, all of them are separable.

The following figures show the computational time results from different filter types:







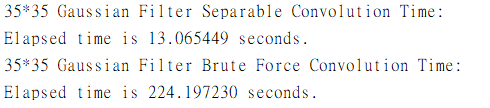


Figure 42: Computational Time for Each Filter

Be noticed that one of the filters: 5\*5 circular shaped filter is not separable. Therefore, the experience can’t be conducted.

**Part II Image Reconstruction**

**Problem 7. Deblurring – Lucy Richardson**

In this section, Lucy-Richardson algorithm is implemented and the following two images are inserted for testing:

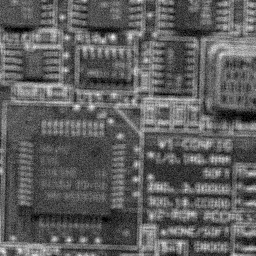


Figure 43: Tested Images for this Section

The implemented filter into Lucy-Richardson algorithm is the median filter implemented in the previous section. To make comparison, the results are compared with the MATLAB built-in deconvlucy() function:



Figure 44: Results of MATLAB deconvlucy() Function with 150 iterations

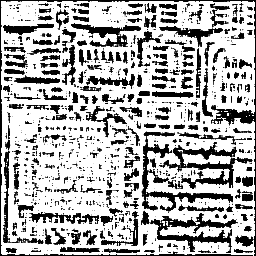
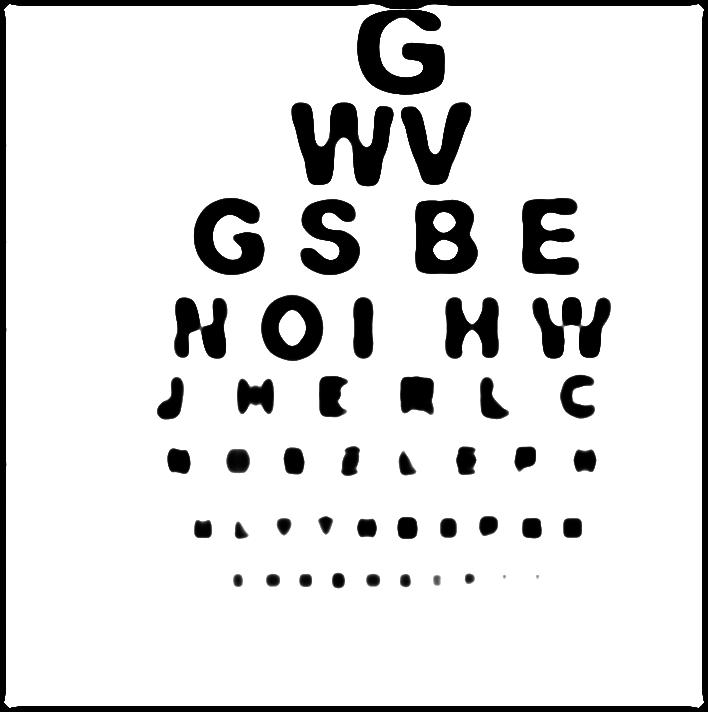


Figure 45: Results of Implemented Lucy-Richardson Algorithm of 150 Iterations

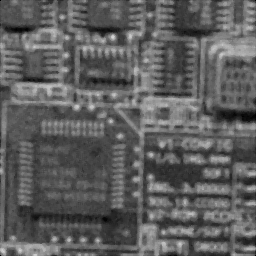


Figure 46: Results of Median Filtering

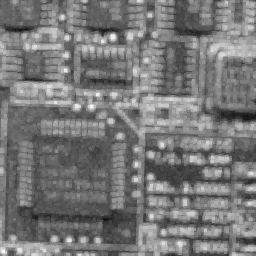


Figure 47: Results of Maximum Filtering

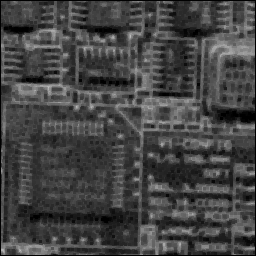


Figure 48: Results of Minimum Filtering

In the results of the implemented algorithm, there are some ringing in the result images. From the original degraded images, the existence of the ring is already there in both of the image. By iterating the algorithm more, the existence of the rings in the image will become more obvious. It might due to the loss of high frequency information from the original image.

One possible solution for this scenario is to implement gaussian filter since the gaussian filter also possesses the function of suppressing the noises from background. The following is the comparison of Lucy-Richardson algorithm and Gaussian Filtering with size of 35\*35 and standard deviation of 0.5.



Figure 49: Results of Lucy Richardson of Iteration of 150 (Left) and Gaussian Filtering



Figure 50: Results of Lucy Richardson of Iteration of 150 (Left) and Gaussian Filtering