# Assignment 1 CS517 Report

In this assignment we implemented various inbuilt functions of MatLab for applying transformations to Images(Grayscale with only Colour Channel).

1. Image Resizing using Nearest Neighbor Interpolation
   Inputs= Name of the input image with extension
   Name of output image without extension
   Size of output Image
   Whether to show the output Image in Subplot or Not
   Returns= RMSE between the Custom Function and matlab's inbuilt implementation of the function.

   Below are my results. I was able to achieve 0 RMSE between Matlab's Implementation and my Implementation of Nearest Neighbour.

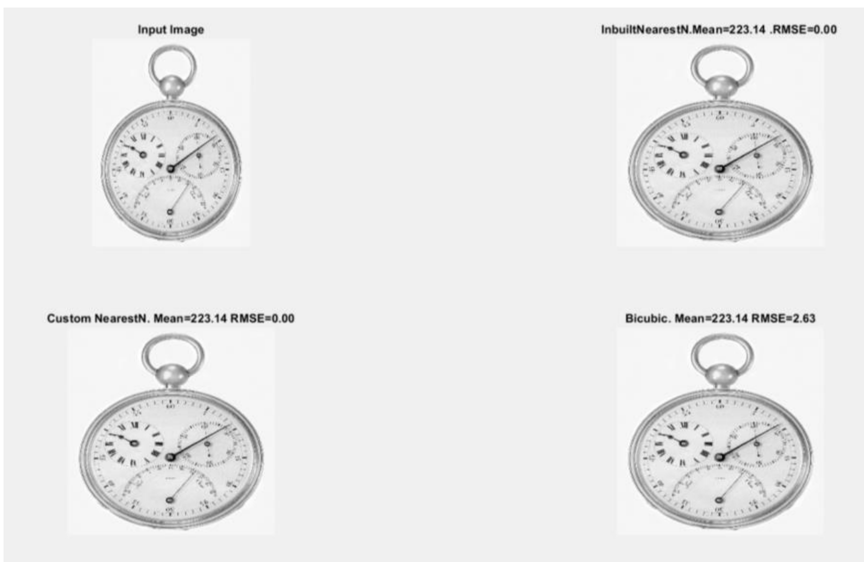


**Figure: Nearest Neighbor resize of watch.tif with RMSE=0**


**Figure: Resized dollars.tif to 5000*5000**

Comparing the output using bicubic in both the cases it could be easily seen that RMSE in second was higher as the Details were in the image were quite more.
RMSE(Bicubic, Nearest)[Watch.tif]=2.6
RMSE(Bicubic,Nearest)[Dollars]=10.85

2. Image Resizing Using Bilinear Interpolation
   Inputs= Name of the input image with extension
   Name of output image without extension
   Size of output Image
   Whether to show the output Image in Subplot or Not

   Returns= RMSE between the Custom Function and matlab's inbuilt implementation of the function.

   Below are my results using using two input images and resizing both of them to 5000*5000 images
   I was able to attain 100% accuracy(i.e 0 RMSE) between Matlab's and My implementation.

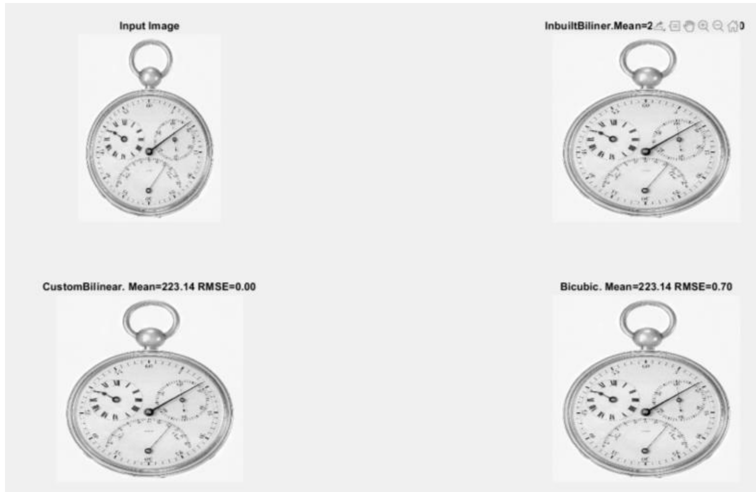


**Figure: Custom Bilinear with Watch.tif**(Note the rmse with bicubic lower than nearest)

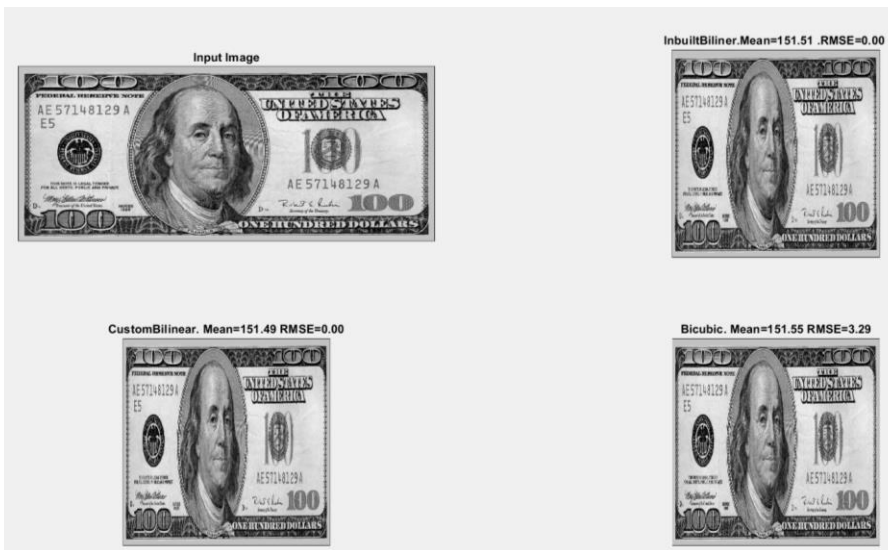


**Figure:Custom Bilinear with dollars.tif**

In this case RMSE with bicubic interpolation was lower in both the cases as compared to Nearest Neighbour. But RMSE in dollars.tif was still higher than that of watch.tif

RMSE(Bicubic, Bilinear)[Watch.tif]=0.70
RMSE(Bicubic,Bilinear)[Dollars]=3.29

3. Image Rotate
   Inputs= Name of input image
   Name of output image
   Angle of Rotation in degrees
   Output= RMSE between custom and inbuilt imrotate using bilinear interpolation


**Figure: Rotation of an image by 45 degrees using bilinear interpolation.**

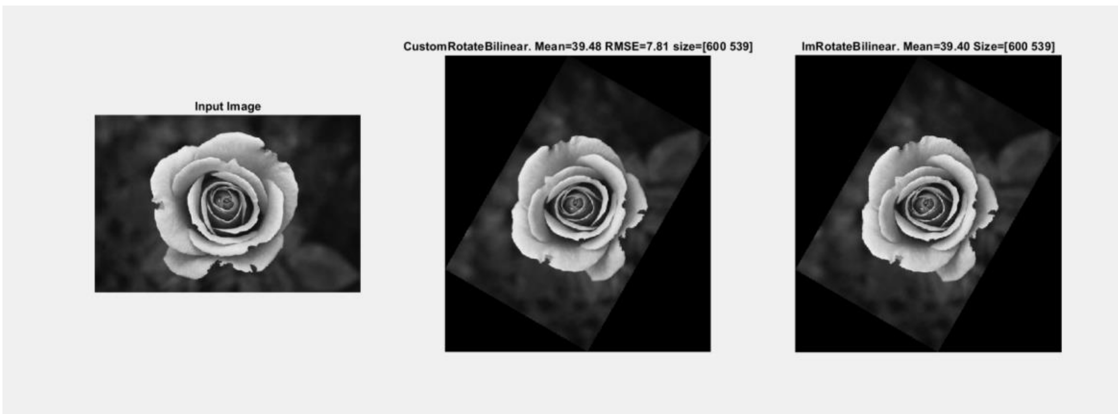


**Figure: Rotation of an image by 60 degrees using bilinear interpolation.**

There were some implementation errors in this the output size was same with both Custom and inbuilt function but due to loss of precision when converting radians to degrees caused minor variations in angle calculations which led to differences in output pixel values and hence an rmse of around 7.5 was observed in this case.

4. Bitplane Slicing
   Inputs=[Name of Input Image, Name of Output image, array of values which needed to be preserved]
   Output= arrays of values containing rmse between input and bitplane sliced images .

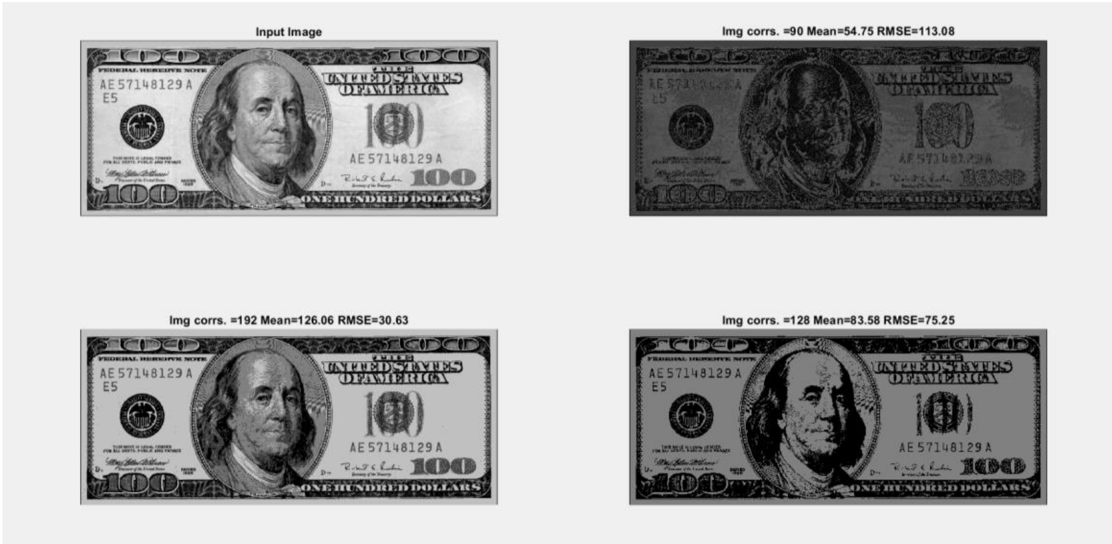


**Figure: Bitplane slicing of Dollars.tif using the values[90,192,128]**

Clearly the RMSE was least in the case of 192 as most of the values were preserved and also the high was a high contrast image.

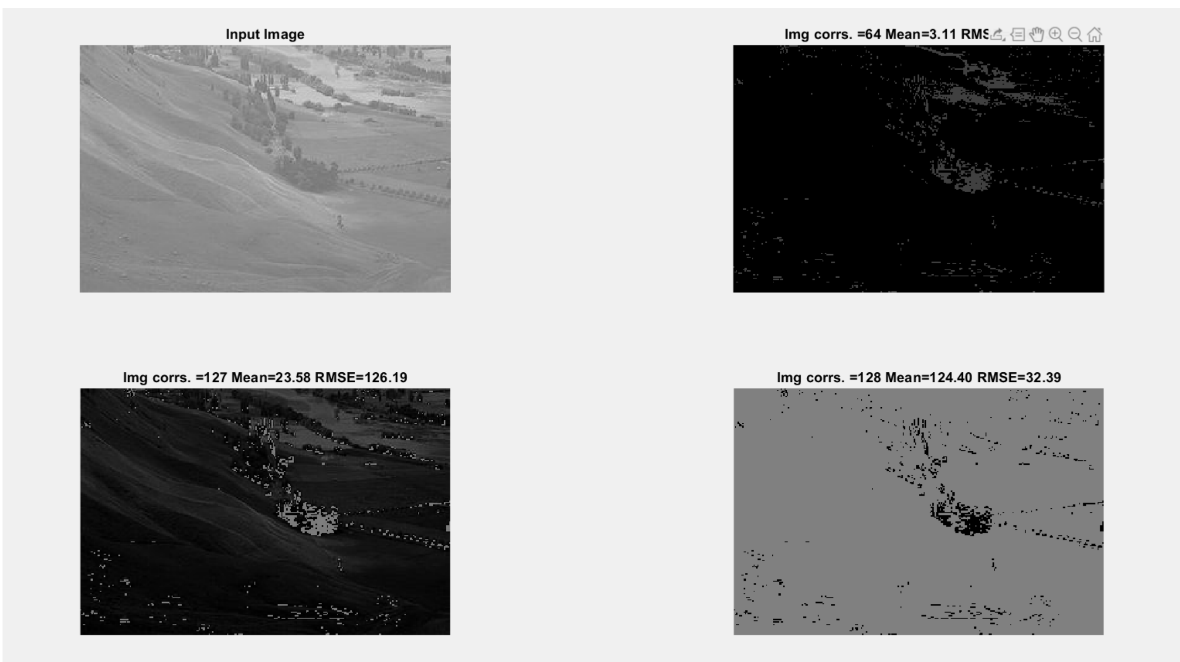


**Figure: Bitplane slicing using middle contrast image**

Clearly 127 was able to capture lot of background information as it has 6 bits on whereas details captured by 128 were low as only one bit is on but striking is the difference values of the two.

5. Image registration given some tie points

Inputs:[Transformed Image,4x4 Matrix Containing the tie points,Actual image name to compute the rmse value,output image name]
Output the rmse between reference and output image.

Tie Points Matrix used by me:
R =

| | | | |
|---|---|---|---|
| 115 | 117 | 192 | 128 |
| 80 | 617 | 487 | 625 |
| 633 | 77 | 684 | 140 |
| 613 | 602 | 1011 | 664 |

I used the images given in the lab task as the reference for this task for benchmarking the algorithm.

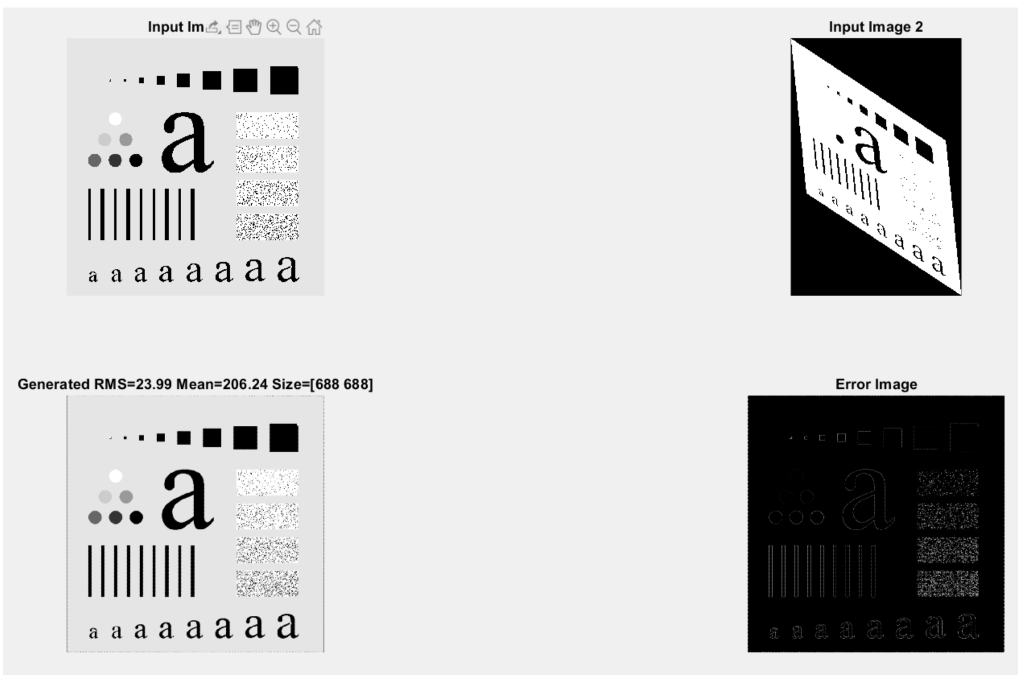


**Figure: Image Registeration**

It can be clearly seen that there are some errors in registering the output image there are some black borders outside the image and large number error is contributed to the black dots in the white square as is evident from the error image Errors also occur around the boundary of the transformed image.

Note: for generating this output is fixed the number of rows(outputm) and columns in the output(outputn). This can be easily modified by uncommenting some code in this functions implementation which will enable automatic calculation of the rows and columns.

6. Histogram equalization
   Inputs=[name of image to be equalized, Name of output image ]
   Output= rmse between matlab implementation and my implementation.

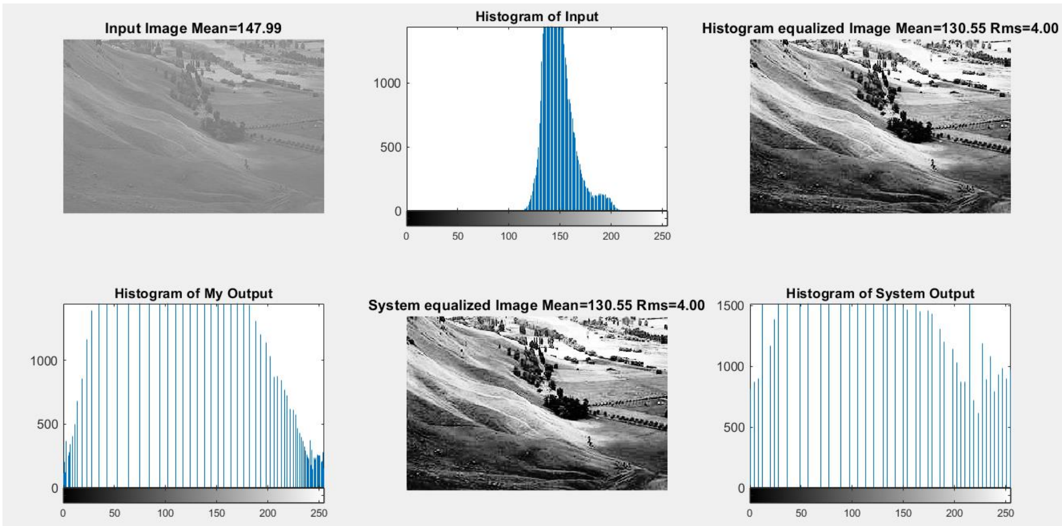


**Fig. Histogram Equalization**

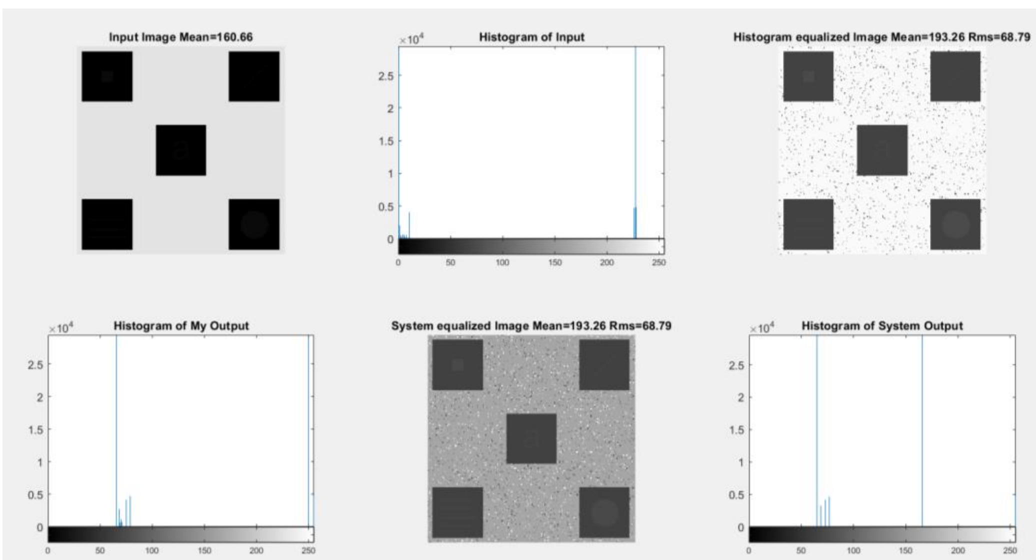


**Fig. Histogram Equalization**

The errors between system implementation and my implementation were due to rounding errors incorporated due to loss of precision but overall the results were good as could be seen with input image 1. The results were skewed in second image due to very less range of pixel values.

7. Histogram matching between two images
   Input=[Name of image to be matched, Reference Image name, output image name]

Output = RMSE between My implementation and Matlab Implementation of histogram Matching

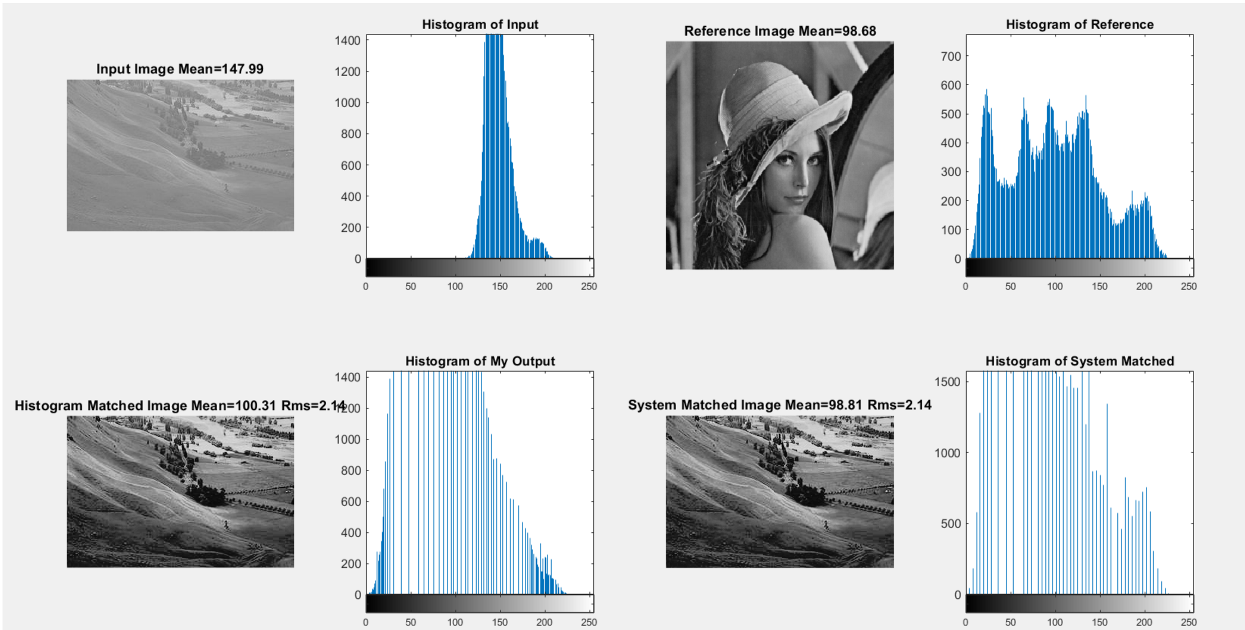


**Fig. Histogram Matching**

In this case also the error were due to rounding due to loss of precision. To avoid losses due to reverse matching as the function is one to many in discrete matching I used a different method to minimize those losses. More details of it could be found on this link.

http://fourier.eng.hmc.edu/e161/lectures/contrast_transform/node3.html

8. Adaptive Histogram Equalization
   Inputs=[name of input image]
   Output = rmse between system implementation and my implementation

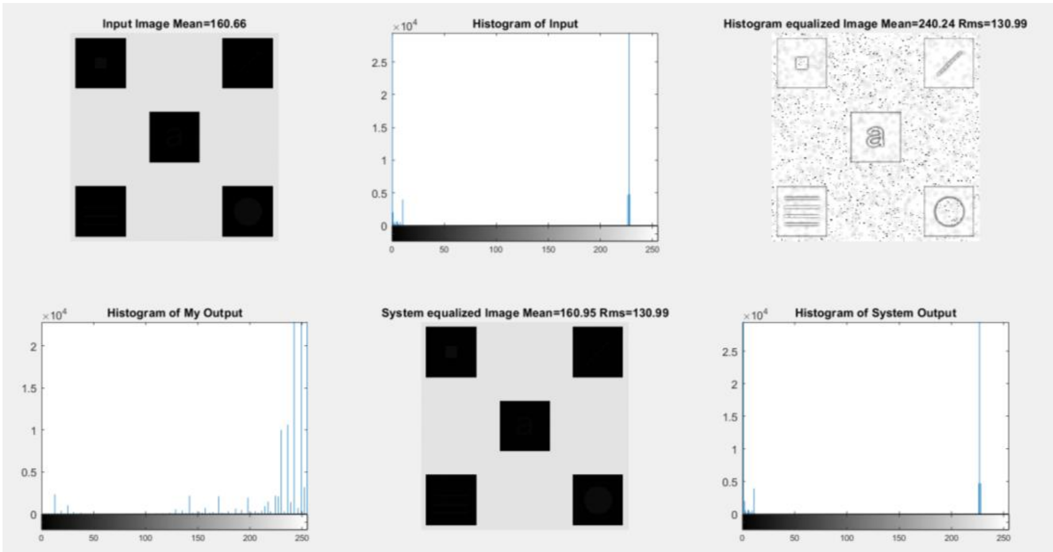


**Fig. Adaptive Histogram Equalization**

I used a windowssize of 8x8 for equalization. It could be easily changed by changing one parameter in the function. Also there was no inbuilt function to implement adaptive Histogram Equalization. The function adapthisteq performed contrast Limited Adaptive Histogram Equalization so the RMSE had no relevance in this task.