Homework 1 Report

// Date: February 4, 20018

// Name: Tamoghna Chattopadhyay

// ID: 8541324935

// email: tchattop@usc.edu

Problem 1.(a)

I. Abstract and Motivation

In this problem we were asked to conduct a series of simple manipulations on grayscale and color images to get familiar with the image data access, processing and output techniques. A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers (e.g. triples in RGB or quadruples in CMYK). Color space identifies a particular combination of the color model and the mapping function.

Each color pixel of an image is described by a triple (R, G, B) for red, green, and blue color intensities. There are many ways to convert a color image to its grayscale image. The **lightness** method averages the most prominent and least prominent colors: (max(R, G, B) + min(R, G, B)) / 2. The **average** method simply averages the values: (R + G + B) / 3. The **luminosity** method is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. The formula for luminosity is 0.21 R + 0.72 G + 0.07 B.

The Cyan-Magenta-Yellow-(Black) (CMY(K)) color space is frequently used in image printing. It is defined by:

$$C = 1-R, M = 1-G, Y = 1-B$$

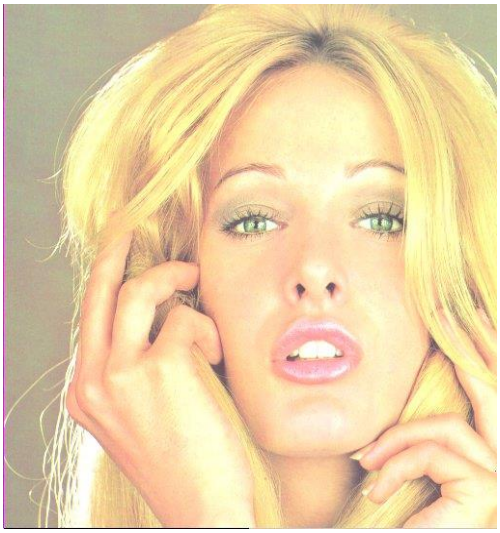
II. Approach and Procedures

I decided to use the C++ language to write the code for the problem statement. Each image can be seen as a 3 dimensional array where the first two dimensions give the location of the pixel of the image and the third dimension gives the value of the intensity in each channel. So for a 256*256 24 bit RGB image, we will have to use an array of [256]*[256]*[3] dimensions. Then the given formulaes are applied to get the corresponding outputs.

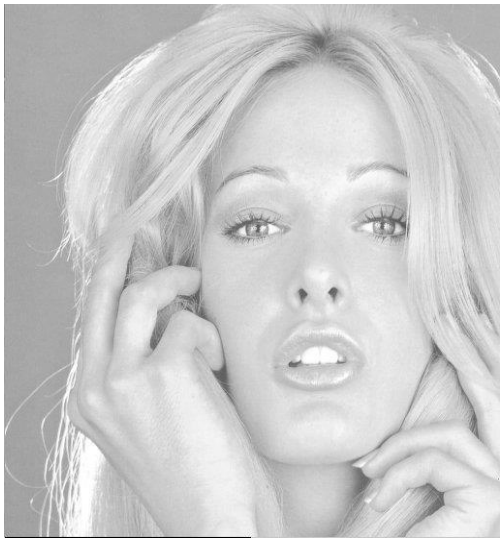
III. Result and Discussion

The luminosity method works best overall for the first problem as cone density in human eye is not uniform across colors and humans perceive green as the strongest color. Hence due to it's weighted method, luminosity gives the best result amongst all three.

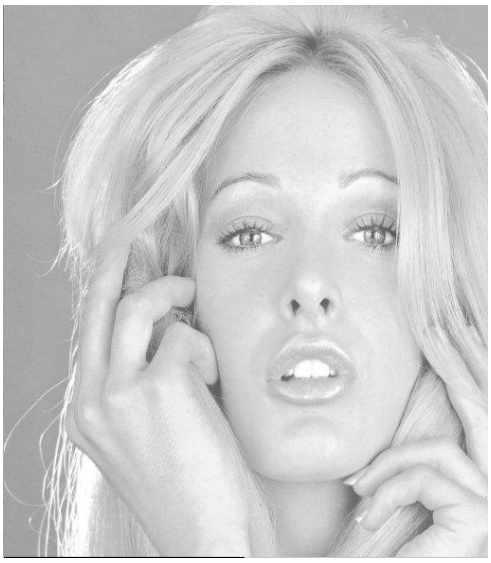
For the second problem, I got three output grayscale images corresponding to cyan, magenta and yellow channels. Each grayscale image is shown in the report below.



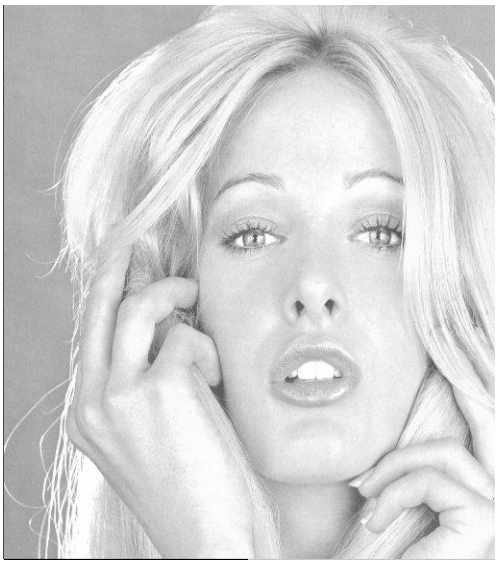
Original Image



Output by lightness method



Output by Average method



Output by Luminosity Method

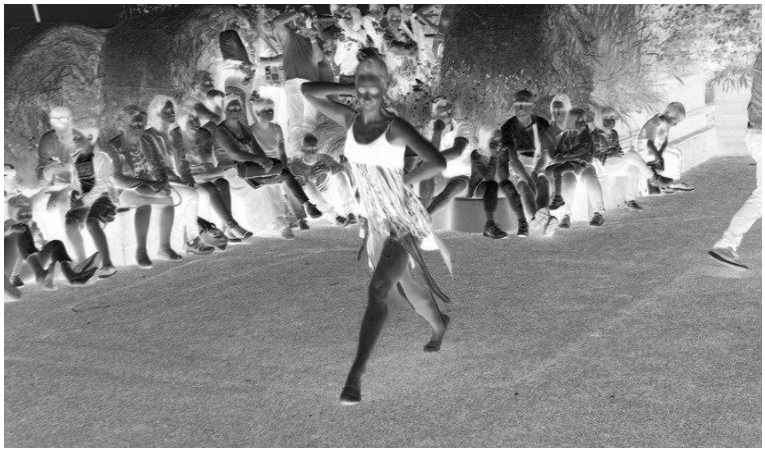
Original Image

Cyan Grayscale Image

Magenta Grayscale Image

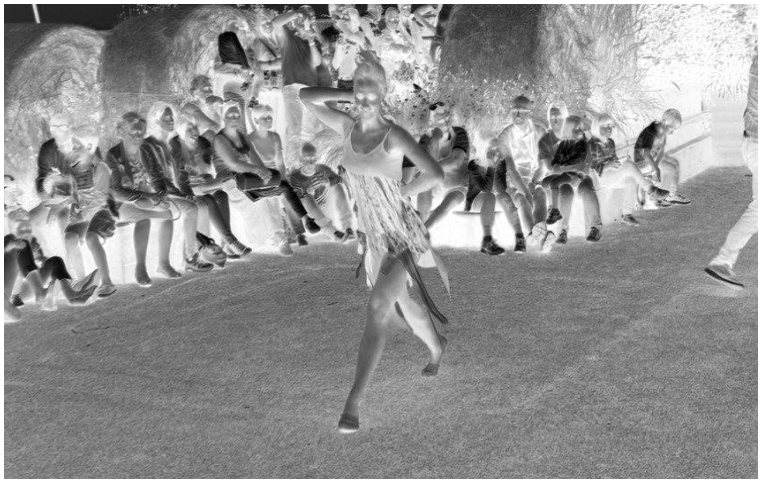
Yellow Grayscale Image

Original Image



Cyan Grayscale Image

Magenta Grayscale Image



Yellow Grayscale Image

Problem 1.(b)

I. Abstract and Motivation

In Bilinear Interpolation, we perform two linear interpolation operations to get the output. The operations are performed in two different directions. In Image processing, it is one of the basic resampling techniques.

When an image needs to be resized, each pixel of the original image needs to be moved in a certain direction based on the scaling factor. When increasing the size, there are pixels that should be assigned appropriate RGB or grayscale values so that the output image is smooth. Bilinear interpolation is used in such cases to assign appropriate intensity values to pixels using the values of the four nearest diagonal pixels.

Resampling Through Bilinear Interpolation

Let $\mathbf{I}$ be an $R \times C$ image.
We want to resize $\mathbf{I}$ to $R' \times C'$.
Call the new image $\mathbf{J}$.

Let $s_R = R / R'$ and $s_C = C / C'$.

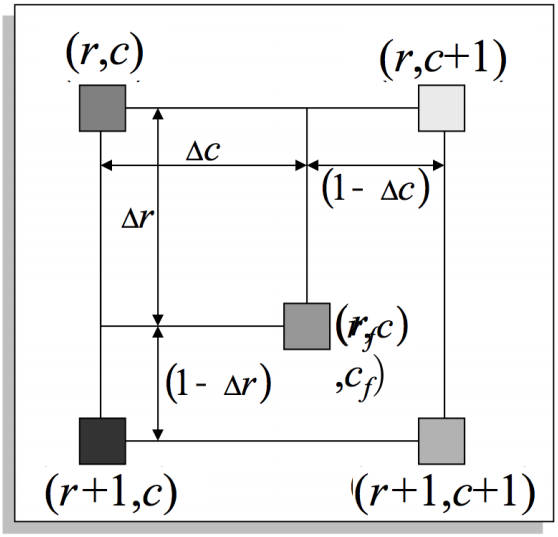
Let $r_f = r' \cdot s_R$ for $r' = 1, \ldots, R'$

and $c_f = c' \cdot s_C$ for $c' = 1, \ldots, C'$.

Let $r = \lfloor r_f \rfloor$ and $c = \lfloor c_f \rfloor$.

Let $\Delta r = r_f - r$ and $\Delta c = c_f - c$.

$$\begin{aligned}\text{Then } \mathbf{J}(r',c') = {} & \mathbf{I}(r,c)\cdot(1-\Delta r)\cdot(1-\Delta c) \\ & + \mathbf{I}(r+1,c)\cdot\Delta r\cdot(1-\Delta c) \\ & + \mathbf{I}(r,c+1)\cdot(1-\Delta r)\cdot\Delta c \\ & + \mathbf{I}(r+1,c+1)\cdot\Delta r\cdot\Delta c.\end{aligned}$$



The above formula is applied for bilinear interpolation and is used in the problem statement to resize the image.

II. Approach and Procedures

I decided to use the C++ language to write the code for the problem statement. Each image can be seen as a 3 dimensional array where the first two dimensions give the location of the pixel of the image and the third dimension gives the value of the intensity in each channel. So for a 256*256 24 bit RGB image, we will have to use an array of [256]*[256]*[3] dimensions. Then the given formulaes are applied to get the corresponding outputs.

Also the ratios are given by the ratio of the size of the original image and the size of the new image. This factor is involved in the formula to give the new image of size 650*650.

III. Result and Discussion

The result showed the bigger output image of size 650*650. The difference of using bilinear interpolation rather than nearest neighbor method is that in this method there is smooth transitions from one pixel to the next in intensity as the intensity of new pixels is combination of nearby pixels whereas in the nearest neighbor method we would have gotten blocky image where pixel intensities were to be repeated. Hence this method gives a better output in this case.

Original Image of size 512*512

New Image of size 650*650

Problem 2.(a)

I. Abstract and Motivation

To make objects distinguishable from each other and the background, contrast plays an important factor. There are two methods to improve the contrast and they are both histogram based:

Method A: the transfer-function-based histogram equalization method,
Method B: the cumulative-probability-based histogram equalization method

II. Approach and Procedures

I decided to use the C++ language to write the code for the problem statement. In the Cumulative Histogram Equalization method, I first found out the histogram of the image and then the cumulative histogram and probability by dividing each value of histogram by the total number of pixels in the image. And then I assigned the new image the pixels according to the new cumulative histogram function creating more contrast.

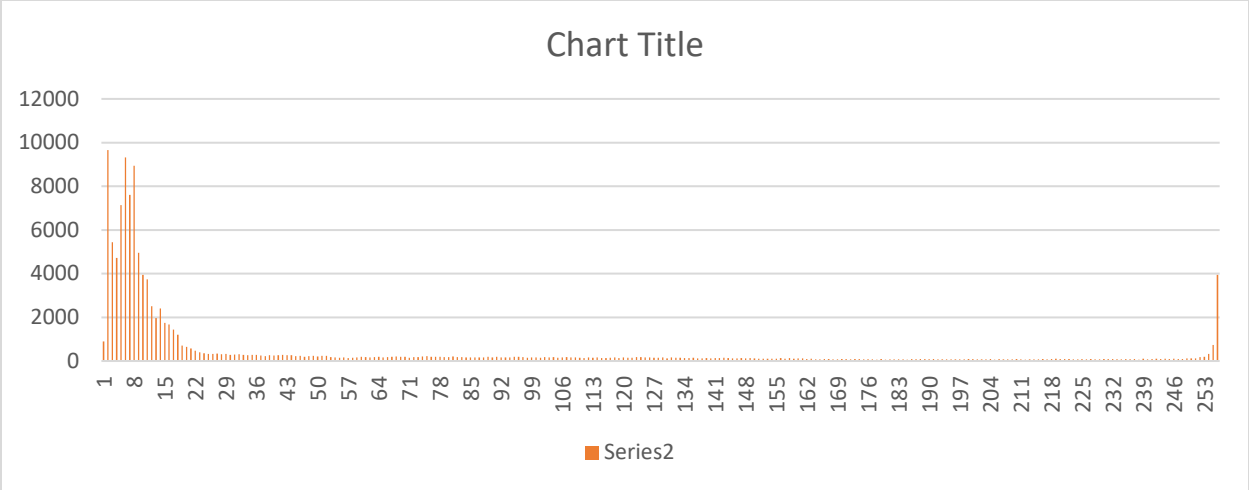
In the second method, for the Bucket Filling Method, I first created a histogram for the image. Then I stored the location of all the pixels which are of same intensity. Then I divided pixels equally in 8 bins based on ascending order and location which I had stored. This assignment is then stored for the new image with more contrast.

III. Result and Discussion

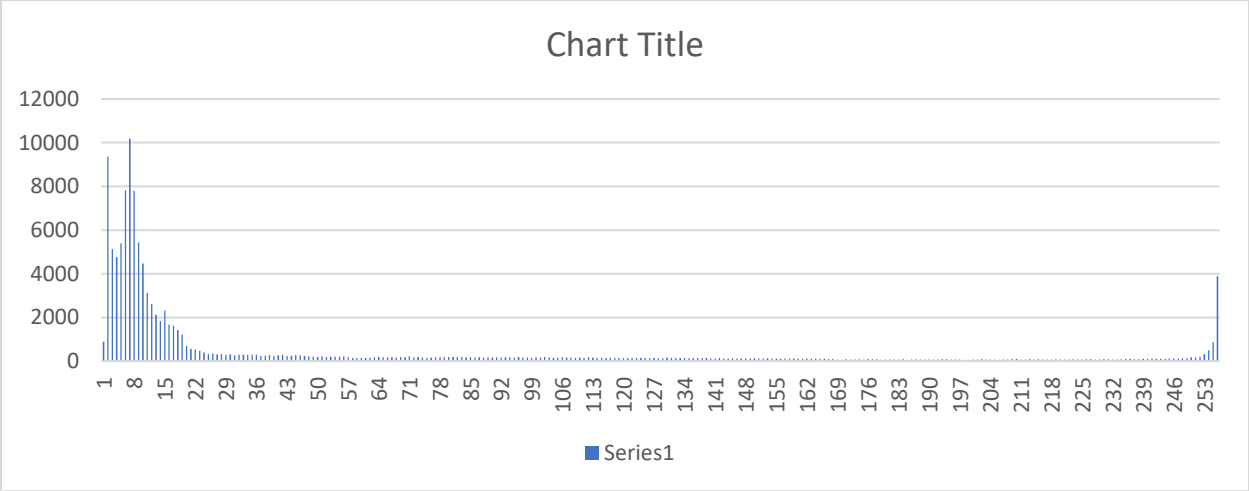
Original Image

Histogram Equalization using Cumulative Probability Method

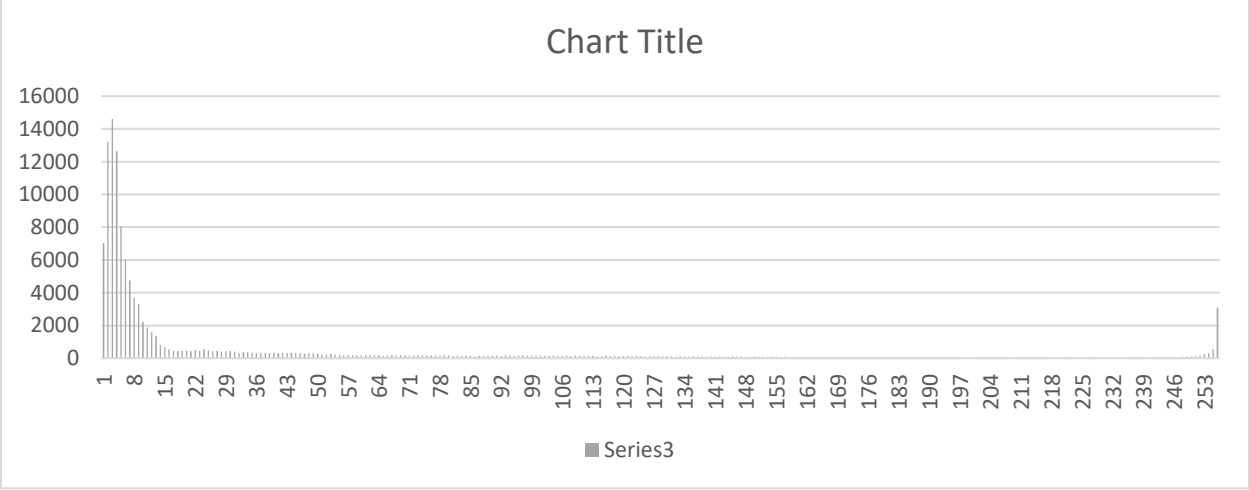
Histogram Equalization using Bucket Filling Method



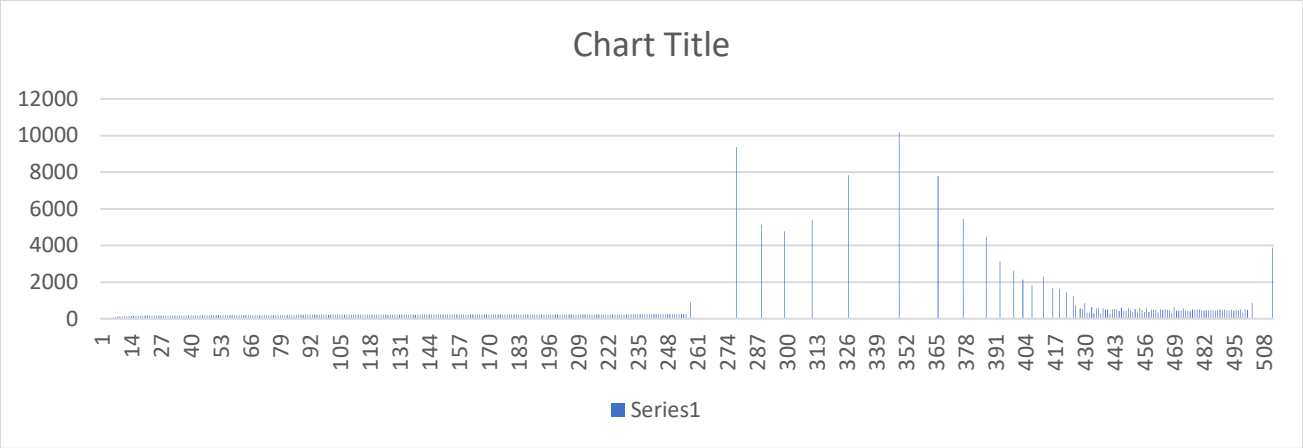
Original Image Histogram for Blue Channel



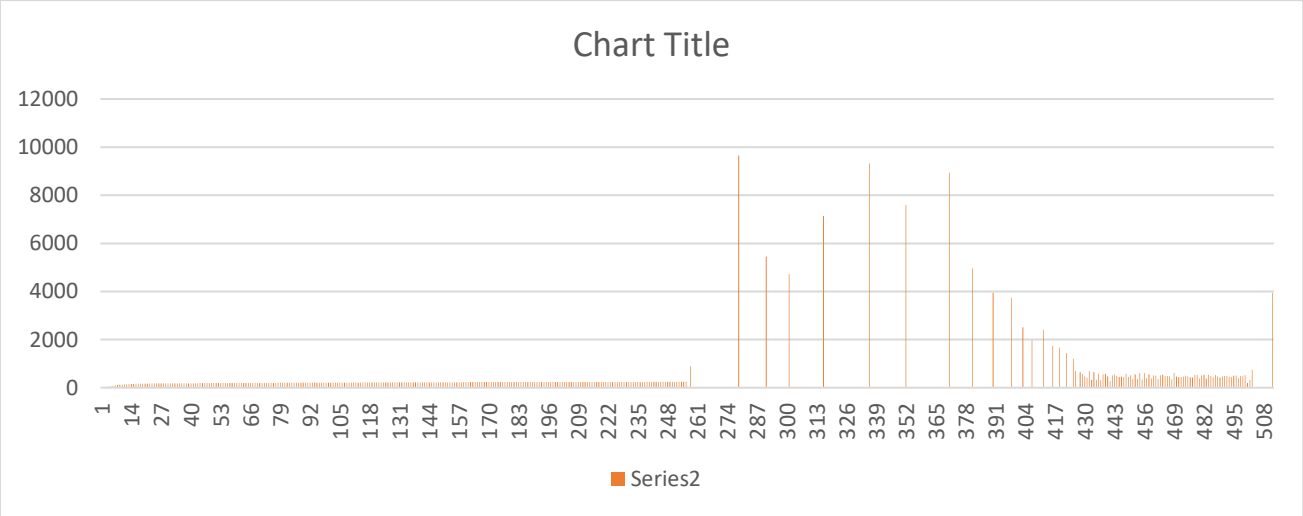
Original Image Histogram for Red Channel



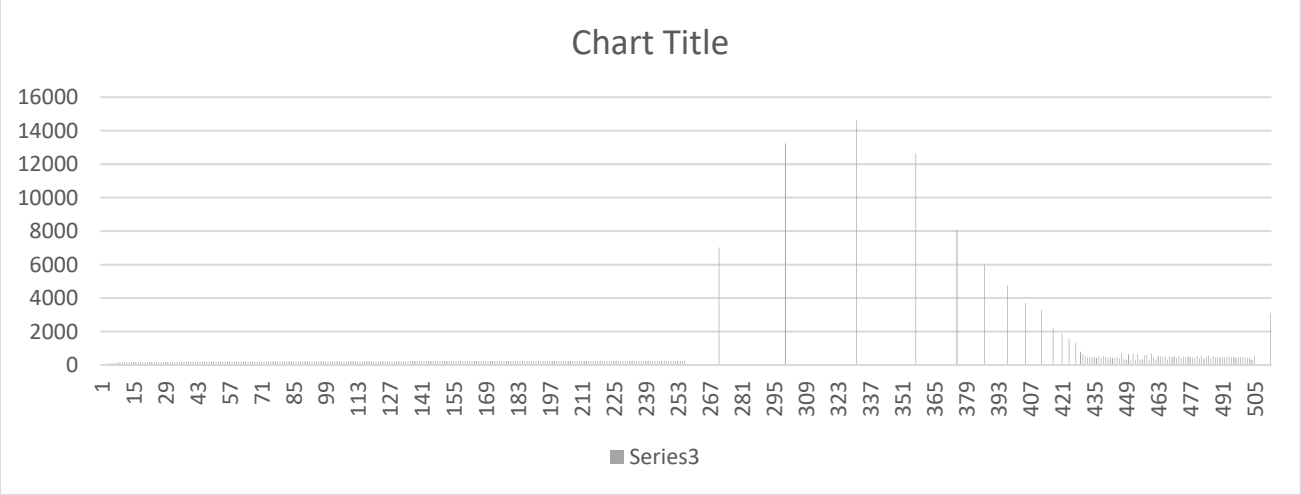
Original Image Histogram for Green Channel



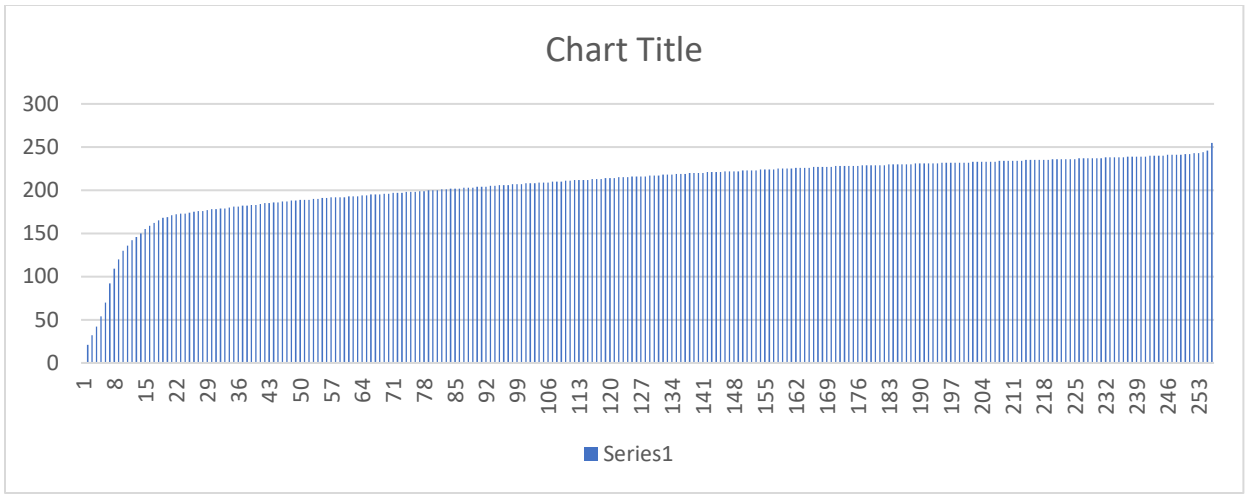
New Histogram for Red Channel for Cumulative Probability Method



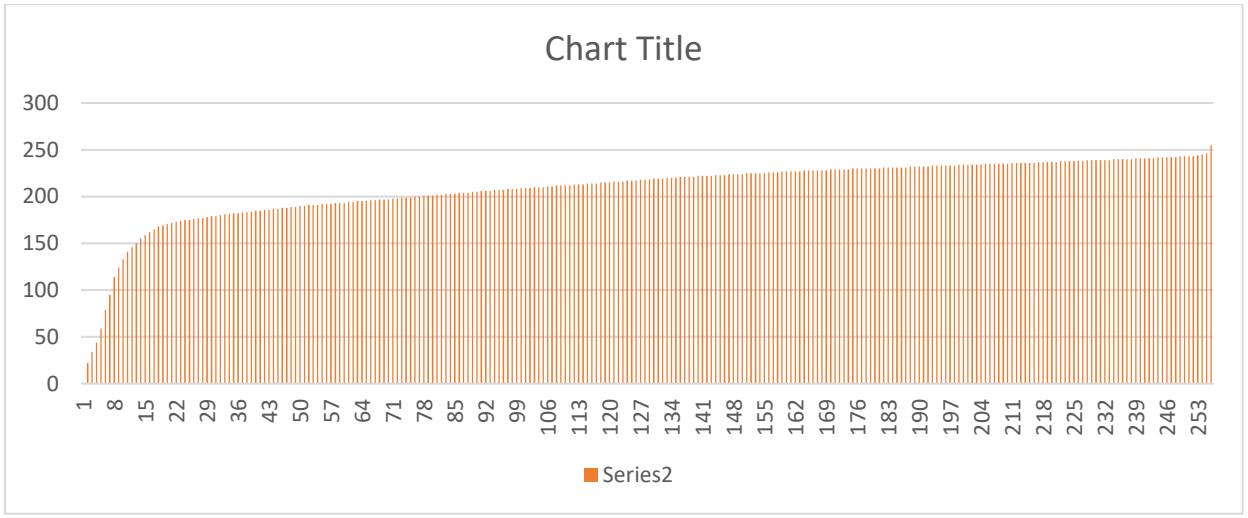
New Histogram for Green Channel for Cumulative Probability Method



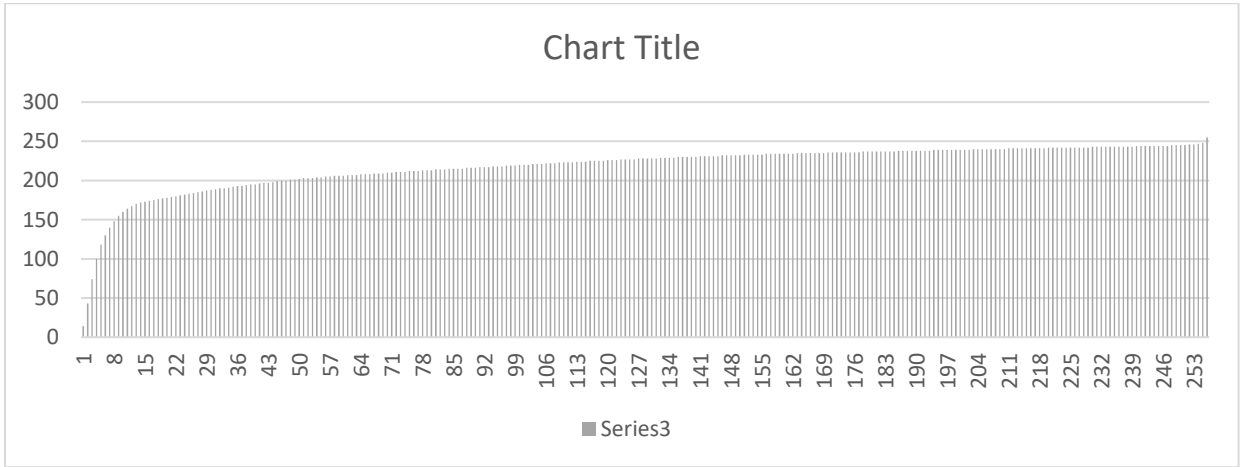
New Histogram for Blue Channel for Cumulative Probability Method



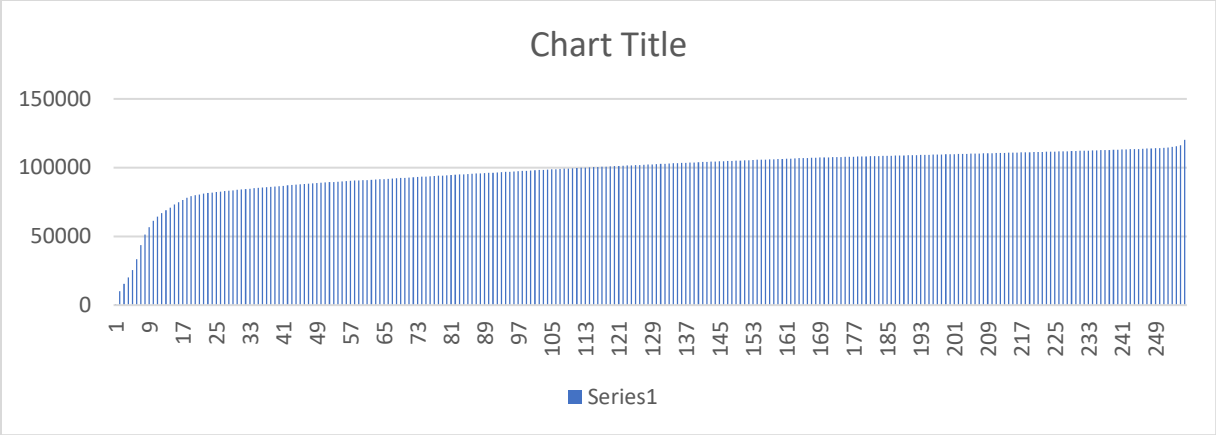
Histogram for Red Channel in Cumulative Probability Method



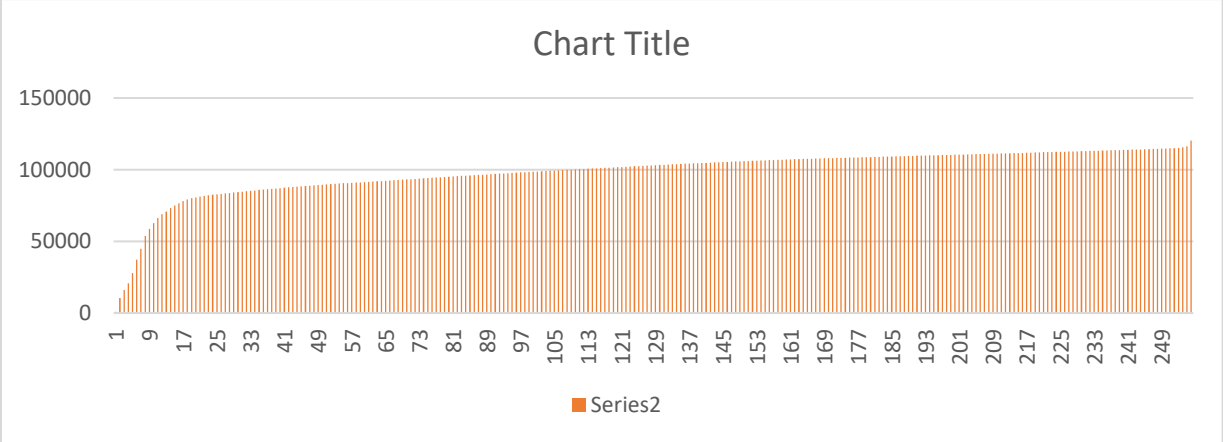
Histogram for Blue Channel in Cumulative Probability Method



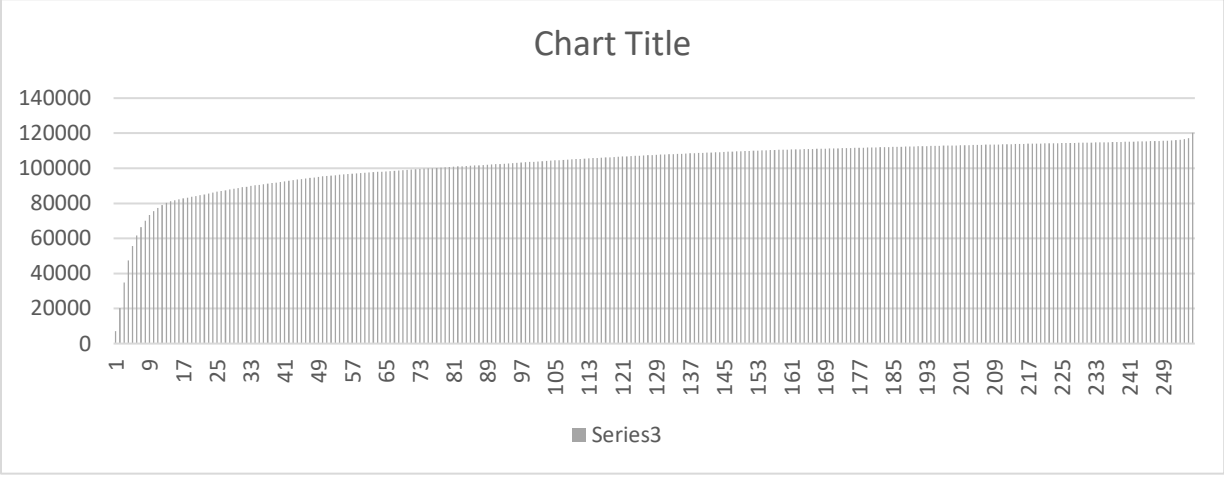
Histogram for Green Channel in Cumulative Probability Method



Histogram for Red Channel in Bucket Filling Method



Histogram for Green Channel in Bucket Filling Method



Histogram for Blue Channel in Bucket Filling Method

The transfer function considers the dynamic range of pixel intensities. The equation of the linear transfer function that maps the input dynamic range to the desired contrast stretched image is given by:

$$G(f) = a + ((b-a)/(c-d))*(f-d)$$

Where:

G(f) = Linear Transfer Function

f = Low Contrast Image pixel intensity

d = Minimum value of intensity for input image

c = Maximum value of intensity for input image

b = Maximum value of intensity for output image

a = Minimum value of intensity for output image

The contrast got in the transfer function method is better than the cumulative probability method. This is because in the transfer function method, the high value in original histogram are expanded and the low values are compressed. Whereas, in the Cumulative Probability method, it just divides the pixels into equal sized bins, regardless of the intensity values. Hence the Method of transfer function is better for histogram equalization.

When we apply both Methods A and B to the image, it results in singular values on the RGB channel. Maybe better results can be obtained when the methods are applied to the YUV color space after convertion from RGB to YUV as YUV also takes human perception into account.

Problem 2.(b)

I. Abstract and Motivation

The oil painting effect is an effect which when applied to an image makes it look like an oil painting. There are two important parameters to keep in mind while doing the oil painting effect : the intensity levels allowed and the size of the window, which defines how many pixels to look at while applying the effect. Oil Paintings create a blocky effect due to reduction in the number of color intensities to be used.

II. Approach and Procedures

I decided to use the C++ language to write the code for the problem statement. First, I checked the intensity of each pixel and found the total number of pixels for each intensity level and stored their location in an array in ascending order. Using these locations, I divided them into bins of equal size and found out the mean intensity for each channel in each of the four bins. We had to quantize to 64 colors, hence 4 bins were utilized. These mean color values were then used to make a quantized image.

Then based on the window size, equal number of rows and columns were added to make a temporary image which had the first few rows and columns mirrored. For example if window size was 5, then 2 rows and 2 columns were added along each boundary of the image. Using the indexes stored in the Color array which showed the mean colors, the color which occurred highest number of times in a

window was selected and assigned to the middle pixel. The window hence kept moving over the whole image and gave us the oil painting effect due to reduction in the number of intensity levels availaible.

For c part where 512 color intensities were required, I increased the number of bins to 8 and then did the same method once again.

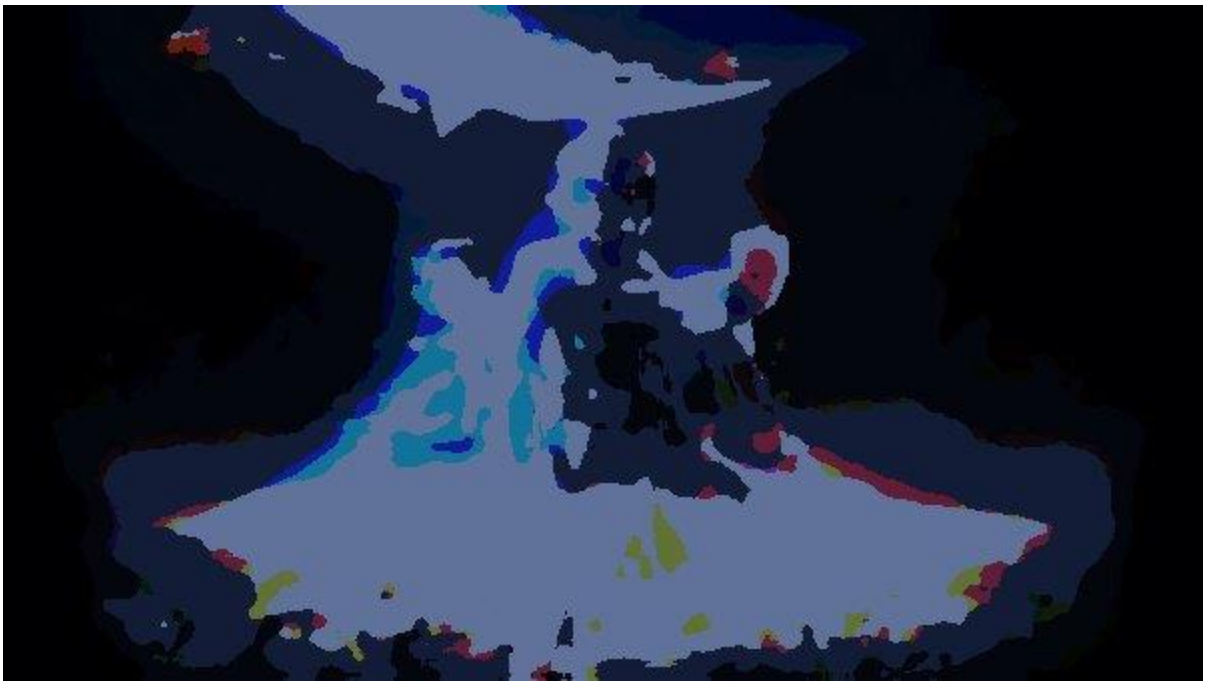
III. Result and Discussion

Original Image

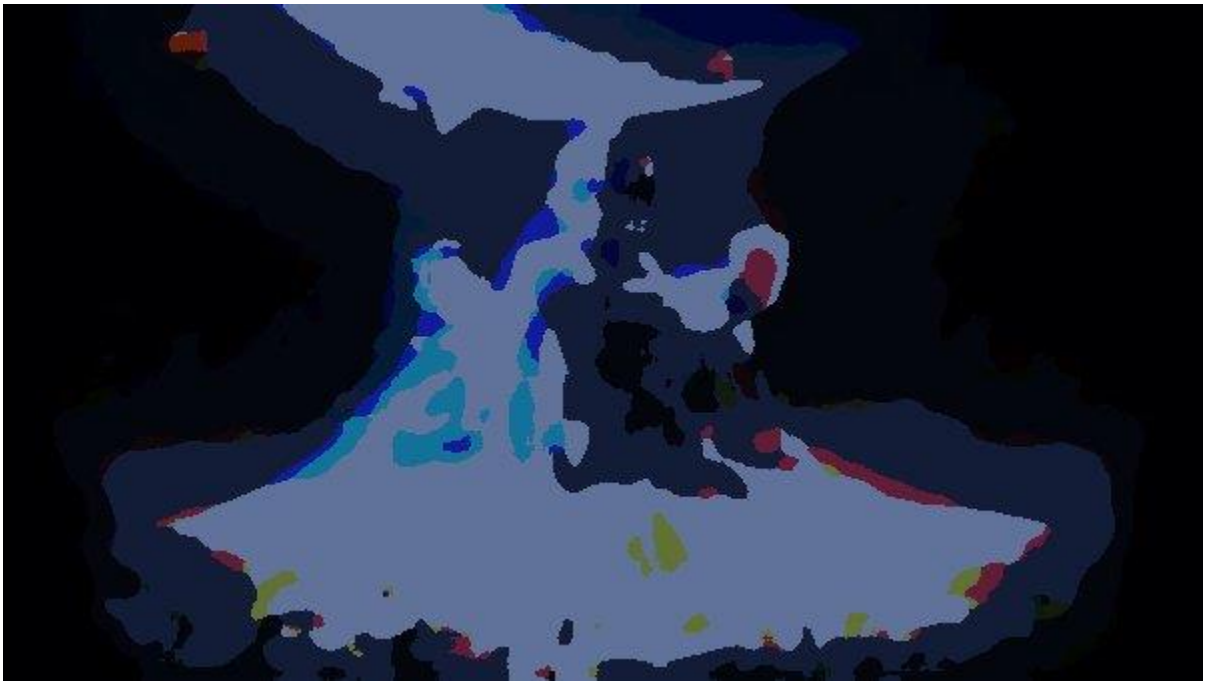
Quantized Image showing image with 64 color intensities

Oil Painting Effect with window size 3

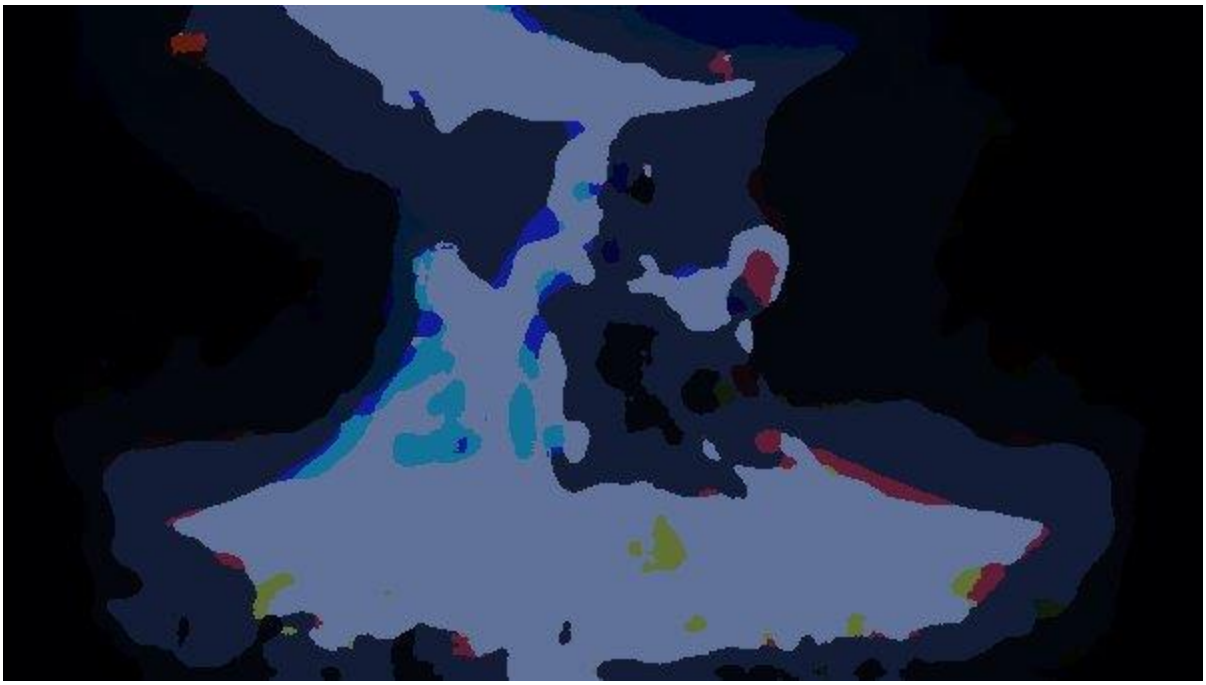
Oil Painting Effect with window size 5



Oil Painting Effect with window size 7



Oil Painting Effect with window size 9



Oil Painting Effect with window size 11

As window size becomes bigger, the oil painting effect becomes more prominent and the details in the image become less, because a larger window size results in larger areas with the same representative color. Some details and edges of the objects in the image disappear.

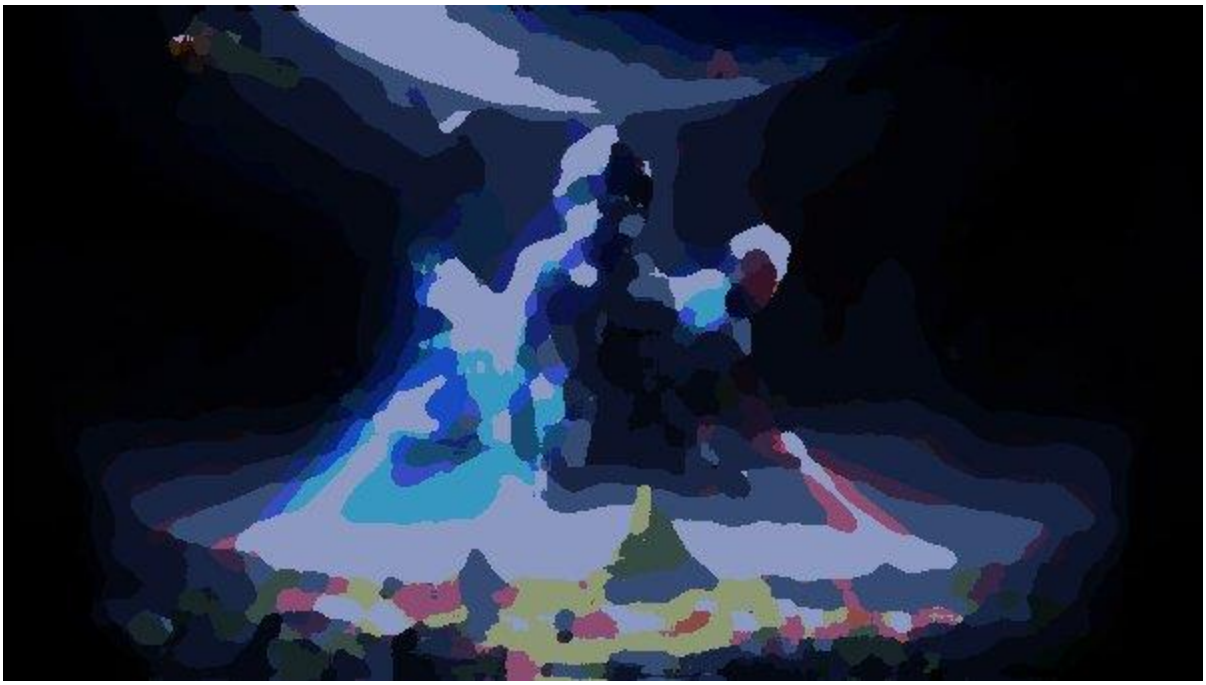
Quantized Image showing image with 512 color intensities

Oil Painting Effect for 512 color quantized image with window size 3

Oil Painting Effect for 512 color quantized image with window size 5

Oil Painting Effect for 512 color quantized image with window size 5

Oil Painting Effect for 512 color quantized image with window size 7



Oil Painting Effect for 512 color quantized image with window size 9

Comparing the oil painting effect on 512-color quantized version and 64-color quantized version of the same image, we see that the former one is more colorful and has more details. This is because it has a more number of color intensities to assign to the pixel. Due to it.s bigger color palette, the images are more detailed in the 512 quantized image.

Another thing I Noticed was that finding the most frequent color for each channel individually gives a different and wrong image than the method used here, i.e., finding the most frequent color as a combination of RGB values.

Problem 2.(c)

I. Abstract and Motivation

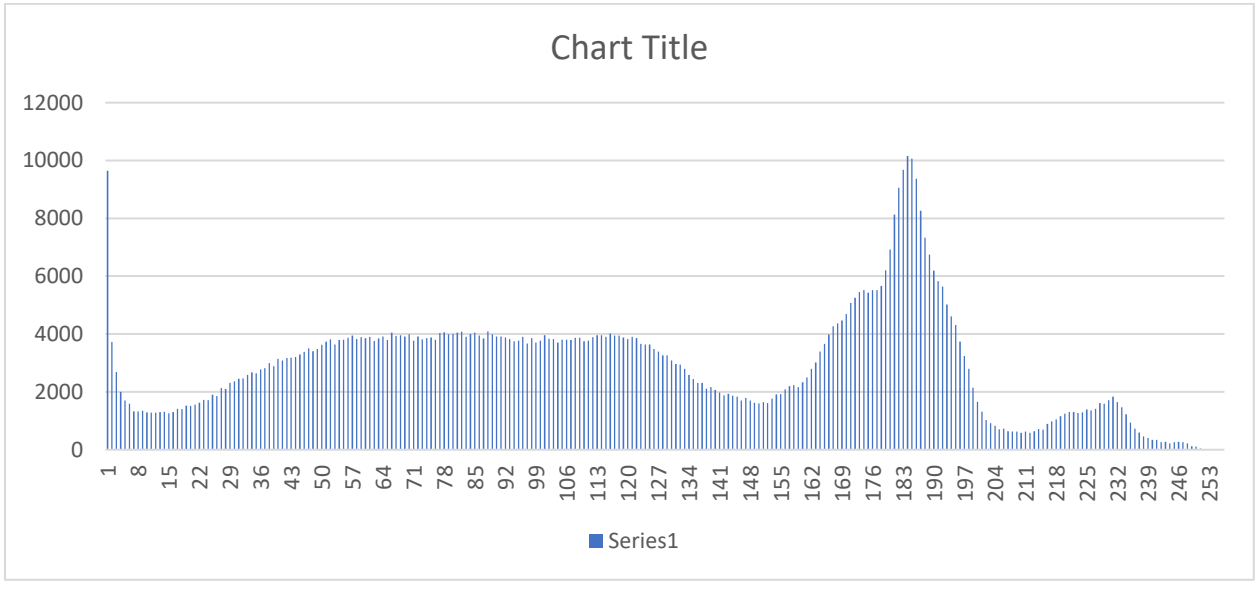
We have to apply an algorithm to bring about the film special effect as shown in the question. It basically deals with histogram matching.

II. Approach and Procedures

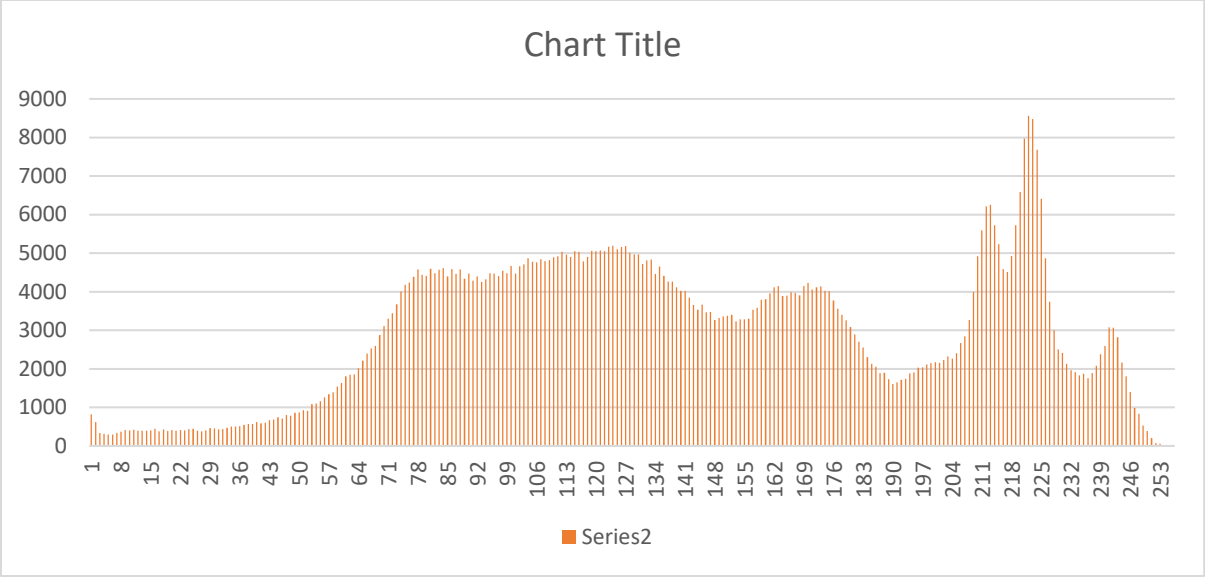
I decided to use the C++ language to write the code for the problem statement. First of all the histogram of the given images were checked and the transformation was noted. Then for the Given image, first it's mirror image was taken. Then it was converted to CMY plane. And then histogram matching was done using the coordinates got from the histogram of example image.

III. Results and Discussion

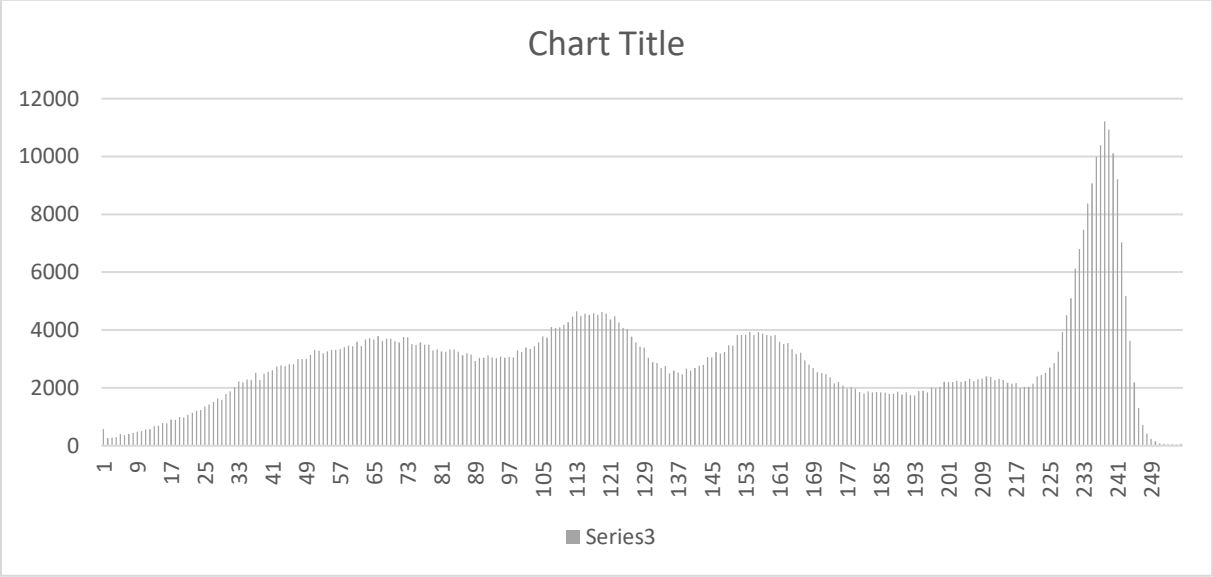
Original Image



Histogram for Red channel

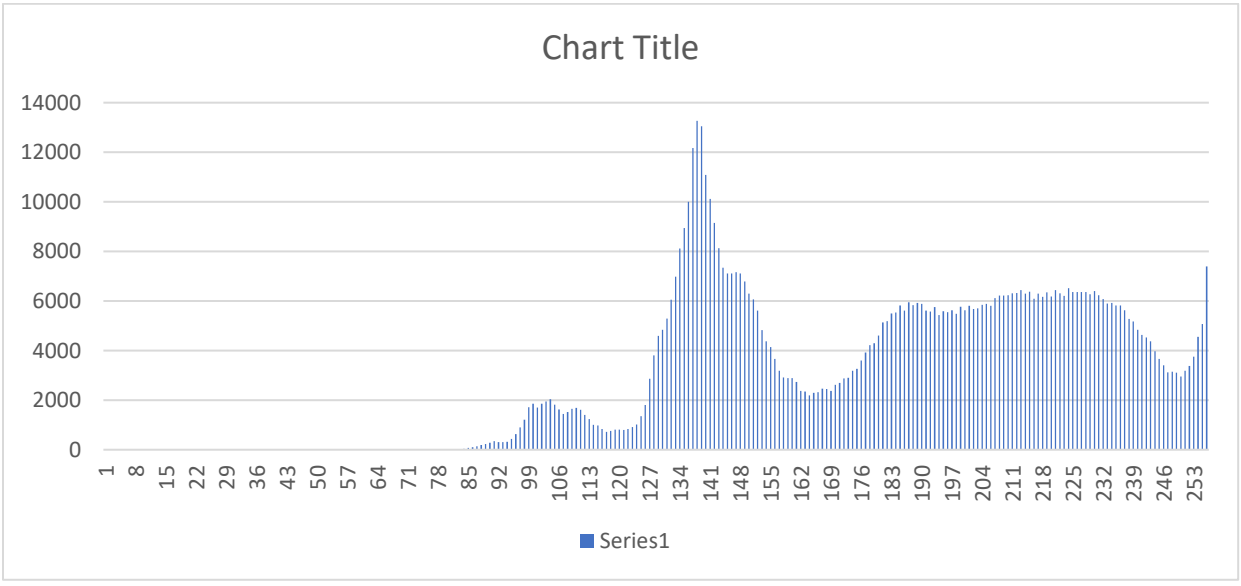


Histogram for Green channel

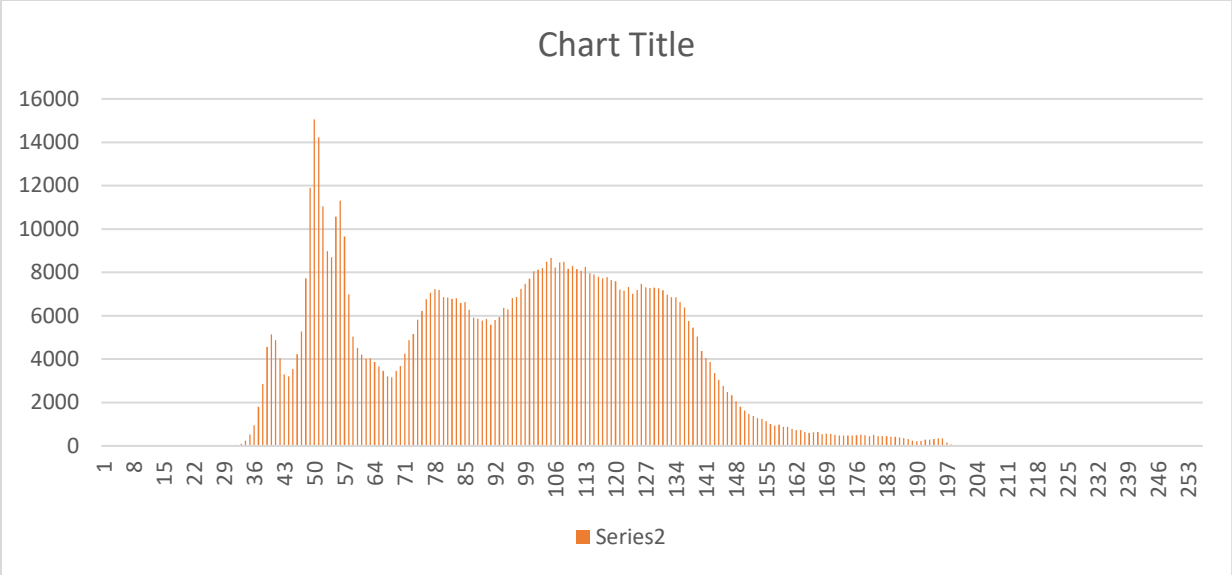


Histogram for Blue Channel

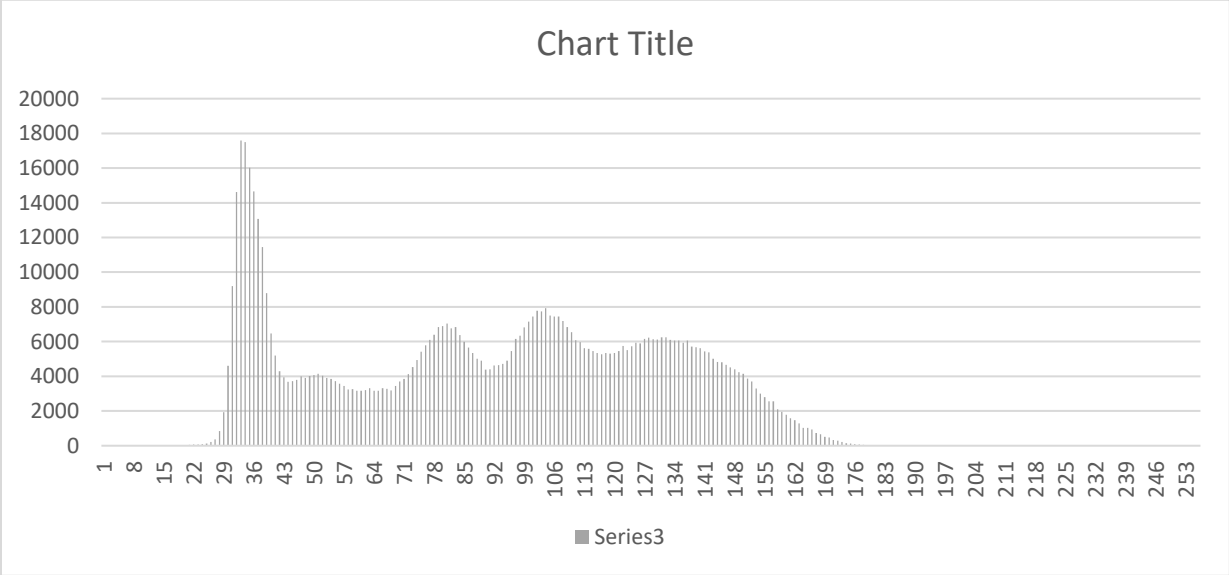
Film Special Effect Image



Histogram for Red Channel

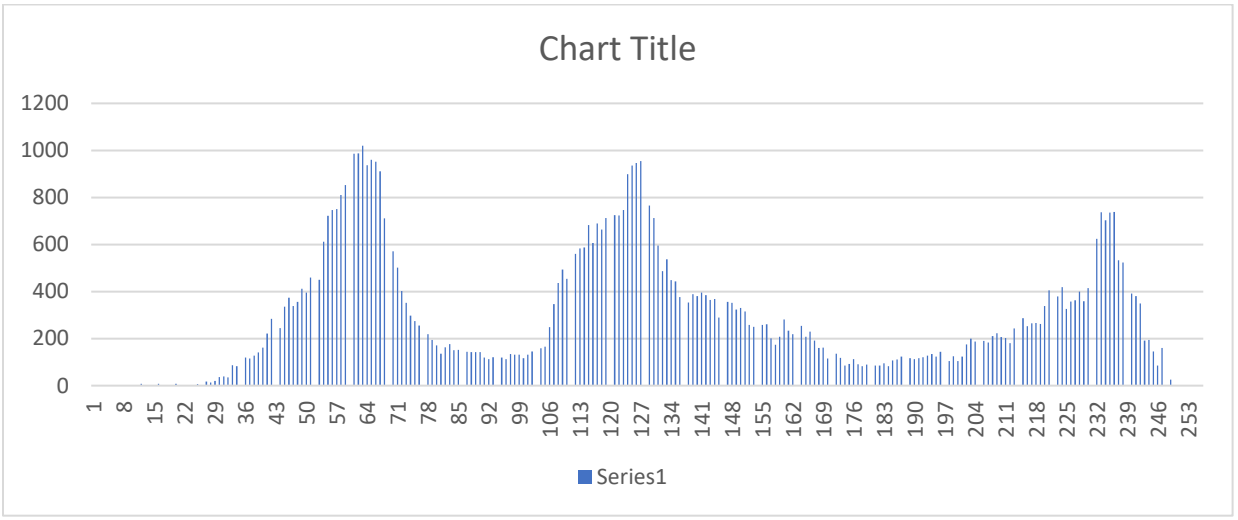


Histogram for Green Channel

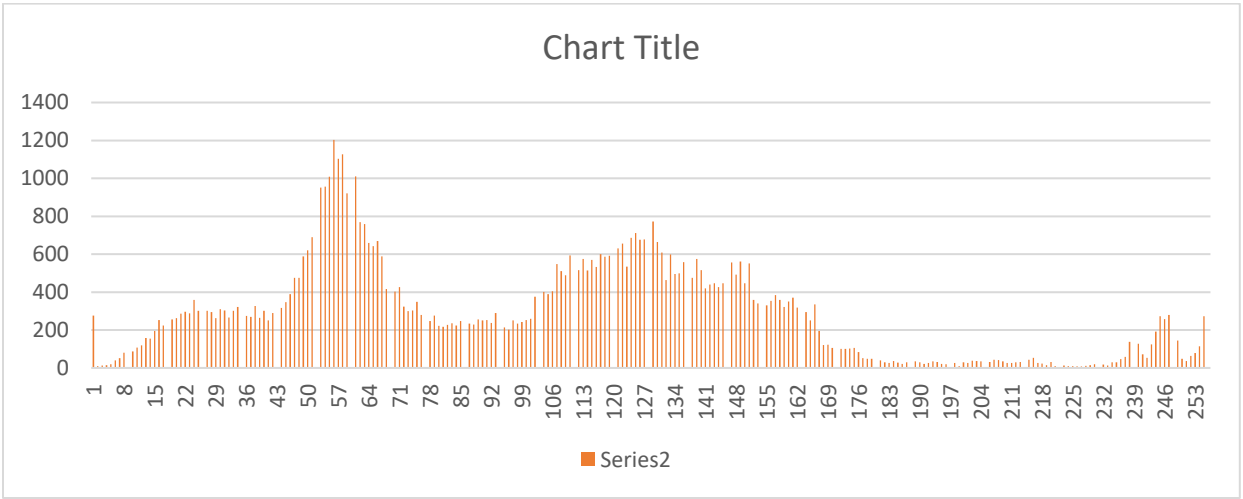


Histogram for Blue Channel

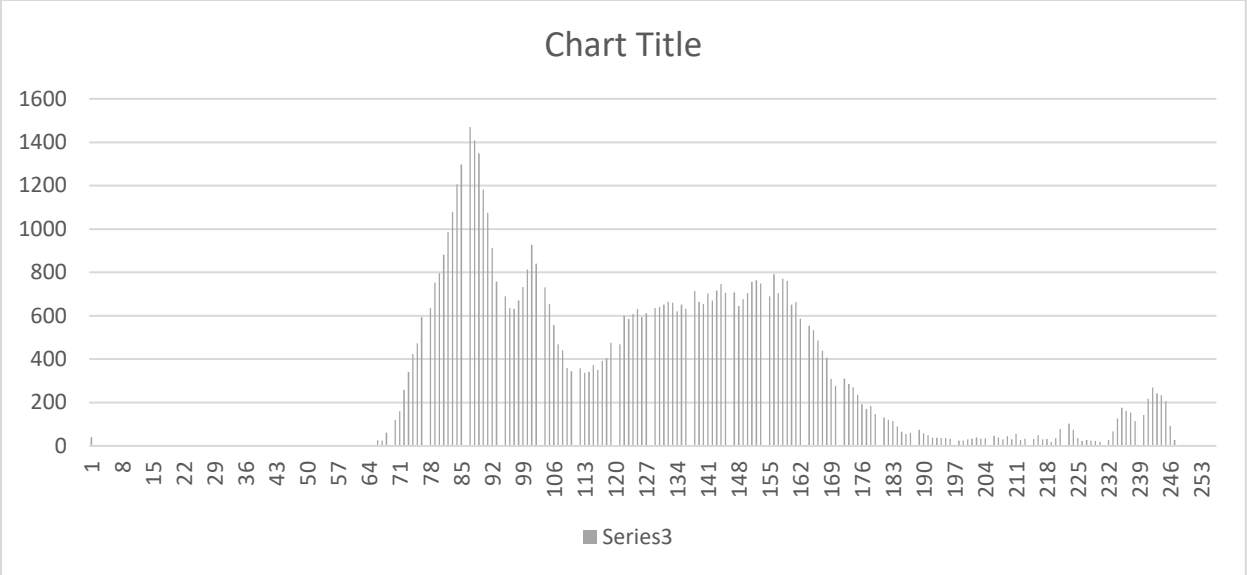
Input image of size 256*256



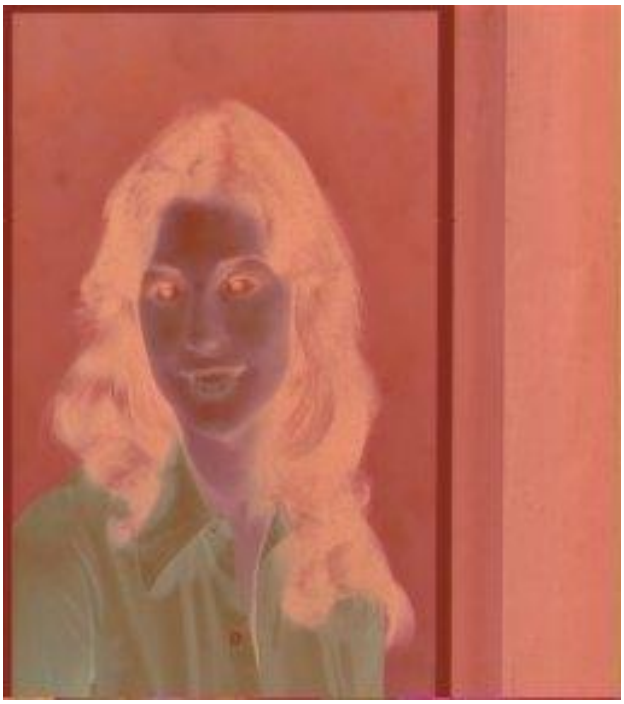
Histogram for red channel for input image



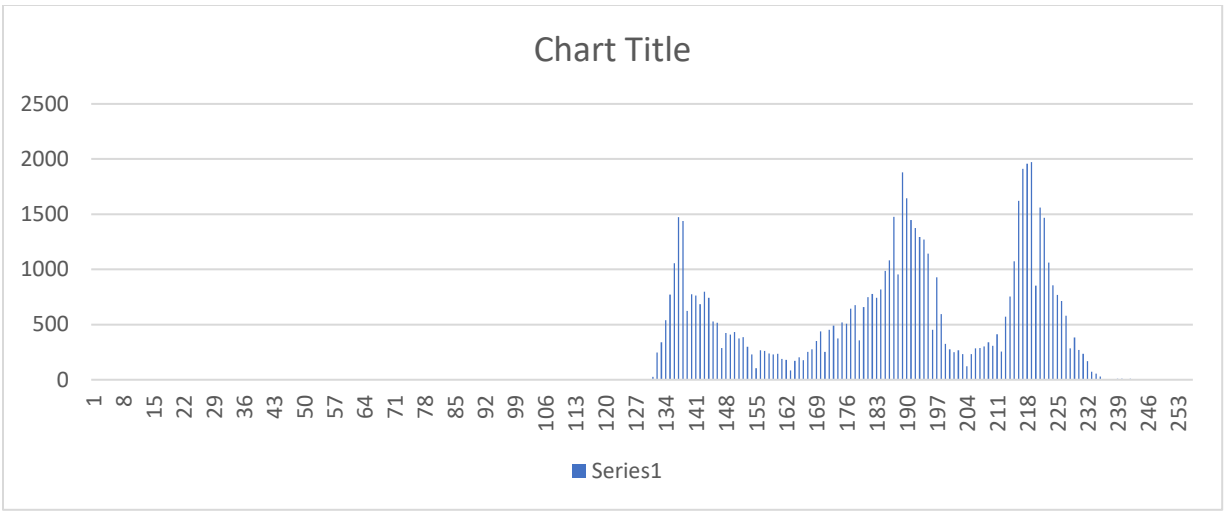
Histogram for green channel for input image



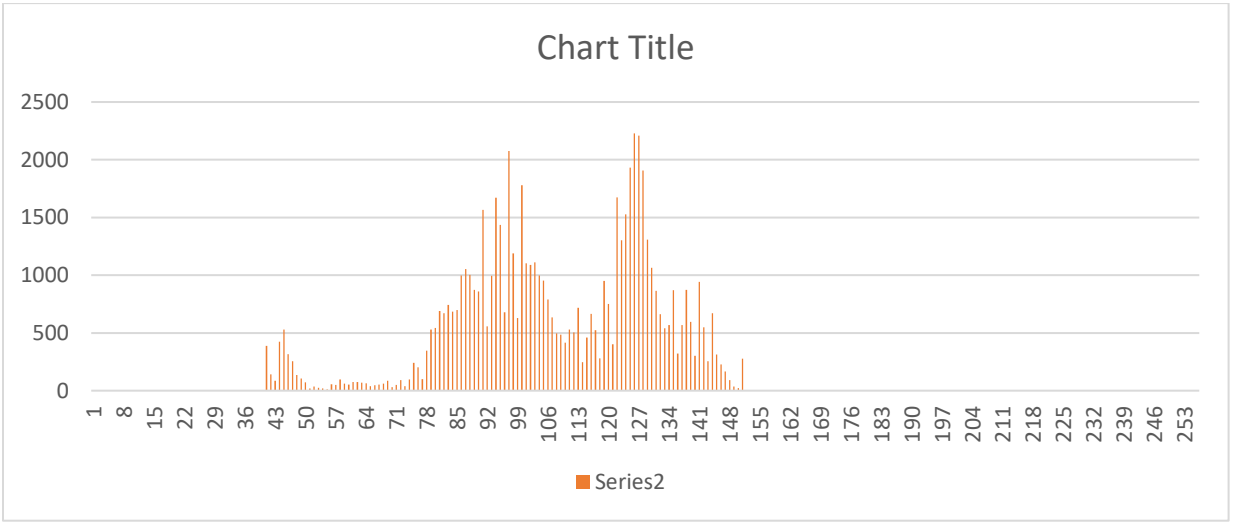
Histogram for blue channel for input image



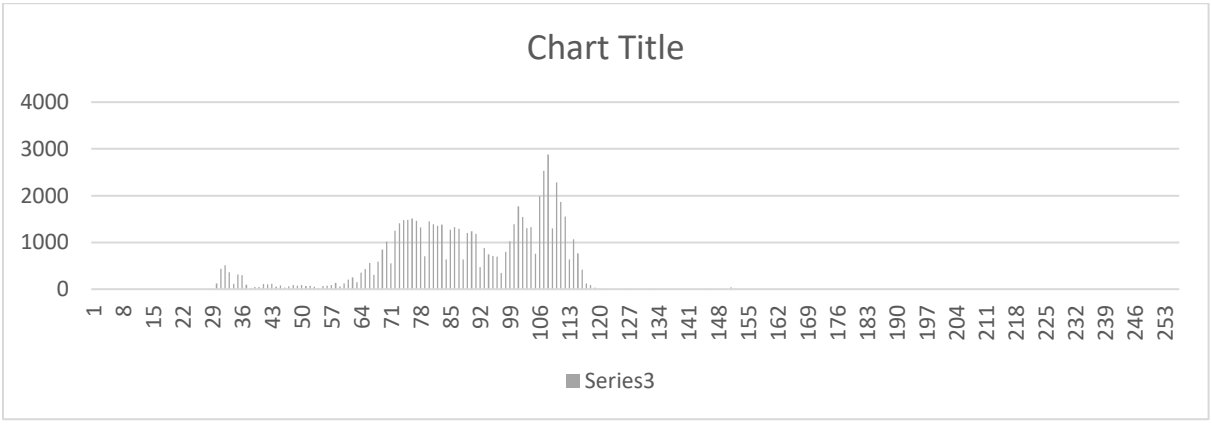
Output Image



Histogram for red channel for output image



Histogram for green channel for output image



Histogram for blue channel for output image

As can be seen from the histogram outputs, the histogram is inverted and contracted. Hence the output image is as seen.

Problem 3.(a)

IV. Abstract and Motivation

Noise are unwanted factors in an image which disturbs the information present in the image. In this problem we are to check the noisesm present in every channel of the image and look for filters to remove noise. Lots of different filters like Mean, Median, Gaussian filters can be used to remove different types of noise.

V. Approach and Procedures

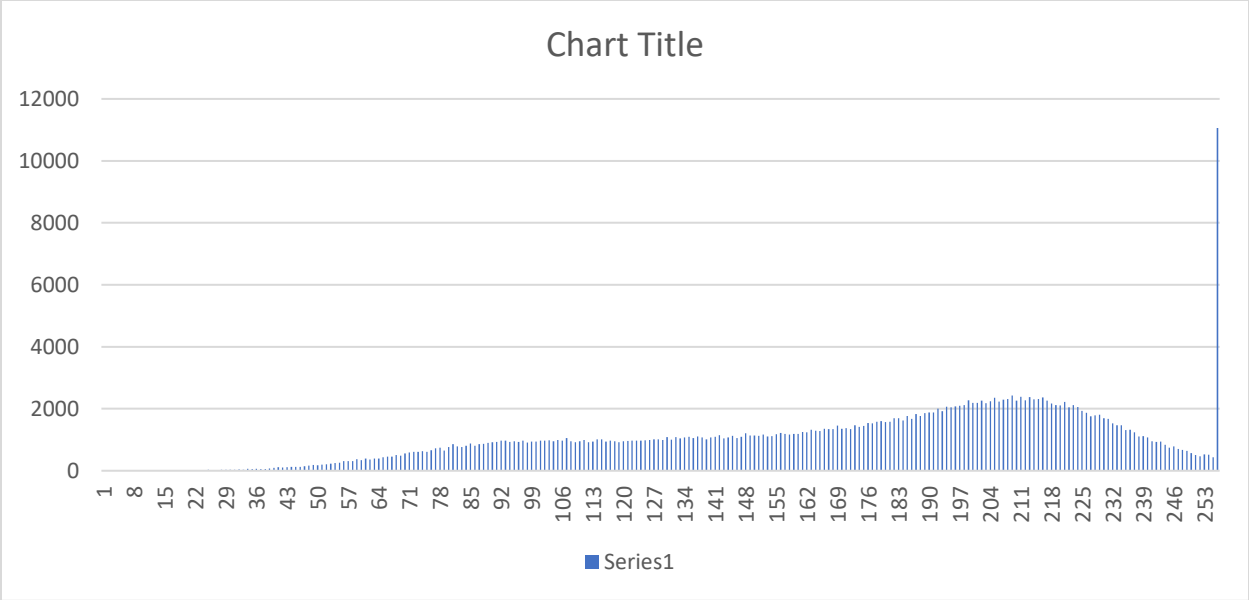
I decided to use the C++ language to write the code for the problem statement. First of all the histogram of the image was checked to find the types of noises present in each channel. Accordingly, noise filters were built and applied to the image. Some of the noise types which were found were salt and pepper noise and gaussian noise. Salt and Pepper noise is inferenced by a very high number of pixels with intensity 0 or 255 in one channel compared to other intensities. They can be remove by median filters which runs in a window and replaces the middle pixel intensity of the window by the median value of all the neighboring intensities in the window. The window keeps on sliding until the entire image is covered.

Similarly Mean Filters can also be utilized as a spatial filter. Mean filter runs in a window and replaces the middle pixel intensity of the window by the mean value of all the neighboring intensities in the window as the window keeps sliding over the image.

Gaussian noise is inferenced by a normal distribution of the histogram of a channel. It has it's pdf which is depicted by the gaussian random variable. Both Mean filters and spatial gaussian filters can be used to remove such noises.

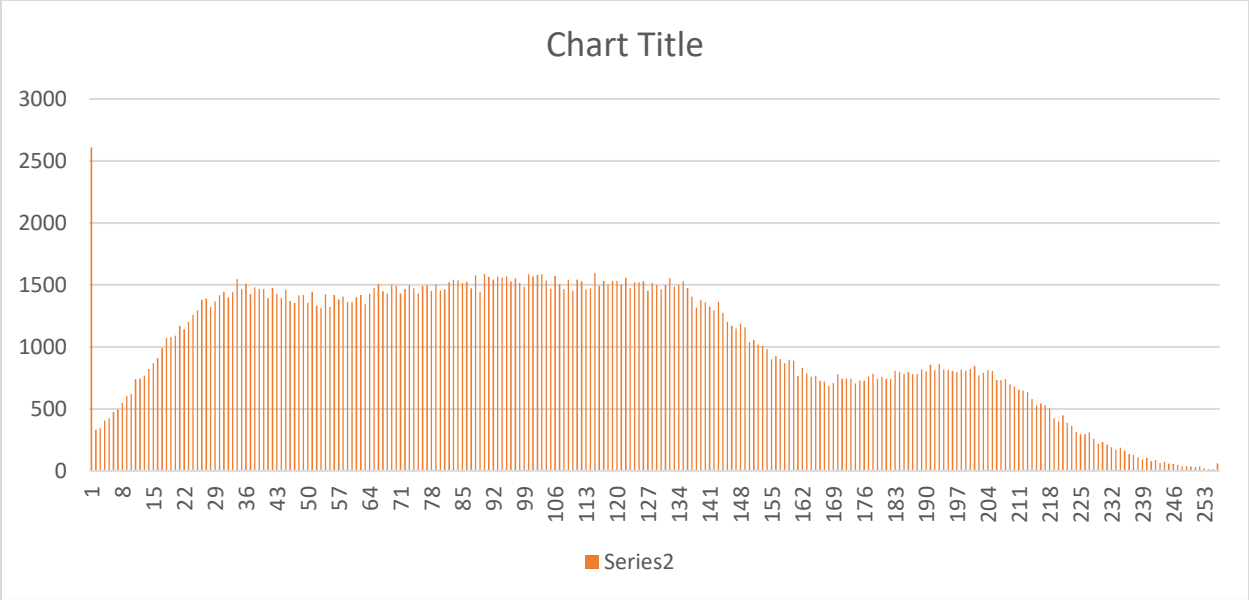
The Peak Signal to Noise Ratio(PSNR) value of an image is calculated to find the performance of the filters utilized. It's a function of mean square error between the original noise free image and denoised image in the denominator. Thus bigger the value of PSNR, better is the quality of the image. An ideal noiseless image will have PSNR value of infinity.

VI. Result and Discussion



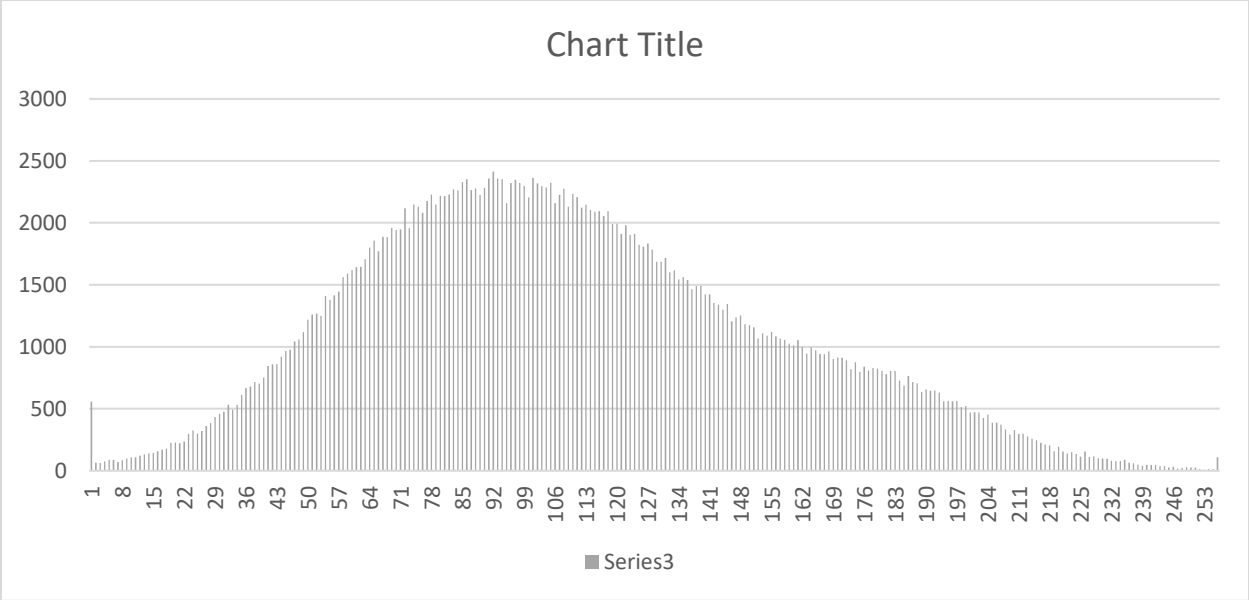
Histogram of Red channel of Noisy Image

As the histogram shows a large number of pixels with intensity 255, it contains salt and pepper noise which can be removed by a median filter.



Histogram of Green channel of Noisy Image

As the histogram shows a large number of pixels with intensity 0, it contains salt and pepper noise which can be removed by a median filter. Also it contains some gaussian noise considering the gaussian shape of the histogram in the middle.



Histogram of Blue channel of Noisy Image

As the histogram shows a gaussian shape, it contains Gaussian Noise. Gaussian Noise can be removed by using mean filter or Gaussian noise.

Thus each channel requires it's own type of filter to remove noise. If there was only one noise type present in the image, cascading wouldn't have mattered. But due to presence of multiple noise in various channels, cascading plays an effect with there being two types present:

Method A: Apply Median Filter first and then Mean Filter
Method B: Apply Mean Filter first and then Median Filter

For Mixed Noise Image:
Channel Red PSNR: 17.0867
Channel Green PSNR 16.5475
Channel Blue PSNR 17.0143

Median Filter of Order 3

Channel Red PSNR: 24.9011

Channel Green PSNR 24.4805

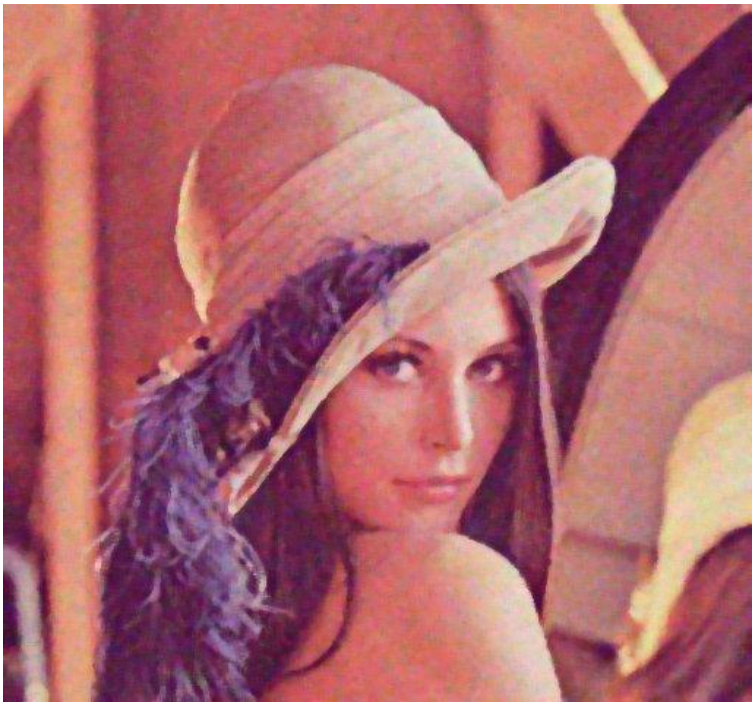
Channel Blue PSNR 22.7546

Median Filter of Order 5

Channel Red PSNR: 26.2266

Channel Green PSNR 26.6009

Channel Blue PSNR 25.2963



Median Filter of Order 7

Channel Red PSNR: 26.0698

Channel Green PSNR 26.1746

Channel Blue PSNR 25.9384

As we can see, as the size of window for the median filter increases, the PSNR value first increases and then decreases. The denoised image is more blurred. This is because larger number of pixels are used in the window to determine the value at the output. This leads to loss of information. Hence the PSNR is best for N = 5 window size.

Mean Filter of Order 3

Channel Red PSNR: 26.0698

Channel Green PSNR 26.1746

Channel Blue PSNR 25.9384

Mean Filter of Order 5

Channel Red PSNR: 23.927

Channel Green PSNR 23.8401

Channel Blue PSNR 25.3769

Mean Filter of Order 7

Channel Red PSNR: 23.7866

Channel Green PSNR 23.5689

Channel Blue PSNR 25.4786

As we can see, as the size of window for the median filter increases, the PSNR value first increases and then decreases. The denoised image is more blurred. This is because larger number of pixels are used in the window to determine the value at the output. This leads to loss of information. Hence the PSNR is best for N = 5 window size.

Cascade Mean Median Order 3

Channel Red PSNR: 23.7866

Channel Green PSNR 23.5689

Channel Blue PSNR 25.4786

Cascade Mean Median Filter Order 5

Channel Red PSNR: 24.417

Channel Green PSNR 23.9528

Channel Blue PSNR 25.7502

Cascade Mean Median Filter Order 7

Channel Red PSNR: 23.8682

Channel Green PSNR 23.4623

Channel Blue PSNR 25.541

The same case is seen when we apply a cascade of Mean filter and then Median filter. As the size of window for the cascaded filters increases, the PSNR value first increases and then decreases. The denoised image is more blurred. This is because larger number of pixels are used in the window to

determine the value at the output. This leads to loss of information. Hence the PSNR is best for N = 5 window size.

Cascade Median Mean Filter Order 3

Channel Red PSNR: 26.6063

Channel Green PSNR 25.4411

Channel Blue PSNR 24.1895

Cascade Median Mean Filter Order 5

Channel Red PSNR: 26.2981

Channel Green PSNR 25.918

Channel Blue PSNR 26.0266

Cascade Median Mean Filter Order 7

Channel Red PSNR: 25.3069

Channel Green PSNR 24.8614

Channel Blue PSNR 25.9541

The same case is seen when we apply a cascade of Median filter and then Mean filter. As the size of window for the cascaded filters increases, the PSNR value first increases and then decreases. The denoised image is more blurred. This is because larger number of pixels are used in the window to determine the value at the output. This leads to loss of information. Hence the PSNR is best for N = 5 window size.

Thus, a cascaded Median Filter and then Mean filter gives a better PSNR value than a cascaded Mean Filter and then Median Filter. This is because Salt and Pepper noise is considered as impulse noise. So when we use Mean Filter at first, we add impulse noises at target pixels as the Mean Filter will calculate the mean value of intensities of neighboring pixel. Thus rather than doing that, first Median Filter should be applied to remove the impulse noise and then Mean Filter can be applied to remove the Gaussian Noise.

Thus a cascaded Median Mean Filter with both window sizes 5 gives the best result and highest PSNR values from the cases considered so I will suggest using this filter to remove noise.

A major shortcoming of this method is that though Mean Filters remove noise, often if edges are present in an image then they are not preserved while using Mean Filter. This is because on both sides of the edge, pixel value changes very fast. And taking the mean of the intensities will just blur out the edge. To improve this, a bilateral filter can be used instead of a mean filter.

Bilateral Filters are non linear filters but they preserve the edge and reduce noise. It does the same thing as the Mean filter, but rather than just averaging, it creates a weighted average for all intensity values in neighboring pixels which is then inserted into the target location. More weight is given to surrounding pixels with similar or closer intensity and vice versa. Thus the edges are conserved in this method.

Problem 3.(b)

I. Abstract and Motivation

Principal Component Analysis(PCA) is a procedure to reduce a number of correlated variables into a smaller number of uncorrelated variables called principal components. The PCA uses covariance matrixes and eigen values to define the principal components. The principal component which comes first corresponding to the largest eigen value accounts for the maximum information. Further components account for lesser amount of data in decreasing order. In this way the important information is stored in the first few variables or components and the rest can be dropped while still retaining the important parts of the data. This is because the first few components define a good summary of the data.

For image denoising, the aim is to remove noise while retaining the important information of the image. PCA creates new images from the uncorrelated values of different images.

II. Approach and Procedures

PCA is always applied to normalized data with the subtraction of the mean from each row of the matrix to represent the image. Then we find the covariance matrix which is real, symmetric and positive semidefinite. So the covariance matrix has some eigen values. The eigen vectors are sorted in descending order along with the corresponding eigen vectors which form the basis. The dimensions corresponding to the smallest eigen values can be removed.

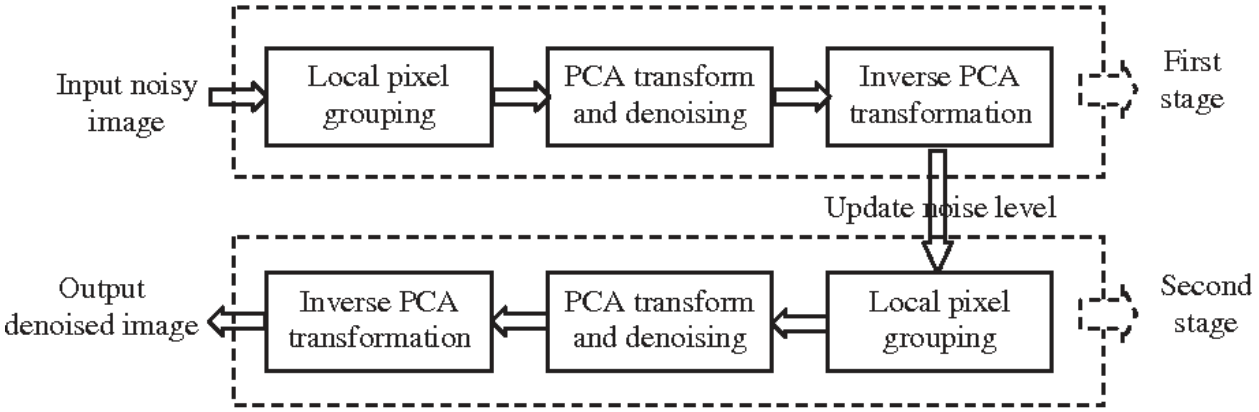


Fig. 1. Flowchart of the proposed two-stage LPG-PCA denoising scheme.

Firstly a moving window is used to calculate the local statistics. Thus patches of pixel which look same are grouped together. An estimate of the image is given by applying PCA on these groups. This estimate of the noisy image is created by dumping away some of the smaller eigen values and their corresponding eigen vectors. This removes most of the noise but retains the important features of the image. Then the inverse PCA transform is applied to the output..This procedure is again repeated on the estimated image to get back the noiseless image. In case of overfitting, the average is taken and given as output.

MATLAB code was got from http://josephsalmon.eu/code/index_codes.php?page=GPPCA which was implemented on the noisy image provided.

III. Result and Discussion

Noisy:
PSNR 34.17

Sigma = 5

PLPCA:
PSNR 39.48

Sigma = 5

Noisy:
PSNR 28.15

Sigma = 10

PLPCA:
PSNR 35.78

Sigma = 10

Noisy:
PSNR 22.13

Sigma = 20

PLPCA:
PSNR 32.37

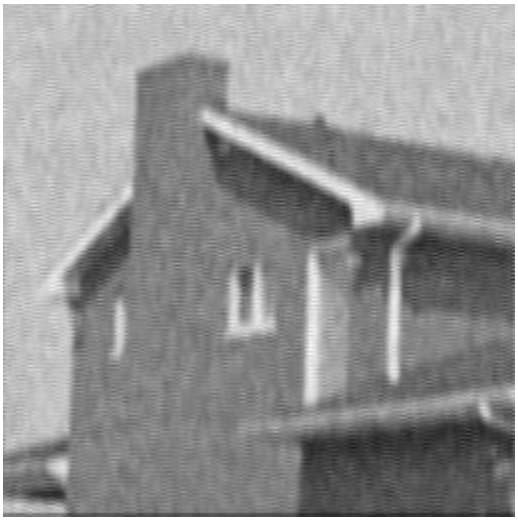
Sigma = 20

The PCA components shows as much variation in the image information properties as possible. These components also help to predict and reconstruct the original image with as less noise as possible. The maximum varaiance while choosing the components give the minimum errors.

The number of components chosen can vary and depending on the number of eigen vectors included in the new image, the output image either losses information or retains the important information of the image.

We can see that the PCLPA algorithm works better as the value of sigma goes down as the PSNR value for the image decreases as Sigma increases. Also the image starts losing information as can be seen by more blockiness and blurring in the image. In this case the code runs with number of parameters chosen as 8 in case sigma is 5, 10 in case sigma is 10 and 11 in case of sigma is 20. Thus optimum value is sigma is 5 and number of parameters as 8 from the cases considered.

Applying the Filters from Question 3 Part a,



Median Mean Filter of order 3

Channel Red PSNR: 24.2519

Channel Green PSNR 24.2495

Channel Blue PSNR 24.2472

Median Mean Filter of order 5

Channel Red PSNR: 23.9185

Channel Green PSNR 23.9163

Channel Blue PSNR 23.9141

Median Mean Filter of Order 7

Channel Red PSNR: 23.0098

Channel Green PSNR 23.008

Channel Blue PSNR 23.0063

As we can see the PSNR value for the denoised image got from PLPCA is much better than that of the Median Mean Filter which was the best filter obtained in question 3(a). Also the image is very blurred when we use the Median Mean cascaded filter.
Thus PCA is a better approach. Due to the use of only those components which contain most of the important information of the image, we get back a better denoised image.

Problem 3.(c)

I. Abstract and Motivation

The BM3D algorithm is an important algorithm to remove noise from image. Given a noisy image, the algorithm first partitions the image into several 2 dimensional blocks. Then grouping is performed for one particular block. In grouping, first one block is taken and similar blocks are found from the image. These blocks are stacked one over the other to form a 3 dimensional array. 3 D transform is then applied to these groups while giving a hard threshold value to the transform coefficients. The transformed 3 dimensional array is then inverted and the results are groups of estimates for the 2 dimensional blocks. These estimates are returned to their original positions to form a basic estimate image. Weighted average are taken into consideration when the blocks overlap.

Then again partition the basic estimate image into several blocks of 2 dimension. For a particular block, then the similar blocks are found from the basic estimated image by the same algorithm. Based on these location, related blocks are found in the noisy image. Both the blocks from the estimated image and the noisy image are grouped separately into two 3 dimensional arrays. Then 3 D transform is applied to both groups, and using the energy spectrum of the first group as standard, Weiner Filtering is done on the second group. The filtered image is then inverted and the estimated blocks are returned to their positions. The final image is obtained by taking the weighted average of the estimate blocks.

II. Approach and Procedures

Online MATLAB code was got from http://www.cs.tut.fi/~foi/GCF-BM3D/ which was implemented on the noisy image provided.

III. Results and Discussion

BM3D Algorithm

Sigma = 25

PSNR = 32.86

BM3D Algorithm

Sigma = 50

PSNR = 29.69

BM3D Algorithm

Sigma = 75

PSNR = 27.51

BM3D Algorithm

Sigma = 100

PSNR = 25.87

Sigma is the standard deviation of the noise. We can see that the best result is got at sigma = 25 as the PSNR is highest in that case. Sigma is used by the filter to eliminate amount of noise thus determining it's performance. When sigma is closest to the actual standard deviation of the noise, the filter can result in a better noiseless image.

BM3D is both a spatial and frequency domain filter. This is because it involves partition of the noisy image into blocks in the spatial domain. Then it deals with the groups formed in the frequency domain to remove noise by utilizing Weiner Filters. Hence it is both a spatial and frequency domain filter.

References:

[1] He, Kaiming, Jian Sun, and Xiaoou Tang. "Guided image filtering," IEEE Transactions on Pattern Analysis and Machine Intelligence, 35.6 (2013): 1397-1409.

[2] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, “Image denoising by sparse 3-d transform-domain collaborative filtering,” Image Processing, IEEE Transactions on, vol. 16, no. 8, pp. 2080–2095, 2007.

[3] http://www.cs.tut.fi/~foi/GCF-BM3D/

[4] Wikipedia.org