EE569 - INTRODUCTION TO DIGITAL IMAGE PROCESSING

Table of Contents

The primary objective behind this assignment is to familiarize with the reading of a '.raw' image, and apply some processing function to it and see the output. Image is a 3D array with bytes repeating in the sequence of R-G-B. Every byte is an unsigned char data type (0-255).

The three important parameters in the reading of an image is its width, height and the number of Channels. The number of channels is 1 if it is grayscale or 3 if it is RGB.

PROBLEM 1

1.1 MOTIVATION :

1.1.a Color Space Transformation

1.1.a.1 Color-to-Grayscale Conversion

A color image is converted to a grayscale image for the following reasons :

- It is easier to understand a grayscale image rather than a colour image as it is only a single channel. The observations can later be used to understand how it is scaled to 3 dimensions - R, G, and B.
- It is computationally faster to read a grayscale image as it only occupies 1/3rd of the memory of a color image.
- Luminance is considered one of the best visual features. For techniques like edge detection, converting an image to a grayscale and checking for edges is easier compared to color images.

1.1.a.2 CMY(K) Color Space

CMYK is mainly used for printing purposes.

- Computer monitors give off light, hence anything digital needs to be RGB.
- CMY is subtractive color model to RGB. Along, with K (Black space), the printer can print the exact colour that was used in the digital copy.

1.1.b Image Resizing via Bilinear Interpolation

Image scaling is one of the most used techniques in image processing. Scaling an image enables the user to crop out (not required) unimportant sections allowing focus on the required portions of the image, or enlarges the image for extensive analysis.

The techniques of cropping and resizing are achieved by downsampling and upsampling of the image data. Downsampling results in loss of image data and thus reduces file size to store the image. On the contrary, upsampling increases the dimensions, filling up the expanded rows and columns.

Bilinear interpolation is the algorithm applied to achieve the above techniques. Interpolation is the process of estimating the value of an unknown pixel based on its neighborhood intensities. As the algorithm works linearly across the width and the height of the image, it is called "Bilinear".

The objective of the problem is to achieve resizing of a 512x512x3 image to a 650x650x3 image. As we are introducing more pixel values in the image, we need to reduce the impulse noise which it generates by evaluating the intensities of its neighborhood values.

1.2 APPROACH :

1.2.a Color Space Transformation

1.2.a.1 Color-to-Grayscale Conversion

The given image is Tiffany.raw of size 512 x 512 x 3. The task is to convert Color image to GrayScale image. Three methods have been proposed to convert an RGB image into Grayscale :

1. Lightness Method :

   'Lightness' of an image corresponds to its brightness with respect to a white color pixel (Intensity value = 255). This contrast reduction method averages the maximum and the minimum component in either Red, Blue or Green channel.

   $$Intensity(x,y) = ((\max(R(x,y), G(x,y), B(x,y)) + \min(R(x,y), G(x,y), B(x,y))) / 2. \qquad (f - 1.1)$$

2. Average method :

This method is the arithmetic mean of the red, green,and blue components of the image. This is an equal weighted method where each pixel obtains ⅓ of its red, ⅓ of its green, and ⅓ of its blue component.

$$Intensity(x,y) = (R(x,y) + G(x,y) + B(x,y)) / 3$$

(f- 1.2)

3. Luminosity method

This weighted method tries to match human perception. Humans are more sensitive to green than any other color. That's why green is weighed more heavily.

$$Intensity(x,y) = 0.21\ R(x,y) + 0.72\ G(x,y) + 0.07\ B(x,y) \quad \text{(f-1.3)}$$

1.2.a.1.ALGORITHM :

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize a set of 'Image' objects to be treated as outputs for the image processing operations.
6. Call the function to convert color image to grayscale using the lightness method applying the formula (f-1.1).
7. Call the function to convert color image to grayscale using the average method applying the formula (f-1.2).
8. Call the function to convert color image to grayscale using the luminosity method applying the formula (f-1.3).
9. Write the obtained images to a '.raw' file and check the output.

1.2.a.2. CMY(K) Color Space

As CMYK is a subtractive color model, the corresponding colors are given by

$$C = 1 - R$$

$$M = 1 - G \quad \text{(f-1.4)}$$

$$Y = 1 - B$$

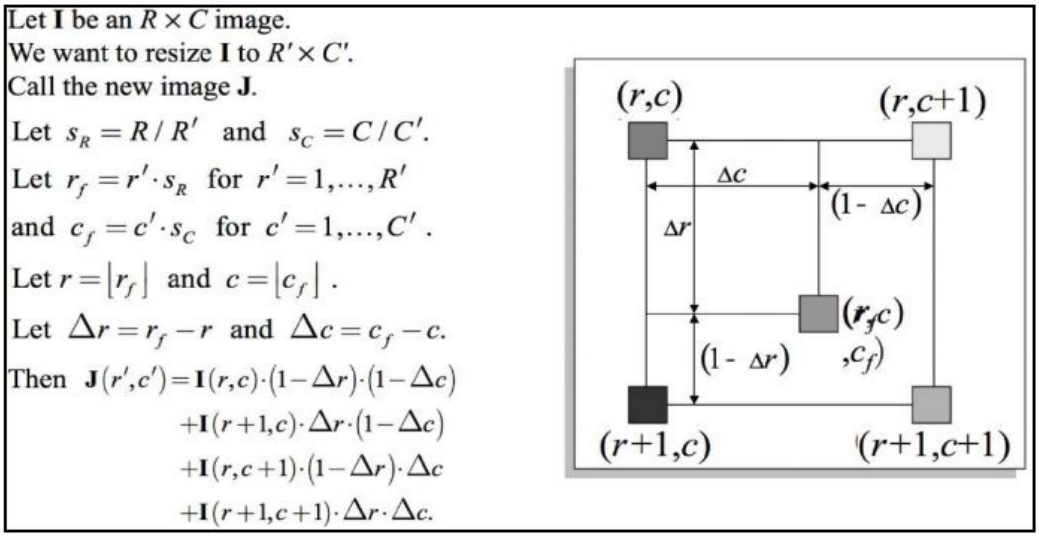
My approach is to normalize the pixel intensities and then subtract from one and then

renormalize the pixels by multiplying with 255.

1.2.a.2.ALGORITHM :

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize a set of 'Image' objects to be treated as outputs for the image processing operations.
6. Split the image into its Red, Green, and Blue components (Single channel images)
7. Normalize the value at the current pixel by dividing by 255 (maximum value). Apply (4) at respective channels.
8. Insert the new pixel intensities at the current location of the output images.
9. Combine all the three single channel CMY channels to get the CMY image of RGB.
10. Write to file the CMY image and the C,M, Y grayscale images to display and check.

1.2.b Image Resizing via Bilinear Interpolation



Source:https://www.codementor.io/tips/7814203938/resize-an-image-with-bilinear-interpolation-without-imresize

The above picture gives an overview of the algorithm. Let's call S_R as the row ratio and S_C as the column ratio. We need to keep in mind that the obtained ratios are not of integer type as it can

have floating point values. r_F and c_f and the new mapping coordinates as they are iterated through the height and width of the new resized image. The floor function truncates the r_f and c_f to the nearest lower integer value. Δr *and* Δc gives us the respective differences in the mapped floating point values to its floored integer values. J is the output resized image.

1.2.b. ALGORITHM:

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize the output 'Image' object to be stored after the image processing operations.
6. Call the resize function which iterates through the width and height of the new resized image size.
7. Get the current pixel byte number = (row-number * width-of-image) + col-number and thus the current pixel byte = current-pixel-number * 3 if it is RGB for respective channels.
8. Apply the above approach to every channel separately - Red, green and Blue. I have combined all 3 channel operations in a single function with '+0'th byte referring to the Red channel, '+1th' byte referring to the Green channel, and the '+2nd' byte referring to the blue channel.
9. Get the new mapping and translated values using the above formulae and insert them into the new resized image pixel values.
10. Pay attention to the index to not be out of bounds at the boundary conditions.
11. Repeat steps 7-10 through the width and height of the resized image.
12. Return the image and write it to file and check the output.

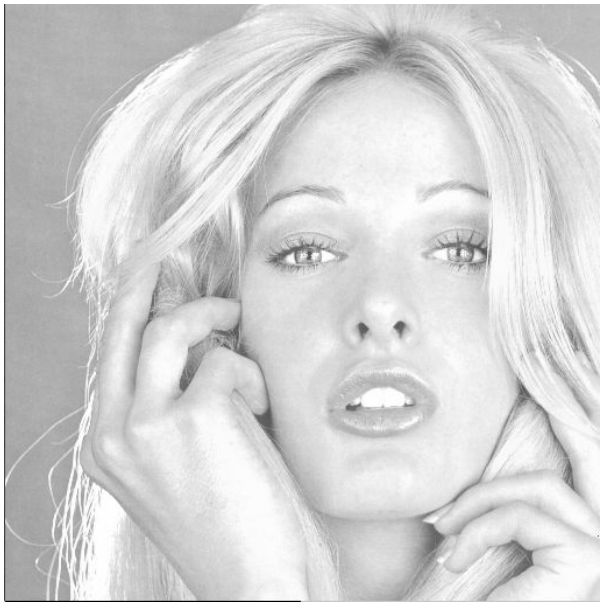
1.3. RESULTS :

1.3.a Color Space Transformation

1.3.a.1 Color-to-Grayscale Conversion

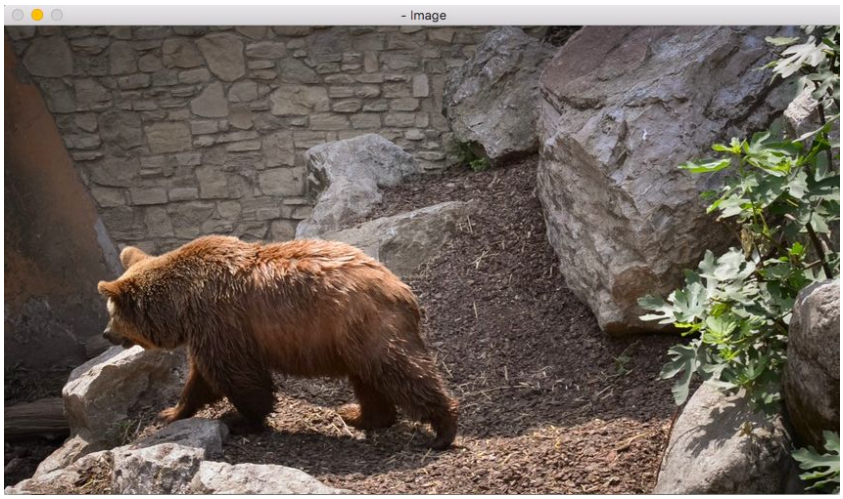
 Original - 'Tiffany.raw' - 512x512x3	 Lightness Method - 'Tiffany.raw' - 512x512x1

Average method - ‘Tiffany.raw’ - 512x512x1



Luminosity method - ‘Tiffany.raw’ - 512x512x1

1.3.a.2. CMY(K) Color Space



Original Bear.raw - 854x480x3

CMYK Bear.raw - 854x480x3

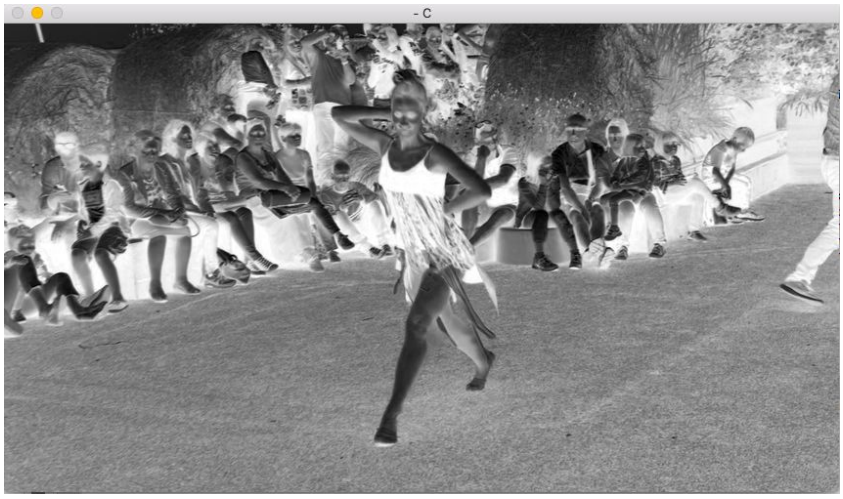
C component Bear.raw - 854x480x1

M component Bear.raw - 854x480x1

Y component Bear.raw - 854x480x1

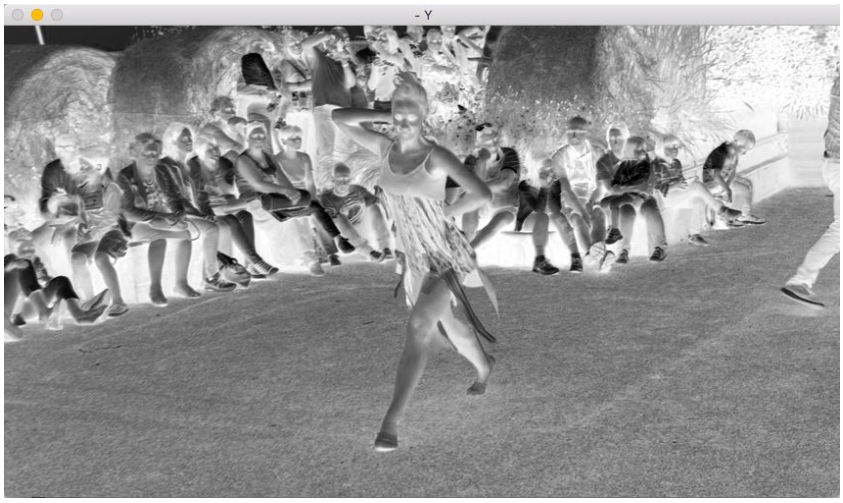
Original image - Dance.raw - 854x480x3

CMYK Image Dance.raw - 854x480x1



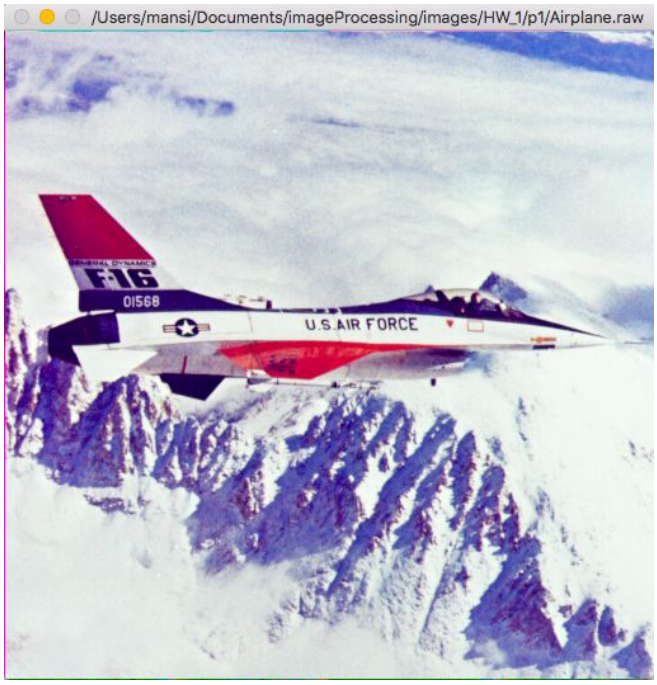
C component Dance.raw - 854x480x1

M component Dance.raw - 854x480x1



Y component Dance.raw - 854x480x1

1.3.b Image Resizing via Bilinear Interpolation



Original Image - Airplane.raw - 512x512x3

Resized image - Airplane.raw - 650x650x3

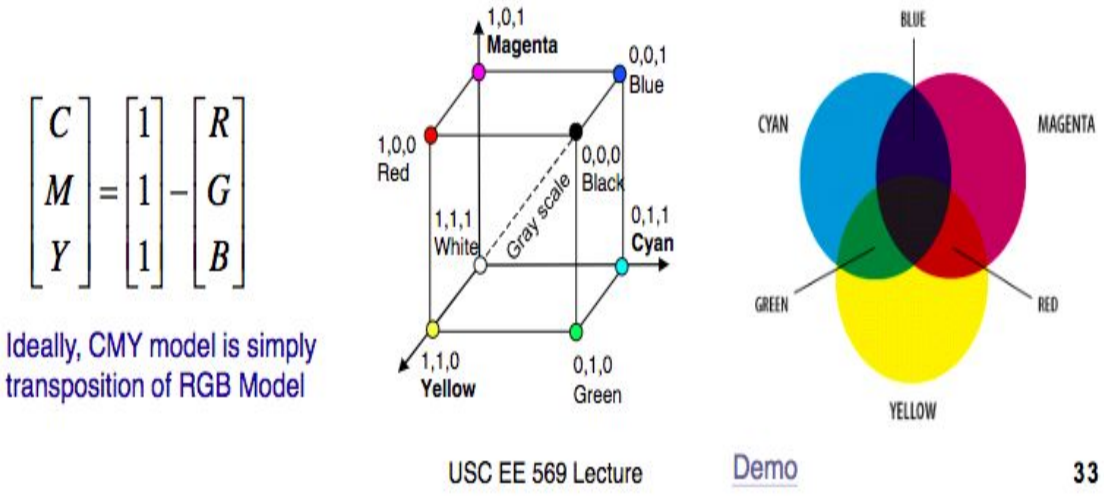
1.4. DISCUSSION :

1.4.a Color Space Transformation

1.4.a.1 Color-to-Grayscale Conversion

Luminosity method looks the best as it is taking the weighted average of the input Red, Green, and Blue channels. Luminance is one of the most appealing visual features. By separating chroma and luma, more defined image characteristics can be observed. Another way to enhance the important characteristics in an image would be to convert it to another color model - HSV, CMY(K) etc.

1.4.a.2. CMY(K) Color Space



Source : Lecture slide

Color is the energy emitted by a luminous object. RGB is the color model used in computer graphical displays. RGB is an additive color model whereas CMYK is a subtractive color model which means that combining two colors in the CMYK color space gives a color in the RGB color Space. For example, combining Cyan and Magenta gives Blue.

1.2.b Image Resizing via Bilinear Interpolation

Bilinear interpolation can be used to enlarge as well as shrink an image to required dimensions. It considers its 2x2 neighborhood, i.e 4 pixels surrounding the current pixel weighted accordingly with respect to its spatial 2D distance to arrive at an estimate for the current pixel value.

As image size expands here, there is upsampling which results in creation of new zero pixel values of data. Bilinear interpolation tends to decrease the sharpness of the image, introduces aliasing, and distorts the edges as only 4 of its neighboring pixels are only considered during its estimation of the current value. Even though this algorithm is easy to implement and compute, it is not preferred as it introduces edge halos. Additionally, this is a non-adaptive algorithm, hence it is not the best resizing algorithm. We can look at other adaptive resizing techniques like 'Adaptive Interpolation Algorithm' to obtain better results.

PROBLEM 2 - HISTOGRAM EQUALIZATION

2.1 MOTIVATION :

2.1.a. Histogram equalization

Frequently when images are retrieved and read, they have low contrast and poor illumination. This results in the image looking dull. Hence, histogram manipulation techniques were introduced to spread the narrow range of contrast of the image intensities, thereby enhancing the contrast.

Histogram equalization technique manipulates image intensities are provides enhanced contrast. Two methods are proposed. They are -

- Method A - the transfer-function-based histogram equalization method

  In this method, we determine a transfer function which maps the original pixel intensity frequencies to the desired (histogram equalized) intensity distribution.

- Method B - the cumulative-probability-based histogram equalization method

  In this method, equal number of pixels are put into 256 bins. This ensures that the histogram is spread out as every intensity value has equal number of pixels in it instead of maximum of the pixels being concentrated to a very narrow range of intensities.

2.1.b. Image Filtering - Creating Oil Painting Effect

Oil painting effect on an image produces an aesthetically pleasing effect to the eye. It consists of two steps -

- Quantizing the color space results in the reduction of representative colors in every channel. The question asks us to quantize to 64 colors - meaning every channel Red, Green, and Blue should have 4 representative colors each.
- The next step comprises of grabbing of the most repetitive color intensity from its NxN neighborhood. This smoothens the image and removes any created impulse noise.

2.1.c. Image Filtering - Creating Film Special Effect

Histogram matching and specification is one of the widely used techniques to produce special film effects. We can either compress, expand, equalize out histogram to obtain the desired effect. In this problem, we are matching the given histogram to the desired histogram

by transfer function based histogram method discussed in previous section and then by CDF mapping.

2.2 APPROACH :

2.2.a. Histogram equalization

Two methods have been proposed to achieve histogram equalization :

- Method A - the transfer-function-based histogram equalization method

A transfer function based histogram equalization method maps the intensity values of the given image to the desired image. The input image is split into its respective Red, Green, and Blue channels. First, the probability densities of the pixel intensities in every channel is calculated. It is followed by calculation of it cumulative density function which gives us our transfer function. The aim of this method is to make the pixel intensity values to follow a uniform distribution.

To calculate the pdf of a certain pixel intensity value 'i', use the formula :

$$pdf[i] = Total\ number\ of\ pixels\ with\ intensity\ value\ 'i' \div Total\ number\ of\ pixels \quad \text{(f-2.1)}$$

CDF is the cumulative density function which sums up the pdf values to 1. The obtained CDF values are multiplied with the maximum intensity value (255) which maps the old pixel intensities to a uniform distribution of pixel intensities. The new intensity value of a pixel is given by :

$$I(x,y) = CDF(I(x,y)) * 255 \quad \text{(f-2.2)}$$

- Method B - the cumulative-probability-based histogram equalization method

This is implemented using the bucket-filling algorithm. The input image is first split into its respective Red, Green, and Blue channels. We set the number of buckets to be 256 (0-255) as there are 256 grayscale intensity levels. We find the number of pixels in each bucket using the formula :

$$Number\ of\ pixels\ per\ bucket = Total\ number\ of\ pixels\ in\ single\ channel \div Number\ of\ buckets \quad \text{(f-2.3)}$$

After calculating the number of pixels in every bucket, we calculate the frequencies of the intensities (0-255) and sort them keeping track of their addresses. So, the address with intensity

value 0 is first stored and then the addresses with intensity value 1. This loops till the end of the intensity range. At the end of the iteration, we have a sorted address array.

Now we fill the buckets with equal number of pixels and the addresses of the pixel locations are provided by the sorted address array. For example, if number of pixels per bucket = 500, then the first 500 locations in the sorted address array will be filled in first bucket ,i.e bucket number 0 (0-255).

This way, we distribute equal number of pixels in every bucket hence resulting in spread out intensity value range.

2.2.a ALGORITHM :

METHOD A : the transfer-function-based histogram equalization method

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize the output 'Image' object to be stored after the image processing operations.
6. Split the input image into its respective channels - Red, Green, and Blue.
7. Calculate the frequencies of pixel intensities and plot a histogram. The histogram gives us an overview of the pixel intensities range and their spread. The histogram of the input image is given in section (2.3.a.(i))
8. Apply the formula given in (f-2.1) to calculate the probability densities of the pixel intensities.
9. Calculate the CDF of pixel intensities using the formula (f-2.2). This CDF gives us an insight to the mapping function.
10. Calculate the new intensities of the pixel values by renormalizing the CDF values, i.e by multiplying the obtained CDF values by 255.
11. Apply the new mapped intensities to the pixels of single channel image.
12. Repeat steps 7-11 for all the channels of the input image.
13. Combine the separate channels into a three channel RGB image and write the output to the file.
14. Calculate the new output intensity frequency values and plot the histogram to check the spreading of the intensity range.

15. Plot the transfer function for all the separate channels - section (2.3.a.(iii))

METHOD B : the cumulative-probability-based histogram equalization method

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize the output 'Image' object to be stored after the image processing operations.
6. Split the input image into its respective channels - Red, Green, and Blue.
7. Calculate the frequencies of pixel intensities and plot a histogram. The histogram gives us an overview of the pixel intensities range and their spread. The histogram of the input image is given in section (2.3.a.(i))
8. Calculate the number of pixels in every bucket using the formula (f-2.3)
9. Begin three nested 'for' loops - first one to check the intensity from 0 to 256, second one to traverse through the height of the image, and the last one to traverse through the width of the array.
10. Check for the i^{th} intensity value at every pixel location as 'i' ranges from 0 to 255.
11. Get the address locations (Byte locations) for that specific 'i'. This way, we already have a address array of sorted intensity values.
12. Get the sorted Address array corresponding to intensities of the entire dimension of the single channel image.
13. Fill the buckets with equal number of pixels (now that we have sorted addresses - we put the bucket number as the data in the address location)
14. Repeat 7-8 for all the three channels.
15. Combine the separate channels into a three channel RGB image and write the output to the file.
16. Calculate the new output intensity frequency values and plot the histogram to check the spreading of the intensity range.
17. Plot the transfer function for all the separate channels - section 2.3.a.(v))

2.2.b. Image Filtering - Creating Oil Painting Effect

Creating Oil Painting Effect comprises of two primary steps -

- Quantization of color space to 64 representative colors - This is done by implementing the bucket filling algorithm with the number of buckets as 4. Equal number of pixels are filled in every bucket using the formula (f-2.2) . After every bucket has equal number of pixels, the representative color is chosen to be the weighted mean of the colour intensities of all the pixels in that respective bucket. In the end, all of the colors in that bucket is represented by a single mean pixel intensity value.
- Most frequent color in its NxN neighborhood - As now the image is blocky, we need to smoothen the image. This is done by selecting the most frequent intensity from its NxN neighborhood. From the NxN neighborhood, an array picks up all of the intensities present there and from that the pixel is inserted with the most frequent intensity. Care is taken with regards to the boundary conditions.

$$searchStep = (WindowSize - 1) \div 2 \qquad \text{(f- 2.3)}$$

My approach is to make all the pixel intensities '-1' as it is invalid if the boundary conditions (index < 0 or index > height or width of the image) less than zero. The searchStep variable iterates over the image in steps of search step and checking for boundary condition in every step of iteration. From the picked up intensities, a frequency histogram is observed and the most frequent pixel intensity is inserted back to the current pixel. This is done for Red, Green, and Blue channels separately.

2.2.b ALGORITHM :

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize the output 'Image' object to be stored after the image processing operations.
6. Split the input image into its respective channels - Red, Green, and Blue.
7. Step 1 - Applying the bucket filling algorithm, 4 buckets will have equal number of pixels from (f-2.2)
8. The sorted Addresses are maintained with respect to the sorted pixel intensities (refer Method B of 2.1.1 for bucket filling).

9. The mean representative color intensity is calculated by taking the weighted average of all the pixel intensities in that respective bucket. Hence every bucket will have a mean representative intensities.
10. Backtrack to all the address in the sorted Address array for every bucket and insert the mean pixel intensity of that bucket in their respective addresses.
11. Repeat 7- 10 for all the three channels and combine them in the end.
12. Write the obtained quantized color output to a file.
13. Step 2 - Find the most frequent pixel intensity by comparing the NxN neighborhood values of that pixel.
14. Initialize two 'for' loops to iterate through the height and width of the single channel quantized image. Calculate searchStep(f-2.3), for every iteration. Initialize two more 'for' loops to loop through the neighborhood.
15. Check for the boundary conditions. If the index of the search becomes less than zero or greater than the image width and height, set the pixel intensity as such locations as '-1' as it will signify that it is invalid.
16. Grab the pixel intensities from its neighborhood in steps of the search step and arrange them in an array.
17. Obtain the frequencies of intensities greater than or equal to zero, as this will skip the '-1' invalid intensities.
18. Find the most frequent intensity and insert that pixel intensity at that current location.
19. Loop through the end of all the four 'for' loops separately for every channel.
20. Write the obtained oil painting effect to a file and check the output.

2.2.c. Image Filtering - Creating Film Special Effect

Looking at the desired film effect image, we observe that the input image needs to be mirrored vertically and then the negative of the image should be taken and analyzed against the desired film effect image. The single channel histograms of the negative of the input image is compared with the single channel histograms of the special effect image(2.3.c.(i)) We observe that the histograms of the individual channels need to be shrinked in order to get the desired effect. This shrinking effect will result in the desired special film effect.

2.2.c ALGORITHM :

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width,

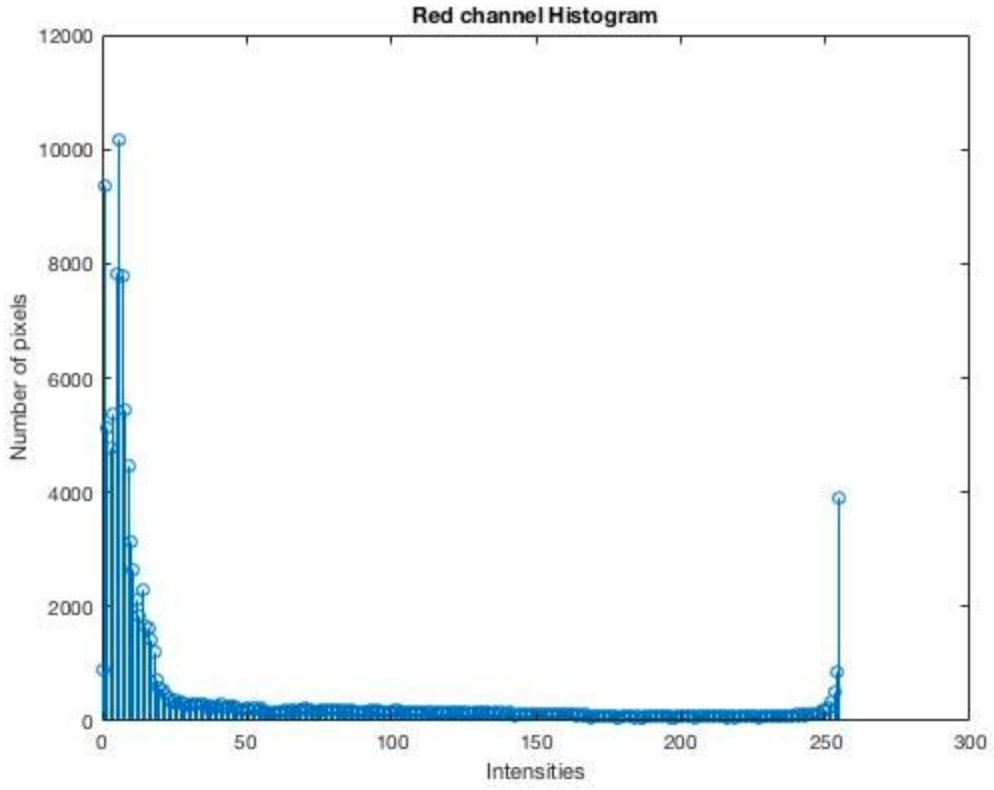
height, and number of channels.

3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested ‘for’ loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize the output ‘Image’ object to be stored after the image processing operations.
6. Mirror the image vertically and then obtain the negative of the image.
7. Load the ‘Original.raw’ and ‘Film.raw’ images using ‘load_image_from_file’ function.
8. Obtain the individual channel histograms of the ‘Original.raw’ and ‘Film.raw’.
9. Obtain the CDF’s of ‘Original.raw’ and ‘Film.raw’.
10. Write the mapping function to match the desired channel histogram.
11. Begin two ‘for’ loops iterating through the range of intensities (0-255) to compare the ‘Original.raw’ CDF and ‘Film.raw’ CDF and calculate the differences in CDF of both and then map the ‘new intensity array’ value as the intensity value with the least absolute minimum difference between CDF’s of ‘Original.raw’ and ‘Film.raw’
12. Get the entire CDF mapping table for all the intensities.
13. Split the input image into its respective channels - Red, Green, and Blue.
14. Map the input image to the new intensities by using the obtained ‘new intensity array’ .
15. Combine the three channels and write image to file.

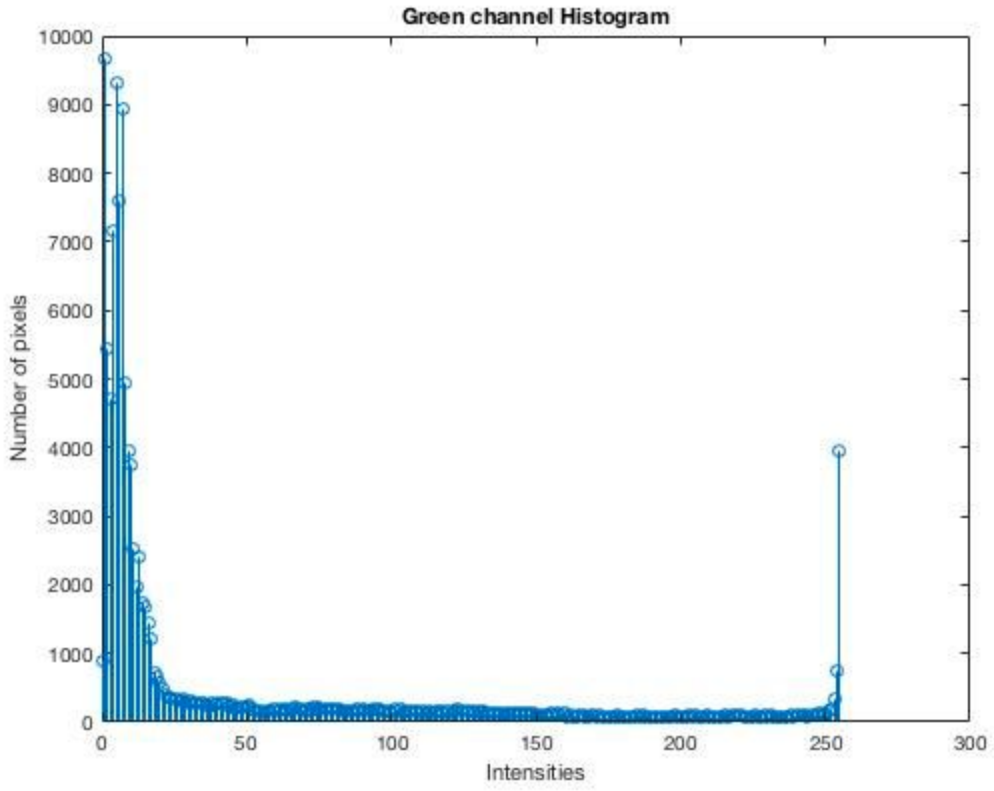
2.3 RESULTS :

2.3.a. Histogram equalization

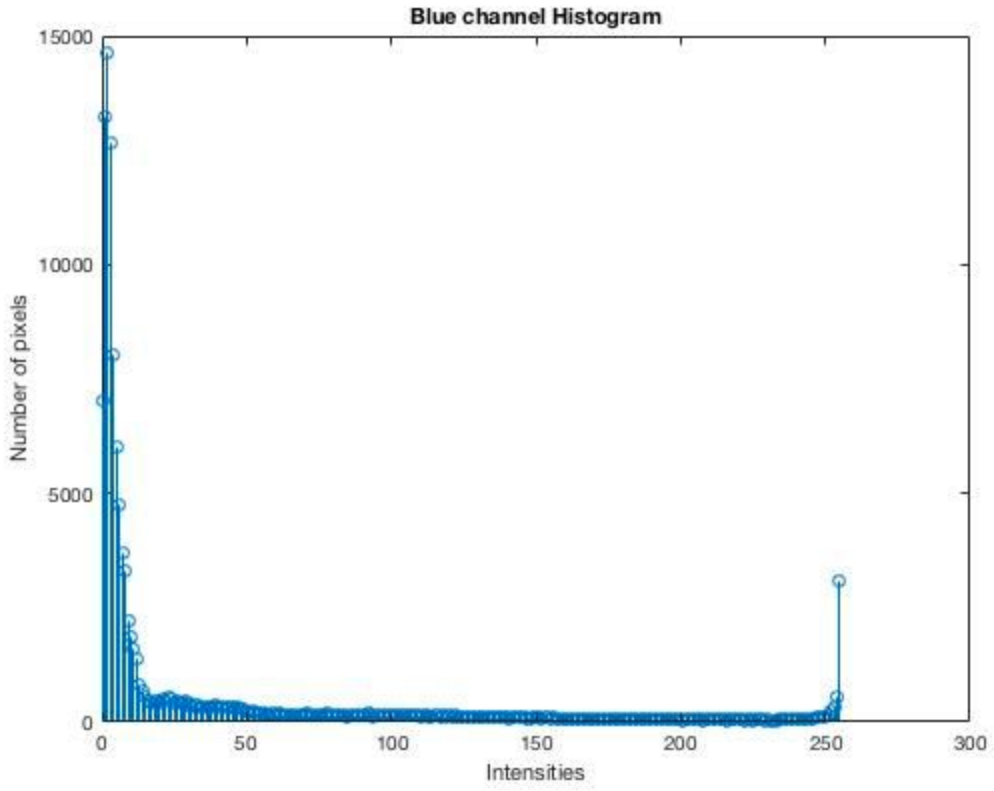
(i) Histogram of the input image



Input image Red channel histogram

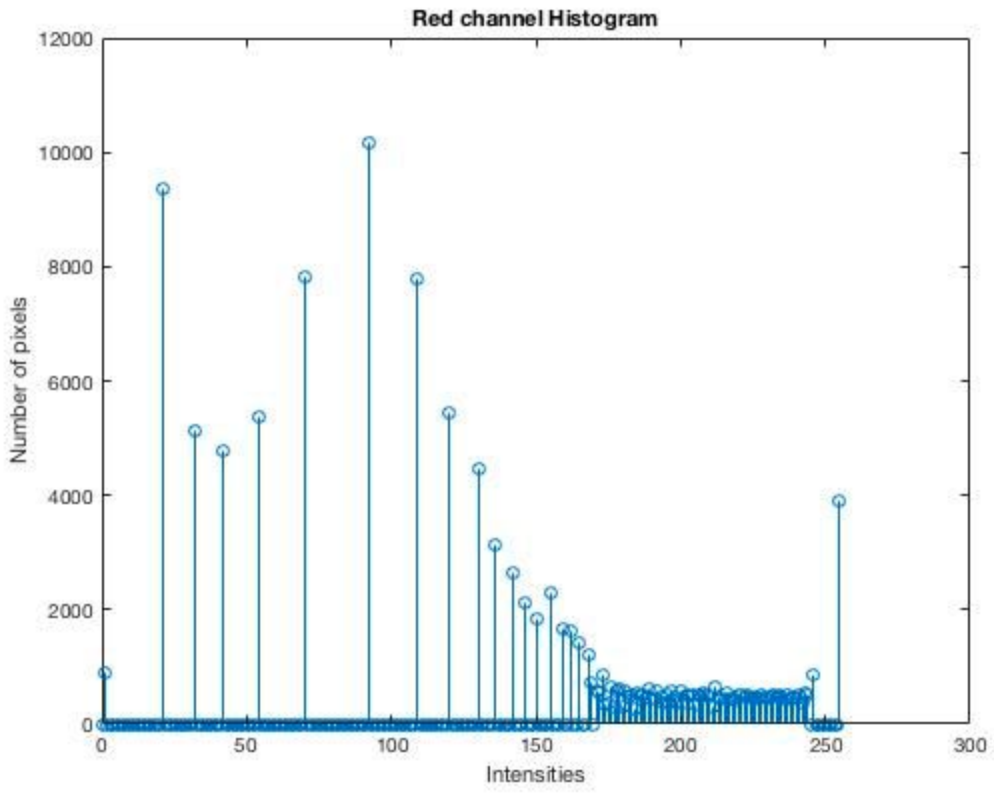


Input image Green channel histogram

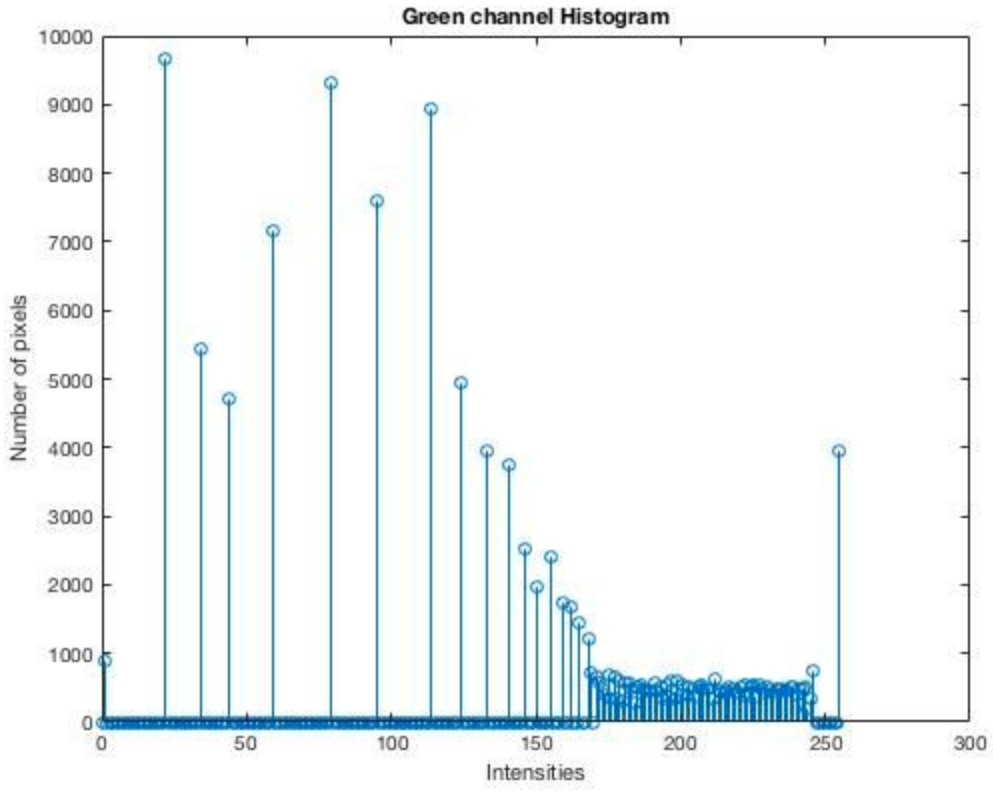


Input image Blue channel histogram

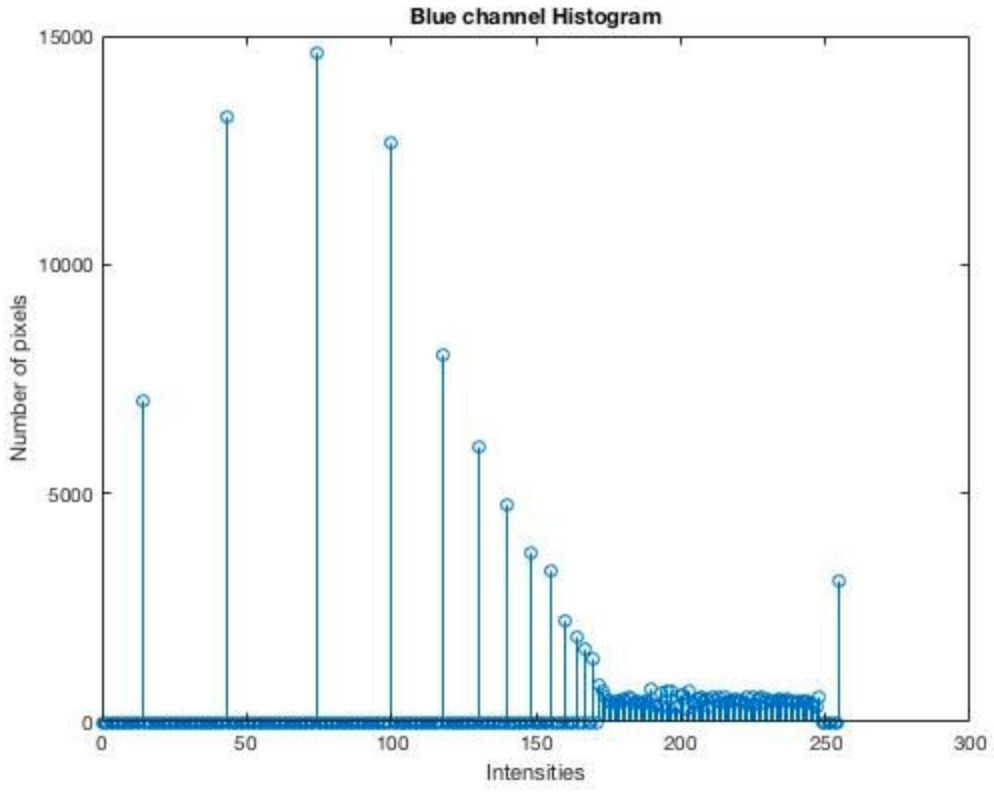
(ii) Histogram after Method A



Red channel histogram after applying Method A

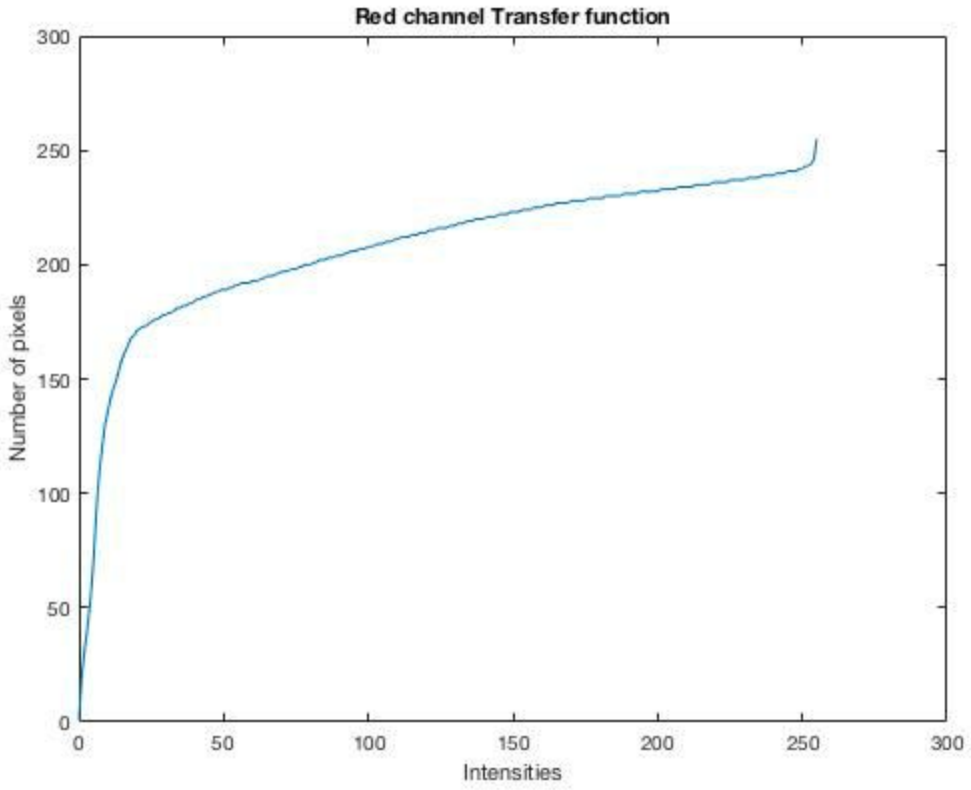


Green channel histogram after applying Method A

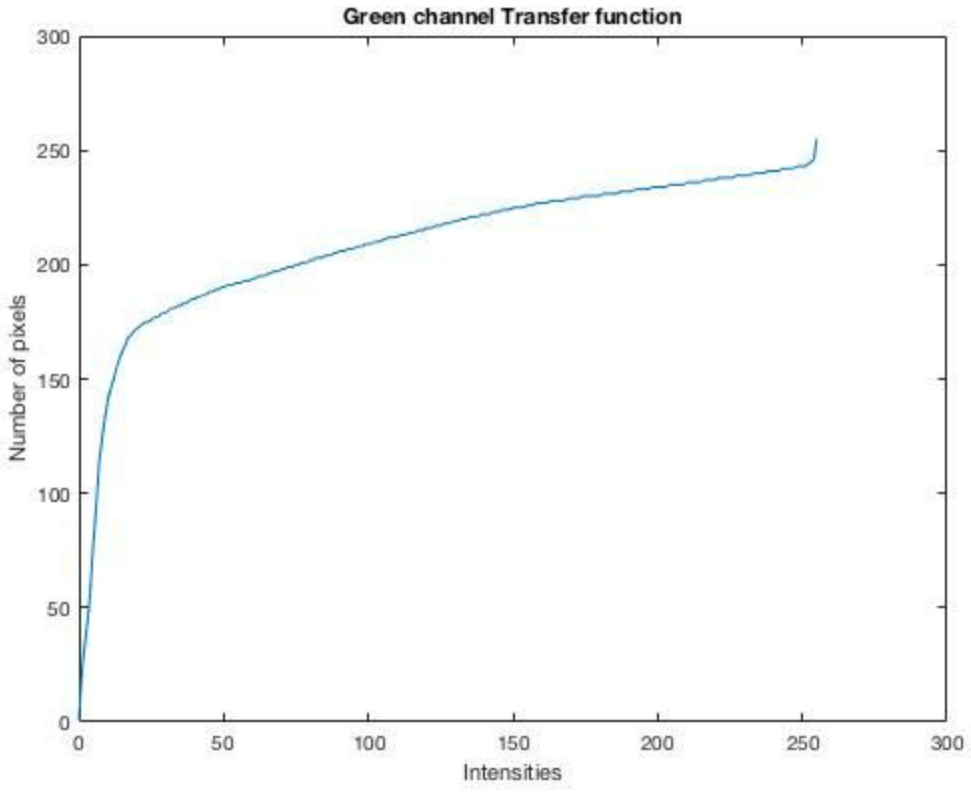


Blue channel histogram after applying Method A

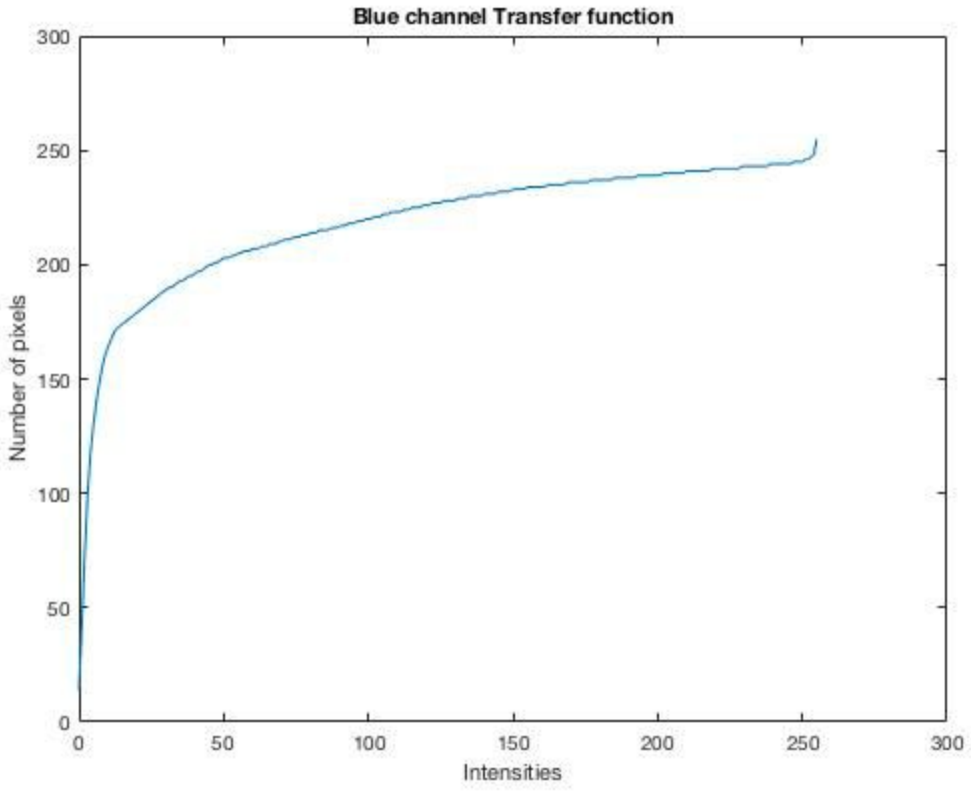
(iii) Transfer function for Method A



Transfer function for red channel - Method A

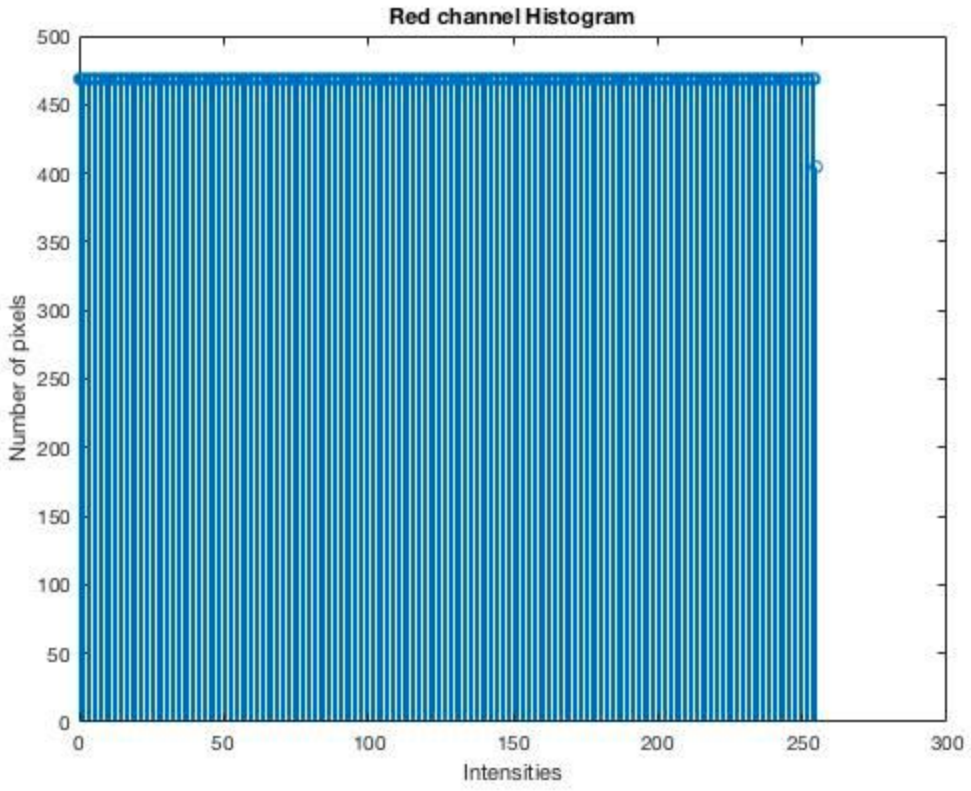


Transfer function for green channel - Method A

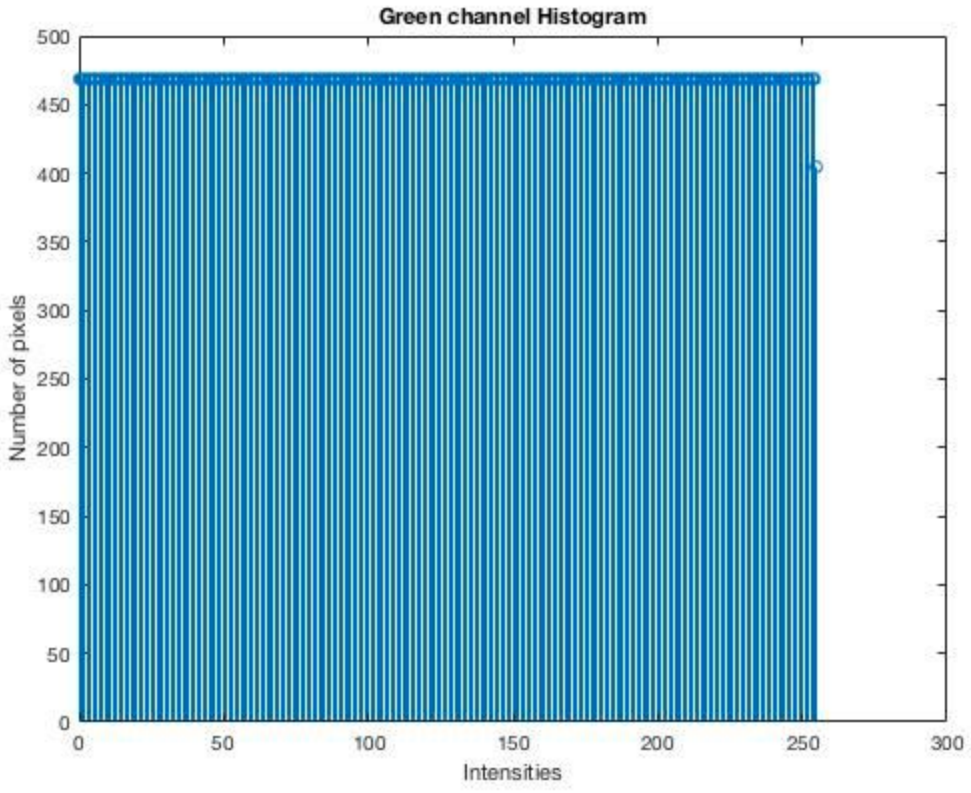


Transfer function for blue channel - Method A

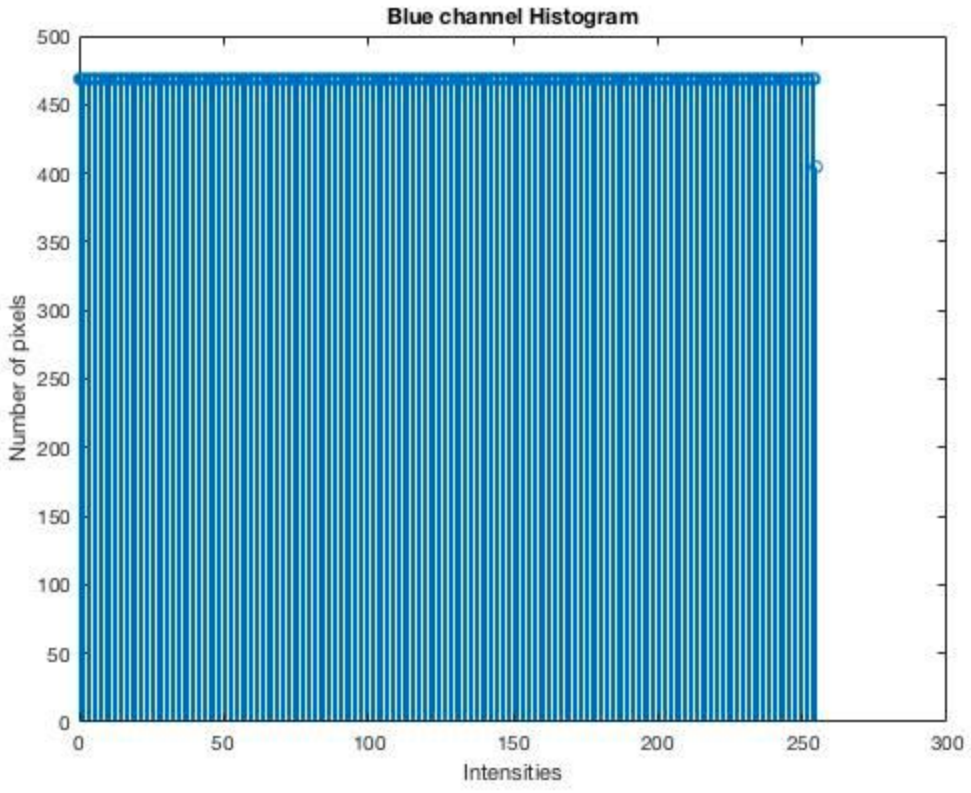
(iv) Histogram after Method B



Red channel histogram after applying Method B

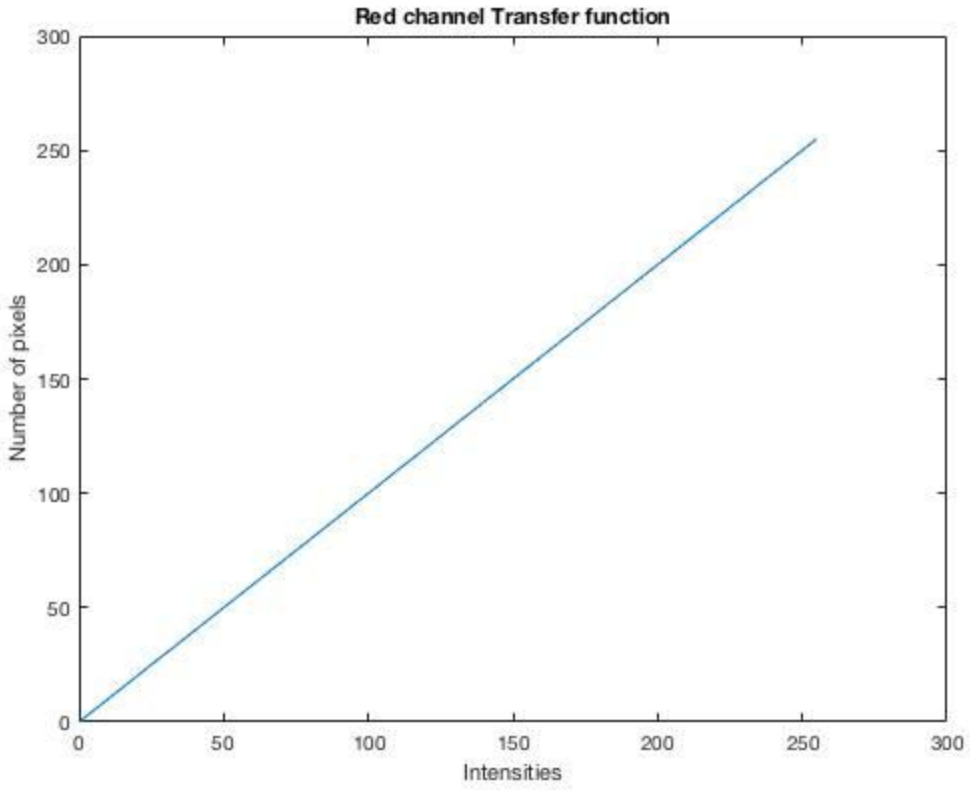


Green channel histogram after applying Method B

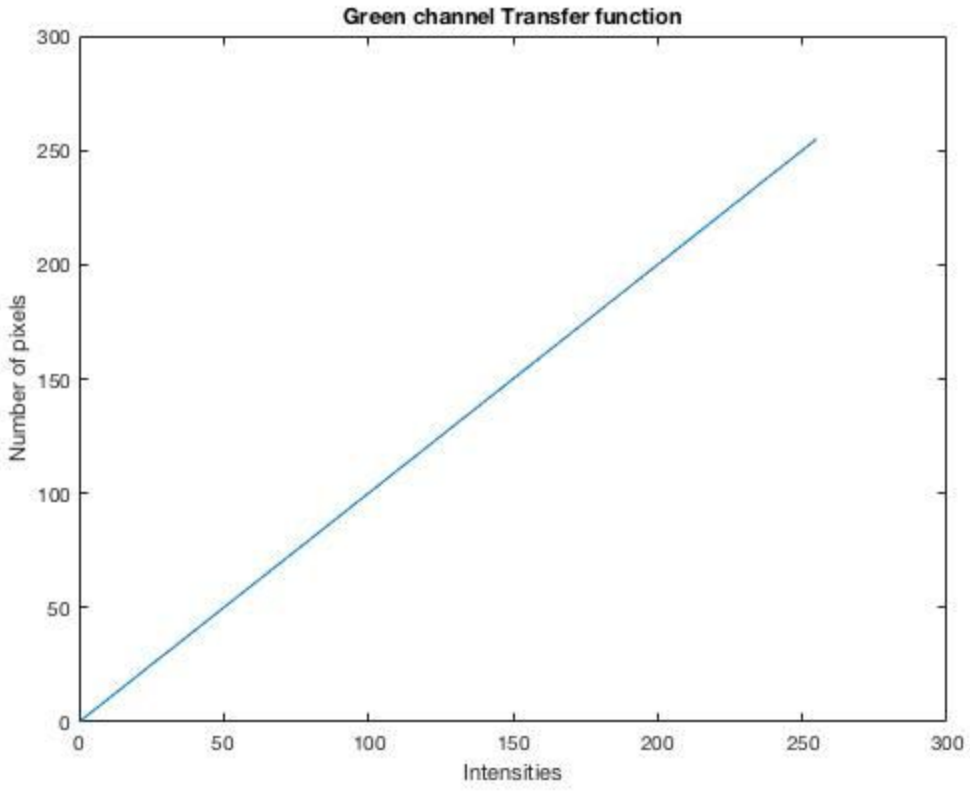


Blue channel histogram after applying Method B

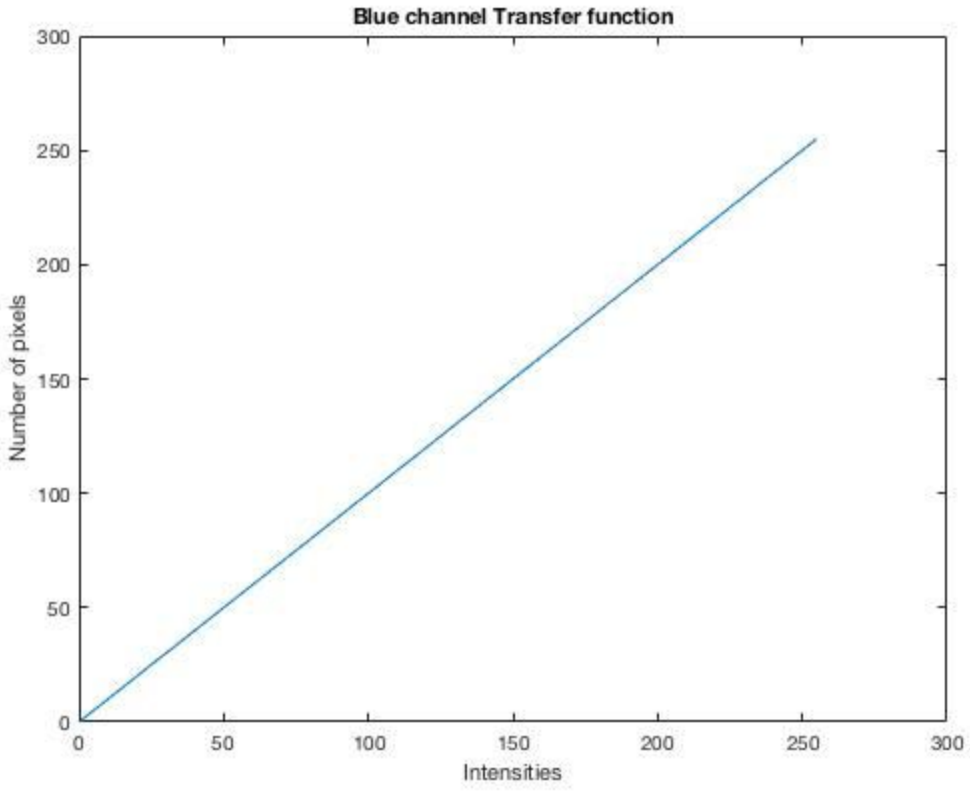
(v) Cumulative histogram for Method B



Red channel Cumulative histogram for Method B



Green channel Cumulative histogram for Method B



Blue channel Cumulative histogram for Method B

(vi) Enhanced images

Input image - Desk.raw - 400x300x3

Histogram equalized image -Method A- Desk.raw - 400x300x3

Histogram equalized image -Method B- Desk.raw - 400x300x3

2.3.b. Image Filtering - Creating Oil Painting Effect

(i) 64 color version of the images with N =5

Original - Trojans.raw - 1800x1200x3

Quantized - Step 1 - Trojans.raw - 1800x1200x3

Nearest Neighbour - Step 2 - Trojans.raw - 1800x1200x3

Original -Star_Wars.raw - 600x338x3

Quantized -Step 1 - Star_Wars.raw - 600x338x3

Nearest neighbor -Step 2 - Star_Wars.raw - 600x338x3

(ii) 512 color version of the images with N=5

Quantized -Step 1 - Star_Wars.raw - 600x338x3

Nearest Neighbour -Step 2 - Star_Wars.raw - 600x338x3

Quantized -Step 1 - Trojans.raw - 1800x1200x3

Nearest Neighbor -Step 2 - Trojans.raw - 1800x1200x3

(iii) Trojans.raw - 64 color version with N= 3

(iv) Trojans.raw - 64 color version with N= 7

(v) Trojans.raw - 64 color version with N= 11

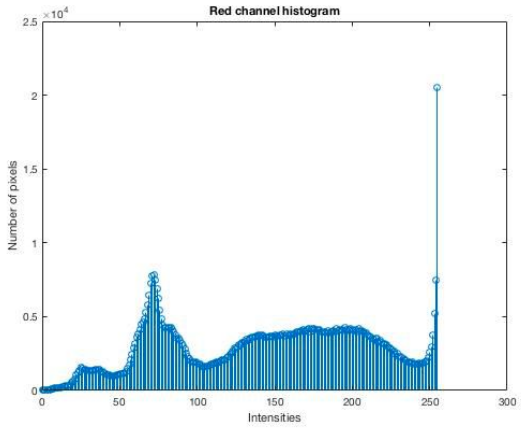
(vi) Star_Wars.raw - 64 color version with N= 3

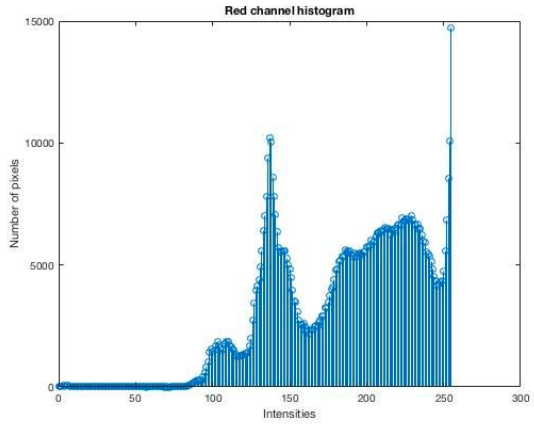
(vii) Star_Wars.raw - 64 color version with N= 7

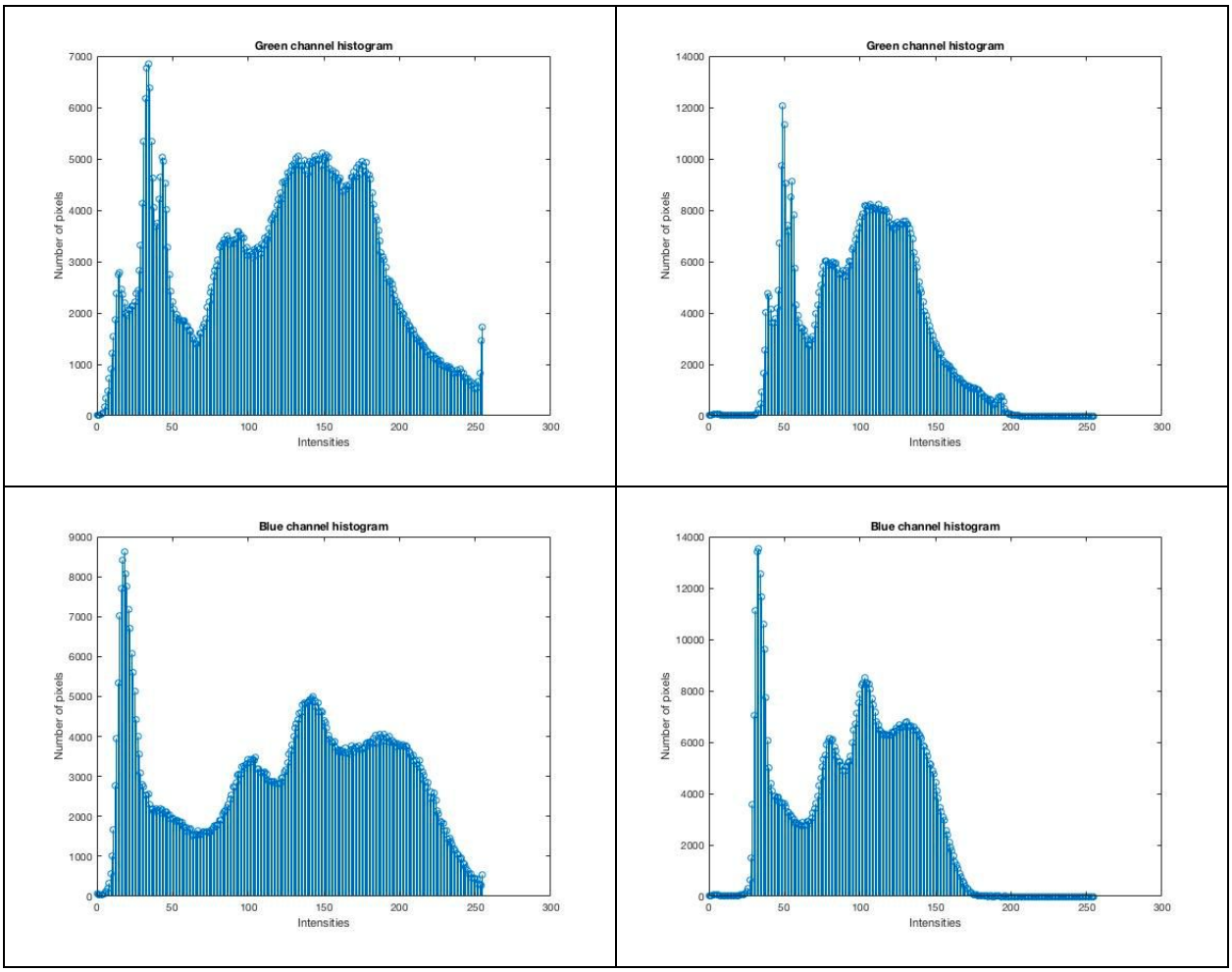
(viii) Star_Wars.raw - 64 color version with N= 11

2.3.c. Image Filtering - Creating Film Special Effect

(i) Histogram of 'Original.raw' (left) and 'Film.raw' (right) - R,G, B order





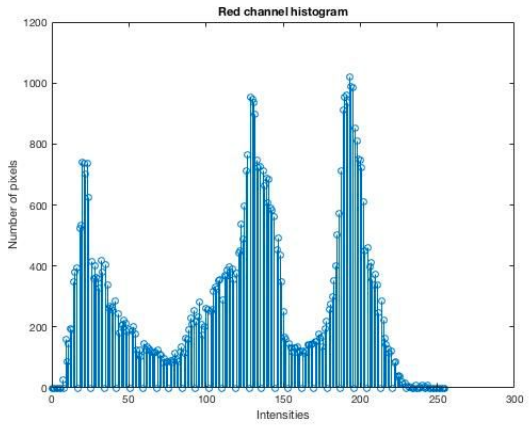

Green channel histogram
Number of pixels
Intensities
Green channel histogram
Number of pixels
Intensities
Blue channel histogram
Number of pixels
Intensities
Blue channel histogram
Number of pixels
Intensities

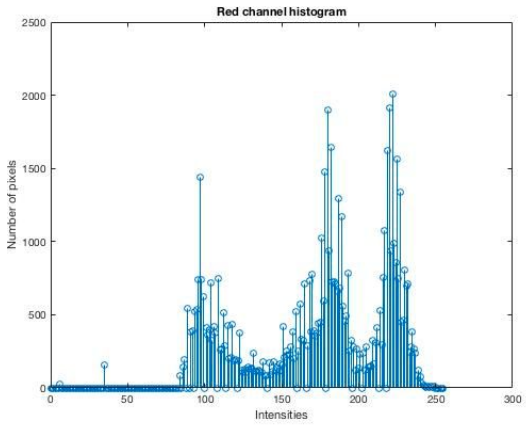
(ii)Special film effect for 'Girl.raw'

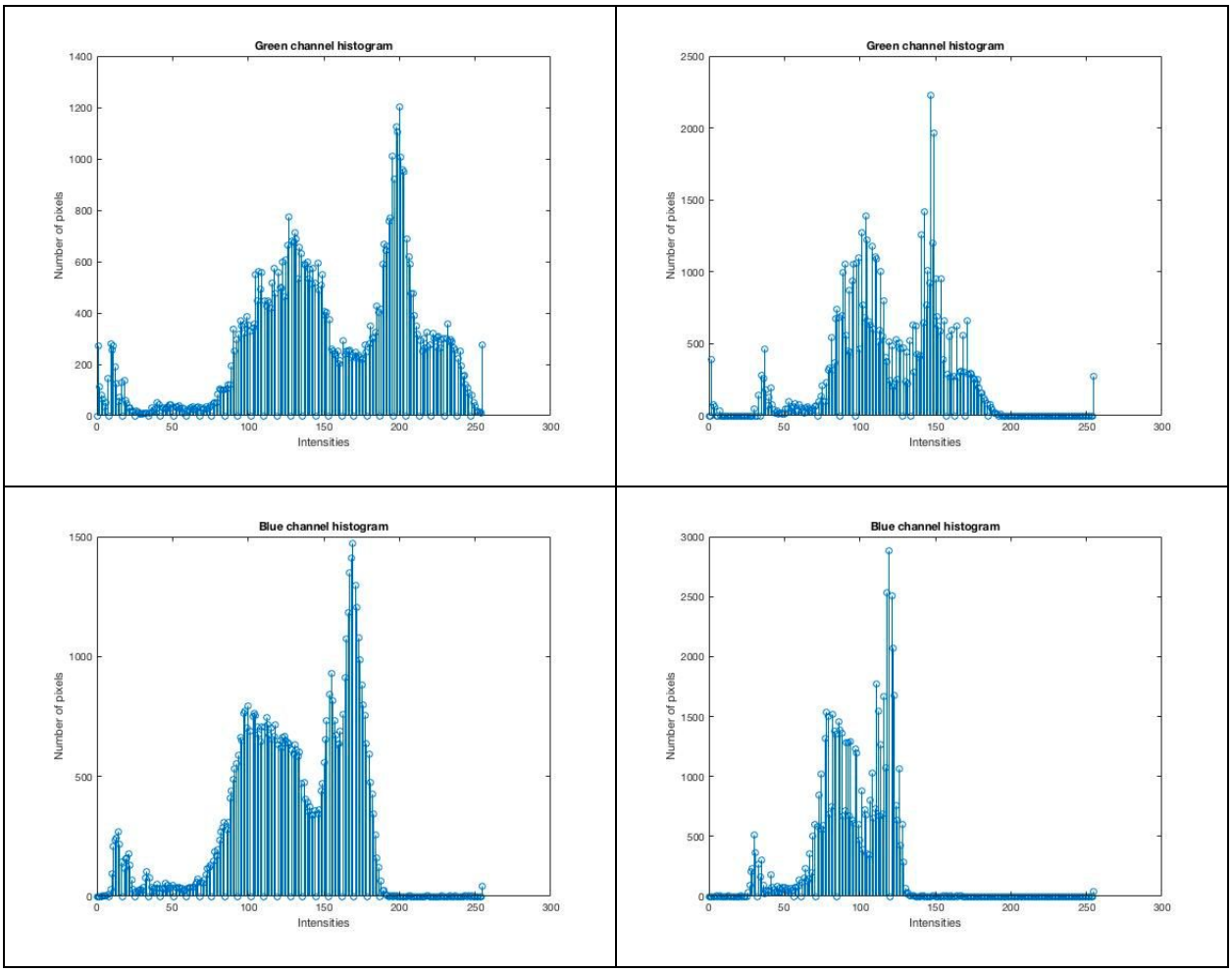
Girl.raw - 256x256x3

GirlEffect.raw - 256x256x3

(iii) Histogram of 'Girl.raw' (left) and 'GirlEffect.raw' (right) - R,G, B order






Green channel histogram
Number of pixels
Intensities
Green channel histogram
Number of pixels
Intensities
Blue channel histogram
Number of pixels
Intensities
Blue channel histogram
Number of pixels
Intensities

2.4. DISCUSSION:

2.4.a. Histogram equalization

Histogram equalization widens the range of the frequency intensities thus improving and enhancing contrast.

Following observations can be made from the above proposed methods -

- Method A -the transfer-function-based histogram equalization method : This method is easy to implement and is not complex. It improved the range of the frequency intensities uniformly. From section () - of the histograms we can see that the entire range is still not used effectively. Also, it generates a one-to-one mapping from the input values to the desired values. We also observe distortion primarily in the top right corner of the image. Distortion is caused as the method is not capable to differentiate between pixels while simultaneously trying to increase the contrast between them.
- METHOD B - the cumulative-probability-based histogram equalization method : The bucket filling algorithm ensures that all the intensities have equal number of pixels in them. This is complex compared to method A as it forcefully demarcates the pixels into its respective bins. An added effort in this method would be that we need to separately maintain the sorted addresses array. But the histogram obtained is much wider than the Method A. The cumulative distribution function is also linear. Distortion is still observed even in method B. It also looks like it has over-smoothed the image.
- To overcome the shortcomings of both the methods, a survey of histogram equalization algorithms was done. 'Recursive Mean Separate Histogram Decomposition' is one of the techniques which recursively decomposes the image 2n times. It found to reduce distortion but increases computational complexity. More details can be found here [1].

2.4.b. Image Filtering - Creating Oil Painting Effect

- Oil painting effect is a compromise between preserving the pixels without blurring and withholding the effect of a painting. The mean values were selected by averaging the weighted mean of all the pixels intensity values in that bucket. This mean value is the representative pixel intensity for that bucket.
- N =5 provides us with a good trade off between preserving the pixel values without

blurring and conserving the effect of oil painting as it is not too large a window to completely smoothen the pixels nor too small a window to preserve any impulse factors. The outputs of 4 colors and 512 colors with N = 5 is shown in section(2.3.b.(ii))

- N = 3 maintains the pixel intensities but does not provide a good oil painting effect as its only compared with 9 other neighboring intensities. N= 7 works good as it compares 49 of its neighbors, N = 11 compares 121 of its neighbors, but it smoothens the image a lot. Hence, N= 5 is an optimum choice for the NxN neighborhood window.
- When the color palette is increased from 64 to 512 color intensities, there are more representative colors and hence it is more colorful than 64 colors. Colors are better preserved as they are better represented. (This can be clearly seen with the Trojans image - where the sword is much clearly visible - section(2.3.b.(ii)))

2.4.c. Image Filtering - Creating Film Special Effect

We can see from the histograms from section(2.3.c.(i)) that the histogram is being shrunk to the desired histogram.

This way we see that histogram matching is achieved for any reference desired CDF via the CDF matching function.

PROBLEM 3 - NOISE REMOVAL

3.1 MOTIVATION :

Noise results from random variation of pixel values which are not consistent with its surroundings. Noise can be introduced in an image through various ways like through sensors and photodetectors. There are several types of noise in digital images - random noise, salt and pepper noise, banding noise etc. Several techniques have also been introduced to combat noisy signals in images. They are -Gaussian filters, Weiner filters, median filters, mean filters etc.

Power signal to noise ratio (PSNR) value quantifies how good the image looks to the human eye. It provides us an idea about how much noise is present in the signal. It's formula is given by -

$$PSNR\ (in\ dB) = 10log10\ (\ Max^2\ \ \div MSE\ )$$

$$MSE = 1\ \div NM(\sum\sum(Y(i,j) - X(i,j))^2)$$

where, $X = original\ noise - free\ image\ of\ size\ N \times M$

$Y = filtered\ image\ of\ size\ N \times M$

$Max = maximum\ possible\ pixel\ intensity = 255$

3.1.a. Mix noise in Color image

In this problem, we need to denoise the input noisy 'Lena.raw'. Looking at the histogram of the noisy input image, application of median(outlier) filter to remove the salt & pepper noise for all the three channels separately was decided. Next, application of mean filter to smoothen the image was decided.

Application of filters is only through the masking window. The size of masking window is kept to be odd N = 3, 5, 7 as it is convenient.

3.1.b. Principal Component Analysis

Another method of noise removal is introduced. This method performs better than the methods in part (a).

3.1.c. Block Matching and 3D Transform filter (BM3D)

PCA poses to be heavy on computation. Hence another noise removal technique is introduced.

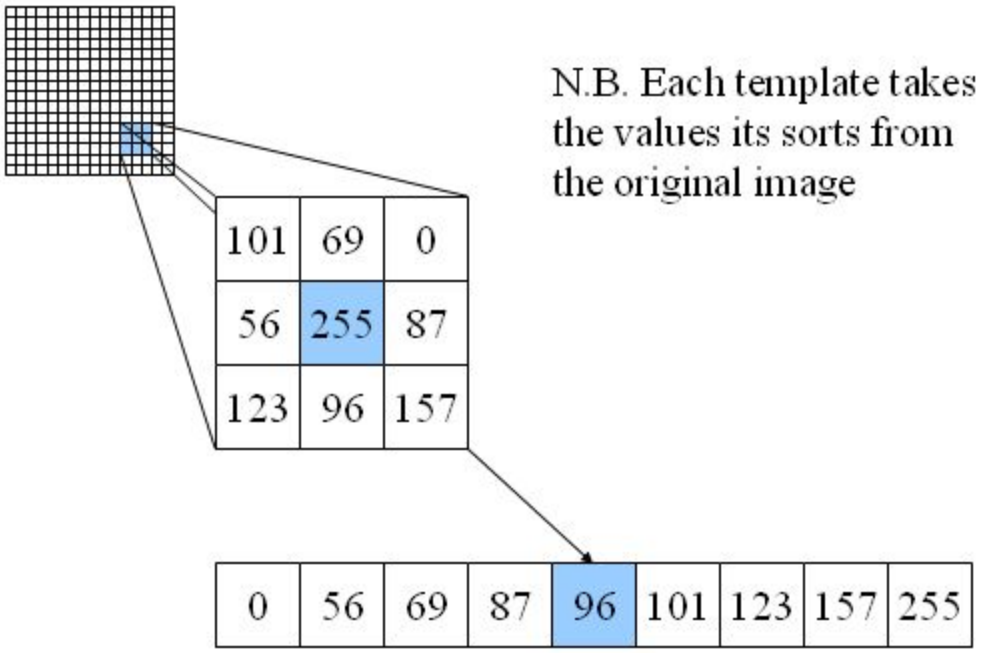
3.2 APPROACH :

3.2.a. Mix noise in Color image

To analyze the noise in the given image, we first need to separate the image into its respective Red, Green , and Blue channels. We can then analyze the single channel noisy images to decide the application of filters to remove the noisy components. The histograms of the single channel noisy images is shown in section (3.3.a.(i))

We can observe from the histograms that there is salt & pepper noise in all the three channels at varying degrees. It is present in the red channel (at 255), green channel (at 0), and blue channel (sudden increase at 255).

To denoise the single channel noisy images and to remove the salt & pepper noise, outlier filter needs to be implemented. The working of the outlier (median filter) is shown below -



Source : http://users.ecs.soton.ac.uk/msn/book/new_demo/median/

The NxN median filter picks up the median value from its neighborhood and assigns the current pixel with that value. So if there are any outliers present, at 0 and 255 which constitute the salt & pepper noise, it is removed as the median value is picked and assigned to be the new value.

The median filtered output is shown in the section(3.3.a.(iii))

After the channels have been filtered using the median values, the obtained single channel images are again analyzed for further steps.

Random noise is observed in the blue channel primarily. A mean filter is applied to blue channel individually. This filters out all of the under-represented and unrepresented pixel values in its surroundings by comparing with its NxN neighbors. This reduces the amount of variation in intensities from its surroundings thereby smoothening the image.

3.2.a. ALGORITHM :

1. Load the input raw image using the function 'load_image_from_file' function.
2. Initialize the 'Image' class variables with respective input image parameters i.e, width, height, and number of channels.
3. Allocate memory for the input image.
4. Read the input array as a single dimension array inside two nested 'for' loops iterating over the rows(till the height) and columns(till the width) of the input image.
5. Initialize the output 'Image' object to be stored after the image processing operations.
6. Separate into three individual channels and analyze the single channel images.
7. Use the NxN neighborhood function used in the previous sections to obtain the frequency distribution of the intensity values.
8. Sort the obtained frequency values and skip '-1' because it is an invalid intensity value. Pick the median value and assign it to the current pixel. Thus, median filter is implemented.
9. For the mean filter for the blue channel, again use the same NxN neighborhood function used in previous sections and obtain frequency distribution of intensity values and apply it to single channel filtered median channels.
10. Skip value '-1' as t is invalid, obtain the weighted mean of the pixel values and assign it back to the current pixel value.
11. Repeat steps 7 - 10 for all the channels separately.
12. Combine all the filtered mean channels and write the denoised image to a file.
13. Calculate the PSNR value for the entire image. It was calculated to be 26.2538dB

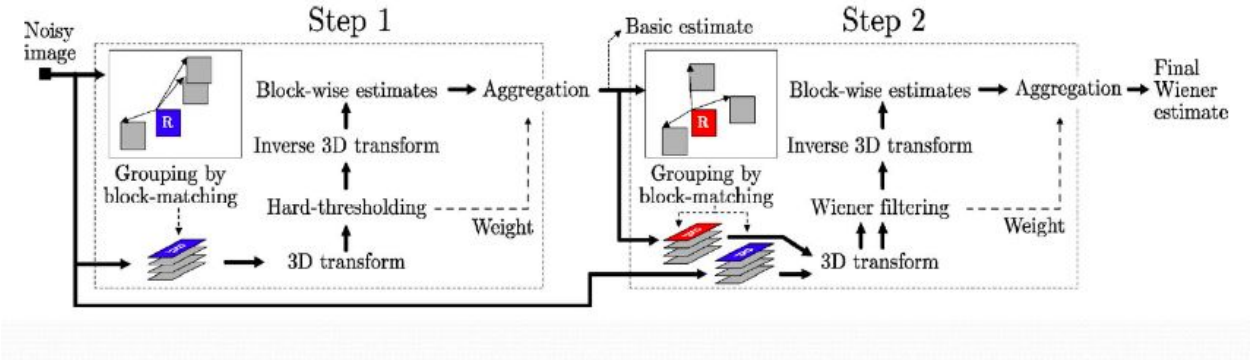
The outputs for all the channels by application of both the filters is shown in section(3.3.a(ii) and (iii)). Later, when dabbing with different sequences of filters discussed in the discussion section, it was figured that Median for R and G, Mean + Median for B 5x5 gives the best PSNR

value.

3.2.b. Principal Component Analysis

PLPCA is the method introduced in this section. The source code from [2] was downloaded and the paper from [3] was referenced from. The parameters - patch size, searching range, and half masking size was varied and the results were recorded in section (3.3.b.(i))

3.2.c. Block Matching and 3D Transform filter (BM3D)



BM3D filter is based on "Collaborative filtering" - it means to stack similar valued blocks with respect to a reference and then applying 3D transform, shrinking of the spectrum for better sparsification and then application of inverse 3D transform. [4]

3.2.c ALGORITHM

The algorithm is referred from [4].

1. The input image is first processed multiple times to retrieve reference blocks.
2. For every block which is similar to the reference block, stack all of the blocks in the form of a 3D array.
3. Collaborative hard thresholding is done by assigning a hard threshold of transform coefficients, and the apply inverse 3D transform to obtain approximated values.
4. Aggregation - approximated value of the actual image is calculated by weighted mean of all the obtained block approximated values. This end step 1 in the above figure.

5. For the obtained block, find more similar blocks within and obtain an approximate from the noisy image.
6. Apply collaborative wiener filtering and apply inverse 3D transform and obtain approximated values.
7. Calculate approximate of the actual image by obtaining weighted average.

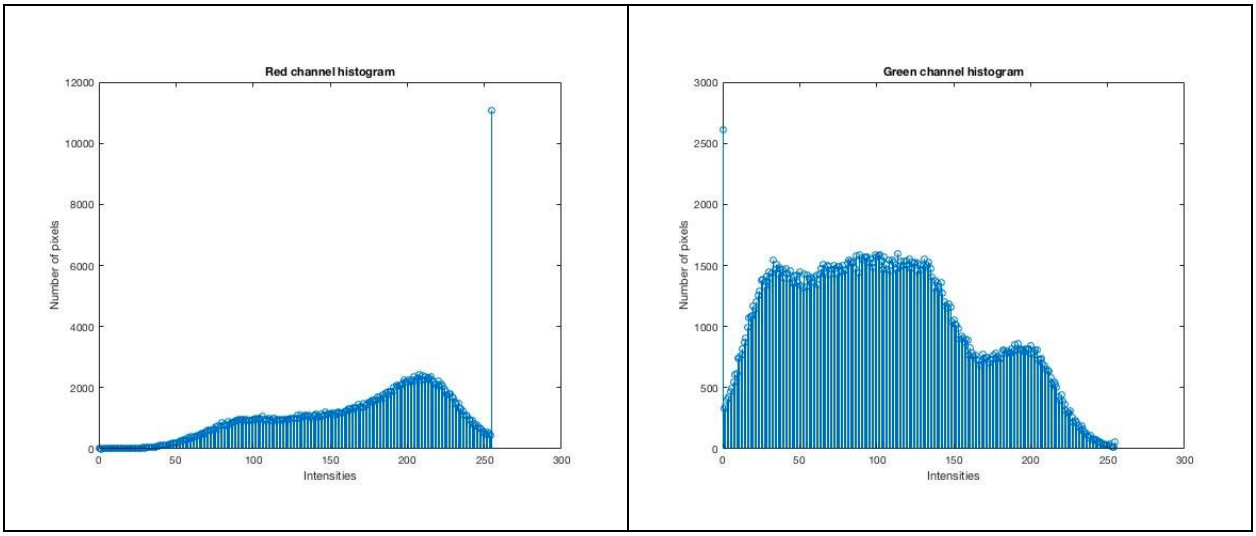
Motivation for step 2 -

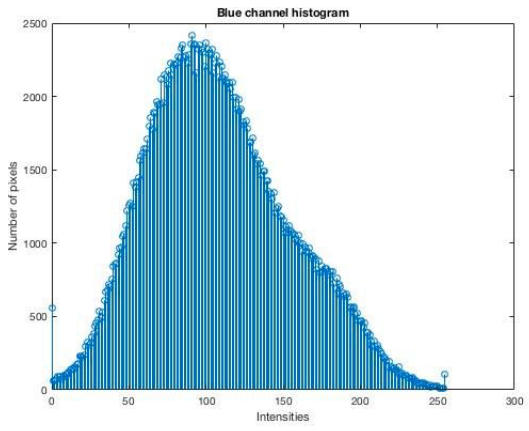
- The basic estimate gives us freedom to evaluate our algorithm as it can enhances grouping by block-matching.

3.3 RESULTS :

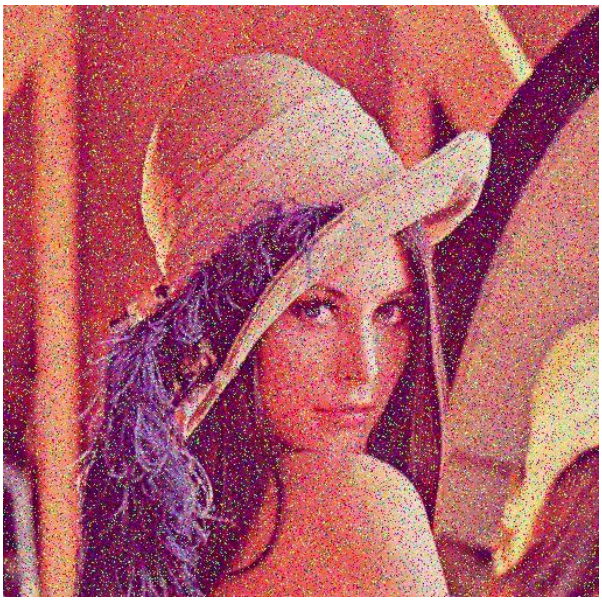
3.3.a. Mix noise in Color image

(i) Noisy Red, Green, and blue histograms of the input noisy image - 'Lena_mixed.raw'





(ii) Noisy Red, Green, and blue channels of the input noisy image - 'Lena_mixed.raw'



Lena_noisy.raw - 512x512x3

Red noisy channel

Green noisy channel

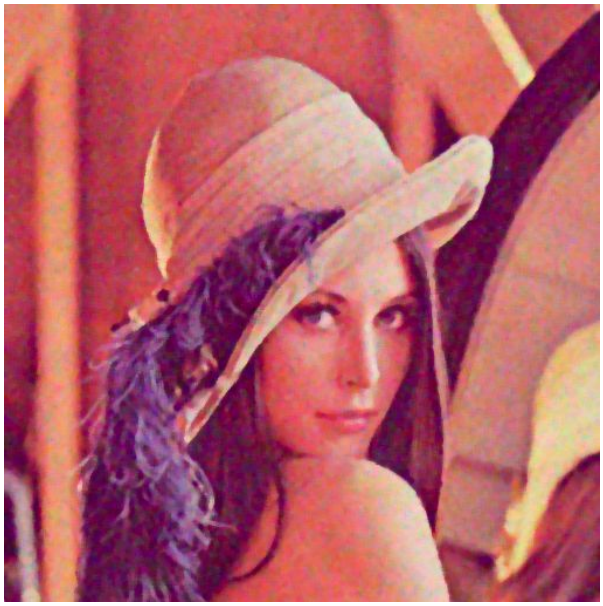
Blue noisy channel

(iii) Median Red, Green, blue, and combined channels of the input noisy image - 'Lena_mixed.raw'

Red channel - after median filter

Green channel - after median filter

Blue channel - after median filter



Combined channels - after median filter

(iv) Mean Red, Green, blue, and combined channels of the input noisy image - 'Lena_mixed.raw'

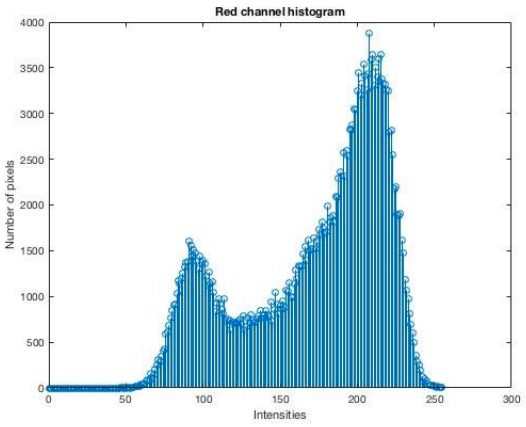
Red channel - after mean filter

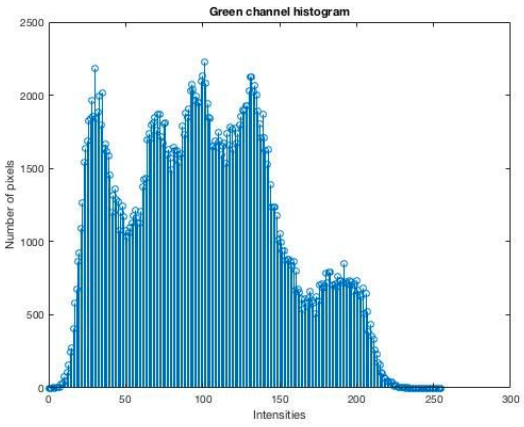
Green channel - after mean filter

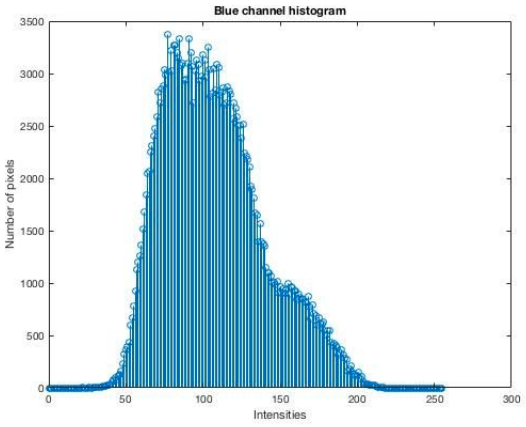
Blue channel - after mean filter

Combined channels - after mean filter

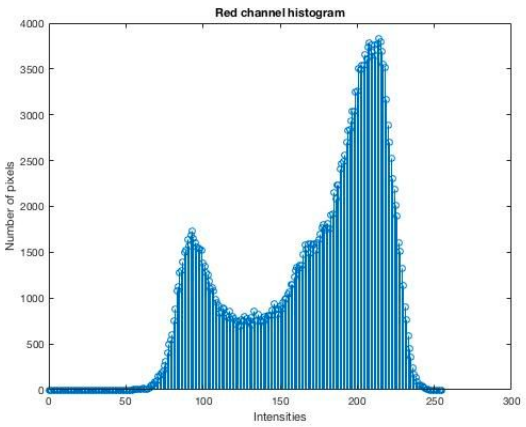
(v)Red, Green, blue histograms of the filtered median noisy image - 'Lena_mixed.raw'

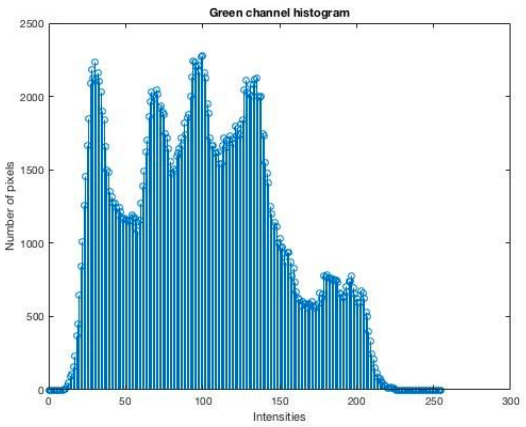


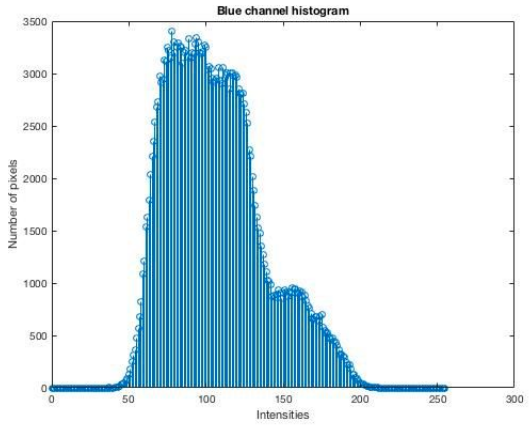




(v)Red, Green, blue histograms of the filtered mean noisy image - 'Lena_mixed.raw'

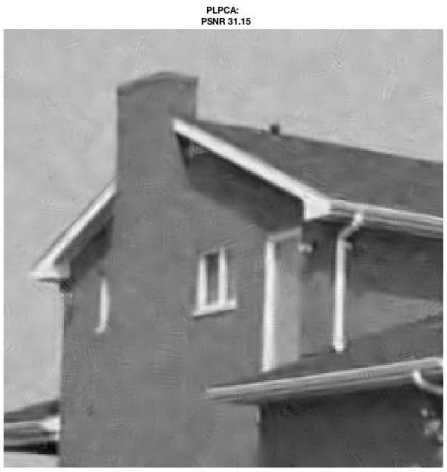




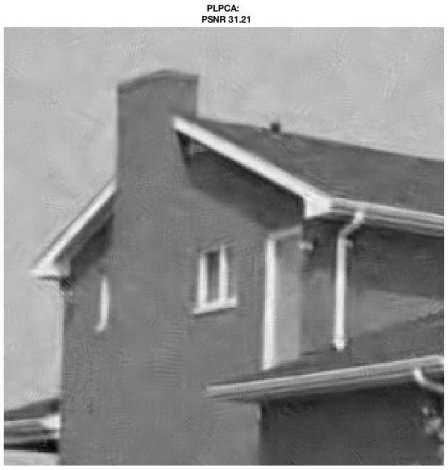
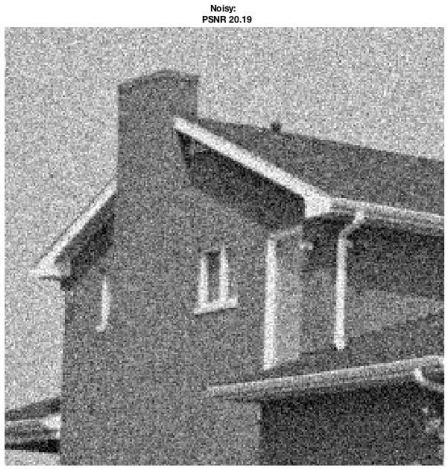
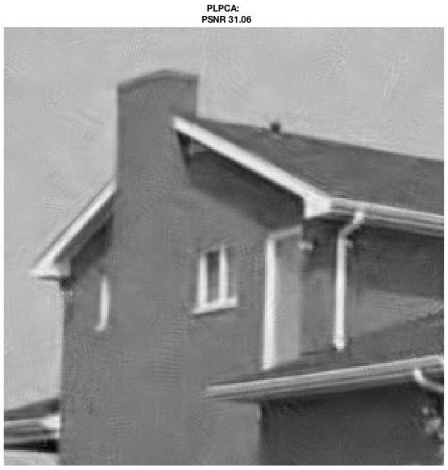


3.3.b. Principal Component Analysis

(i) PCA outputs



WP = 9, hW = 9

 **WP = 9, hW=11**	 **Noise - 20.19dB**
 **WP = 9, hW = 9**	**<u>All the values are specified for sigma = 25</u>**

<u>3.3.c.BM3D</u>

Denoised House$_n$oisy, PSNR: 22.557 dB 	Noisy House$_n$oisy, PSNR: 20.145 dB (sigma: 25) 

3.4 DISCUSSION :

3.4.a. Mix noise in Color image

All the channels have mixed noise type. We can see the presence of salt & pepper(impulse)noise in all the three channels and also random noise in all the 3 channels. So, filtering was performed on individual channels for both noise types.

The filter used to remove salt & pepper noise was outlier (median) filter and then it was cascaded with a mean filter to smoothen the image and to reduce sudden contrast variations.

Effect of different window sizes - if the size of the window is small then the calculated value might not be in coherence with the surrounding pixels as only a small neighborhood was considered. There might be sudden color variations observed in the picture. If the size of the window is too big, there the image might be over smoothened as its value should be closer to a bigger neighborhood. This way contrast is lost.

Filters	Red	Green	Blue	Combined

Only Median	25.443	27.1105	25.6769	26.0168
Only Mean	23.9296	23.8445	25.3791	24.3296
Mean + Median	24.2057	24.1275	25.8787	24.6657
Median + Mean for all channels 5x5	25.4558	26.2457	26.3691	26.0043
Median for R and G, Mean + Median for B 3x3	23.9294	26.675	25.5618	25.2408
Median for R and G, Mean + Median for B 5x5	25.443	27.1105	26.3691	26.2538
Median for R and G, Mean + Median for B 7x7	25.6189	26.3411	26.0649	25.998

Shortcomings - Even though the implementation of filters is easy and not complex it does not provide us with better results. The result picture looks like it has been over-smoothened and as the values are either mean or median, the sharpness in the edges is lost. Contrast is also lost.

We can use other filtering techniques like PCA, Non-Local Means and BM3D to improve the PSNR and to restore the image efficiently.

3.4.b. PCA

- Principal Component analysis is a dimensionality reduction algorithm where principal axes is formed in the direction of maximum variance. PCA is a de-correlation algorithm and after applying PCA reserves the most important components of the image thereby reducing noise.

  Patch size for PCA is the input component. For example, consider a patch size of 8x8 thereby resulting in 64 pixels, but only the pixels which lie in the direction of maximum variance can be considered. For example, 90% of the variance could be well represented with 30 pixels out of the 64.

When PCA is applied to a noisy image, it is observed that the signal lies in the direction of maximum variance and the noisy components constitutes low variance. This was by preserving only the direction of maximum variance, noise is reduced.

Optimum ratio is observed to be - $Signal\ variance \div Noise\ variance = (2.5 - 3)$ (3.4.b.1)

PCA can be applied to images in patches as there might be a lot of repeated or similar data in the image. The axes in the direction of maximum variance (signal) is chosen removing the noisy component. The components to be chosen are size of patches, threshold level, and searching zone. The searching zone is obtained from the patch window size for searching and grabbing pixel values.

- Patch based Local PCA - PCA is a technique which finds the direction in which the maximum variance lies and thus constituting the signal. In patch based Local PCA, multiple patches exhibiting less variance is taken into consideration at once and then PCA is applied to a subset of the similarly varying patches. This makes sure that the dimensionality reduction to only the signal component is happening on with respect to similar features. For example, in an image showcasing a skyline, patches of image are considered to be similar and then PCA is applied as they are similarly varying. If in the same image, there was a unique part showing a UFO, this would not be counted into the similarly varying patches.

  A sliding window can moved across the search size and then the uniform average is calculated for that patch. As sliding windows are allowed to overlap, there are redundancies observed. Hence one pixel can have multiple approximated values and thus need to be reprojected.

Shortcomings are that time complexity is higher and it produces over-smoothened images as there are a lot of redundancy terms.

-

WP (half size of the patch),hW(half size of searching zone)	Ima_patchs - Patch size considered	PSNR
5,9	252x252x25	30.81

9,11	248x248x81	31.21
3,7	254x254x9	29.36

We see that the optimum value is obtained when WP = 9, hW = 11, then PSNR is 31.2dB . Results can be seen in section(3.3.b.(i))

- PCA Vs filter in (a) approach

House_noisy.raw - filtered by part(a) gives PSNR 26.7051dB

Advantages of PCA over filter in (a) :

- PCA takes into consideration similarly varying patches of pixels instead of just its neighboring, like in part (a)
- In part (a), values are directly obtained from its neighborhood - either in the form of median or mean, but in PCA, all the signal components lie in the direction of maximum variance and those components uniform mean is calculated. There might be redundancies, but the values is obtained by uniform averaging.
- Instead of applying cascading of filters, PLPCA gives a better result in one take. This can be compared with the PSNR values of the house image with PCA - 31.21 dB and with filter in (a) - 26.7051dB.
- To improve performance, we can use Wiener filter and cascade with other filters like

NonLocal Means.

3.4.c. Block Matching and 3D Transform filter (BM3D)

- BM3D is both a spatial domain and a frequency filter as it is filtering in reference to a noisy block patch and then 3D transform is being applied. This reduces the high frequency components in the noisy block as the estimate is calculated from the uniform average of all the pixels. This is calculate multiple times and hence we have multiple estimates to the same pixel. This way random frequency components are being eliminated, hence reducing noise.
- For different parameters -

Lambda 3D	NStep	N2	Beta	NStepWeiner	BetaWeiner	PSNR without step 2	PSNR with step 2
2.7	3	16	8	3	8	22.031	22.011
2.5	3	16	8	3	8	22.330	22.447
2.5	2	64	4	2	4	22.556	22.549

REFERENCES :

[1]Sayali Nimkar, Sucheta Shrivastava and Sanal Varghese, (2013).CONTRAST Enhancement and brightness preservation using multidecomposition histogram Equalization Signal & Image Processing : An International Journal (SIPIJ) Vol.4, No.3, June 2013.

[2]http://josephsalmon.eu/code/index_codes.php?page=PatchPCA_denoising

[3]Deledalle, Charles-Alban, Joseph Salmon, and Arnak S. Dalalyan. "Image denoising with patch based PCA: local versus global." BMVC. Vol. 81. 2011.

[4] 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.