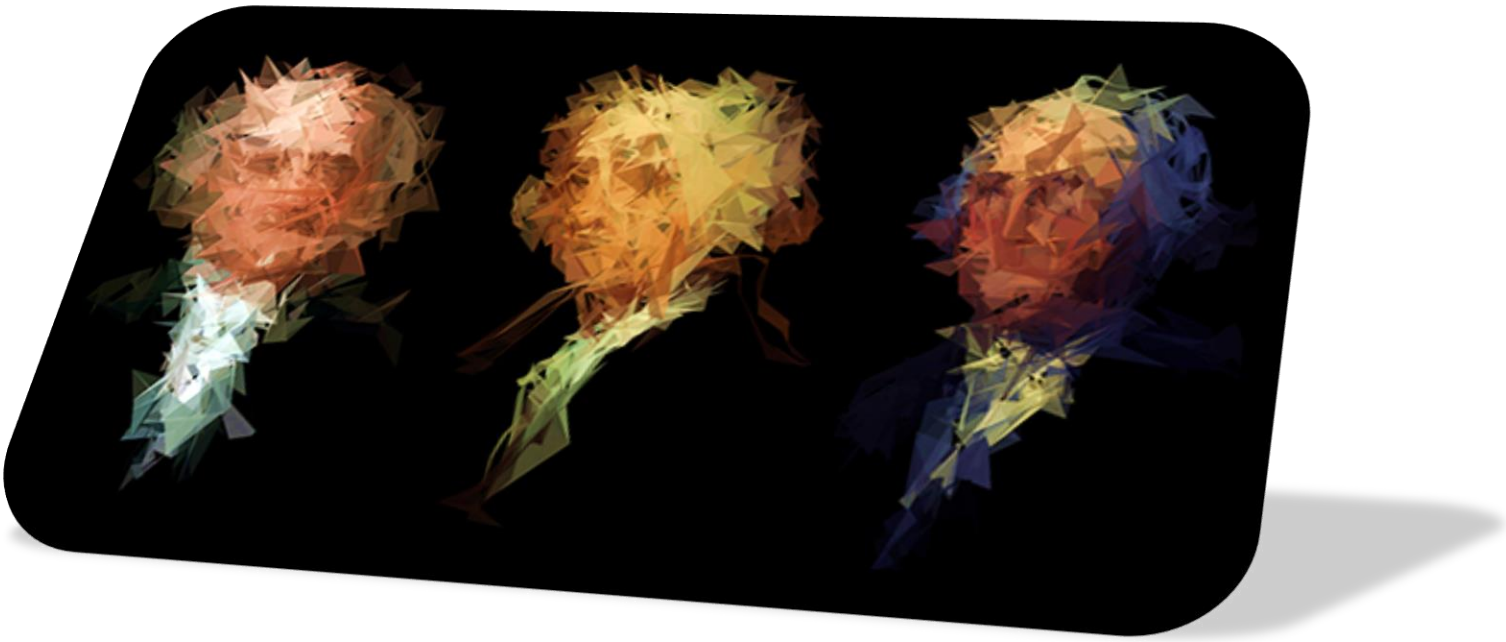
EE 569

HOMEWORK #1



SHIVA SHANMUGAPPA PATRE

spatre@usc.edu

6129918110

1. Problem 1

1.1 Motivation

Image Processing, as the name says is the processing or manipulation of the images in order to get a better quality image or an enhanced version of the same image. There are several uses of image manipulation such as down-sampling or down sizing an image helps save storage space and thus saving on the costs.

Color Space Transformation is one of the basic image manipulation techniques. Color spaces are nothing but a representation of the range of colors of the visible spectrum. We can use the color space to achieve different colors in digital images by mixing various components of the color space. There are many color spaces for our benefit such as RGB, CMY(K), HSL, YCrCb etc. Each of these spaces have their own set of advantages and can be used in our image processing problems suitably. It makes sense to have a transformation mechanism so that we can exploit these advantages for our various image processing challenges.

To support this statement, we shall see the advantages of having CMY in printing. If we need a particular color, we need to determine what colors have to be injected. CMY uses subtractive color mixing and this is advantageous to be used in the printers.

In this problem, we explore conversion from color(RGB) to Grayscale and RGB to CMY colorspace using certain interesting techniques.

Also, one more interesting and a basic image processing application is image resizing. Sometimes we may require to rescale the given image by either shrinking or expanding it for various tasks. Image upsizing or upsampling is an interesting case where we expand a given image. In this case there will be an introduction of noise as there are more empty(zero pixels) than the image information in the expanded image and hence the clarity reduces. So we need to fill in these missing pixels somehow.

Therefore, in this problem we resize an image by using bilinear interpolation technique. It basically fills the empty pixels by considering the neighboring pixels, which we shall see more of below.

So let us discuss the approach to be taken to implement the above methods in detail. Let the fun begin.

1.2 Approach

There are three parts in this section. The Color to Grayscale conversion is described in section 1.2.1, RGB to CMY is described in 1.2.2 and the image resizing by bilinear interpolation in 1.2.3.

1.2.1 Color to Grayscale Conversion

The grayscale images are good for human perception because of the luminance content in them. So it is sometimes necessary to convert to grayscale. In the given problem we have to convert to grayscale using three methods namely the lightness method, the average method and the luminosity method. This is very straightforward. We first read a RAW image with RGB values and store it in a 3D array. To convert to grayscale using lightness method, we calculate the maximum value among R,G and B and the minimum value among the same R,G and B intensities of a particular pixel. We then take the average of these two values by summing them and dividing by 2.

Lightness method : (max(R,G,B) + min(R,G,B))/2

The average method is the simplest which just considers the average of R,G, and B at each pixel location as follows.

Average Method : (R + G + B)/3

The luminosity method is a bit sophisticated in the sense it gives weight to each of R,G and B. This considers human perception for weighting. Since humans are more sensitive to green than red and blue, more weight is given to green than the rest. The formula is as follows.

Luminosity Method : 0.21*R+0.72*G+0.07*B

1.2.2 CMY(K) Color Space

The CMY is basically the inversion of RGB. So first we extract the R, G and B values from the 3D array and store it in a 1D array separately. Then we need to divide each intensity value by 255, to normalize the values. We then subtract the obtained value from 1 and assign it to C, M and Y respectively as follows.

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

We then multiple the values by 255 to get back the intensities. These values are stored in a 2D array which then goes as the output. We thus get Cyan, Magenta and Yellow channels from the Red, Green and Blue channels.

1.2.3 Image Resizing by Bilinear Transformation

If we have to scale up or up sample an image using bilinear transformation, we need a mapping to map the pixels in the resized image to that of the original image. This new resized image has holes or missing pixels as not all the pixels from the original image can be mapped to the new image. There are various methods to do this such as bicubic interpolation, nearest neighbor interpolation, Bilinear interpolation etc. In this problem, we use the bilinear interpolation to fill these holes with pixel intensities.

The simple idea behind this terrifying method is described in the following picture.

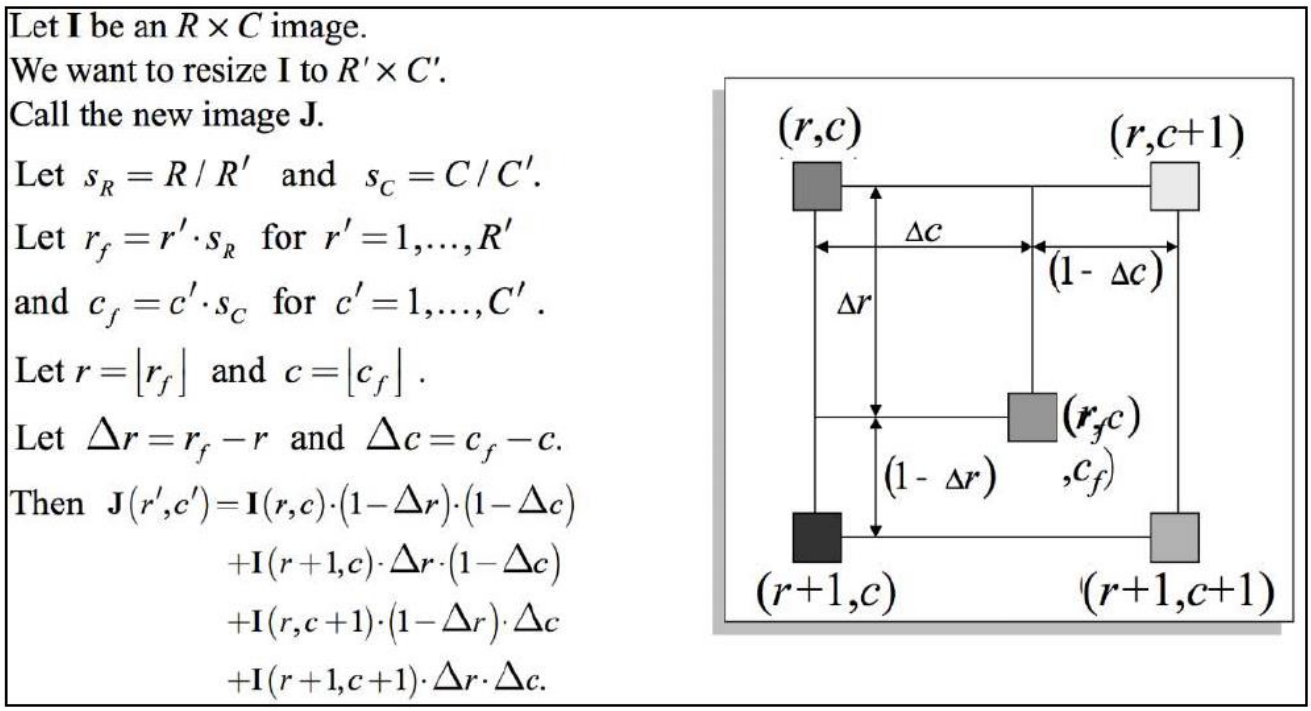


Image resizing by bilinear interpolation

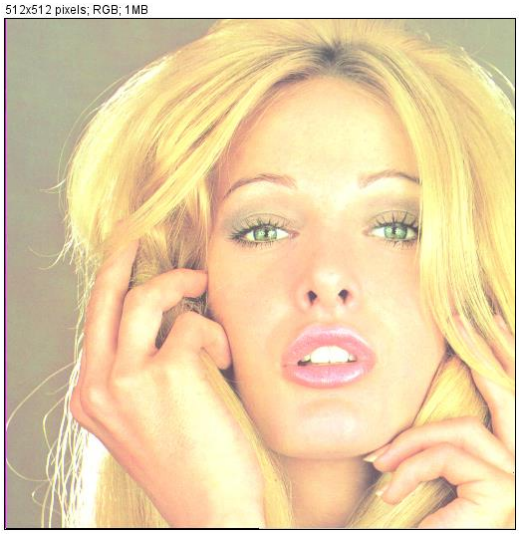
Source: https://i.stack.imgur.com/t7z2N.png

Sr and Sc are nothing but the ratio of height and width to be used for resizing. rf and rc are the new co-ordinates for the pixel to move to r' and c'. J(r',c') is the new image pixel location. (r,c) is the co-ordinate location of the input image, which are used in various combinations , so as to facilitate bilinear interpolation on them to get the missing pixel. From the above formula we can see how four neighboring pixels can be used to get the value of the new pixel. Each pixel in the original image is multiplied by weighted parameters Δr and Δc. This gives the distance or spread of the pixels from one another.

I implemented the above formula directly in C++ to get successful results.

1.3 Results

- Figure 1 represents the solution to problem 1a part 1
- Figure 2 represents the solution to problem 1a part 2
- Figure 3 represents the solution to problem 1a part 2
- Figure 4 represents the solution to problem 1b

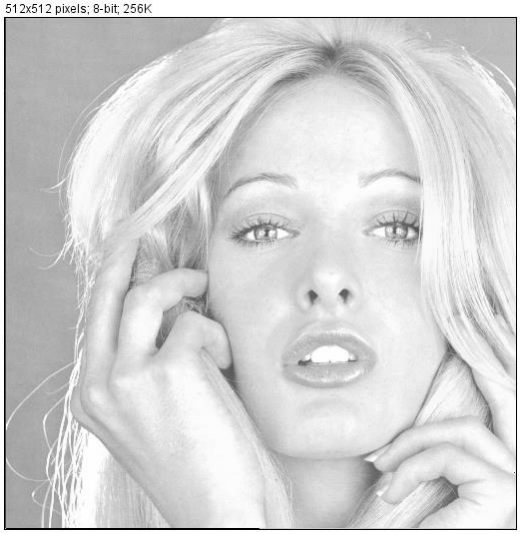


Original Image of Tiffany

(a) Lightness Method

(b) Average Method

512x512 pixels; 8-bit; 256K



(b) Luminosity Method

Figure 1: Grayscale Tiffany Image

854x480 pixels; RGB; 1.6MB

Original Image of Bear

854x480 pixels; 8-bit; 400K

(a)Cyan Channel

854x480 pixels; 8-bit; 400K

(b)Magenta Channel

854x480 pixels; 8-bit; 400K

(c)Yellow Channel

Figure 2: CMY channels for Bear

854x480 pixels; RGB; 1.6MB

Original Image of Dance

854x480 pixels; 8-bit; 400K

(a) Cyan Channel

854x480 pixels; 8-bit; 400K

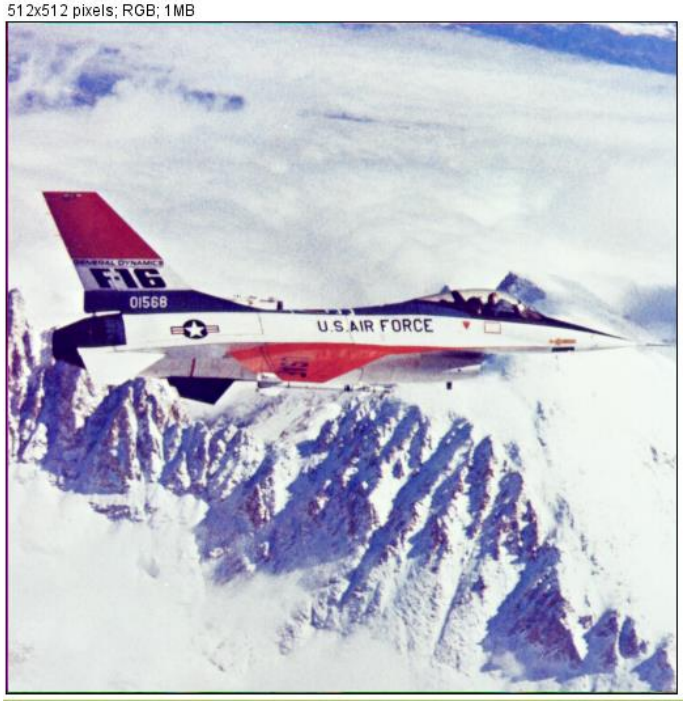
(b) Magenta Channel

854x480 pixels; 8-bit; 400K

(c) Yellow Channel

Figure 3: CMY channels for Dance

512x512 pixels; RGB; 1MB



Airplane 512x512

Airplane 650x650

Figure 4: Image Resizing by Bilinear Interpolation

1.4 Discussion

- **Color to Grayscale conversion:** As can be seen in **figure 1(a,b,c)** the original Tiffany image is converted into grayscale by using the lightness method, average method and the luminosity method. The pixel intensities range from 0 to 255 and is a byte long. 0 represents black(when no electrical signal is passed) and 255 represents white(when the electrical signal is passed in full). I feel, the **luminosity** method works best overall as it gives weight to the most perceptible color(green) and hence gets a better contrast and viewability to the observer. Whereas the other methods just average the values differently in some fashion.
- **CMY(K) Color Space:** The gray level intensity images is given in **figure 2(a,b,c)** and **figure 3(a,b,c)** and they represent the Cyan, Magenta and Yellow values for each channel. The CMY along with K is used for printing because of subtractive color mixing. It can be seen from the above images that CMYK adds black to the mix.
- **Image resizing by bilinear interpolation:** There is always some loss in image data in the estimation of the values of the new pixels as this method just gives an approximation and not a definite value. This technique gives sort of blurry image as it just considers the neighboring pixels for interpolation. **Figure 4** shows the resized image to 650x650, and a little amount of blurriness can be seen in the image. But as we increase this size to say 1024x1024, more blurriness can be seen in the resulting image. To get better results we need to use adaptive interpolation techniques.

2. Problem 2

2.1 Motivation

Sometimes the pictures which we click are under dull lighting and will result in poor image quality and the subject might not get illuminated enough as it must. So we can use Image processing to enhance the contrast and hence the quality of the subject in the image.

Usually such poorly illuminated images have the pixel values distributed in a narrow range. So we can redistribute these pixels in a wider range and thus gets a brighter or better contrast image. In this problem, we shall implement the transfer function based histogram equalization method and cumulative probability based histogram equalization method.

In the transfer function based histogram equalization method, we need to basically find a transfer function that maps the original pixel intensity distributions to the desired distribution. To achieve this and to get a proper wide range we need to map the original distribution to a uniform distribution.

In the cumulative probability based histogram equalization method, we use a concept called bucket filling algorithm. According to this method, we distribute all the pixels equally in 256 bins. The pixel intensity is nothing but the corresponding bin number. This method will distribute the pixels somewhat uniformly.

In this problem, we will implement both the methods discussed above and we can see the results as such. Histogram equalization is one powerful tool to enhance the contrast of dull/darker and too bright images.

Another interesting problem is the oil painting effect. The normal looking image can be transformed to look like an oil painting. We basically need to reduce the colors than was in the original image and distribute the pixels in a particular fashion to achieve this effect.

We Quantize the original image to have only 64 colors. We then consider an NxN neighborhood and pick the most frequent color in that neighborhood and apply that pixel intensity to the output image pixel. Thus we get a beautiful oil painting effect. In this problem we shall implement this method and see the effects for various values of N.

Creating Film special effect is a creative way of depicting the image for viewer satisfaction. In this problem, we analyze a film effect filtered image and apply the algorithm after learning the method to a new image to get the same film effect to our image.

Let's get the ball rolling!

2.2 Approach

2.2.1 Histogram equalization method

We shall get first discuss the implementation of the transfer function based histogram equalization method and then discuss the cumulative probability based histogram equalization method.

Transfer function based: The main purpose of this method is to redistribute the pixels evenly or uniformly to enhance the contrast of the image. To achieve this result, we first map the pixel intensities of the output image using a transfer function such that the distribution is uniform.

The Probability density function of the pixel intensities is calculated. This probability of occurrence of the pixel is stored in an array. The pdf of a pixel x is given by the following.

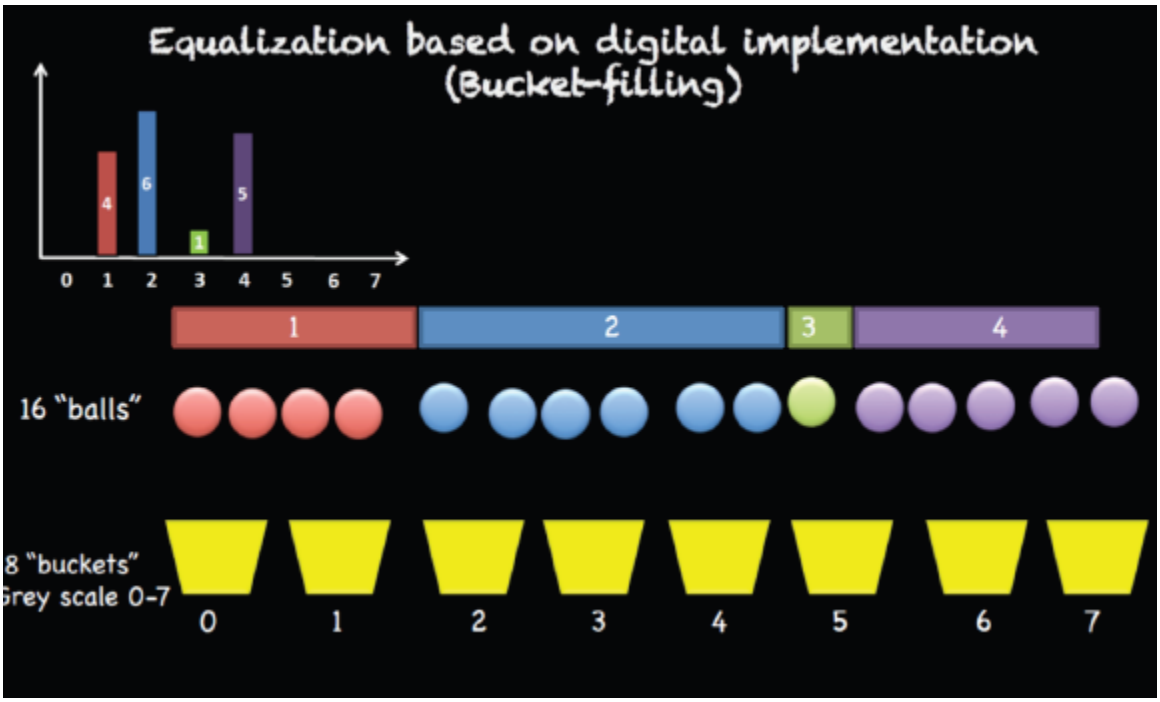
pdf[x]= (Total number of pixels with pixel intensity x)/Total number of pixels in the image

Then we use the above values to calculate the CDF(cumulative density function). The CDF values thus obtained gives us the transfer function which we are looking for. We now need to map the pixel intensities to the CDF distribution. This is done by.

$$J(i,j) = CDF[J(i,j)] * 255$$

J(i,j) gives the pixel intensities of the ith row and jth column. The CDF values lie in the interval [0,1]. So to renormalize, we multiply the CDF by 255 to get back the new pixel intensities which will be used to display as the new image.

Cumulative Probability based: In the previous approach we get certain contours in the histogram. These are not desirable. So We use the Cumulative probability based histogram equalization metod. In this approach we follow something called as the **bucket filling algorithm** as shown below.



source: Discussion Slides

We basically need to redistribute the pixels equally in 256 bins. This way we get an equalized output.

We first calculate the bin size by dividing the number of pixels by 256. For the given Desk image, this value is around 471. We choose 467 to get an even distribution based on trial and error. (This won't make a visible difference to the human perception). We initialize a counter to 467 and also have a flag initialized to 0 for each pixel location. We also have a color value initialized to 0. We then check if the value of each pixel location is equal to the pixel intensity value and also check if the flag is 0. If so, then we assign the value of color to the image at that location. We then assign the flag value at that location to 1. We decrement the counter. We also check if the counter reaches 0. If it reaches 0, the counter value is reinitialized to 467 and the color value is incremented. What this does is, it finds the first 467 0's and keeps them as is and changes the remaining 0's to 1's and so on. If the first 467 values is not all 0's then it converts the 1's to accommodate the 0's. This way all the pixel intensities are being equally distributed. We then display the resulting image as the output and the image can be seen to be enhanced.

Also the plots of all the histograms are plotted using Excel, by extracting the corresponding values using cout command in C++ and then copy-pasting it to a csv file.

2.2.2 Image Filtering – Creating Oil Painting Effect

This method is used to create an oil painting effect on the original image.

We first calculate the histogram values of the RGB channels of the original image. We the how many pixels are of each intensities from this. We then need to reduce the color of the image by quantization. We will need only 64 colors. But we have 256 colors in the original image. So we divide the number of pixels in the image by 4 to get the bin size. Then for each channel, we keep adding the number of pixels starting from the 0'th intensity until the binsize is reached. We then store the value of the starting pixel and the ending pixel which results in a total value just less than the binsize(as the pixels are not evenly distributed). We then calculate a weighted mean for values between the starting point and endpoint until 256 pixel intensities are considered. Thus we get 4 values for each channel to represent the intensity values. We then use these values to replace all the existing pixel values based on the range from where the midpoint or mean is considered.

We then choose a window size N, and pad the original image accordingly. Say N=3, we pad one extra row in the top and bottom and one extra column to the left and right. Once the padding is done, we implement four for loops to traverse a window size of NxN over the entire image. We then compare each the NxN pixels with the midpoint values which we got in the quantization and increment the corresponding array location. We then find the max value of the array and consider the corresponding index and choose the value of the index from the midpoint array. We then assign this value to the center pixel of the NxN window.

We then convert this expanded image due to padding to the original image. This is the output for oil painting and we check for different window sizes.

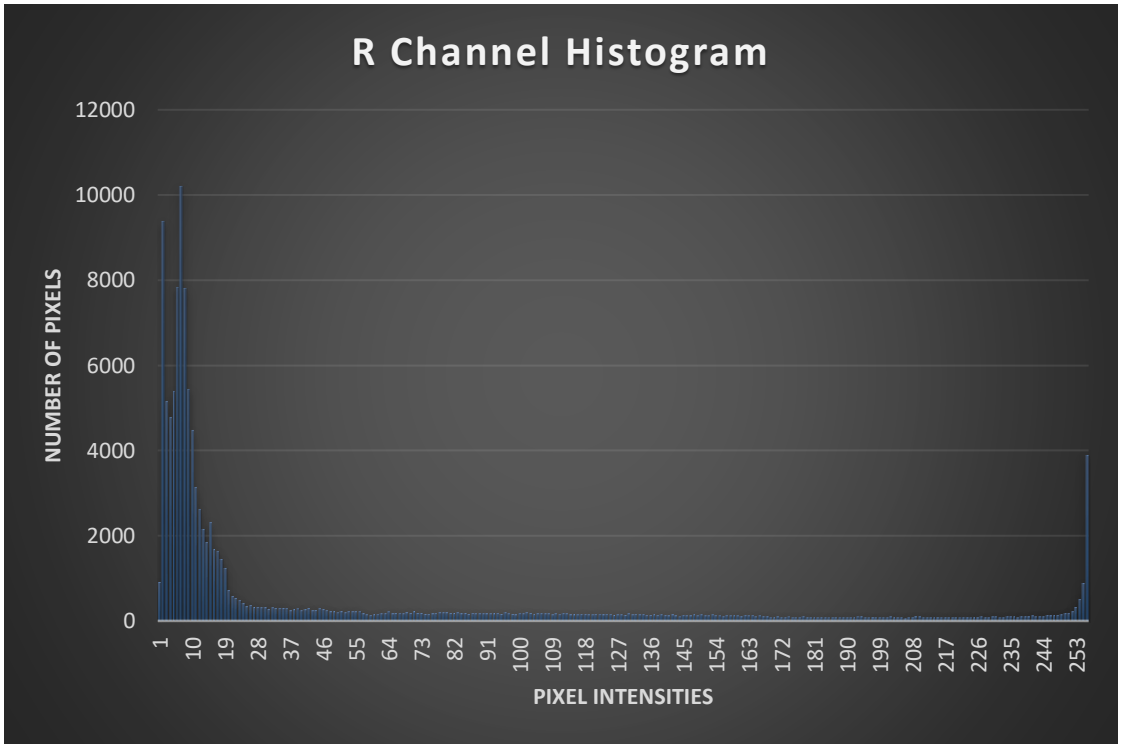
2.2.3 Image Filtering – Creating Film Special Effect

This method is used to create a special filter to achieve a film special effect. As can be seen from the image. The final image looks like a mirror image of the original with some effects. So we first flip the original image and get a mirrored image. Next we convert the image from RGB to CMY as described previously to achieve color inversion. To find out the filter effect, we need to analyze the histogram of the film effect image which is given to us. So we basically need to Shrink the histogram of the original image and shifting it by a certain value to achieve this effect.

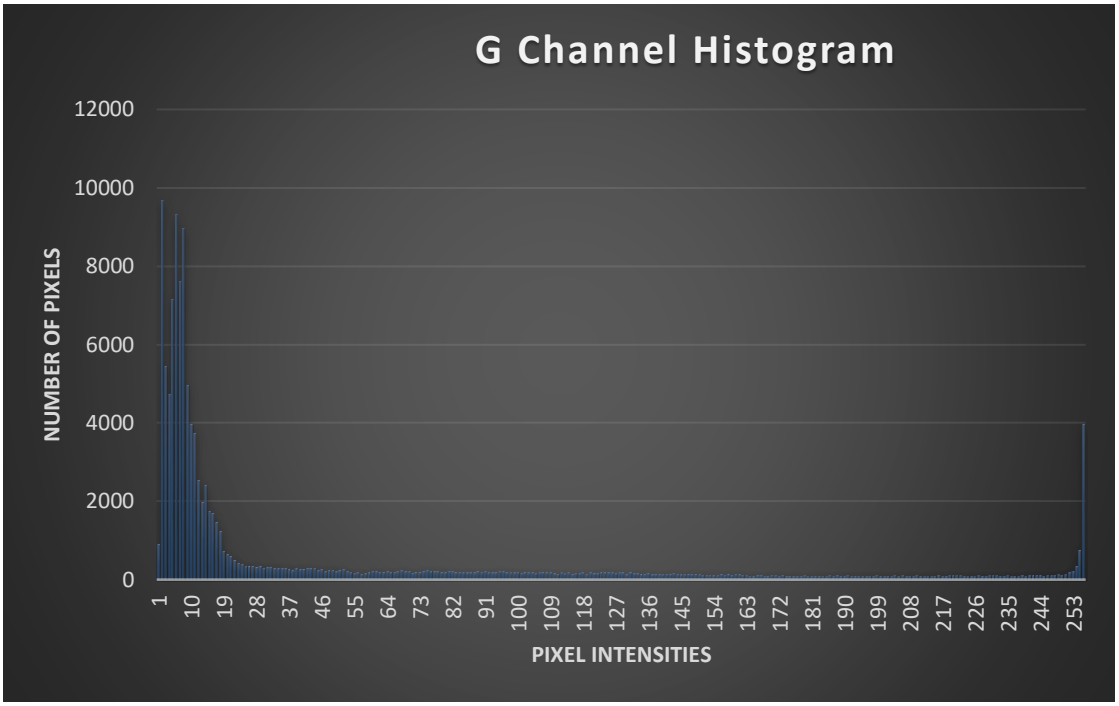
We first calculate the max and min value of the histogram where it is densely populated and subtract them (max-min) for each of R,G, and B channels. We do this by opening the image in ImageJ software and analyzing the histogram directly. We then multiply this value to the pixel value of the resulting image. And divide by 255(max-min, for an 8bit image this is 255-0) value of the Original image histogram to shrink the histogram. We then add the min value to this result to shift the histogram to obtain the film special effect.

2.3 Results

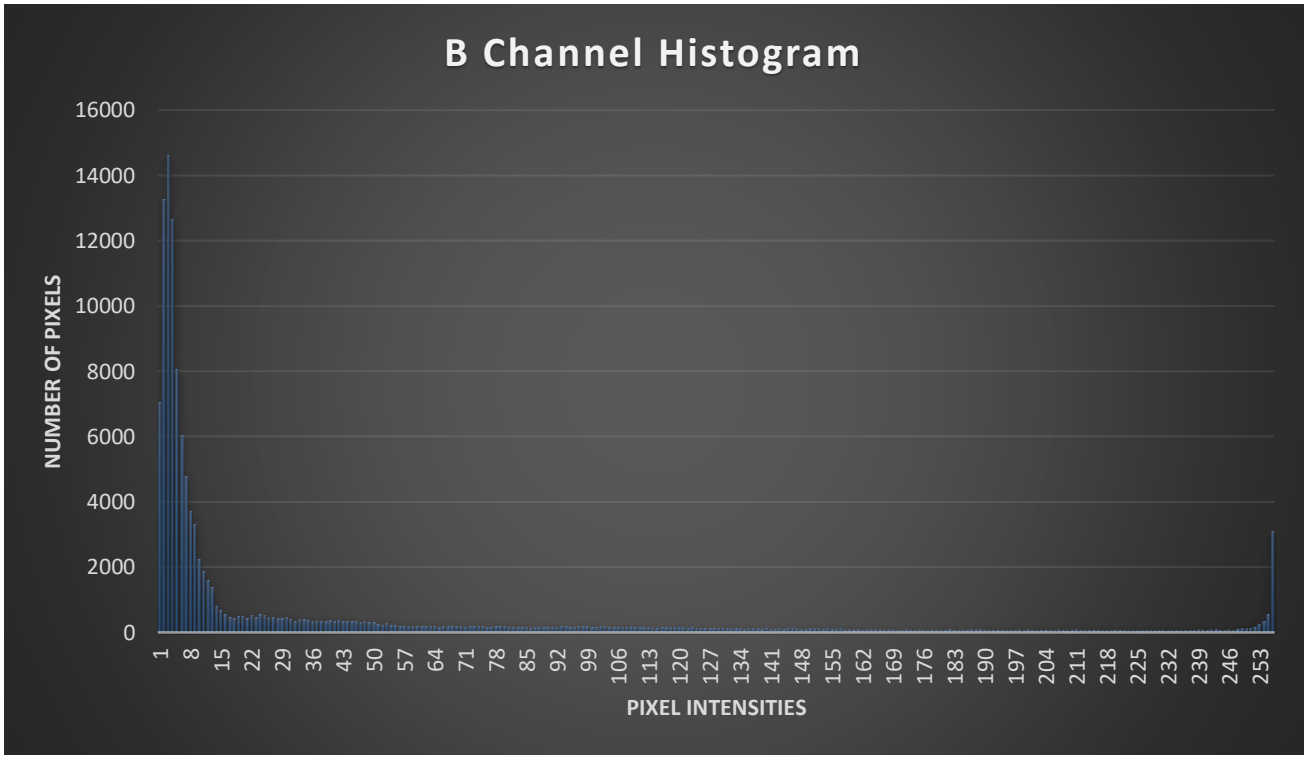
Original Desk Image



(a) R Channel Histogram



(b) G Channel Histogram



(c) B Channel Histogram

Figure 5: R,G and B Channel histogram values

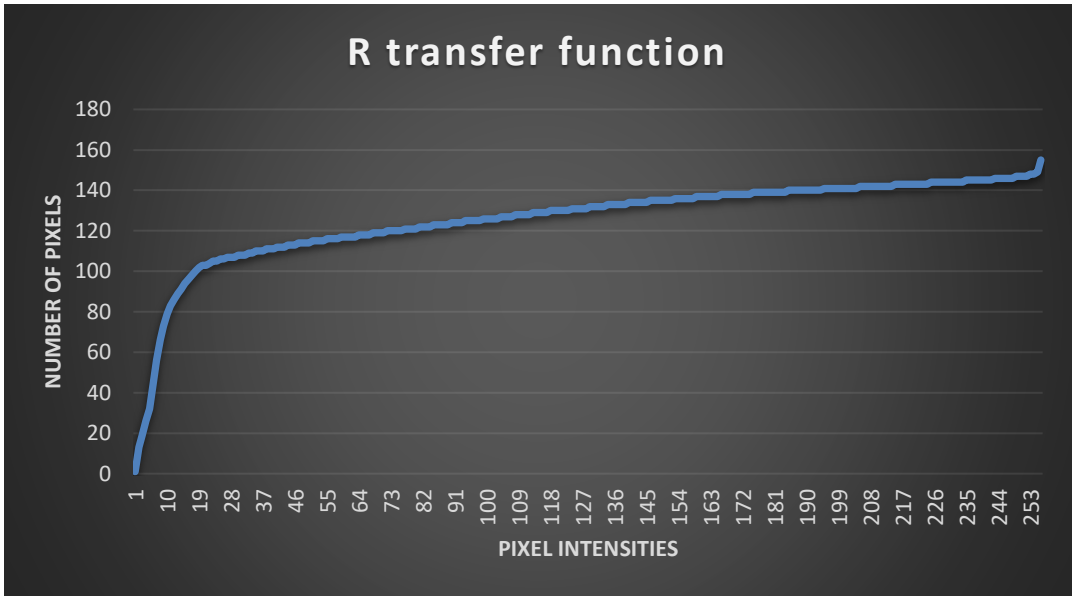
400x300 pixels; RGB; 469K

(a) The transfer-function-based Desk output

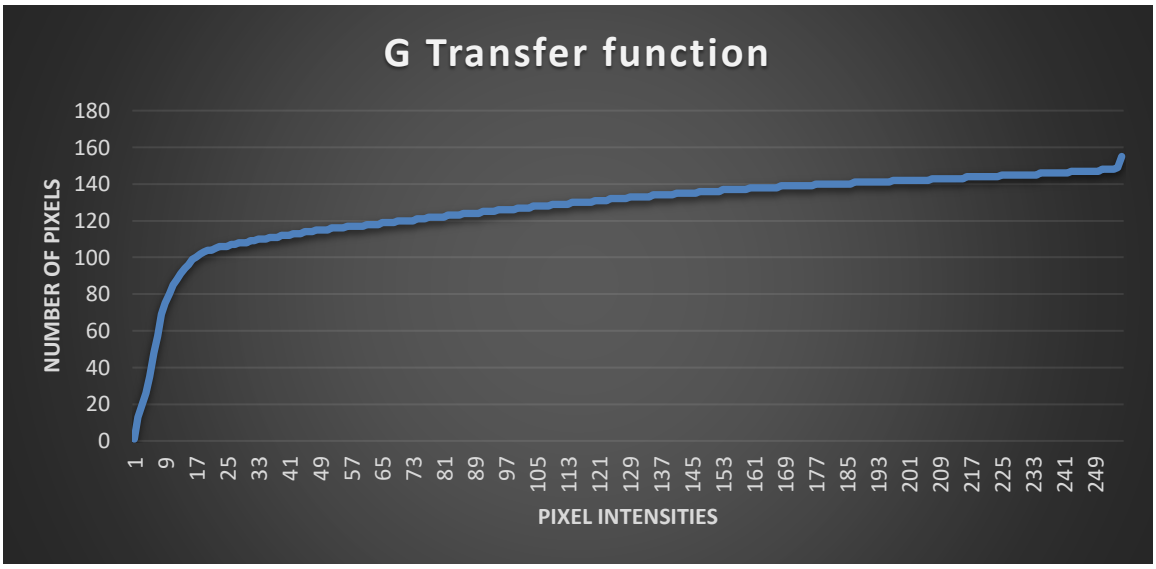
400x300 pixels; RGB; 469K

(b) The cumulative-probability-based Desk output

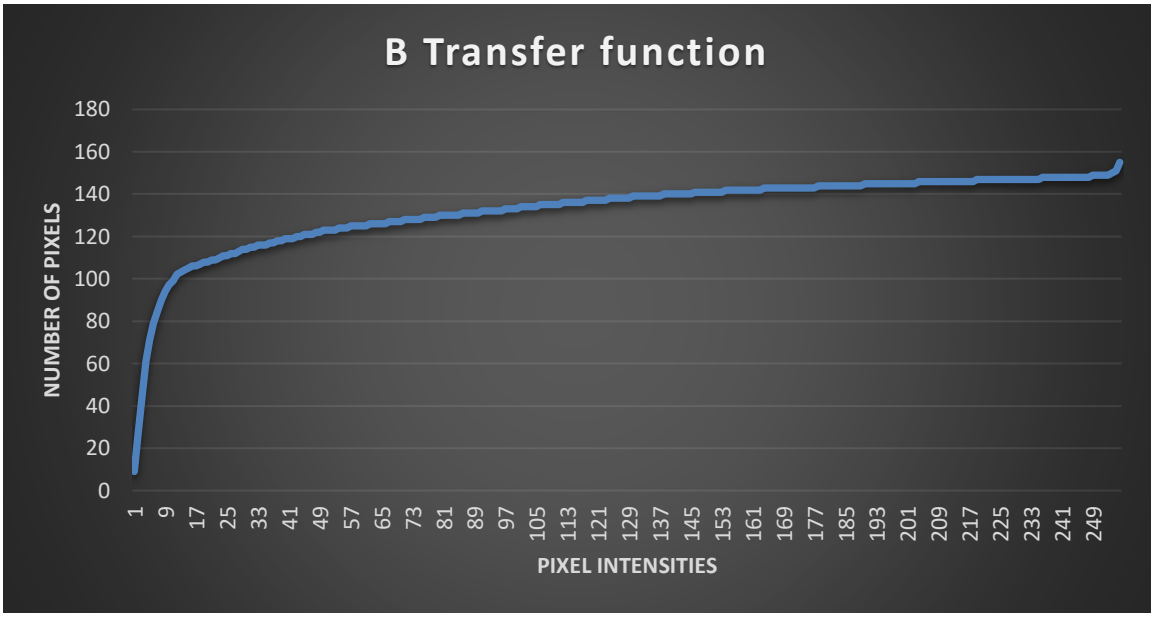
Figure 6: Histogram Equalization of Desk



(a) R Channel Transfer Function

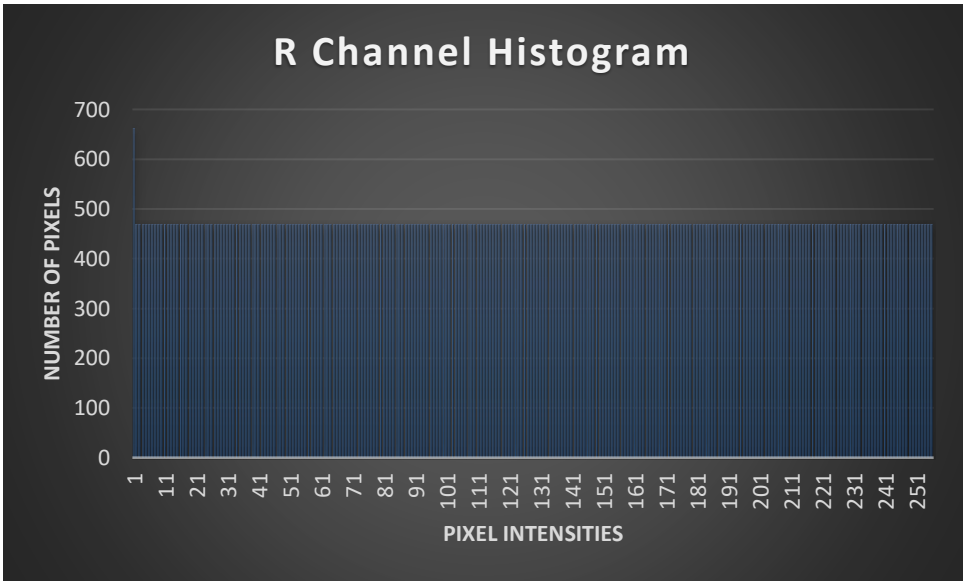


(b) G Channel Transfer Function

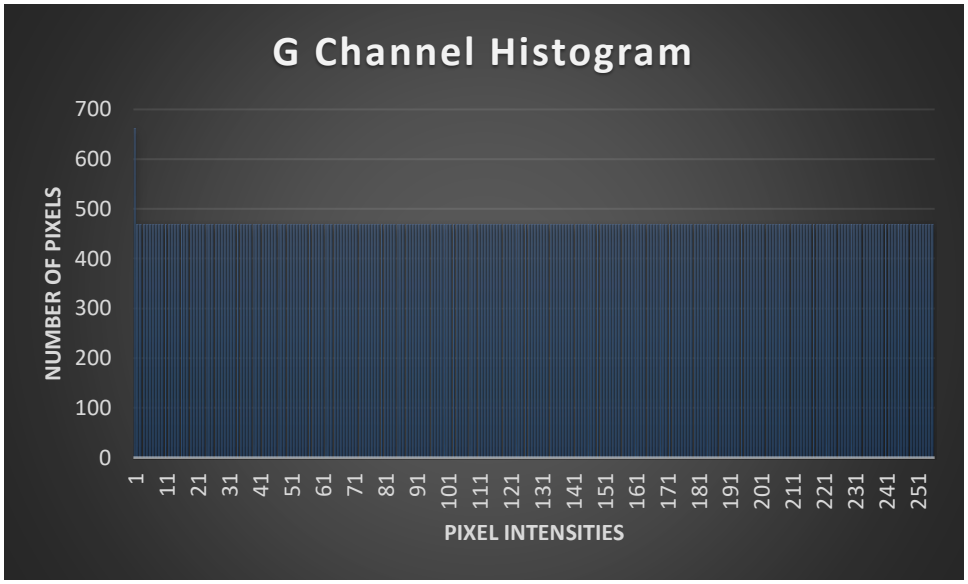


(c) B Channel Transfer Function

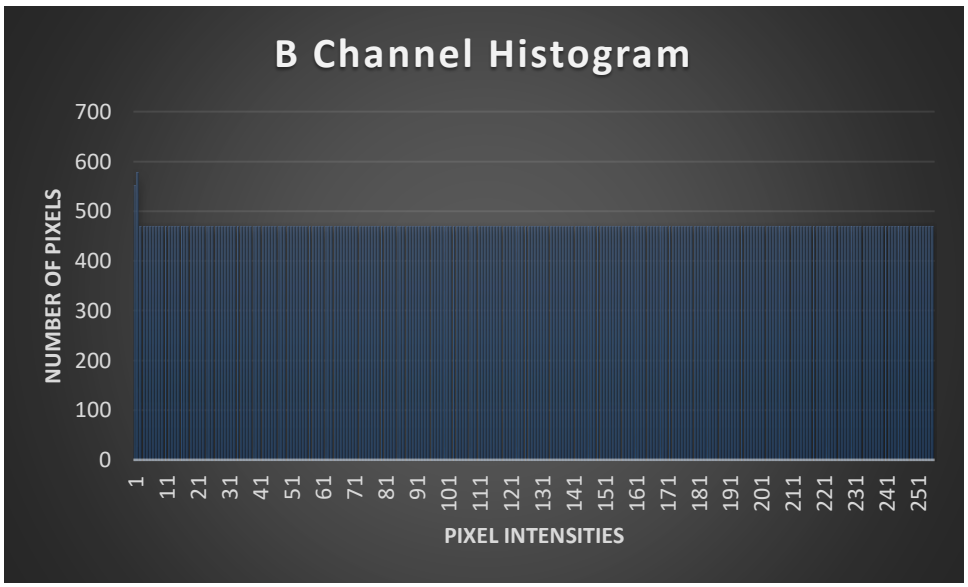
Figure 7: R, G and B Channel Transfer Function



(a) R Channel Histogram

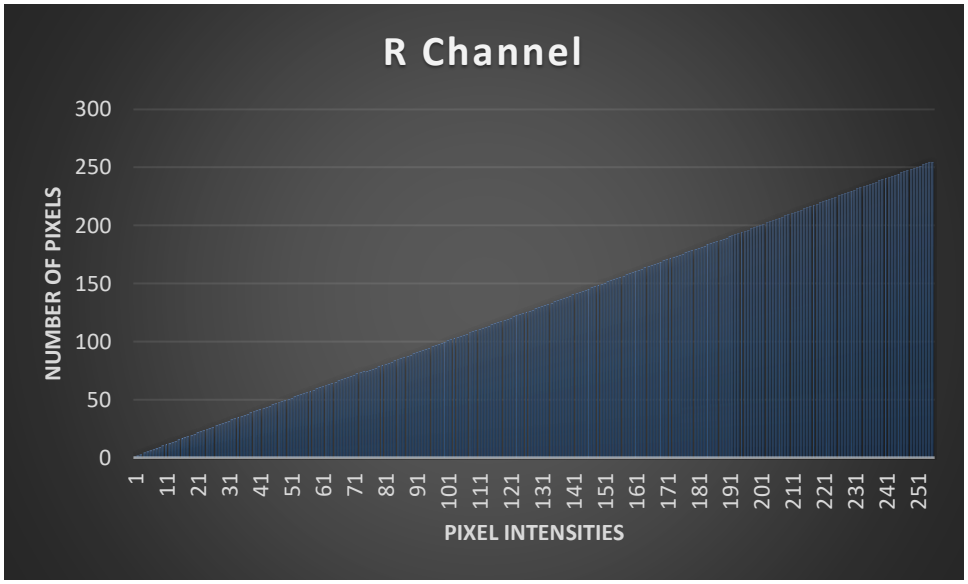


(b) G Channel histogram

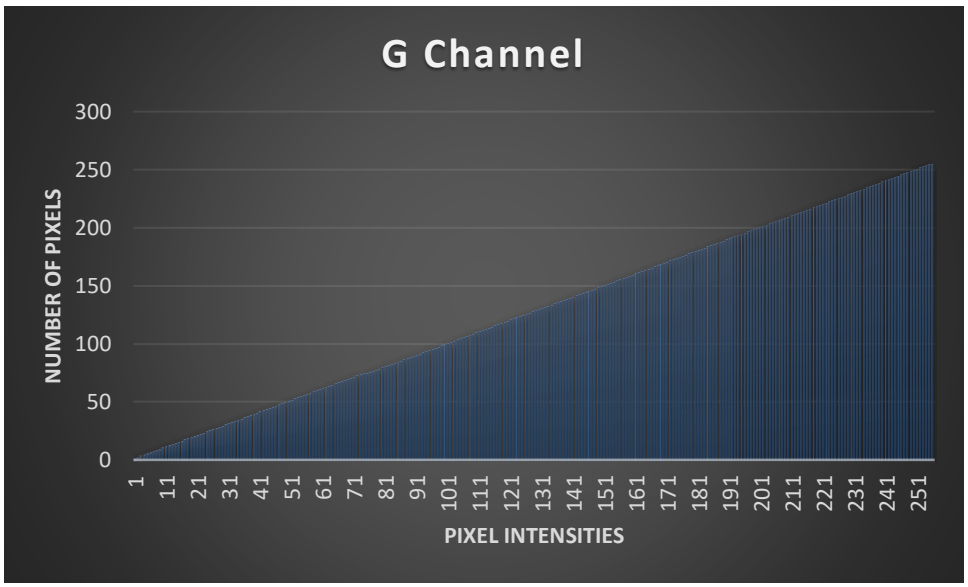


(c) B Channel histogram

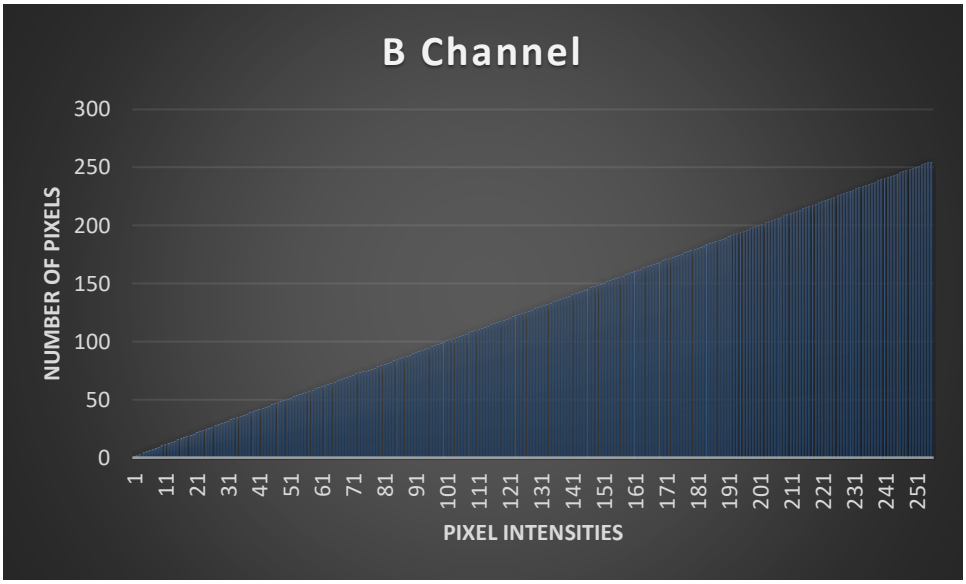
Figure Z: Method B bucket filled histogram



(a) R-Cumulative Histogram



(b) G-Cumulative Histogram



(c) B-Cumulative Histogram

Figure X: RGB Cumulative Histograms

Original Image of Star Wars

600x338 pixels; RGB; 792K

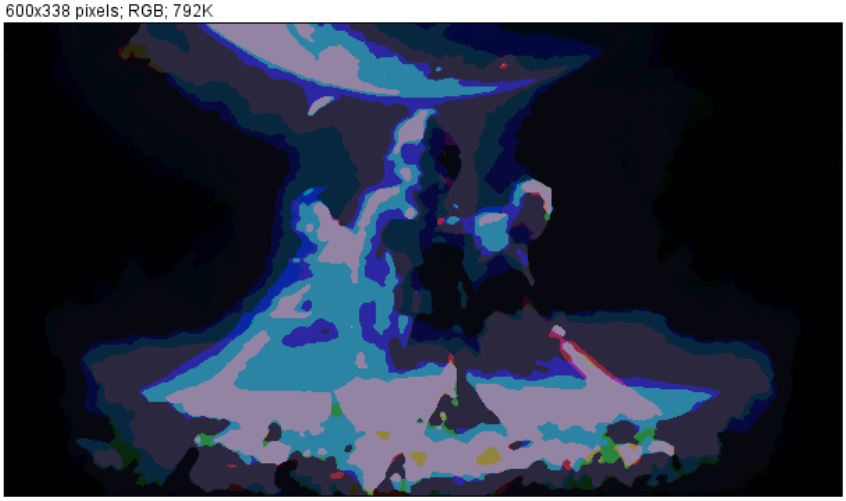
(a) 64 Colors Star Wars Image

Figure 7: Reduced Color Image of Star Wars

600x338 pixels; RGB; 792K

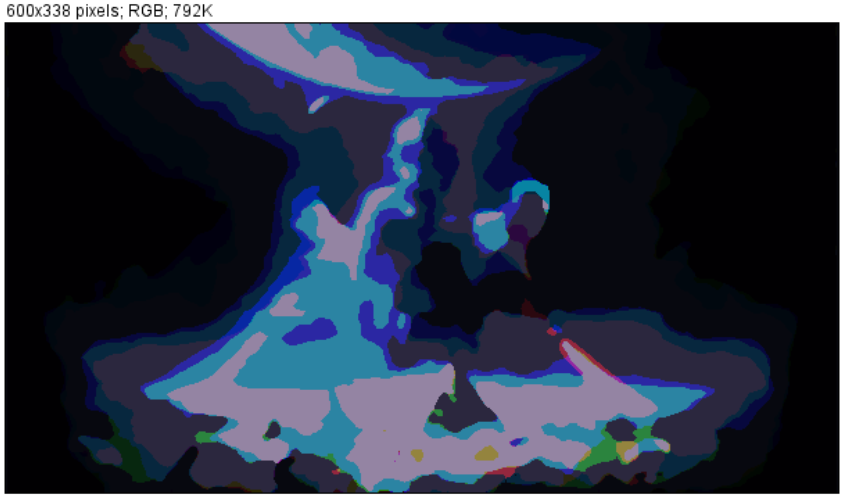
(a) N=3

600x338 pixels; RGB; 792K



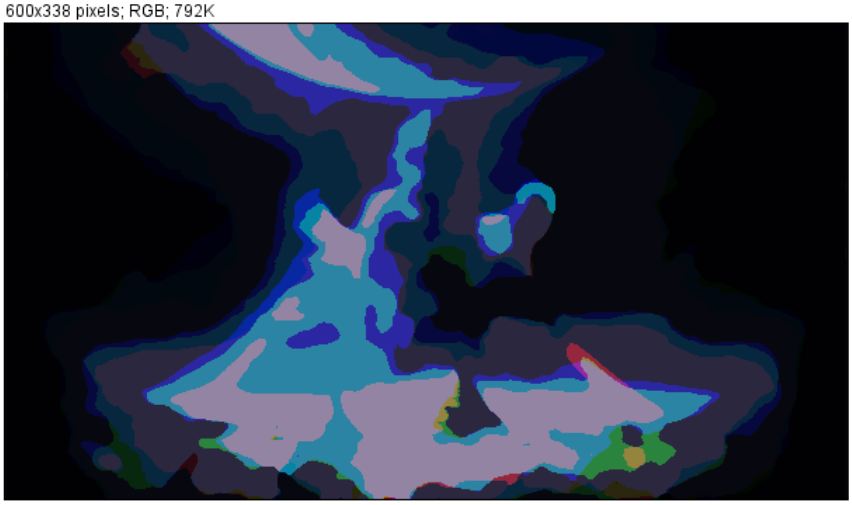
(b) N=5

600x338 pixels; RGB; 792K



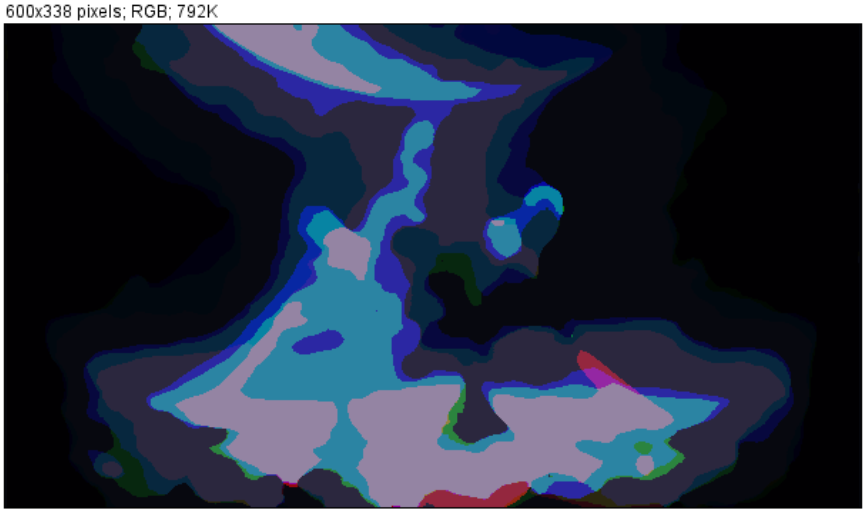
(c) N=7

600x338 pixels; RGB; 792K



(d) N=9

600x338 pixels; RGB; 792K



(e) N=11

Figure 8: Oil painting of Star Wars

600x338 pixels; RGB; 792K

Figure 9: 512 Colors Star Wars Image

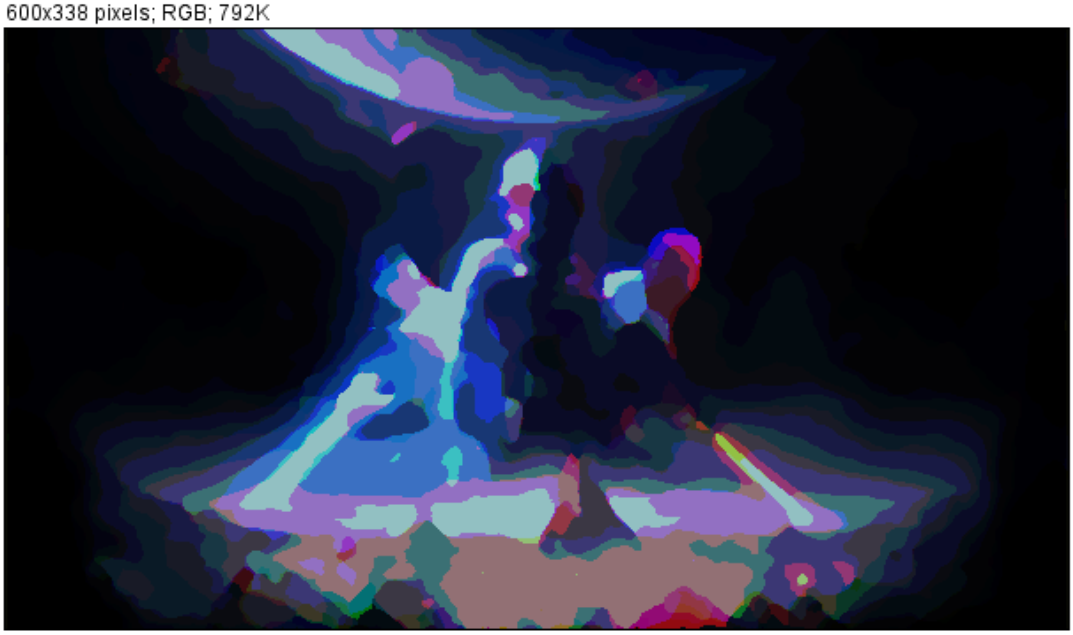
600x338 pixels; RGB; 792K

N=3

600x338 pixels; RGB; 792K

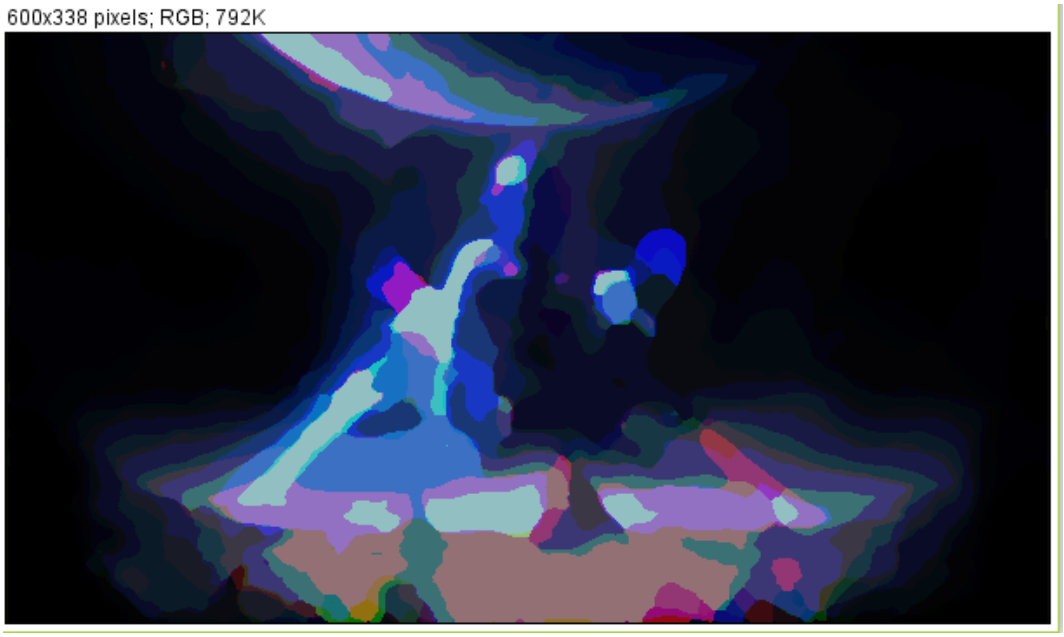
N=5

600x338 pixels; RGB; 792K



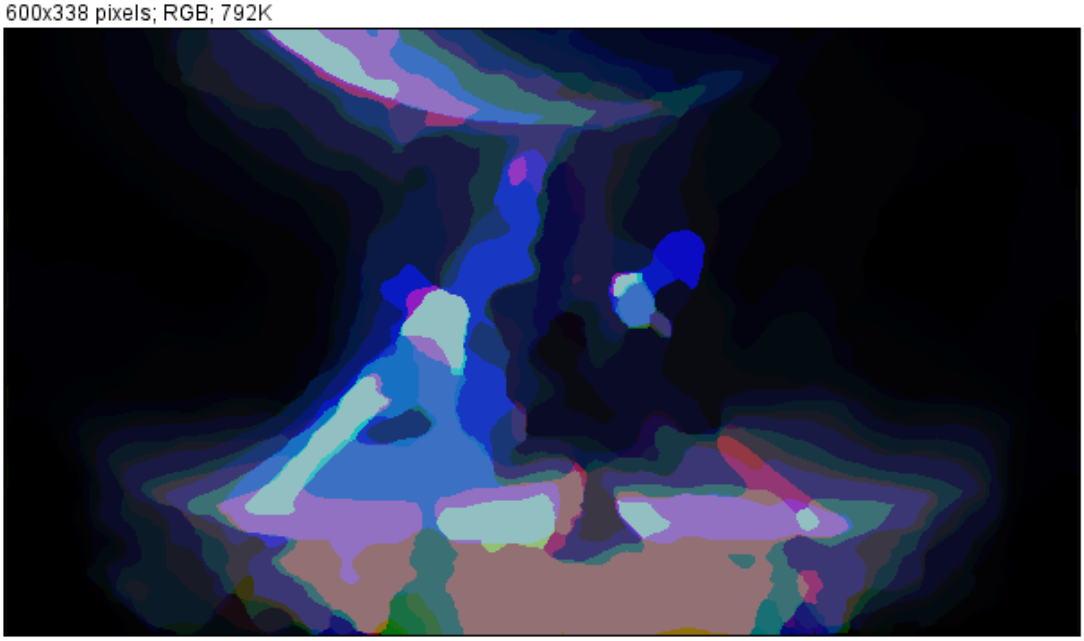
N=7

600x338 pixels; RGB; 792K



N=9

600x338 pixels; RGB; 792K



N=11

Figure 10: Oil Painting of Star Wars 512 Colors

720x480 pixels; RGB; 1.3MB

Original Trojans Image

720x480 pixels; RGB; 1.3MB

(a) 64 Colors Trojans Image

Figure 11: Reduced Color Image of Trojans

720x480 pixels; RGB; 1.3MB

(a) N=3

720x480 pixels; RGB; 1.3MB

(b) N=5

720x480 pixels; RGB; 1.3MB

(c) N=7

720x480 pixels; RGB; 1.3MB

(d) N=9

720x480 pixels; RGB; 1.3MB

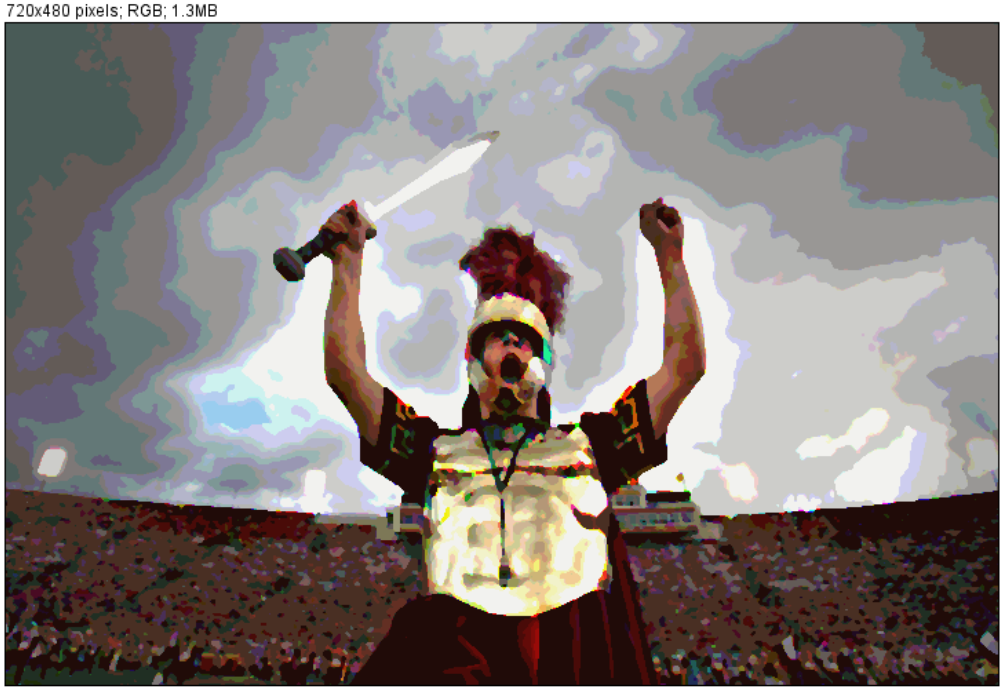
(e) N=11

Figure 12. Oil Painting of Trojans

720x480 pixels; RGB; 1.3MB

Figure 13. 512 Colors Trojans

720x480 pixels; RGB; 1.3MB



N=3

720x480 pixels; RGB; 1.3MB

N=5

720x480 pixels; RGB; 1.3MB

N=7

720x480 pixels; RGB; 1.3MB

N=9

720x480 pixels; RGB; 1.3MB

N=11

Figure 14: Oil Painting of Trojans 512 Colors

256x256 pixels; RGB; 256K

Figure 15: Original Image of Girl

Figure 16: Film effect of Girl

2.4 Discussion

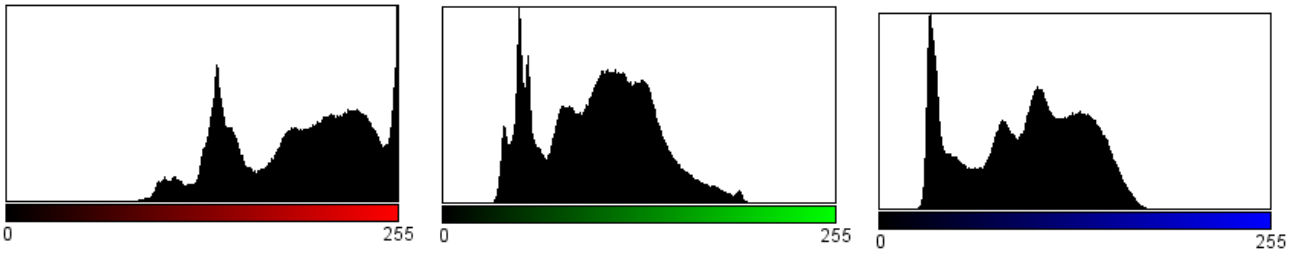
- **Histogram Equalization:**
  1) The Red, Green and Blue Channel histograms are as plotted in the **Figure 5(a,b,c).** The x-axis has the pixel intensities and the y-axis has the Number of pixels.
  2) Method A: The transfer function based output is seen in **Figure 6a.** We can see the entire study table clearly which wasn't visible in the original image. This method works successfully to an extent as it is based on uniform distribution. The corresponding Transfer function output for each channel is seen in **Figure 7(a,b,c).** We can see a uniform distribution.
  3) Method B: The Cumulative probability based output is seen in **Figure 6b.** We can see the entire study table clearly again which wasn't visible in the original image. This method works better than the previous method as in the artificial contours in the histogram of the previous image is eliminated and the bins are filled uniformly this time. The corresponding cumulative histograms for each channel can be seen in **Figure X(a,b,c). Figure Z(a,b,c)** gives the bucket filled histogram of all the channels.
  4) Both methods are fairly good in terms of the resulting output images. We are able to get a better quality image by enhancing the original image. But according to me, the Method B is better than Method A as artificial contours are generated in the histogram of Method A. Whereas this isn't the case with Method B. But these two methods also amplify or enhance the noise present in the original image. Therefore it is not very efficient.
     So we need to use something called adaptive histogram equalization method, which is tile based and considers the enhancement on a tile basis. So the enhancement of noise is limited and gets a better output image. This is used in professional software like Adobe Photoshop.
- **Image Filtering – Creating Oil Painting Effect:**
  1) **Figure 7a** and **Figure 11a** show the 64 color implementation of the original image. The threshold is chosen as follows. We need to reduce 256 colors to 64 colors. This is done by dividing the total number of pixels by 4. We get the bin size from this. So each channel will have only 4 colors. Therefore 4x4x4=64. This bin size is used as a metric or threshold to

classify the pixels into 4 bins. The threshold for each bin is the last pixel's intensity and the intensity of each pixel is the mean of the intensity of all the pixels in that particular bin. The rest of the logic is as explained in the Approach.

2) **Figure 8** and **Figure 12** Show the corresponding oil painting effect for the both the images with various N values. As N goes larger, the oil painting effect becomes more prominent and the image details reduce. This is because a larger window size results in large areas with the same intensity values. Edges and some details in the image reduces and we get a better effect.
3) **Figure 9, Figure 10 and Figure 13, Figure 14** represent the 512 color image of the Star Wars, Oil painting with 512 colors for different values of N for Star Wars and 512 color image of the Trojans, Oil painting with 512 colors for different values of N for Trojans. When we compare this 512 colors with the 64 colors image, we can see that the former is a lot more colorful and has more details in it. This is because it has a wider set to assign the most frequent color in a given neighborhood to the pixel. Hence The 512 Color oil painting is better than its counterpart.

- **Image Filtering – Creating Film Special Effect:**

Figure 16 gives the output of the film special effect filter to the Girl image. This is obtained by achieving the algorithm. We did reverse engineering to find the algorithm by analyzing the color channel relationship between the input and output film effect of the original image of Yosemite. The same algorithm is applied and we can see the same film effect on the Girl image. The below histograms are the Film effect histogram's of the given image(Yosemite) and the algorithm is taken by analyzing the histogram shrink from the below histograms.



3. Problem 3

3.1 Motivation

Noise is something which no human being likes. It is considered to be unwanted by everyone. Even in image processing, noise is expressed with disgust and is best to be removed. Images get corroded or the quality deteriorates if there is noise in the image. In mathematics, especially in probability, noise is considered as a random error.

In Image processing, noise may be introduced due to many factors such as poor or dull lighting, thermal noise, etc. They induce salt and pepper noise, Gaussian noise etc.

In this problem, we implement different methods and algorithms to reduce noise and improve image quality. We use median filter, mean filter, PLPCA method filtering, BM3D filtering.

A moving rectangular window is used to capture pixel intensities and perform the required operation. It is done on all three channels separately.

In median filter, the center value is replaced with the median of all the values in that particular window. Likewise, in mean filter, the center value is replaced with the mean of all the values in that particular window.

To analyze the performance of each filter, we use the PSNR value(Peak Signal to Noise Ratio). Higher the value of PSNR, better is the filter quality to reduce the noise effectively.

3.2 Approach

3.2.1 Mix noise in color Image

From the histogram of the noisy Lena image we can say that the image has Salt and Pepper noise and Gaussian Noise. We need to reduce the. So we have to use a median filter to reduce Salt and Pepper noise and a Mean filter to reduce the Gaussian Noise. There is no particular order to be used to reduce the noise as long as we reduce it somehow. So best is to cascade them. I found that the PSNR value is high if I cascade Median followed by mean filter.

We basically have to pad the image by replicating the pixels so as to avoid indexing errors and then choose a window size and traverse across the image just like we did in Oil Painting. We first apply the median filter by finding the median of all the pixels in the window and then replacing the center pixel value with the value obtained. We then again repeat the same process but by calculating the mean of the values around the center pixel and then outputting the same to the finale image.

This can be implemented by using bubble sort to find the median value and then find the average of all the values like we normally do.

We can then calculate the PSNR by using the formula given in the questions doc.

The PSNR value for R, G, B channels can be, respectively, calculated as follows:

$$\text{PSNR (dB)} = 10\log_{10}\left(\frac{\text{Max}^2}{\text{MSE}}\right)$$

$$\text{where MSE} = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\left(Y(i,j)-X(i,j)\right)^2$$

X : Original Noise-free Image of size $N \times M$

Y : Filterd Image of size $N \times M$

Max: Maximum possible pixel intensity = 255

3.2.2 Principal component analysis (PCA)

1) **PCA** can be used as the filtering approach for noisy data. This is because images have a something called redundancy. It means that there are patches of pixels which are repeated throughout the image. This might not be visible to naked eye but intelligent algorithms are able to group such patches together. That is how PLPCA works, by exploiting the redundancy of the image. Sigma is the gaussian noise.
2) The PLPCA is implemented by using a MATLAB code posted by the authors on the website Source : http://josephsalmon.eu/code/index_codes.php?page=GPPCA
3) The best parameters I got are for sigma=25, hw=12(half size of searching zone),hp= 9(half size of the patch), factor_thr= 2.8(factor in front of the variance term for hardthresholding the co-efficients) The smaller the patch size, better is the PSNR. Also small size of searching zone fastens the process and gets better result.

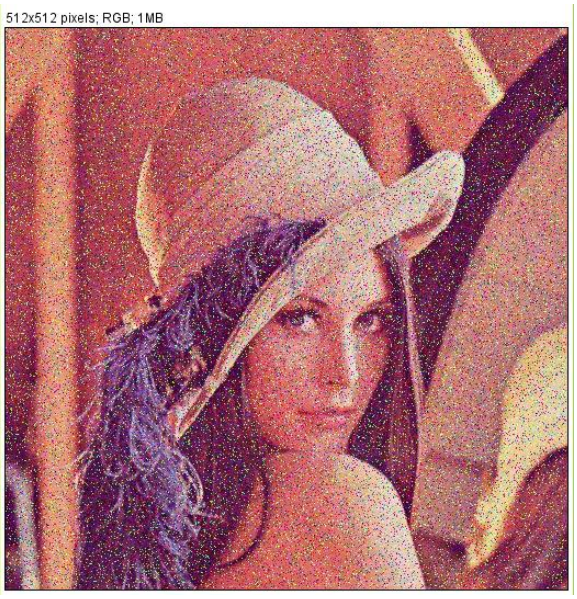
4) The value of PSNR obtained from the method is part 3a is around **27.69** and the best that can be obtained in PCA Approach is more than **31**. PCA is better in the sense that it reduces noise based on redundant patches rather than a standard window as in the 3a method.

3.3 Results

Filter type	PSNR(Lena)
Mean(3x3)	**23.45**
Median(3x3)	**25.93**
Mean(5x5)	**24.21**
Median(5x5)	**24.93**
Mean + Median (3x3)	**24.66**
Median + Mean (3x3)	**26.18**
Mean + Median(5x5)	**24.36**
Median + Mean(5x5)	**24.25**

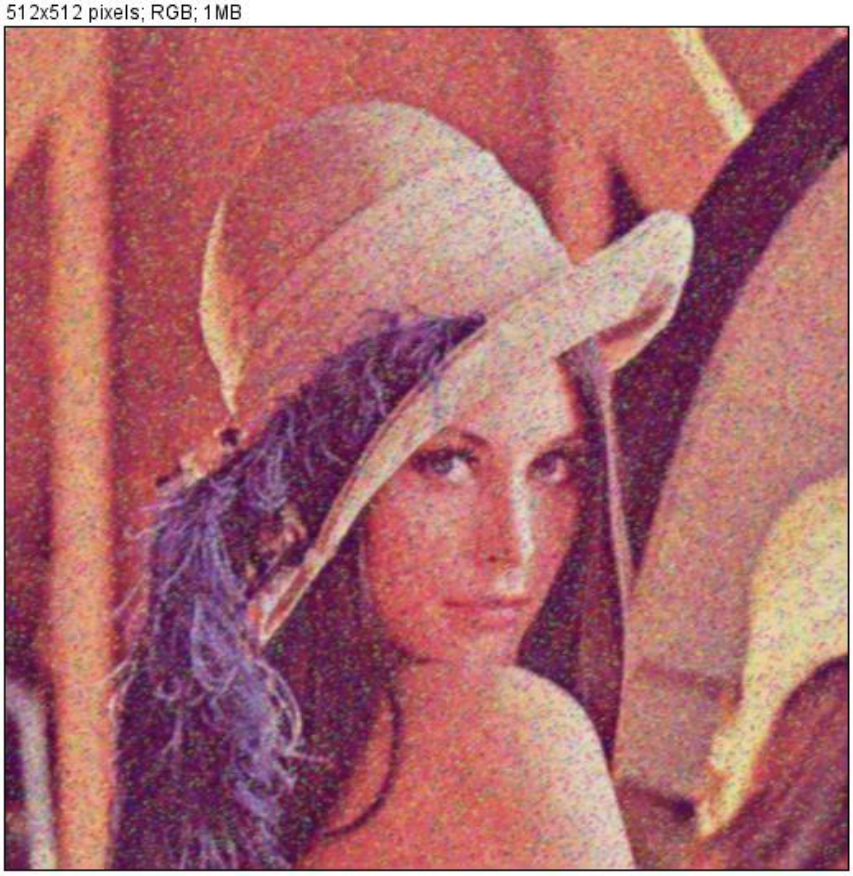
Table for 3a Method

Original Lena



Mixed Noise Lena

512x512 pixels; RGB; 1MB

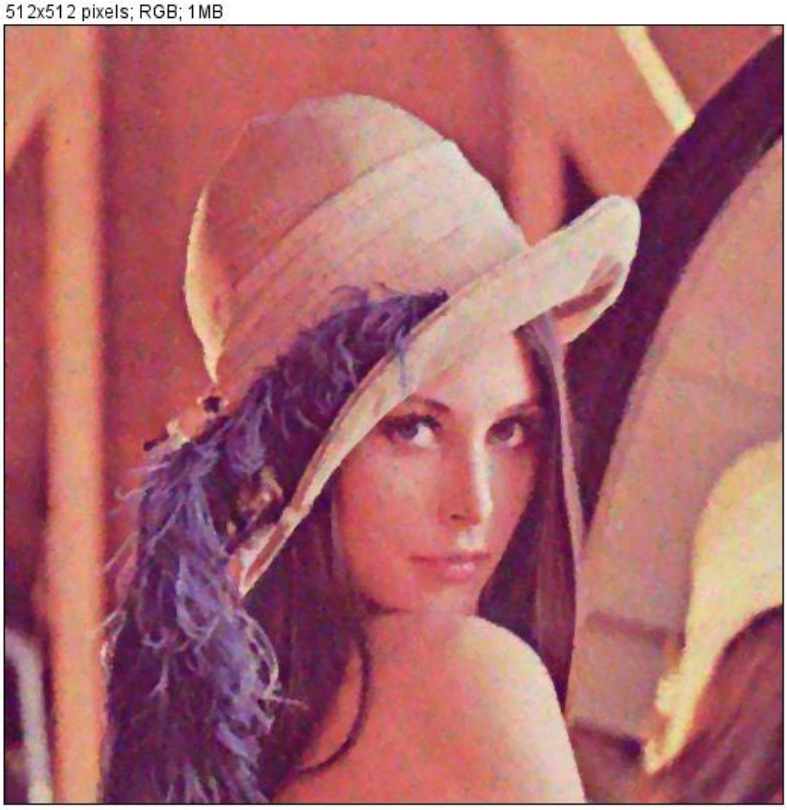


Mean filter with N=3

512x512 pixels; RGB; 1MB

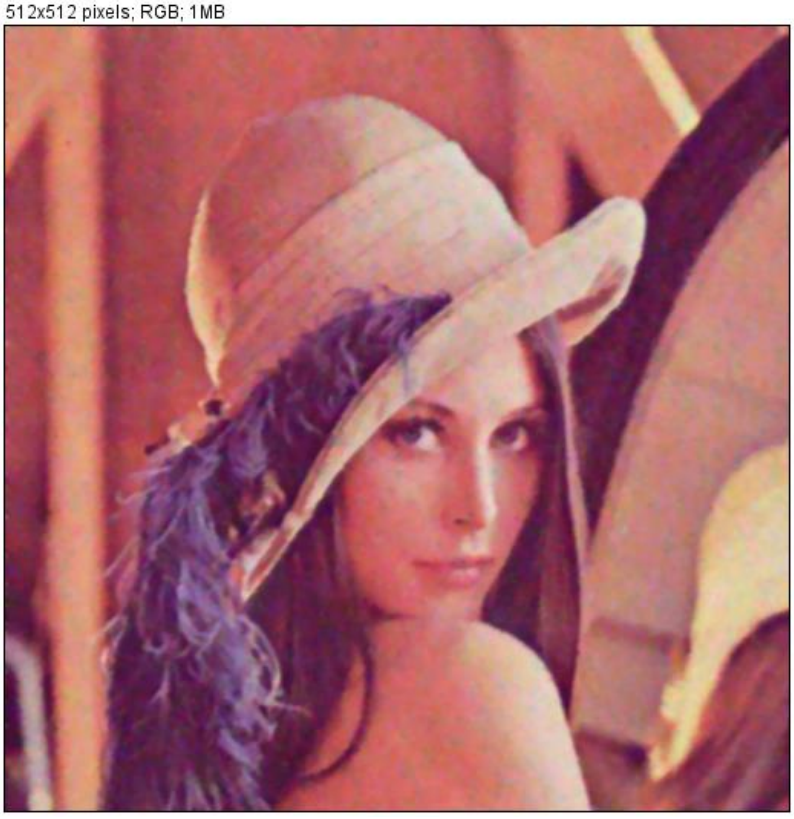
Mean+Median with N=3

512x512 pixels; RGB; 1MB



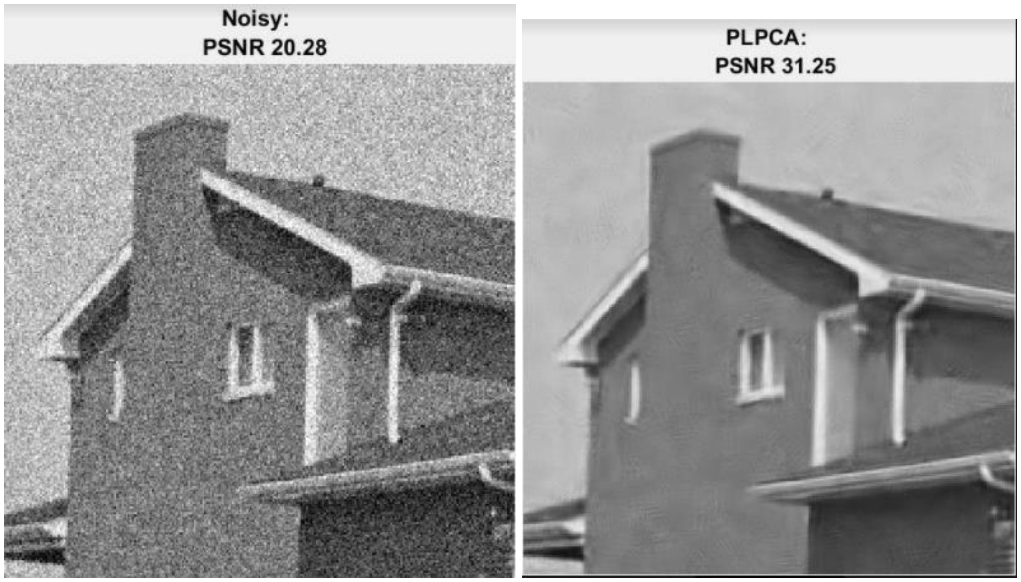
Median filter with N=3

512x512 pixels; RGB; 1MB



Median+Mean with N=3

Figure A: Lena denoised images



PLPCA OUTPUTS

Method 3a output for House

3.4 Discussion

3.4.1 Mix noise in color image

1)

1) Yes. All the channels have the same noise types. The noise types are salt and pepper and Gaussian Noise.

2) Yes, We have to perform filtering on individual channels separately for both noise types and then club them to display the final image.

3) We use Median filter for Salt and Pepper noise and Mean filter for Gaussian noise.

4) You may or may not cascade them in any order. It actually doesn't matter as long as we can reduce the noise. From What I found from the results, Median followed by Mean gives the best PSNR as shown in the above table.

5) The different filter and window size comparisons are found in the table above in the results. The best PSNR is achieved 26.18 for 3x3 Median+Mean filter.

2)

1) The method I followed is described in the approach above. The results are as above.

2)The shortcomings are the image gets blurry and details are lost from the image because as the window size increases the averaging will reduce the details.

3) To improve the performance we need to choose the right window size and the right cascade combination. Which in this case is Median+Mean of size 3x3.

3.4.2 PLPCA Method

The results are as shown above and we can clearly see that PLPCA Method is better than the method in 3a as the PSNR value is higher(31.25) when compared to 27.69

3.4.3 BM3D Method

1) **Consider a noisy image. There will be a lot of redundant patches. Partition the noisy image several 2D blocks. For a particular block find the similar blocks in the noisy image(redundancy) and stack them by grouping them in a 3D array. Apply 3D transform to this 3D array and give some threshold. Invert this 3D array and the result is a 2D blocks. Now return these blocks to the original image. If they are overlapping, do weighted averages of the blocks. Again repeat the above steps for the resultant image. Now compare the energy spectrum of both the blocks. Perform Weiner filtering on the noisy image. Then again return the blocks and if there is overlap, the do averaging by weights method.**
2) **The author uses block matching as all images are bound to have a redundancy. And they can be matched in blocks.**
3) **BM3D is both spatial domain and frequency domain filter in the sense that it used 3D transforms and stores the blocks in a 3D array (3-Dimensional). It is Frequency filter in the sense, it matches the blocks based on redundancy, which is the frequency or repetition of the blocks.**
4) **I think BM3D is a better method when compared to PLPCA because it uses blocks and does spatial and frequency domain filtering, which isn't done in PLPCA.**

Note: I haven't implemented BM3D in MATLAB. The above is based on my theoretical understanding of BM3D algorithm.

References:

http://imagine.enpc.fr/publications/papers/BMVC11.pdf **(for 3b)**

https://www.wikipedia.org/

https://www.google.com/

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**.(for 3c)**