EE 569

PROJECT #1

BY

PRANAV GUNDEWAR

USC ID: 4463612994

EMAIL: gundewar@usc.edu

Table of Contents

PROBLEM 1: BASIC IMAGE MANIPULATIONS

A) COLOR SPACE TRANSFORMATION

1- Color – To – Grayscale Conversion

I. Abstract and Motivation

The idea for this part is to get familiarize with importing images, its representation in different color spaces and interpretation of information conveyed through all this color spaces such as RGB, CMY, grayscale image etc. Color spaces is a way to define complete set of colors in visible spectrum. Every space has different application and is equally important in this part, we have given RGB image. It has 3 planes R, G & B. Each channel has 1 byte = 8 bits = 2^8 = 256 possible values. For RGB color space, total combination has 3 bytes per pixel. Examples asks us to convert 3 bytes per pixel to 1 byte because grey scale image 1 byte per pixel. Hence grey image can be obtained from color image by using 3 main methods

- Averaging Method: This is simplest method. In this method, we take pixel values of all 3 planes and find out average of 3 pixels.

$$\boldsymbol{Pixel\ value = \frac{(R+G+B)}{3}}$$

- Lightness Method: This method uses the concept of most prominent and least prominent color pixel. Most prominent pixel is the color pixel which has highest intensity among all color planes. This method averages the most prominent and least prominent pixel values.

$$\boldsymbol{Pixel\ value = \frac{(max(R,G,B)+min(R,G,B))}{2}}$$

- Luminosity Method: This is sophisticated version of averaging method. Weighted average of pixels is taken in this method where green color is weighted more than red and blue.

$$\boldsymbol{Pixel\ value = 0.21R + 0.72G + 0.07B}$$

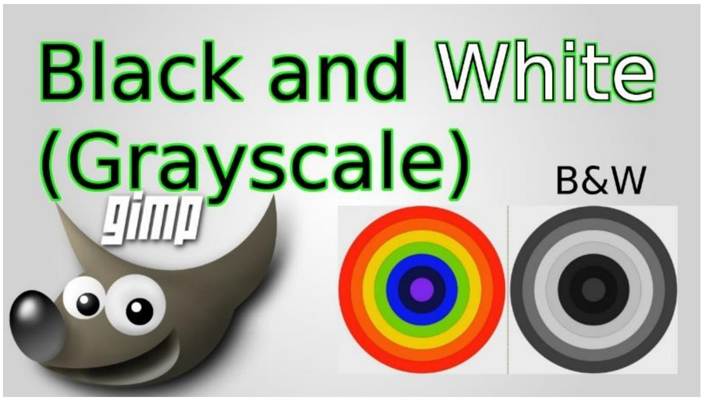


Fig1. Representation of RGB to greyscale image
(Source: https://www.youtube.com/watch?v=iHDnt0hZjiQ)

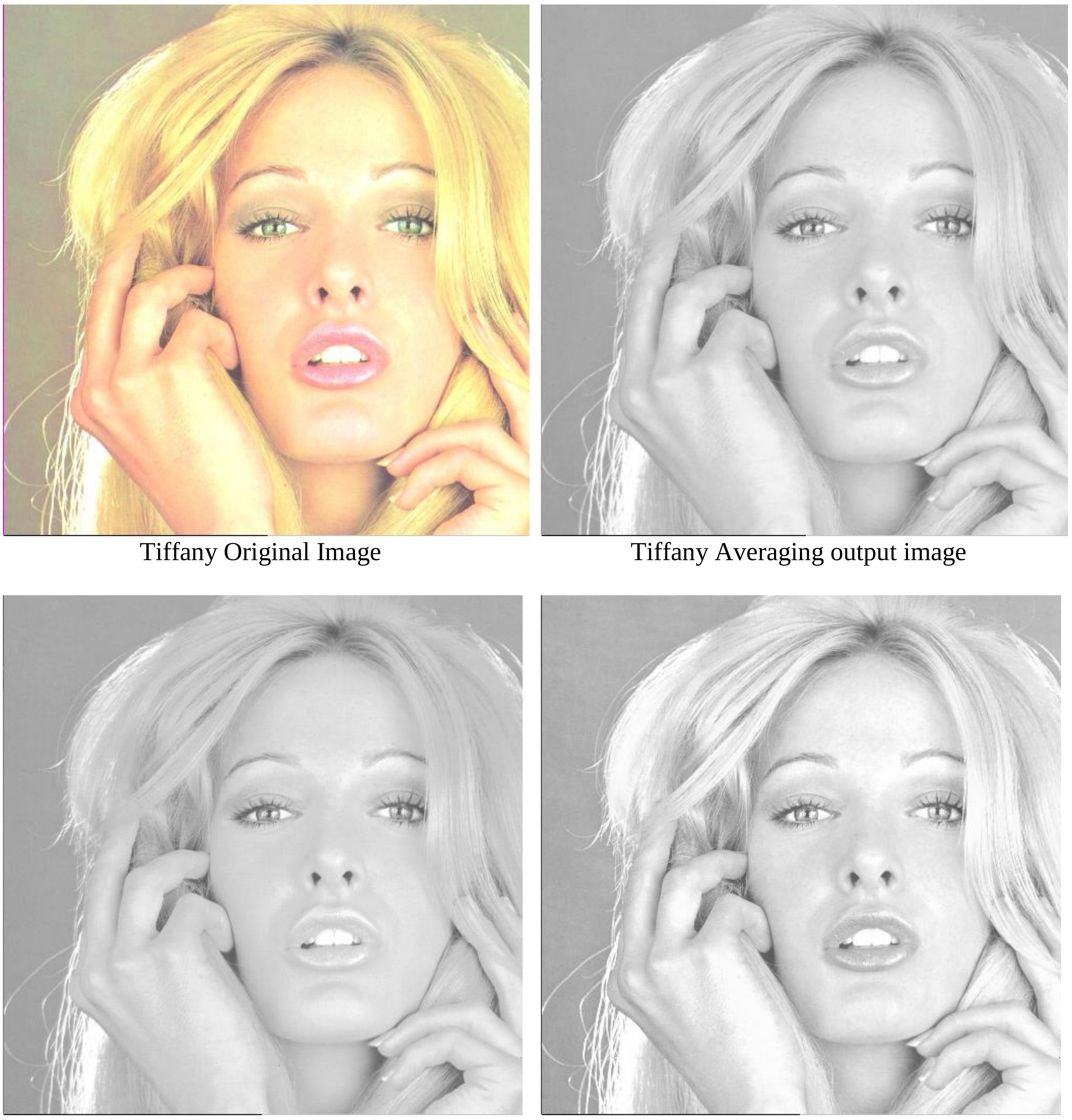
Task: Convert RGB image into greyscale using 3 given methods

II. Approach and Procedure

Input image is read into an 3D array. It is converted into grey-scale images using 3 methods mentioned above.

Algorithm:
STEP 1 - Read input image using 'fileread' function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel and convert to grey scale value using conversion formula and save the new pixel value in another 3D array.
STEP 3 – Use 'filewrite' function to write the new 3D array into raw file for displaying the output.

III. Experimental Results


Tiffany Original Image

Tiffany Averaging output image

Tiffany Lightness output image

Tiffany Luminosity output image

Fig2. Tiffany image and its conversion using various methods

IV. Discussion

Figure 2 shows grey-scale conversion using averaging, lightness and luminosity method. The averaging method as we can observe reduces the contrast of image and hence appears little darker. The lightness method works better than averaging and produces good grey-scale output but still edges looks little sharp than it supposed to be. The luminosity method works best among all three methods and produces better output among all these methods and retains most of the pixel contrast of original image. Humans are more sensitive towards green color and this method forms weighted average by considering this concept. Hence luminosity method is used almost all the applications.

This experiment helps us to understand processing image data and pixel transversal into different color spaces.

2- CMY(K) Color Space

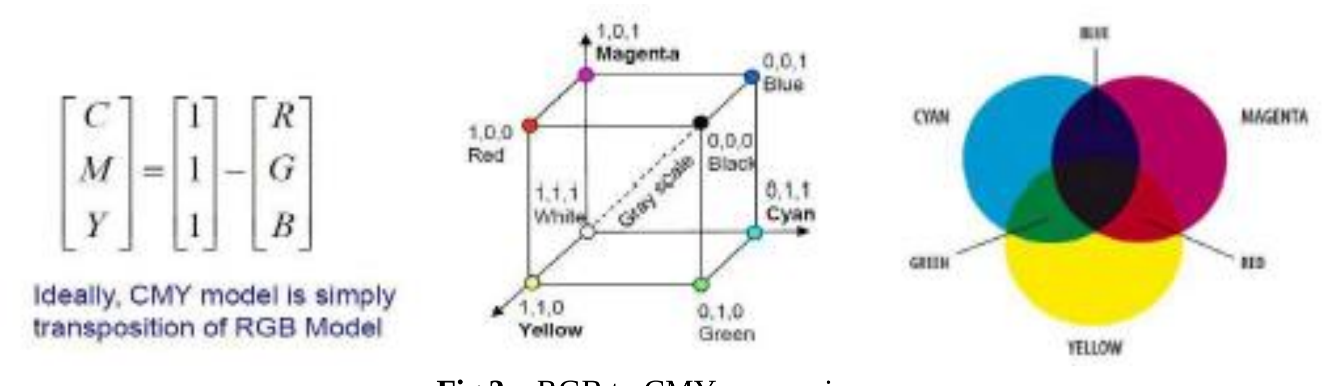
I. Abstract and Motivation

CMY(K) color space is generally used in imaging and printing application. It stands for Cyan, Magenta, Yellow and Black (CMY-K). RGB is an additive projected light color system i.e. all colors start with black and other colors are obtained by adding different light colors [1]. On the other hand, CMYK is subtractive, reflected light color system. It starts with white color. All the colors are obtained when certain colors are added to absorb light that is reflected hence known as reflected system [1]. To obtain black, we need add 3 colors. Sometimes black color (K) is added for the printing process. Regular display devices like monitors and TV uses RGB color space where you can find all 3 RGB pixels at every location. For imaging and printing process, CMY(K) color is used for as it uses relatively less ink and as printing paper is white and we need something that can absorb instead of reflecting.

Task: Convert RGB image into CMY image and display C, M, Y planes separately.

II. Approach and Procedure

There is conversion formula that converts RGB plane into CMY plane.



$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$
Ideally, CMY model is simply transposition of RGB Model
1,0,1 Magenta
0,0,1 Blue
1,0,0 Red
0,0,0 Black
1,1,1 White
Gray scale
0,1,1 Cyan
1,1,0 Yellow
0,1,0 Green
BLUE
CYAN
MAGENTA
GREEN
RED
YELLOW

Fig.3 – RGB to CMY conversion
Source – Lecture notes

From figure 3, cyan can be obtained from red by subtracting 1 from its red pixel. Similarly, magenta and yellow colors are obtained. Both RGB and CMY models use cubic color model.

Algorithm:
STEP 1 - Read input image using ‘fileread’ function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel and convert to CMY value using conversion formula. First normalize the RGB pixel ranging from 0 to 255 into range (0.1) by dividing by 255.
STEP 3- Now subtract 1 from each plane pixel value. Now renormalize image by multiplying by 255 and save the new pixel value in another 3D array.
STEP 3 – Use ‘filewrite’ function to write the new 3D array into raw file for displaying the output. Also use ‘filewrite’ to save three C, M, Y planes separately in different raw files to check separate planes.

III. Experimental Results:

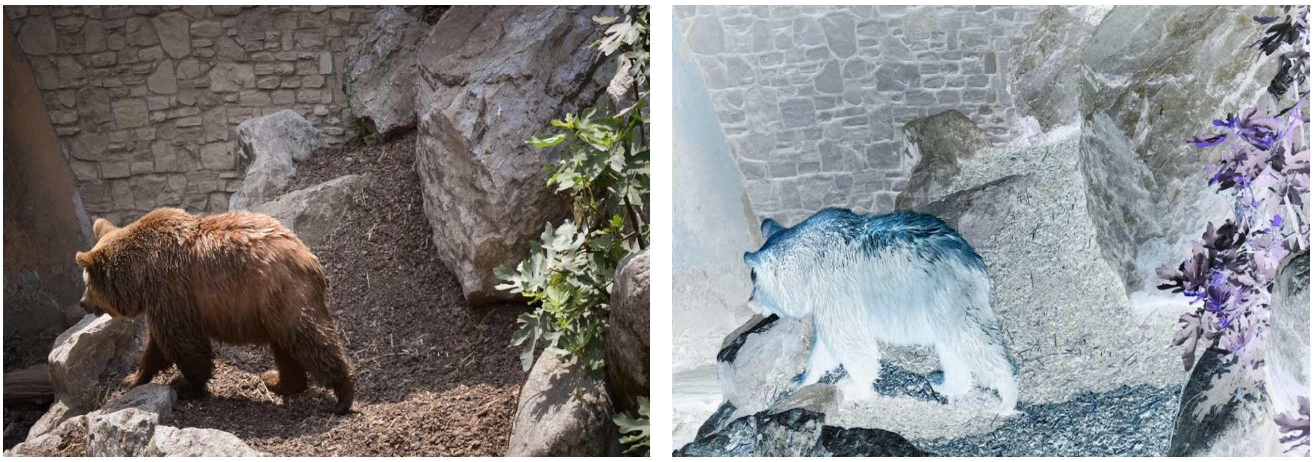


Fig.4 – RGB Input Image Bear and CMY output image

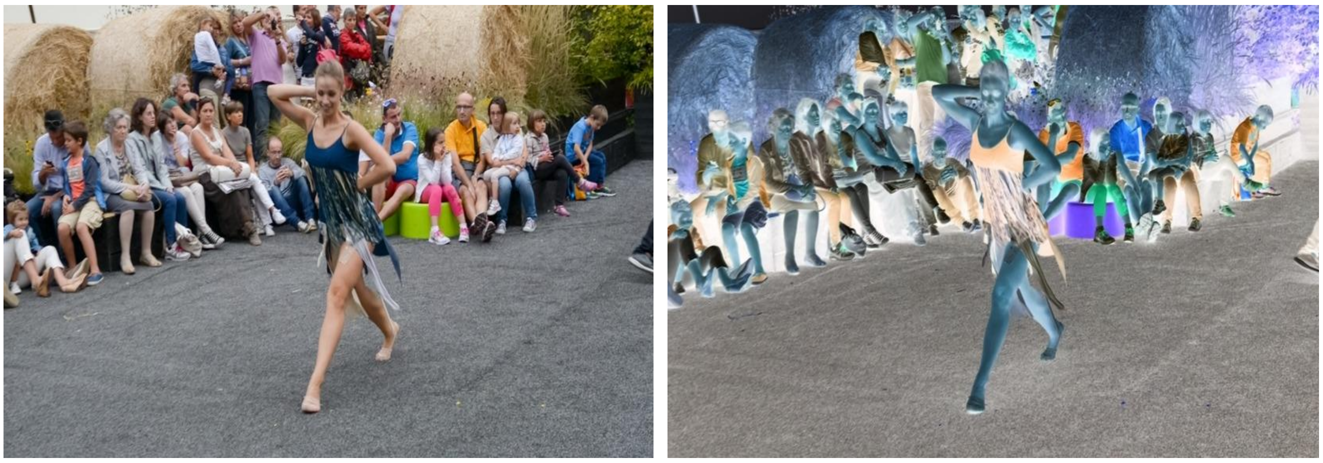


Fig.5 – RGB Input Image Dance and CMY output image

Figure 4 and 5 shows RGB input image and its conversion into CMY output image. Figure 6 shows C, M, Y planes of output images separately.

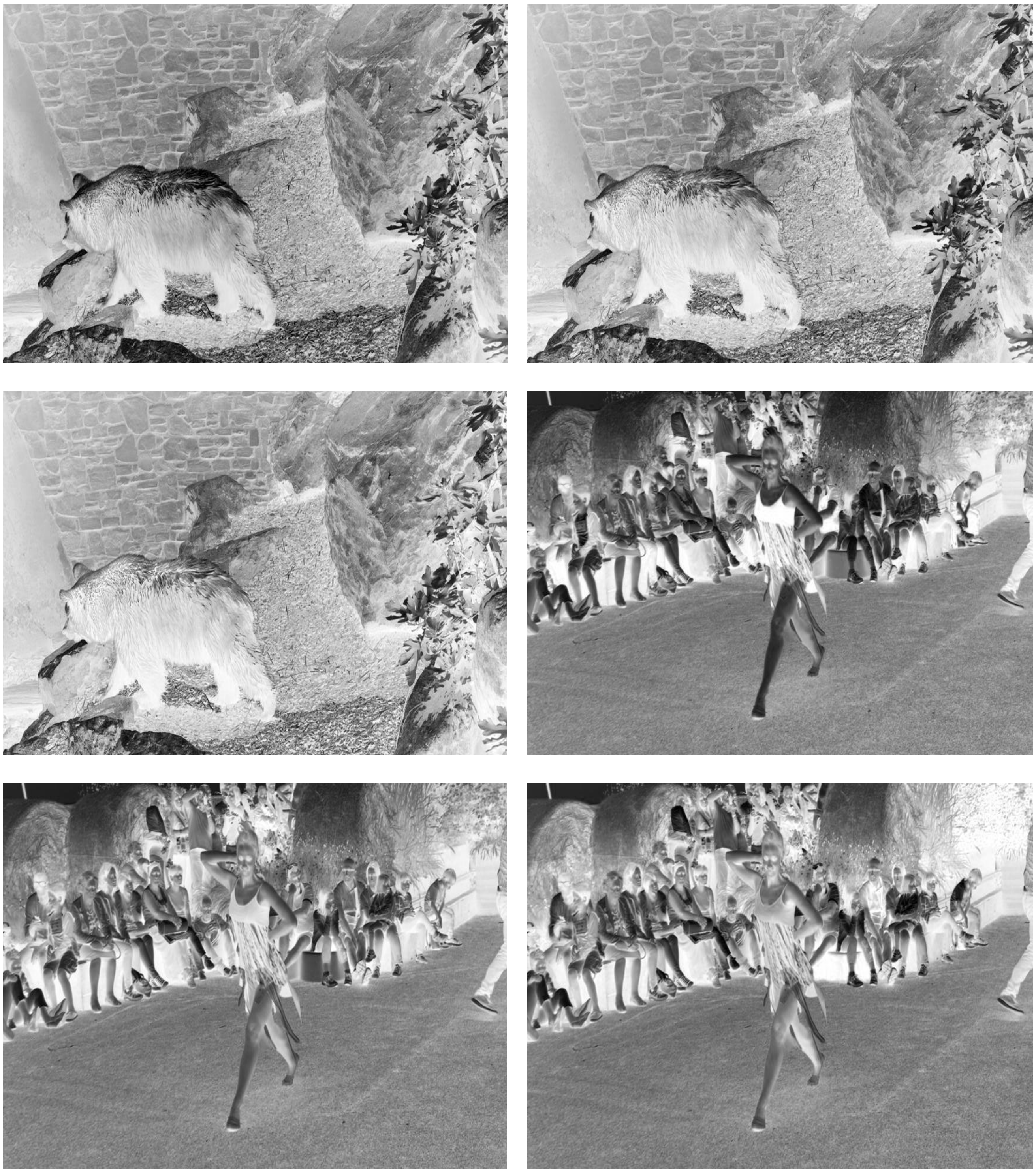


Fig.6 – C, M and Y components respectively

IV. Discussion

RGB to CMY conversion as well as C, M, Y components of output images are shown in above figures. CMYK model has adds pure black color to the mixture and hence is being used in imaging and printing industry as it has less ink consumption.

B) IMAGE RESIZING VIA BILINEAR INTERPOLATION

I. Abstract and Motivation

Image cropping and resizing are one of the most popular image processing operations. In this part of experiment, a simple method of bilinear interpolation used to explain the concept of image resizing. Image resizing is often used for up-sampling or down-sampling depending upon the application. When an image size has been increased, we say image is up-sampled. The basic idea behind this experiment is effects of resizing and its effect in quality of the image.

Task: Resize 512*512 image into 650*650 image

II. Approach and Procedure

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

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

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

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

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

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

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

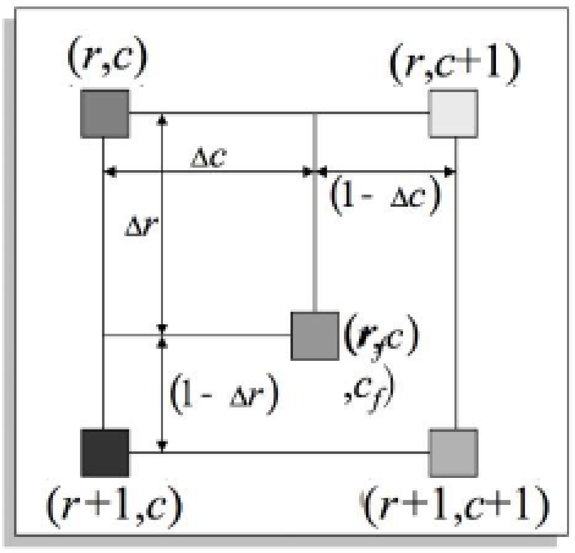


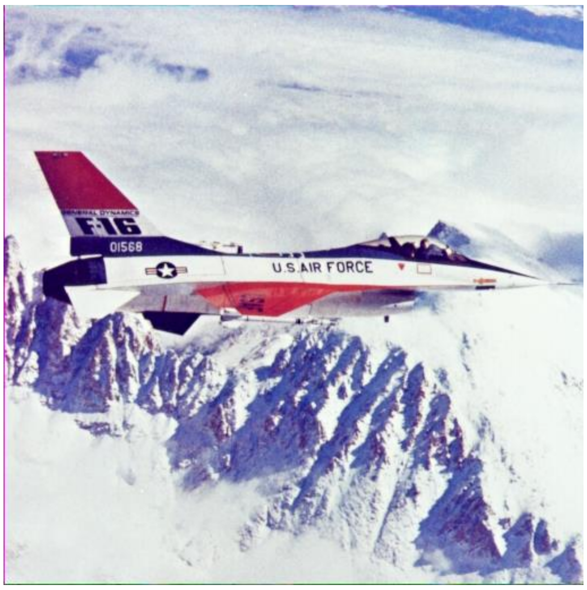
Fig.7 – Bilinear Interpolation Algorithm
(Source: https://www.codementor.io/tips/7814203938/resize-an-image-with-bilinear-interpolation-without-imresize)

Figure 7 explains bilinear interpolation algorithm. To resize an image, we first have to calculate ratio of resized and original image sizes. We must map original image co-ordinates to the new co-ordinates using the calculated ratio. In the given figure, sr & sc are the ration for row and columns. Also, rf and cf are the new mapped co-ordinates for the resized image.

Algorithm:
STEP 1 - Read input image using ‘fileread’ function and get height, width and bytes per pixel from user.
STEP 2 - Run 3 nested loops for height, width and bytes to access each pixel of original image and calculate ratio of sizes for original image co-ordinates to the new co-ordinates.
STEP 3 – Using the calculate ratio, map old co-ordinates into new co-ordinates. Calculate new height and width using the described formula.

STEP 4 – Calculate the translation parameter by which co-ordinate would change when mapped into the resized image.
STEP 5 – Use this values and formula given above to find resized image pixel value which is J(r’,c’).
STEP 6 – Use ‘filewrite’ function to write the output resized image in new 3D array into raw file for displaying the output.

III. Experimental Results:



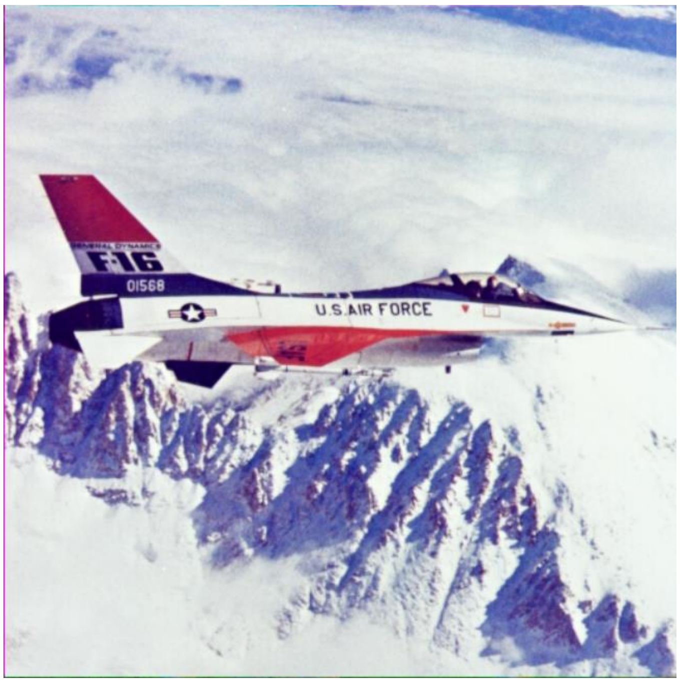


Fig.8 – Original and Resized Image

IV. Discussion

After resizing an image, we can observe there is loss of quality. This is because when we up-sampled image, we introduced the zero values pixel in it thus it appears little pixelated. Bilinear interpolation is non-adaptive algorithm to resize or crop an image and creates aliased edges and edge halos. Hence it deteriorates the image quality and not an effective method. Adaptive interpolating algorithms that changes the pixel values depending upon current conditions might produce better output than bilinear interpolation algorithm.

PROBLEM 2: HISTOGRAM EQUALIZATION

A) HISTOGRAM EQUALIZATION

I. Abstract and Motivation

Histogram equalization technique deals with intensities and contrast of an image. Histogram equalization is a technique which improves image contrast by changing pixel intensities. Contrast is a factor which makes image distinguishable or not. Here, the concept of image enhancement is used which produces image which is more soothing to the eye than original. It is an objective method which differs from person to person. There are two methods used in this experiment to improve image contrast. Contrast enhancement changes pixel value distribution and it spans the distribution into wider range.

- The cumulative probability based histogram equalization:

This method considers the entire dynamic range of current pixels. It follows cumulatively adding probability of histogram input data. Here, we map pixel values of input using transfer function which has uniform distribution. We first calculate probability distribution function and for that we calculate probability of occurrence of each grey level by [3]

$$P(g(k)) = \frac{(nk)}{N} \ for\ k = 0\ to\ L-1$$

Where,

K – number of grey scales

N – Total number of pixel in input image

L – Number of possible intensity values

The transfer function which represents CDF based equalized image is given by [3]

$$T(k) = floor\left((L-1)\sum_{j=0}^{k}\frac{nj}{n}\right)k = 0\ to\ L-1$$

The output image is mapping of each pixel value with grey intensity k into corresponding grey intensity using transfer function.

$$I(r,c) = CDF\big(I(r,c)\big) * 255$$

I(r,c) is new pixel value corresponding location. CDF value ranges between 0 to 1. Hence renormalizing the image pixel by multiplying with 255.

- The bucket filling algorithm based histogram equalization

The CDF method introduces unnecessary extra contours. To avoid such things, bucket filing algorithm is used widely. We first create a map which stores location of all pixels for all pixel intensities available in input image. After that, we concatenate all pixel location into 1 array by looping through the map that we created for storing pixel location. We divide all the pixel into equal pixel bins.

$$Number\ of\ pixels\ per\ bin = \frac{Total\ number\ of\ pixels}{255}$$

II. Approach and Procedures

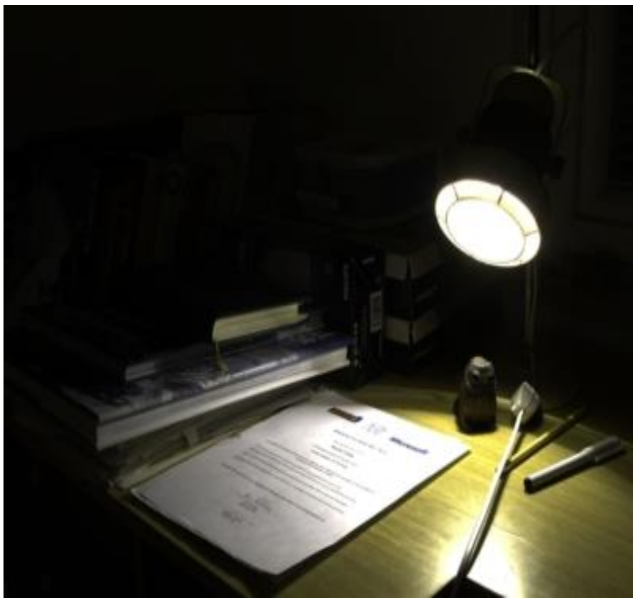
Algorithm for CDF:

```
STEP 1 - Read input image using 'fileread' function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel of original image. Initialize 3 arrays [256] for calculating histogram of each color plane. Create another array for CDF.
STEP 3 – For all grey values,
            Hist [i] = data[i] / total poxels.
            CDF [i] = hist [i] + CDF [i-1].
            Op [i] = floor (CDF [i] * 255)
STEP 4 - Create 3D array for storing output image Op where new pixel values will placed in the array.
STEP 5 – Use 'filewrite' function to write the output resized image in new 3D array into raw file for displaying the output.
```

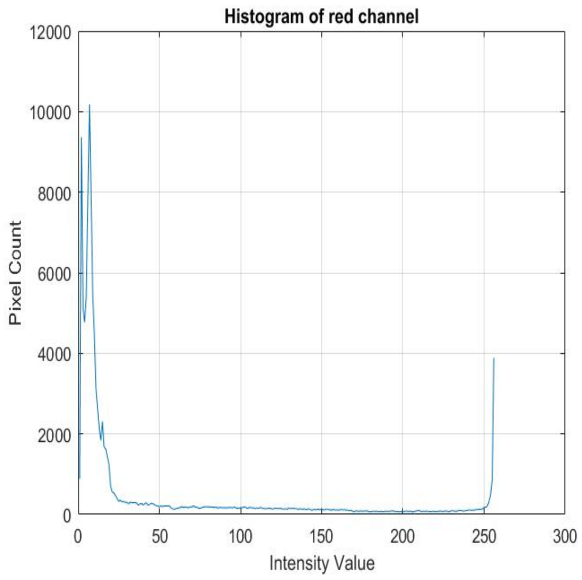
Algorithm for Bucket:

```
STEP 1 - Read input image using 'fileread' function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel of original image. Initialize 3 arrays [256] for calculating histogram of each color plane. Create another array for CDF. Find out total pixels in every bin.
STEP 3 – For all pixel location,
            For each grey value
                For each bin pixel count,
                Op [i] = grey value
                Increment the grey value counter.
STEP 4 - Create 3D array for storing output image Op where new pixel values will placed in the array.
STEP 5 – Use 'filewrite' function to write the output resized image in new 3D array into raw file for displaying the output.
```

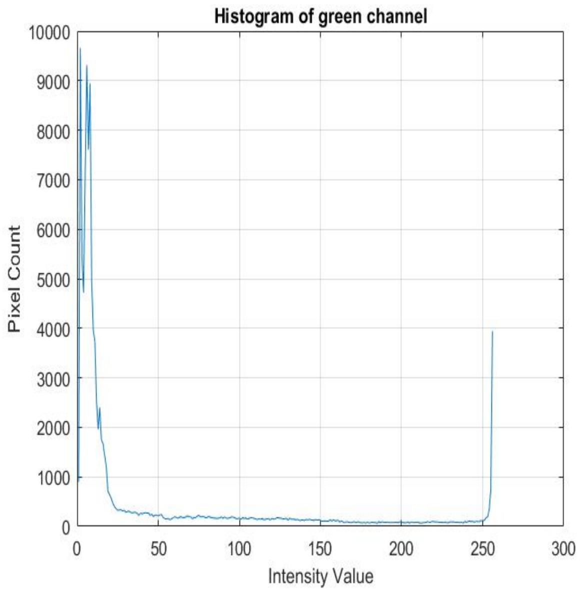
III. Experimental Results:



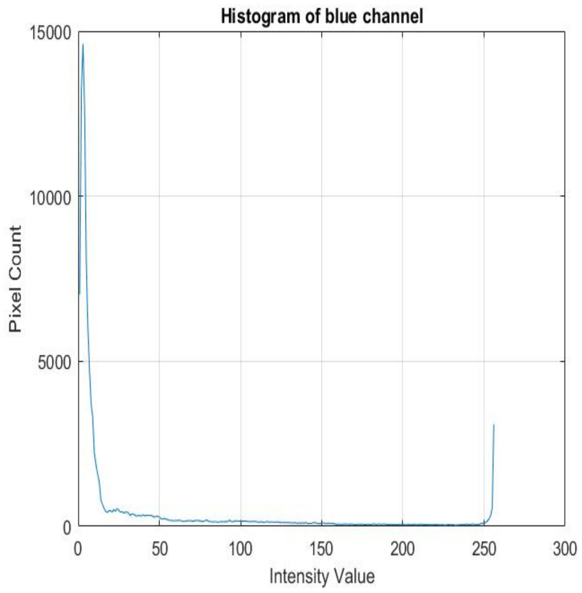
Original Input Image



Histogram of input red channel



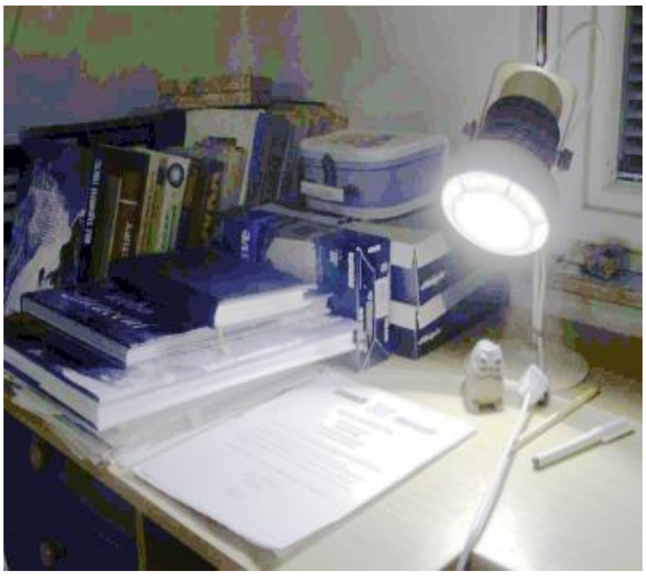
Histogram of input green channel

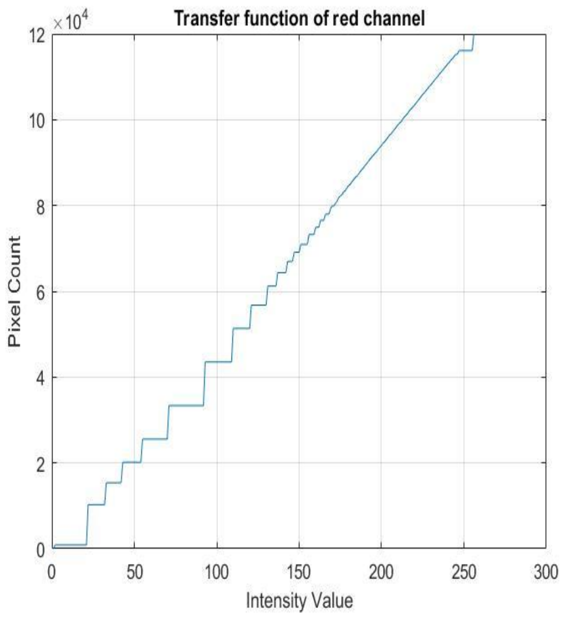


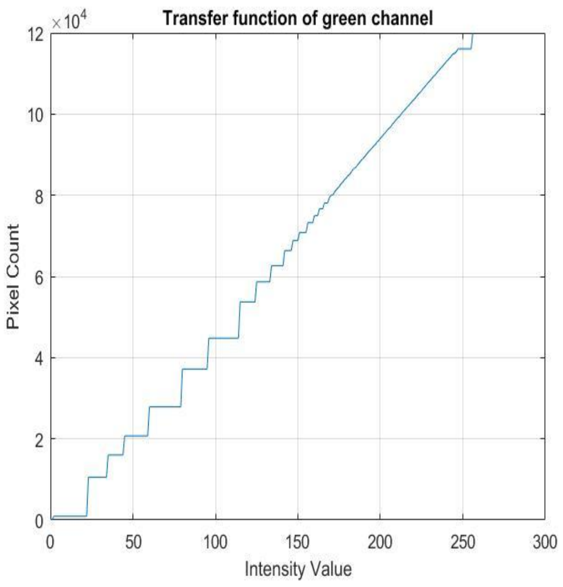
Histogram of input blue channel

Fig.9 – Input Image and histogram of each channels

In figure 9, we can histogram of each channel of observe original. The basic idea behind this plot is to know for what range of pixel, transfer function should have more slope. From histogram, we get better idea about input range of pixels and that can help while using different parameters for contrast enhancement. For all three channels, pixels are located more in the region 0 to 40 and 250 to 256. So transfer function will be having more slope for these regions







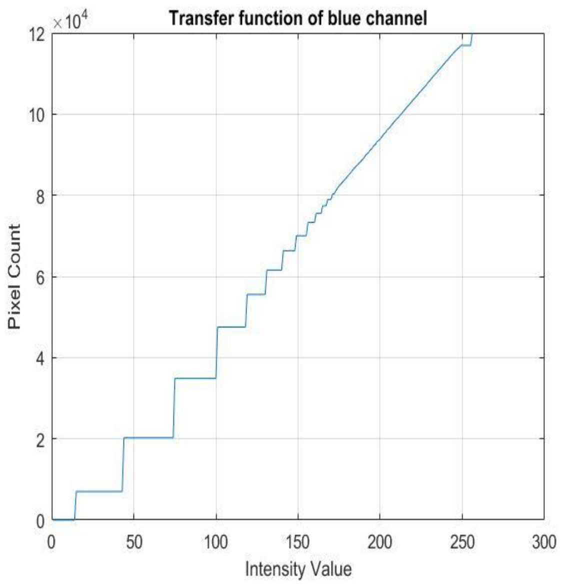
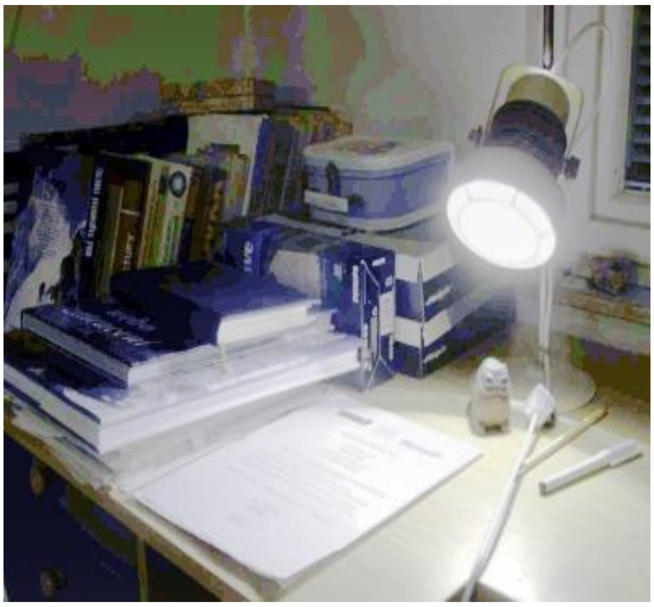
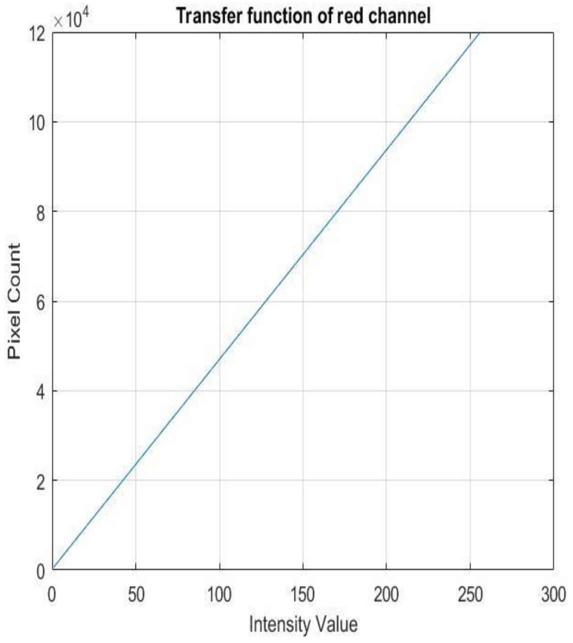
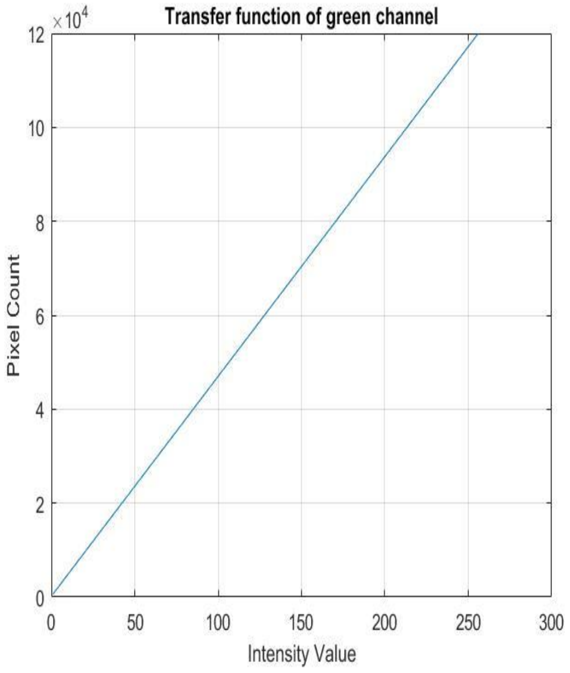


Fig.10 – Output using CDF method and transfer function of each channels

The cumulative distribution based method makes sure all the input pixels follow the CDF mapped distribution. Hence produced output image is smooth. The implementation logic is a bit tedious and lots of calculation are involved due to normalization and denormalization. This method improves the dynamic range of pixels as well as gives a well enhanced image.







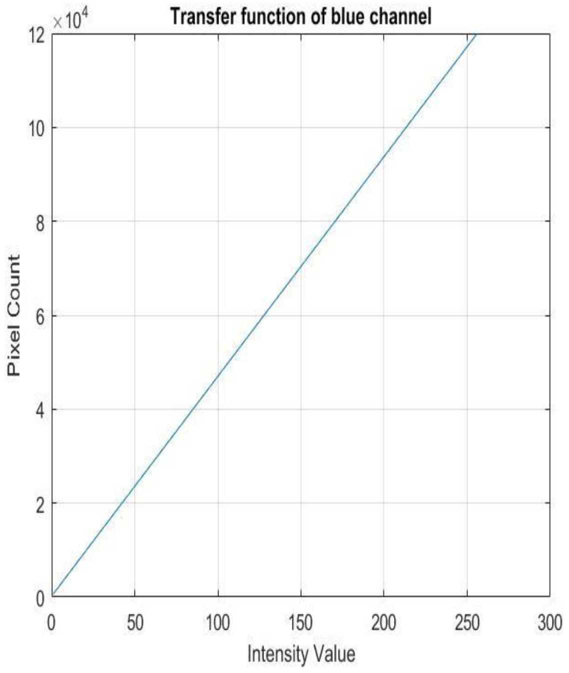


Fig.11 – Output using bucket filling method and transfer function of each channels

IV. Discussion

Figure 11 describes contrast enhanced output image using bucket filling algorithm. This method produces best enhanced output compared to transfer function or CDF method. It has whole dynamic range for pixels that means every grey intensity has at least 1 pixel. It improves chromaticity of an input image and does not produce any loss of quality. Edges are sharp and has less distortion. Transfer function is linear for entire range which is desirable hence is widely used in industry.

Applying histogram equalization to each channel introduces distortions as each channel has different color intensity distribution. A better approach would be convert input image into HSL domain which separates various parameters like chroma, luminance etc. Applying histogram equalization now will help us to retain the H, S, I of the image.

B) OIL PAINITING EFFECT

I. Abstract and Motivation

Proposed oil painting effect is an extension of bucket filling algorithm and imaging filter to find most frequent color. First, we have to divide input pixel color histogram into 4 bins of equal quantity.

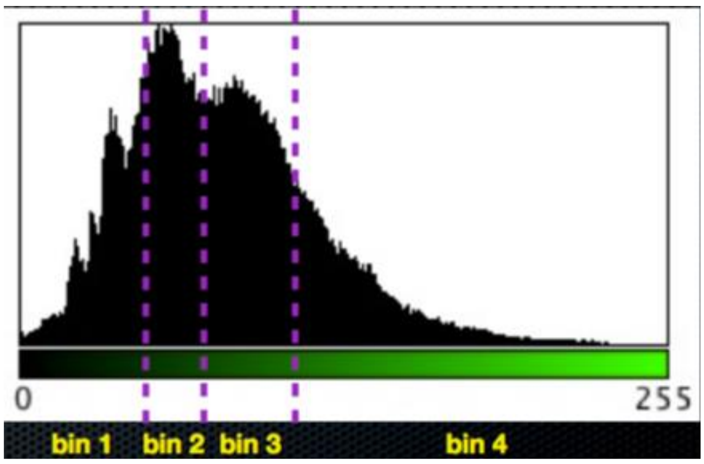


Fig.12 – Color quantization
(Source: Lecture notes)

As shown in figure, divide 256 levels into 4 bins where each bin has approx. same number of pixels. Repeat same for all color planes. Now use weighted mean to calculate mean for each bin.

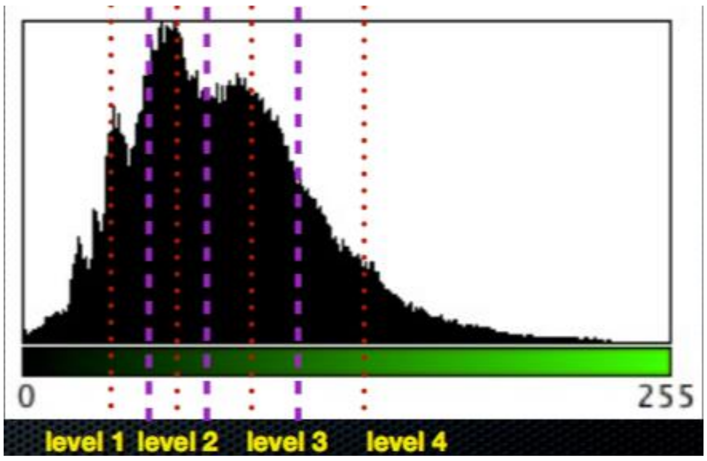


Fig.13 – Color quantization
(Source: Lecture notes)

Now replace every pixel with one of those means depending upon intensity value. Now zero pad the image using pixel reflection technique mentioned below.

```
A A | A B C D E F G H | H H
A A | A B C D E F G H | H H
---------------------------
A A | A B C D E F G H | H H
I I | I J K L M N O P | P P
Q Q | Q R S T U V W X | X X
Y Y | Y Z a b c d e f | f f
g g | g h i j k l m n | n n
o o | o p q r s t u v | v v
---------------------------
o o | o p q r s t u v | v v
o o | o p q r s t u v | v v
```

Fig.14 – Pixel reflection technique
(Source: http://www-cs.engr.ccny.cuny.edu/~wolberg/cs470/hw/hw2_pad.txt)

After padding the image, apply oil painting filter to find the maximum frequent color in selected window size and replace with that.

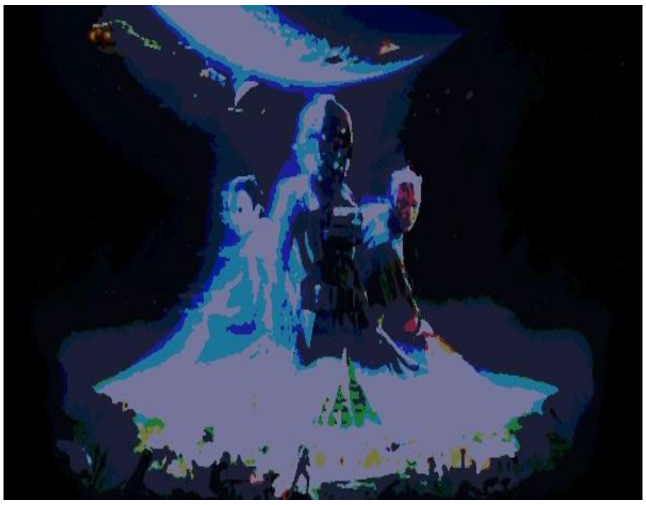
Purpose of this experiment is to understand concept of color quantization and how to apply filter to input image.

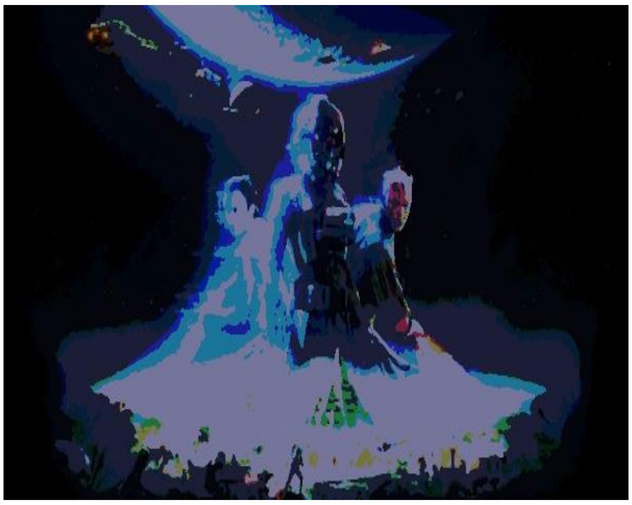
Task: We have to quantize input image and apply the oil painting filter to produce oil painting effect to the input image.

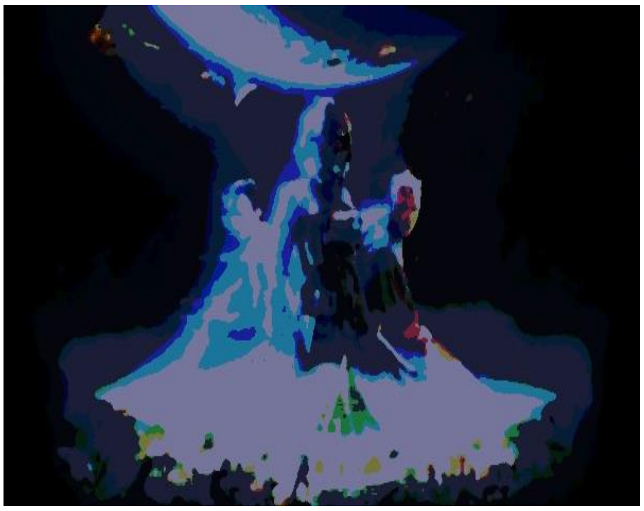
II. Approach and Procedures

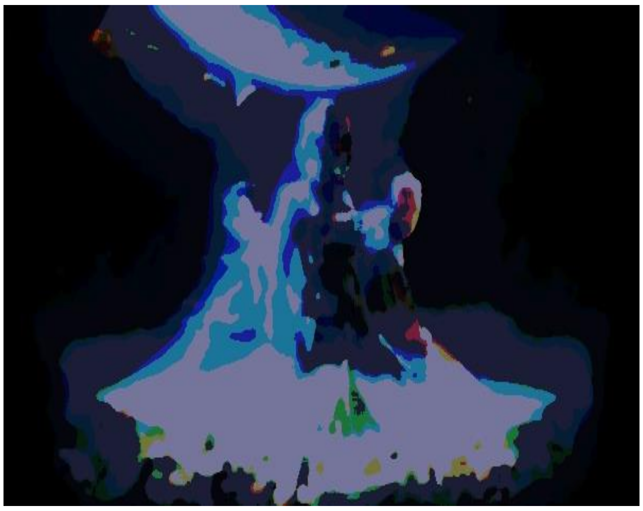
Algorithm:
STEP 1 - Read input image using 'fileread' function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel of original image. Initialize 3 arrays [256] for calculating histogram of each color plane. Create another array for calculating bins and weighted mean. Find out total pixels in every bin.
STEP 3 – For all pixel location, apply weighted average mean value and quantize the image.
STEP 4 – To quantized image, apply oil painting filter of selected window size and change the centre pixel with most frequent color.
STEP 5 - Create 3D array for storing output image Op where new pixel values will be placed in the array.
STEP 6 – Use 'filewrite' function to write the output resized image in new 3D array into raw file for displaying the output.

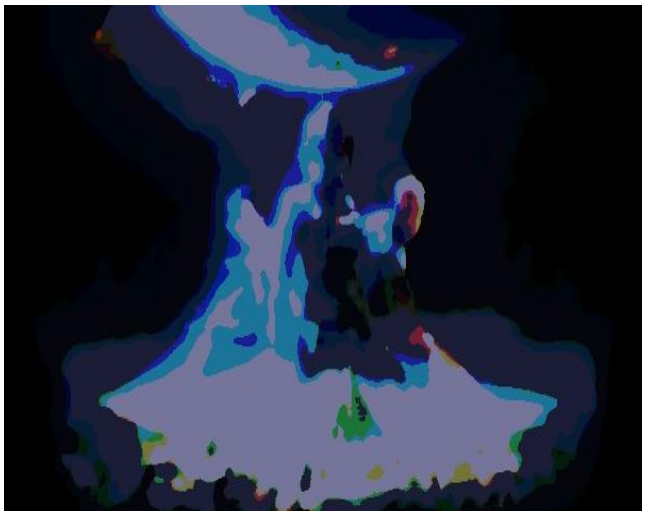
III. Experimental Results

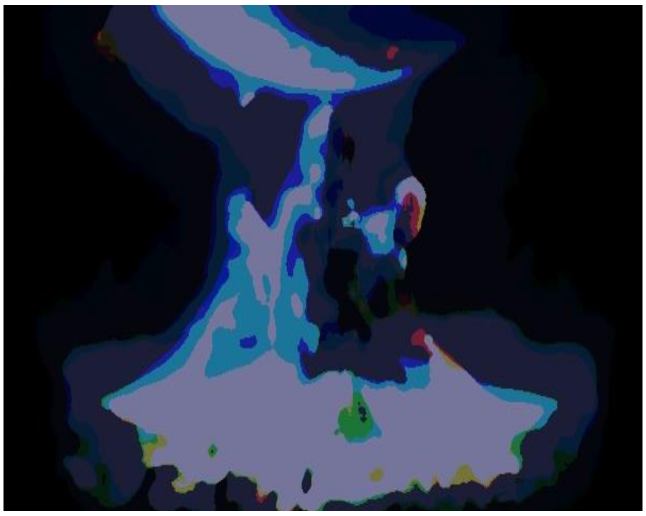

Quantized 64 color image

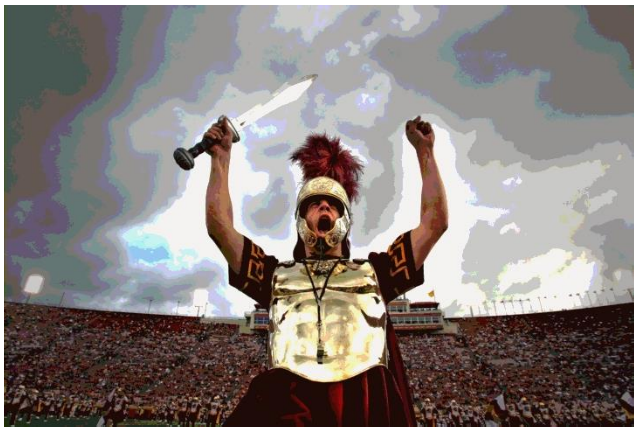

Output for N = 3

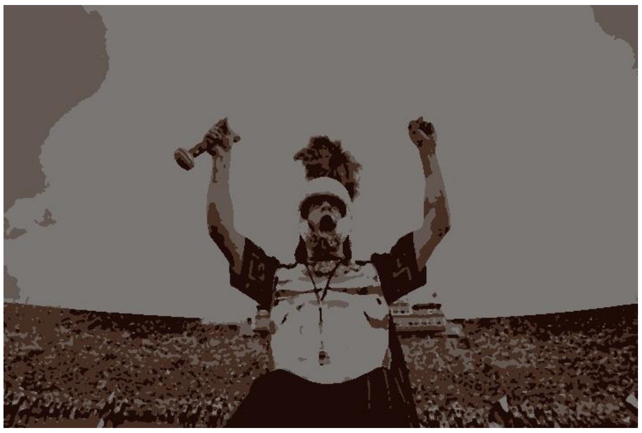

Output for N = 5

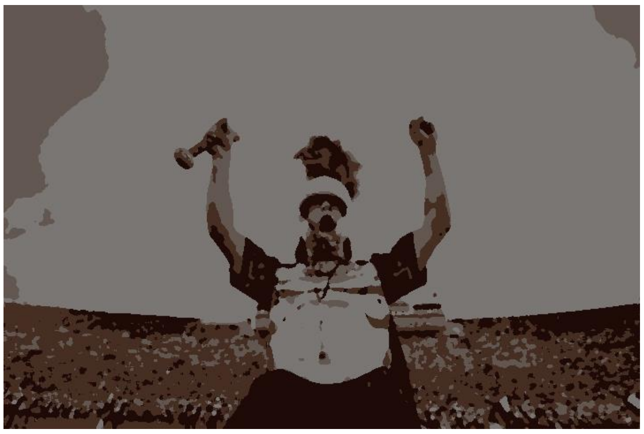

Output for N = 7

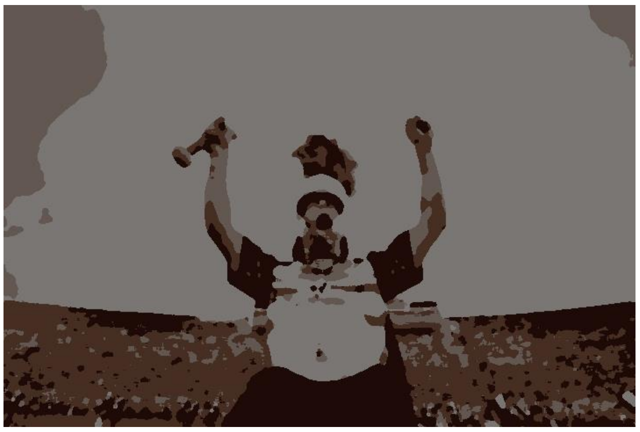

Output for N = 9

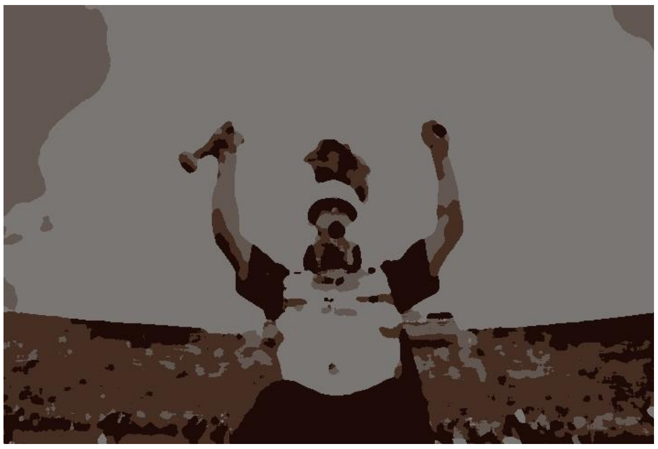

Output for N = 11

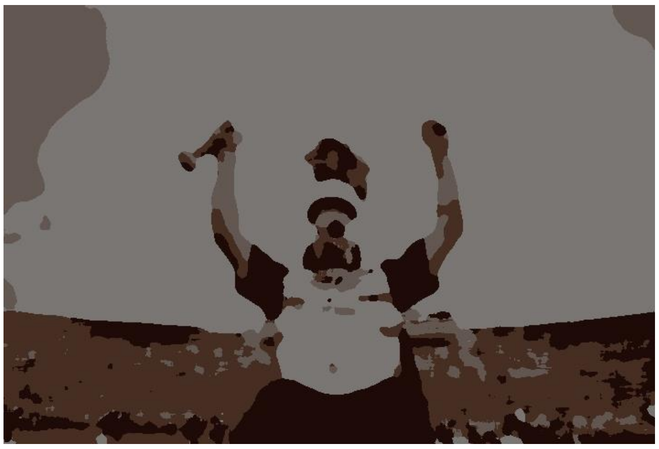

Quantized 512 color image


Output for N = 3


Output for N = 5


Output for N = 7


Output for N = 9


Output for N = 11

IV. Discussion

As N goes larger, oil painting effect becomes very significant and image tends to lose its details. As window size increases, edges start to disappear. For 512 color representation, it has more colors and hence more detail of image is being preserved. Hence it produces much better output visually as it has more colors to choose while applying oil painting effect.

C) SPECIAL EFFECT

I. Abstract and Motivation

This experiment is related to contrast enhancement, histogram matching. Histogram matching allows us to match the input pixel value distribution to a certain given output image histogram. This method asks us to study histogram of input and output histogram and relation between them. It helps us to navigate our way through different color spaces as it enables us to utilize whole dynamic range of pixel values.

By changing histogram distribution, we can play with colors distribution hence apply special filter effect to given input.

Task: Configure the special effect applied to input output pair, check the distribution and color space and apply the same effect to the given input image.

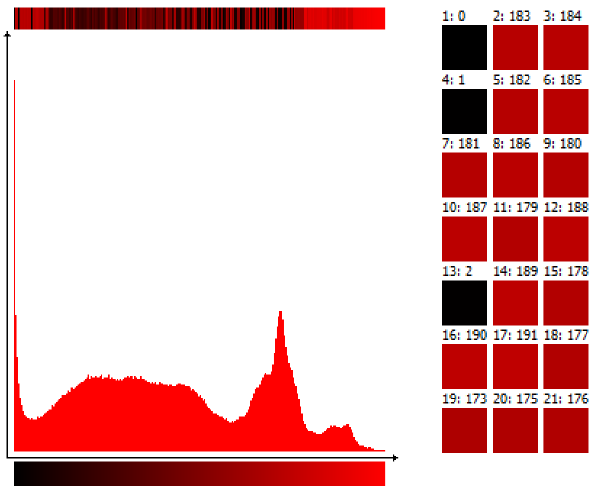
II. Approach and Procedures

Conceptually, we need to find out histogram distribution of input and output image. Then we need to match distribution and check if both images are in same color spaces. We need to create a transfer function that maps the pixels according to distribution of sample image.

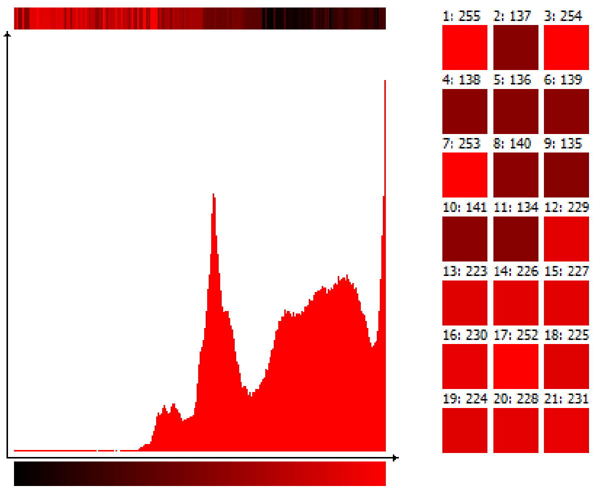
First, we need to check whether pair has same color space or not. If not, then change the color space so that image transversal becomes easy. Then we should check whether pair has same orientation. If not, then perform the required operation.

Algorithm:
STEP 1 - Read input image using ‘fileread’ function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel of original image. Check for color space and orientation.
STEP 3 – Convert RGB image to CMY space. Flip the image about Y axis.
STEP 4 – Check histogram distribution for separate channel. Compare the co-ordinates for histograms and normalize histogram to 0 to 1.
STEP 5 – Multiply the normalized histogram by desired width of distribution. Shift the histogram by required starting point.
STEP 6 - Create 3D array for storing output image Op where new pixel values will be placed in the array.
STEP 7 – Use ‘filewrite’ function to write the output resized image in new 3D array into raw file for displaying the output.

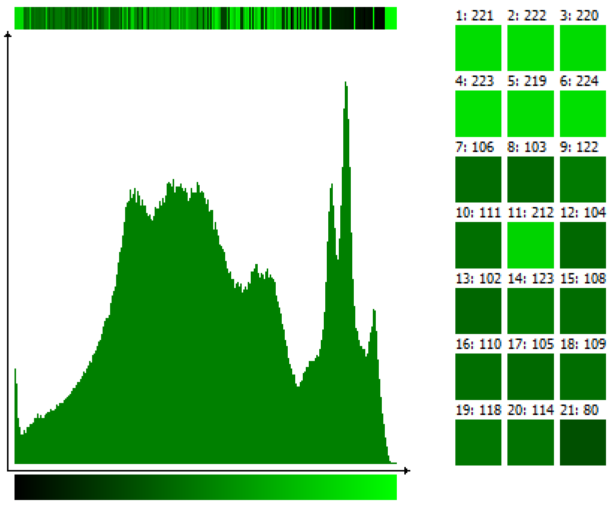
III. Experimental Results



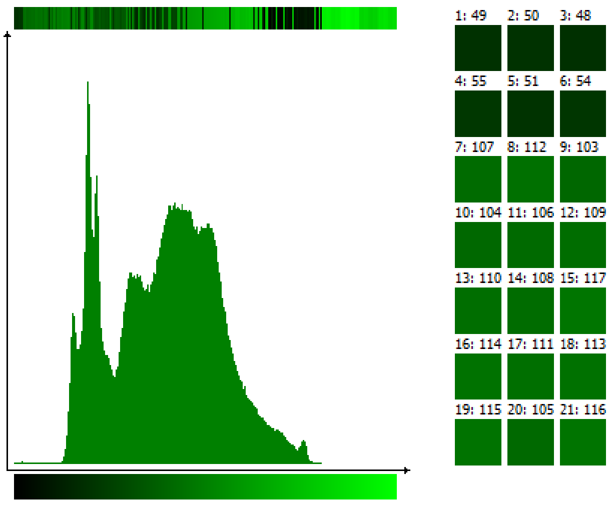
Red channel histogram of input image



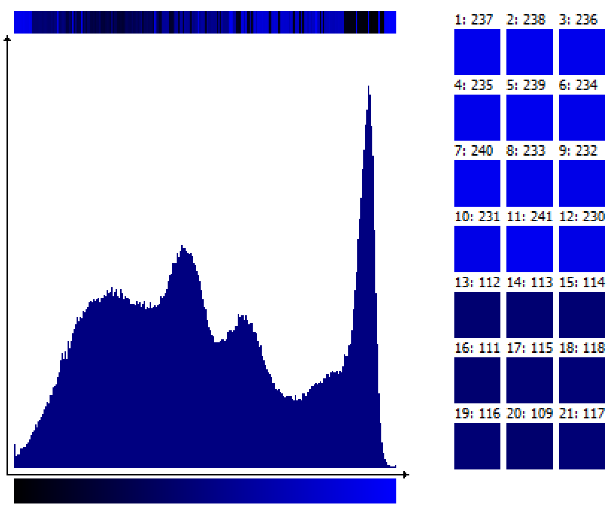
Red channel histogram of output image



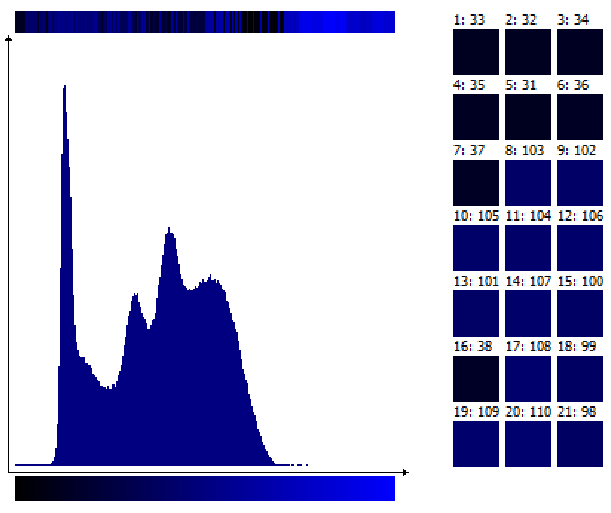
Green channel histogram of input image



Green channel histogram of input image

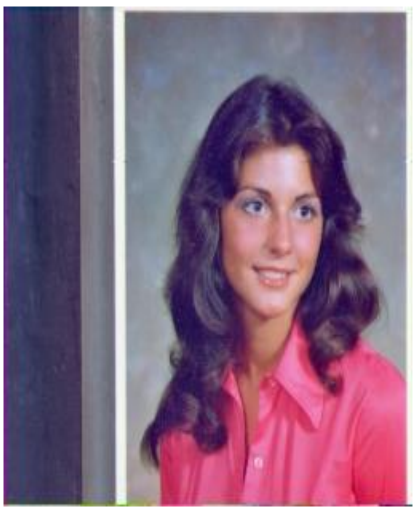


Blue channel histogram of input image

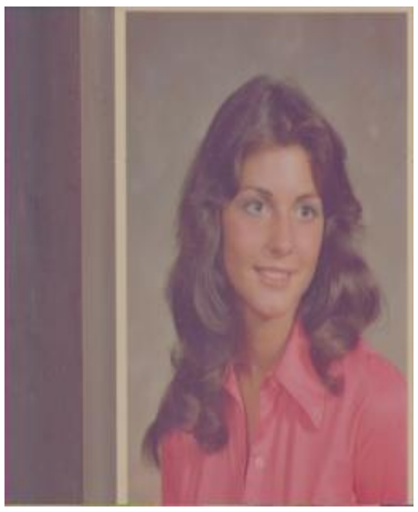


Blue channel histogram of input image

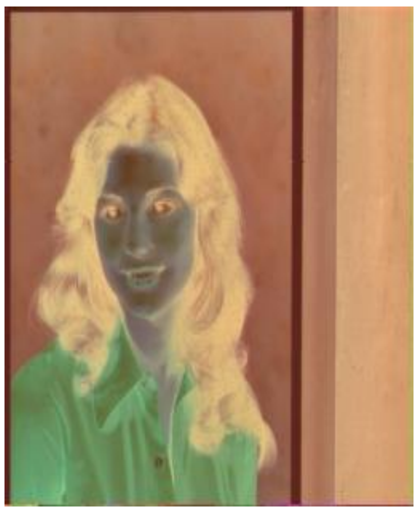
Fig.15 – Special film effect



Original Image



New Histogram Distribution



Output

Fig.16 – Special film effect output

Figure shows transformation of input image to output image. Histogram distribution has been changed as well as image has been flipped along the vertical axis. Then image has been converted into CMY domain to obtain the desired output.

IV. Discussion

Each output image color plane has different histogram distribution. Hence to create special film effect, we have work separately on every plane. Every color has different width of pixel intensities hence 3 planes are separated from input image and shrunk and shifted to get the output. Then image has been converted into CMY domain to obtain the desired output.

PROBLEM 3: NOISE REMOVAL

A) MIX NOISE IN COLOR IMAGE

I. Abstract and Motivation

Noise in image processing is major area of concern as it affects the entire information to be transmitted and processed. Introduction of noise creates distortion. It can enter while capturing the information or while transmitting it might get induced. It is random signal of random magnitude which deteriorates the signal quality. There are various types of noises.

Noise types are gaussian noise, impulse noise, uniform noise etc. There are various filters which are used to remove various types of noises. We will evaluate various filters like mean, median filters by changing parameters and window sizes. Filtering can be performed on different channel depending upon noisy channels or whole image if all channel contains same channel. A moving 2D window is used to capture neighbouring pixels and replace centre pixel with particular logic.

Performance of each filter will be evaluated based on PSNR (Peak Signal to Noise Ratio), SSIM index.

Task: Identify the noises present in input image and apply various filter to remove the present noise. PSNR must be calculated for every channel, for each combination, same filter for all channels, cascading of filter, order in which they are being cascade, window size etc.

II. Approach and Procedures

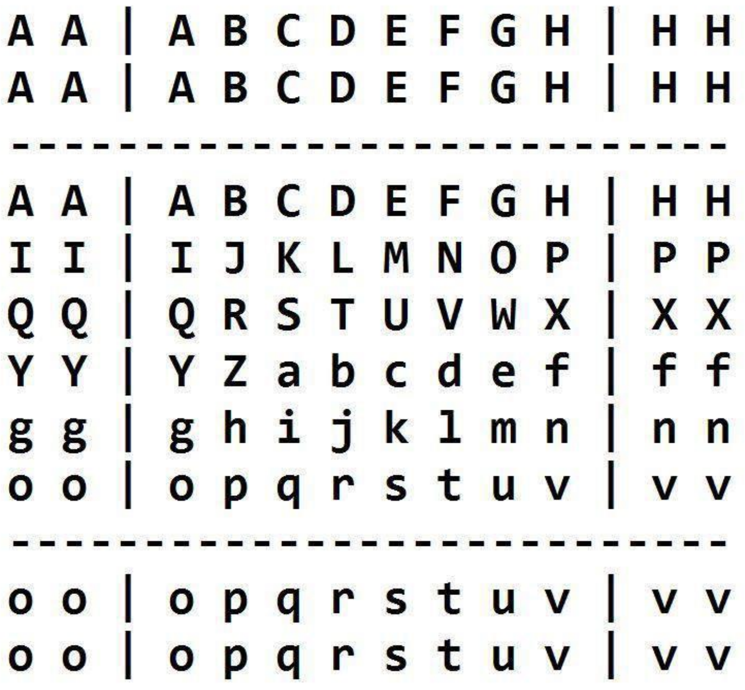
The basic step to apply filter is creating and choosing a window. It transverse all the pixels in the image. Now the new pixels are saved in image.

Given image contains two types of noise- impulse noise and gaussian noise. Channel R, G and B has both gaussian as well as impulse noise. Gaussian noise has bell shaped noised distribution and impulse noise is also known as salt and paper noise. It adds black and white spots all over image making it looks distorted.

For removing impulse noise, median filter of different window sizes is used. For gaussian noise, mean filter of different window sizes is used. R, G, B are extracted from input image and is been applied to filter individually.

To apply filter to the boundary pixels, zero padding is needed first. It can be done using several ways. Since neighbourhood operations will require you to access pixels "off the edge" of the image when you are near the borders, it is necessary to first pad the image. In this manner, you effectively make a somewhat larger image (with padding) and can begin processing the image by centering the kernel in the interior of the image (instead of the upper left corner) [4]. I have used pixel reflection technique stated in figure below. Basic idea behind it is for N dimensional window,

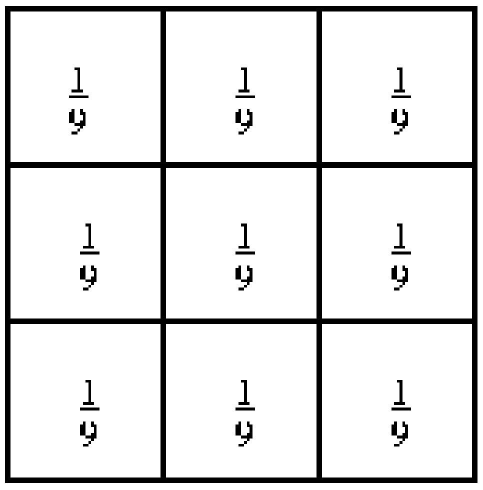
(Window size – 1)/2 edges are being replicated. Hence for window size 5, 2 edges are being replicated. Mean filter used for removing gaussian noise is nothing but averaging filter.



```
A A | A B C D E F G H | H H
A A | A B C D E F G H | H H
---------------------------
A A | A B C D E F G H | H H
I I | I J K L M N O P | P P
Q Q | Q R S T U V W X | X X
Y Y | Y Z a b c d e f | f f
g g | g h i j k l m n | n n
o o | o p q r s t u v | v v
---------------------------
o o | o p q r s t u v | v v
o o | o p q r s t u v | v v
```

Fig.17 – Pixel reflection technique
(Source: http://www-cs.engr.ccny.cuny.edu/~wolberg/cs470/hw/hw2_pad.txt)

Mean filter: The 3*3 mean filter is given by



$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$
$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$

Fig.18 – 3*3 mean filter
(Source: https://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm)

The n*n window for mean filter is given by [4]:

$$Filter\colon \frac{1}{sum\ of\ all\ weights\ in\ window} 1_{n*n}$$

Where,

N – window size. For 3*3 window, 9 elements are considered for averaging.

Median filter: This filter computes the median of elements traversed by n*n window and replaces the centre pixel with median of the traversed elements.

Peak Signal to Noise Ratio (PSNR):

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

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

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

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

Max: Maximum possible pixel intensity $= 255$

Fig.19 – PSNR calculations
(Source: Lecture notes)

PSNR can be calculated for R, G, B channel separately using given formula.

Algorithm:
STEP 1 - Read input image using ‘fileread’ function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel of original image. Check for window size
STEP 3 – Pad the image for the desired window size using pixel reflection.
STEP 4 –Apply median filter / mean filter logic .
STEP 5 – Run 3 nested loops for height, width and bytes per pixel of padded image to apply mean or median filter.
STEP 6 - Create 3D array for storing output image Op where new pixel values will be placed in the array.
STEP 7 – Extract noisy and noiseless channel from original and noisy images and compare the performance of filter using PSNR measure.
STEP 8 - Use ‘filewrite’ function to write the output resized image in new 3D array into raw file for displaying the output.

III. Experimental result

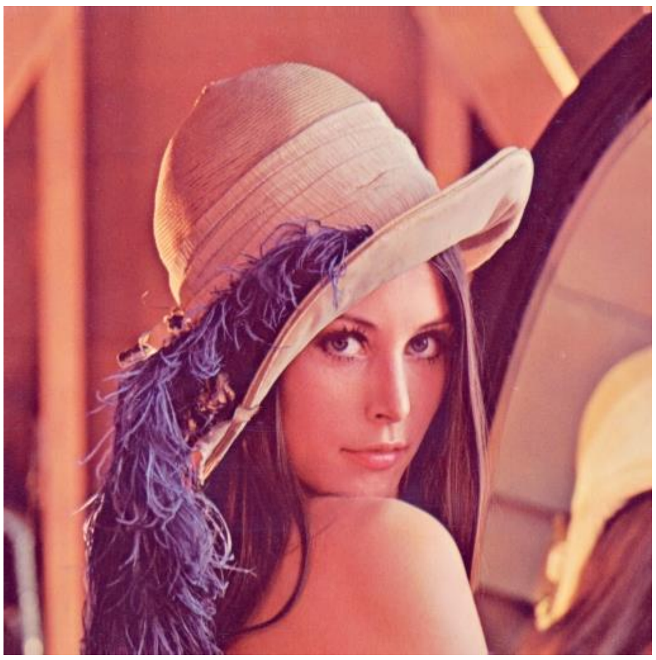
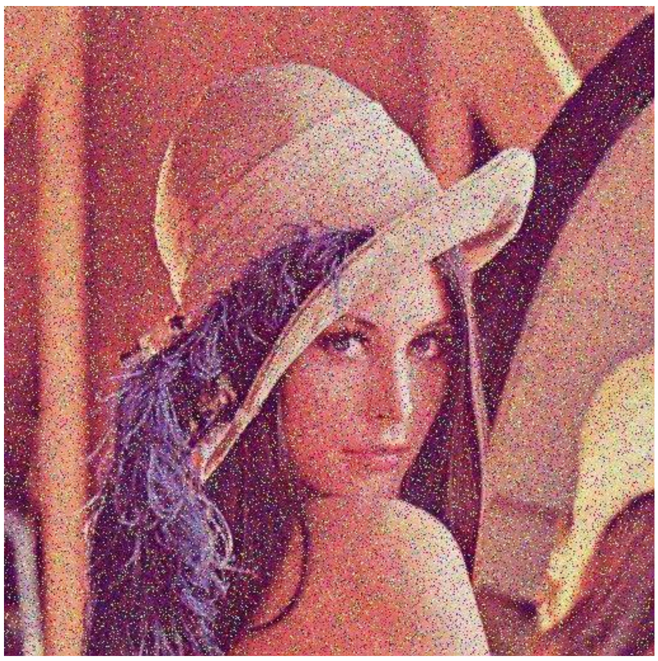
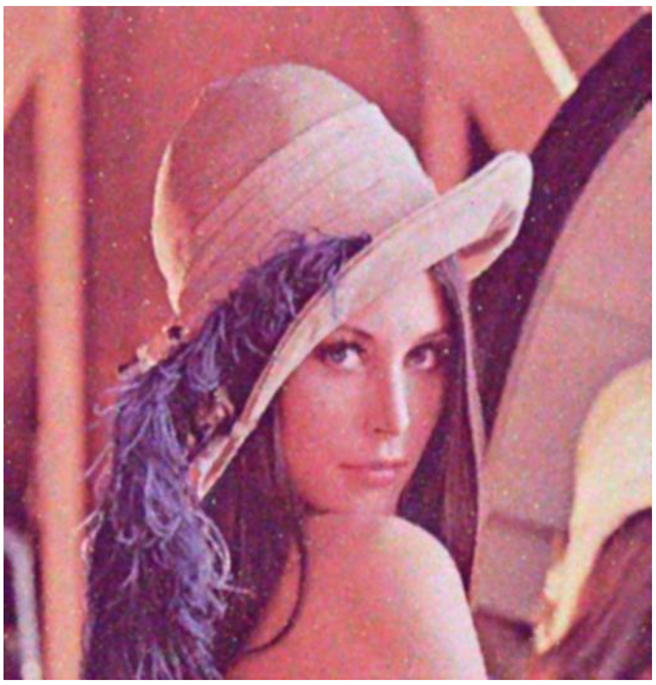
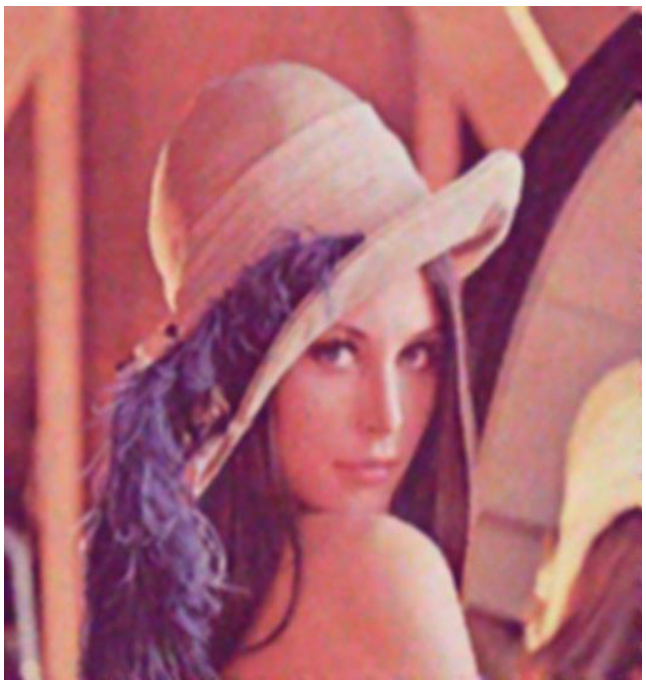



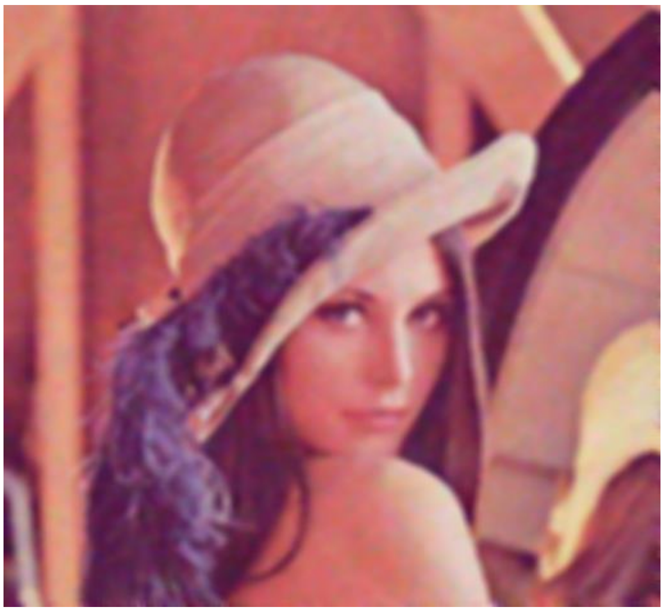
Fig.11 – Input and noisy image



N = 3



N = 5


N = 7

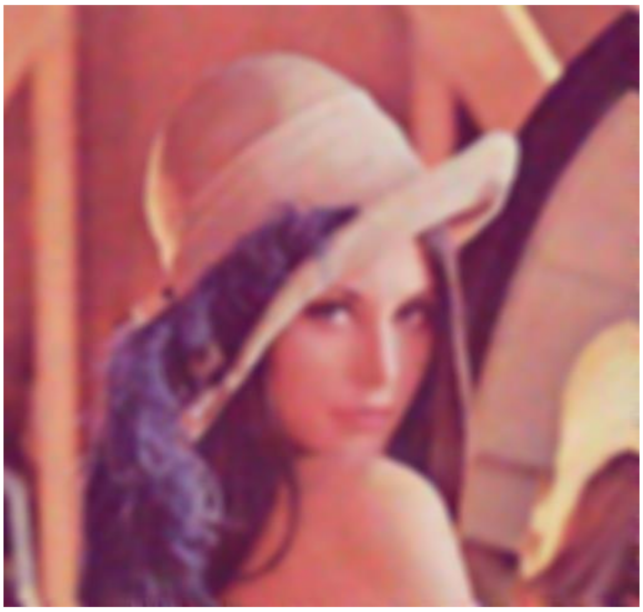

N = 9

Fig.20 – Noisy and output image

PSNR table:

N = 3	PSNR for red channel:24.7448 PSNR for green channel:23.5357 PSNR for blue channel:19.9798
N = 5	PSNR for red channel:**23.0111** PSNR for green channel:**22.236** PSNR for blue channel:**23.2426**
N = 7	PSNR for red channel:21.5849 PSNR for green channel:20.8563 PSNR for blue channel:23.224
N = 9	PSNR for red channel:20.5566 PSNR for green channel:19.8211 PSNR for blue channel:22.7075

IV. Discussion

From the table above, PSNR explains about best window size, cascading or not and in what order you should use the filters. Higher the PSNR, better is denoising. I have made best combination of filters and window size bold output based on PSNR. Best denoising result is obtained when median filter is applied first and then mean filter both for the window size of 5.

When we observe histogram of input noisy image, all channel contains mix noise as histogram tends to be bell curved and has impulse noise of some form. R and G has minor gaussian noise but major impulse noise. Whereas channel B has major gaussian noise and minor noise.

Based on noise types, I choose to apply median filter to tackle impulse noise and mean filter to tackle gaussian filter. Filters can be cascaded in any order but if mean filter is applied first before median, But the problem with applying mean filter that is distributed impulse noise with neighbouring pixels which introduced distortion in image which is difficult to remove using median filter. Hence it does not produce best output denoised image. But if median filter is applied first, it removes impulse noise easily and then mean filter can be used to remove gaussian noise. This combination produced better denoised image. Window size plays an important part in denoising as it might produce better or not so good denoising depending upon neighbouring pixels. As window size increases, more neighbouring pixels are taken into account hence produces better output theoretically.

To improve denoising performance, NLM filter, bilateral filters, block matching can be used and state of the art denoising.

B) PATCH BASED LOCAL PRINCIPAL COMPONENT ANALYSIS

I. Abstract and Motivation

Principal component analysis is dimension reduction algorithm but can also be used for denoising the image [5]. It can be used to filter out random component of noise. It takes an advantage of high spectral correlation between noisy channel of the image. Most of the denoising algorithms uses dictionaries from noisy images or input noisy data sets. PCA has 3 different approach based on use of dictionary and learning pattern – local PCA, hierarchical PCA, global PCA.

PCA can be used improve SNR by identifying and removing principal components of random noise signal in data. However, this reduction is lossy compression technique and we should make sure it does not affect signal components and only removes principal components of random error signals. PCA can be described as "a multivariate technique for reducing matrices of data to their lowest dimensionality using orthogonal factor space and transformation that yield predictions and/or recognizable factors [6]. PCA uses covariance matrix C, data matrix M which is used to calculate covariance matrix and is symmetric and hence C matrix positive semi definite (PSD) matrix as it has non-negative eigen values.

PCA consists of two steps:

- Finding orthogonal basis for noisy image by performing PCA analysis and decompose the noisy patch for each channel.
- Denoise the obtained patch by zeroing all the coefficients of noise in noisy patch using dictionary.

Since we find out orthonormal basis using Gram Schmidt procedure, PCA has an advantage over other denoising algorithms like wavelets as it can be adapted to any image and has high efficiency and low computations are required.

PLPCA is patch based local PCA analysis method which will be used to denoise the image. It uses dynamic localization to build axes and locate noisy patch. A sliding window of size n is traversed through the image extracting local patches to proceed further algorithm. After algorithm implementation, denoised patches are traced back to original image and hence we get denoised image.

II. Approach and Procedure

PLPCA method is based on dynamic localization of axes to find patches. By selecting axes of high variance, PCA retrieve most frequent patterns of image [7]. Due to dynamic localization of patches, PLPCA has an advantage over PGPCA as it presents several PCAs on subset patches to reduce the variance. It results into basis not only adapted to the image but also image containing patch of interest. Due to sliding window mechanism, process is repeated again and again and hence might result into overfitting increased computations.

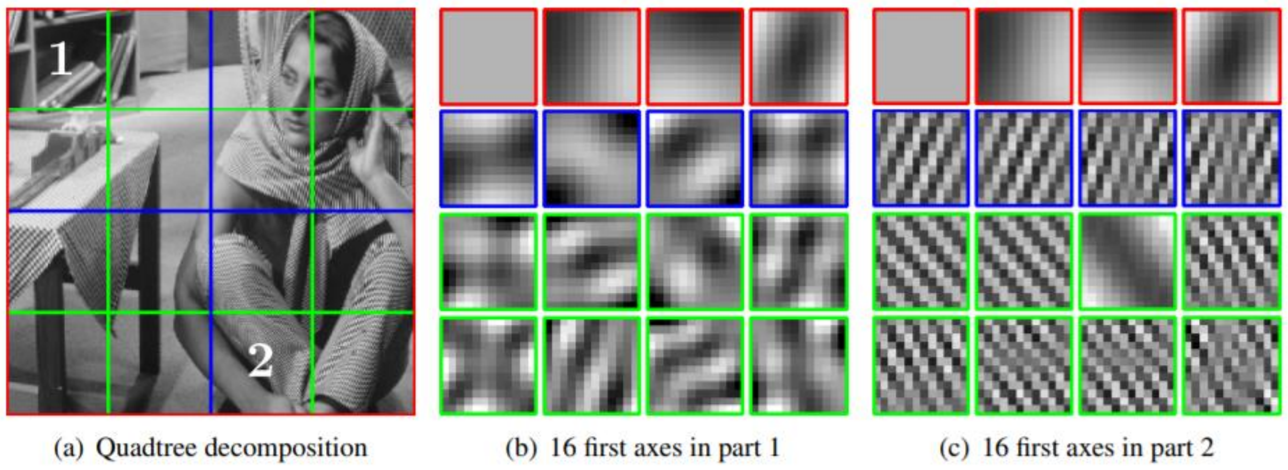

1
2
(a) Quadtree decomposition
(b) 16 first axes in part 1
(c) 16 first axes in part 2

Fig.21 – An image and its 16 first axes obtained over two stacks extracted respectively in two different leaves of the quadtree decomposition.
(Source: [7])

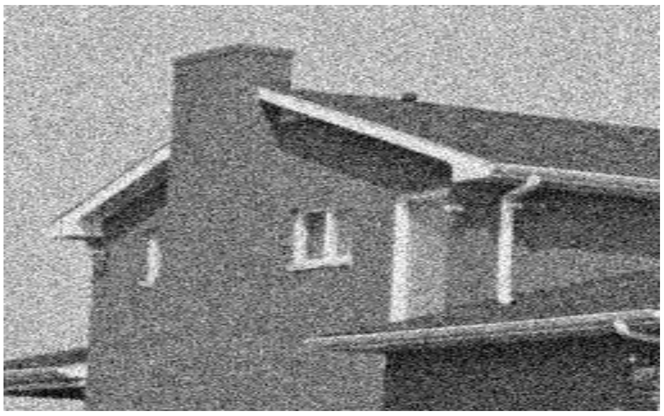
Parameters: sliding step δ = (window size -1) /2 – divides computational time by $\delta^{2.}$

hW: half size of the searching zone - ideally it should be small but not small (close to 0) generally ranges from 5 to 15.

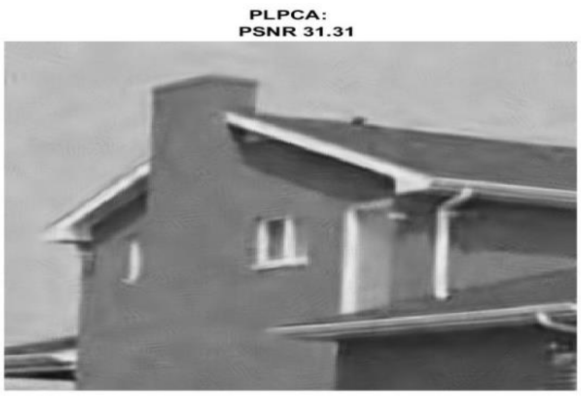
hP: half size of the patch – it should relative to window size and is generally from 5 to 13

factor_thr: factor in front of the variance term for hard thresholding the coefficients – generally should be more than 1.50

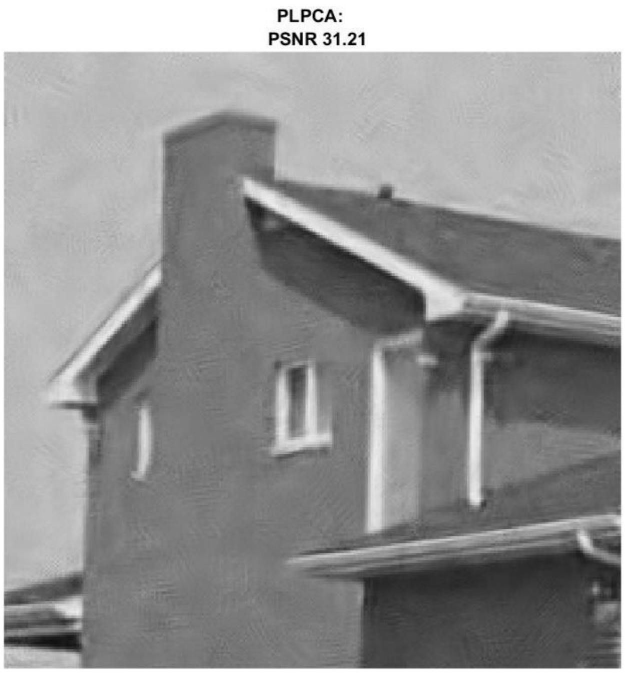
III. Experimental Results

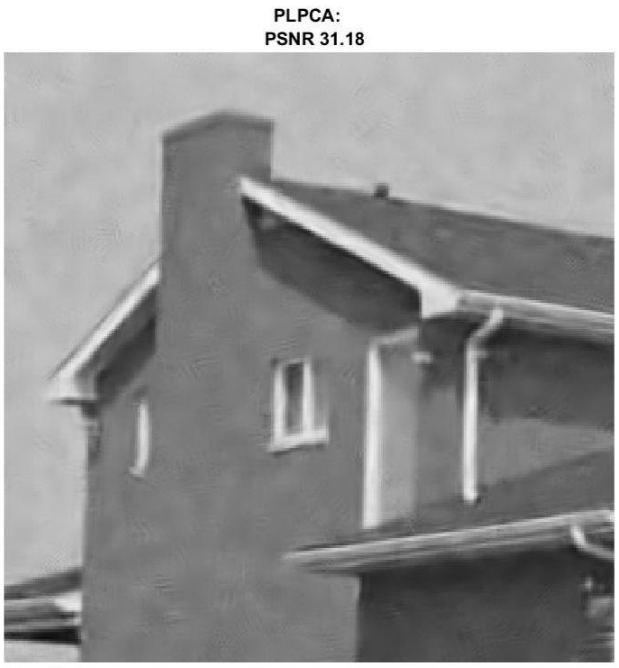


Noisy image



Output Image with PSNR = 31.31 dB





Output Image with PSNR = 31.21 dB Output Image with PSNR = 31.21 dB

Fig.22 - Various output images using PLPCA approach

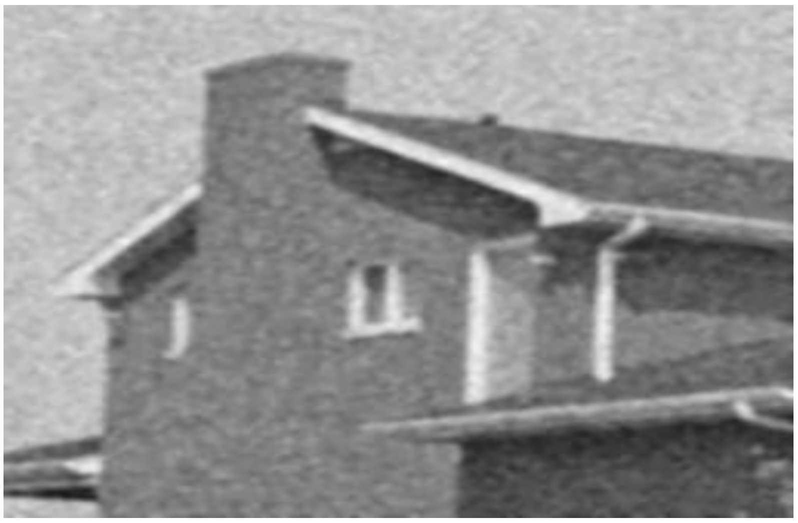


Fig.23 – Output image using Median and Mean filter PSNR=24.4822

Figure shows various output images various parameters. Most of the noise is successfully removed.

For the same image, if we use median and mean filter, PSNR comes 24.4822.

Higher the PSNR, better is the denoised image.

Table for PSNR values using different parameters:

hP	factor_thr	hW	PSNR	Time taken to execute (sec)
9	2.75	12	31.21	2.29

7	3	12	31.18	1.61
9	**3**	**12**	**31.31**	**2.30**

IV. Discussion

PLPCA has several advantages over other PCA algorithms as well as various denoising algorithms like NLM, mean, median filters, etc. It has high accuracy, less time consumption although more computation complexity due to repeatedly carrying local patch denoising. For given parameters, third entry in my table gives best PSNR. I have marked them bold. It has best window size as well as thresholding.

For given parameters, image has been successfully denoised but loses its sharpness and edges looks little smudgy. As window tends to be smaller, denoising gets poor.

Comparing output using PLPCA and median and mean filters, PLPCA output is better in two senses. Output image has minimal noise compared to the other image and PSNR value is greater than other. Also, image looks much shaper, edges are crisp hence PLPCA works better algorithm for denosisng.

C) BLOCK MATCHING AND 3D (BM3D) TRANSFORM FILTER

I. Abstract and Motivation

A sparse image representation is image representation using optimal features and information. Block matching is novel denoising algorithm based on enhanced sparse representation in transform-domain. The enhancement of the sparsity is achieved by grouping similar 2D image fragments (e.g. blocks) into 3D data arrays which we call "groups" [8].

Collaborative filtering is special type of algorithm which is used while working in 3D groups. Input noisy image is divided in various 2D blocks and similar blocks are stacked together forming one 3D groups. We get hard threshold transform coefficients after applying transform. Now converting groups again back to separate 2D blocks gives a basic estimate of an image. By reducing noise up to great extent, collaborative filtering reveals very fine details shows by stacked 2D groups. The filtered blocks are returned to the original position. Just like PCA, overlapping of blocks occurs in this algorithm and aggregation of block output is taken to produce better estimate. Even better improvement is obtained using special Weiner filter.

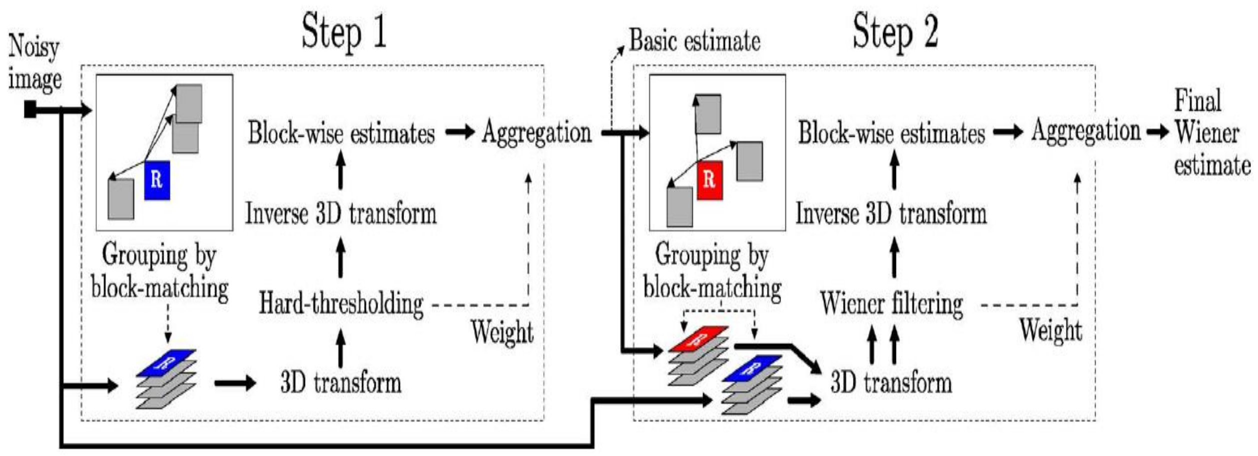


Fig.11 – BM3D algorithm

II. Approach and Procedure

We first to convert the input raw image to 2D grey image using given code in assignment.

Algorithm:

STEP 1 - Read input image using 'fileread' function and get height, width and bytes per pixel from user.
STEP 2 – Run 3 nested loops for height, width and bytes to access each pixel of original image. BM3D code will extract all the color planes from input image.
STEP 3 – Initialize 2 arrays for weight and buffer match the respective blocks within the image. Let the co-ordinate of blocks having smallest distance remain there.
STEP 4 – Use hard-thresholding and aggregation to denoise obtained 3 D group.
STEP 5 – Produce Weiner filter estimate and after done with computing initial estimate, compute final estimate to find the output blocks.
STEP 6 - BM3D .m will save output image in grey scale image according to the code.
STEP 7 – Repeat for all color planes and save all 3 planes in 3D array.
STEP 8 - Use 'filewrite' function to write the output resized image in new 3D array into raw file for displaying the output.

III. Experimental Result

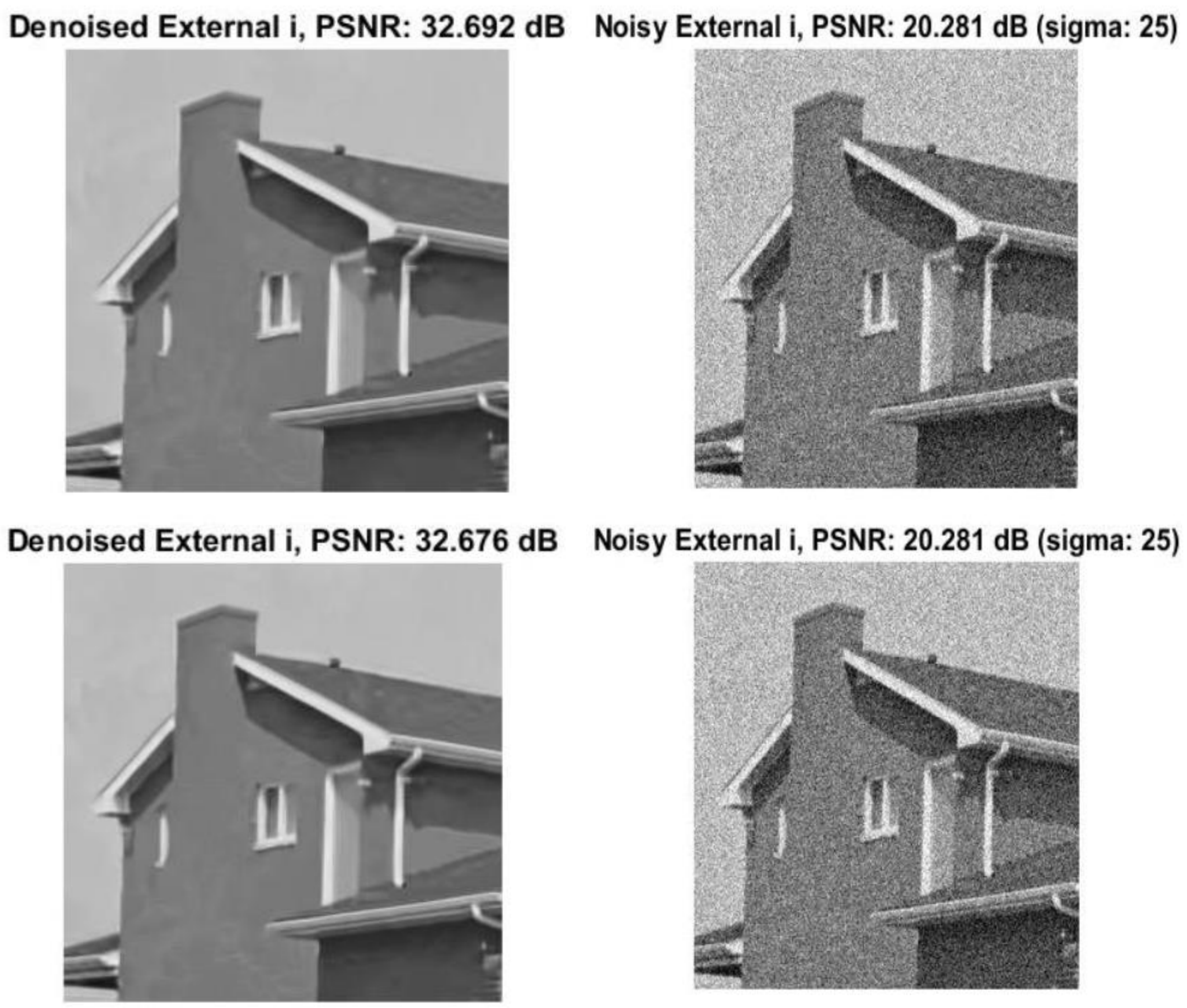

Denoised External i, PSNR: 32.692 dB
Noisy External i, PSNR: 20.281 dB (sigma: 25)
Denoised External i, PSNR: 32.676 dB
Noisy External i, PSNR: 20.281 dB (sigma: 25)

Fig.24 – BM3D output image and corresponding input noisy image

For hard thresholding, I changed 2 parameters mainly Nstep and N2.

Nstep	**N2**	**PSNR (dB)**
5	16	32.64
3	**32**	**32.69**
3	16	32.67

IV. Discussion

Sigma is taken as 25 which produces very good denoised image. BM3D gives very shape edges and very less distortion is produced. It is collaborative algorithm where from input image 2D blocks of similar patterns are grouped together, filter is applied to this group and Weiner estimate gives output 2D blocks which are then placed in original image.

BM3D is both spatial domain as well as frequency domain filter. It involves partition of input image into similar blocks in spatial domain and Weiner filter is applied and estimated in frequency domain to remove the noise.

Generally, higher PSNR indicates better denoising but often undermines the blurring effect. So, an image which is blurred output might have high PSNR. Window size alters sharpness of image as it generally loses it high frequency content. Comparing performance of BM3D and PLPCA, BM3D produces better output in both sense. One being PSNR is greater than PLPCA and image has very sharp edges and there is no distortion.

Hence BM3D might be considered as better algorithm to denoise due to novel collaborative algorithm but there might be certain cases where PLPCA works better than BM3D.

REFERENCES:

[1] https://graphicdesign.stackexchange.com/questions/60/what-is-the-difference-between-cmyk-and-rgb-are-there-other-color-spaces-i-shou

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

[3] https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf

[4] http://www-cs.engr.ccny.cuny.edu/~wolberg/cs470/hw/hw2_pad.txt

[5] Lecture notes, slide and homeworks

[6] https://journals.ametsoc.org/doi/full/10.1175/JTECH1906.1

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

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