HOMEWORK #1

Issued: 8/26/2016 **Due: 9/18/2016**

By: Hariprasad Ravishankar **USC ID: 8991379333**

Problem 1: Basic Image Manipulation

1.1. Abstract and Motivation

Cropping, Resizing and Color Space Transformation are some of the most commonly used operations on images and are useful in wide scenarios and requirements. For example, resizing operation is useful in situations when the image is too small to be viewed or too big to be used in presentations and documents. In such situations it is necessary to resize the image smaller or larger to suit the need. Further, color space transformations are widely used in printing and image manipulation software such as Photoshop and Paint.

The CMY(K) color space is used in printing as the colour perceived by our eyes is the color not absorbed by the printed pigment, or in other words the component of R,G and B is used for printing.

The HSL(Hue, Saturation , Luminance) color space is often used in applications and editing tools [6] to make it easier to choose color instead of identifying the R, G and B values to obtain colour. Hue and Saturation define the chroma of the image whereas L defined the luminance. Hue gives the exact color of the image and has a range from 0° to 360°. Saturation defines the purity of the image and Luminance the brightness of the image.

Resizing is implemented using Bilinear interpolation wherein pixels in the output are mapped from the source image by solving the linear equation in both x and y directions, by knowing the ratio of the input and output images.

The color space transformation is performed by using the definition of CMY and HSL in terms of RGB values.

1.2. Approach and Procedures

Cropping

The cropping function first takes in two parameters (x1, y1) and (x2, y2) corresponding to the top-left and bottom-right locations in the source image. These two coordinates define the cropped image size and shape.

To crop the image, we declare a new image with dimensions (x2-x1+1) * (y2-y1+1)*3. We then copy all pixels across all three channels from the original image to the new image defined by the coordinates.

Resizing

The resizing operation takes as input the size of the resized image. Bilinear interpolation is used to estimate the pixel values of the resized images by mapping back to the closest pixels in the original image.

If R represents the ratio of old image size to new image size, we have

$$R = \frac{Old\ image\ size}{New\ image\ size}$$

For any pixel I in the new image with coordinates (i, j), we can estimate the corresponding pixel in the original to be (x,y) defined by

$$y = j * R \text{ and } x = i * R$$

But, x and y needn't be integers as R may be a fraction. Thus we can calculate the closest integer to x and y to be x' and y'. We then define parameters a and b as follows

$$a = x - x' \text{ and } b = y - y'$$

Knowing parameters x', y', a and b we can estimate the value of pixel I using the formula.

$$I(i,j) = (1-b)[(i-a)F(x',y') + aF(x'+1,y')] + b[(1-a)F(x',y'+1) + aF(x'+1,y'+1)]$$

Bilinear Interpolation is Linear Interpolation implemented in both x (horizontal) and y(vertical) directions to best estimate the values of pixels in the resized image.

Color Space Transformation
CMY Color Space

The Cyan – Magenta – Yellow – (Black) Color space transformation is achieved by replacing the R, G, and B values in the image by their compliments defined by.

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

To achieve this, we first normalize the R, G and B values to range 0 – 1 by dividing with 255. The computed values are then renormalized by multiplying with 255.

HSL Color Space

The Hue- Saturation – Lightness color space transformation is achieved as follows.

The R, G , B values are normalized to range 0 – 1 by dividing with 255. The following formula is then used to compute H, S and L values.

$$M = \max(R,G,B)$$
$$m = \min(R,G,B)$$
$$C = M - m$$

$$H = \begin{cases} 0 & C = 0 \\ 60\left(\frac{G-B}{C} mod 6\right) & M = R \\ 60\left(\frac{B-R}{C} + 2\right) & M = G \\ 60\left(\frac{R-G}{C} + 4\right) & M = B \end{cases}$$

$$L = \frac{M + m}{2}$$

$$S = \begin{cases} 0, & L = 0 \\ \frac{C}{2L} & 0 < L < 0.5 \\ \frac{C}{2-2L}, & otherwise \end{cases}$$

In order to compute max and min values, the C++ standard library's *max()* and *min()* function commands are called, which is available under the *math.h* header file. To compute mod, *fmod()* function available under the *math.h* header file is used to compute floating point mod values.

Three new Image variables for H, S and L with dimensions identical to number of rows and columns in the original image are created.

Since, R, G, B values have been normalized, the values of L and S are renormalized by multiplying with 255 before copying into data arrays after direct application of the formula for each pixel. Further, since H can assume values only between 0 and 360, the value of H is first normalized by dividing by 360 and is then renormalized to range 0-255 by multiplying with 255.

1.3. Experimental Results

Shown below are the results for Problem 1 a) Cropping.

(x1, y1) and (x2, y2) are given as (55,85) and (184,214) for *Anna (130 x 130)*
(x1, y1) and (x2, y2) are given as (35,65) and (194,194) for *Rebel (160 x 130)*

130x130

160x130

Figure 1: Cropped Images of *Anna* and *Rebel*.

2. Face Resizing

Shown below are the results from Problem 1 a) Resizing.

100x100

200x200

300x300

Figure 2a: Resized images of cropped face image of *Anna* sized 130 x 130

100x100

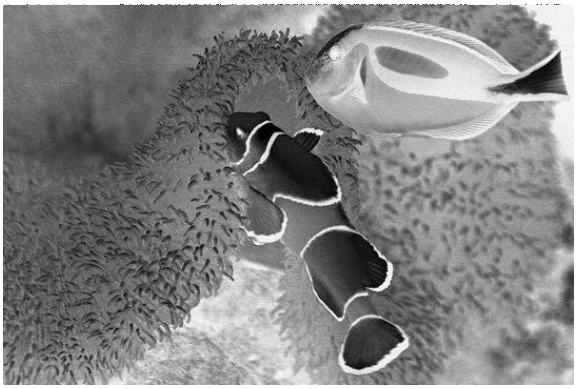
200x200

300x300

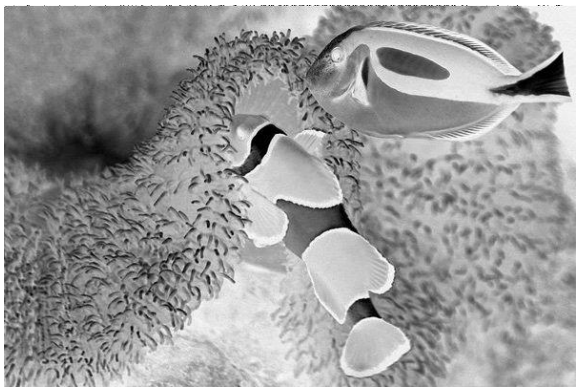
Figure 2b: Resized images of cropped face image of *Rebel* sized 130 x 130

Problem 1 b) CMY(K) Color Space

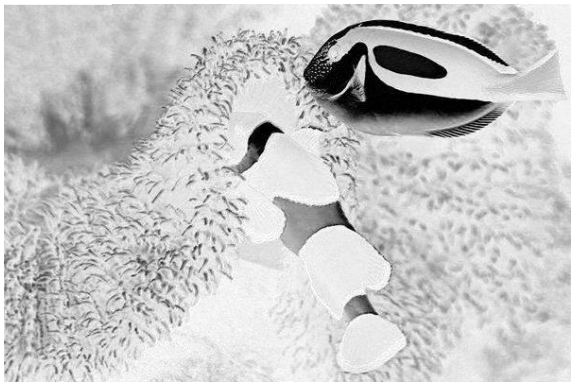
Shown below are three grayscale images corresponding to the Cyan, Magenta, and Yellow channels, first for *Clownfish* and then *Octopus*.



Cyan Channel

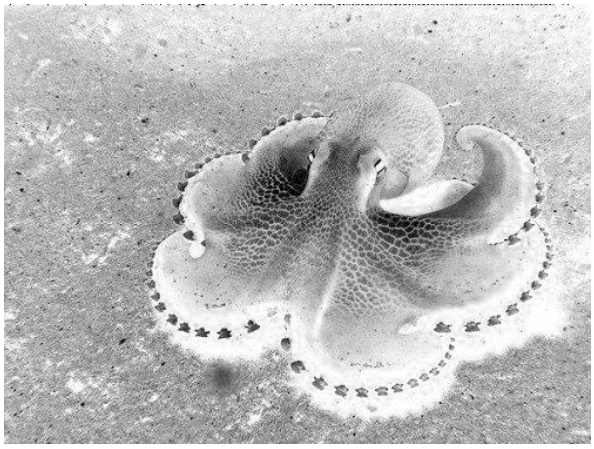


Magenta Channel

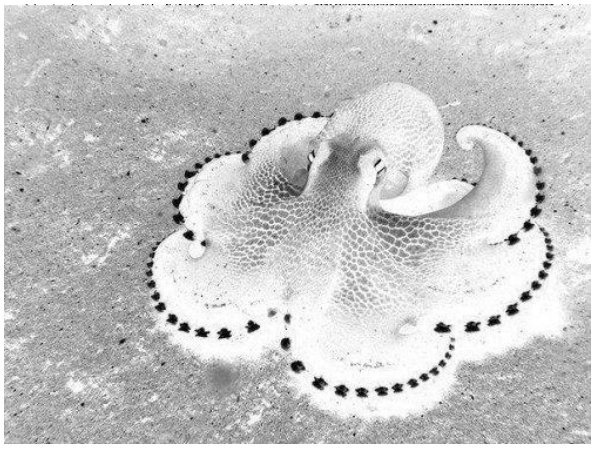


Yellow Channel

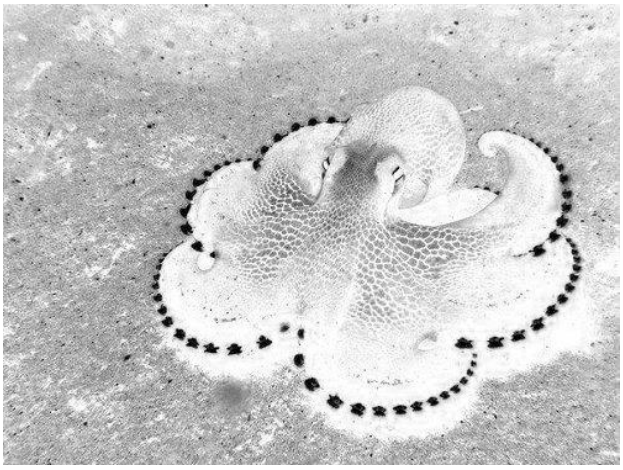
Figure 3a: Cyan, Magenta and Yellow Channels od Clownfish image



Cyan Channel



Magenta Channel



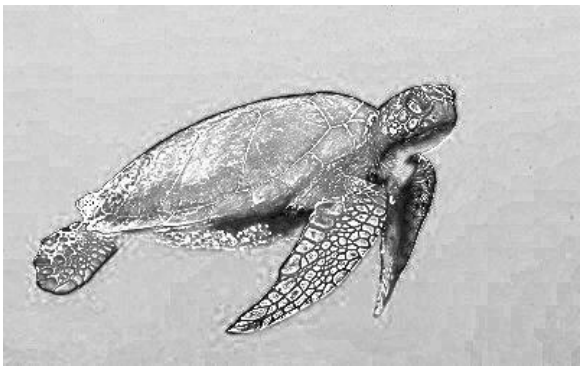
Yellow Channel

Figure 3b: Cyan, Magenta and Yellow Channels od Octopus image

1b) HSL Color Space Transformation

Shown below are three grayscale images corresponding to Hue, Saturation and Lightness channels for *Turtle* and *Jellyfish* images.

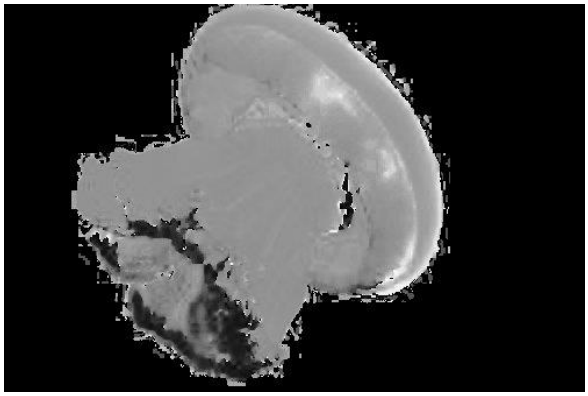
Hue channel



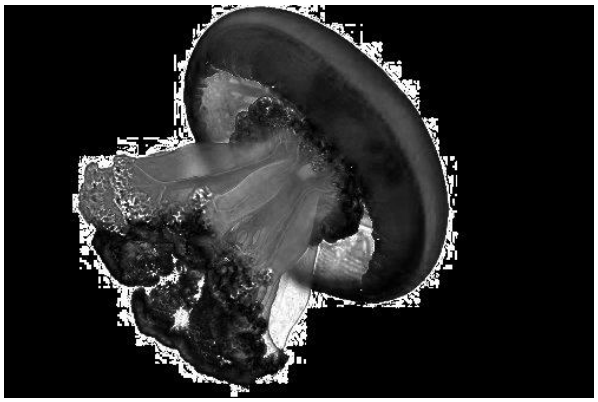
Saturation channel

Lightness/Luminance channel

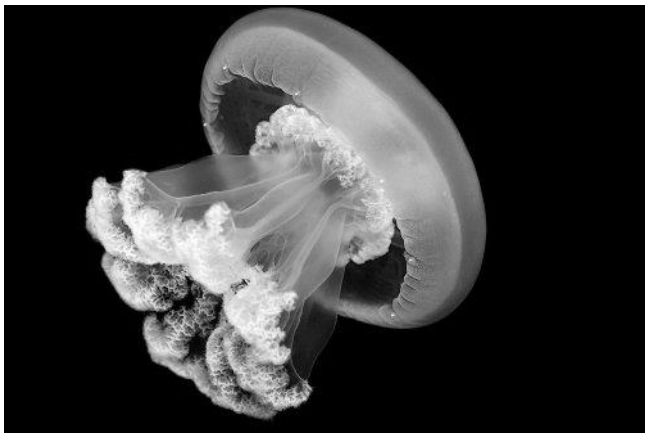
Figure 4a: Hue, Saturation and Lightness channel grayscale images of *Turtle* image



Hue channel



Saturation channel



Lightness/Luminance channel

Figure 4a: Hue, Saturation and Lightness channel grayscale images of *Jellyfish* image

1.4. Discussion

Figure 1 shows two cropped face images of Anna and Rebel, one of size 130x130 and the other of rectangular size 160 x 130. Thus, these images show that given two coordinates representing top-left and bottom-right corners, a cropped image can be created.

Figure 2 a and b shows the results of resizing face images of Anna and Rebel which of initial size 130 x 130. It can thus be seen that the smallest image of size 100 x 100 looks very clear and crisp showing high resolution, whereas the resolution drops in the 200 x 200 and 300 x 300 resized images. This is primarily because when the resize ratio is greater than 0, the pixels can easily be better approximated. When is ratio is less than 0, we have fewer pixels from which we have to estimate a lot more pixels leading to a poorer solution image.

The quality of the resized images can be further improved by using Biquadratic or Bicubic interpolation instead of bilinear interpolation as the pixels can be better estimated. However, these dramatically increase computation time which is less desired.

Figure 3 a and b shows the result of Cyan, Magenta and Yellow grayscale channels of Clownfish and Octopus images respectively. Since Cyan, Magenta and Yellow channels are the components of Red, Green and Blue channels, it can be seen that in regions of the Clownfish which are of reddish shade, such as on the skin of fist of clown fish, the Cyan channel represents darker values. Similarly, the clownfish having a blue skin tone has a darker shade in the Yellow channel as it corresponds to the component of Blue. Further, white color, which is a combination of Red, Green and Blue values appears dark in all three images.

In 3b, the octopus has white spots around its tentacles contributing to darker shade in yellow and magenta channels. Similarly values that represent red intensity appear darker shade in the Cyan channel. The black regions bordering the tentacles appear white as white is the complement of black.

Figures 4 a and b represent the grayscale images of Hue, Saturation and Lightness channels of Turtle and Jellyfish images. Hue represents the actual color in the image and has a range from 0 to 360 degrees where 0 represents red primary, 120 represents green primary, 240 represents blue primary. When normalizing to range 0-255, we would expect pixels in the green and red regions to appear darker as these will have lower grayscale values after renormalization.
Eg. 120 which represents green would have an approximate value of 84 thus appearing darker whereas 240 which represent blue primary will have an approximate grayscale value of 170, appearing gray.

Thus it can be seen that the green regions on the turtle appear darker than the background which has higher blue values causing it to appear gray in the Hue channel. In the Jellyfish image, the black background represents all three R,G,B values to be 0 and thus its representation in the Hue channel is also black. Further, Since the white regions of the Jellyfish represent R,G,B values of 255, 255 and 255, the hue channel has a Gray-ish tone corresponding to an average. The Jellyfish image also shows constant tone in the hue channel thus denoting that there is no transition in color and is of constant color.

Saturation represents the purity of the color. Thus purer the color, the higher is pixel value, the brighter is its grayscale representation. In Turtle image, the background and the shell contains pure color and hence appear white in the grayscale image.

In the Jellyfish image, the white body of the Jellyfish is the mix of colors from the Red, Green and Blue channels and hence has lower purity. This contributes to its darker shade in its grayscale representation.

Problem 2: Histogram Equalization

2.1. Abstract and Motivation

In many images, most of the pixel values may fall between a very small range of values. This sometimes leads to images appearing too bright or too dark with very less intensity change between the foreground and the background. This is frequent in low light or over-bright scenes and are undesirable. For example, in X-ray images is it highly desirable that the bone structures and clearly differentiable from the background. In such situations we may want to increase the contrast of the image by uniformly distributing pixel values across all intensity values to obtain a uniformly distributed histogram. Such an operation will boost the contrast of the image and improve the perceptual quality of the image.

The two methods implemented in the first part achieve this and produce an image that distribute the pixels uniformly across all intensity values. The result on test with two grayscale images and one color image are presented in later section.

The motivation being the method A is to generate the Cumulative Distribution Function from the Probability Distribution of the pixels in the image. This will then be multiplied by 255 as we desire to distribute the histogram between a range 0-255. The obtained result is floor rounded and gives us an intensity wise mapping scheme to enhance contrast. For colour images, this process is carried out across all three channels.

Method B is a bucket filling method where each intensity value starting from 0 is orderly filled with pixels till a threshold (uniform level) is achieved. This then generate the intensity-wise mapping scheme.

In addition to contrast enhancement, we may also desire to match the histogram of one image to another image or to another distribution to simulate the perceptual effect of one image on the other. This can also be carried out using Method A and Method B techniques. The special effect can be achieved by equalizing the two images independently and then relating the two histograms to generate a lookup table for pixel intensity values.

Histogram matching is also used to redistribute pixels to match a Gaussian function through the bucket filling methods, where the bucket size for each intensity value is computed from the Gaussian function.

2.2. Approach and Procedures

Method A: Transfer-function-based histogram equalization method

The probability of occurrence of pixel of every intensity value is calculated for the image.

Ex. Let there be n_i pixels with intensity i, and total number of pixels in image be n. Then the probability of occurrence pixel with intensity i is given by

$$p(i) = \frac{n_i}{n} \text{ where } 0 \leq i < 256 \text{ for an 8-bit image}$$

Since a PDF will have values only between [0,1], we normalize by dividing by the total number of pixels in the image. We then calculate the cumulative distribution function given by the formula

$CDF(i) = \sum_j^i p(i)$ where i represents the intensity value

We wish to transform the image with an arbitrary CDF to a new image with a linear cumulative distribution function. To perform this transformation, we multiply the value of CDF for every intensity value with 255. As the new intensity levels can only have integer values, the floating point values are floored using the *floor()* function available in C++. This gives us new intensity values j ranging from 0-255 for every intensity level i and a mapping scheme is then generated.

$map(i,j), j = floor(CDF(i) * 255)$ for $0 \leq i < 256$.

Then all pixels with value i are replaced with their correspondingly mapped j into a new Image to generate a contrast enhanced image.

For a colour image, this process is repeated for all three channels, R, G and B separately.

Method B: The Cumulative-Probability-Based Histogram equalization method

In this approach, we first calculate the number of pixels in each bucket to generate a uniform distribution.
If n is the total number of pixels in the image, and we wish to uniformly distribute it into 256 buckets, the total number of pixels in each bucket is equal to $n/256$.

Next, we iterate in an orderly manner from 0 to 255, filling n/256 pixels in each bucket, while scanning through each channel of the image. While filling, a map is created that stores the pixel intensity value being copied and bucket number (which ranges from 0 to 255).

Since n/256 may be a fraction, we use the floored value and fill the last bucket with extra pixels to compensate for the fractional loss. Thus, using the generated map, we replace the value of every pixel with its mapped bucket index, generating a contrast enhanced image. This process is repeated for all three channels in color images.

Special Effect via Contrast Manipulation

In this problem, we match the histogram of an input image with that of a reference image which in this case is the *Skyline image*. This is a problem of histogram specification. The algorithm used is as follows.

Step 1:

Let the input image be x and the reference image be y. We calculate the probability density of occurrence of pixel with intensity i , across all intensity values for both images.

Let p_x and p_y represent the probability histogram generated normalized to [0,1].

Step 2:

We proceed by equalizing the histogram of image x and image y by computing the CDF. We can represent this as follows.

For image x, (input image)

$CDF_x(i) = \int_0^i p_x(i)d(i)$

Similarly for image y, (reference image)

$CDF_y(j) = \int_0^j p_y(j)d(j)$

Step 3:

In this step, we build an unordered map or a lookup table to relate the above two mapping. We obtain an output level j for every input level i such that $CDF_y(j)$ matches closest to $CDF_x(i)$.

Ie. If $CDF_x(i) \leq CDF_y(j)$

map(i,j)

else{

increment j till $CDF_x(i) > CDF_y(j)$

check if $CDF_y(j) - CDF_x(i) > CDF_x(i) - CDF_y(j-1)$

map(i,j-1)

else

map(i,j)

}

Step 4:

Using the mapping function, replace pixels with value i in the input image with its corresponding map j.

Step 5:

Repeat the above steps for all three channels, namely R,G and B.

We will then observe that the newly generated image will have a histogram that matches the nature of the reference image.

The reference for this approach can be found in [3]
http://fourier.eng.hmc.edu/e161/lectures/contrast_transform/node3.html

Histogram Transformation

In this problem we wish to match the histogram of a given input image to a Gaussian function with specified mean and standard distribution. The Method B, Bucket filling method is used to implement the histogram matching method.

The bucket is an array having 256 elements each representing an intensity level. The threshold level for the bucket at every intensity level is computed by evaluating the Gaussian function with parameters as intensity level i, mean and standard deviation.

We normalize the threshold value in each element of the bucket to have domain (0,1) by dividing with the sum across all elements.

i.e for every intensity level i, let G(i) be the Gaussian kernel value.
We compute $\mathrm{G(i)} / \sum_{i=0}^{i=255} G(i)$ for all intensity level.

Let Threshold(i) = $\mathrm{G(i)} / \sum_{i=0}^{i=255} G(i)$. Thus, Threshold(i) will have domain (0,1).

Next, we compute n*Threshold(i) for all i = [0,255] intensity values. Where n = total number of pixels in each channel..

This will give the approximate number of pixels to be filled at every bucket with intensity level i such that the sum of the elements in the bucket is equal to the total number of pixels in the image.

We then proceed by filling the buckets determined by the threshold limit obtained for each intensity value. This will ensure that the distribution has a Gaussian function with given mean and standard deviation.

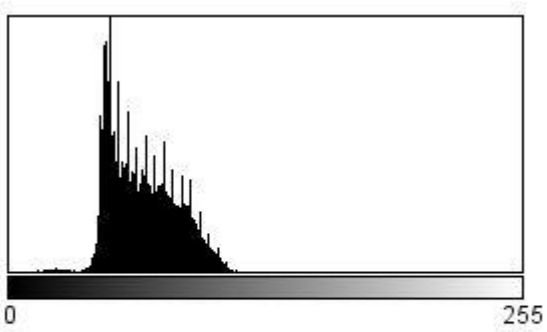
We repeat this for all three channels.

The histogram plots of all images have been obtained using the Histogram plot tool in ImageJ.

2.3. Experimental Results

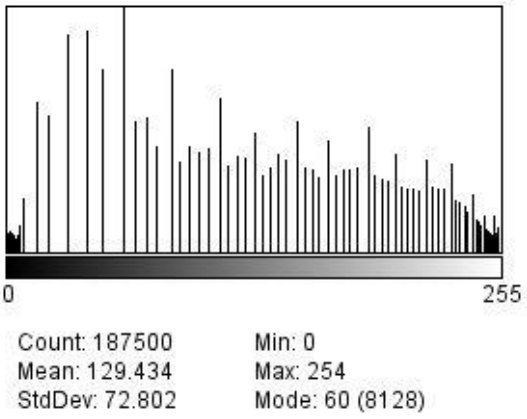
Shown below are the results obtained for contrast enhancement of beach_bright and beach_dark images using Method A Transfer-function-based histogram equalization method, and Method B The Cumulative-Probability-Based Histogram equalization method.

f) Original Input



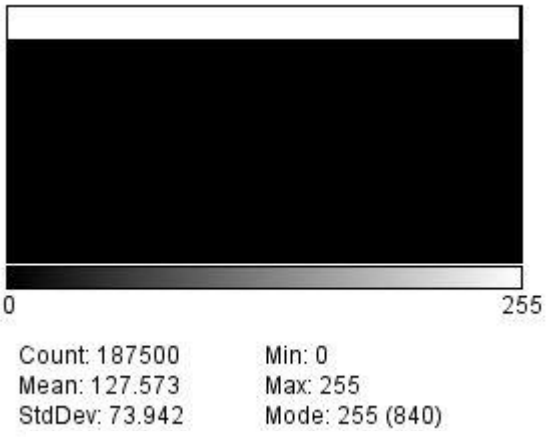
e) Histogram of Input

d) Method A



c) Method A Histogram

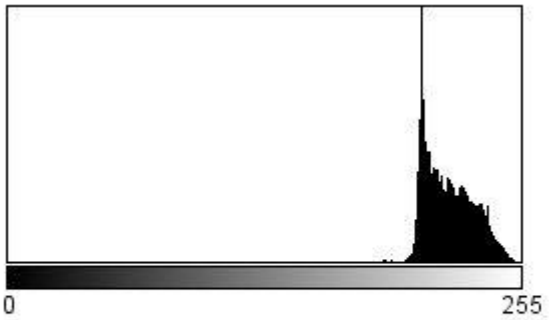
b) Method B Histogram



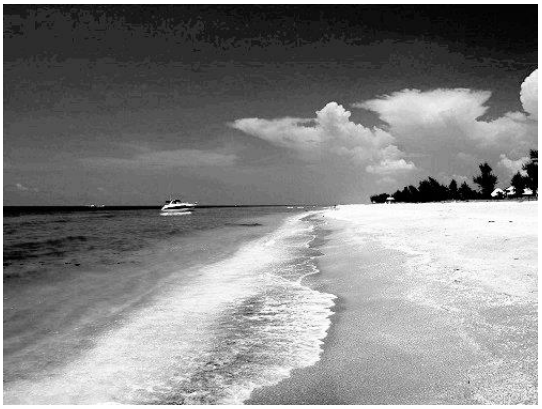
a) Method B Histogram

Figure 5: Histogram Equalization of *Beach_dark* using Method A and Method B

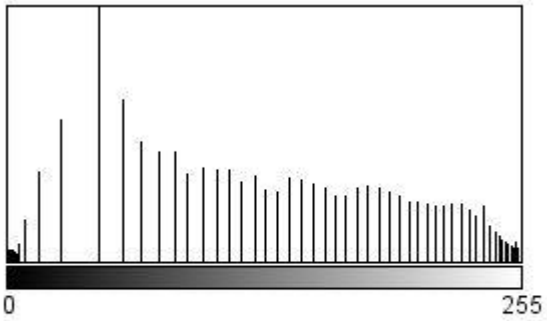
b) Original Input



a) Histogram of Input

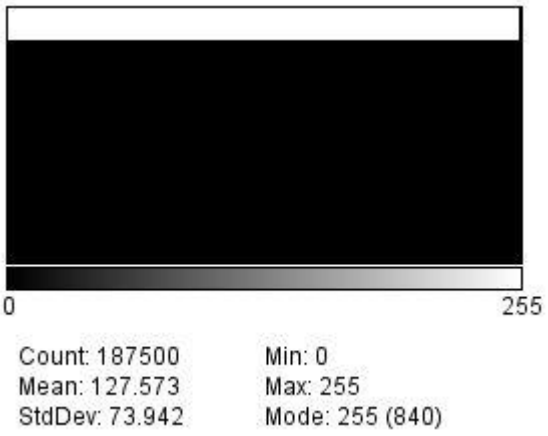


c) Method A



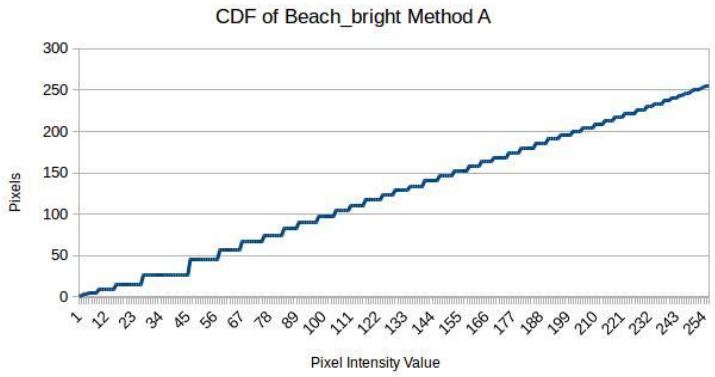
d) Histogram of Method A

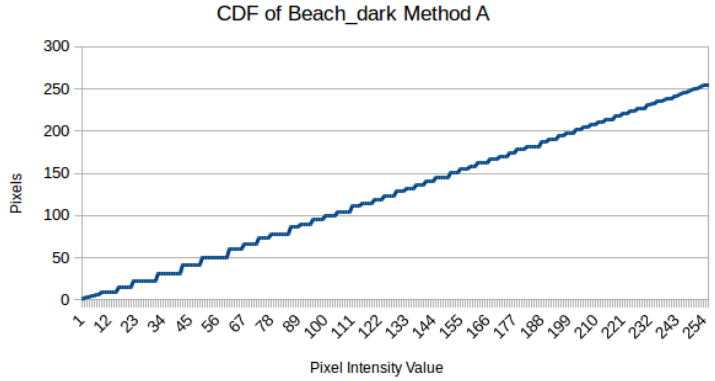
f) Histogram of Method B

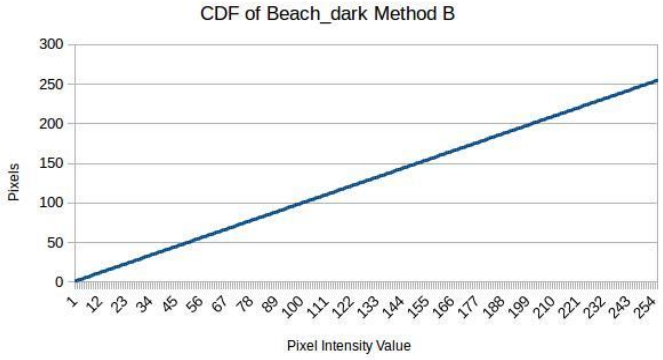


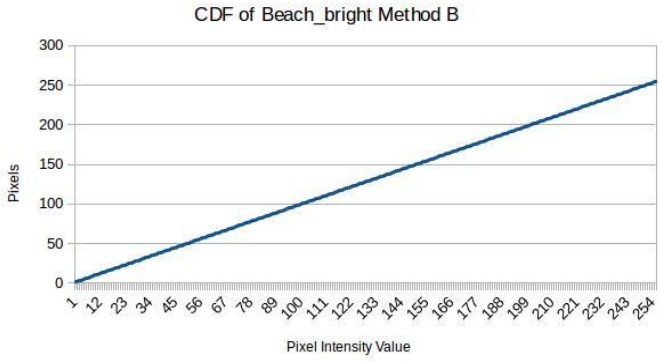
e) Histogram of Method B

Figure 6: Histogram Equalization of *Beach_bright* using Method A and Method B







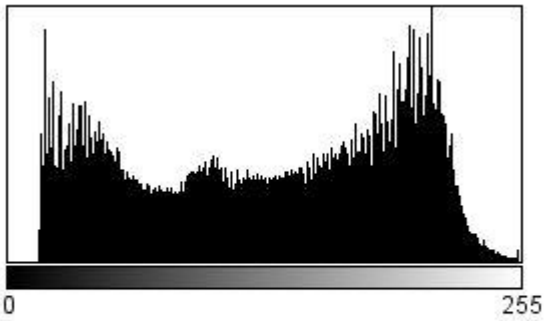


Transfer functions of *Beach_dark* and *Beach_bright* using both methods

b) Histogram Equalization for Color Images

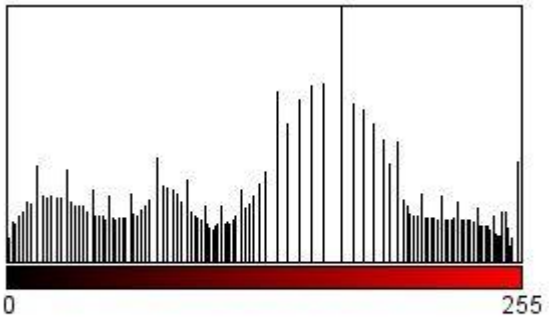
Shown below are the results of applying histogram equalization, Method A and B on Color Image.

a) Output of Method A



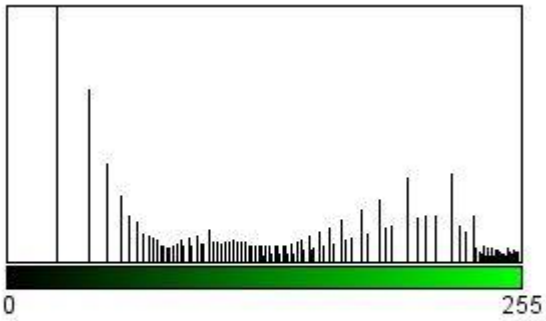
Count: 240000 Min: 15
Mean: 129.394 Max: 254
StdDev: 66.765 Mode: 211 (2551)

e) Histogram of all channels together



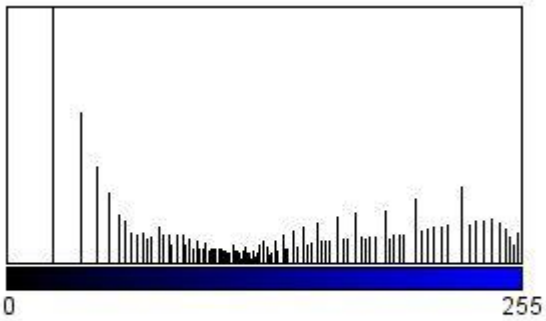
Count: 240000 Min: 0
Mean: 128.435 Max: 254
StdDev: 73.718 Mode: 166 (8454)

d) Histogram of all Red Channel



Count: 240000 Min: 24
Mean: 130.063 Max: 254
StdDev: 71.435 Mode: 24 (22834)

c) Histogram of all Green Channel

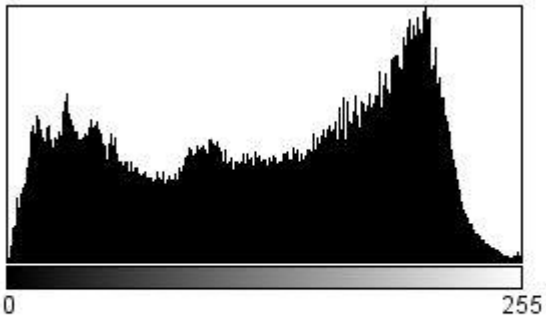


Count: 240000 Min: 22
Mean: 129.672 Max: 255
StdDev: 71.869 Mode: 22 (21351)

b) Histogram of all Blue Channel

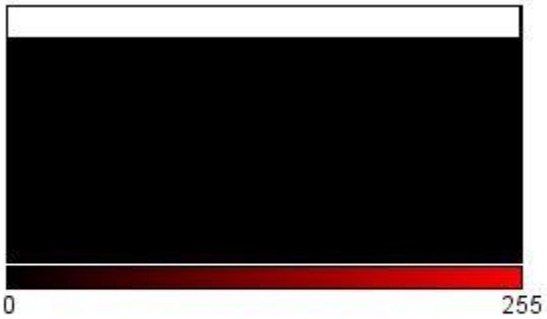
Figure 7: Histogram Equalization of *Skyline image* using Method A

a) Output of Method B



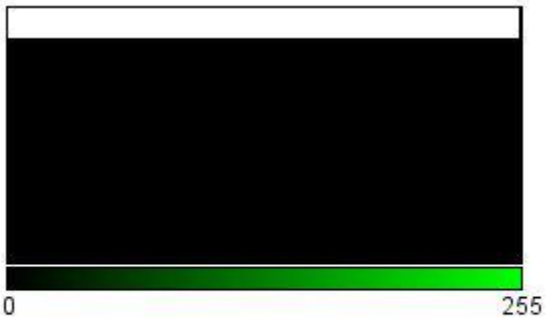
Count: 240000
Mean: 127.568
StdDev: 68.334
Min: 0
Max: 255
Mode: 208 (2091)

c) Histogram of all three channels



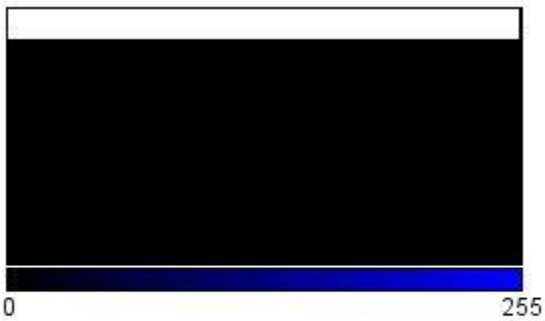
Count: 240000
Mean: 127.568
StdDev: 73.939
Min: 0
Max: 255
Mode: 255 (1065)

b) Histogram of all Red channel



Count: 240000
Mean: 127.568
StdDev: 73.939
Min: 0
Max: 255
Mode: 255 (1065)

d) Histogram of all Green channel



Count: 240000
Mean: 127.568
StdDev: 73.939
Min: 0
Max: 255
Mode: 255 (1065)

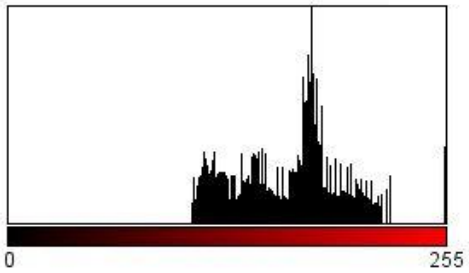
e) Histogram of all Blue channel

Figure 8: Histogram Equalization of *Skyline image* using Method B

c) Special Effects

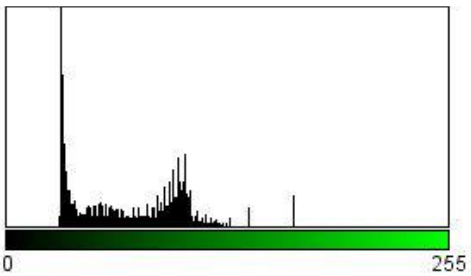
Shown below are the results obtained on performing histogram matching for images *Trojan* and *Park*

a) Special Effect applied to *Trojan Image*



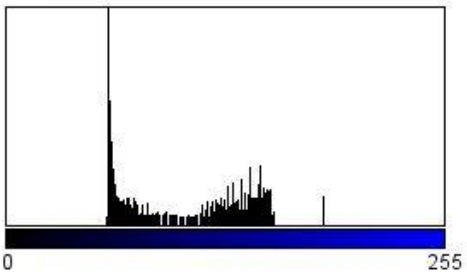
Count: 135000
Mean: 163.845
StdDev: 31.229
Min: 107
Max: 255
Mode: 177 (4774)

d) Red Channel Histogram



Count: 135000
Mean: 70.321
StdDev: 31.965
Min: 30
Max: 166
Mode: 31 (12703)

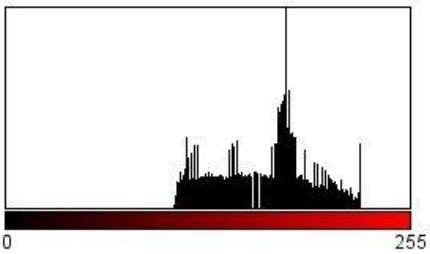
b) Green Channel Histogram



Count: 135000
Mean: 105.023
StdDev: 35.945
Min: 58
Max: 185
Mode: 59 (12074)

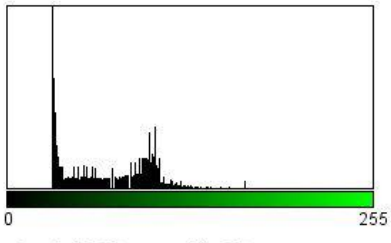
c) Blue Channel Histogram

e) Special Effect on *Park* Image



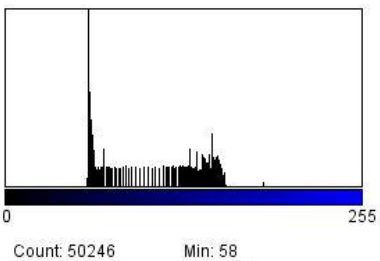
Count: 50246
Mean: 163.505
StdDev: 30.205
Min: 106
Max: 224
Mode: 177 (2136)

h) Red Channel Histogram



Count: 50246
Mean: 70.146
StdDev: 31.373
Min: 30
Max: 166
Mode: 31 (5012)

g) Green Channel Histogram



Count: 50246
Mean: 104.921
StdDev: 35.375
Min: 58
Max: 185
Mode: 59 (4509)

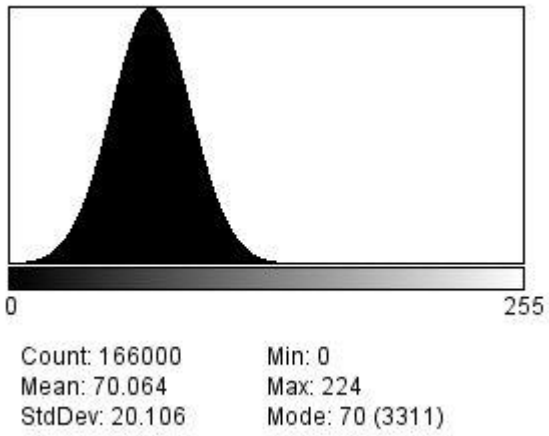
f) Blue Channel Histogram

Figure 9: Histogram Matching for Special Effect

d) Histogram Transform

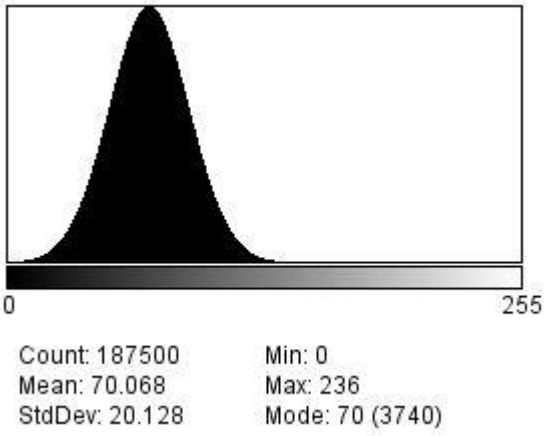
Shown below are the results for Histogram matching for images *Student_1* and *Student_2* respectively along with their histogram.

b) Histogram Matching for *Student_1*



a) Histogram of *Student_1*

d) Histogram Matching of *Student_2*



c) Histogram of *Student_2*

Figure 10: Histogram Matching to Gaussian Function

2.4. Discussion

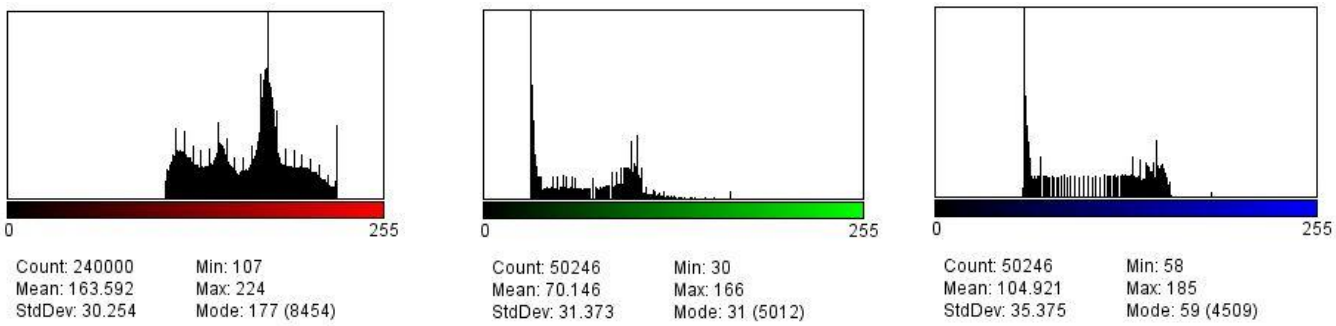
Figure 5 and Figure 6 show the results obtained on applying histogram equalization to *Beach_dark* and *Beach_bright* images. It can be seen that both methods perform equally well in enhancing the contrast of the dark and bright images. The histogram images on the right show the difference in the implementation of Method A and Method B. As expected Method B uses a bucket filling method wherein each bucket or pixel intensity level is filled with equal number of pixels. Thus the histogram of the output from Method B shows a uniform distribution of pixels.

Method A performs histogram equalization by calculating the transfer function based on cumulative probability density function. The pixels that are initially at a narrow range are distributed over the range between 0-255 while preserving the features of the original image. It can also be noticed that both *Beach_bright* and *Beach_dark* images are equalized to comparable outputs with very little perceivable change. This is expected as the bright and dark images possess the same histogram with shifted values thus their redistribution remains the same.

Bucket filling method forces the image to have its pixels to be uniformly distributed. This may sometimes lead to a distortion in the image if the pixels of the image are centered at one particular value, and these pixels are forced to redistribute over all intensity levels.

Figure 7 and Figure 8 show the results of Histogram Equalization methods applied on *Skyline* image. The *Skyline* image has concentration of R, G and B channels in such a way that Red channel pixels are shifted to the right, Green to the left and Blue centered about 100. This gives rise to an effect to simulate old days. The histogram equalized image distributes the pixels in all three channels to range 0~255 similar to the operation on beach images explained above. The obtained output has its contrast significantly enhanced and both Method A and Method B perform equally well.

Figure 9 shows the result on applying the special effect on *Trojan* and *Park* images. Objective was to perform a histogram matching of input image with that of *Skyline* image for all three channels. The histogram of Red, Green and Blue channels of *Skyline* image is shown below.



It can be observed that Red channel is centered at around 163, Green at 70 and Blue at 105. The histogram matching redistributes pixels to match the distribution of Red, Green and Blue channels of the *Skyline* image. Thus, the output images too are centered at 163 for Red, 70 for Green and 105 for Blue. Further, the two images also have the same standard deviation, mode, max and min values. The resulting effect that similar to the effect on the *Skyline* image as visible in ***Figure 9 a and e***.

Figure 10 shows the output on performing Histogram Matching on *Student_1* and *Student_2* images to match a Gaussian Function with Mean 70 and Standard deviation of 20. Since the two output images are centered at a low grayscale value, both the image have a darker shade.

The histogram of the output for *Student_1 and Student_2* display a Gaussian Distribution of the pixels with mean at approximately 70 and standard deviation of 20 pixels. Thus Histogram Matching through the Bucket-Filling Method demonstrates that any image of arbitrary pixel distribution can be matched to another distribution such as a Gaussian.

Problem 3: Noise Removal

3.1. Abstract and Motivation

The motivation being this problem is to restore an image that is corrupted by various types of noise. Noise is a very relevant problem is image capture and rendering. Digital Cameras are susceptible to noise such as Gaussian and Salt and Pepper noise. This is because Digital Cameras contain electric circuits, sensor elements which contribute to noise. Electronic circuits cause noise due to photodiode leakage currents and may appear as Salt and Pepper noise [7]. Sensors are susceptible to noise due to the presence of dust particles or because of heat. Such noises may also appear as Salt and Pepper noise. It may sometimes be noticed that in images with poor illumination, the image appears to be noisy. This might be because of long exposure contributing to sensor noise.

In all these cases, it is highly desirable to filter out the noise to improve image quality. Salt and Pepper noise can be removed by applying Mean and Median filters. In this problem, 3 different methods have been applied for noise removal and the resulting images are compared based on Peak Signal to Noise Ratio. Specifically, Mean and Median filters and their different cascaded combinations have been applied on *Lena_noisy* image and the best combination has been reported. Further the Non-Local Means Filtering has been implemented on C++ and has been applied on *Lena_noisy*, *Buildings_noisy*, *Trees_noisy* images. The non-local means is an intuitive solution to image noise reduction wherein the pixel value at every location is estimated by summing up contribution by most similar neighboring pixels by comparing the pixel neighborhoods.

In comparison, the Mean and the Median filters perform a low pass filtering of the image. The high frequency components are effectively replaced by the mean or median of their neighborhood determined by filter size. The underlying assumption is that most noise components are higher frequencies and hence a low pass filter is applied.

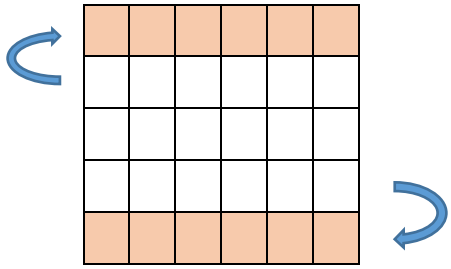
3.2. Approach and Procedures

For the first part of the problem, two types of filters have been applied on Lena_noisy image, namely Mean and Median Filter.

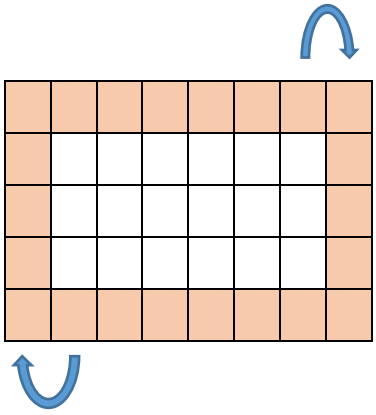
Mean Filter

A filter size of 3x3 was chosen which was centered at every pixel. Before convolving, the original noisy image was first padded by a row and column of pixels on all 4 edges of the image. Padding was implemented as follows,

The first and last row of the image was replicated and appended on top and bottom of the noisy image for every channel of the image.



Next an extra column is appended to both the first and last column of the newly generated image, giving us a padded image of size *(width+2) * (height+2)*3*



The mean filter is applied by replacing the value of every pixel with the average of 9 pixels in the 3x3 neighborhood centered about that pixel, giving us a new Image Y from original image X.

i.e for every pixel u in X

Y(u) = $\frac{1}{9}\sum N(u)$ Where N(u) corresponds to the 3x3 neighborhood of pixel u in X

Median Filter

The median filter is implemented by replacing every pixel in the image by the median of its neighboring 3x3 pixels centered on the pixel. Ex. If we wish to replace the centre pixel of the below 3x3 grid, then 92 will be replace by the median of 2,5,41,47,67,90,129,192,245 which is 67.

47	41	5
129	192	2
245	90	67

*	*	*
*	67	*
*	*	*

Each of the 9 elements were appended to a vector element defined under the Standard Template Library in C++ with the header file <vector>

Sorting was performing using the *sort()* function available under the Standard Template Library under the <*algorithm*> header. The middle element corresponding to location 4 in the vector then represents the median.

Noise Metric – PSNR

Peak Signal to Noise Ratio is the metric used to compare the performance of different combinations of Mean and Median filters applied to the input noisy image.

PSNR is defined as follows

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

$$\text{Where MSE} = \frac{1}{NM}\left(\sum_{i=0}^{N}\sum_{j=0}^{M}\big(Y(i,j) - X(i,j)\big)^2\right.$$

Where X = Original Noise-free Image of size N x M

Y = Filtered Image of Size N x M

Max = Maximum possible pixel intensity = 255

Thus PSNR has a higher value for a better filtered image and a lower value for a noisy image defined by the Mean Squared Error.

Different combinations of Mean and Median filters were cascaded and the PSNR value for all three channels were computed and tabulated, giving us an idea about the best performing cascaded combination.

Non-Local Means(NLM) Filter

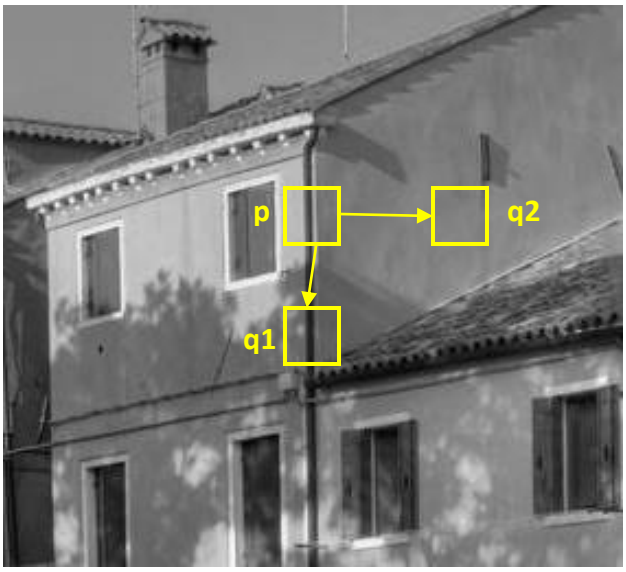
The Non-Local Means filter is implemented following the pixel-wise implementation mentioned in [1]

A **search window of 21x21 pixels and a neighborhood size of 7x7** has been used as recommended in [1]. In order to facilitate this the Original noisy image was **padded by 10 pixels** on all four edges by following the method similar to the above mentioned padding scheme. The first 10 rows are copied and mirrored about the top edge, and the same is repeated in the bottom.

The principle behind Non-Local Means is that for any local neighborhood in an image, there exists similar neighborhood within a search window which can be used to estimate the value of the pixel about which the neighborhood was centered. Neighborhoods that are more similar are given a larger weight than neighborhoods that are more dissimilar. Thus this principle can be used to denoise the image by replacing the pixel by the weighted average of most similar pixels.

The method is detailed as follows. Consider *figure 11* shown below containing three neighborhoods. Our objective it to estimate the value of the pixel **p** and we compare the neighborhoods pixels **q1** and **q2** to check for similarity

It can be noted that the neighborhood of **q1** will be more similar that the neighborhood of **q2** and thus it should contribute to a higher weight.



Thus for any given channel we can estimate the value of the pixel as follows,

$$BL(V(p)) = \frac{1}{C(p)} \sum_{q \epsilon N(p,r)} V(q) w(p,q)$$

Where $C(p) = \sum_{q \epsilon N(p,r)} w(p,q)$

Where N(p,r) represents the search window surrounded by pixel p with size 2r+1 x 2r +1. In this implementation the search window was chosen to be **21 x 21** with r having value 10. Thus, q represents every other pixel present inside the search window defined by r. The weight is defined by w(p,q) which will be a function of the similarity between the neighborhoods of pixel p and pixel q.

w(p,q) is written as the function of the Euclidean distance d^2 between the neighborhoods of p and q.

$$d^2 = d^2(N(p,f),N(q,f)) = \frac{1}{(2f+1)^2} \sum_{k \epsilon N(p,f)} (V(p+k) - V(q+k))^2$$

N(p,f) defines the neighborhood region defined by f and is equal to 2f+1 x 2f+1.

In this implementation, the neighborhood size is taken as **7 x7**, with f =2.

The Euclidean distance gives a measure of the similarity between neighborhoods of p and q, the higher value making it less similar and wise versa.

Next having found the value of d^2, we use an exponential weighing function such that the weight for a lower d^2 is large with a maximum of 1 and the weight for higher d^2 drops exponentially. We define a parameter σ to denote the standard deviation of the noise and h as a filter parameter. Thus weight w(p,q) is defined as follows.

$$w(p,q) = e^{\frac{-(\max(d^2 - 2\sigma^2, 0.0))}{h^2}}$$

If the dissimilarity between the patches is larger than $2\sigma^2$, a lower weight is given to the averages of the patch. If the similarity between the patch is less that $2\sigma^2$, a weight value of 1 is multiplied.

Thus the pixel value gets the average of most similar pixels. This process is repeated for all three channels. The parameters σ and h control the effect of denoising in the image.

σ defines the standard deviation in similarity allowed, whereas h controls the weight decay. Different σ and h were chosen for *Lena_noisy, Buildings_noisy and Trees_noisy* and the performance of NLM filter was compared by computing the PSNR. Taking into account the search window size and neighborhood window size, **h =0.35σ** has been chosen [1].

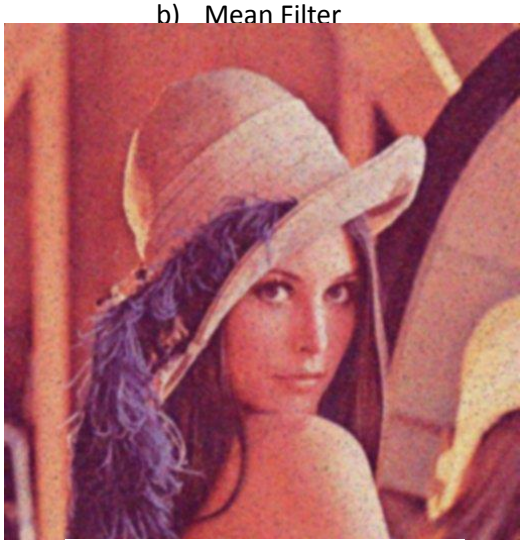
To decrease computation time, this weight function is adopted instead of a Gaussian weighing function.

3.3. Experimental Results

Shown below are the results on applying Mean, Median and cascaded combination of Mean and Median Filters.

b) Mean Filter

a) Median Filter



d) Mean + Mean Filter

c) Mean + Median Filter

f) Mean + Median +Mean Filter

e) Mean + Median +Median Filter

h) Median + Mean + Mean Filter

g) Median + Mean +Median Filter

j) Median + Mean +Median + Mean Filter

i) Median + Mean +Median + Median Filter

l) Median + Mean +Median + Median +Median Filter

k) Median + Median Filter

m) Median + Median +Median

Figure 11: Cascaded Combination of Mean and Median Filters applied to *Lena_noisy.raw*

Table 1: PSNR Performance of different cascaded filters

	PSNR – Red (dB)	PSNR – Green (dB)	PSNR-Blue (dB)
No Filter	17.8596	18.7194	19.2236
Mean	24.7143	25.2352	25.8602
Median	25.7123	25.5004	25.4008
Mean + Median	25.6991	25.8982	26.6708
Median + Mean	27.1212	26.5421	26.7087
Median + Mean + Median	27.3388	26.5973	26.8761
Mean + Median + Mean	25.7648	25.6822	26.5934
Median + Median	26.894	26.458	26.4869
Mean + Mean	25.4607	25.6437	26.519
Median + Mean + Mean	27.0342	26.249	26.6534

Mean + Median + Median	26.0112	26.0023	26.8011
Median+Median+Median	27.2165	26.6266	26.7513
Mean + Mean + Mean	25.3742	25.3475	26.3424
Median + Mean + Median + Mean	27.0724	26.1905	26.6863
Median + Mean + Median + Median	27.393	26.5709	26.9233
Median + Mean+Median+Median+Median	27.4068	26.5319	26.9291

Shown below are the results of applying Non-Local Means filter to Lena_noisy, Trees_noisy and Buildings_noisy

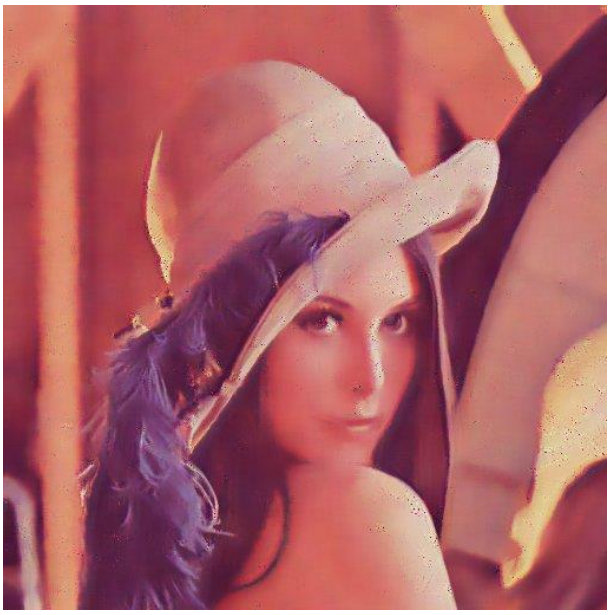
b) σ = 5, h = 1.75

a) σ = 10, h = 3.5

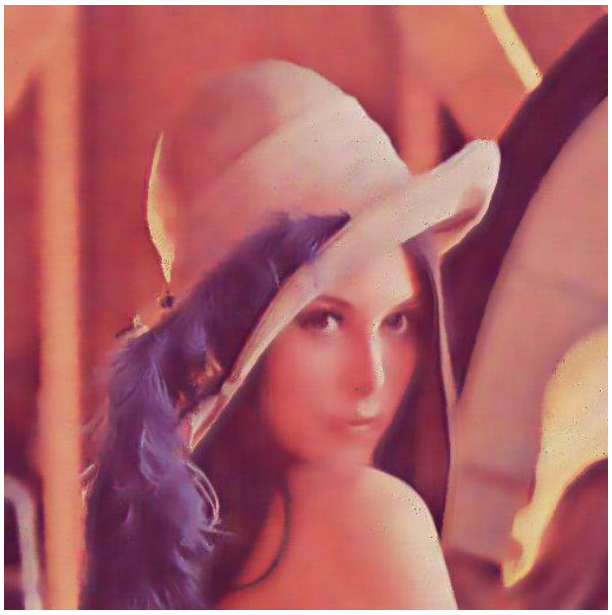
c) σ = 20, h = 7

Figure 12: Non-Local Means Filter on *Buildings_noisy.raw*

c) σ = 30, h = 10.5

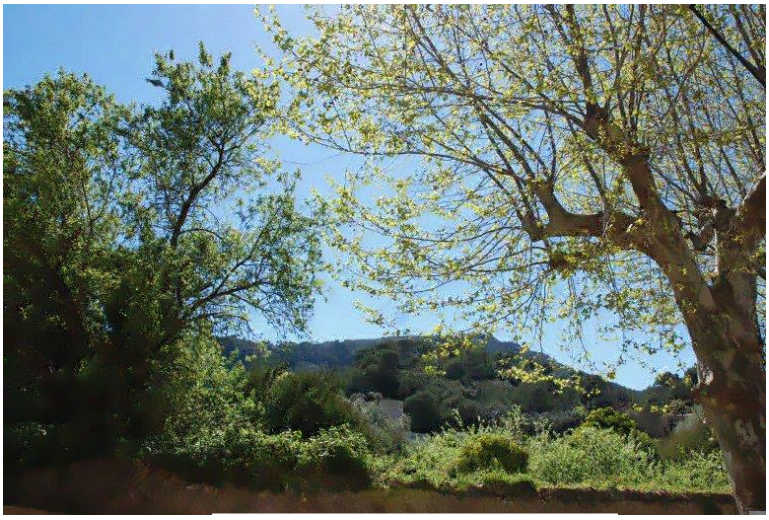


b) σ = 35, h = 12.25



a) σ = 40, h = 10

Figure 13: Non-Local Means Filter on *Lena_noisy.raw*



a) σ = 20, h = 7

c) σ = 25, h = 8.75

b) σ = 30, h = 10.5

Figure 14: Non-Local Means Filter on *Trees_noisy.raw*

Table 2: PSNR values for NLM filter

	σ	h	PSNR- Red (dB)	PSNR-Green (dB)	PSNR-Blue (dB)
Lena	30	10.5	21.079	20.217	23.0287
Lena	35	12.25	21.6331	20.6784	23.3916
Lena	40	10	21.9946	21.0254	23.533
Buildings	5	1.75	15.96	16.106	15.388
Buildings	10	3.5	15.6736	16.2357	15.4897
Buildings	20	7	16.1996	16.8221	15.9368
Trees	20	7	15.6674	14.7126	15.8112
Trees	25	8.75	16.2807	15.0849	13.7
Trees	30	10.5	17.388	15.84	14.9

3.4. Discussion

a) Mix-noise in color image

Figure 11 a) through m) show the results obtained on cascading different combinations of Mean and Median filters on *Lena_noisy* image. It can be noted that in terms of performance, the Median filter outperforms the Mean filter in terms of PSNR. Since these filters are essentially low-pass filters, there happens to be a blurring of the image as the high frequency components are removed.

Table 1 gives the PSNR value for various cascaded combinations of Mean and Median filter.

1. The types of noise present in Lena_noisy is Uniform and Impulse noise which appear as Mixed noise. ImageJ was used to generate histogram.

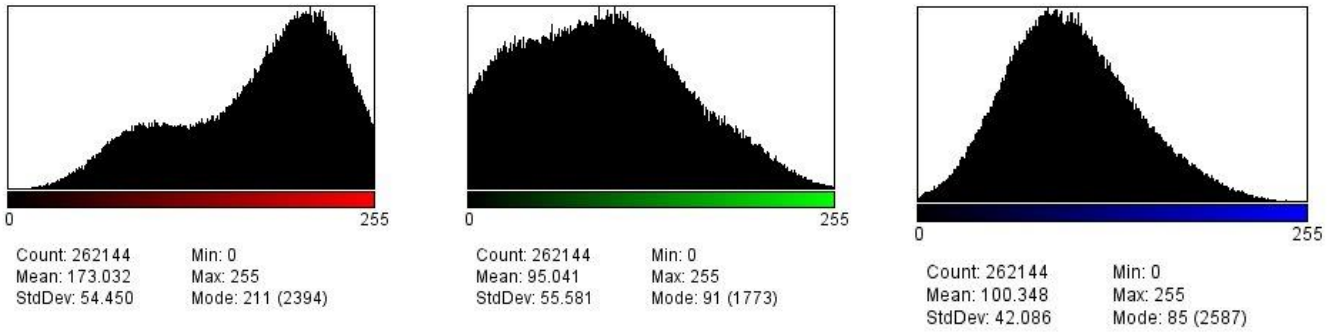


Figure 15: Histograms or R, G, B channels of *Lena_noisy*

a. The above histogram shows that all three channels have the same Uniform and impulse noise.

b. Filtering must be performed on all three channels as all three channels have the same noise.

c. Linear and non-linear filters can be used to remove mixed noise. Linear filters such as mean filter help in removing uniform noise whereas Non-linear filters such as Median filter perform well on removing Impulse noise like Salt and Pepper. This is because the mean of Uniformly distributed noise is zero and thus mean filter succeeds in eliminating this noise. Impulse noise does not have mean zero and thus a mean filter cannot be used. Median filters help removing impulse noise because they are scattered randomly and thus median pick the optimum value in a neighborhood, eliminating randomly scattered noise.

d. The order of cascading highly matters and thus filters cannot be cascaded in any order. As shown in Table 1, Mean + Median filter gives an overall lower PSNR value than Median + Mean filter combination. This is because the mean filter when applied first, does a good job in removing uniform noise but does not help in removing impulse noise like salt and pepper. Further inherent impulse noise is added on every pixel on averaging. When Median filter is applied first, impulse noise is reduced whereas and no extra noise is added in the process. Applying Mean filter later helps remove zero mean uniform noise.

e. When window size increases, a greater number of pixels are used for averaging or calculating the Median. An effect of increasing the window size is that features that are not part of the local neighborhood of the pixel influence the value of the pixel just affecting the quality and PSNR value. The PSNR value for Mean and Median Filter using window size 3 and 7 are shown below. The PSNR value increases on increasing window size for median. This might be because a 7x7 patch is more appropriate for Lena. An optimum window size can only be chosen while considering the local neighborhoods of the image.

	3 x 3 filter PSNR (dB)	7 x 7 filter PSNR (dB)
Mean Filter (Lena_noisy)	25.26	25.1
Median Filter (Lena_noisy)	25.53	26.23

The best performing combination is *Median + Mean + Median + Median + Median* filters, which gives a PSNR value of ***27.40, 26.53 and 26.92*** Decibels respectively for Red, Green and Blue channels. It is also observed that the combination *Median + Mean + Median* filters performs equally well giving a PSNR value of ***27.33, 26.59 and 26.87*** Decibels for Red, Green and Blue channels.

Shortcomings

Although these methods perform really well in eliminating noise and PSNR level, they cause significant blurring of the image by removing high frequency components. Thus the edge information is lost and the perceptual quality of the image is lost.

Improvements

These methods also do not take advantage the spacial similarity that exists in an image such as repeated patterns and features that might occur at regions far away from the local neighborhood of the pixel. Other filters such as Non-local means and BM3D take into account these similarities and perform better as compared to these methods.

b) Non-Local Means Filter

Figures 12, 13 and 14 show the results on applying the Non-Local Means Filter on *Lena_noisy, Buildings_noisy and Trees_noisy* images for various parameters. As mentioned in section 3.2, the relation between σ and h is chosen to be $\mathbf{h = 0.35\sigma}$ in accordance with [1]. σ determines the tolerance level chosen for the similarity between neighborhoods, lower the σ, the more tolerant to noise and vice versa. The parameter h determines the weight decay. A high value of h will cause the image to become blurry and a low value will allow more noise. Thus these parameters are fine tuned to provide optimum results.

In figure 12, it can be seen that a small value of σ and h do a good job at denoising the image and the best value is when **σ = 10 and h = 3.5**. On increasing σ to 20, the image becomes washed out and loses information such as the shadows (edges) of the buildings. This is also true in *Figure 14*, where a high value of σ causes the lower half of the image to look washed out and blurry. To preserve high frequencies, the optimum value chosen is **σ = 20** for *Trees* image.

In *figure 13*, a very **high value** of **σ** is required to sufficiently denoise the image. This is because of the nature of the image having a high standard deviation of Gaussian noise. As we increase **σ from 30 to 40**, noise is gradually removed but the images become **more blurred** and loses high frequency components.

It can be observed from *table 2* that although the results from NLM on Buildings and Trees images appear crisp and better, the PSNR score is very low highlighting the drawback of using PSNR as a metric.

Advantages of Non-Local Means Filter

- The Non-Local Means filter takes into account the redundancy in an image where features and patterns are repeated. The NLM filter scans the entire image for similar patches of color and averages the values of the most similar patches to the reference patch, thereby recovering a contaminated pixel value.
- Since most of the images in real world come with some redundant information, this method is highly effective in removing mixed noise with causing too much blurring of the image compared to simpler filters such as Mean and Median filters.
- The filtered image is thus of higher clarity and better visual performance.

c) Block matching and 3-D (BM3D) transform filter

Motivation and Abstract

The BM3D Filter proposed in [5] is similar to Non-local means where redundancy in an image is advantaged. In comparison with the results from Non-Local Means, it has shown to produce superior results which has been discussed in further sections.

Approach and Procedure

The BM3D algorithms is a two step process. The first step uses patches similar to the implementation in Non-Local means but with some modification in the procedure. For every color patch, a group of similar color patches are searched and scanned. These similar patches are then stacked up in decreasing order of similarity and are referred to as a group.

This group of images are then subjected to a 3D transform by exploiting the intra patch correlation to denoise the image. At the same time, the inter patch correlation is also leveraged. A Hard- Threshold is applied to the 3D stacked transform followed by an inverse 3D transform to obtain block wise estimates.
Now that block wise estimates is available for every block in the image, They can be positioned back into an image to obtain an intermediary estimate image. Since the blocks can overlap, overlapping blocks are placed back using weighted averaging.

This forms an aggregate which is sent to Step 2 of the filtering process.

Step 2

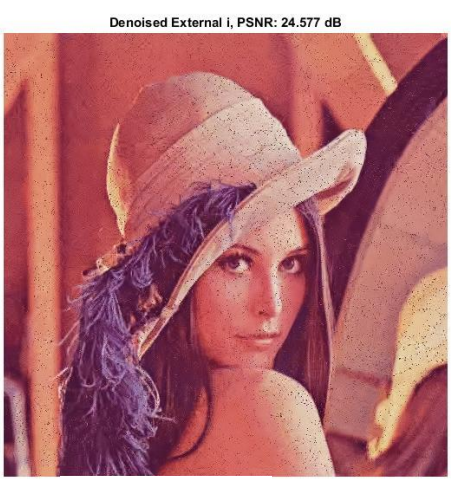
Step 2 of the BM3D algorithm is similar to step 1 but the Hard -Threshold is replaced by a Wiener filter. The aggregated basic estimate of the image is split into blocks of similar patches and compared with blocks of patches from the original noisy image

3D transform is applied to both blocks. The energy spectrum of the basic estimate is used to apply Wiener filtering on the block from the noise image.

Inverse 3D transform is then applied on these new estimates which are positioned back to form an image by weighted averaging.

The implementation of BM3D algorithm was carried by using the open source MATLAB software code provided by [5] in [4]. It can be downloaded from [4]. The BM3D algorithm was applied on *Lena_noisy, Buildings_noisy and Trees_noisy* and the results have been tabulated.

Experimental Results

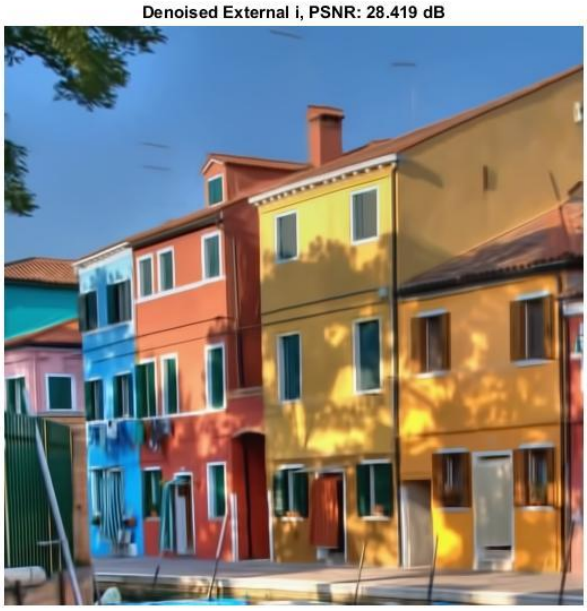


a) σ = 25

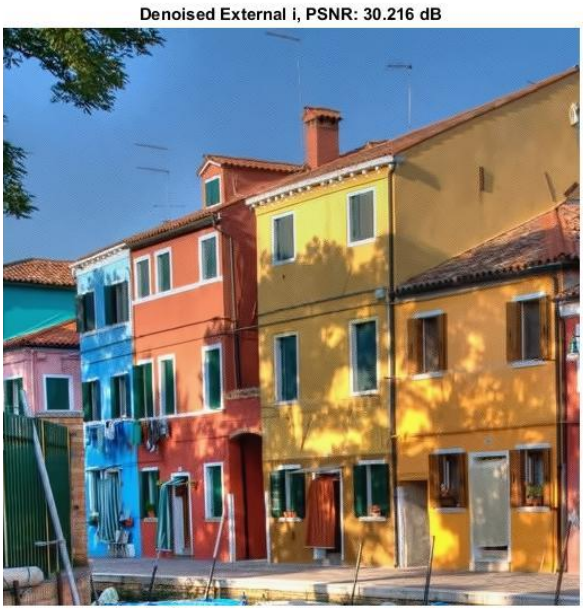
b) σ = 45

c) σ = 50

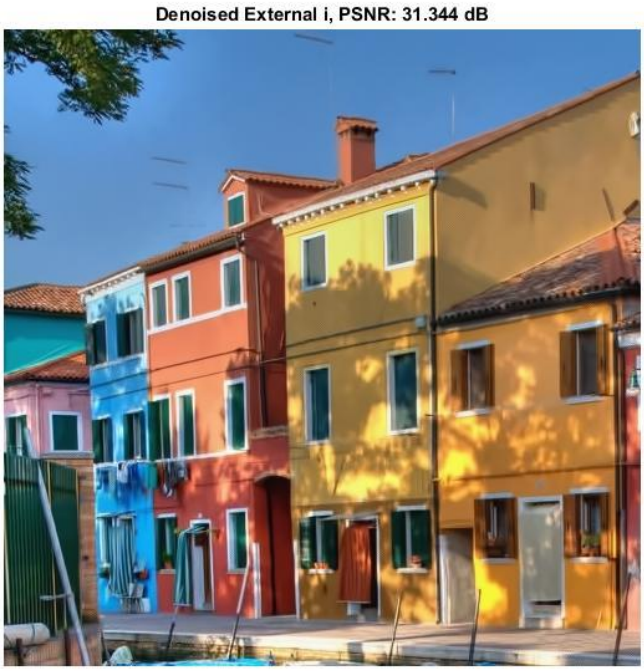
Figure 16: BM3D on Lena_noisy.raw



b) σ = 50



a) σ = 20



c) σ = 25

Figure 17: BM3D on Buildings_noisy.raw

a) σ = 25

b) σ = 50

Figure 18: BM3D on Trees_noisy.raw

Discussion

Different values of σ give varying PSNR values but overall the performance of BM3D is better and Non-Local Means in terms of PSNR and perceptual quality.

Higher σ cause the image to be more blurred and an optimum σ needs to tuned to obtain high PSNR.

The superior performance of BM3D can be attributed to using the Spectral domain as well as the Spacial Domain to perform filtering.

The best PSNR obtained for Buildings is **31.344** dB at **σ = 25**, for Lena is at **27.874** dB at **σ =50**, and for Trees is **25.885** for **σ = 25**.

References

[1] "Non-Local Means Denoising", http://www.ipol.im/pub/art/2011/bcm_nlm/

[2] Buades, B. Coll, J.M. Morel "A non local algorithm for image denoising" IEEE Computer Vision and Pattern Recognition 2005, Vol 2, pp: 60-65, 2005. DOI: 10.1109/CVPR.2005.38

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

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

[5] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian "Image denoising by sparse 3D transform-domain collaborative filtering" IEEE Transactions on Image Processing, Vol.. 16, No.8, August 2007.

[6] "HSL and HSV" Wikipedia, https://en.wikipedia.org/wiki/HSL_and_HSV

[7] "Image noise" Wikipedia, https://en.wikipedia.org/wiki/Image_noise