Lab 1: Convert RGB image to binary image and gray scaled image.

- **Theory:**

RGB (Red, Green, Blue) Image:

An RGB image is a type of digital image that represents colors using a combination of three color channels: red, green, and blue. Each pixel in an RGB image is composed of three values, one for each color channel, specifying the intensity of that color. By varying the intensities of these three colors, a wide range of colors can be represented, allowing for the creation of colorful and realistic images. RGB images are commonly used in digital photography, computer graphics, and various multimedia applications.

Binary Image:

A binary image, on the other hand, is a type of digital image where each pixel can only have one of two possible values: 0 or 1. These values typically represent black and white, respectively. Binary images are often used in computer vision, image processing, and pattern recognition. They are simpler than RGB images and are particularly useful for tasks where only the presence or absence of an object or a feature in an image is relevant.

Grayscale (Gray) Image:

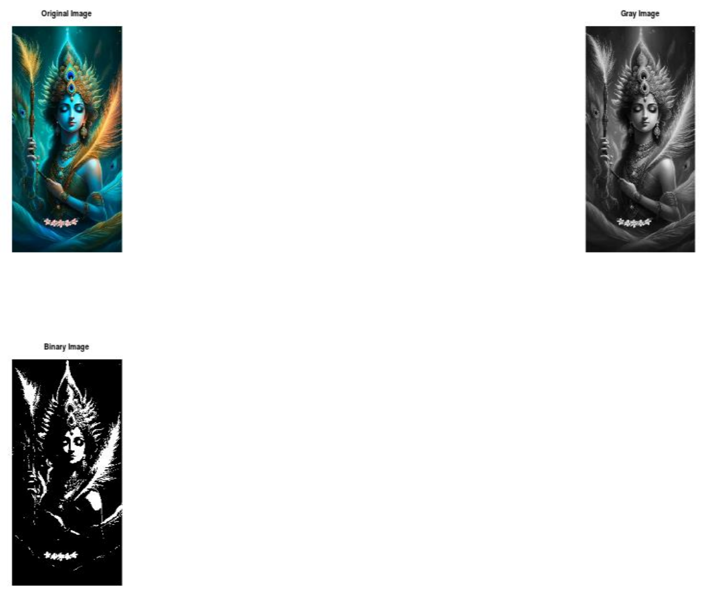
A grayscale image is an image in which each pixel can take on any value within a specified range that represents the intensity of light. Typically, the intensity values range from 0 (black) to 255 (white) in an 8-bit grayscale image. Grayscale images are used to represent variations in intensity, allowing for a smooth transition between black and white. Each pixel can have a different intensity level, providing a spectrum of shades of gray.

- **Program Code:**

```
img = imread('img/camrgb.jpeg');
gray = rgb2gray(img);
binary = gray;
[row,col]=size(binary);
for i = 1:row
        for j = 1:col
                intensity = binary(i,j);
                if intensity<=127
                        binary(i,j)=0;
                elseif intensity>127 && intensity<256
                        binary(i,j)=255;
                endif
        endfor
endfor
```

```
figure;
subplot(2,2,1); imshow(img); title('Original Image');
subplot(2,2,2); imshow(gray); title('Gray Image');
subplot(2,2,3); imshow(binary); title('Binary Image');
```

- **<u>Output:</u>**



Lab 2: Convert RGB image to negative image transformation.

- **Theory:**

Negative Transformation

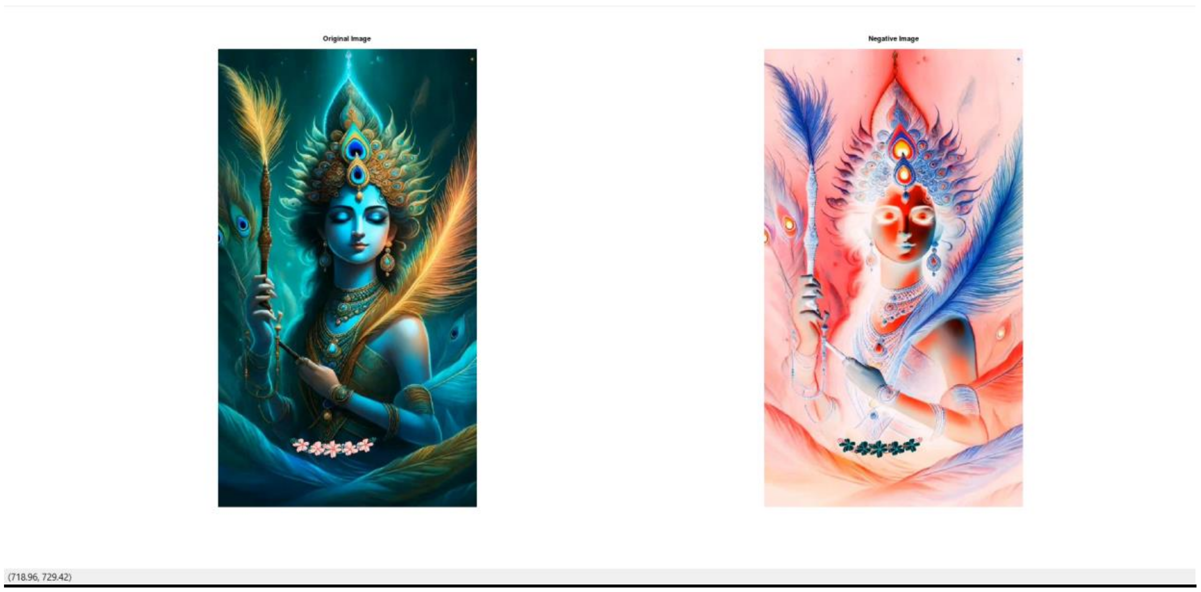
The negative of an image is achieved by replacing the intensity 'i' in the original image by 'i-1', i.e. the darkest pixels will become the brightest and the brightest pixels will become the darkest. Image negative is produced by subtracting each pixel from the maximum intensity value. For example, in an 8-bit grayscale image, the max intensity value is 255, thus each pixel is subtracted from 255 to produce the output image.

The transformation function used in image negative is: $s = T(r) = (L - 1) - r$
Where, $L - 1$ is the max intensity value, s is the output pixel value and r is the input pixel value

- **Program Code:**

```
ori_img = imread('sample1.jpg');

L = 256;

neg_img = (L-1)-ori_img;

figure;

subplot(1,2,1); imshow(ori_img);title('Original Image');

subplot(1,2,2); imshow(neg_img);title('Negative Image');
```

- **Output:**



Lab 3: Apply Log image transformation for the RGB image.

- **Theory:**

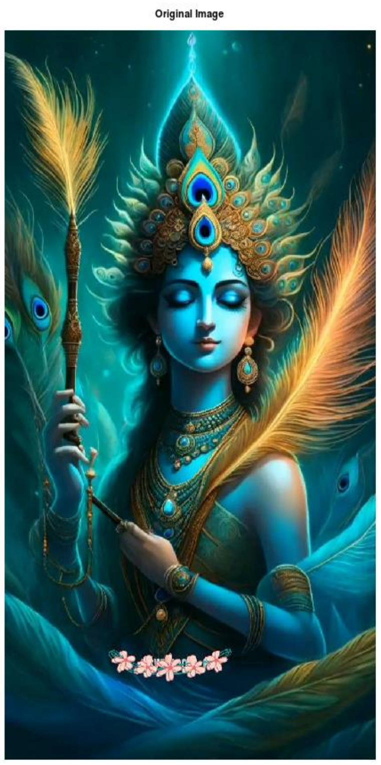
Log Transformation
The logarithmic transformation of an image is one of the gray level image transformations. Log transformation of an image means replacing all pixel values, present in the image, with its logarithmic values. Log transformation is used for image enhancement as it expands dark pixels of the image as compared to higher pixel values.

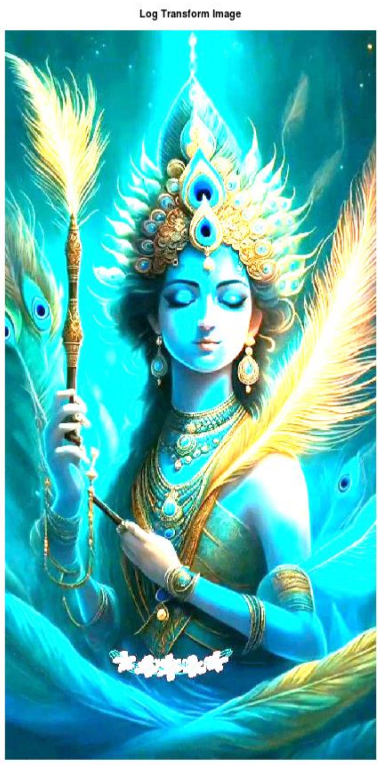
The formula for applying log transformation in an image is: $S = c * \log(1 + r)$
Where, R = input pixel value, C = scaling constant and S = output pixel value.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

double_img = im2double(ori_img);

new_img = double_img;

[row, col] = size(ori_img);

c = 3;

for i=1:row

        for j=1:col

                new_img(i,j) = c*log(1+double_img(i,j));

        endfor

endfor

figure;

subplot(1,2,1); imshow(ori_img); title('Original Image');

subplot(1,2,2); imshow(new_img); title('Log Transform Image');
```

- **Output:**

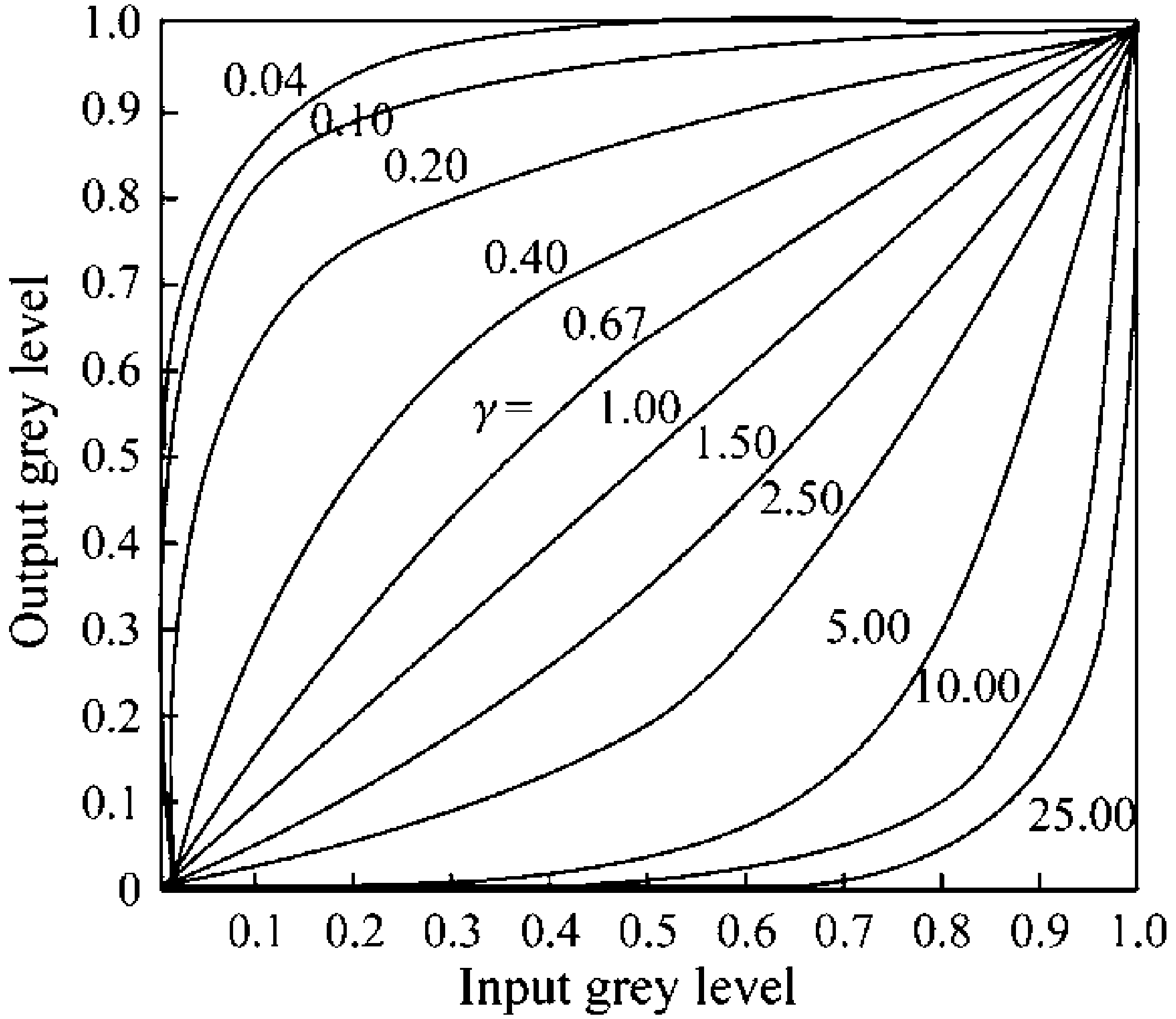




Lab 4: Apply Gamma image transformation for the RGB image.

- **Theory:**

The general form of Power law (Gamma) transformation function is: $s = c*r^{\gamma}$
Where, 's' and 'r' are the output and input pixel values, respectively and 'c' and γ are the positive constants. Like log transformation, power law curves with γ <1 map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher input values.

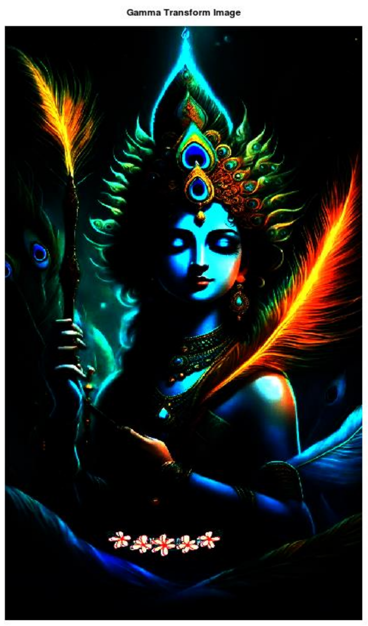


- **Program Code:**

```
ori_img = imread('sample1.jpg');
double_img = im2double(ori_img);
new_img = double_img;
[row, col] = size(ori_img);
gamma = 5;
for i=1:row
    for j=1:col
        new_img(i,j) = c*double_img(i,j)^gamma;
    endfor
endfor
figure;
subplot(1,2,1); imshow(ori_img); title('Original Image');
subplot(1,2,2); imshow(new_img); title('Gamma Transform Image');
```

- **Output:**

Original Image



Lab 5: Perform Histogram Equalization of RGB image.

- **Theory:**

The histogram of a digital image with gray levels in the range *[0, L-1]* is a discrete function. Histogram of an image provides a global description of the appearance of an image. Information obtained from histogram is very large in quality. Histogram of an image represents the relative frequency of occurrence of various gray levels in an image.

$$H(r_k) = n_k$$

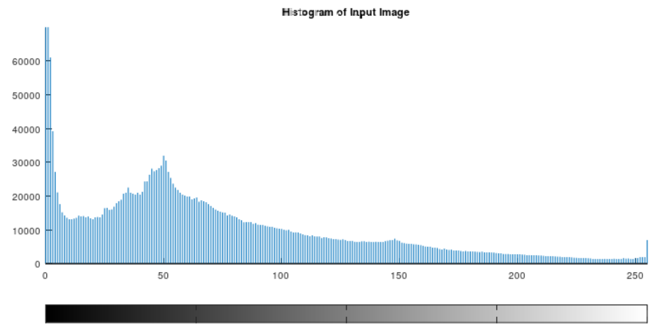
Histogram Equalization**:**
The histogram of a digital image, with intensity levels between 0 and (L-1), is a function $h(r_k) = n_k$, where r_k is the kth intensity level and n_k is the number of pixels in the image having that intensity level. We can also normalize the histogram by dividing it by the total number of pixels in the image.

- **Program Code:**

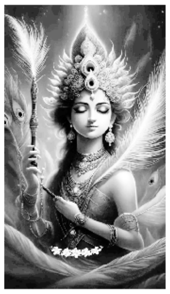
```
img = imread("sample1.jpg");

subplot(2, 2, 1);

imshow(img);

title("Original Picture");

subplot(2, 2, 2);

imhist(img);

title("Histogram of Input Image");

img1 = rgb2gray(img);

hist_eq = histeq(img1);

subplot(2, 2, 3);

imshow(hist_eq);

title("Equalized Image");

subplot(2, 2, 4);

imhist(hist_eq);

title("Histogram of Equalized Image");
```

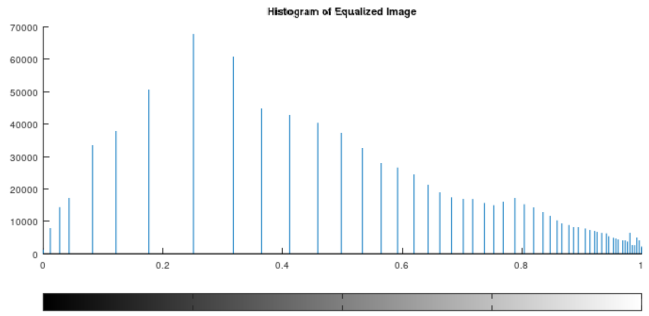
- **Output:**

Original Picture



Equalized Image



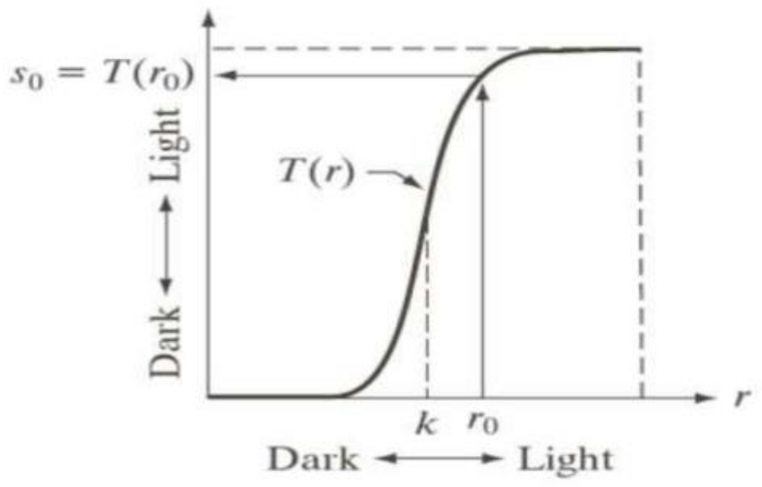


Lab 6: Perform Contrast Stretching of RGB image.

- **Theory:**

The goal of the contrast-stretching transformation is to enhance the contrast between different parts of an image, that is, enhances the gray contrast for areas of interest, and suppresses the gray contrast for areas that are not of interest.

Graph below shows two possible functions:



If T(r) has the form as shown in the figure above, the effect of applying the transformation to every pixel to generate the corresponding pixels produce higher contrast than the original image, by:

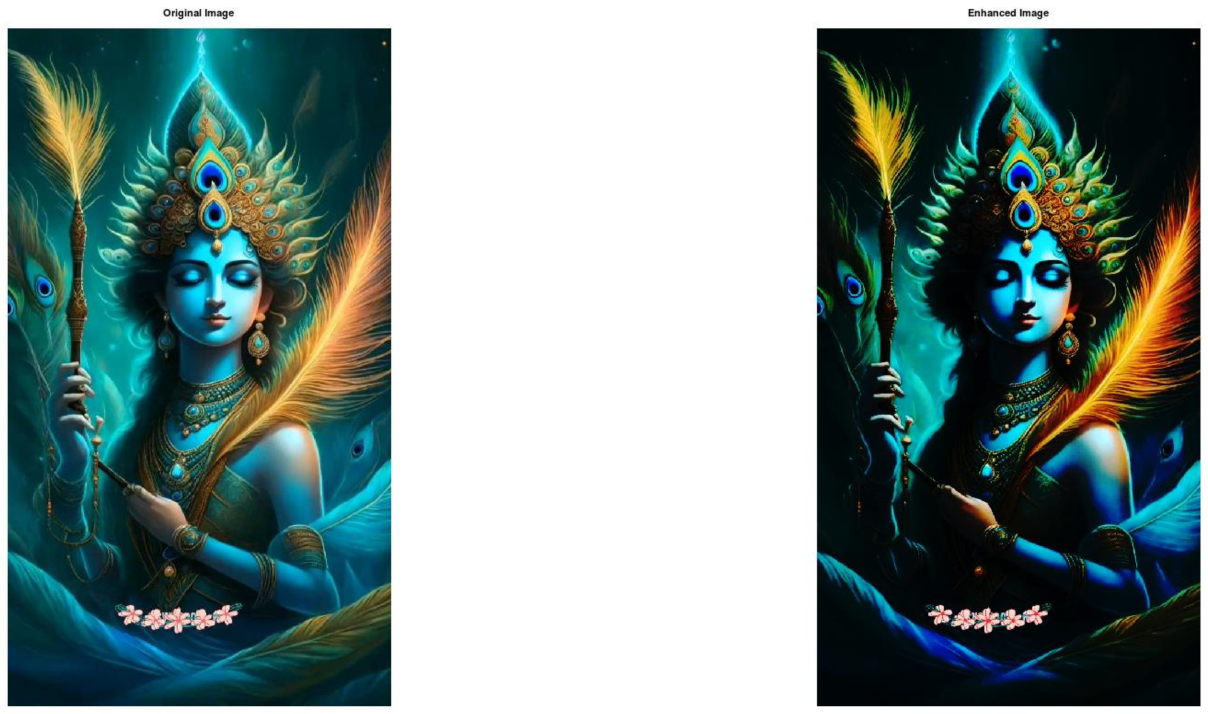
- Darkening the levels below k in the original image
- Brightening the levels above k in the original image

- **<u>Program Code:</u>**

```
ori_img = imread('sample1.jpg');
new_img = ori_img;
s1 = 20;
s2 = 150;
r1 = 100;
r2 = 150;
L = 256;
alfa = s1/r1;
beta = (s2-s1)/(r2-r1);
gamma = ((L-1)-s2)/((L-1)-r2);
[x,y,z] = size(new_img);
for i=1:x
for j=1:y
for k=1:z
if new_img(i,j,k)<=r1
r = new_img(i,j,k);
new_img(i,j,k) = alfa*r;
elseif new_img(i,j,k)>r1 && new_img(i,j,k)<=r2
r = new_img(i,j,k);
new_img(i,j,k) = (beta*(r-r1))+s1;
else
r = new_img(i,j,k);
new_img(i,j,k) = (gamma*(r-r2)+s2);
endif
endfor
endfor
endfor
```

```
figure;
subplot(1,2,1); imshow(ori_img); title('Original Image');
subplot(1,2,2); imshow(new_img); title('Enhanced Image');
```

- **Output:**



Lab 7: Perform Gray Level Slicing of RGB image.

- **Theory:**

This technique is used to highlight a specific range of gray levels in a given image, the process, often called intensity-level slicing, and can be implemented in several ways.

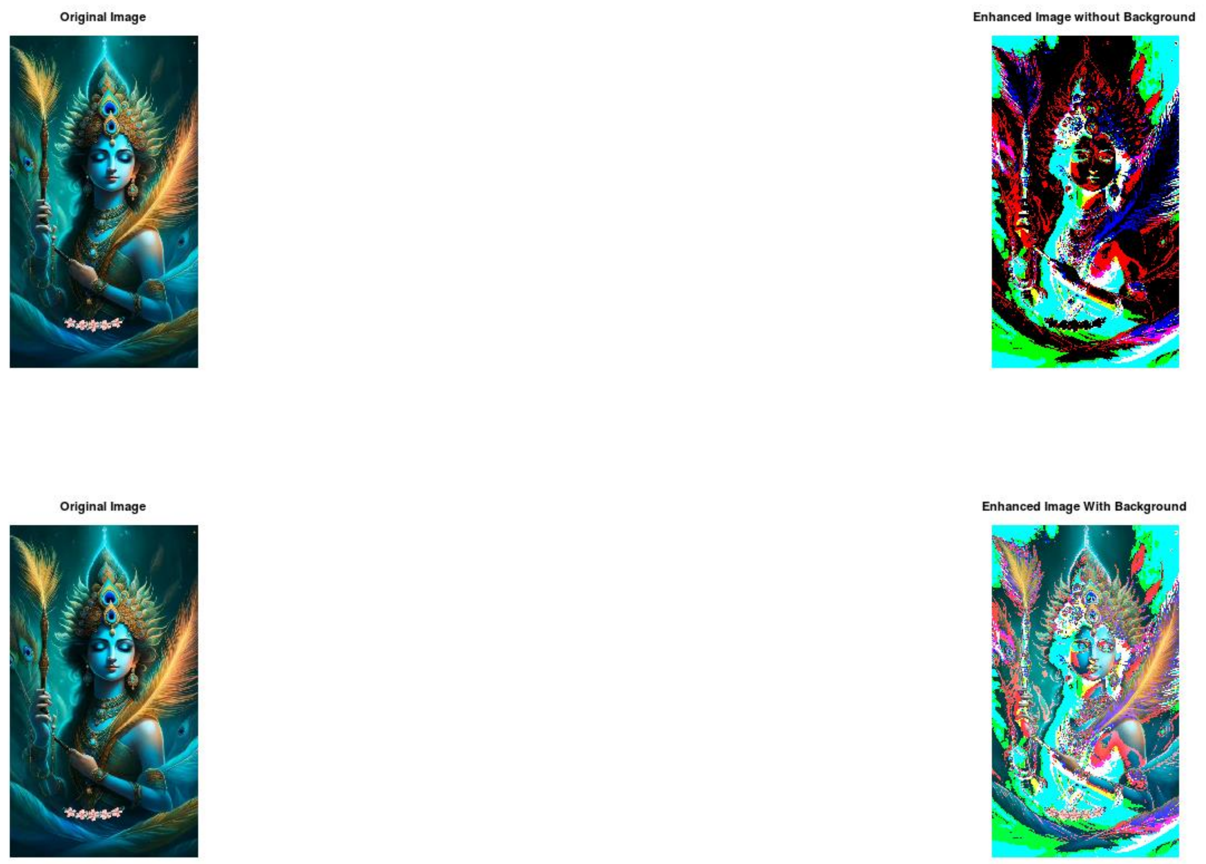
- Display all high values (say, white) in the range of interest, and low value (say, black) for other intensities. This transformation produces a binary image.
- The second approach Brightens or darkens the desired range of intensities but leaves all other intensities in the image unchanged.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

img = ori_img;

L = 256;

r1 = 10;

r2 = 50;

[row,col] = size(ori_img);

subplot(2,2,1); imshow(ori_img); title('Original Image');

subplot(2,2,3); imshow(ori_img); title('Original Image');

% Slicing Without Background

for i=1:row

for j=1:col

if ori_img(i,j) >= r1 && ori_img(i,j)<=r2

ori_img(i,j) = (L-1);

else

ori_img(i,j) = 0;

endif

endfor

endfor

subplot(2,2,2); imshow(ori_img); title('Enhanced Image without Background');
```

```
% Slicing With Background
for i=1:row
for j=1:col
if img(i,j) >= r1 && img(i,j)<=r2
img(i,j) = (L-1);
else
img(i,j) = img(i,j);
endif
endfor
endfor
subplot(2,2,4); imshow(img); title('Enhanced Image With Background');
```

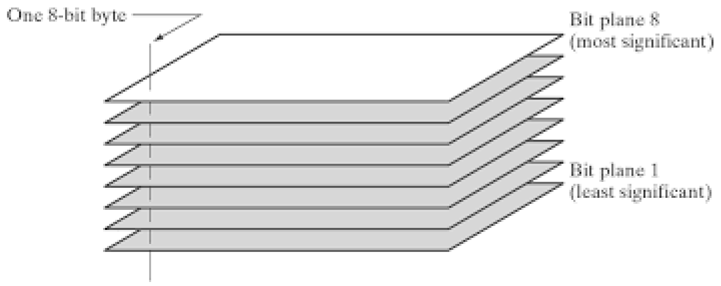
- **Output:**


Original Image
Enhanced Image without Background
Original Image
Enhanced Image With Background

Lab 8: Perform Bit Plane Slicing of RGB image.

- **Theory:**

Pixels are digital number composed of bits, For example, the intensity of each pixel in a 256 gray scale image is composed of 8 bits (i.e. 1 byte). Instead of highlighting intensity-level range, we could highlight the contribution made by each bit, this method is useful and used in image compression.



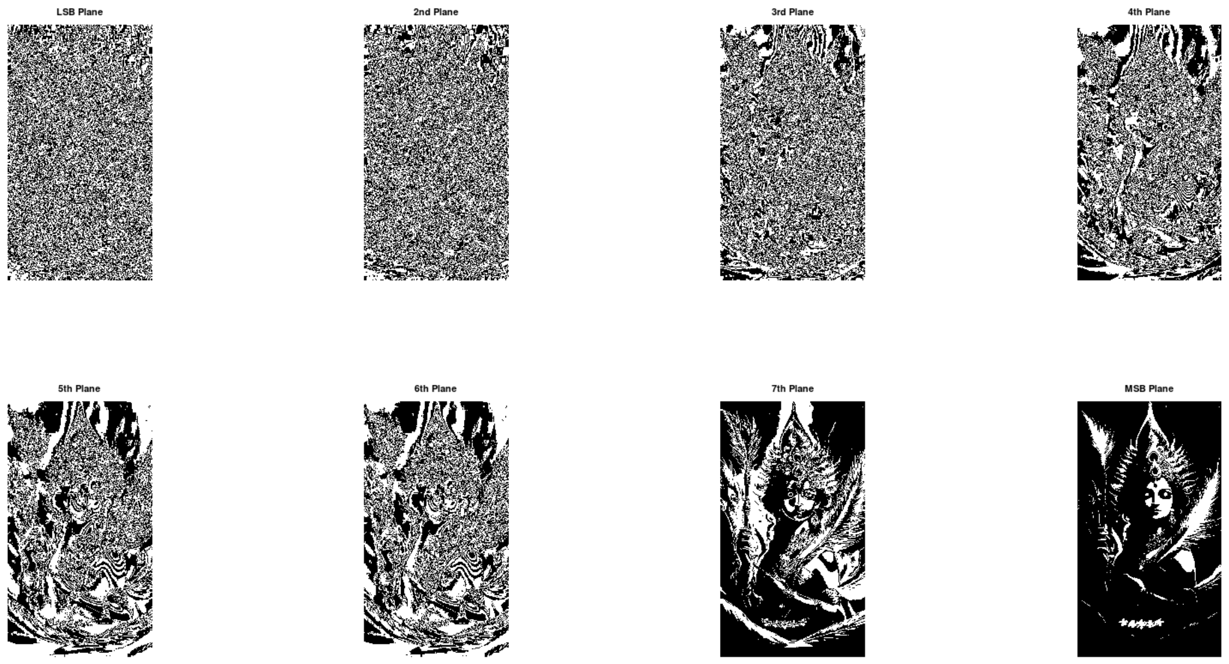
Assume that each pixel is represented by 8 bits, the image is composed of eight 1-bit panes. Plane 0 containing the lowest order bit of all pixels in the image plane remaining 7 are the higher bits.

- **Program Code:**

```
% Take gray image or change rgb to gray first if image is in rgb

img = imread('sample1.jpg');

gray_img = rgb2gray(img);

double_img = double(gray_img);

first_plane = bitget(double_img,1);

subplot(2,4,1); imshow(first_plane); title('LSB Plane');

second_plane = bitget(double_img,2);

subplot(2,4,2); imshow(second_plane); title('2nd Plane');

third_plane = bitget(double_img,3);

subplot(2,4,3); imshow(third_plane); title('3rd Plane');

fourth_plane = bitget(double_img,4);

subplot(2,4,4); imshow(fourth_plane); title('4th Plane');

fifth_plane = bitget(double_img,5);
```

```
subplot(2,4,5); imshow(fifth_plane); title('5th Plane');
sixth_plane = bitget(double_img,6);
subplot(2,4,6); imshow(fifth_plane); title('6th Plane');
seventh_plane = bitget(double_img,7);
subplot(2,4,7); imshow(seventh_plane); title('7th Plane');
eighth_plane = bitget(double_img,8);
subplot(2,4,8); imshow(eighth_plane); title('MSB Plane');
```

- **<u>Output:</u>**



Lab 9: Implement Minimum Filter or Salt Noise Filter.

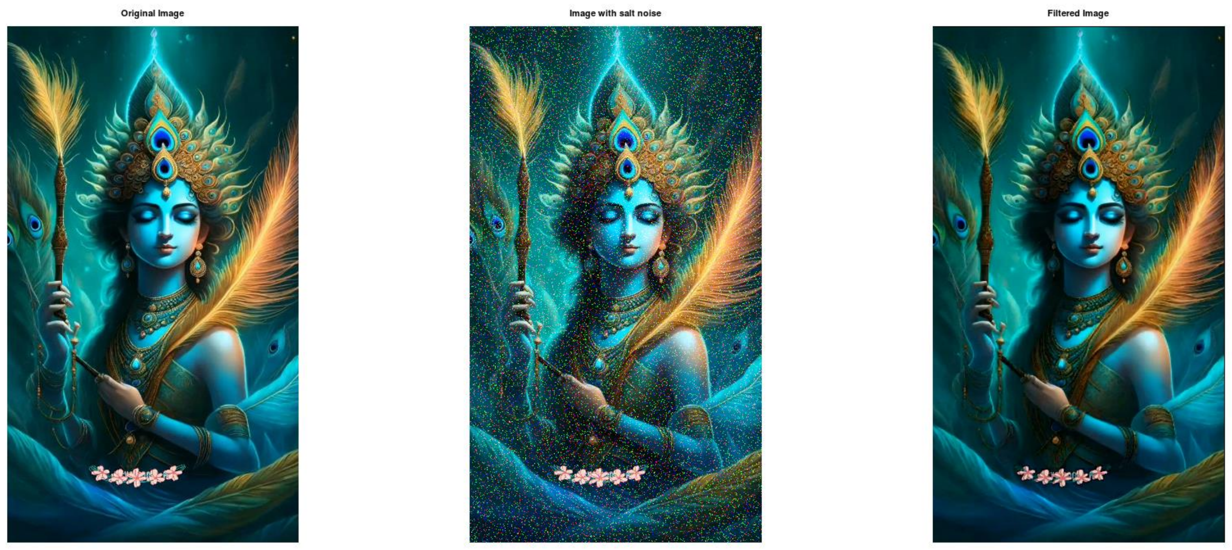
- **Theory:**

The minimum filter selects the smallest value within an ordered window of pixels values and replaces the central pixel with the smallest value (darkest one) in the ordered window. The minimum filters work best for removing salt-type noise. A minimum or low rank filter will tend to darken an image.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

noisy_img = ori_img;

x = rand(size(noisy_img));

noisy_img(x(:)>0.95)=255; %adding salt noise

minf = @(x)min(x(:));

filtered_img = nlfilter(noisy_img,[3,3],minf);

subplot(1,3,1);imshow(ori_img); title('Original Image');

subplot(1,3,2);imshow(noisy_img); title('Image with salt noise');

subplot(1,3,3);imshow(filtered_img); title('Filtered Image');
```

- **Output:**



Lab 10: Implement Maximum Filter or Pepper Noise Filter.

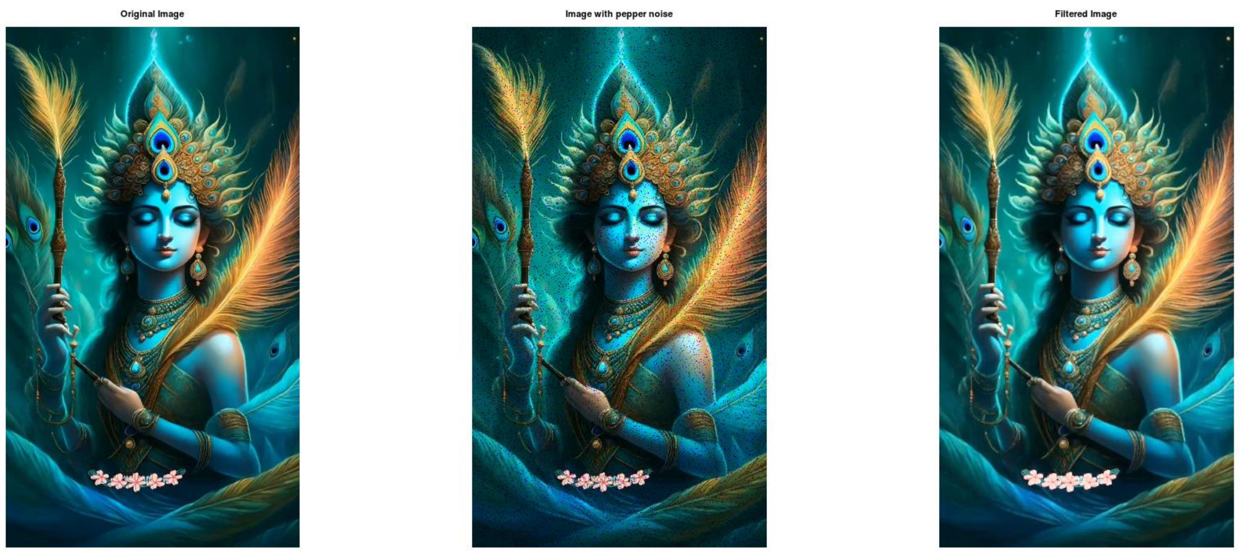
- **Theory:**

The maximum filter selects the largest value within an ordered window of pixels values and replaces the central pixel with the largest value (lightest one). Used to find the brightest points in an image. The maximum filters work best for removing pepper-type noise. Maximum or high rank filter will tend to brighten an image.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

noisy_img = ori_img;

x = rand(size(noisy_img));

noisy_img(x(:)>0.95)=1; %adding pepper noise

maxf = @(x)max(x(:));

filtered_img = nlfilter(noisy_img,[3,3],maxf);

subplot(1,3,1);imshow(ori_img); title('Original Image');

subplot(1,3,2);imshow(noisy_img); title('Image with pepper noise');

subplot(1,3,3);imshow(filtered_img); title('Filtered Image');
```

- **Output:**



Lab 11: Implement Median Filter or Salt and Pepper noise filter.

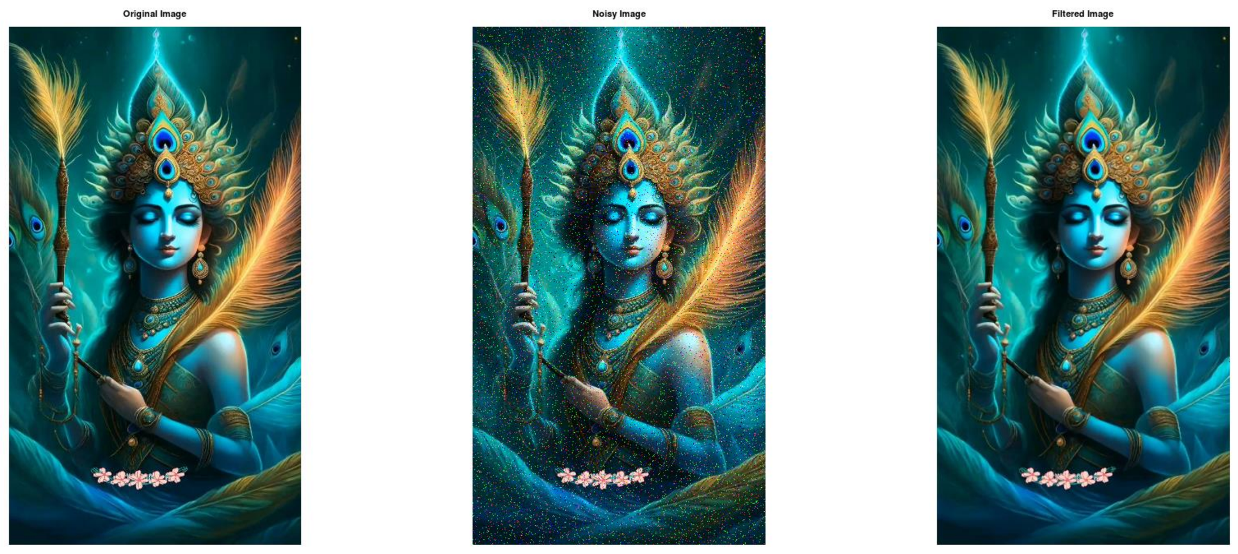
- **Theory:**

The best known order statistic filter is the median filter, which replaces the value of the central pixel by the median of the intensity levels in the neighborhood pixel. To find the median value we have to sort the pixels of given image in ascending order first. Then, Result will replace the central pixel of the given image.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

noisy_img = imnoise(ori_img,'salt & pepper', 0.05); %adding salt and pepper noise

medf = @(x)median(x(:));

filtered_img = nlfilter(noisy_img,[3,3],medf);

subplot(1,3,1); imshow(ori_img); title('Original Image');

subplot(1,3,2); imshow(noisy_img); title('Noisy Image');

subplot(1,3,3); imshow(filtered_img); title('Filtered Image');
```

- **Output:**



Lab 12: Implement Average or Mean Filtering to smoothing the image.

- **Theory:**

This is the simplest of the mean filters, S_{xy} represent the set of coordinates in a rectangular sub-image window (neighborhood) of size m*n, centered at point (x,y). The arithmetic mean filter computes the average value of the corrupted image g(x,y) in the area defined by S_{xy}. The value of the restored image f' at point (x,y) is simply the arithmetic mean computed using the pixels in the region defined by S_{xy}. A mean filter smooth local variations in an image, and noise is reduced as a result of blurring.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

mask1 = (1/9)*[1,1,1;1,1,1;1,1,1];

mask2 = (1/25)*[1,1,1,1,1;1,1,1,1,1;1,1,1,1,1;1,1,1,1,1;1,1,1,1,1];

filtered_img1 = imfilter(ori_img,mask1);

filtered_img2 = imfilter(ori_img,mask2);

subplot(1,3,1); imshow(ori_img); title('Original Image');

subplot(1,3,2); imshow(filtered_img1);title('3x3 filtered Image');

subplot(1,3,3); imshow(filtered_img2);title('5x5 filtered Image');
```

- **Output:**

Original Image
3x3 filtered Image
5x5 filtered Image

Lab 13: Implement Laplacian filter to edge detection.

- **Theory:**

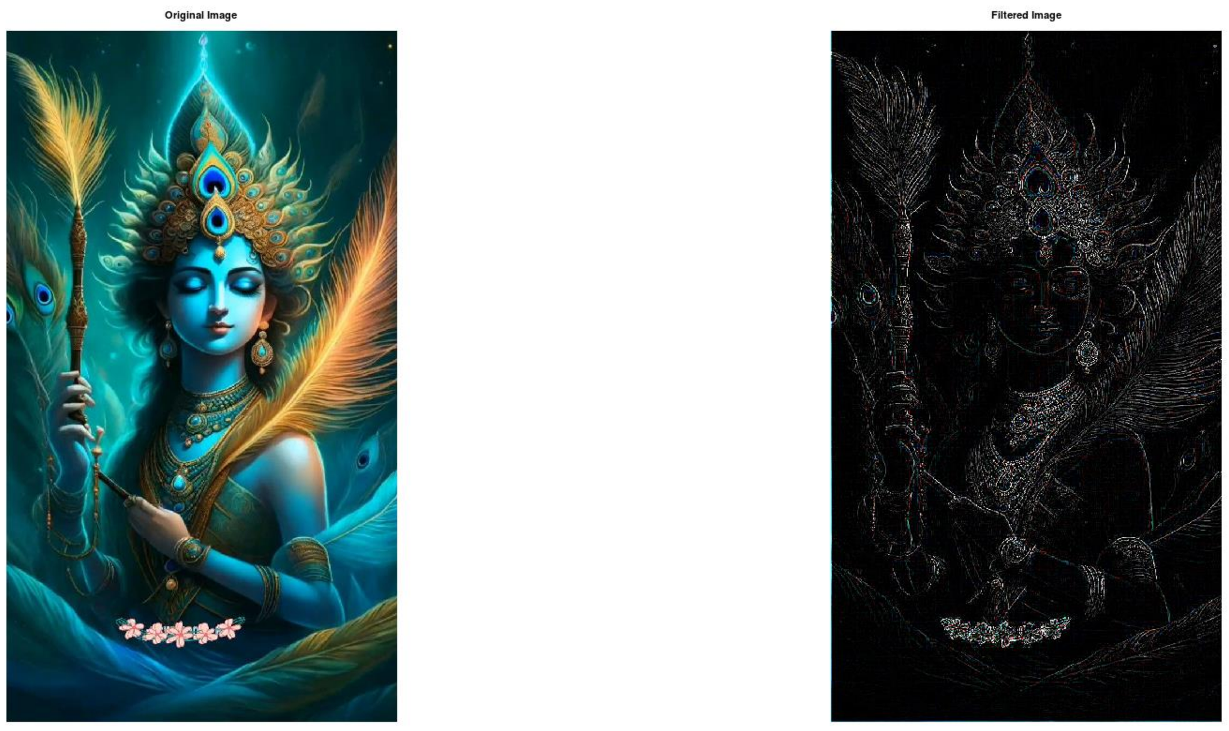
In 1st order derivative filters, we detect the edge along with horizontal and vertical directions separately and then combine both. But using the Laplacian filter we detect the edges in the whole image at once.

The Laplacian Operator/Filter is = [0 1 0; 1 -4 1; 0 1 0] here the central value of filter is negative. Or Filter is = [0 -1 0; -1 4 -1; 0 -1 0], here the central value of filter is positive. The sum of all values of the filter is always 0.

- **Program Code:**

```
ori_img = imread('sample1.jpg');

mask = [-1,-1,-1;-1,8,-1;-1,-1,-1];

filtered_img = imfilter(ori_img,mask);

subplot(1,2,1); imshow(ori_img); title('Original Image');

subplot(1,2,2); imshow(filtered_img); title('Filtered Image');
```

- **Output:**



Lab 14: Implement High-Boost Filter to sharping the image.

- **Theory:**

High boost filtering is an image sharpening technique in digital image processing. It is used to enhance the edges and details of an image without affecting the smooth areas. This technique is applied to an image after using the Laplacian filter to the image. Because, the Laplacian filter highlights the edges and details of the image.

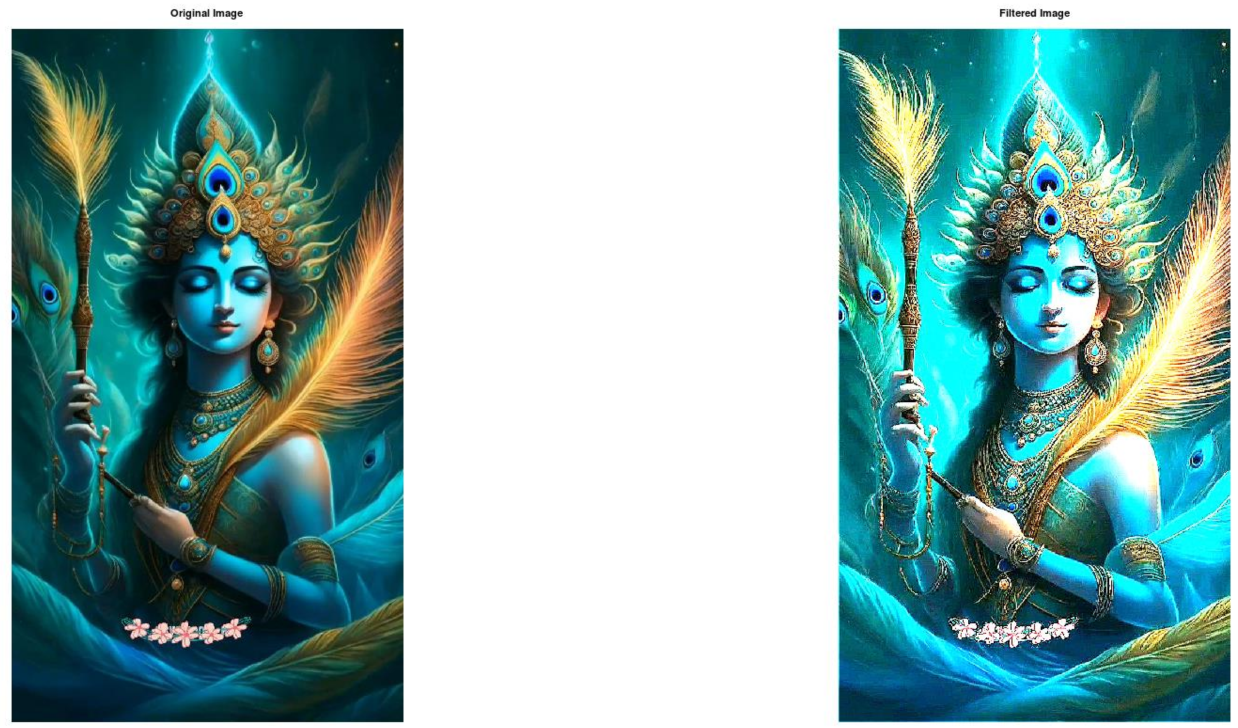
In this technique, the original image is combined with the Laplacian filtered image i.e,

High Boost Filtering = Original Image + (Scaling Factor × Original Image – Laplacian Filtered Image)

- **Program Code:**

```
ori_img = imread('sample1.jpg');
a = 2;
mask = [-1,-1,-1; -1,a+8,-1; -1,-1,-1];
filtered_img = imfilter(ori_img,mask);
subplot(1,2,1); imshow(ori_img); title('Original Image');
subplot(1,2,2); imshow(filtered_img); title('Filtered Image');
```

- **Output:**



<u>Lab 15</u>: Implement Interpolation and Resizing to an image.

- **<u>Theory:</u>**

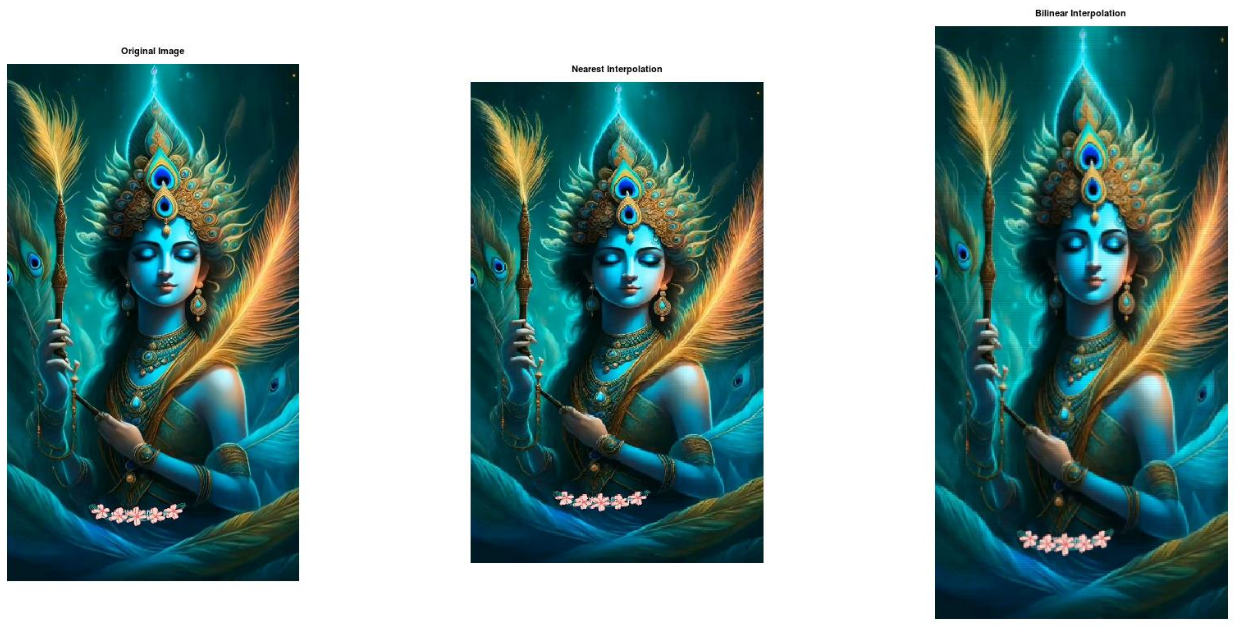
Resizing refers to the process of changing the dimensions (width and height) of an image or signal. It involves adjusting the number of pixels in each dimension to make the image larger or smaller. Resizing is often performed for various reasons, such as preparing images for display on different devices, reducing file sizes, or fitting images into a specific layout.

Interpolation is the process of using known data to estimate values at unknown locations**.** This works in two directions and tries to achieve the best approximation of a pixel's intensity based on the values of surrounding pixels. As it's an approximation method image will always lose some quality when interpolated. Interpolation is commonly employed when you need to resample an image at a different resolution or scale.

- **<u>Program Code:</u>**

```
ori_img = imread('sample1.jpg');
resized_img1 = imresize(ori_img,[512,312],'nearest');
resized_img2 = imresize(ori_img,[512,253],'bilinear');
subplot(1,3,1); imshow(ori_img); title('Original Image');
subplot(1,3,2); imshow(resized_img1); title('Nearest Interpolation');
subplot(1,3,3); imshow(resized_img2); title('Bilinear Interpolation');
```

- **<u>Output:</u>**


Original Image
Nearest Interpolation
Bilinear Interpolation