National University of Sciences & Technology (NUST),
Baluchistan Campus (NBC), Department of Computer Science

CS-433 Digital Image Processing

Assignment – 01

Submitted By

Mustafa Ali | 311297

Submitted To

Dr. Imran Usman

Course Supervisor,

Department of Computer Science

5th Semester
BACHLORS OF COMPUTER SCIENCE
(2019 - 2023)

ASSIGNMENT

Problem: Write **your own** MATLAB Routines for the following:

1. Histogram, Normalized Histogram, Histogram equalization
2. Contrast Stretching
3. Bit-plane slicing
4. Power Law Transformation
5. Intensity Level Slicing (both cases)

All Inputs should be in the form of Images. Design a graphical user interface (E.g. using GUIDE tools in Matlab) which prompts the user to select an image file, initially, from any directory from the computer system. Display the options in points 1-5 in the form of radio buttons. For every transformation (1-5) appropriate range of operations should be provided in the form of text inputs or slide bars, radio buttons etc.

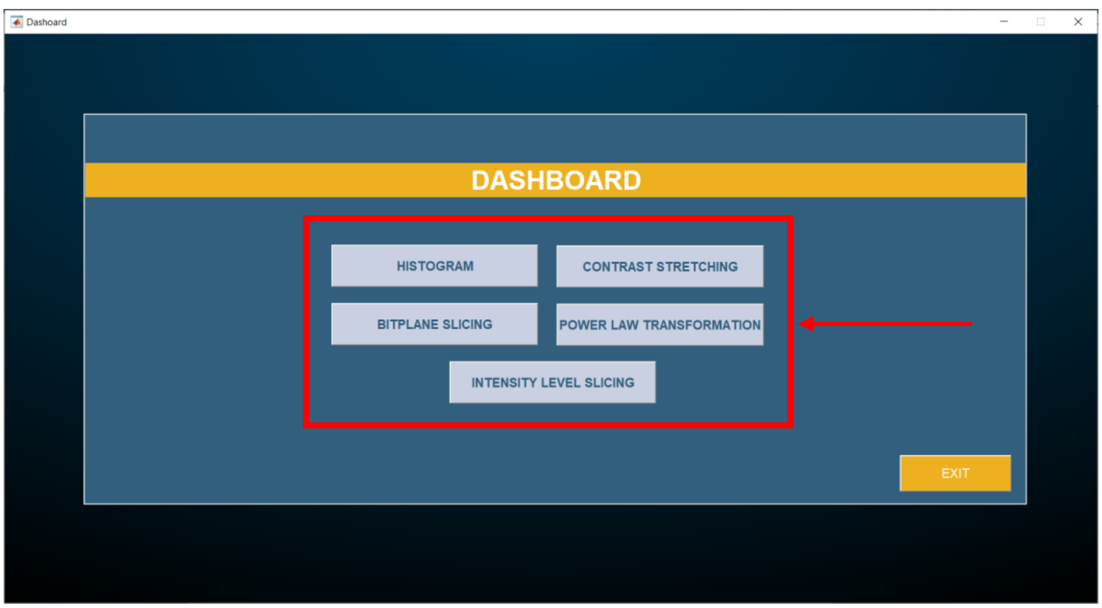
All outputs should be in the form of images, and additionally, in the form of graphs (e.g. in case of histograms, or contrast stretching) demonstrating the initial transformation function or the effect of Transformation after application.

Explanation – In this digital image processing application, we can apply different transformation and make histogram of every image. This stand-alone application have different function like Histogram, Normalized Histogram, Histogram equalization, Contrast Stretching, Bit-plane slicing, Power Law Transformation, Intensity Level Slicing (both cases). We can upload an image from any location in our system and apply these transformation on that selected image. Now I will discuss each Transformations and Image Processing Functions that I used in GUI According to given assignment step by step.

APPLICATION START INTERFACE

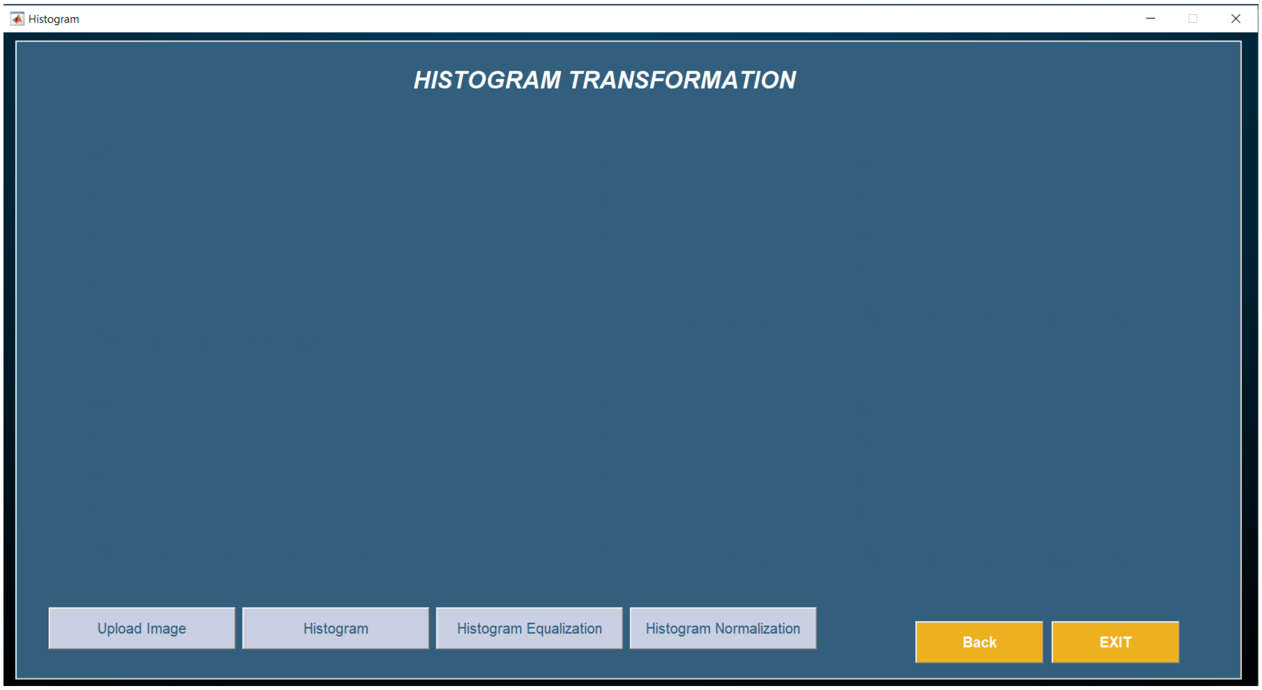
EXPLANATION – When user start the application then it will display this *STARTING* interface and from this user can move to the next screen by pressing start button.

DASHBOARD



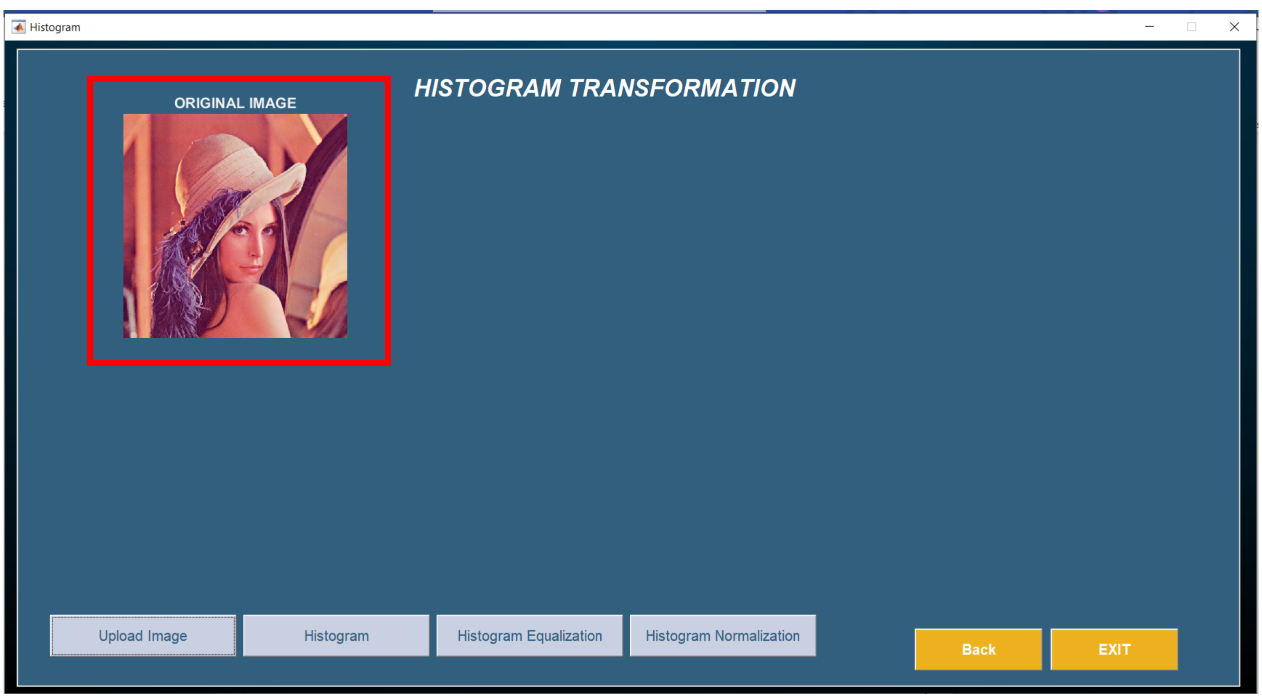
EXPLANATION – After pressing start button, dashboard screen will be appeared in which we have different options such as *Histogram*, *Contrast Stretching*, *Bitplane Slicing etc.*

HISTOGRAM

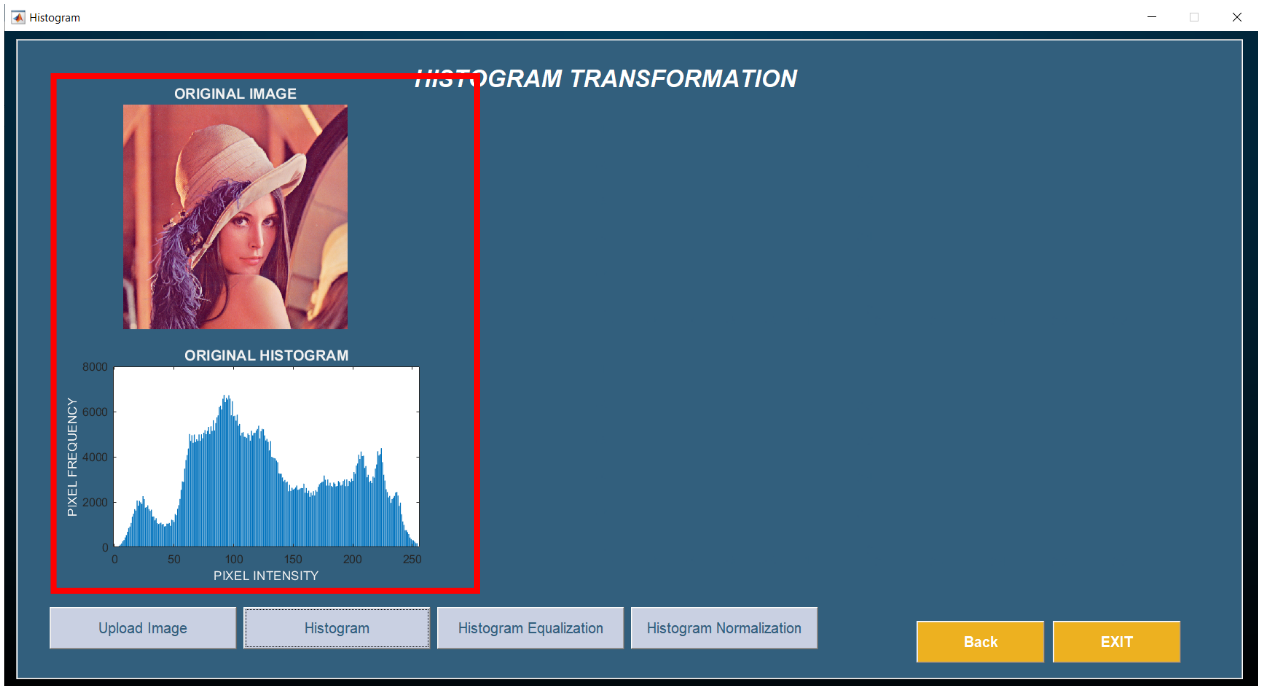


EXPLANATION – In this Histogram Screen we have 4 different functions such as uploading image, displaying histogram of that uploaded image also we can find histogram equalization and histogram normalization.

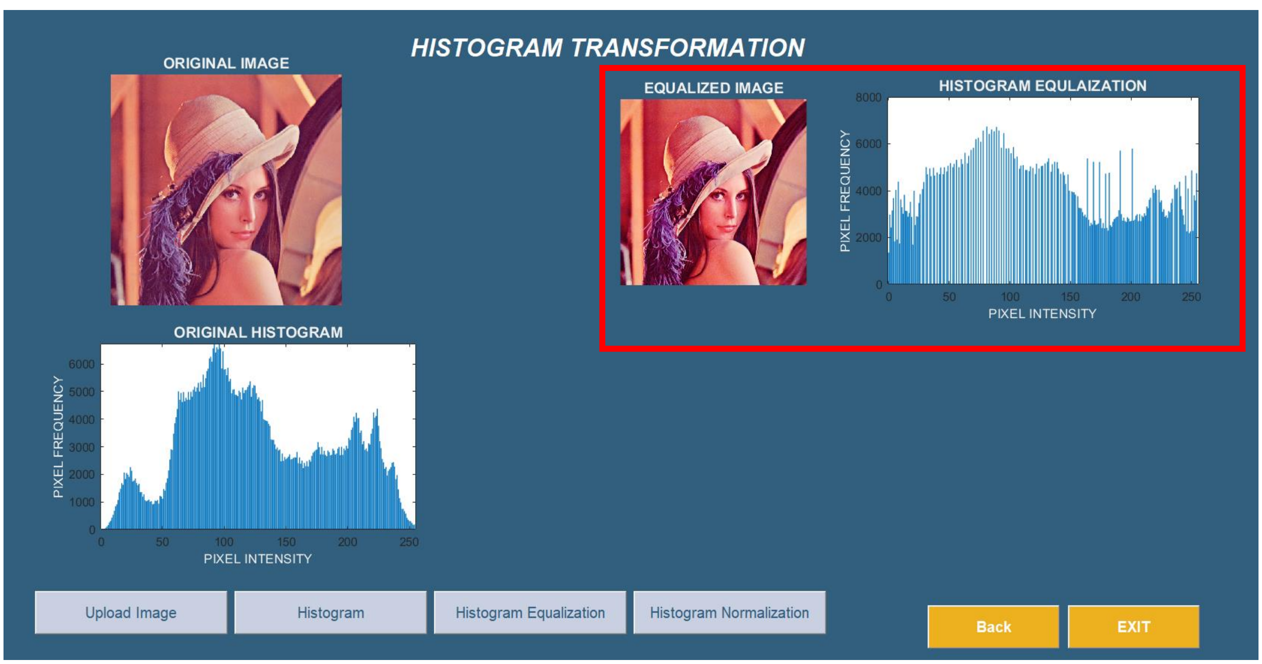
UPLOADING IMAGE



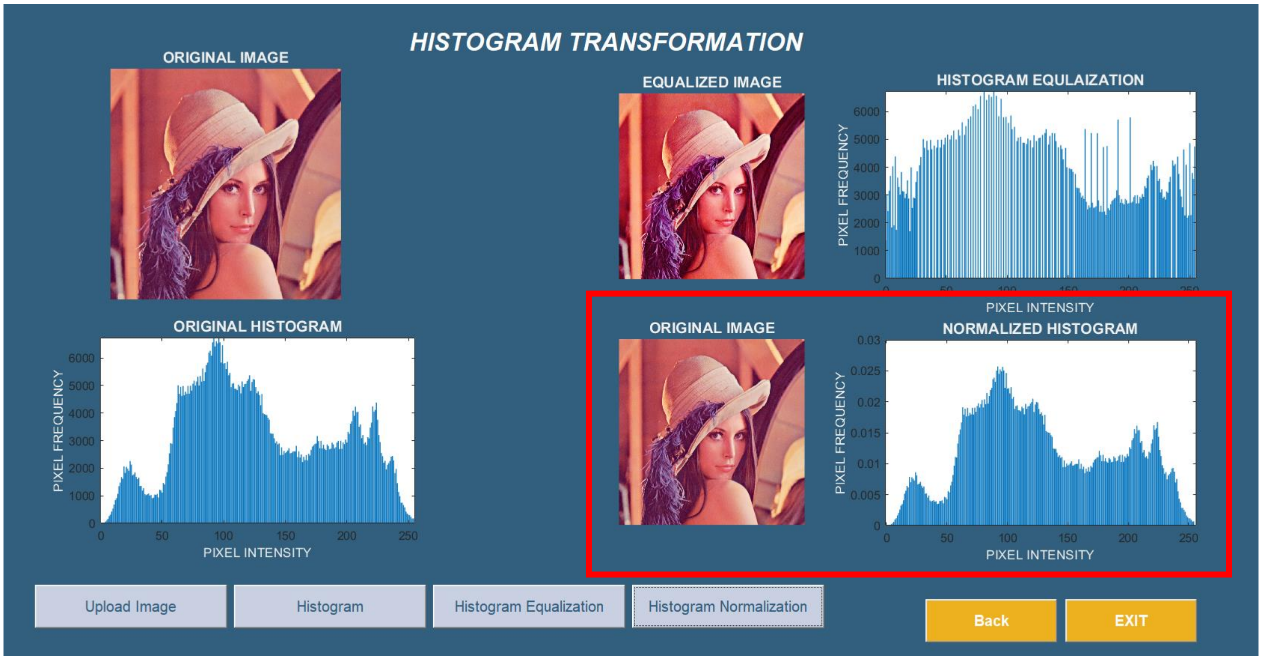
HISTOGRAM OF ORIGINAL IMAGE



HISTOGRAM EQUALIZATION



HISTOGRAM NORMALIZATION



EXPLANATION – As we can, in histogram normalization pixel frequency is between 0 – 1 and there is no such difference between original and normalized histogram. Now we will press back button to move to dashboard screen and will choose any other option.

HISTOGRAM ROUTINE

```
% --- Executes on button press in Histogram_btn.
function Histogram_btn_Callback(hObject, eventdata, handles)
global OriginalImage;

%Histogram Manual Function
x = imread(OriginalImage);
h=zeros(1,256);
[row, col, color]=size(x);
for color = 1:color
    for i=1:row
        for j=1:col
            f = x(i,j, color);
            h(1+f)=h(1+f)+1;
        end
    end
end
axis tight;
%Displaying this histogram to the defined Axes
axes(handles.Histogram);
NumberofPixels = 0: 255;
bar(NumberofPixels, h), title("ORIGINAL HISTOGRAM",'Color', '#F1F1F1' , 'Fontsize', 12);
xlabel("PIXEL INTENSITY",'Color', '#F1F1F1');
ylabel("PIXEL FREQUENCY",'Color', '#F1F1F1');
```

HISTOGRAM EQUALIZATION ROUTINE

```
% --- Executes on button press in Histogram_Equalization_btn.
function Histogram_Equalization_btn_Callback(hObject, eventdata, handles)

% HISTOGRAM EQUALIZATION ROUTINE

global OriginalImage
x = imread(OriginalImage);

h=zeros(1,256);
[r, c]=size(x);
total_no_of_pixels=r*c;

%Calculating Histogram without built-in function of the Image
for i=1:r
    for j=1:c
        h(x(i,j)+1)=h(x(i,j)+1)+1;
    end
end

%%
%Calculating Probability of the Image
for i=1:256
    h(i)=h(i)/total_no_of_pixels;
end
```


```
%%
%Calculating Probability of the Image
for i=1:256
    h(i)=h(i)/total_no_of_pixels;
end
```


```
%%
%Calculating Cumulative Probability of the Image
temp=h(1);
for i=2:256
    temp=temp+h(i);
    h(i)=temp;
end

%%
%Mapping
for i=1:r
    for j=1:c
        x(i,j)=round(h(x(i,j)+1)*255);
    end
end
axis tight;
axes(handles.axes1);
imshow(x), title("EQUALIZED IMAGE",'Color', '#F1F1F1' , 'Fontsize', 12);
axes(handles.axes2);
```

```
%%
%Mapping the equalized Histogram
h=zeros(1,256);
[row, col, color]=size(x);
for color = 1:color
    for i=1:row
        for j=1:col
            f = x(i,j, color);
            h(f+1)=h(f+1)+1;
        end
    end
end
NumberofPixels = 0: 255;
bar(NumberofPixels, h), title("HISTOGRAM EQULAIZATION",'Color', '#F1F1F1' , 'Fontsize', 12);
xlabel("PIXEL INTENSITY",'Color', '#F1F1F1');
ylabel("PIXEL FREQUENCY",'Color', '#F1F1F1');
```

HISTOGRAM NORMALIZATION ROUTINE

```
% --- Executes on button press in Histogram_Normalization_btn.
function Histogram_Normalization_btn_Callback(hObject, eventdata, handles)
% HISTOGRAM NORMALIZATION WITHOUT BUILT IN FUNCTION
global OriginalImage;

x = imread(OriginalImage);
h=zeros(1,256);
[row, col, color]=size(x);
for color = 1:color
    for i=1:row
        for j=1:col
            f = x(i,j, color);
            h(1+f)=h(1+f)+1;
        end
    end
end
axis tight;
axes(handles.axes3);
imshow(x), title("ORIGINAL IMAGE",'Color', '#F1F1F1' , 'Fontsize', 12);
axes(handles.axes4);
NumberofPixels = 0: 255;
h=h/(row*col);
bar(NumberofPixels, h), title("NORMALIZED HISTOGRAM",'Color', '#F1F1F1' , 'Fontsize', 12);
xlabel("PIXEL INTENSITY",'Color', '#F1F1F1');
ylabel("PIXEL FREQUENCY",'Color', '#F1F1F1');
```

CONTRAST STRETCHING

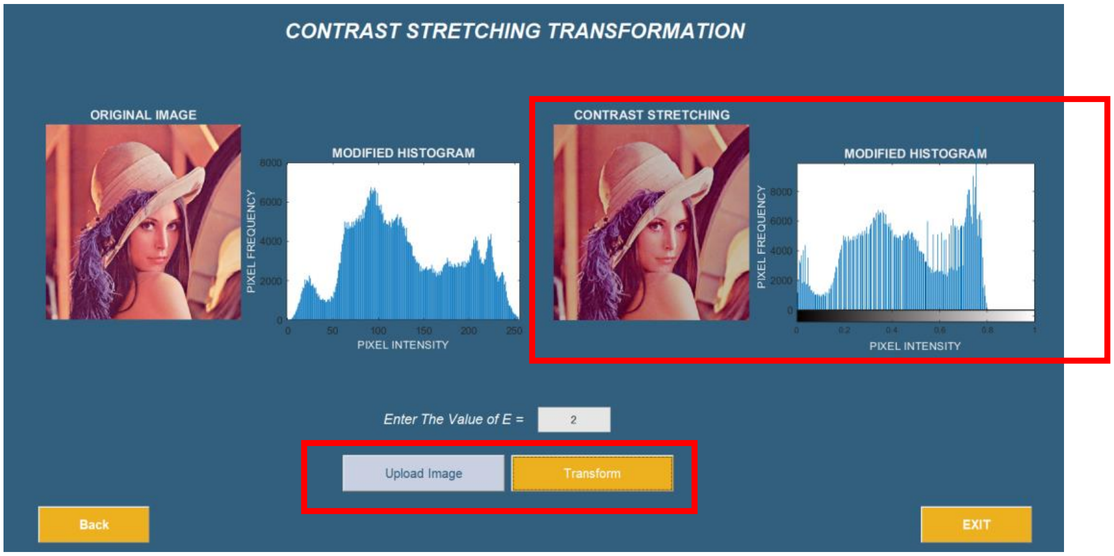
EXPLANATION – Formula for contrast stretching is given below:

g=1./(1 + (m./(double(f) + eps)).^E)

I used this formula to perform contrast stretching transformation, Contrast-stretching transformations increase the contrast between the darks and the lights. In this GUI of Contrast-stretching we can take input of value E which control the slope of the function.

INPUT AND DISPLAYING – 1

INPUT AND DISPLAYING – 2

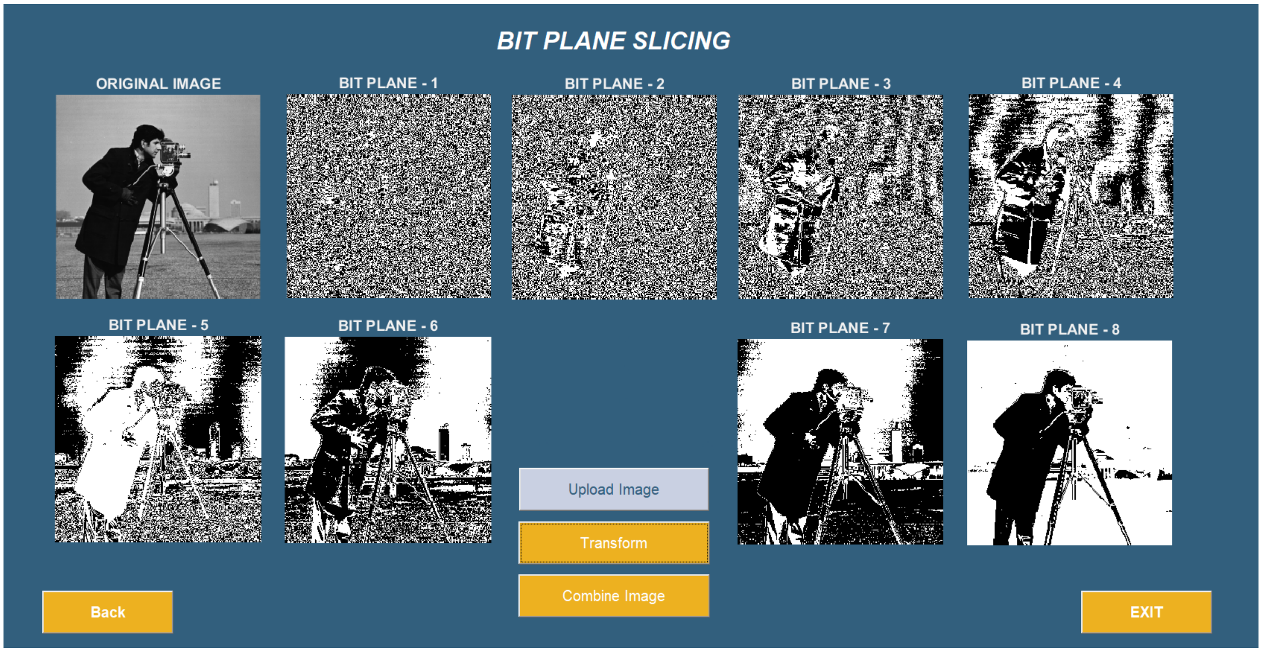


EXPLANATION – In 1st screenshot value of E is 0.5 and in 2nd screenshot value of E is 1, we can see the clear difference between original and modified image as well as in histogram also.

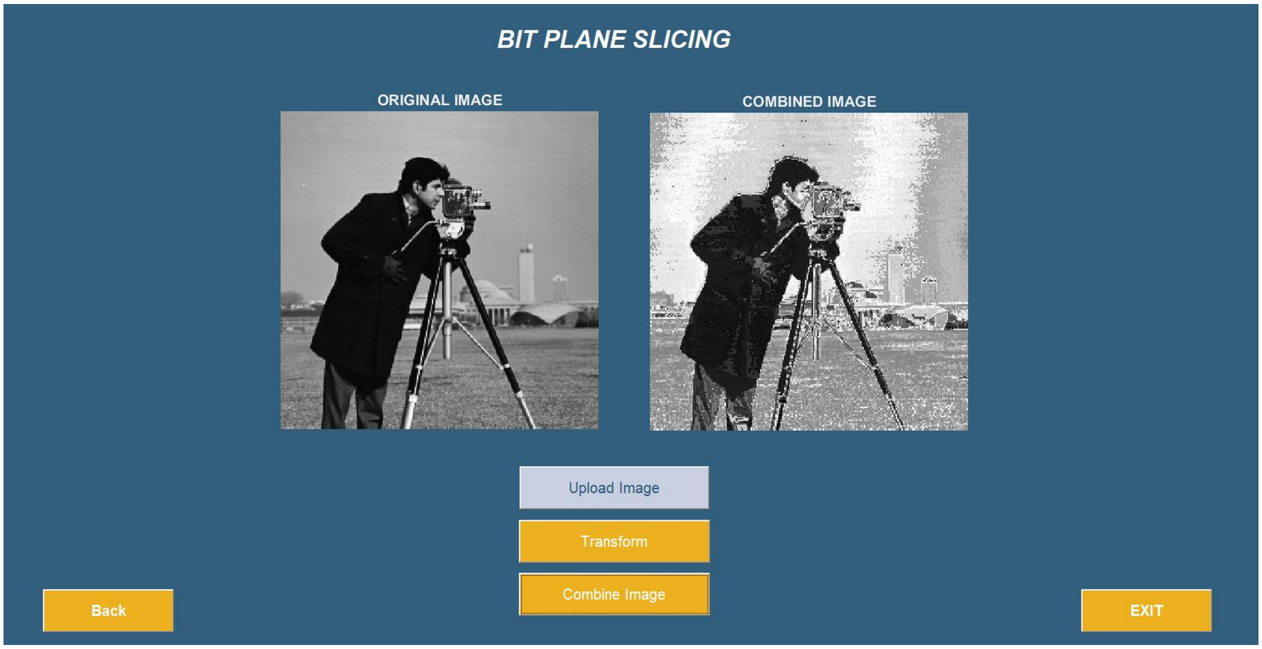
CONTRAST STRETCHING ROUTINE

```
function Contrast_Stretching_btn_Callback(hObject, eventdata, handles)

%CONTRAST STRECTHING
%FORMULA: g=1./(1 + (m./(double(f) + eps)).^E)
global OriginalImage;

%TAKING INPUT
E = str2double(get(handles.E_Value_text,'String'));

contraststretchingimg = imread(OriginalImage);

%TAKING MEAN
m=mean2(contraststretchingimg);

%APPLYING CONTRAST STRETCHING
C = 1./(1 + (m./(double(contraststretchingimg) + eps)).^E);

%DSIPLAYING
axes(handles.axes3);
imshow(C), title("CONTRAST STRETCHING",'Color', '#F1F1F1' , 'Fontsize', 12);
axes(handles.axes4);
imhist(C), title("MODIFIED HISTOGRAM",'Color', '#F1F1F1' , 'Fontsize', 12);
xlh = xlabel("PIXEL INTENSITY",'Color', '#F1F1F1');
xlh.Position(2) = xlh.Position(2) - abs(xlh.Position(2) * 7);
ylabel("PIXEL FREQUENCY",'Color', '#F1F1F1');
```

BITPLANE SLICING



EXPLANATION – In Bitplane slicing I extract bits of the image and displayed it into their dedicated axes. As we know Bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel now, we will recombine the image by pressing *Combine Image* button.



BIT PLANE SLICING ROUTINE

```
% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% BIT PLANE SLICING

delete(handles.Original);
delete(handles.axes2);
delete(handles.axes0);
delete(handles.axes3);
delete(handles.axes4);
delete(handles.axes5);
delete(handles.axes6);
delete(handles.axes7);
delete(handles.axes8);

global OriginalImage;
I2=im2gray(imread(OriginalImage));

axes(handles.axes10);
imshow(I2), title('ORIGINAL IMAGE', 'Color', '#F1F1F1' , 'Fontsize', 12);

Bit1 = mod(I2, 2);
Bit2 = mod(floor(I2/2), 2);
Bit3 = mod(floor(I2/4), 2);
Bit4 = mod(floor(I2/8), 2);
Bit5 = mod(floor(I2/16), 2);
Bit6 = mod(floor(I2/32), 2);
Bit7 = mod(floor(I2/64), 2);
Bit8 = mod(floor(I2/128), 2);
combine = (2 * (2 * (2 * (2 * (2 * (2 * (2 * Bit8 + Bit7) + Bit6) + Bit5) + Bit4) + Bit3) + Bit2) + Bit1);

axes(handles.axes11);
imshow(combine), title('COMBINED IMAGE', 'Color', '#F1F1F1' , 'Fontsize', 12);
```

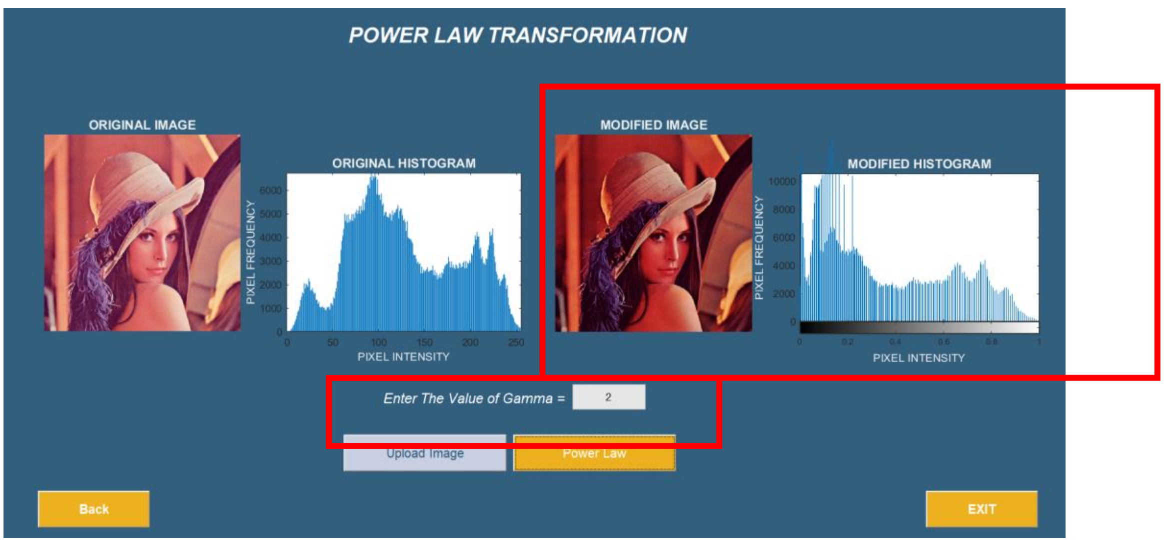
POWER LAW TRANSFORMATION

EXPLANATION – Formula for contrast Power Law is given below:

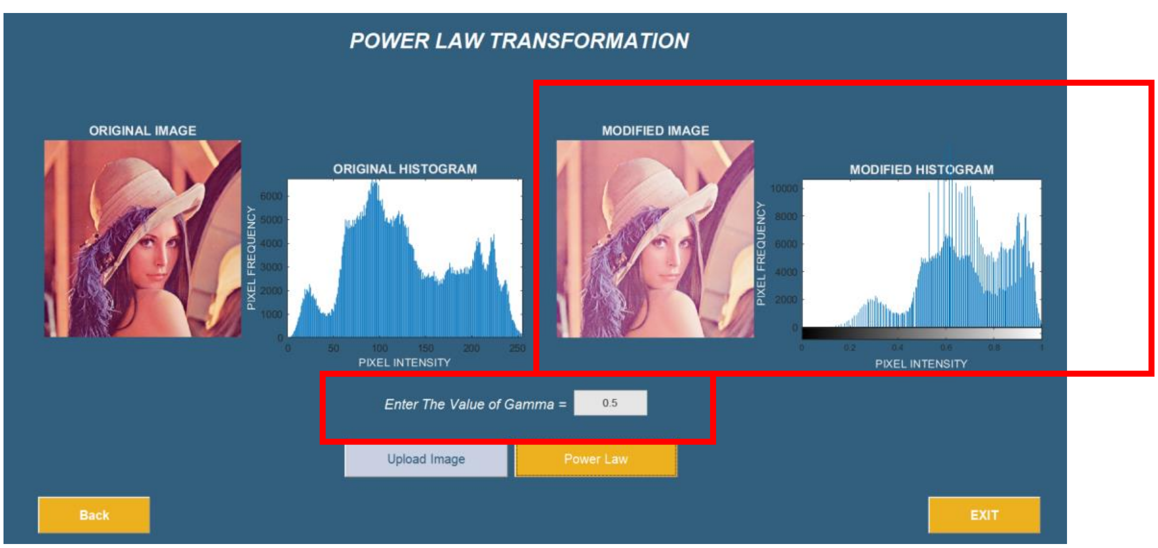
$$s = c * r^{\gamma}$$

I used this formula to perform Power Law Transformation, using y value which gamma value we can curve the grayscale either darken the intensity when value of gamma is greater than one or brighter the intensity when value of gamma is less than one. Now we will apply it to the image.

GAMMA > 1 (DARK IMAGE)



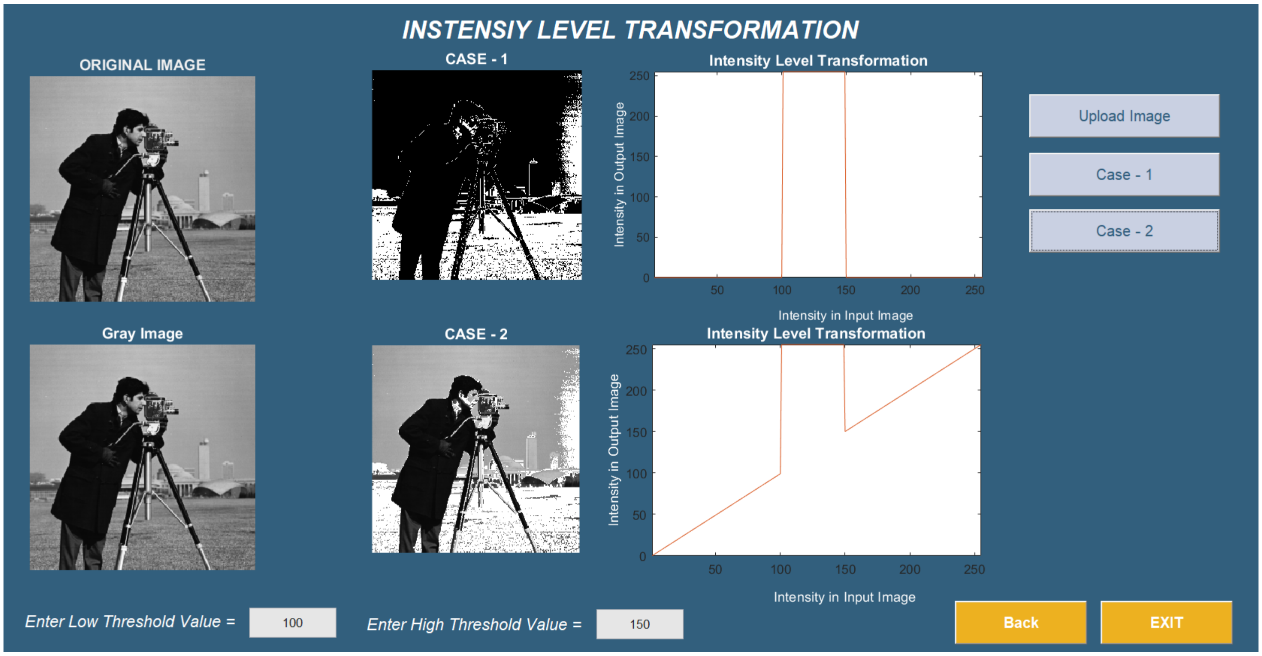
GAMMA < 1 (BRIGHTER IMAGE)



POWER LAW TRANSFORMATION ROUTINE

```
% --- Executes on button press in Power_Law_btn.
function Power_Law_btn_Callback(hObject, eventdata, handles)
% POWER LAW TRANSFORMATION WITHOUT BUILT IN FUNCTION
% Formula s = c*r^y

%TAKING INPUT
y = str2double(get(handles.Gamma_Value_text,'String'));
global OriginalImage;
pwtransformationimg = imread(OriginalImage);
img = im2double(pwtransformationimg);
c=1;
% APPLYING POWER LAW
s = c*img.^y;

%DSIPLAYING
axis tight;
axes(handles.axes3);
imshow(s), title("MODIFIED IMAGE",'Color', '#F1F1F1' , 'Fontsize', 12);
axes(handles.axes4);
imhist(s), title("MODIFIED HISTOGRAM",'Color', '#F1F1F1' , 'Fontsize', 12);
xlh = xlabel("PIXEL INTENSITY",'Color', '#F1F1F1');
xlh.Position(2) = xlh.Position(2) - abs(xlh.Position(2) * 7);
ylabel("PIXEL FREQUENCY",'Color', '#F1F1F1');
```

INTENSITY LEVEL SLICING



EXPLANATION – In Intensity Level Slicing we have two cases which discussed below:

Case – 1: In this case we will set all the pixels values in a range of interest to one value which is white and all others to another value which is black.

Case – 2: In this case we will set all the pixels values in a range of interest to one value which can be white or black and all others to another will be unchanged value.

As you can see, I am taking the threshold values from the user which can be any number between 0 – 255, in above implementation low threshold value is 100 and high threshold value is 150. We use this slicing to highlight specific part of the image by specifying the range of intensity of image.

INTESITY LEVEL SLICING ROUTINE (CASE – 1)

```
function Case_1_btn_Callback(hObject, eventdata, handles)

%CASE - 01
lt = str2double(get(handles.LowThreshold_Value_text,'String'));
ht = str2double(get(handles.HighThreshold_Value_text,'String'));
global OriginalImage;

x = im2gray(imread(OriginalImage));

% APPLYING INTENSITY LEVEL SLICIING
y=x;
[w h]=size(x);
for i=1:w
    for j=1:h
        if x(i,j)>=lt && x(i,j)<=ht y(i,j)=255;
        else y(i,j)=0;
        end
    end
end

%MAPPING
dd=[];
hold on;
dd(1:100)=0;
dd(101:149)=255;
dd(150:256)=0;
axis tight;
```

```
%DISPLAYING
axes(handles.Gray);
imshow(x), title("Gray Image",'Color', 'white' , 'Fontsize', 12);
axes(handles.axes2);
imshow(y), title("CASE - 1",'Color', 'white' , 'Fontsize', 12);
axes(handles.axes3);
plot(dd), title("Intensity Level Transformation",'Color', 'white' , 'Fontsize', 12);
xlh = xlabel("Intensity in Input Image",'Color', 'white');
xlh.Position(2) = xlh.Position(2) - abs(xlh.Position(2) * 0.1);
ylabel("Intensity in Output Image",'Color', 'white');
```

INTESITY LEVEL SLICING ROUTINE (CASE – 2)

```
function Case_2_btn_Callback(hObject, eventdata, handles)
%CASE - 02

lt = str2double(get(handles.LowThreshold_Value_text,'String'));
ht = str2double(get(handles.HighThreshold_Value_text,'String'));
global OriginalImage;

% APPLYING INTENSITY LEVEL SLICIING
x = im2gray(imread(OriginalImage));
y=x;
[w, h]=size(x);
for i=1:w
    for j=1:h
        if x(i,j)>=lt && x(i,j)<=ht y(i,j)=255;
        else y(i,j)=x(i,j);
        end
    end
end

%MAPPING
g2=[];
hold on;
g2(1:100)=0:99;
g2(101:149)=255;
g2(150:255)=150:255;
axis tight;
%DISPLAYING
axes(handles.axes10);
imshow(y), title("CASE - 2",'Color', 'white' , 'Fontsize', 12);
axes(handles.axes12);
plot(g2), title("Intensity Level Transformation",'Color', 'white' , 'Fontsize', 12);
xlh = xlabel("Intensity in Input Image",'Color', 'white');
xlh.Position(2) = xlh.Position(2) - abs(xlh.Position(2) * 0.1);
ylabel("Intensity in Output Image",'Color', 'white');
```

OTHER FUNCTIONS (EXIT, BACK, UPLOAD)

UPLOAD BUTTON CODE

```
% --- Executes on button press in Upload_btn.
function Upload_btn_Callback(hObject, eventdata, handles)

global OriginalImage;
global FileName;
global PathName;

%using it to browse the file of an image
[FileName, PathName] = uigetfile('*.png;*.bmp;*.jpg', 'File Selector');
OriginalImage = strcat(PathName, FileName);
I = imread(OriginalImage);
axes(handles.Original_Image);
imshow(I), title("ORIGINAL IMAGE",'Color', '#F1F1F1' , 'Fontsize', 12);
```

BACK BUTTON CODE

```
% --- Executes on button press in Back_btn.
function Back_btn_Callback(hObject, eventdata, handles)
%it will direct to the dashboard window
Dashoard
%it will close the Histogram window
close(Histogram)
```

EXIT BUTTON CODE

```
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
pause(0);
close();
close();
```

NOTE: FILE OF THIS APPLICATION IS ATTACHED WITH THIS PDF.

-- THE END --