Indian Institute of Information Technology, Nagpur

Digital Image Processing Lab Report

Submitted By :
Anirudh Senani D(BT17ECE073)
Semester VI
Electronics and Communication Engineering Dept.

Submitted To :
Dr. Tapan Kumar Jain
Assistant Professor
Electronics and Communication Engineering Dept.

Experiment 1:

Histogram equalization

AIM:- To write a program to do histogram equalisation of an

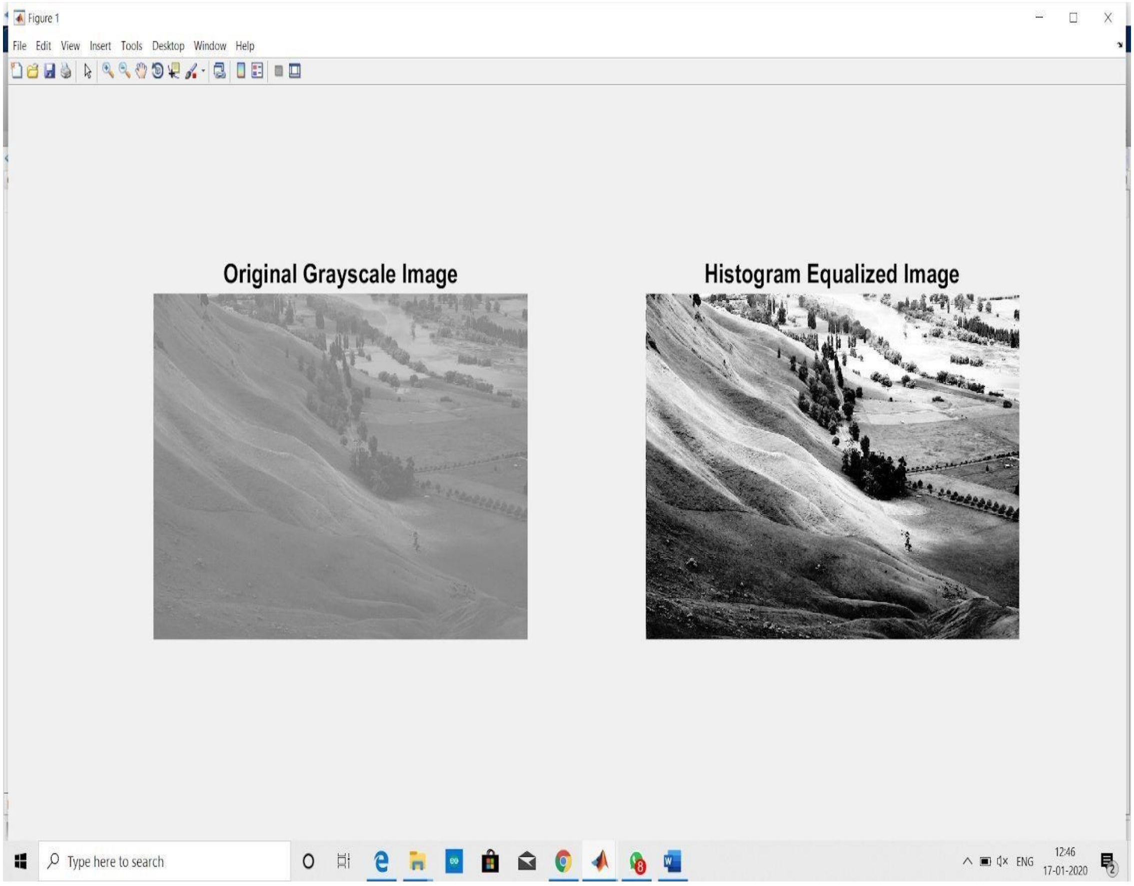
image.

SOFTWARE:- Matlab

CODE:-

```
%Image Histogram Equilization
clc;
close all;
clear all;

image = imread('C:\Users\ANIRUDH SENANI\Downloads\histeq.jpg');

image1 = histeq(image);
figure(1)
subplot(2,2,1)
imshow(image)
title('Before Equalization')
subplot(2,2,3)
histogram(image)

subplot(2,2,2)
imshow(image1)
title('After Equalization')
subplot(2,2,4)
histogram(image1)
```

OUTPUT:-



EXPERIMENT 2:

RGB layer image

AIM:- To write a program to split an image into its individual

layers i.e RGB

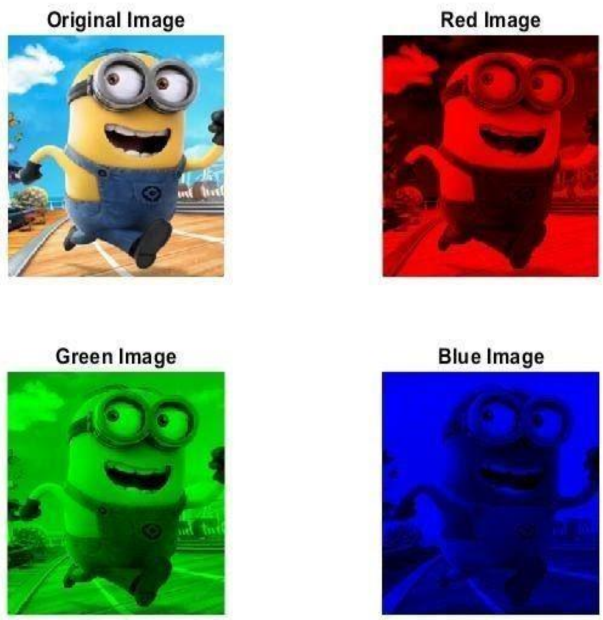
SOFTWARE:- Matlab

CODE:-

```
% Red Green and Blue Layers of an Image
%%   clc; clear;
close all
;

% Reading Test Image test_image = imread('Minion.png'  ) ;
```

```
% Resizing Test Image to 256x256x3 test_image
= imresize(test_image,[256 256]);

% Displaying Test Image figure('Name' ,
'Result' ) ; subplot(221);
imshow(test_image); title('Original
Image' ) ;

% Displaying Only Red Image
red_image = test_image;
red_image(:,:,[2 3]) = 0 ; subplot(222);
imshow(red_image); title('Red Image' ) ;

% Displaying Only Green Image
green_image = test_image; green_image(:,:,[1
3]) = 0 ; subplot(223); imshow(green_image);
title('Green Image' ) ;

% Displaying Only Blue Image blue_image
= test_image; blue_image(:,:,[1 2]) = 0 ;
subplot(224); imshow(blue_image);
title('Blue Image') ;
```

Output:



<u>EXPERIMENT 3:</u>

<u>AIM</u>:- To write a program do bit slicing of the given image and reconstruct the given image after compression

<u>SOFTWARE</u>:- Matlab

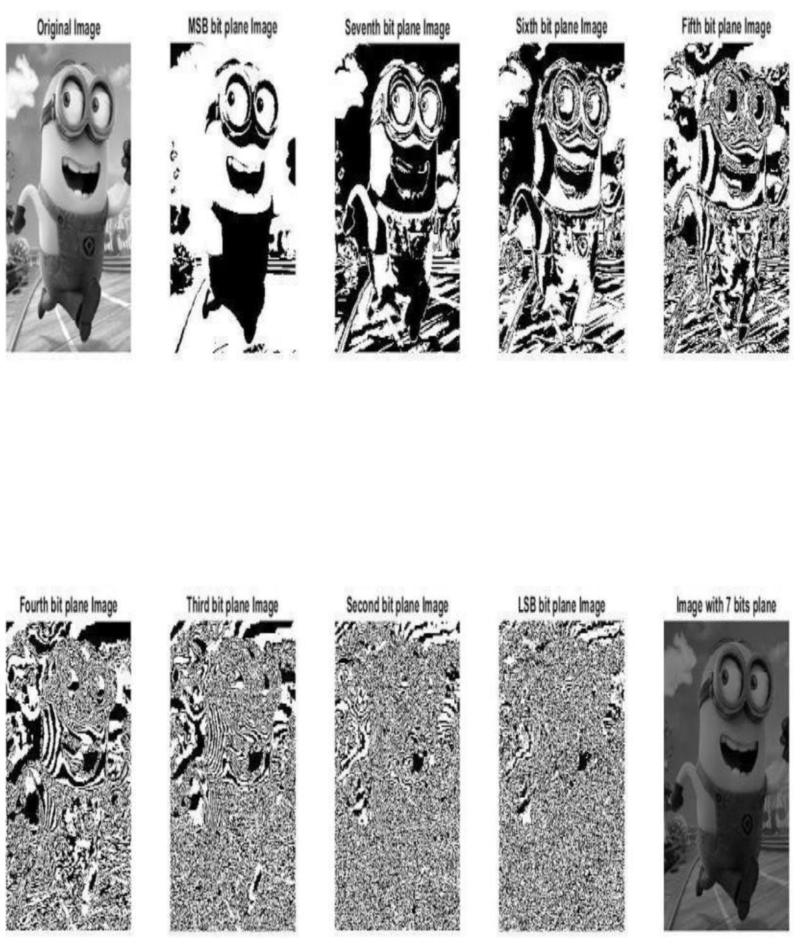
<u>CODE</u>:-

```
% Bit Plane Slicing of an Image
clc; clear; close all ;

% Reading Test Image test_image = imread('Minion.png' ) ;
% Resizing Test Image to 256x256x3 test_image
= imresize(test_image,[256 256]);

% Converting test image to grayscale
gray_test_image = rgb2gray(test_image);
figure('Name' , 'Result'     ) subplot(251);
imshow(gray_test_image); title('Original Image' ) ;

%% Plotting MSB Bit Plane Image

% Getting MSB bit of each pixel msb_bit_plane =
bitget(gray_test_image,
8); subplot(252);
imshow(logical(msb_bit_plane)); title('MSB bit plane Image' ) ;

%% Plotting Seventh Bit Plane Image    % Getting seventh bit
of each pixel seven_bit_plane = bitget(gray_test_image, 7);
subplot(253); imshow(logical(seven_bit_plane)); title('Seventh
bit plane Image' ) ;
%% Plotting Sixth Bit Plane Image    % Getting sixth bit of each
pixel sixth_bit_plane = bitget(gray_test_image, 6);
subplot(254); imshow(logical(sixth_bit_plane)); title('Sixth bit
plane Image' ) ;
```

```
%% Plotting Fifth Bit Plane Image    % Getting fifth bit of each
pixel fifth_bit_plane = bitget(gray_test_image, 5);
subplot(255); imshow(logical(fifth_bit_plane)); title('Fifth bit
plane Image' ) ;


%% Plotting Fourth Bit Plane
Image
% Getting fourth bit of each pixel fourth_bit_plane =
bitget(gray_test_image,
4);                                    subplot(256);
imshow(logical(fourth_bit_plane));    title('Fourth
bit plane Image' ) ;


%% Plotting Third Bit Plane Image    % Getting third bit of each
pixel third_bit_plane = bitget(gray_test_image, 3);
subplot(257); imshow(logical(third_bit_plane)); title('Third bit
plane Image' ) ;


%% Plotting Second Bit Plane
Image
% Getting second bit of each pixel second_bit_plane =
bitget(gray_test_image,
2);                                    subplot(258);
imshow(logical(second_bit_plane)); title('Second
bit plane Image' ) ;


%% Plotting LSB Bit Plane Image


% Getting lsb of each pixel lsb_bit_plane =
bitget(gray_test_image, 1); subplot(259);
imshow(logical(lsb_bit_plane)); title('LSB bit plane Image' ) ;

%% Image of 7 bits except LSB


[rows cols] = size(gray_test_image);
seven_bits_image = zeros(rows,cols);    for ii =
1:rows      for jj = 1:cols
```

```
    % Getting seven bits except lsb          temp =
bitget(gray_test_image(ii,jj), 2:1:8);
    % Converting it to decimal
value                          temp2 = bi2de(temp);
% Storing these values         seven_bits_image(ii,jj) =
temp2;    end  end   subplot(2,5,10);
imshow(uint8(seven_bits_image));  title('Image with 7
bits plane') ;
```

<u>OUTPUT</u>:-



EXPERIMENT 4:

AIM:- To write a program to perform nearest neighbourhood

pixels algorithm

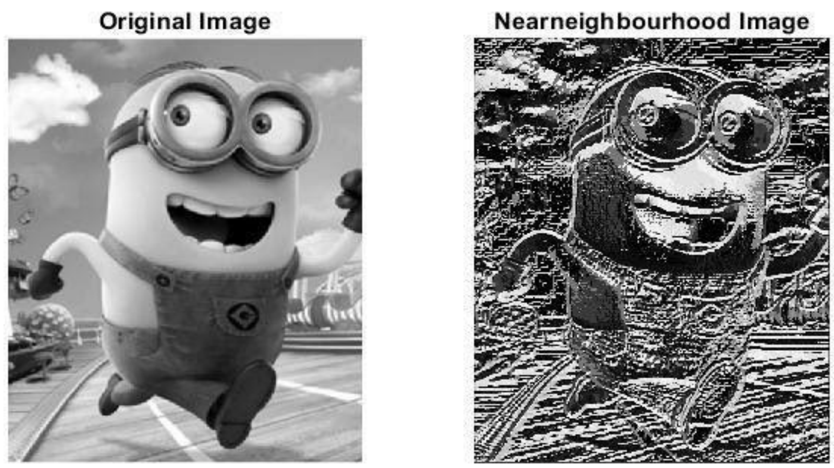
SOFTWARE:- Matlab

CODE:-

```
clc; clear
all; close
all;

I = imread('Minion.png');
I = rgb2gray(I);

[y,x] = size(I); new =
[zeros(y,1),I,zeros(y,1)]; new =
[zeros(1,x+2);new;zeros(1,x+2)];


I_New = zeros(y,x); for Row
= 1:y
   for Col = 1:x         ithrow = Row+1; %Save the current Row
and Col Values         ithcol = Col+1;

       ithpixel = new(ithrow,ithcol);

       newpixel = 1*(ithpixel<new(ithrow,ithcol+1))+
2*(ithpixel<new(ithrow-1,ithcol+1))+...
4*(ithpixel<new(ithrow-1,ithcol))+ 8*(ithpixel<new(ithrow-
1,ithcol-1))+...             16*(ithpixel<new(ithrow,ithcol-1))+
32*(ithpixel<new(ithrow+1,ithcol-1))+...
64*(ithpixel<new(ithrow+1,ithcol))+
```

```
128*(ithpixel<new(ithrow+1,ithcol+1)); %Calculation of New
Value of Pixel

        I_New(Row,Col) = newpixel;
end end
I_New = uint8(I_New); %Convert to Unsigned 8-Bit Data Type

figure(2); subplot(121); imshow(I);
title('Original Image');

subplot(122); imshow(I_New); title('Nearneighbourhood
Image');
```

OUTPUT:



EXPERIMENT 5:

AIM:- To write a program to perform Histogram Stretching on

an image

SOFTWARE:- Matlab

CODE:-

```
clc; clear;
close all;

% Reading Test Image test_image =
imread('Lenna.png');
figure('Name','Original Test Image'); imshow(test_image);
title('Original
Test Image');

% Converting Test Image to Gray Scale test_image_gray
= rgb2gray(test_image);

[rows, cols] = size(test_image_gray);

%% Number of occurance of each gray scale level

num_pixel = zeros(1,256);

for ii = 1:rows     for jj = 1:cols
num_pixel( test_image_gray(ii,jj) + 1 ) =
num_pixel( test_image_gray(ii,jj) + 1 ) + 1;     end end


%% Commulative Distribution Function (c.d.f.)

cdf_num_pixel = cumsum(num_pixel);
```

```
%% Histogram Equalization

% h(v) = round((cdf(v) - cdf_min)*(L-1))
%             ------------------------ %
((MxN) - cdf_min)
%
% M = rows , N = cols

h_v = round((cdf_num_pixel - 1) ./ (512*512 - 1) .* 255);


equilized_image= zeros(size(test_image_gray));

for ii=1:rows
   for jj= 1:cols
t=(test_image_gray(ii,jj));
equilized_image(ii,jj)=h_v(t);     end
end

%% Plotting Histogram of test as well as equalized image

% Plotting histogram of test image figure('Name','Histogram
of test as well as equalized image'); subplot(121);
imhist(test_image_gray); title('Histogram of test image');

% Plotting histogram of Histogram Equalized Image subplot(122);
imhist(uint8(equilized_image)); title('Histogram of
Histogram Equalized Image');

suptitle('Plotting Histogram of test as well as equalized image');

%% Displaying test as well as equalized image without using
Built-in Command

% Displaying test image
figure('Name','Histogram Equalization without built-in
```

command'); subplot(121); imshow(test_image_gray); title('Original Image');

% Displaying Histogram Equalized Image subplot(122); imshow(uint8(equilized_image)); title('Histogram Equalized Image');

suptitle('Histogram Equalization without using Built-in Command');

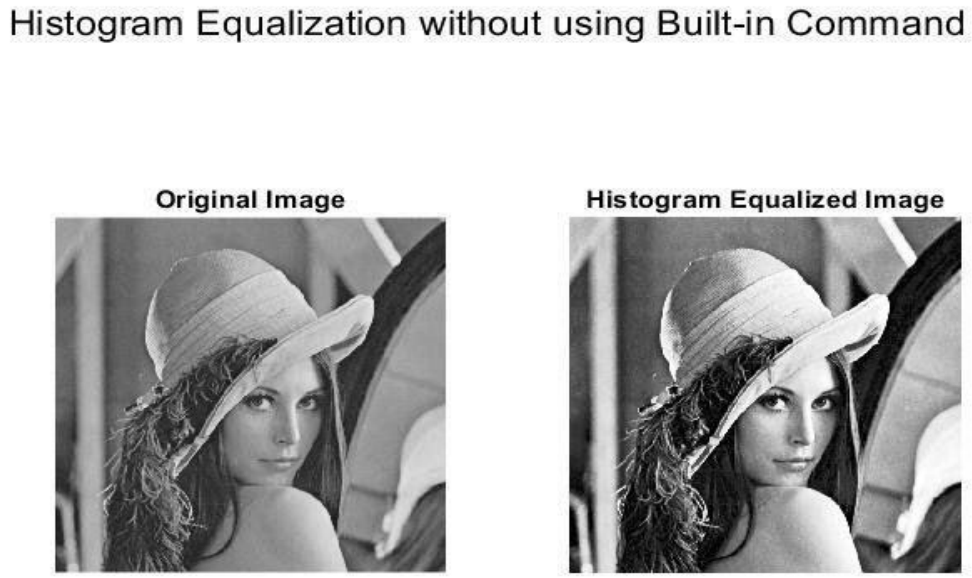
%% Histogram Equalization using Built-in Command

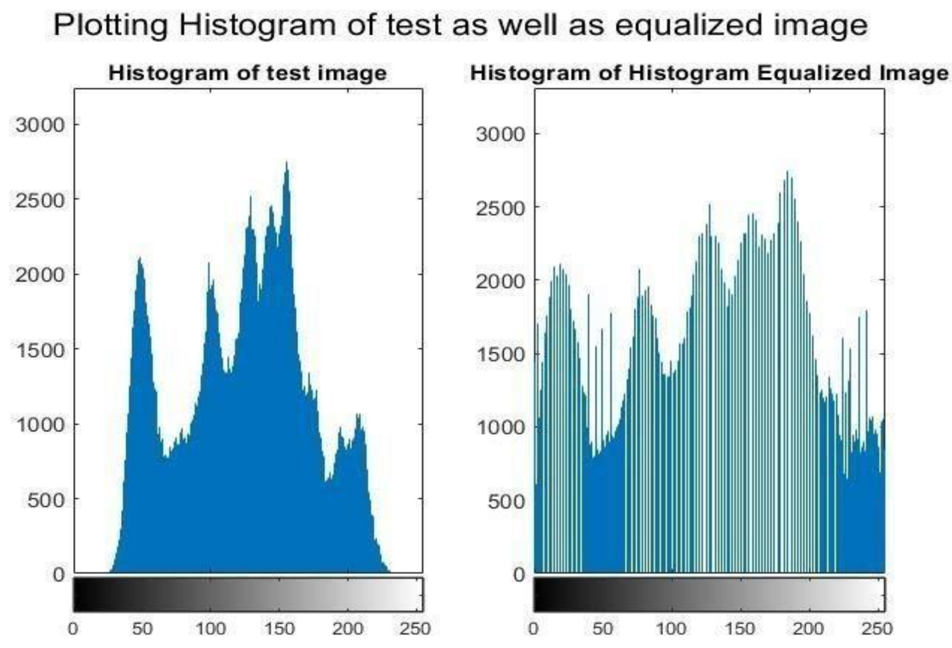
figure('Name','Histogram Equalization using Built-in Command'); subplot(121); imshow(test_image_gray); title('Original Test Image');

subplot(122); histeq(test_image_gray); title('Histogram Equalized Image');

suptitle('Histogram Equalization using Built-in Command');

OUTPUT:


Histogram Equalization without using Built-in Command
Original Image
Histogram Equalized Image


Plotting Histogram of test as well as equalized image
Histogram of test image
Histogram of Histogram Equalized Image
0
500
1000
1500
2000
2500
3000
0
50
100
150
200
250

Histogram Equalization using Built-in Command

Original Test Image

Histogram Equalized Image

Original Test Image

EXPERIMENT 6:

AIM:- To write a program to perform Histogram Specification on

an image

SOFTWARE:- Matlab

CODE:-

```
clc; clear all; close all;

image = [1 3 5; 4 4 3; 5 2 2];
level = [0 1 2 3 4 5 6 7]; pixels
= zeros(1,9);

for i = 1:7     for j = 1:9         if
image(j) == level(i)
pixels(i) = pixels(i) + 1;          end
end end pixels = pixels(1:8);

cdf = zeros(1,8); cdf(1) = pixels(1);
for i = 2:8     cdf(i)
= cdf(i-1) + pixels(i); end

input_equ = round(cdf*7./9);

target = [0 0 0 0 2 2 4 1];

cdf_t = zeros(1,8); cdf_t(1) =
target(1); for i = 2:8
cdf_t(i) = cdf_t(i-1) +
target(i); end

target_equ = round(cdf_t*7./9);

map = zeros(1,8);
```

```
j = 1; for i = 1:8     for j = 1:8         if input_equ(i)
<= target_equ(j)
map(i) = level(j);             break;
end  end
end
```

OUTPUT:

map=[0,4,4,6,6,7,7,7]

EXPERIMENT 7:

AIM:- To write a program to perform Histogram Specification on

an image

SOFTWARE:- python

CODE:-

```
filt= [[2,2,3,2],[3,5,1,1],[2,9,4,2],[0,1,0,2]] img =
[[256,256,256,256,256,256],[256,12,14,23,13,256],[256,16,11,21
,18,256],[256,21,24,23,12,256],[256,12,21,20,10,256],[256,256,2
56,256,256,256]] img1 =
[[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]] for i in range(1,5):
for j in range(1,5):
        m, n = i-1,j-1
        l1 = [img[i-1][j-1], img[i-1][j], img[i-1][j+1], img[i][j-1],
img[i][j], img[i][j+1], img[i+1][j-1], img[i+1][j], img[i+1][j+1]]         min1 = l1.index(min(l1))
n = n + min1%3          m = m + min1//3
        img1[i-1][j-1]=filt[m-1][n-1]

print(img1)
```

OUTPUT:

img1=[[7, 7, 7, 2], [7, 7, 7, 2], [7, 7, 2, 2], [0, 0, 2, 2]]

EXPERIMENT 8:

AIM:- To write a program to perform Watermarking on an image

SOFTWARE:- Matlab

CODE:-

```
%%
clc; clear;
close all;

% Reading Test Image
test_image = imread('Minion.png');

% Resizing Test Image to 256x256x3
test_image = imresize(test_image,[256 256]);

% Converting test image to grayscale
gray_test_image = rgb2gray(test_image);
figure('Name','Result') subplot(131);
imshow(gray_test_image); title('Original
Image');

%% Plotting MSB Bit Plane Image

% sign image sign = imread('lenna.png');
sign_resize = imresize(sign,[256 256]);
gray_sign = rgb2gray(sign_resize);

% Getting MSB bit of each pixel of sign msb_bit_sign
= bitget(gray_sign, 8);

% Plotting image of sign subplot(132);
```

```
imshow(logical(msb_bit_sign));
title('Sign');

%% Image of 7 bits except LSB of Original Image

[rows cols] = size(gray_test_image); watermark_image
= zeros(rows,cols);

for ii = 1:rows
for jj = 1:cols
        % Getting seven bits except lsb        temp =
bitget(gray_test_image(ii,jj), 2:1:8);

        pixel_value_bin = zeros(1,8);

        % MSB bit as the pixel value of Sign
pixel_value_bin(8) = msb_bit_sign(ii,jj);

        % Next 7 bits as the pixel values of Image whose 7 bits are
taken
        for ll = 2:8
pixel_value_bin(ll1) = temp(ll-1);        end

        % Converting final pixel value to decimal value
pixel_value_dec = bi2de(pixel_value_bin);
        % Storing these values
        watermark_image(ii,jj) = pixel_value_dec;
end end

% Plotting Watermarked Image subplot(133);
imshow(uint8(watermark_image));
title('Watermarked Image');
```

OUTPUT:

EXPERIMENT 9:

AIM:- To write a program to perform Canny Edge Detection on

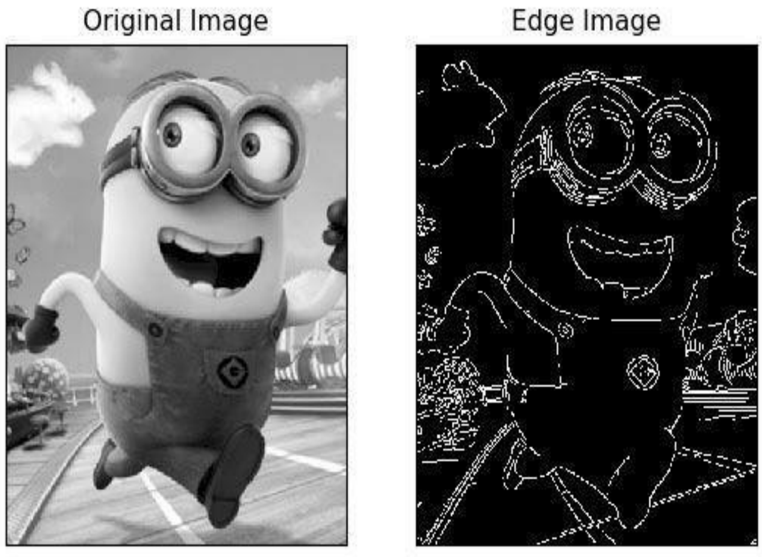
an image

SOFTWARE:- python

CODE:- import cv2

```
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('Minion.png',0) edges
= cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray') plt.title('Edge
Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

OUTPUT:



EXPERIMENT 10:

AIM:- To write a program to perform Sobel Edge Detection on an image

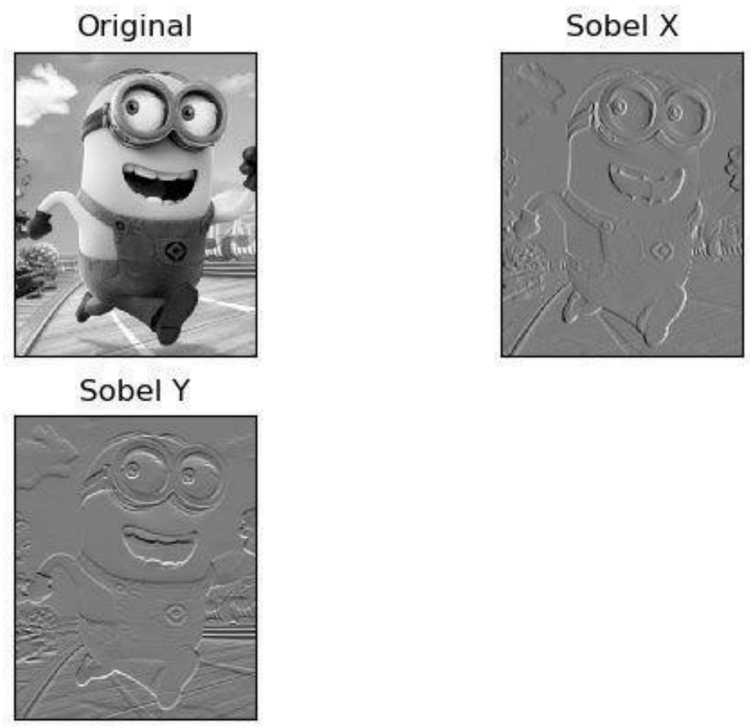
SOFTWARE:- python

CODE:- import cv2 import numpy as np from matplotlib import pyplot as plt

img = cv2.imread('Minion.png',0)

```
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)
plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,2),plt.imshow(sobelx,cmap = 'gray') plt.title('Sobel
X'), plt.xticks([]), plt.yticks([])

plt.subplot(2,2,3),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([]) plt.show()
```

OUTPUT:



EXPERIMENT 11:

AIM:- To write a program to perform high pass and low pass

filters on an image

SOFTWARE:- matlab

CODE:-

```
%% Part 1 : Import the Image

I_Leena = imread('Minion.png'); I_Leena =
rgb2gray(I_Leena); figure(1); title('Original
Image'); imshow(I_Leena);

%% Part 2 : Design the Filters

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

LPF = [1,1,1;1,1,1;1,1,1] .* (1/9);


%% Part 3 : Apply the Filters

I_Leena_HPF = conv2(I_Leena,HPF);
I_Leena_LPF = conv2(I_Leena,LPF); figure(2);
subplot(121); imshow(uint8(I_Leena_HPF)); title("Leena - HPF
(Observe the Edges)");  subplot(122);
imshow(uint8(I_Leena_LPF)); title("Leena - LPF
(Observe the Blur Effect)");
```

OUTPUT:

Leena - HPF (Observe the Edges)

Leena - LPF (Observe the Blur Effect)

EXPERIMENT 12:

AIM:- To write a program to perform JPEG Algorithm usind DCT on an image

SOFTWARE:- matlab

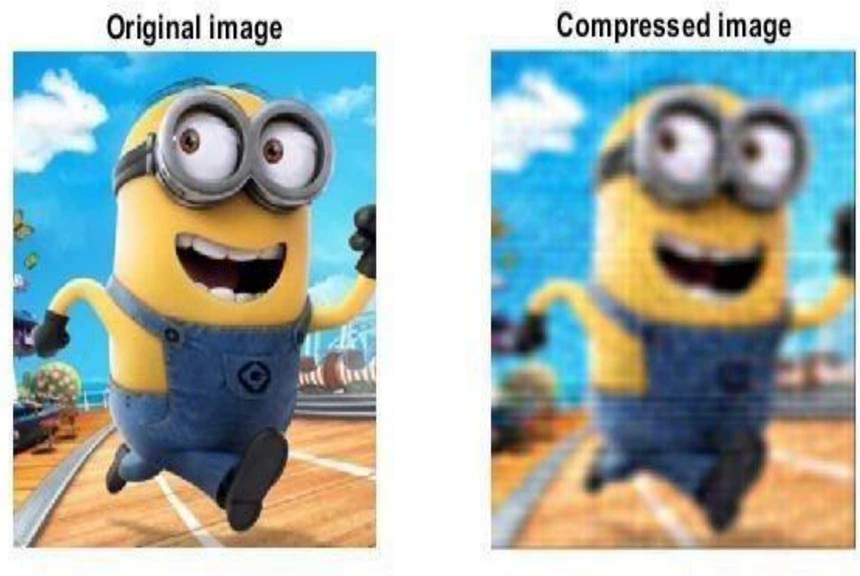
CODE:-

```
% Implementing JPEG ALgorithm using DCT coefficients.
clc; clear all; close all;


%% Reading an input image
image=double((imread('Minion.png'))); %% Computing the size of
an image [m1,n1,dim]=size(image); z=min(m1,n1);

%% Threshold Value prompt = 'Enter the
threshold value?  '; thresh =
input(prompt);

%% Resize the image to make it square
image_square=(imresize(image,[z z]));

%% Compute the size of a square image
[m2,n2]=size(image_square);

%% Calculation the DCT basis matrix for i=1:m2     for j=1:m2
if(i==1)          z(i,j)=sqrt(1/n2)*(cos((((2*j-
1)*(i-1)*pi))/(2*n2)));        else           z(i,j)=sqrt(2/n2)*(cos((((2*j-1)*(i-
1)*pi))/(2*n2)));
end     end end
```

```
%% Calculate the DCT coefficents for each RGB components of
an image
DCT_red=z*image_square(:,:,1)*z';
DCT_green=z*image_square(:,:,2)*z';
DCT_blue=z*image_square(:,:,3)*z';


%% Truncating the DCT coefficients to achieve compression for
each channel
DCT_red(abs(DCT_red)<thresh)=0;
DCT_green(abs(DCT_green)<thresh)=0;
DCT_blue(abs(DCT_blue)<thresh)=0;

DCT(:,:,1)=DCT_red;
DCT(:,:,2)=DCT_green;
DCT(:,:,3)=DCT_blue;
%% Reconstruction of the compresed image from each channel
image_compressed(:,:,1)=z'*DCT_red*z;
image_compressed(:,:,2)=z'*DCT_green*z;
image_compressed(:,:,3)=z'*DCT_blue*z;

imwrite(uint8(image_compressed),'Compressed_image_coloure
d. jpeg');

%% Plotting the results figure(); subplot(121);
imshow(uint8(image_square)),title('Original image'); subplot(122);
imshow(uint8(image_compressed)),title('Compressed image');
```

OUTPUT:

Setting input threshold to 80 gives output image as follows



EXPERIMENT 13:

AIM:- To write a program to perform Run Length Encoding and performing RunLength Encoding for 100 bits and also to output whether it is positive or Negative Encoding

SOFTWARE:- Python

CODE:-

```
# Importing Required Libraries
import random  import numpy as
np import math

# Function to get binary value of a decimal value
def decimal_to_binary(digit):     # If number is
zero return 0000     if (digit == 0):        return
'0000'

   # If number is from 1 to 15 return binary value in 4 bits
elif ((digit > 0) and (digit < 16)):        bin_value = ''
for i in range(4):
          # To get binary value divide the number by 2 each time
          # Storing remainder in string
bin_value = bin_value + str(digit % 2)
digit = digit // 2
      # To get binary value reverse the string as we take
remainder in reverse order
      return bin_value[::-1]

   # If number is greater than 15 return binary value in required
number of bits     else:
      # Calculating number of bits required to represent given
number in binary
      no_of_bits = math.floor(math.log2(digit)) + 1
```

bin_value = ''   for i in range(no_of_bits):
            # To get binary value divide the number by 2 each time
            # Storing remainder in string
bin_value = bin_value + str(digit % 2)
digit = digit // 2
        # To get binary value reverse the string as we take
remainder in reverse order
        return bin_value[::-1]

Function to get Run Length ENcoding of given bit
stream def run_length_encoding(bit_stream):   #
Convert Input Bit Stream in an array   bit_stream =
np.asarray(bit_stream)   print('Input Bit Stream : ')   #
Print array of input bit stream   print(bit_stream)

    # List to store the bit followed by its count in decimal
decimal_encoding = []

    # List to store the bit followed by its count in binary
binary_encoding = []

    # Previous bit
prev_bit = ''

    # Initial Count = 0
count = 0

    # Getting count in decimal as well as in binary
for j in range(len(bit_stream)):

        # If present bit is equal to previous bit increment the count
if (bit_stream[j] == prev_bit):
            count = count + 1

        # If present bit is not equal to previous bit then store the
count value
        # in decimal in decimal_encoding list and in binary in
binary_encoding list

```
        # Also, make count equal to 1 as we got new bit i.e.
different from previous bit         else:
            # Combining bit and its count(decimal)
            decimal_encoded_value = str(prev_bit) + str(count)

            # Combining bit and its count(binary)
binary_encoded_value = str(prev_bit) +
str(decimal_to_binary(count))

            # Store these values in particular list
decimal_encoding.append(decimal_encoded_value)
binary_encoding.append(binary_encoded_value)

            # Make count equal to 1 as we got new bit i.e. different
from previous bit             count = 1

        # Now previous bit = present bit for next bit
prev_bit = bit_stream[j]


    # Now store the last bit and its count in particular list
decimal_encoding.append(str(prev_bit) + str(count))     # Print
decimal_encoding list which contain bit followed by its count in
decimal
    print('\nEach Bit and its count in decimal : ')
print(decimal_encoding[1:])

    # Print binary_encoding list which contain bit followed by its
count in binary
    binary_encoding.append(str(prev_bit) +
str(decimal_to_binary(count)))
final_encoding = binary_encoding[1:]
print('\nEach Bit and its count in binary : ')
print(final_encoding)

    # Converting this list binary_encoding into an array
s = ''     for i in final_encoding:         s = s + i
```

```
    l =
[]
for i in s:
      l.append(int(i))

   # Printing an array of final encoded output in binary
rle_output = np.asarray(l)     print('\nFinal Encoded
Output : ')     print(rle_output)


   # Computing if it is negative compression or positive
compression
   # If length of Run Length Encoded output is less than input bit
stream then
   # it is positive compression otherwise it is negative
compression     if (len(rle_output) < len(bit_stream)):        #
Printing length of input bit stream array        print('\nLength of
Input Bit Stream(N1) : ', len(bit_stream))

      # Printing Length of Run Length Encoded output array
print('Length of Final Encoded Output(n2) : ', len(rle_output))
print(len(rle_output), '(N2) < (N1)', len(bit_stream))

      # Stating that it is positive Compression
print('Hence, it is a Positive Compression')

      # Printing Compression Ratio
print('Compression Ratio (N1/N2) =
',len(bit_stream)/len(rle_output))


   elif (len(rle_output) > len(bit_stream)):        #
Printing length of input bit stream array
print('\nLength of Input Bit Stream(N1) : ',
len(bit_stream))

      # Printing Length of Run Length Encoded output array
print('Length of Final Encoded Output(N2) : ', len(rle_output))
```

print(len(rle_output), '(N2) > (N1)', len(bit_stream))

..............................…part2...

....

# Stating that it is positive Compression
print('Hence, it is a Negative Compression')
# Enter input bit stream in an Array
# For Example bit_stream = [1,0,0,0,1,1,1]
# Then, call run_length_encoding(bit_stream) to get output

List for storing input bits bit_stream
= []

Generating 100 bits randomly for
i in range(100):
random_bit = random.randint(0,1)
bit_stream.append(random_bit)

Call run_length_encoding(bit_stream) to compute Run Length Encoding of input bit stream run_length_encoding(bit_stream)

OUTPUT:

Input Bit Stream :
[0 1 0 1 1 1 0 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0 1 1 1 1 1 0 0 0 1
0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 1 0 1 1
0 1 1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 1 0 0 1 1 1 1]

Each Bit and its count in decimal :
['01', '11', '01', '13', '01', '12', '03', '11', '03', '11', '02', '12', '01',
'11', '02', '11', '02', '15', '03', '11', '05', '11', '02', '11', '03', '11', '04',
'11', '01', '15', '01', '13', '02', '11', '02', '11', '01', '12', '01', '13', '01', '12',
'01', '13', '07', '12', '02', '14']

Each Bit and its count in binary :

['00001', '10001', '00001', '10011', '00001', '10010', '00011', '10001', '00011', '10001', '00010', '10010', '00001', '10001', '00010', '10001', '00010', '10101', '00011', '10001', '00101', '10001', '00010', '10001', '00011', '10001', '00100', '10001', '00001', '10101', '00001', '10011', '00010', '10001', '00010', '10001', '00001', '10010', '00001', '10011', '00001', '10010', '00001', '10011', '00111', '10010', '00010', '10100']

Final Encoded Output :
[0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0 0 0 1 1 1
0
 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 1 0 0
0 1
 0 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 0 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 0 1
0
 0 0 1 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 1 1 0
1
0 1 0 0 0 0 1 1 0 0 1 1 0 0 0 1 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0
0 0 1
1 0 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1
1 0 0
 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 0 1 0 0]

..............................…part2..
……

Length of Input Bit Stream(N1) : 100
Length of Final Encoded Output(N2) : 240
240 (N2) > (N1) 100
Hence, it is a Negative Compression
>>>

Github

link: https://github.com/Anirudh-Senani/DIP