Indian Institute of Information Technology, Nagpur

Digital Image Processing (ECL-415)

Final Report

Submitted By :
Sarthak Babra (BT19ECE028)

Semester 6
Electronics and Communication Engineering Dept.

Submitted To :
Dr. Tapan Jain
Course Instructor

INDEX

Assignment - 1

Converting colour image to rgb gray layers, gray scale image and binary image. Also, performing pixel addition on gray scale image.

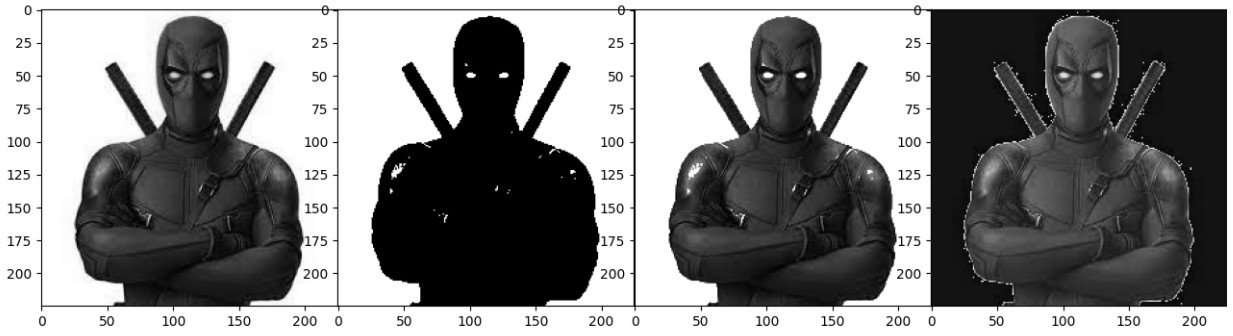
Code

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def rgb_to_gray(img):
        grayImage = np.zeros(img.shape)
        R = np.array(img[:, :, 0])
        G = np.array(img[:, :, 1])
        B = np.array(img[:, :, 2])

        R = (R *.299)
        G = (G *.587)
        B = (B *.114)

        Avg = (R+G+B)
        grayImage = img.copy()

        for i in range(3):
           grayImage[:,:,i] = Avg

        return grayImage

image = mpimg.imread("yoyoy.jpg")
grayImage = rgb_to_gray(image)

img = np.array(grayImage)
binarr = np.where(img>128, 255, 0)

img3 = np. add(grayImage, binarr)

img4 = np.add(20,grayImage)


#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(1,4)
axarr[0].imshow(grayImage)
axarr[1].imshow(binarr, cmap='gray')
axarr[2].imshow(img3, cmap='gray')
axarr[3].imshow(img4 , cmap='gray')
```

```
40 plt.show()
```

input

ouput



Assignment - 2

Performing subtraction operation on gray scale image.

Code

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

#read images
img1 = mpimg.imread('input.jpg')
img2 = mpimg.imread('input.jpg')

for i in range(0,400):
      for j in range(0,1200):
            for k in range(0,3):
              img2[i+400,j+200,k] = 0

#obtain the final image by subtracting the original and edited ...
    image
img3= img1 - img2
#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(1,3)
axarr[0].imshow(img1)
axarr[1].imshow(img2)
axarr[2].imshow(img3)
plt.show()
```

input

ouput

Assignment - 3

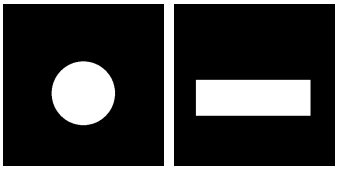
Performing logical operations - OR, AND, NOR, NAND, XOR AND XNOR.

Code

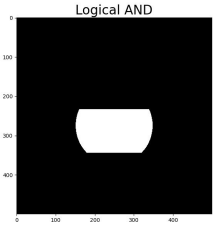
```
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import operator

def image_to_binary(img):
 #code below is to convert image to binary

 R = img[:,:,0]
 G = img[:,:,1]
 B = img[:,:,2]
 binary = (R+G+B)/3

 for i in range(500):
    for j in range(500):
        if binary[i,j]>10:
                binary[i,j]=1
        else:
                binary[i,j]=0

 return binary


#read images
img_circle = mpimg.imread('circle.jpg')
img_rectangle = mpimg.imread('rectangle.jpg')

# now calling function to convert image in binary
binary_circle = image_to_binary(img_circle)
binary_rectangle = image_to_binary(img_rectangle)

#showing circle
plt.imshow(binary_circle, cmap='gray')
plt.title('Circle',fontsize=24)
plt.show()
#showing rectangle
plt.imshow(binary_rectangle, cmap='gray')
plt.title('Rectangle',fontsize=24)
plt.show()
#logical and
```

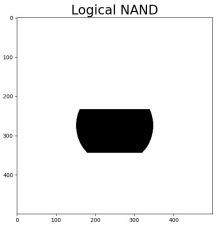
```
41 img_and = binary_circle.astype(int) & binary_rectangle.astype(int...
       )
42 plt.imshow(img_and, cmap='gray')
43 plt.title('Logical AND',fontsize=24)
44 plt.show()
45
46 #logical or
47 img_or = binary_circle.astype(int) | binary_rectangle.astype(int)
48 plt.imshow(img_or, cmap='gray')
49 plt.title('Logical OR',fontsize=24)
50 plt.show()
51
52 #logical xor
53 img_xor = binary_circle.astype(int) ^ binary_rectangle.astype(int...
       )
54 plt.imshow(img_xor, cmap='gray')
55 plt.title('Logical XOR',fontsize=24)
56 plt.show()
57
58 #logical nand
59 img_nand = ¬img_and
60 plt.imshow(img_nand, cmap='gray')
61 plt.title('Logical NAND',fontsize=24)
62 plt.show()
63
64 #logical nor
65 img_nor = ¬img_or
66 plt.imshow(img_nor, cmap='gray')
67 plt.title('Logical NOR',fontsize=24)
68 plt.show()
69
70 #logical xnor
71 img_xnor = ¬img_xor
72 plt.imshow(img_xnor, cmap='gray')
73 plt.title('Logical XNOR',fontsize=24)
74 plt.show()
```

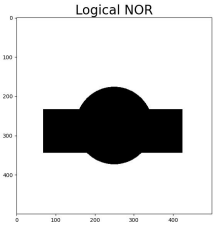
input

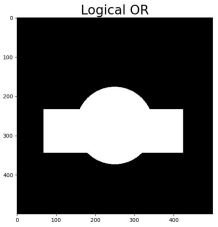


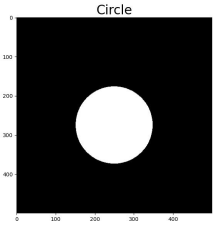
ouput

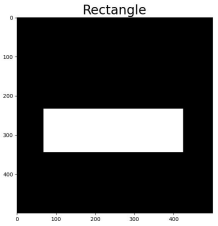

Logical AND

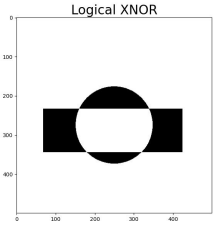

Logical NAND

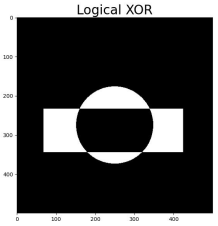

Logical NOR


Logical OR


Circle


Rectangle


Logical XNOR


Logical XOR

Assignment - 4

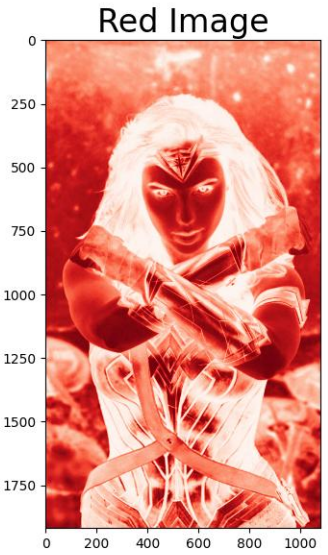
4 Extracting r,g,b colour layers from a colour image.

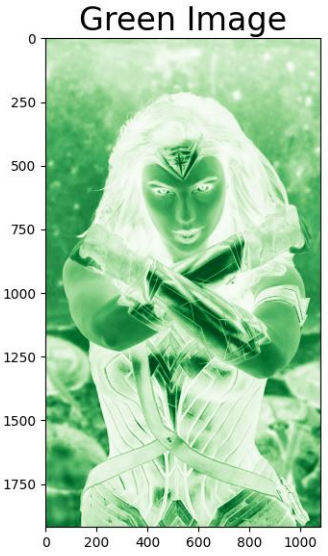
Code

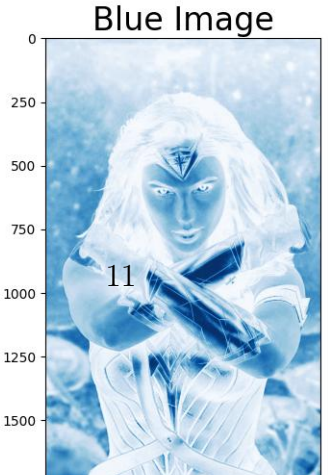
```
from matplotlib import pyplot as plt
import matplotlib.image as mpimg

#read image
img = mpimg.imread('input.jpg')

#read the red, green and blue colourspaces differently
R = img[:,:,0]
G = img[:,:,1]
B = img[:,:,2]

#display the red image
plt.imshow(R, cmap='Reds')
plt.title('Red Image',fontsize=24)
plt.show()

#display the green image
plt.imshow(G, cmap='Greens')
plt.title('Green Image',fontsize=24)
plt.show()

#display the blue image
plt.imshow(B, cmap='Blues')
plt.title('Blue Image',fontsize=24)
plt.show()
```

input

ouput


Red Image
0
250
500
750
1000
1250
1500
1750
0
200
400
600
800
1000


Green Image
0
250
500
750
1000
1250
1500
1750
0
200
400
600
800
1000


Blue Image
0
250
500
750
1000
1250
1500

Assignment - 5

Histogram generation and equalisation.

Code

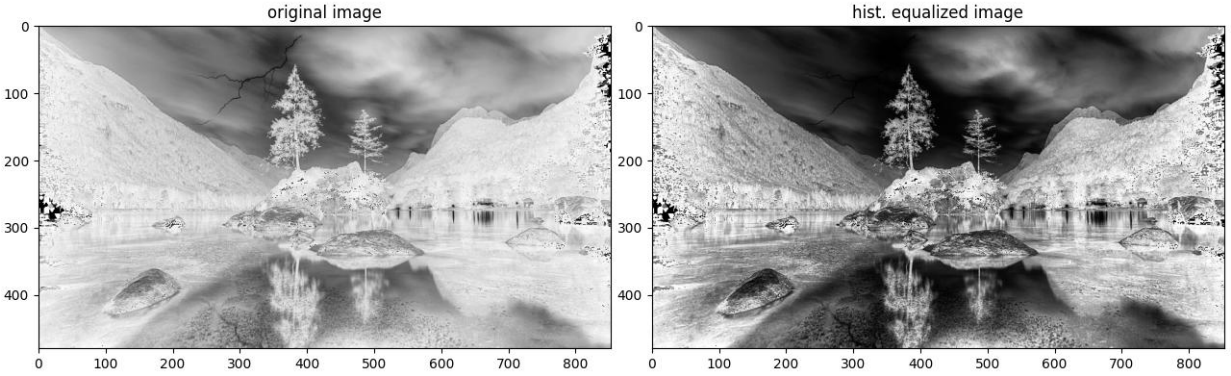
```python
import numpy as np

def imhist(im):
  # calculates normalized histogram of an image
  m, n = im.shape
  h = [0.0] * 256
  for i in range(m):
    for j in range(n):
      h[im[i, j]]+=1
  return np.array(h)/(m*n)

def cumsum(h):
  # finds cumulative sum of a numpy array, list
  return [sum(h[:i+1]) for i in range(len(h))]

def histeq(im):
  #calculate Histogram
  h = imhist(im)
  cdf = np.array(cumsum(h)) #cumulative distribution function
  sk = np.uint8(255 * cdf) #finding transfer function values
  s1, s2 = im.shape
  Y = np.zeros_like(im)
  # applying transfered values for each pixels
  for i in range(0, s1):
    for j in range(0, s2):
      Y[i, j] = sk[im[i, j]]
  H = imhist(Y)
  #return transformed image, original and new istogram,
  # and transform function
  return Y , h, H, sk
```

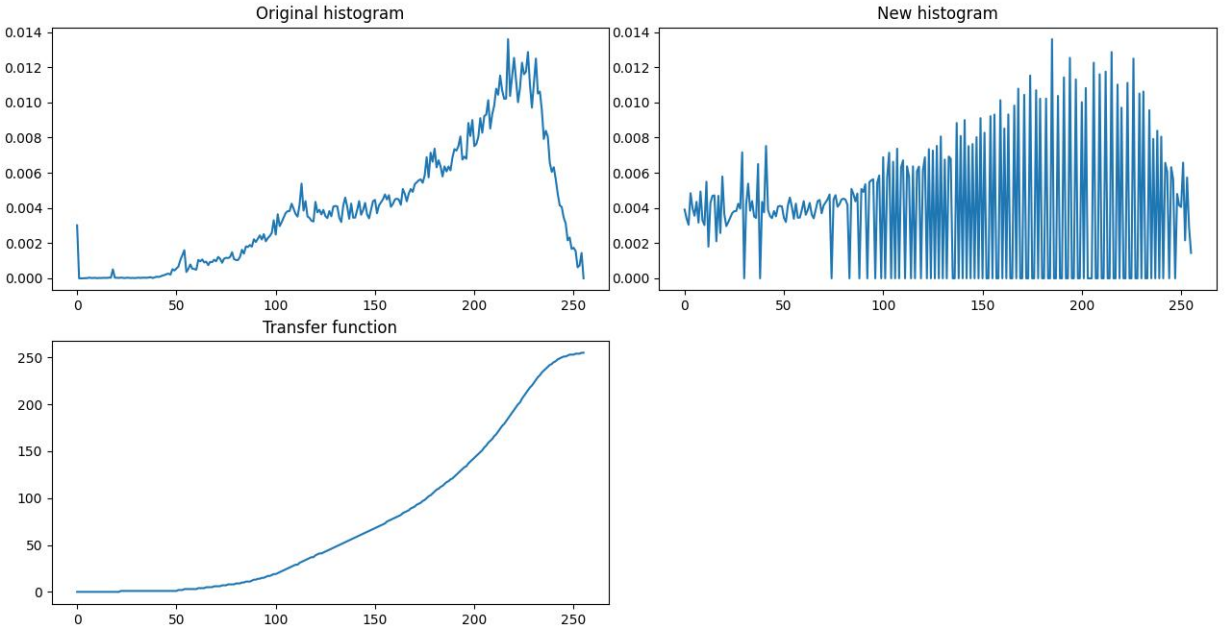
```python
import pylab as plt
import matplotlib.image as mpimg
import numpy as np
# load image to numpy arrayb
# matplotlib 1.3.1 only supports png images
# use scipy or PIL for other formats
img = np.uint8(mpimg.imread('input.jpg')*255.0)
# convert to grayscale
```

```
# do for individual channels R, G, B, A for nongrayscale images

img = np.uint8((0.2126* img[:,:,0]) + \
      np.uint8(0.7152 * img[:,:,1]) +\
       np.uint8(0.0722 * img[:,:,2]))

# use hist module from hist.py to perform histogram equalization
from hist import histeq
new_img, h, new_h, sk = histeq(img)

# show old and new image
# show original image
plt.subplot(121)
plt.imshow(img)
plt.title('original image')
plt.set_cmap('gray')
# show original image
plt.subplot(122)
plt.imshow(new_img)
plt.title('hist. equalized image')
plt.set_cmap('gray')
plt.show()

# plot histograms and transfer function
fig = plt.figure()
fig.add_subplot(221)
plt.plot(h)
plt.title('Original histogram') # original histogram

fig.add_subplot(222)
plt.plot(new_h)
plt.title('New histogram') #hist of eqlauized image

fig.add_subplot(223)
plt.plot(sk)
plt.title('Transfer function') #transfer function

plt.show()
```

input

ouput


original image
hist. equalized image


Original histogram
New histogram
Transfer function

Assignment - 6

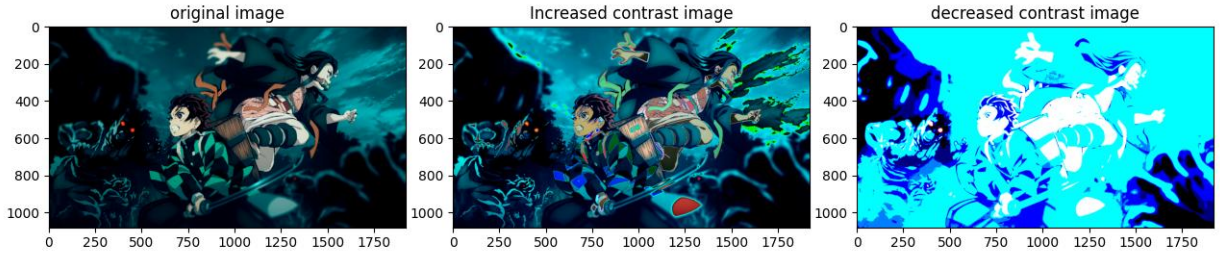
Constrast manipulation of colour and gray scale image.

Code

```
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
import numpy as np

#read image
img1 = mpimg.imread('input.jpg')

#Increased contrast
img2 = img1*2

#decreased contrast
img3 = img1/2

#subplot(r,c) provide the no. of rows and columns
f, axarr = plt.subplots(1,3)
axarr[0].imshow(img1)
axarr[0].set_title('original image')
axarr[1].imshow(img2)
axarr[1].set_title('Increased contrast image')
axarr[2].imshow(img3)
axarr[2].set_title('decreased contrast image')
plt.show()
```

input

ouput


original image
Increased contrast image
decreased contrast image
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

Assignment - 7

7 Bit Plane Slicing.

Code

```
clear all; close all; clc;

% Reading the image
Img = imread('Image.jpg');
% Converting the image into gray scale
gray = rgb2gray(Img);
% Extracting the size of the image
[m,n] = size(gray);
% Converting the unsigned integer to a double type
gray1 = double(gray);
% Decimal to binary converter
bin = de2bi(gray1);

% Extracting LSB plane
plane1 = bin(:,1);
p1 = reshape(plane1,m,n);
% Extracting 2 bit(from the left) plane
plane2 = bin(:,2);
p2 = reshape(plane2,m,n);
% Extracting 3 bit(from the left) plane
plane3 = bin(:,3);
p3 = reshape(plane3,m,n);
% Extracting 4 bit(from the left) plane
plane4 = bin(:,4);
p4 = reshape(plane4,m,n);
% Extracting 5 bit(from the left) plane
plane5 = bin(:,5);
p5 = reshape(plane5,m,n);
% Extracting 6 bit(from the left) plane
plane6 = bin(:,6);
p6 = reshape(plane6,m,n);
% Extracting 7 bit(from the left) plane
plane7 = bin(:,7);
p7 = reshape(plane7,m,n);
% Extracting MSB plane
plane8 = bin(:,8);
p8 = reshape(plane8,m,n);

% Plotting all the bit plane results
subplot(2,5,1);
imshow(Img);title('Original Image');
```

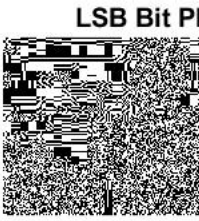
```
subplot(2,5,2);
imshow(gray);title('Gray scale Image');
subplot(2,5,3);
imshow(p1);title('LSB Bit Plane');
subplot(2,5,4);
imshow(p2);title('2nd Bit Plane');
subplot(2,5,5);
imshow(p3);title('3rd Bit Plane');
subplot(2,5,6);
imshow(p4);title('4th Bit Plane');
subplot(2,5,7);
imshow(p5);title('5th Bit Plane');
subplot(2,5,8);
imshow(p6);title('6th Bit Plane');
subplot(2,5,9);
imshow(p7);title('7th Bit Plane');
subplot(2,5,10);
imshow(p8);title('MSB Bit Plane');
```

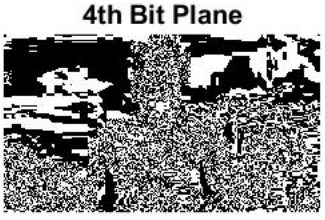
input

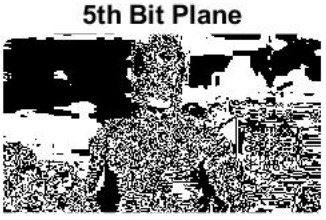
ouput


Original Image


Gray scale Image


LSB Bit Pl


4th Bit Plane


5th Bit Plane


6th Bit Pl

Assignment - 8

Shannon-Fano Coding.

Code

```
% Shannon–Fano Coding

clc;
clear all;
close all;

disp('Enter the probabilities:');

ss=[0.2 0.2 0.12 0.08 0.08 0.08 0.04];

%outputs = string of codewords,average codeword length
ss=ss./sum(ss); %if occurrences are inputted, probabilities are ...
    gained
ss=sort(ss,'descend');  %the probabilities are sorted in ...
    descending order

%siling=ceil(log2(1/ss(1))); %initial length is computed

siling=log2(1/ss(1)); %initial length is computed
siling=round(siling,1,'significant');

sf=0;
fano=0;
%initializations for Pk
n=1;Hx=0; %initializations for entropy H(X)

for i=1:length(ss)
   Hx=Hx+ ss(i)*log2(1/ss(i)); %solving for entropy
end

for k=1:length(ss)
   info(k)=-(log2(ss(k))); %Information
end

for j=1:length(ss)-1
   fano=fano+ss(j);
   sf=[sf 0]+[zeros(1,j) fano]; %solving for Information for ...
    every codeword
   siling=[siling 0]+[zeros(1,j) ceil(log2(1/ss(j+1)))]; %solving...
     for length every codeword

```

```
38 end
39
40 for r=1:length(sf)
41     esf=sf(r);
42     for p=1:siling(r)
43         esf=mod(esf,1)*2;
44         h(p)=esf-mod(esf,1); %converting Pk into a binary number
45     end
46     hh(r)=h(1)*10^(siling(r)-1); %initializtion for making the ...
       binary a whole number
47     for t=2:siling(r)
48         hh(r)=hh(r)+h(t)*10^(siling(r)-t);    %making the binary ...
       a whole number
49     end                                       %e.g. 0.1101 ==> ...
       1101
50 end
51
52 c={'0','1'};
53 disp('Codeword');
54 for i=1:length(hh)
55     u=1;                                      %converting the ...
       codes into a string
56    for t=siling(i):-1:1
57        f=floor(hh(i)/10^(t-1));              %1001 ==>1 (getting...
        the first highest unit of a number)
58        hh(i)=mod(hh(i),10^(t-1));            %1001 ==>001(...
       eliminating the first highest unit of a number)
59        if f==1
60            if u==1
61                d=c{2};                       %conversion part (...
       num(1001) to str(1001))
62            else
63                d=[d c{2}];
64            end
65        else
66            if u==1
67                d=c{1};
68            else
69                d=[d c{1}];
70            end
71        end
72
73        codex{i,:}={d};
74        u=u+1;
75    end
76    disp([d])
77 end
78
79 tao=siling(1)*ss(1); %initialization for codeword length
```

```
for u=1:length(ss)-1 %computing for codeword length
    tao=tao+siling(u+1)*ss(u+1);
end

T=tao/n; %computing for average codeword length
B=[flipud(rot90(ss)),flipud(rot90(siling)),flipud(rot90(info))];
disp(['Probability','    Length','    Information'])
disp(B)
disp(['Entropy H(X) = ',num2str(Hx),'bits/symbol'])
disp(['Average length,L = ',num2str(T),'bits/symbol'])
eff=((Hx/T)*100); %Coding efficiency
disp(['Efficiency=',num2str(eff),'%'])
redu=100-eff;    %Redundancy
disp(['Redundancy=',num2str(redu),'%'])
```

input

```
disp('Enter the probabilities:');

ss=[0.2 0.2 0.12 0.08 0.08 0.08 0.04];
```

ouput

```
Command Window
  Enter the probabilities:
  Codeword
  00
  01
  100
  1010
  1100
  1101
  11110
  Probability   Length   Information
      0.2500    2.0000    2.0000
      0.2500    2.0000    2.0000
      0.1500    3.0000    2.7370
      0.1000    4.0000    3.3219
      0.1000    4.0000    3.3219
      0.1000    4.0000    3.3219
      0.0500    5.0000    4.3219

  Entropy H(X) = 2.6232bits/symbol
  Average length,L = 2.9bits/symbol
  Efficiency=90.4559%
  Redundancy=9.5441%
fx >>
```

Assignment - 9

Huffman Coding.

Code

```
function [code,compression]=huffman(p)
%HUFFMAN
%HUFFMAN CODER FOR V5
% Format [CODE,COMPRESSION]=HUFFMAN(P)
%
% P is the probability (or number of occurences) of each alphabet...
     symbol
% CODE gives the huffman code in a string format of ones and ...
    zeros
% COMPRESSION gives the compression rate
p=p(:)/sum(p);    %normalises probabilities
c=huff5(p);
code=char(getcodes(c,length(p)));
compression=ceil(log(length(p))/log(2))/ (sum(code' ~= ' ')*p);
%-----------------------------------------------------------------
function c=huff5(p);
% HUFF5 Creates Huffman Tree
% Simulates a tree structure using a nested cell structure
% P is a vector with the probability (number of occurences)
%   of each alphabet symbol
% C is the Huffman tree.
c=cell(length(p),1);      % Generate cell structure
for i=1:length(p)       % fill cell structure with 1,2,3...n
   c{i}=i;             % (n=number of symbols in alphabet)
end
while size(c)-2         % Repeat till only two branches
  [p,i]=sort(p);          % Sort to ascending probabilities
  c=c(i);               % Reorder tree.
  c{2}={c{1},c{2}};c(1)=[]; % join branch 1 to 2 and prune 1
  p(2)=p(1)+p(2);p(1)=[];   % Merge Probabilities
end
%-----------------------------------------------------------------
function y= getcodes(a,n)
% Y=GETCODES(A,N)
% Pulls out Huffman Codes for V5
% a is the nested cell structure created by huffcode5
% n is the number of symbols
global y
y=cell(n,1);
getcodes2(a,[])
%-----------------------------------------------------------------
```

```
function getcodes2(a,dum)
% GETCODES(A,DUM)
%getcodes2
% called by getcodes
% uses Recursion to pull out codes
global y
if isa(a,'cell')
          getcodes2(a{1},[dum 0]);
          getcodes2(a{2},[dum 1]);
else
   y{a}=setstr(48+dum);
end
```

ouput

```
>> p = [0.4 0.3 0.2 0.1];
>> huffman(p)

ans =

  4×3 char array

    '1  '
    '01 '
    '001'
    '000'
```

Assignment - 10

1 Converting a colour image into a gray scale image and performing edge detectio

Code

```
clc;
clear all;
close all;

Image=(imread('Image.jpg'));

% Converting colour image to grayscale image
r = Image(:,:,1);
g = Image(:,:,2);
b = Image(:,:,3);
gray = double(uint8(double(r/3)+double(g/3)+double(b/3)));

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

In=gray;
mask1=[1, 0, -1;1, 0, -1;1, 0, -1];
mask2=[1, 1, 1;0, 0, 0;-1, -1, -1];
mask3=[0, -1, -1;1, 0, -1;1, 1, 0];
mask4=[1, 1, 0;1, 0, -1;0, -1, -1];

mask1=flipud(mask1);
mask1=fliplr(mask1);
mask2=flipud(mask2);
mask2=fliplr(mask2);
mask3=flipud(mask3);
mask3=fliplr(mask3);
mask4=flipud(mask4);
mask4=fliplr(mask4);

for i=2:size(gray, 1)-1
  for j=2:size(gray, 2)-1
    neighbour_matrix1=mask1.*In(i-1:i+1, j-1:j+1);
    avg_value1=sum(neighbour_matrix1(:));

    neighbour_matrix2=mask2.*In(i-1:i+1, j-1:j+1);
    avg_value2=sum(neighbour_matrix2(:));

    neighbour_matrix3=mask3.*In(i-1:i+1, j-1:j+1);
    avg_value3=sum(neighbour_matrix3(:));
```

```
42
43      neighbour_matrix4=mask4.*In(i-1:i+1, j-1:j+1);
44      avg_value4=sum(neighbour_matrix4(:));
45
46     %using max function for detection of final edges
47      I(i, j)=max([avg_value1, avg_value2, avg_value3, avg_value4])...
       ;
48
49    end
50 end
51
52 subplot(2,1,2);
53 imshow(uint8(I));
54 title('Edge Detected Image');
```

input

ouput