ASSIGNMENT 7

AIM 1:- Write a program to find entropy of an image. (Without using function)

Code:-

```
I=imread('lena.jpg');
if ~islogical(I)
  I = im2uint8(I);
end

% calculate histogram counts
p = imhist(I(:));

% remove zero entries in p
p(p==0) = [];

% normalize p so that sum(p) is one.
p = p ./ numel(I);

E = -sum(p.*log2(p));
```

Output:-

>> entropy
Entropy:

E =

7.5976

AIM 2:- Write a program for image compression using huffman coding. Display compression ratio, Relative data redundancy and error. Display total no. of bits for original image and compressed image.

Code:-

Huff_code.m

```
%clearing all variableas and screen
clear all;
close all;
clc;

%Reading image
a=imread('lena.jpg');
a= imresize(a,[256 256]);
figure,imshow(a)
imwrite(a,'original.jpg');
%converting an image to grayscale
%I=rgb2gray(a);
I=a;

%size of the image
[m,n]=size(I);
Totalcount=m*n;

%variables using to find the probability
cnt=1;
sigma=0;

%computing the cumulative probability.
for i=0:255
k=I==i;
count(cnt)=sum(k(:))

%pro array is having the probabilities
pro(cnt)=count(cnt)/Totalcount;
sigma=sigma+pro(cnt);
cumpro(cnt)=sigma;
cnt=cnt+1;
end;
```

```
%Symbols for an image
symbols = [0:255];

%Huffman code Dictionary
dict = huffmandict(symbols,pro);

%function which converts array to vector
vec_size = 1;
for p = 1:m
for q = 1:n
newvec(vec_size) = I(p,q);
vec_size = vec_size+1;
end
end

%Huffman Encodig
hcode = huffmanenco(newvec,dict);

%Huffman Decoding
dhsig1 = huffmandeco(hcode,dict);

%convertign dhsig1 double to dhsig uint8
dhsig = uint8(dhsig1);

%vector to array conversion
dec_row=sqrt(length(dhsig));
dec_col=dec_row;

%variables using to convert vector 2 array
arr_row = 1;
arr_col = 1;
vec_si = 1;

for x = 1:m
for y = 1:n
back(x,y)=dhsig(vec_si);
arr_col = arr_col+1;
vec_si = vec_si + 1;
end
arr_row = arr_row+1;
end

imwrite(back,'decoded.jpg');
% %converting image from grayscale to rgb
% [deco, map] = gray2ind(back,256);
% RGB = ind2rgb(deco,map);
% imwrite(RGB,'decoded.JPG');

%end of the huffman coding
```

Hufftree.m

```
% hufftree.m
%
% given alphabet and probabilities: create huffman-tree


function [tree, table] = hufftree(alphabet,prob)

for l=1:length(alphabet)       % create a vector of nodes (leaves),
one for each letter
   leaves(l).val = alphabet{l};
   leaves(l).zero= '';
   leaves(l).one='';
   leaves(l).prob = prob(l);
end

% combine the two nodes with lowest probability to a new node with
the summed prob.
% repeat until only one node is left
while length(leaves)>1
   [dummy,I]=sort(prob);
   prob = [prob(I(1))+prob(I(2)) prob(I(3:end))];
   node.zero = leaves(I(1));
   node.one  = leaves(I(2));
   node.prob = prob(1);
   node.val = '';
   leaves = [node leaves(I(3:end))];
end


% pass through the tree,
% remove unnecessary information
% and create table recursively (depth first)
table.val={}; table.code={};
[tree, table] = descend(leaves(1),table,'');




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
function [tree, table] = descend(oldtree, oldtable, code)

table = oldtable;
if(~isempty(oldtree.val))
   tree.val = oldtree.val;
   table.val{end+1}  = oldtree.val;
```

```
   table.code{end+1} = code;
else
   [tree0, table] = descend(oldtree.zero, table, strcat(code,'0'));
   [tree1, table] = descend(oldtree.one,  table, strcat(code,'1'));
   tree.zero=tree0;
   tree.one= tree1;
end
```

huffencode.m

```
% huffencode.m
%
% takes a cell-vector and a huffman-table
% returns a huffman encoded bit-string

function bitstring = huffencode(input, table)

bitstring = '';
for l=1:length(input),
   bitstring =
strcat(bitstring,table.code{strcmp(table.val,input{l})}); %
omits letters that are not in alphabet
end;
```

huffdecode.m

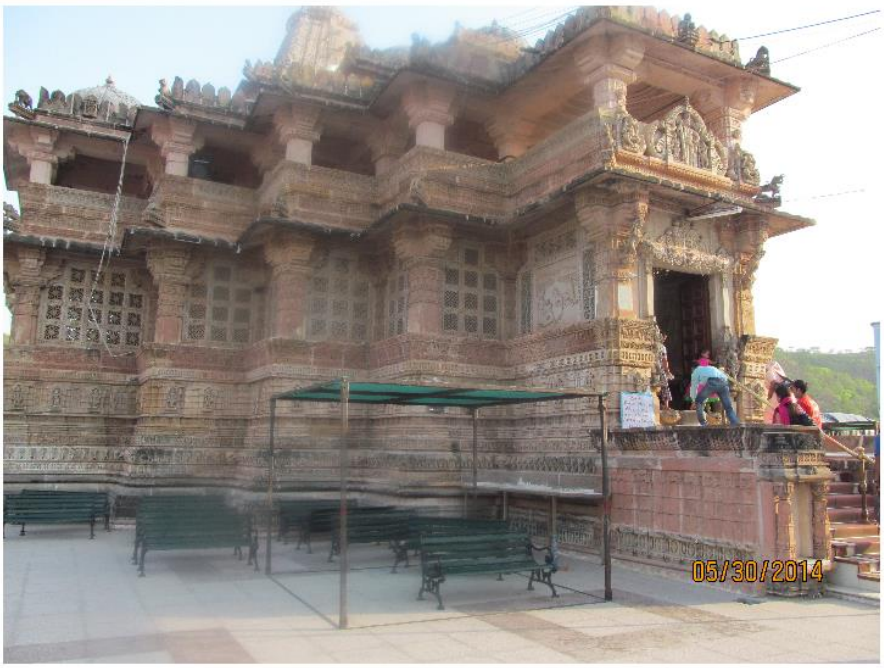
```
% huffdecode.m
%
% takes a bit-string and a huffman-tree
% returns a decoded cell array

function message = huffdecode(bitstring, tree)

treepos = tree;
counter = 1;
for l=1:length(bitstring)
   if(bitstring(l) == '1')
      treepos = treepos.one;
   else
      treepos = treepos.zero;
   end
   if(isfield(treepos,'val'))
      message{counter} = treepos.val;
      counter = counter+1;
      treepos = tree;
   end
end
```

Output:-

Original image



DCT Compressed Image

AIM 3:-Write a program for image compression using DCT. Display compression ratio, Relative data redundancy and error. Display compression ratio, Relative data redundancy and error.

Code:-

```
clc
clear all
close all
I = imread('cameraman.tif');
I = im2double(I);
T = dctmtx(8);
B = blkproc(I,[8 8],'P1*x*P2',T,T');
mask = [1   1   1   1   0   0   0   0
        1   1   1   0   0   0   0   0
        1   1   0   0   0   0   0   0
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);
imshow(I), figure, imshow(I2)
imwrite(I2,'cmp.tif')

fileName = 'cameraman.tif';
rm1=rms(fileName);
fileInfo = imfinfo(fileName);
sz1 = fileInfo.FileSize;


fileName = 'cmp.tif';
rm2=rms(fileName);
fileInfo = imfinfo(fileName);
sz2 = fileInfo.FileSize;
display('Compression Ratio is:');
display(sz2/sz1);

display('ERROR IS');
er=(rm1*rm1)-(rm2*rm2);
er=sqrt(er);
display(er);
display('Relative Data Redundency');


I=imread('cameraman.tif');
```

```
if ~islogical(I)
  I = im2uint8(I);
end

% calculate histogram counts
p = imhist(I(:));

% remove zero entries in p
p(p==0) = [];

% normalize p so that sum(p) is one.
p = p ./ numel(I);

E1 = -sum(p.*log2(p));


I=imread('cmp.tif');
if ~islogical(I)
  I = im2uint8(I);
end

% calculate histogram counts
p = imhist(I(:));

% remove zero entries in p
p(p==0) = [];

% normalize p so that sum(p) is one.
p = p ./ numel(I);

E2 = -sum(p.*log2(p));

display('Relative Data Redundancy');
ex=E2-E1;
display(ex);
```

Output:-

Compression Ratio is:

ans =

0.9068

ERROR IS

er =

11.7973

Relative Data Redundency
Relative Data Redundancy

ex =

0.0971

Original Image:-

Output compressed image:-

AIM 4:-Write a program for any one application of image compression in your group.

Code:-

```
clc;
clear all;
close all;
datain=input('enter the string in single quote with symbol $ as End
of string =');%input data
lda=length(datain);%length of datainput
dictionary=input('enter the dictionary in single quote(symbol used
in string are to be included)=');%input dictionary
ldi=length(dictionary);%length of dictionary
j=1;%used for generating code
n=0;%used for
%loop used for string array to cell array conversion
for i=1:ldi
dictnew(i)={dictionary(i)};
end

p=datain(1);%first symbol
s=p;%current symbol
k=1; %used for generating transmitting output code
i=1;%for loop
m=0;
while datain(i)~= '$'%end of symbol
c=datain(i+1);
if c~='$'
comb=strcat(s,c);%just for see combination
if strcmp(dictnew,strcat(s,c))==0
dictnew(j+ldi)={strcat(s,c)};
%lopp and check used for generating transmitting
%code array
check=ismember(dictnew,s);
for l=1:length(check)
if check(l)==1
tx_trans(k)=l;
k=k+1;
break;
end
end

s=c;
j=j+1;
i=i+1;
```

```
m=m+1;

else

s=strcat(s,c);
i=i+1;
end

else
%for sending last and eof tx_trans
check=ismember(dictnew,s);
for l=1:length(check)
if check(l)==1
tx_trans(k)=l;
k=k+1;
tx_trans(k)=0;
end
end
break;
end
end
display('new dictionary=')
display(dictnew);
display(tx_trans);

%decoding
dicgen=dictionary;
ldgen=length(dicgen);
ldtx=length(tx_trans);
index=length(dictionary);
string='';
%loop and below inst. used for cell array to char array
dicgen=cellstr(dictionary);
for i=1:ldi
dicgen(i)={dictionary(i)};
end
g=1;
entry=char(dictionary(tx_trans(1)));%first symbol
g=g+1;% next symbol
while tx_trans(g)~=0 %for EOF
s=entry;
entry=char(dicgen(tx_trans(g)));
string=strcat(string,s); %detected string
index=index+1; % next index
dicgen(index) = {strcat(s,entry(1))};%upgrade dictionary
g=g+1; % next index

end
string=strcat(string,entry)
```

```
disp(dicgen);
display('received original string=');
disp(string);
```

Output:-

```
enter the string in single quote with symbol $ as End of string ='abbcdbabdbbabbaccbd$'
enter the dictionary in single quote(symbol used in string are to be included)='abcd'
new dictionary=

dictnew =

  Columns 1 through 9

    'a'    'b'    'c'    'd'    'ab'    'bb'    'bc'    'cd'    'db'

  Columns 10 through 17

    'ba'    'abd'    'dbb'    'bab'    'bba'    'ac'    'cc'    'cb'

  Column 18

    'bd'


tx_trans =

  Columns 1 through 12

     1     2     2     3     4     2     5     9    10     6     1     3

  Columns 13 through 16

     3     2     4     0

string =

abbcdbabdbbabbaccbd

  Columns 1 through 9
    'a'    'b'    'c'    'd'    'ab'    'bb'    'bc'    'cd'    'db'

  Columns 10 through 17
    'ba'    'abd'    'dbb'    'bab'    'bba'    'ac'    'cc'    'cb'
  Column 18
    'bd'

received original string=
abbcdbabdbbabbaccbd
```