EE596-HUFFMAN CODING

FERNANDO P.D.R.

E/16/103

SEMESTER 7

08/04/2022

**LABORATORY ACTIVITY**

**Step 1- Download the images from the webpage (Instructor will provide the URL at the lab).**

**Step 2- Read the original image into a Matrix.).**

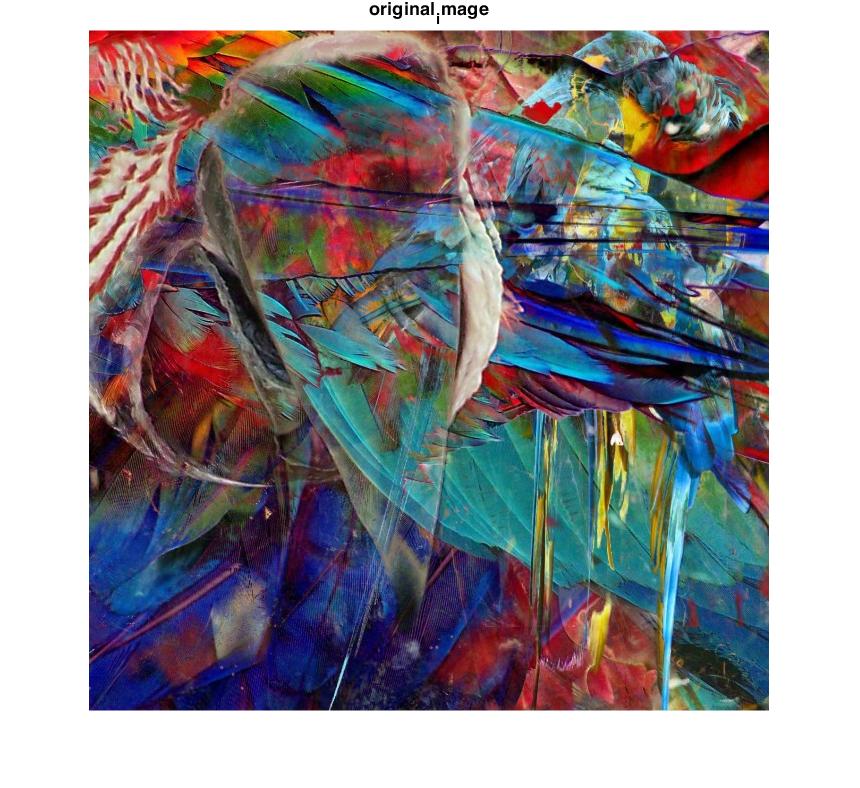


Figure 1 : Original Image

**Step 3- Select 16×16 cropped sub-image from your input at step2. Note that the starting point of the cropping window will depend on your Registration number. (Instructor will provide these details at the lab.)**

%e/16/103

%x position = 1\*50 = 50

%y position = 03\*4 = 12

cropped\_image = imcrop(original\_image,[50,12,15,15]);

cropped\_image\_gray = rgb2gray(cropped\_image);

figure;

imshow(cropped\_image);title('cropped\_image');

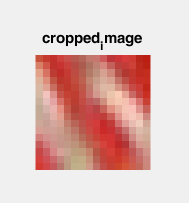
****

Figure 2 : Cropped image

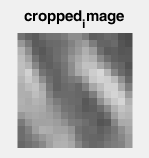
****

Figure 3 : Cropped image gray

**Step 4- Quantize the output at Step 3 into 8 levels (level 0-7) using uniform quantization.**

%step 3 - Quantize the output at Step 3 into 8 levels (level 0-7) using uniform quantization.

%CHANGE IMAGE HERE

choose\_image = cropped\_image\_gray;

thresh = multithresh(choose\_image ,7);

valuesMax = [thresh max(choose\_image (:))];

[quant\_image,index] = imquantize(original\_image ,thresh,valuesMax);

figure;

imshow(quant\_image);title('quantized\_image');

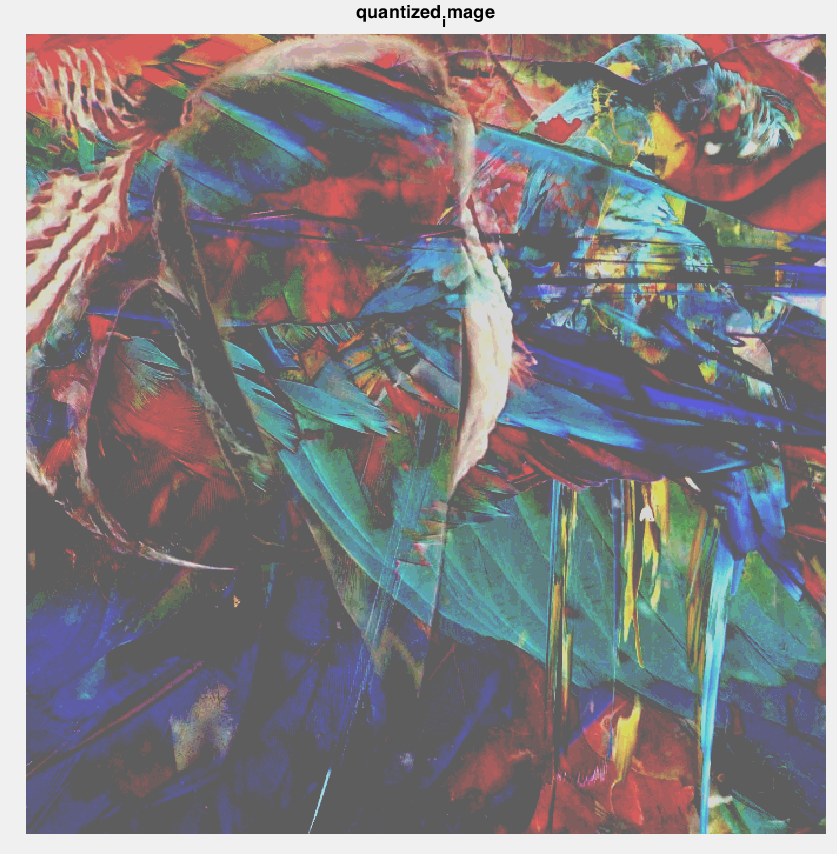
****

Figure 4 : Original image quantized with respect to cropped gray image

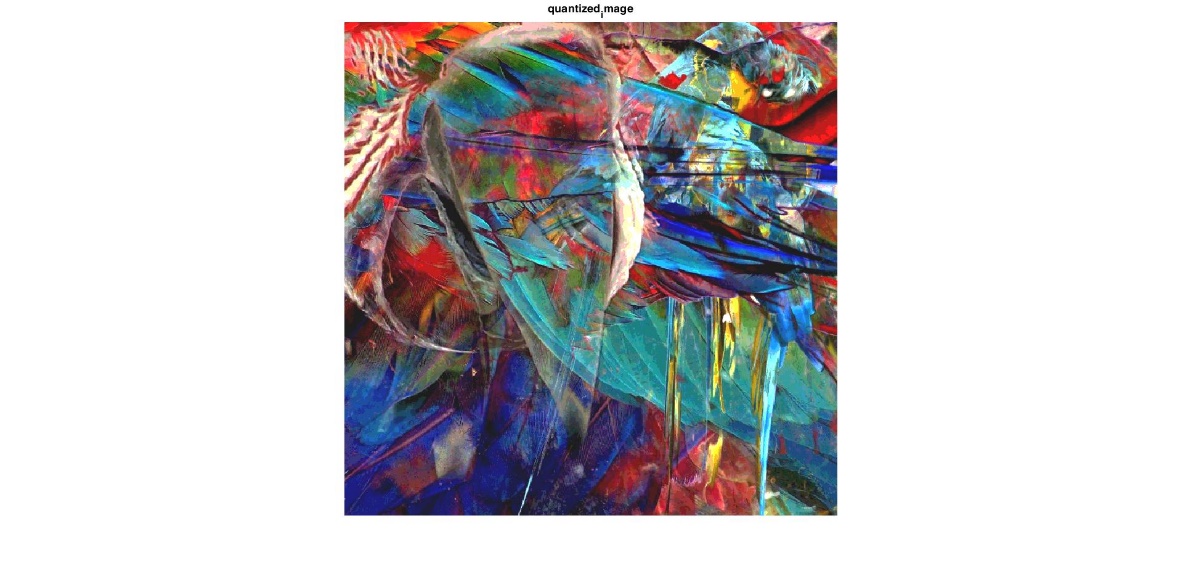
****

Figure 5 : Original image quantized with respect to original image

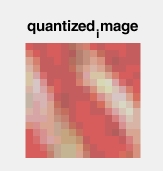


Figure 6 : Cropped image quantized with respect to cropped gray image

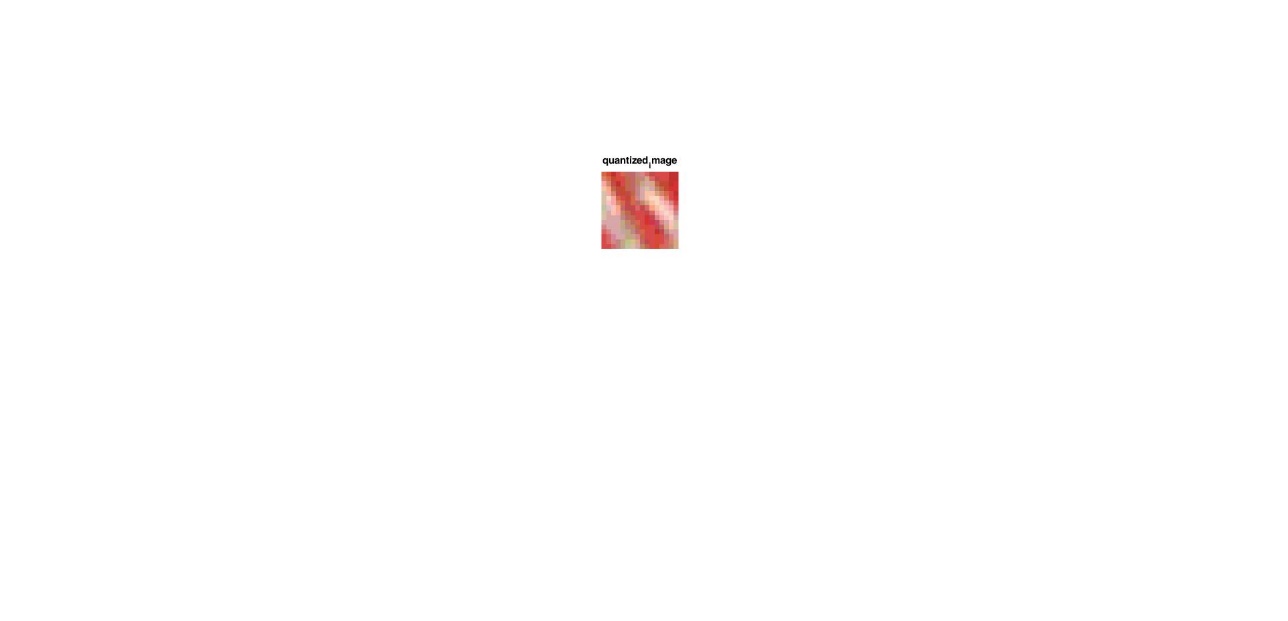
****

Figure 7 : Quantized cropped image

**Step 5- Find the probability of each symbol distribution of the output at Step 4.**

%step 4 - Find the probability of each symbol distribution of the output at Step

[g,~,intensity\_val] = grp2idx(quant\_image(:));

Frequency = accumarray(g,1);

[intensity\_val Frequency];

proability = Frequency./sum(Frequency);

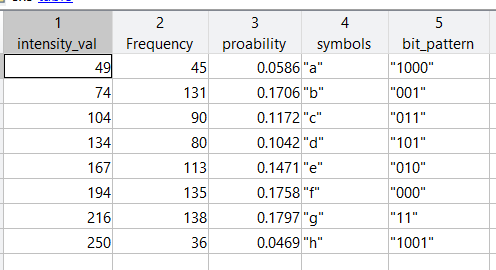


Figure 8 : Probability of symbols

**Step 6- Construct the Huffman coding algorithm for cropped image at Step 4.( Do not use inbuilt algorithms.)**

symbols = ["a" ;"b"; "c"; "d"; "e"; "f"; "g"; "h"]; %defining symbols as I find easy working with string and if we work wit intensity values there's a higher chance of performing arithmetic on them and we wont be able to use the as symbols

T=table(intensity\_val,Frequency,proability,symbols); %With tables we canstore different types of data organized.for example symbols(strings) and probabilities(double) connection is obvious with tables

Table\_ascending\_order = sortrows(T,{'proability'}); %sort rows in table considering the probability column in ascending order

Update\_array = table(Table\_ascending\_order.symbols,Table\_ascending\_order.proability); %temporary table for te ease of updating

new\_bits=[];

bit\_pattern=[];

for i=1:length(Table\_ascending\_order.proability)

Var1 = Update\_array.Var1(1)+Update\_array.Var1(2); %merge symbols with lowest probabilitis

Var2 = Update\_array.Var2(1)+Update\_array.Var2(2); %add lowest probabilitis

%bit sequence generation

for j=1:length(T.symbols) %This for loop is to traverse through "symbols" matrix and fill 1,0 acordingly to the "bit pattern" matrix

if( sum(char(T.symbols(j))== char(Update\_array.Var1(1))+0) ) %checking whether 'first merged symbol with lowest probabilities' above matches with the elements in "symbol(j)"

if(Update\_array.Var2(1) <= Update\_array.Var2(2)) %lowest probbabilities gets the 1

new\_bits = [new\_bits ; "1"];

else

new\_bits = [new\_bits ; "0"];

end

elseif( sum(char(T.symbols(j))== char(Update\_array.Var1(2))+0) ) %checking whether 'second merged symbol with lowest probabilities' above matches with the elements in "symbol(j)"

if(Update\_array.Var2(1) <= Update\_array.Var2(2))

new\_bits = [new\_bits ; "0"];

else

new\_bits = [new\_bits ; "1"];

end

else

new\_bits = [new\_bits ; " "];

end

end

bit\_pattern = [new\_bits bit\_pattern];

new\_bits=[];

if(Var2 == 1)

break;

else

new\_row = table(Var1, Var2);

Update\_array =[Update\_array;new\_row];

Update\_array([1,2],:) = [];

Update\_array = sortrows(Update\_array,{'Var2'});

end

end

bit\_pattern = erase(join(bit\_pattern(:,:)," ")," ");

T=[T table(bit\_pattern)];

encoded\_message = join(T.bit\_pattern(g(:)),""); %join bit pattern columns together to form the bit sequence

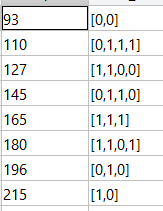
****

Figure 9 : Code word generated by the algorithm developed

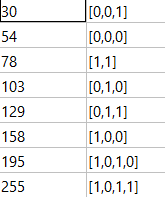
****

Figure 10 : Code word generated by the inbuilt algorithm

**Step 7- Compress both cropped and original images using the algorithm and the codebook generated at step 6. You may round any intensity values outside the codebook, to the nearest intensity value in the codebook, where necessary**.

%step 6 - : Save the compressed image into a text file.

fileID = fopen('original\_image\_gray\_huffman\_encoded.txt','w');

fprintf(fileID,encoded\_message);

fclose(fileID);

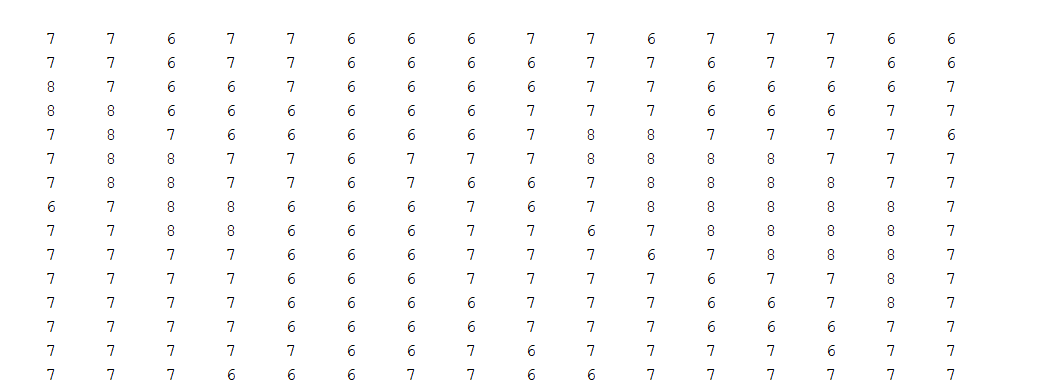


Figure 11 : Index array

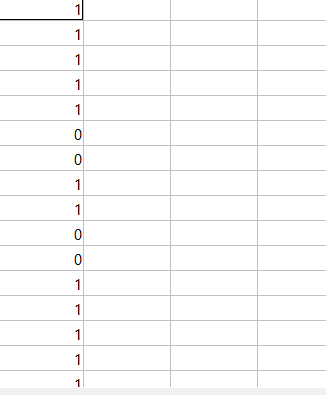


Figure 12 : generated code

**Step 8- Save the compressed image into a text file.**

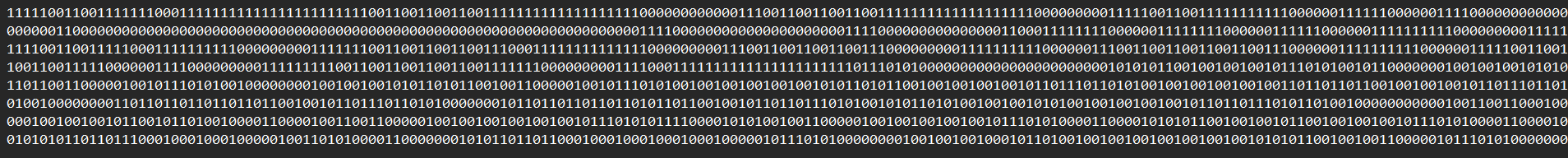
****

Figure 13 : cropped image encoded file

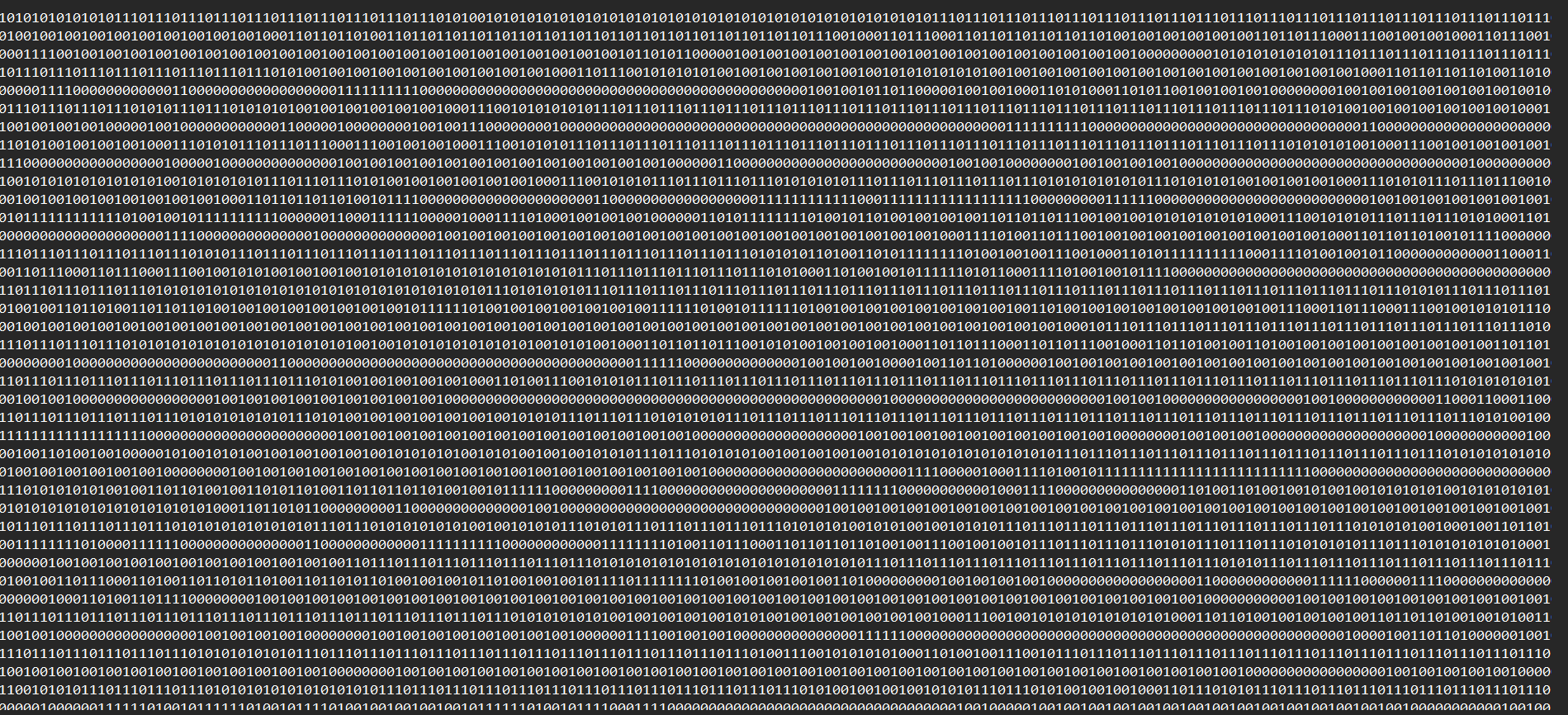


Figure 14 : full image encoded file

**Step 9- Compress the original image using Huffman encoding function in the Matlab tool box and save it into another text file.**

clear all;

clc;

%step 1 - Read the original image into a Matrix.

original\_image=imread('Parrots-680x680.jpg');

original\_image\_gray = rgb2gray(original\_image);

% figure;

% imshow(original\_image);title('original\_image');

%--------------------------------------------------------------------------

%step 2 - Select 16×16 cropped sub-image the starting point of the cropping window will depend on your Registration number

%e/16/103

%x position = 1\*50 = 50

%y position = 03\*4 = 12

cropped\_image = imcrop(original\_image,[50,12,15,15]);

cropped\_image\_gray = rgb2gray(cropped\_image);

% figure;

% imshow(cropped\_image);title('cropped\_image');

%--------------------------------------------------------------------------

%step 3 - Quantize the output at Step 3 into 8 levels (level 0-7) using uniform quantization.

thresh = multithresh(original\_image ,7);

valuesMax = [thresh max(original\_image(:))];

[quant\_image,index] = imquantize(original\_image,thresh,valuesMax);

% figure;

% imshow(quant\_image);title('quantized\_image');

%--------------------------------------------------------------------------

%step 4 - Find the probability of each symbol distribution of the output at Step

[g,~,intensity\_val] = grp2idx(quant\_image(:));

Frequency = accumarray(g,1);

[intensity\_val Frequency];

proability = Frequency./sum(Frequency);

input\_signal =reshape(quant\_image,[],1); %sould feed in a vector to huffman encode function

dict = huffmandict(intensity\_val,proability);

code = huffmanenco(input\_signal,dict);

text\_message = join(string(reshape(code,1,[])),"");

% file = fopen('cropped\_image\_rgb\_huffman\_inbuilt.txt','w');

% fprintf(file,text\_message);

% fclose(file);

%decoding recieved signal

sig = huffmandeco(code,dict);

pic=reshape(sig,680,680,3);

figure;

imshow(pic);

title('recovered image huffman inbuilt');

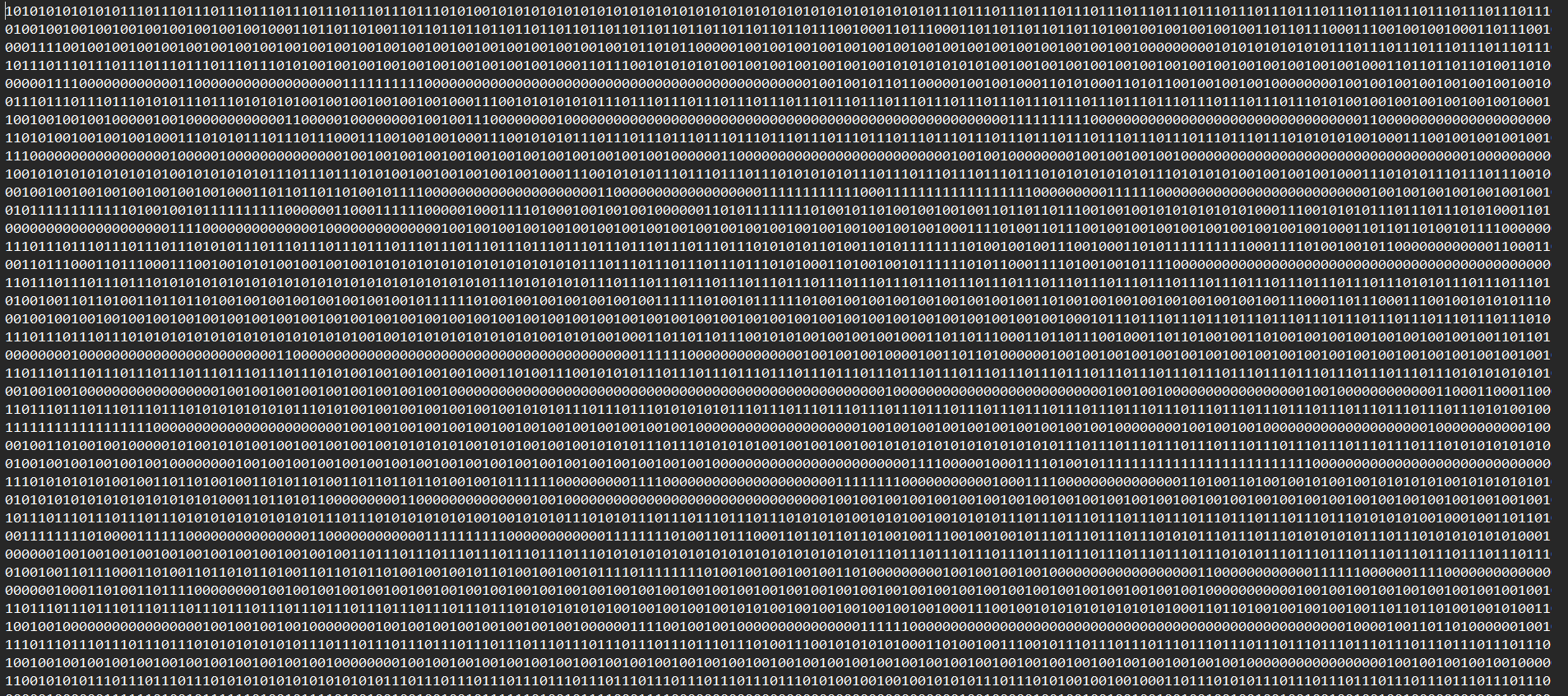
****

Figure 15 : full image encoded with matlab inbuilt

**Step 10- Decompress the outputs at Step 8 and 9, by reading in the text files.**

%step 7 - : Decompress the outputs at Step 8 and 9, by reading in the text files

code=double(reshape(char(encoded\_message),[],1))-48; %convert encoded message in to a form that inbuilt hhuffman decoder can understand

%dictionary file which is required to inbuilt huffman decoder

dict={};

for k=1:length(T.symbols)

cell={T.intensity\_val(k) double(char(T.bit\_pattern(k)))-48};

dict=[dict ; cell];

average\_length = length( (double(char(T.bit\_pattern(k)))-48) )\*T.proability(k); %croppped\_rgb\_image\_average\_length = 0.1875 , original rgd image average length = 0.1579

end

%decoding recieved signal

sig = huffmandeco(code,dict);

pic=reshape(sig,length(quant\_image),length(quant\_image),3);

figure;

imshow(pic);

title('recovered cropped image using huffman algorithm I developed');



Figure 16 : decompressed cropped image which was encoded with matlab inbuilt function

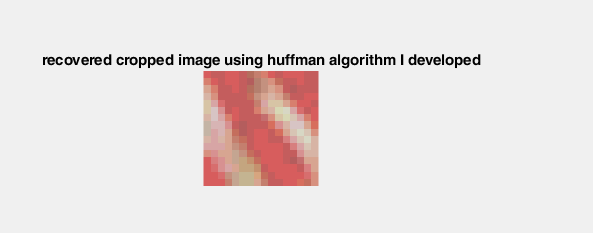


Figure 17 : decompressed cropped image which was encoded with the algorithm developed

****

Figure 18 : decompressed original image which was encoded with matlab inbuilt function

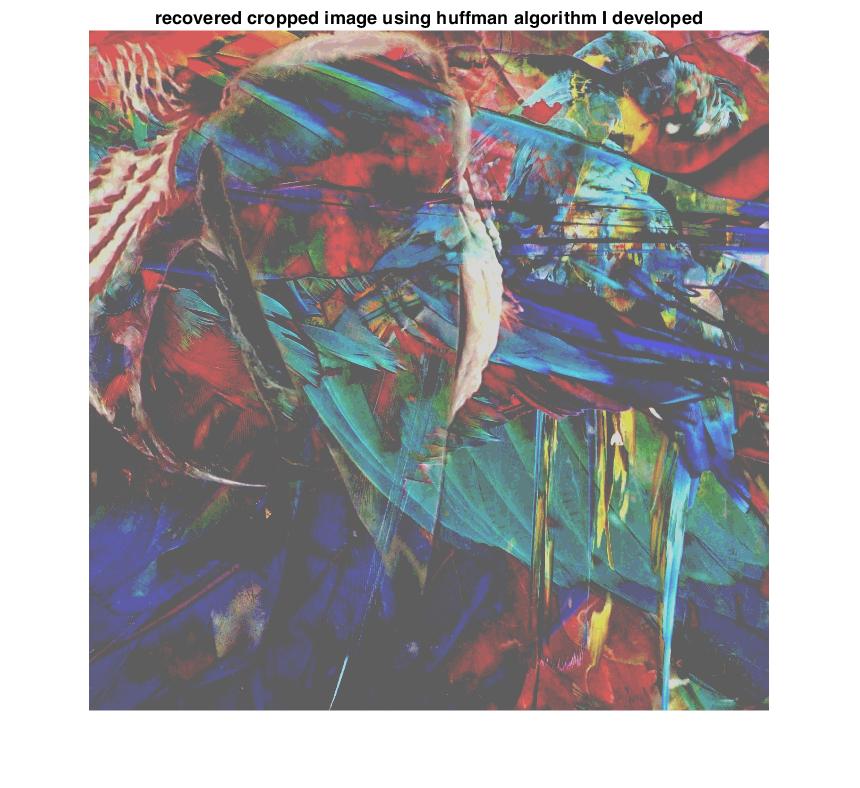


Figure 19 : decompressed original image which was encoded with with the algorithm developed (quantized w.r.t cropped gray image)

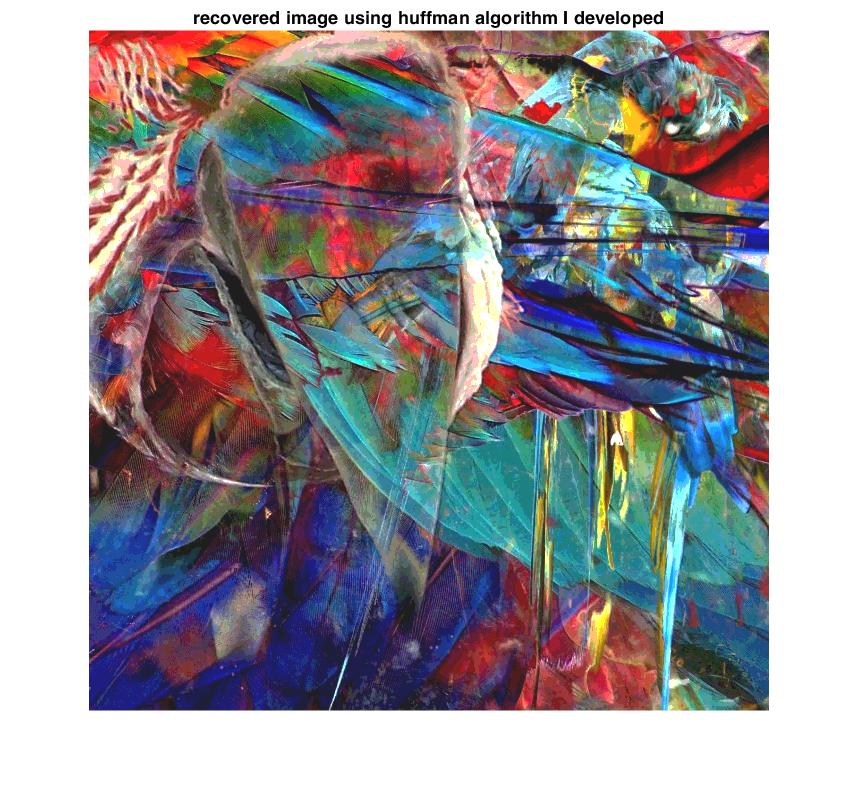
****

Figure 20 : decompressed original image which was encoded with with the algorithm developed (quantized w.r.t original image)

**Step 11- Calculate the entropy of the Source**

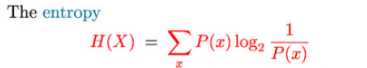
****

Figure 20 : Formula for the entropy

entropy\_of\_original = entropy(original\_image); %answer = 7.6683

entropy\_of\_decompressed = entropy(pic); %answer = 2.8953

entropy\_of\_cropped = entropy(cropped\_image); %answer = 7.3631

% entropy\_of\_decompressed\_cropped = entropy(pic); %answer = 2.8770

**Step 12- Evaluate the PSNR**

[peaksnr, snr] = psnr(pic, original\_image); %peak snr = 23.3000 , snr = 15.4179

PSNR original image = inf

PSNR of the decoded source image which was = 16.2686

encoded with algorithm developed

PSNR of the decoded cropped image which was = 24.2753

encoded with algorithm developed

PSNR of the decoded source image which was = 23.300

encoded with inbuilt algorithm

PSNR of the decoded cropped image which was = 24.2753

encoded with inbuilt algorithm

**DISCUSSIONS**

**1) Calculate entropy**

Entropy of original image = 7.6683

Entropy of decompressed original image = 2.8953

Entropy of cropped images = 7.3631

Entropy of decompressed cropped images = 2.8770

**2) Calculate the average length of the cropped image**.

for k=1:length(T.symbols)

average\_length = length( (double(char(T.bit\_pattern(k)))-48) )\*T.proability(k); %croppped\_rgb\_image\_average\_length = 0.1875 , original rgd image average length = 0.1579

end

Average code length original image = 0.1579

Average code length cropped image = 0.1875

**3) Compare the performance of your algorithm and inbuilt algorithm of Matlab by comparing the compression ratios, for cropped and original images**

%step 10 - :

compression\_ratio cropped = length(reshape(cropped\_image,[],1))/length(code);

compression\_ratio original = length(reshape(original\_image,[],1))/length(code);

For algorithm developed

Compression ratio for original image = 3.7393

Compression ratio for cropped image = 2.9383

For algorithm inbuilt

Compression ratio for original image = 2.7128

Compression ratio for cropped image = 2.7344

Results are contrasting. Hence the algorithm can be developed to works as good as inbuilt algorithm if we use the original full source image in quantization rather than using a gray scale cropped image.

**4) Discuses about Entropy of the input image, the compression ratio achieved, and the output quality of the decompressed image.**

Entropy of original image = 7.6683

Entropy of decompressed original image = 2.8953

We can see that the entropy has dropped when compressed. Because of the quantization more predictable the image becomes. Hence the information such a file carries is low. The reason for drop in entropy is justified with this fact.

Compression ratio is high. Hence higher compression is achievable. Quality is degraded because the image is quantized to 8 bits with respect to a cropped gray scale image. Algorithm can be developed to perform as good as the inbuilt function by taking the whole image into account in quantization process.

**5) How can you improve the compression ratio of the given image? Discuss.**

Optimize the code to save processing power

Transmitting recursive bits with the frequency and the binary value

Using non uniform quantization

Using a large and a fairer cropped image in the quantization process.