JPEG Image Decoder and its Implementation in Matlab

By Vaibhav Sharma

**Introduction**

The term "JPEG" is an acronym for the Joint Photographic Experts Group which created the standard. JPEG is a commonly used method of lossy compression for photographic images. The degree of compression can be adjusted, allowing a selectable trade off between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality. JPEG compression is used in a number of image file formats. JPEG/Exif is the most common image format used by digital cameras and other photographic image capture devices. It is the most common format for storing and transmitting photographic images on the World Wide Web.

The JPEG standard specifies the codec, which defines how an image is compressed into a stream of bytes and decompressed back into an image, but not the file format used to contain that stream.

**Typical Usage**

* The JPEG compression algorithm is at its best on photographs and paintings of realistic scenes with smooth variations of tone and color.
* Most common image format used by digital cameras.
* Most common format for storing and transmitting photographic images on the www.
* May not be as well suited for line drawings and other textual or iconic graphics, where the sharp contrasts between adjacent pixels can cause noticeable artifacts.
* Also not well suited to files that will undergo multiple edits.
* It should not be used in scenarios where the exact reproduction of the data is required.

**Problem Statement**

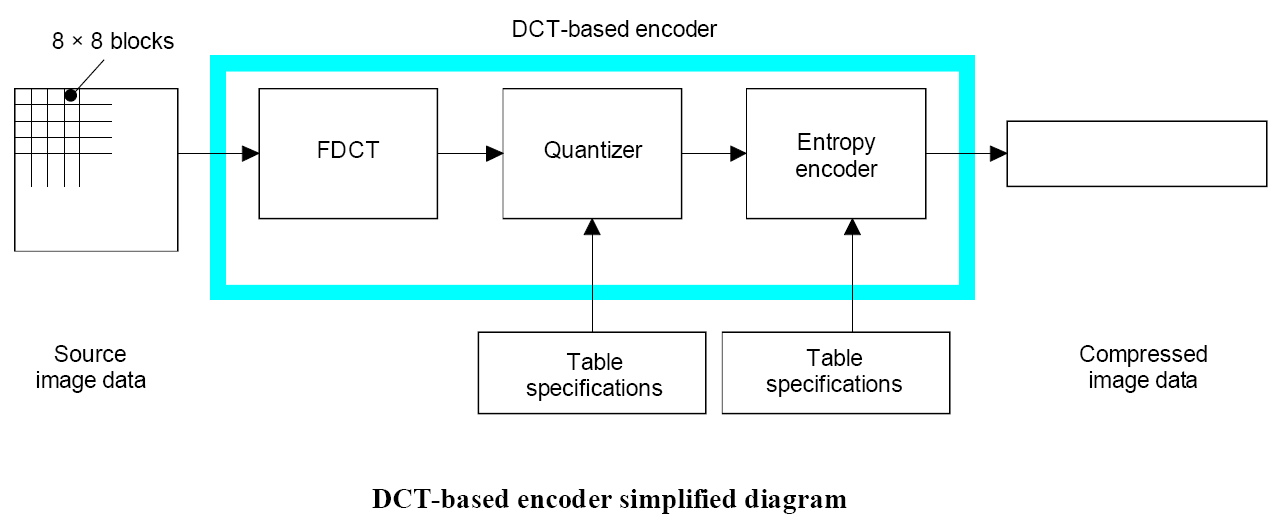
Implement a JPEG decoder in MATLAB for Baseline Sequential decoding process using Huffman encoding as entropy encoding scheme. This is not a robust JPEG decoder. The idea is to implement a very basic JPEG decoder to understand the logic behind JPEG compression. This JPEG decoder is implemented using some of the fundamental JPEG functions.

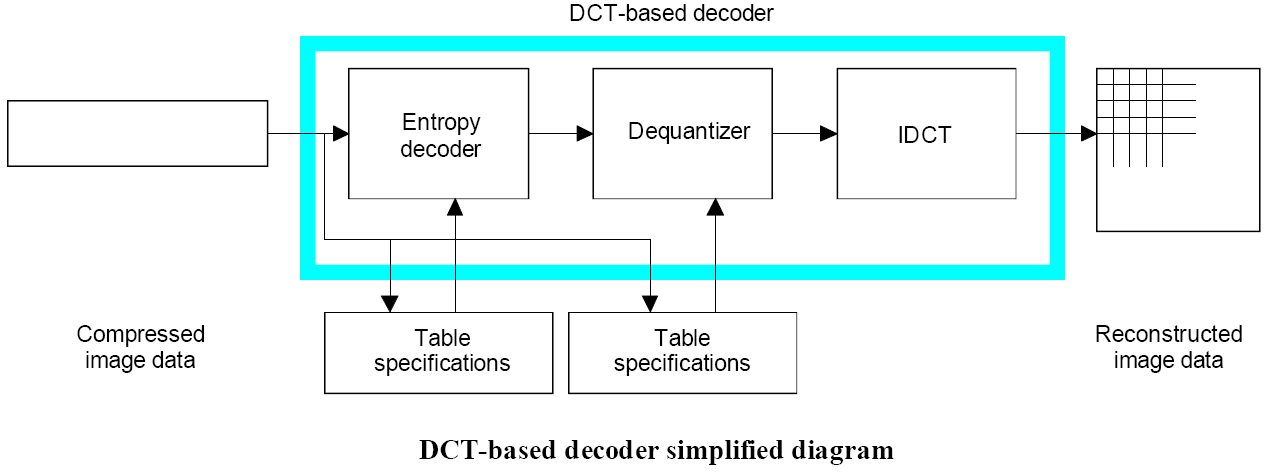
**JPEG CODEC**

The encoding process consists of several steps:

1. The representation of the colors in the image is converted from RGB to Y′CBCR, consisting of one luma component (Y'), representing brightness, and two chroma components, (CB and CR), representing color. This step is sometimes skipped.
2. The resolution of the chroma data is reduced, usually by a factor of 2. This reflects the fact that the eye is less sensitive to fine color details than to fine brightness details.
3. The image is split into blocks of 8×8 pixels, and for each block, each of the Y, CB, and CR data undergoes a discrete cosine transform (DCT). A DCT is similar to a Fourier transform in the sense that it produces a kind of spatial frequency spectrum.
4. The amplitudes of the frequency components are quantized. Human vision is much more sensitive to small variations in color or brightness over large areas than to the strength of high-frequency brightness variations. Therefore, the magnitudes of the high-frequency components are stored with a lower accuracy than the low-frequency components. The quality setting of the encoder affects to what extent the resolution of each frequency component is reduced. If an excessively low quality setting is used, the high-frequency components are discarded altogether.
5. The resulting data for all 8×8 blocks is further compressed with a lossless algorithm, a variant of Huffman encoding.

The decoding process reverses these steps, except the quantization because it is irreversible.





**Modes of operation**

There are four distinct modes of operation under which the various coding processes are defined:

1) Sequential DCT-based

2) Progressive DCT-based

3) Lossless, and

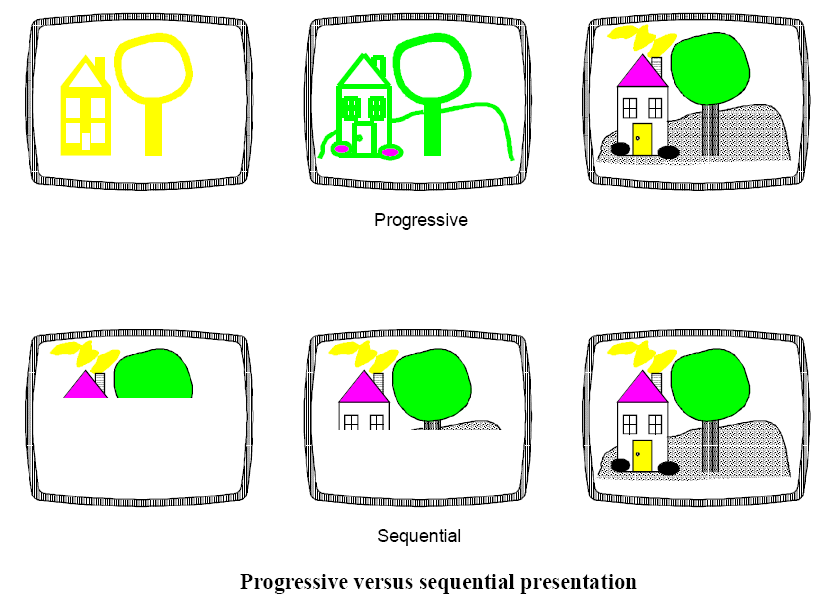
4) Hierarchical

**1) Sequential DCT-based**

* For the sequential DCT-based mode, 8 x 8 sample blocks are typically input block by block from left to right, and block-row by block-row from top to bottom.
* After a block has been transformed by the forward DCT, quantized and prepared for entropy encoding, all 64 of its quantized DCT coefficients can be immediately entropy encoded and output as part of the compressed image data.

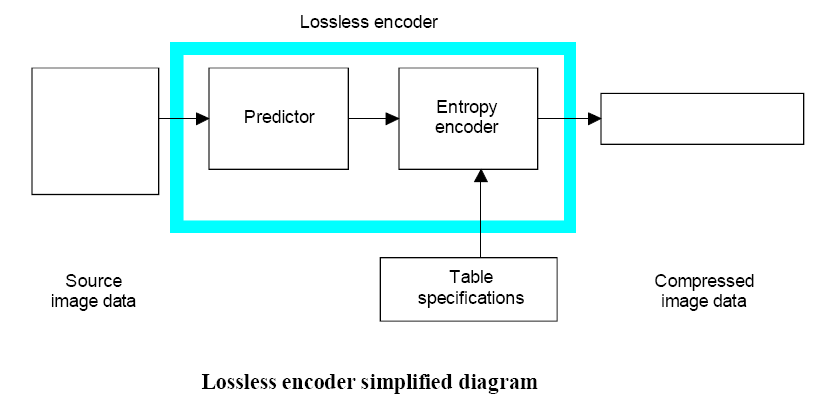
**2) Progressive DCT-based**

* For the progressive DCT-based mode, 8 x 8 blocks are also typically encoded in the same order, but in multiple *scans* through the image.
* This is accomplished by adding an image-sized coefficient memory buffer between the quantizer and the entropy encoder.
* As each block is transformed by the forward DCT and quantized, its coefficients are stored in the buffer.
* The DCT coefficients in the buffer are then partially encoded in each of multiple scans.



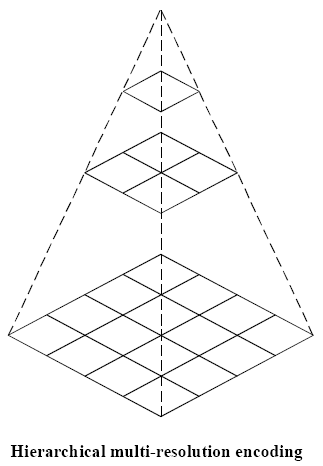
**3) Lossless**

* This class of coding processes is not based upon the DCT.
* This mode is provided to meet the needs of applications requiring lossless compression.
* These lossless encoding and decoding processes are used independently of any of the DCT-based processes.



**4) Hierarchical**

* In hierarchical mode, an image is encoded as a sequence of frames.
* These frames provide reference reconstructed components which are usually needed for prediction in subsequent frames.
* Except for the first frame for a given component, differential frames encode the difference between source components and reference reconstructed components.
* The coding of the differences may be done using only DCT-based processes, only lossless processes, or DCT-based processes with a final lossless process for each component.
* Downsampling and upsampling filters may be used to provide a pyramid of spatial resolutions.
* Useful in environments which have multi-resolution requirements



**Decoder Requirements**

A decoder shall

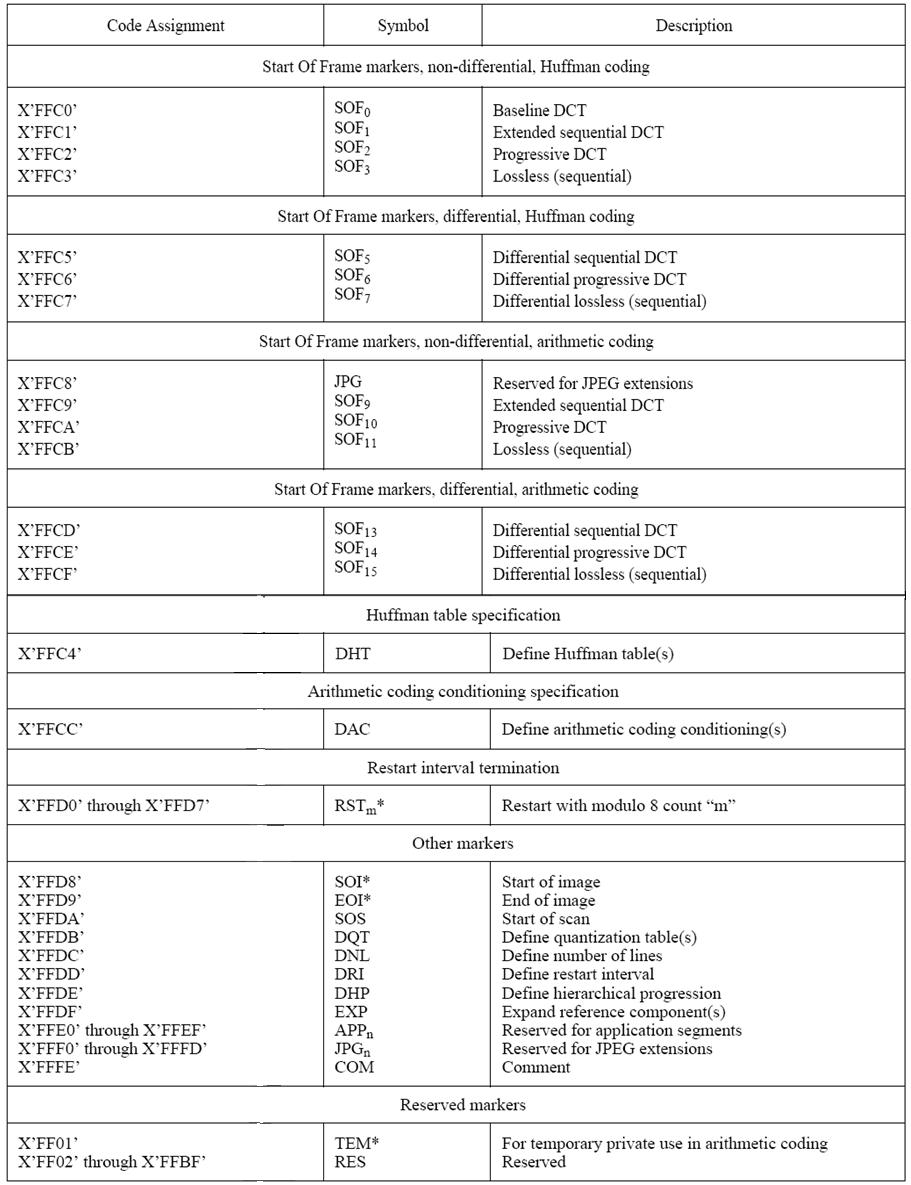
**a)** with appropriate accuracy, convert to reconstructed image data any compressed image data with parameters within the range supported by the application, and which comply with the interchange format syntax for the decoding process embodied by the decoder;

**b)** accept and properly store any table-specification data which comply with the abbreviated format for table specification data syntax for the decoding process embodied by the decoder;

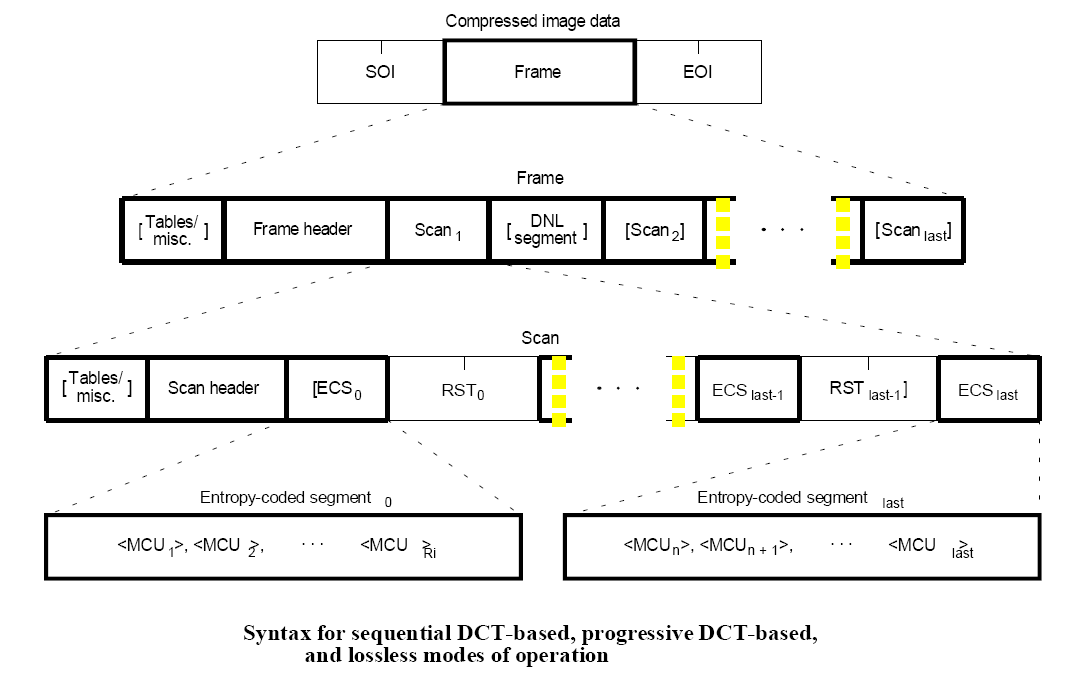
**c)** with appropriate accuracy, convert to reconstructed image data any compressed image data which comply with the abbreviated format for compressed image data syntax for the decoding process embodied by the decoder, provided that the table-specification data required for decoding the compressed image data has previously been installed into the decoder.

**Markers**

Markers serve to identify the various structural parts of the compressed data formats. Most markers start marker segments containing a related group of parameters; some markers stand alone. All markers are assigned two-byte codes: an X’FF’ byte followed by a byte which is not equal to 0 or X’FF’. Any marker may optionally be preceded by any number of fill bytes, which are bytes assigned code X’FF’.



**High Level Syntax**

****

**Implementation of JPEG Decoder in MATLAB**

**1. Declare all markers in the beginning.**

%declaration of markers-I have declared only those markers that were required for decoding of the jpeg image with following specifications:-

1) baseline (sequential) profile

2) Huffman entropy coding

3) No restart intervals present in image

4) Only luminance tables are considered

SOF0=[255; 192];

DHT=[255;196];

RST1=[255;208];

RST2=[255;209];

RST3=[255;210];

RST4=[255;211];

RST5=[255;212];

RST6=[255;213];

RST7=[255;214];

RST8=[255;215];

SOI=[255;216];

EOI=[255;217];

SOS=[255;218];

DQT=[255;219];

DNL=[255;220];

DRI=[255;221];

APP1=[255;224];

APP2=[255;225];

APP3=[255;226];

APP4=[255;227];

APP5=[255;228];

APP6=[255;229];

APP7=[255;230];

APP8=[255;231];

APP9=[255;232];

APP10=[255;233];

APP11=[255;234];

APP12=[255;235];

APP13=[255;236];

APP14=[255;237];

APP15=[255;238];

APP16=[255;239];

TC\_TH\_DC1=[0];

TC\_TH\_DC2=[1];

TC\_TH\_AC1=[16];

TC\_TH\_AC2=[17];

**2. Open the jpeg file in read mode and read the file byte by byte. Check for Start of Image (SOI) marker. This marker indicates that the code following the SOI marker is compressed image data. We process the image data until we detect the End Of Image (EOI) marker.**

%Open the jpeg file in read mode and read the first two bytes.

fid=fopen('lena64.jpg','r');

soi=fread(fid,2);

%Checking for valid start of image marker

if soi(1,1)~=SOI(1,1) || soi(2,1)~=SOI(2,1)

disp('Incorrect SOI marker: ')

disp(soi)

exit;

end

**3. Keep reading the next two bytes till the Start Of Frame (SOF) marker is detected. Process all the markers that are detected before SOF marker. The markers detected before SOF marker are called ‘Interpret markers’. The following markers may be detected before SOF marker:-**

**(i) Marker for Application segments (These are application specific markers. The application marker for all jpeg images have “74 70 73 70 00” code defined in the application marker. Presence of this code states that the image follows JPEG specification.)**

**(ii) Marker for Quantization table (This marker deals with creation of Quantization tables.)**

**(iii) Marker for AC/DC Huffman tables (This marker deals with creation of Huffman tables for DC/AC coefficients.)**

**(iv) Marker for Restart Interval (This marker defines the restart interval.)**

%check whether the next two bytes are SOF marker or the interpret marker.

Appropriate functions are called for each of the interpret markers detected.

temp=fread(fid,2);

while temp(1,1)~=SOF0(1,1) || temp(2,1)~=SOF0(2,1)

if temp(1,1)==APP1(1,1) && temp(2,1)==APP1(2,1)

define\_application\_marker(fid);

%application marker

temp=fread(fid,2);

elseif temp(1,1)==DQT(1,1) && temp(2,1)==DQT(2,1)

quant\_table=define\_quantization\_table(fid);

%create quantization table

temp=fread(fid,2);

elseif temp(1,1)==DHT(1,1) && temp(2,1)==DHT(2,1)

length=bin2dec(strcat(dec2bin(fread(fid,1)),dec2bin(fread(fid,1))));

tc\_th=fread(fid,1);

if (tc\_th==TC\_TH\_DC1) || (tc\_th==TC\_TH\_DC2)

[dc\_huff\_table dc\_mat]=define\_dc\_huffman\_table(fid);

%create huffman table for dc coeeficients

elseif (tc\_th==TC\_TH\_AC1) || (tc\_th==TC\_TH\_AC2)

[ac\_huff\_table ac\_mat]=define\_ac\_huffman\_table(fid);

%create huffman table for ac coeeficients

end

temp=fread(fid,2);

elseif temp(1,1)==DRI(1,1) && temp(2,1)==DRI(2,1)

rst\_int=define\_restart\_interval(fid);

%restart interval marker

temp=fread(fid,2);

end

end

**4. When Start Of Frame (SOF) marker is detected, then, process the ‘Frame Header’ for that SOF marker.**

%SOF marker has been detected. Now, we evaluate frame header.

frame\_length\_count=2;

length\_frame=bin2dec(strcat(dec2bin(fread(fid,1)),dec2bin(fread(fid,1))));

prec\_frame=fread(fid,1);frame\_length\_count=frame\_length\_count+1;

y\_frame=fread(fid,2);frame\_length\_count=frame\_length\_count+2;

x\_frame=fread(fid,2);frame\_length\_count=frame\_length\_count+2;

components\_frame=fread(fid,1);frame\_length\_count=frame\_length\_count+1;

for i=1:components\_frame

component\_id(1,i)=fread(fid,1);frame\_length\_count=frame\_length\_count+1;

h\_v\_sampling(1,i)=fread(fid,1);frame\_length\_count=frame\_length\_count+1;

quant\_dest\_select(1,i)=fread(fid,1);frame\_length\_count=frame\_length\_count+1;

end

if frame\_length\_count < length\_frame

fread(fid,1);

frame\_length\_count=frame\_length\_count+1;

end

**5. Keep reading the next two bytes till the Start Of Scan (SOS) marker is detected. Process all the markers that are detected before SOS marker. The markers detected before SOS marker are called ‘Interpret markers’. The following markers may be detected before SOF marker:-**

**(i) Marker for Application segments (These are application specific markers. The application marker for all jpeg images have “74 70 73 70 00” code defined in the application marker. Presence of this code states that the image follows JPEG specification.)**

**(ii) Marker for Quantization table (This marker deals with creation of Quantization tables.)**

**(iii) Marker for AC/DC Huffman tables (This marker deals with creation of Huffman tables for DC/AC coefficients.)**

**(iv) Marker for Restart Interval (This marker defines the restart interval.)**

%At this point, frame header has been evaluated. We Move ahead to evaluate Start Of Scan marker.

%check whether the next two bytes are SOS marker or the interpret marker.

Appropriate functions are called for each of the interpret markers detected.

dc\_huff\_table=0;

dc\_mat=0;

ac\_huff\_table=0;

ac\_mat=0;

temp=fread(fid,2);

while temp(1,1)~=SOS(1,1) || temp(2,1)~=SOS(2,1)

%check whether the sos marker has been detected or it is the interpret marker.

if temp(1,1)==APP1(1,1) && temp(2,1)==APP1(2,1)

define\_application\_marker(fid);

%application marker

temp=fread(fid,2);

elseif temp(1,1)==DQT(1,1) && temp(2,1)==DQT(2,1)

quant\_table=define\_quantization\_table(fid);

%create quantization table

temp=fread(fid,2);

elseif temp(1,1)==DHT(1,1) && temp(2,1)==DHT(2,1)

length=bin2dec(strcat(dec2bin(fread(fid,1)),dec2bin(fread(fid,1))));

tc\_th=fread(fid,1);

if (tc\_th==TC\_TH\_DC1) || (tc\_th==TC\_TH\_DC2)

[dc\_huff\_table dc\_mat]=define\_dc\_huffman\_table(fid,length);

%create huffman table for dc coeeficients

elseif (tc\_th==TC\_TH\_AC1) || (tc\_th==TC\_TH\_AC2)

[ac\_huff\_table ac\_mat]=define\_ac\_huffman\_table(fid,length);

%create huffman table for ac coeeficients

end

temp=fread(fid,2);

elseif temp(1,1)==DRI(1,1) && temp(2,1)==DRI(2,1)

%restart interval marker

define\_restart\_interval(fid);

temp=fread(fid,2);

end

end

**6. When Start Of Scan (SOS) marker is detected, then, process the ‘Scan Header’ for that SOS marker.**

%SOS marker has been detected. Now, we evaluate scan header.

scan\_length\_counter=2;

scan\_header\_length=bin2dec(strcat(dec2bin(fread(fid,1)),dec2bin(fread(fid,1))));

image\_components=fread(fid,1);

for i=1:image\_components

scan\_component\_select(1,i)=fread(fid,1);

dc\_ac\_dest\_select(1,i)=fread(fid,1);

end

start\_spectral=fread(fid,1);

end\_spectral=fread(fid,1);

ah\_al=fread(fid,1);

%scan header interpretation complete

**7. After Scan Header has been interpreted, decode all the entropy encoded segments for the given scan component. We store the data corresponding to these entropy encoded segments in a string variable ‘s’. String ‘s’ contains data values in binary form.**

%we store the entropy encoded value in a string variable ‘s’.

s='';s\_len=0;num\_eof=1;

temp=fread(fid,1);temp1=fread(fid,1);fseek(fid,-1,'cof');

while temp~=EOI(1,1) || temp1~=EOI(2,1)

if temp==255 && temp1==0

fread(fid,1);

end

temp\_bin=dec2bin(temp,8);

s=strcat(s,temp\_bin);s\_len=s\_len+8;

temp=fread(fid,1);temp1=fread(fid,1);fseek(fid,-1,'cof');

end

**8. Process the entropy encoded binary data in String ‘s’. First, calculate the DC coefficient value using DC Huffman Table. Then calculate the 63 AC coefficient values using AC Huffman Table. Also, undo the zig-zag ordering of DCT coefficients.**

%decoding the dc coefficient

JJ=2;KK=1;

j=JJ;

k=KK;

flag=0;

str\_test='';

for i=1:s\_len-k+1

str\_test(i)='1';

end

%This if statement is to make sure that the binary values left to be decoded are stuffed ‘1’ values(‘1s’ are padded to entropy coded data (during encoding) so that size of the resultant entropy coded data is integral number of bytes).

if ~strcmp(s(k:s\_len),str\_test)

tab\_data=[];

while flag==0

s\_data=s(k:j);s\_data\_len=j-k+1;

[res category]=check\_dc(s\_data,s\_data\_len,dc\_huff\_table,dc\_mat);

if res==[1]

flag=1;

if ~strcmp(s\_data,'00')

data\_extract=s(j+1:j+category);

if data\_extract(1)=='1'

data\_extract\_dec=bin2dec(data\_extract);

else

data\_extract\_dec=bin2dec(data\_extract);

data\_extract\_dec=bitcmp(data\_extract\_dec,category);

data\_extract\_dec=-data\_extract\_dec;

end

k=j+category+1;

j=k+1;

else

data\_extract\_dec=0;

k=j+1;j=k+1;

end

else

j=j+1;

end

end

data\_elem=1;

tab\_data(data\_elem)=data\_extract\_dec;

data\_elem=data\_elem+1;

%decoding the ac coefficients until EOB is detected

for i=k:s\_len

s\_data=s(k:j);s\_data\_len=j-k+1;

[res r\_val cat\_val]=check\_ac(s\_data,s\_data\_len,ac\_huff\_table,ac\_mat);

if res==1

if strcmp(s\_data,'1010')

k=j+1;j=k+1;

break;

end

if ~strcmp(s\_data,'11111111001')

data\_extract=s(j+1:j+cat\_val);

if data\_extract(1)=='1'

data\_extract\_dec=bin2dec(data\_extract);

else

data\_extract\_dec=bin2dec(data\_extract);

data\_extract\_dec=bitcmp(data\_extract\_dec,cat\_val);

data\_extract\_dec=-data\_extract\_dec;

end

else

data\_extract\_dec=0;

end

for z=1:r\_val

tab\_data(data\_elem)=0;

data\_elem=data\_elem+1;

end

tab\_data(data\_elem)=data\_extract\_dec;

data\_elem=data\_elem+1;

k=j+cat\_val+1;

j=k+1;

if data\_elem==65

break;

end

else

j=j+1;

end

end

KK=k;JJ=j;

%all ac coefficients after EOB are assigned value = 0

for i=data\_elem:64

tab\_data(i)=0;

end

u=1;

for w=1:8

for v=1:8

data\_matrix1(w,v)=tab\_data(u);

u=u+1;

end

end

%Undo the zig-zag ordering of dct coefficients in the 8x8 block.

data\_matrix12=undo\_the\_zigzag(data\_matrix1);

end

**9. Multiply all the 8x8 blocks in the JPEG image file by 8x8 Quantization Table.**

fhandle2=@(x) x.\*quant\_table;

new\_image2=blkproc(data\_matrix\_final1,[8 8],fhandle2);

**10. The DC coefficient in the 8x8 block is actually the difference (DIFF) between the quantized DC coefficient of the current block (*DCi*) and that of the previous block of the same component. In this step, we perform the following operation:-**

**DCi=DCi-1+PRED;**

**PRED=DCi;**

**11. Compute the inverse DCT of the resultant 8x8 matrix block.**

fhandle3=@idct2;

new\_image3=round(blkproc(new\_image2,[8 8],fhandle3));

**12. Add 2precision-1 to the 8x8 matrix block values.**

new\_image3=new\_image3 + 128;

**13. Display the resultant matrix to view the decompressed image.**

imshow(new\_image3,[0 255])

**14. Close the file**

status=fclose(fid); %close the file