DIGITAL SIGNAL PROCESSING

JPEG IMAGE COMPRESSION

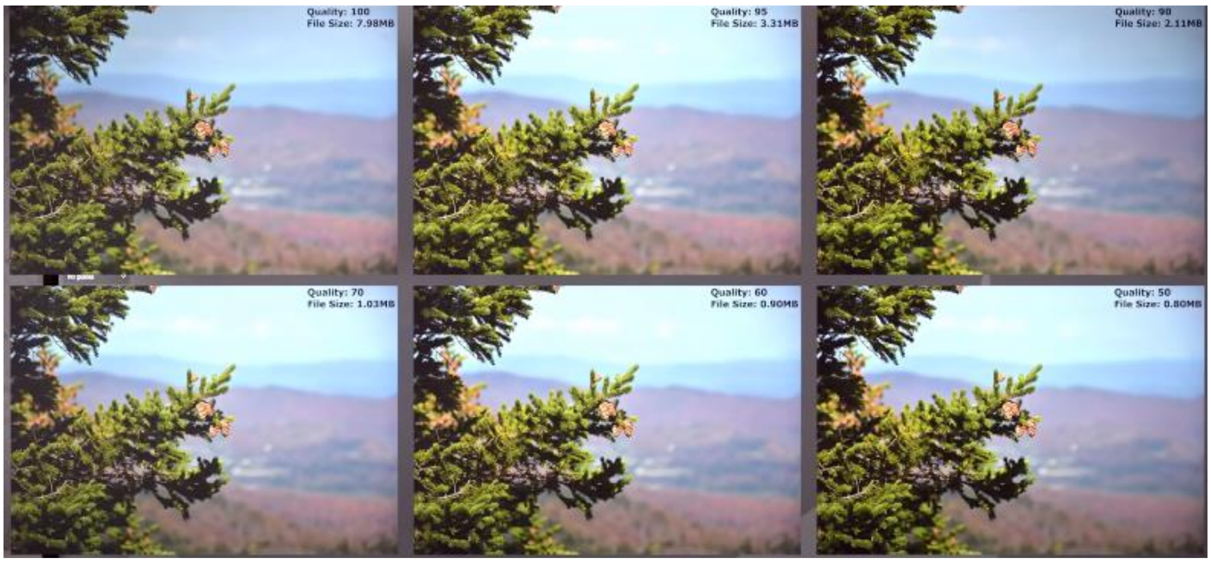
INTRODUCTION:

JPEG Compression is the name given to an algorithm developed by the Joint Photographic Experts Group whose purpose is to minimize the file size of photographic image files. JPEG Compression can greatly reduce the size of an image file but it is also lossy compression. So, it can also compromise the quality of an image.

In computers, photographic images can be thought of as a grid of individual blocks called "pixels". Each pixel has its own color value, and the larger the image, the more pixels. The more pixels, the larger the resulting file will be.

In order to reproduce the image, the unique color of every pixel must be recorded. JPEG compression uses the DCT(Discrete Cosine Transform). It allows a tradeoff between storage size and the compression ratio can be adjusted.

JPEG goes through and analyses each section of an image and finds and removes elements that our eyes can't easily perceive. JPEG exploits the fact that our eyes are not great at perceiving certain frequencies and removes them.



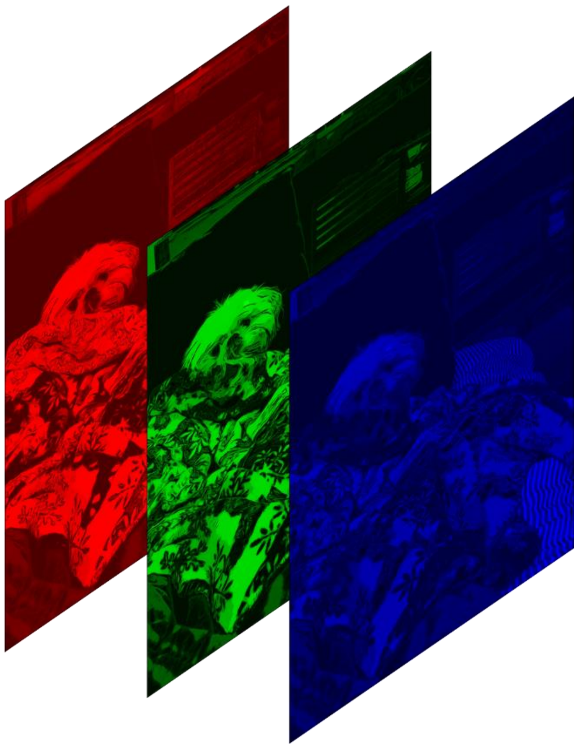
EXPLANATION:

JPEG compression is broadly of five stages:

- Extraction of Red, Green, and Blue channels in an image
- Dividing these channels into 8X8 blocks
- Applying DCT on each 8X8 block and then quantization
- Encoding using Zig-Zag technique
- Decoding and display of compressed image

❖ EXTRACTION OF RED, GREEN, AND BLUE CHANNELS OF AN IMAGE:

Every image is composed of small blocks called pixels. Each pixel represents the intensity of the image at that location. Every color can be represented by combinations of different values of Red, Green, and Blue each ranging from 0 to 255. The image is stored as 3 two dimensional matrices each representing pixels values at that location of Red, Green and blue colors.



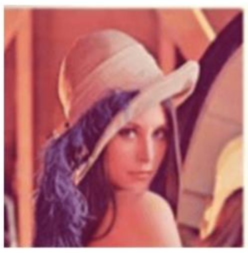
In Matlab, the "imread " command is used to extract the red, green, and blue channels of an image. It gives a three-dimensional array as output

i.e. 3 two-dimensional arrays each for red, green, and blue colors respectively.
But before extracting the color channels we have to check the pixels of the given image were multiples of 8. If not we zero-padded the rows and columns where it is necessary such that we get pixels that are multiples of 8.

```
old_img = imread("C:\Users\91630\OneDrive\Documents\sample4.jpg");
imshow(old_img);

old_img = double(old_img);

[size1,size2,size3] = size(old_img)

rows = size1;
columns = size2;

k1 = rem(size1,8)
k2 = rem(size2,8)

if k1~=0
    rows = size1 + 8-k1
end

if k2~=0
    columns=size2+8-k2
end

i =1;
j = 1;
img = zeros(rows,columns,3);
while logical(rem(1,2))
    if i>size1
        break;
    end
    j = 1;
    while logical(rem(1,2))
       if j>size2
           break;
       end
            img(i,j,1)=old_img(i,j,1);
            img(i,j,2)=old_img(i,j,2);
            img(i,j,3)=old_img(i,j,3);
            j = j+1;
    end
     i = i+1;
end

imshow(uint8(img))
```

DIVIDING EACH CHANNEL INTO 8X8 BLOCKS:

JPEG Compression tends to even out transitional colors, the points in an image where one color becomes a different color. The input image is divided into a small block which has 8X8 dimensions to compute DCT.


Original image

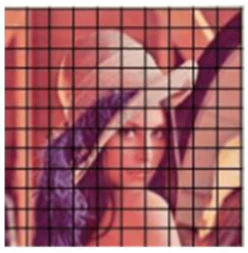
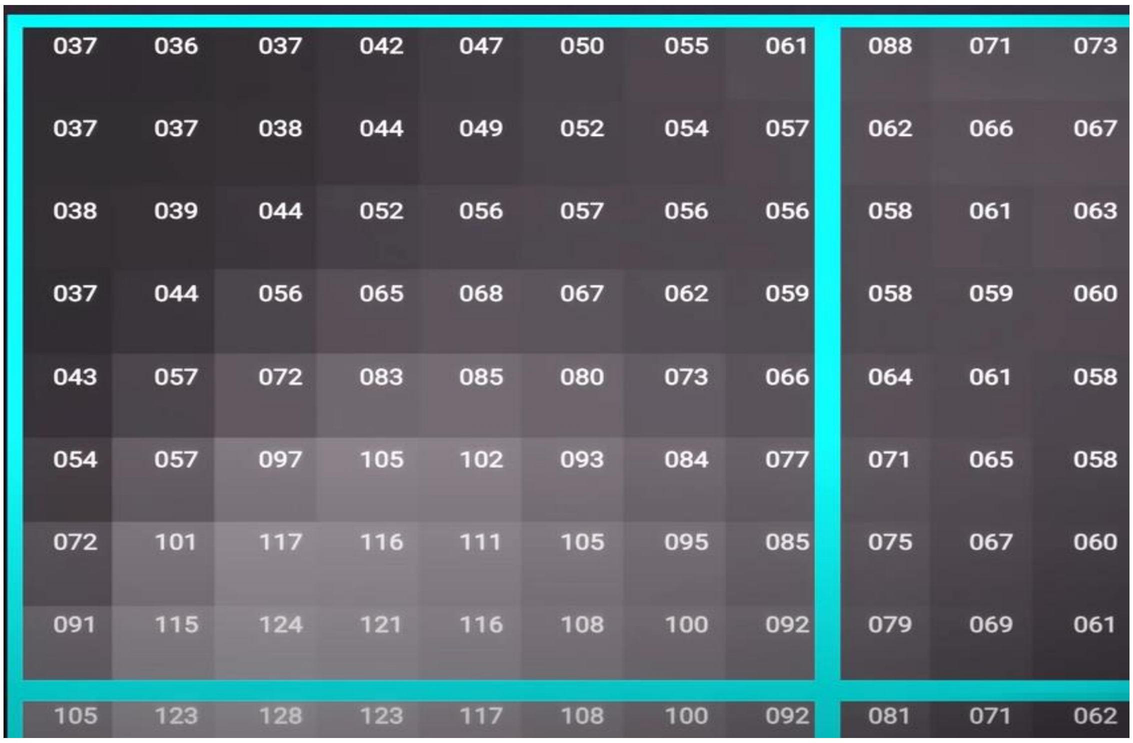


Image split into 8x8 pixel blocks



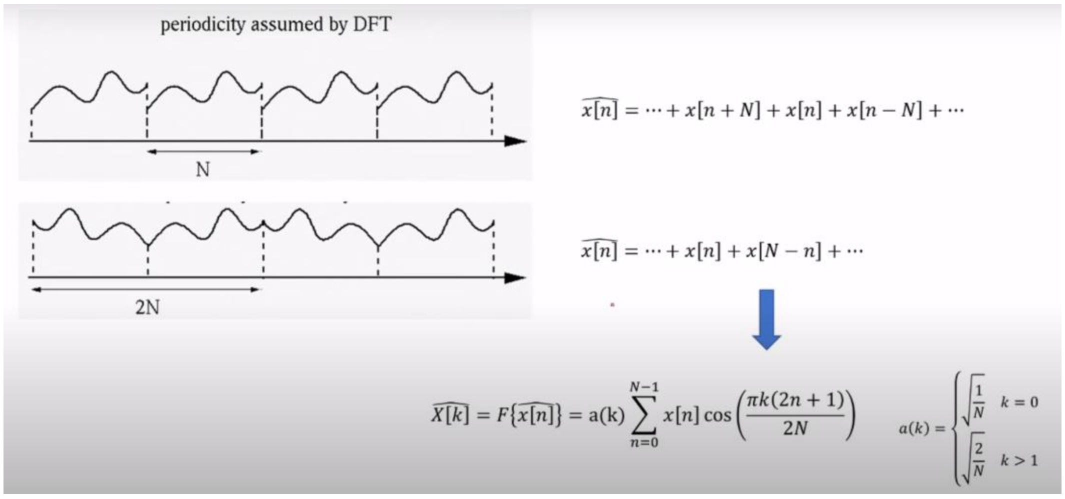
037	036	037	042	047	050	055	061	088	071	073
037	037	038	044	049	052	054	057	062	066	067
038	039	044	052	056	057	056	056	058	061	063
037	044	056	065	068	067	062	059	058	059	060
043	057	072	083	085	080	073	066	064	061	058
054	057	097	105	102	093	084	077	071	065	058
072	101	117	116	111	105	095	085	075	067	060
091	115	124	121	116	108	100	092	079	069	061
105	123	128	123	117	108	100	092	081	071	062

❖ APPLYING DCT ON EACH 8X8 BLOCK AND THEN QUANTIZATION:

A **Discrete cosine transform (DCT)** expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. The DCT, first proposed by Nasir Ahmed in 1972, is a widely used transformation technique in signal processing and data compression.

Advantages of DCT over DFT to get characteristics of an image:

DFT is calculated for a finite-length sequence. It is made periodic by repeating the sequence with specific periodicity. So, there will be discontinuities in the signal which results in the 'Artifacts' in image. So we use DCT in which the flipped signal is repeated.

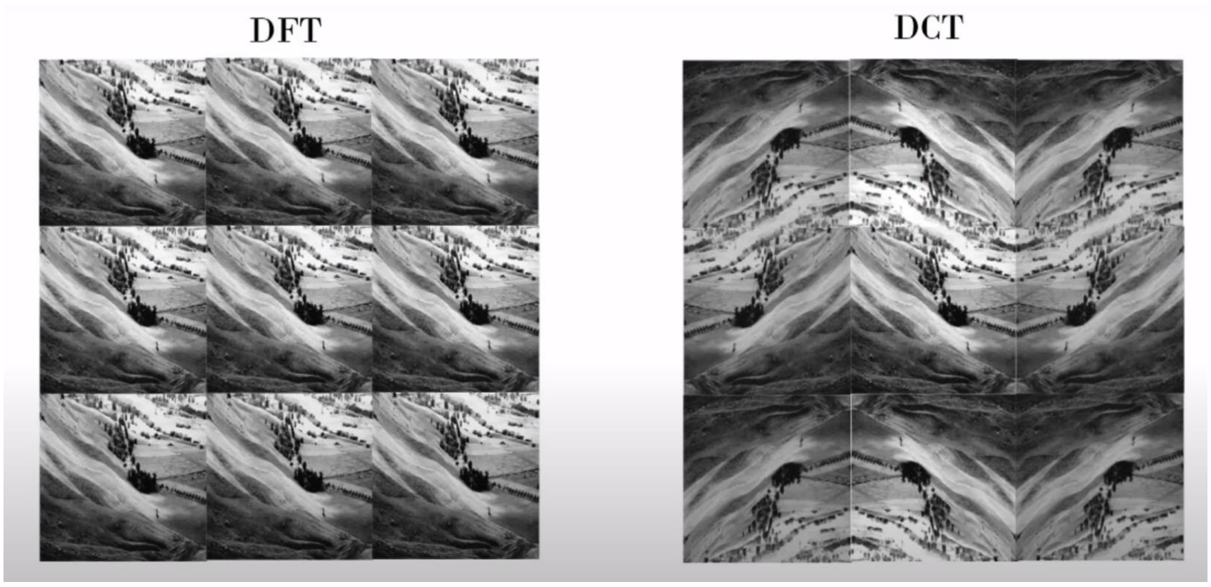


$$\overline{x[n]} = \cdots + x[n+N] + x[n] + x[n-N] + \cdots$$

$$\overline{x[n]} = \cdots + x[n] + x[N-n] + \cdots$$

$$\overline{X[k]} = F\{\overline{x[n]}\} = \mathrm{a}(\mathrm{k}) \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi k(2n+1)}{2N}\right) \qquad a(k) = \begin{cases} \sqrt{\frac{1}{N}} & k=0 \\ \sqrt{\frac{2}{N}} & k>1 \end{cases}$$

In the above figure, we can see that the flipped signal is repeated which results in no discontinuities as a result there will be no artifacts.



The gradual variation in the signal results in no vertical lines as in DFT.

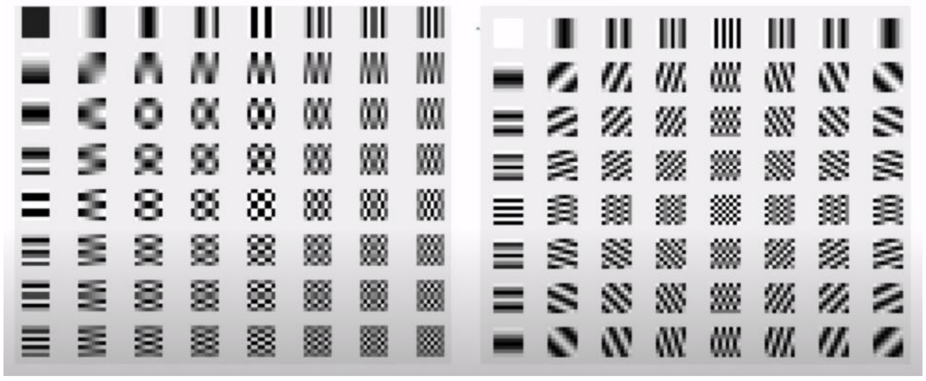
DFT uses complex numbers, unlike DCT which uses only cosine functions.

We can calculate the DCT 2D array using this:

$$F(u, v) = u_k v_k \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f[x, y] \cos\left(\frac{\pi u(2x+1)}{2N}\right) \cos\left(\frac{\pi v(2y+1)}{2N}\right)$$

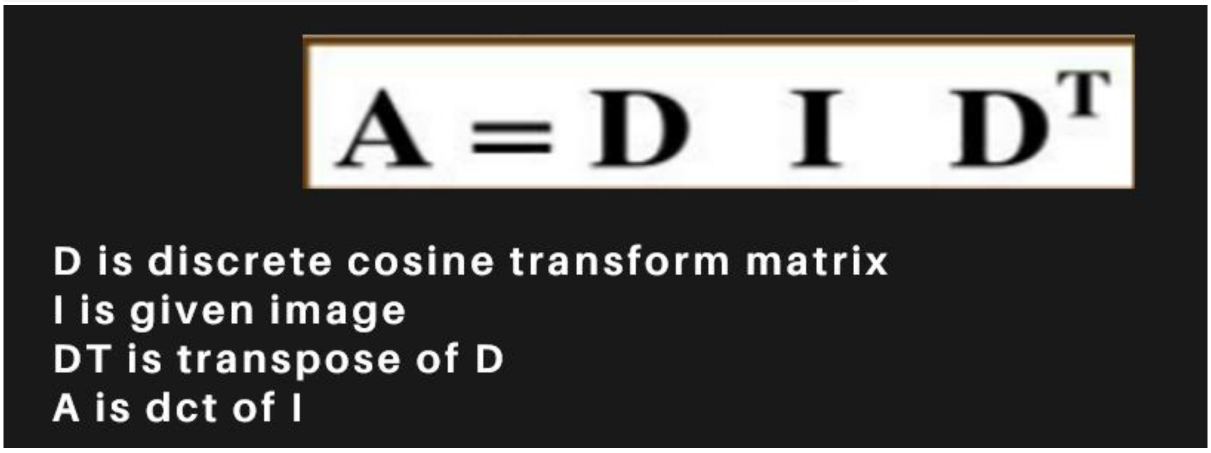
$$u_k \& v_k = \begin{cases} \sqrt{\frac{1}{N}} & u, v = 0 \\ \sqrt{\frac{2}{N}} & u, v > 1 \end{cases}$$

In DCT there will be a gradual increase in the frequencies as we traverse along the 2D array whereas in DFT it will be repetitive.



DCT of each channel is calculated individually for every 8X8 block. We used a cosine matrix for the calculation of DCT:

```
cosine_matrix = [0.3536 0.3536 0.3536 0.3536 0.3536 0.3536 0.3536 0.3536;
    0.4904 0.4157 0.2778 0.0975 -0.0975 -0.2778 -0.4157 -0.4904;
    0.4619 0.1913 -0.1913 -0.4619 -0.4619 -0.1913 0.1913 0.4619;
    0.4157 -0.0975 -0.4904 -0.2778 0.2778 0.4904 0.0975 -0.4157;
    0.3536 -0.3536 -0.3536 0.3536 0.3536 -0.3536 -0.3536 0.3536;
    0.2778 -0.4904 0.0975 0.4157 -0.4157 -0.0975 0.4904 -0.2778;
    0.1913 -0.4619 0.4619 -0.1913 -0.1913 0.4619 -0.4619 0.1913;
    0.0975 -0.2778 0.4157 -0.4904 0.4904 -0.4157 0.2778 -0.0975;
    ];
```



$$A = D \quad I \quad D^T$$

D is discrete cosine transform matrix
I is given image
DT is transpose of D
A is dct of I

After DCT calculation we level-shift the 8X8 by subtracting 128 pixels such that we get values in the range -128 to 127 instead of 0 to 255 as DCT is designed to work on pixel values ranging from -128 to 127.

```
R_channel_img = img(:,:,1) ;

number_of_horizontal_blocks = columns/8

number_of_vertical_blocks = rows/8

i = 1;
j = 1;
while i<=number_of_vertical_blocks
    j = 1;

    while j<=number_of_horizontal_blocks
         x=1;
        y =1;
        eight_cross_eight_block = R_channel_img(8*i-7:8*i,8*j-7:8*j);

        m = eight_cross_eight_block - 128;


dct = cosine_matrix*m*cosine_matrix';

quantizied = round(dct./quantization);

%  Code for Encoding continues....
```

Quantization matrix:

This is the lossy part of the compression.

We quantize the 8X8 block 'A' obtained from DCT by dividing it with the quantization matrix and rounding off the values.

```
quantization = [16 11 10 16 24 40 51 61;
                12 12 14 19 26 58 60 55;
                14 13 16 24  40 57 69 56;
                14 17 22 29 51 87 80 62;
                18 22 37 56 68 109 103 77;
                24 35 55 64 81 104 113 92;
                49 64 78 87 103 121 120 101;
                72 92 95 98 112 100 103 99;];



compression_ratio = 20;

quantization = quantization * compression_ratio;
```

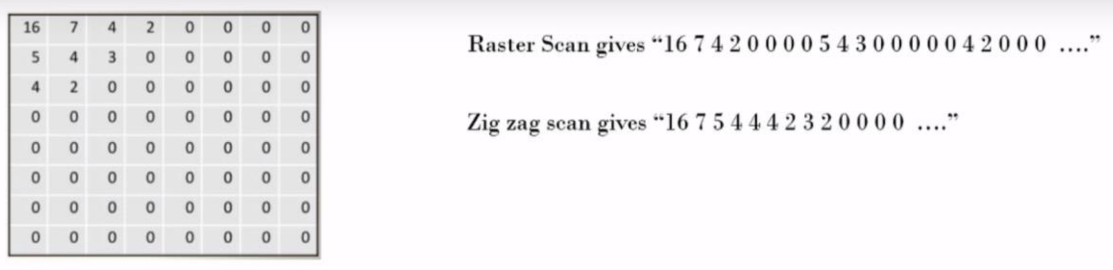
The quantization matrix decides how much an image is to be compressed.

The amount of compression can be controlled by the compression ratio.

In the quantization matrix we can see that the top left values are small since distinct patterns of an images are located. As we go to the right the values increases where higher frequencis are present which our eyes can't easily perceive.

Standard Formula: $$\hat{F}(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

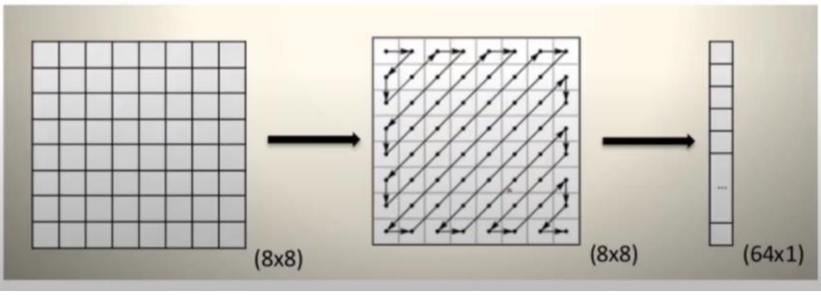
❖ ENCODING USING ZIG-ZAG TECHNIQUE:



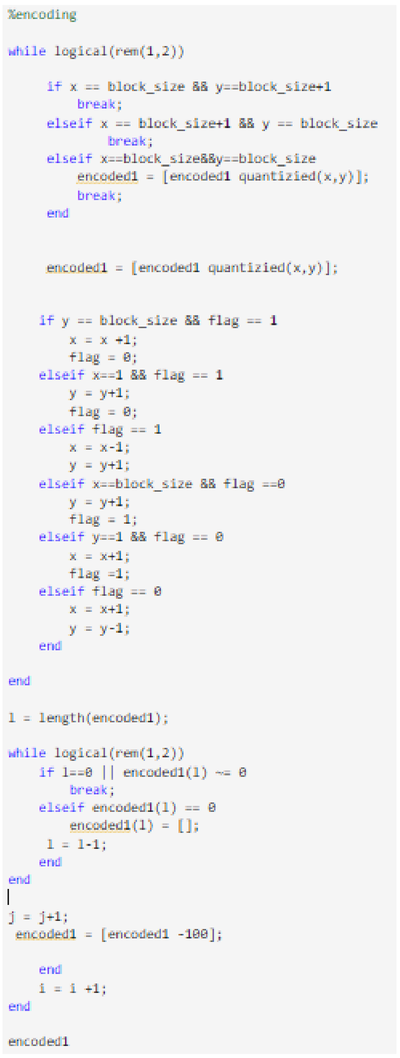
16	7	4	2	0	0	0	0
5	4	3	0	0	0	0	0
4	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Raster Scan gives "16 7 4 2 0 0 0 0 5 4 3 0 0 0 0 0 4 2 0 0 0 ...."

Zig zag scan gives "16 7 5 4 4 4 2 3 2 0 0 0 0 ...."



After calculating DCT and quantization we get a 8X8 matrix. To encode this we can do Raster Scan or Zig-zag scan.
Raster scan scans the matrix row by row which might result in zero value first and then again high value. So we follow a zig-zag scan such that we get values in a decreasing order. When zero is encountered we can simply move to the next 8X8 block which greatly decreases the space for storage.



```
%encoding

while logical(rem(1,2))

     if x == block_size && y==block_size+1
         break;
     elseif x == block_size+1 && y == block_size
             break;
     elseif x==block_size&&y==block_size
         encoded1 = [encoded1 quantizied(x,y)];
         break;
     end


     encoded1 = [encoded1 quantizied(x,y)];


    if y == block_size && flag == 1
        x = x +1;
        flag = 0;
    elseif x==1 && flag == 1
        y = y+1;
        flag = 0;
    elseif flag == 1
        x = x-1;
        y = y+1;
    elseif x==block_size && flag ==0
        y = y+1;
        flag = 1;
    elseif y==1 && flag == 0
        x = x+1;
        flag =1;
    elseif flag == 0
        x = x+1;
        y = y-1;
    end

end

l = length(encoded1);

while logical(rem(1,2))
    if l==0 || encoded1(l) ~= 0
        break;
    elseif encoded1(l) == 0
        encoded1(l) = [];
     l = l-1;
    end
end
|
j = j+1;
 encoded1 = [encoded1 -100];

    end
    i = i +1;
end

encoded1
```

To recognize the truncation in the 8X8 block we added -100 to the array when we encountered a zero. The same is repeated for the other two color channels.

❖ DECODING AND DISPLAY OF COMPRESSED IMAGE:

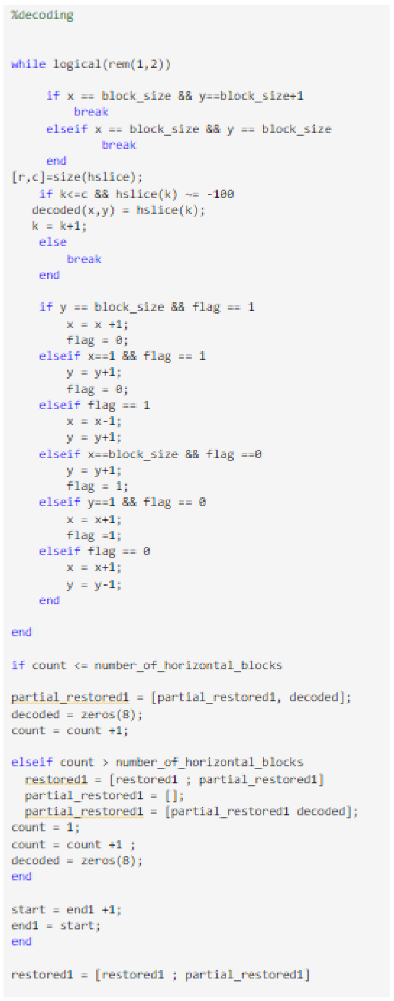
To get the compressed image we decode the encoded values by retracing the encoded values to get 8X8 block back.
After getting the 8X8 block we multiply it with the quantisation matrix and then apply the inverse DCT.

inverse discrete cosine transform (IDCT)

the general inverse transform equation for DCT is

$$\boxed{I = D^T A \ D}$$

Finally, we level-shift the obtained matrix by adding 128 pixels.



```
%decoding


while logical(rem(1,2))

     if x == block_size && y==block_size+1
         break
     elseif x == block_size && y == block_size
             break
     end
[r,c]=size(hslice);
    if k<=c && hslice(k) ~= -100
   decoded(x,y) = hslice(k);
   k = k+1;
    else
        break
    end

    if y == block_size && flag == 1
        x = x +1;
        flag = 0;
    elseif x==1 && flag == 1
        y = y+1;
        flag = 0;
    elseif flag == 1
        x = x-1;
        y = y+1;
    elseif x==block_size && flag ==0
        y = y+1;
        flag = 1;
    elseif y==1 && flag == 0
        x = x+1;
        flag =1;
    elseif flag == 0
        x = x+1;
        y = y-1;
    end

end

if count <= number_of_horizontal_blocks

partial_restored1 = [partial_restored1, decoded];
decoded = zeros(8);
count = count +1;

elseif count > number_of_horizontal_blocks
  restored1 = [restored1 ; partial_restored1]
  partial_restored1 = [];
  partial_restored1 = [partial_restored1 decoded];
count = 1;
count = count +1 ;
decoded = zeros(8);
end

start = end1 +1;
end1 = start;
end

restored1 = [restored1 ; partial_restored1]
```

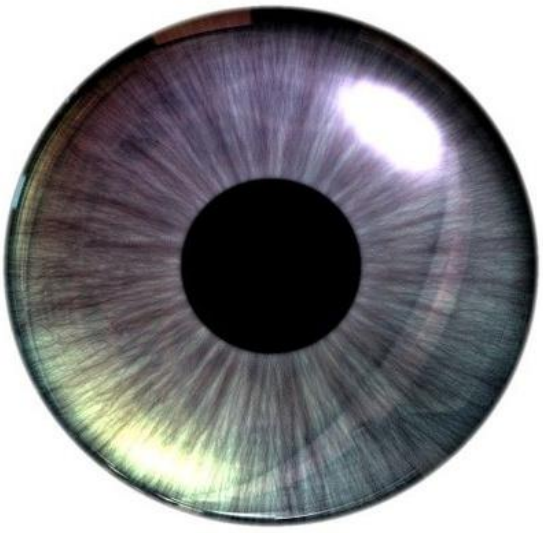
EXECUTION:

Run the Matlab code in the following order:

1) DCT_ENCODING.mlx
2) DECODING.mlx
3) RESTORE.mlx

Sample Test:

Original Image:

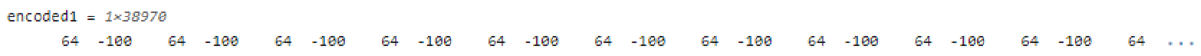


The original image is of 512 X 512 pixels.

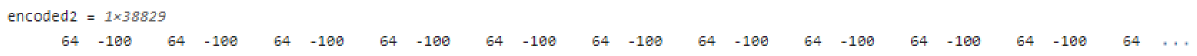
Size of the image = 512*512*3*8

= 6291456 bits

When compressed with a compression factor 1, following are the encoded values:



```
encoded1 = 1×38970
      64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  ...
```



```
encoded2 = 1×38829
      64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  -100    64  ...
```

```
encoded3 = 1×39203
      64   -100     64   -100     64   -100     64   -100     64   -100     64   -100     64   -100     64   -100     64   -100     64   -100     64  ...
```

Size of the encoded image = (38970+38829+39203)*8

= 936016 bits

We can clearly see the encoded image has a smaller size than the original

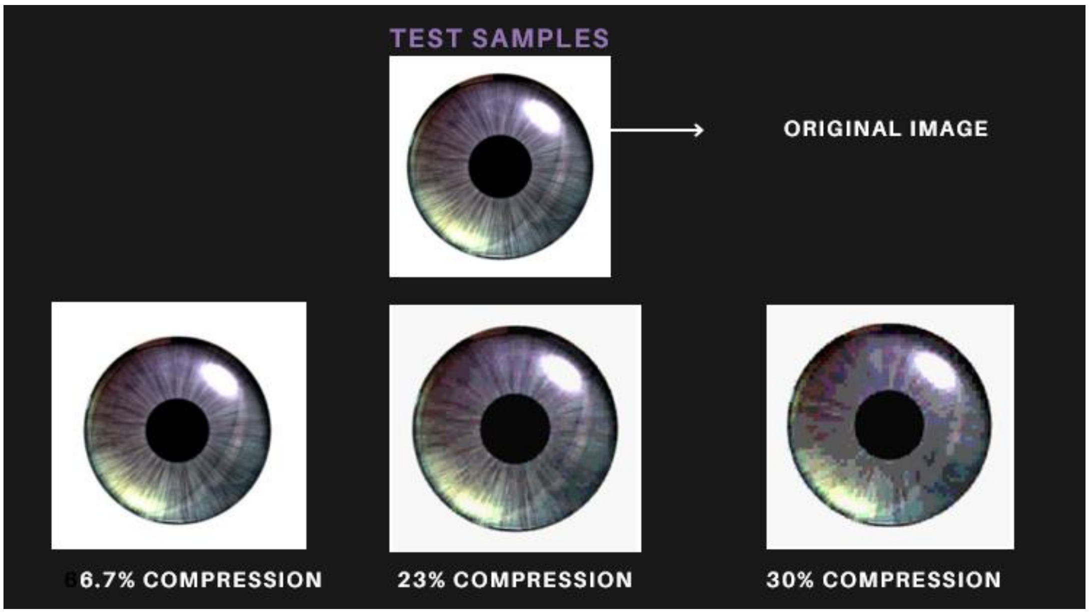
We can see the compressed image hasn't lost much of its clarity since the compression ratio is less.

When the compression factor is increased to 10. The image size has decreased further.

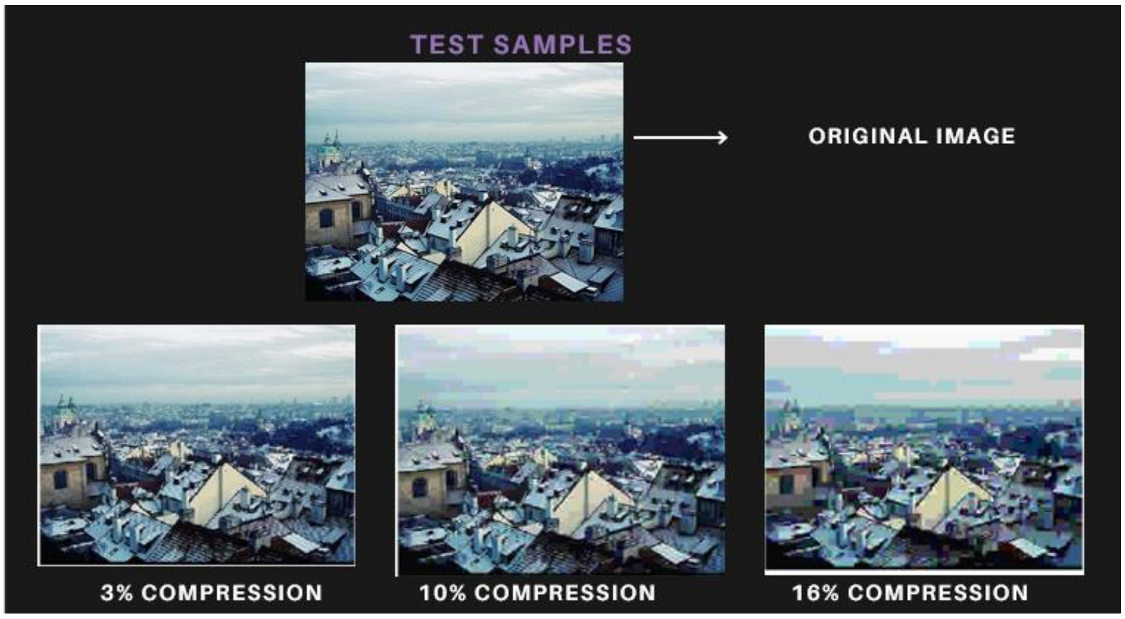
```
encoded1 = 1×11358
      6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6  ...

encoded2 = 1×11423
      6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6  ...

encoded3 = 1×11318
      6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6   -100     6  ...
```

Size of the encoded image = (11358+11423+11318)*8

= 272792 bits



TEST SAMPLE FOR UNEVEN SIZE(300 X 400Px) AND PIXELS NOT MULTIPLE OF 8:



REFERENCES:

- https://youtu.be/Kv1Hiv3ox8I?si=JXP-TdidhbxpKsXl
- https://www.slideserve.com/tanner-humphrey/dct
- https://youtu.be/42IJJKfRKrM
- https://youtu.be/X5LXecsGYIc?si=brg6QqCWW0i8SDka
- https://youtu.be/H54z3-2KuPA?si=D91kSH-OsyTl7X1t
- https://www.javatpoint.com/jpeg-compression

GROUP MEMBERS:

EE21B001-A.VINODH

EE21B004-A.JAYA PRAKASH

THANK YOU