AMRITA VISHWA VIDYAAPEETHAM

ETTIMADAI, COIMBATORE

IMAGE COMPRESSION

DEPARTMENT – CSE-AI

COURSE – MATHEMATICS FOR INTELLIGENT SYSTEMS

SEMESTER – 1

INSTRUCTOR – Dr. K.P SOMAN

SERIAL NUMBER	**NAME**	**ROLL NUMBER**
1	MANGAMURU SAI RISHITH REDDY	CB.EN.U4AIE20036
2	M KARTHIKEYAN	CB.EN.U4AIE20029

PROJECT SUBMITTED FOR THE END SEMESTER EXAMINATION OF 19MAT105

ON 25. 02.2021

EXTERNAL EXAMINER INTERNAL EXAM

ACKNOWLEDGEMENT

We would like to thank to our professor Dr. K.P. Soman who gave us his valuable suggestions and ideas when we were in need of them to work on our project "Image Compression".
We are also grateful to our university, Amrita Vishwa Vidyapeetham for giving us the opportunity to work on this project. Last but not the least we would like to thank our group Mangamuru Sai Rishith Reddy and M Karthikeyan as without their effort and co-operation, we would not have been able to complete the project within the prescribed time.

INDEX

AIM

To understand and implement Image compression using Discrete Cosine Theorem

ABSTRACT

Image compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. T: The processing of digital images took a wide importance in the knowledge field in the last decades ago due to the rapid development in the communication techniques and the need to find and develop methods assist in enhancing and exploiting the image information. The field of digital images compression becomes an important field of digital images processing fields due to the need to exploit the available storage space as much as possible and reduce the time required to transmit the image

INTRODUCTION:

As our use of and reliance on computers continues to grow, so too does our need for efficient ways of storing large amounts of data. For example, someone with a web page or online catalog – that uses dozens or perhaps hundreds of images will more than likely need to use some form of image compression to store those images. This is because the amount of space required to hold unadulterated images can be prohibitively large in terms of cost. These fall into two general categories: lossless and lossy image compression. The JPEG process is a widely used form of lossy image compression that centers around the Discrete Cosine Transform.

The DCT works by separating images into parts of differing frequencies. During a step called quantization, where part of compression actually occurs, the less important frequencies are discarded, hence the use of the term ”lossy.” Then, only the most important frequencies that remain are used retrieve the image in the decompression process. As a result, reconstructed images contain some distortion; but as we shall soon see, these levels of distortion can be adjusted during the compression stage.

PROCEDURE

The following is a general overview of the image compression process. Later, we will be explaining the detailed explanation of it works mainly focusing each step by step which are crucial in the compression stages

The image is broken into 8x8 blocks of pixels.

Working from left to right, top to bottom, the DCT is applied to each block.

Each block is compressed through quantization.

The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space.

When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform (IDCT).

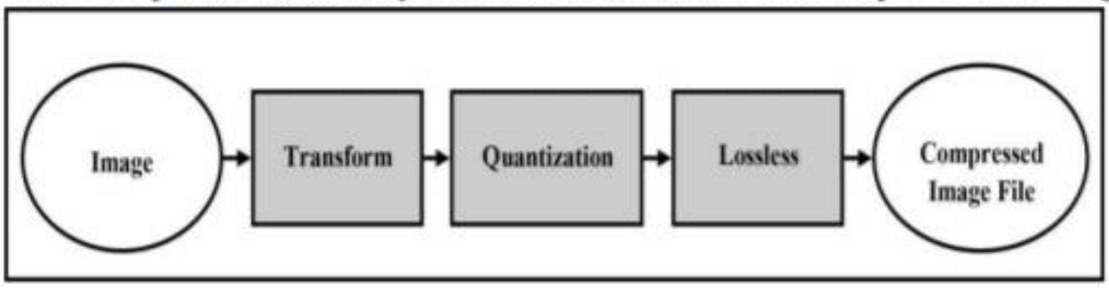


Figure (1): A Typical Image Compression System

COMPONENTS OF IMAGE COMPRESSION SYSTEM

(a) Source Encoder (or Linear Transformer) It is aimed at decorrelating the input signal by transforming its representation in which the set of data values is sparse, thereby compacting the information content of the signal into smaller number of coefficients. a variety of linear transforms have been developed such as Discrete Cosine Transform (DCT).

(b) Quantizer- A quantizer aims at reducing the number of bits needed to store transformed coefficients by reducing the precision of those values. Quantization performs on each individual coefficient i.e. Scalar Quantization (SQ) or it performs on a group of coefficients together i.e. Vector Quantization (VQ).

(c) Entropy Coding Entropy encoding removes redundancy by removing repeated bit patterns in the output of the Quantizer. The common entropy coders we have used in the project is the Huffman Coding.

CONCEPTS USED IN THE COMPRESSION CODE

DCT EQUATION

The DCT equation (Eq. 1) computes the i,j th entry of the DCT of an image.

$$D(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left[\frac{(2x+1)i\pi}{2N}\right] \cos\left[\frac{(2y+1)j\pi}{2N}\right] \qquad 1$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u = 0 \\ 1 & \text{if } u > 0 \end{cases} \qquad 2$$

p(x, y) is the x,yth element of the image represented by the matrix p. N is the size of the block that the DCT is done on. The equation calculates one entry (i,jth) of the transformed image from the pixel values of the original image matrix.

DISCRETE COSINE TRANSFORM

The next step is to divide the three colour components of the image into many 8×8 blocks. For an 8-bit image, in the original block each element falls in the range [0,255]. Data range that is centred around zero is produced after subtracting The mid-point of the range (the value 128) from each element in the original block, so that the modified range is shifted from[0,255] to [-128,127]. Images are separated into parts of different frequencies by the DCT. The quantization step discards less important frequencies and the decompression step uses the important frequencies to retrieve the image.

This equation gives **the forward 2D_DCT transformation:**

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

for $u = 0,\ldots,N-1$ and $v = 0,\ldots,N-1$ (5)

where $N = 8$ and $C(k) = \begin{cases} 1/\sqrt{2} & \text{for } k = 0 \\ 1 & \text{otherwise} \end{cases}$

This equation gives **the inverse 2D_DCT transformation:**

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \qquad (6)$$

for $x = 0,\ldots,N-1$ and $y = 0,\ldots,N-1$ where $N = 8$

After DCT transformation, the “DC coefficient” is the element in the upper most left corresponding to (0,0) and the rest coefficients are called “AC coefficients

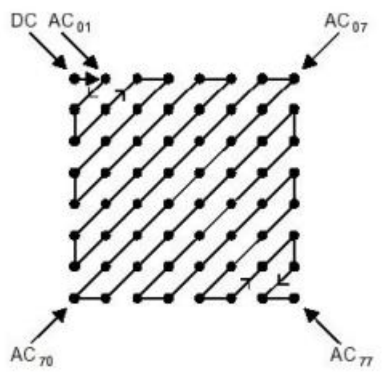
QUANTIZATION

We actually throw away data through the Quantization step. We obtain the Quantization by dividing transformed image DCT matrix by the quantization matrix used . Values of the resultant matrix are then rounded off. Quantization aims at reducing most of the less important high frequency DCT coefficients to zero, the more zeros the better the image will compress. Lower frequencies are used to reconstruct the image because human eye is more sensitive to them and higher frequencies are discarded.

$$F(u,v)_{Quantization} = round\left(\frac{F(u,v)}{Q(u,v)}\right)$$

ZIG ZAG SCANNING

After quantization, the "zig-zag" scanning is done. In the "zig-zag" sequence, firstly it encodes the coefficients with lower frequencies (typically with higher values) and then the higher frequencies (typically zero or almost zero). The result is an extended sequence of similar data bytes, permitting efficient entropy encoding. It will covert two dimensional coefficient block to one dimensional coffeicents.to generate long runs of zeros



HUFFMANN CODING

Huffman coding is an entropy encoding algorithm used for lossless data compression. It encodes a source symbol into variable-length code which is derived in a particular way based on the estimated probability of occurrence of the source symbol.

1. Start with a list of symbols and their frequency in the alphabet.

2. Select two symbols with the lowest frequency.

3. Add their frequencies and reduce the symbols.

4. Repeat the process starting from step-2 until only two values remain.

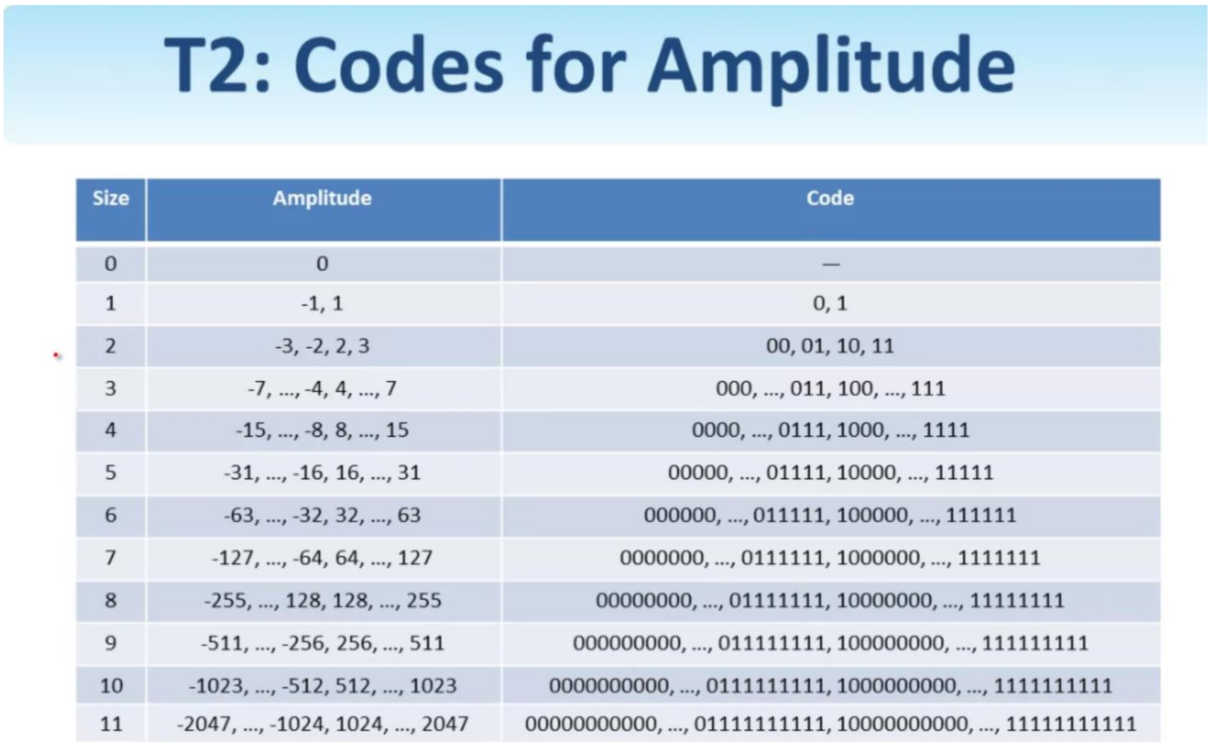
5. The algorithm creates a prefix code for each symbol from the alphabet simply by traversing the symbols back. It assigns 0 and 1 for each frequency value in each phase .

T1 values for dc hauffman coding

size	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	11110

7	5	111110
8	6	1111110
9	7	111111110
10	8	1111111110
11	9	11111111110

T2 : Codes for Amplitude



T2: Codes for Amplitude

Size	Amplitude	Code
0	0	—
1	-1, 1	0, 1
2	-3, -2, 2, 3	00, 01, 10, 11
3	-7, ..., -4, 4, ..., 7	000, ..., 011, 100, ..., 111
4	-15, ..., -8, 8, ..., 15	0000, ..., 0111, 1000, ..., 1111
5	-31, ..., -16, 16, ..., 31	00000, ..., 01111, 10000, ..., 11111
6	-63, ..., -32, 32, ..., 63	000000, ..., 011111, 100000, ..., 111111
7	-127, ..., -64, 64, ..., 127	0000000, ..., 0111111, 1000000, ..., 1111111
8	-255, ..., 128, 128, ..., 255	00000000, ..., 01111111, 10000000, ..., 11111111
9	-511, ..., -256, 256, ..., 511	000000000, ..., 011111111, 100000000, ..., 111111111
10	-1023, ..., -512, 512, ..., 1023	0000000000, ..., 0111111111, 1000000000, ..., 1111111111
11	-2047, ..., -1024, 1024, ..., 2047	00000000000, ..., 01111111111, 10000000000, ..., 11111111111

Huffmann Coding : AC Components

AC components are coded in two parts:

0Part 1(Run Length/ Size)

- Run Length : The length of the consecutive zero values[0,..1,5]
- Size: The number of bits needed to code the next nonzero AC component's value.[0,-A]
- (0,0) is the End of Block for the 8x8 block
- Part 1 Table

Run size	Code length	Code

0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

Part2: (Value)

- Value: is the actual value of the AC component

Run/Size	Code Length	Code
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	1111110110
1/6	16	111111110000100
1/7	16	111111110000101
1/8	16	111111110000110
1/9	`16	111111110000111
1/A	16	111111110001000

DECOMPRESSION

The compression phase is reversed in the decompression process, and in the opposite order. The first step is restoring the Huffman tables from the image and decompressing the Huffman tokens in the image. Next, the DCT values for each block will be the first things needed to decompress a block. The other 63 values in each block are decompressed by JPEG, filling in the appropriate number of zeros. The last step is combined of decoding the zigzag order and recreating the 8 x 8 blocks .The inverse DCT(IDCT) takes each value in the spatial domain and examines the contributions that each of the 64 frequency values make to that pixel. Reconstruction of our image begins by decoding the bit stream representing the quantized matrix C. Each element of C is then multiplied by the corresponding element of the quantization matrix originally used.

$$R_{i,j} = Q_{i,j} \times C_{i,j}$$

The IDCT is next applied to matrix R, which is rounded to the nearest integer. Finally, 128 is added to each element of that result, giving us the decompressed version N of our original 8x8 image block M.

$$N = \text{round}(T' R T) + 128$$

MATLAB CODE

Image Reading

```
clc;

clear all;

close all;

[file, path]=uigetfile('*.*');

I=imread(file);

figure;imshow(I);title('Original Image')

q=1;

OriginalImage=I;Q=q;

OriginalImage=double(OriginalImage);

ImageSub=OriginalImage-128;

[Row,Col]=size(OriginalImage);

BlockNumber=Row*Col/64;
```

Block Processing and applying DCT

```
Coef=blkproc(ImageSub,[8,8],'dct2(x)');
```

Quantization Matrix

```
L=Q*[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];


[CoefDCchanged,CoefAfterQ]=Quntization_pro(L,Row,Col,Coef,BlockNumber);
```

ZigZag Scanning

```
ImageBitSeq=[];

ImageBitLen=[];

rowloop=0;

for row=1:Row/8

    colloop=0;

    for col=1:Col/8

       m(1:8,1:8)=CoefDCchanged((row-1)*8+1:(row-1)*8+8,(col-1)*8+1:(col-
1)*8+8);

       k= round(m);
```

ZigZag Scaning

```
       %%*********************************************************

       t=zigzag(k);
```

Removing Extra Zeros

```
       %%*********************************************************

       w=0;

       u=64;

       while u ~= 0

             if t(u) ~= 0

                w=u;

                break;

             end
```

```
            u=u-1;
      end
      w;
      if w==0
         w=1;
      end
      e(w)=0;
      for i=1:w
          e(i)=t(i);
      end
      %e;
```

Entropy Encoding with Huffman

```
      [blockDCbit_seq,blockDCcode_len]=DCHuffmanEncoding(e(1));
      blockDCbit_seq;
      blockDCcode_len;
      eob_seq=dec2bin(10,4);
      blockACbit_seq=[];
      blockbit_seq=[];
      zrl_seq=[];
      trt_seq=[];
      zerolen=0;
      zeronumber=0;

      if numel(e)==1
         blockACbit_seq=[];
         blockbit_seq=[blockDCbit_seq,eob_seq];
         blockbit_len=length(blockbit_seq);
      else
         for i=2:w
             if ( e(i)==0 & zeronumber<16)
                 zeronumber=zeronumber+1;
             elseif (e(i)==0 & zeronumber==16);
```

```
                    bit_seq=dec2bin(2041,11);
                    zeronumber=1;
                    blockACbit_seq=[blockACbit_seq,bit_seq];
                elseif (e(i)~=0 & zeronumber==16)
                    zrl_seq=dec2bin(2041,11);
                    amplitude=e(i);
                    trt_seq=ACHuffmanEncoding(0,amplitude);
                    bit_seq=[zrl_seq,trt_seq];
                    blockACbit_seq=[blockACbit_seq,bit_seq];
                    zeronumber=0;
                elseif(e(i))
                    zerolen=zeronumber;
                    amplitude=e(i);
                    zeronumber=0;
                    bit_seq=ACHuffmanEncoding(zerolen,amplitude);
                    blockACbit_seq=[blockACbit_seq,bit_seq];
                end
            end
        end
        blockbit_seq=[blockDCbit_seq,blockACbit_seq,eob_seq];
        blockbit_len=length(blockbit_seq);
        blockbit_seq;
        blockbit_len;
        ImageBitSeq=[ImageBitSeq,blockbit_seq];
        ImageBitLen=numel(ImageBitSeq);
        colloop=colloop+1;
    end
    rowloop=rowloop+1;
end
ImageBitSeq;
ImageBitLen;
AverageBit=ImageBitLen/Row/Col;
```

Parameter Calculation

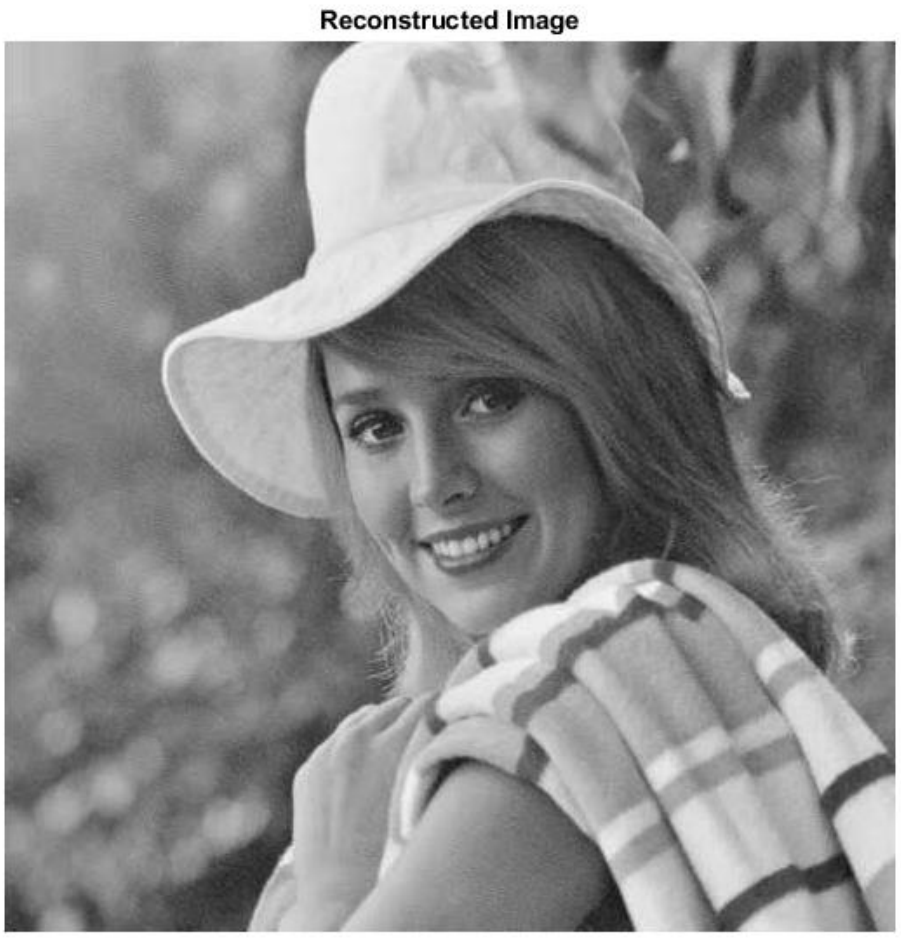
```
CoefInverseQ=blkproc(CoefAfterQ,[8,8],'x.*P1',L);

ImageSubRecon=blkproc(CoefInverseQ,[8,8],'idct2(x)');

ReconImage=round(ImageSubRecon)+128;

CR=Row*Col*8/ImageBitLen;

PSNR = PSNR(ReconImage,OriginalImage);
```

Output Paramters

```
figure;imshow(uint8(ReconImage));title('Reconstructed Image')

fprintf('Compression ratio is = %f\n',ceil(CR))

fprintf('Peak Signal to Noise ratio is = %f\n',ceil(PSNR))

fprintf('Average Bit per Pixel = %f\n',AverageBit)

imwrite(uint8(ReconImage),'rec.jpg')
```

Outputs

Original Image



```
Compression ratio is = 8.000000
Peak Signal to Noise ratio is = 34.000000
Average Bit per Pixel = 1.020550
```

CONCLUSION

Compression can be achieved by using DCT techniques that divide the image into different frequency components. The unnecessary information can then be removed from the image by quantization. This means that DCT plays a vital role in JPEG image compression. As the compression ratio is getting bigger and bigger, more and more information. Therefore, the need to introduce high efficiency DCT algorithm to achieve better image compression.