Contents

1. Introduction

With the ever increasing dependence on computers, the need for efficient storage of large amounts of data continues to grow. For example, someone with a web page that uses a large number of images will more likely need some sort of image compression to store and transmit images over the internet. This is needed because the amount of space required to transmit or store such unadulterated images is quite large. So, image compression needs to be employed in order to make efficient use of the digital information by reducing the irrelevancy and redundancy of the image data.

The two fundamental components of compression are redundancy and irrelevancy reduction. "**Redundancy reduction"** is aimed at removing duplication from the signal source (image/video) while "**Irrelevancy reduction"** omits parts of the signal that will not be noticed by the signal receiver, namely the **Human Visual System** (HVS).

Image Compression can be of two types: "**Lossy"** or "**Lossless"**. Lossless compression means that the exact original data can be reconstructed from the compressed data. Image quality is not reduced when using lossless compression. Unlike lossless compression, lossy compression reduces the image quality. The exact original image cannot recovered back after using lossy compression methods since some information is always lost. In order to have a good compression ratio without losing too much of information when the image is decompressed we use **Discrete Cosine Transform** (DCT).

Discrete Cosine Transform

The fundamental idea behind any picture compression is that the numeric values stored in a picture matrix can be transformed from one basis to another, where the new basis stores the relevant information in a more compact form.

The original basis is "**2-D special basis**" – every entry in picture matrix represents an actual square pixel which has a special position in the picture. The basis that would store the image values more compactly is the **frequency basis**, where frequency represents "changes in the value of luminosity". Higher frequency represents quick changes in luminosity from pixel to pixel and lower frequency represents gradual changes across the entire picture.

The definition of the two-dimensional forward DCT for an input matrix A and output matrix B is

$$B(p,q) = \alpha(p)\alpha(q)\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} A(m,n)\cos\left[\frac{\pi(2m+1)p}{2M}\right]\cos\left[\frac{\pi(2n+1)q}{2N}\right],$$

$$0 \le p \le M-1 \qquad \text{and} \qquad 0 \le q \le N-1$$

where $\alpha(p) = \begin{cases} \frac{1}{\sqrt{M}}, p = 0 \\ \sqrt{\frac{2}{M}},\ 1 \le p \le\ M-1 \end{cases}$ and $\alpha(q) = \begin{cases} \frac{1}{\sqrt{N}}, q = 0 \\ \sqrt{\frac{2}{N}},\ 1 \le q \le\ N-1 \end{cases}$

M and N are the row and column size of A, respectively. The two-dimensional transform is equivalent to a one-dimensional DCT performed along a single dimension followed by a one-dimensional DCT in the other dimension.

Properties of 2-D DCT:

De-correlation

The main advantage of image transformation is the removal of redundancy between neighbouring pixels which leads to uncorrelated transform coefficients that can be encoded independently.

Energy Compaction

Efficacy of a transformation scheme can be directly gauged by its ability to pack input data into as few coefficients as possible. This allows the quantizer to discard coefficients with relatively small amplitudes without introducing visual distortion in the reconstructed image. DCT exhibits excellent energy compaction for highly correlated images.

Separability

The DCT transform equation can be expressed as

$$B(p,q) = \alpha(p)\alpha(q) \sum_{m=0}^{M-1} \cos\left[\frac{\pi(2m+1)p}{2M}\right] \sum_{n=0}^{N-1} A(m,n) \cos\left[\frac{\pi(2n+1)q}{2N}\right]$$

This property, known as separability, has the principle advantage that B (p, q) can be computed in two steps by successive 1-D operations on rows and columns of an image.

The definition of the two-dimensional Inverse DCT for an input matrix A and output matrix B is

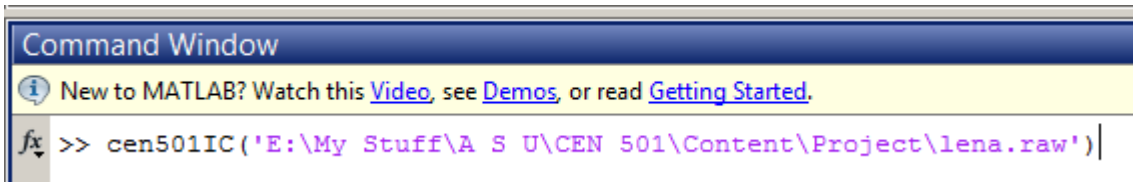
$$A(m,n) = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} \alpha(p)\alpha(q)B(p,q) \cos\left[\frac{\pi(2m+1)p}{2M}\right] \cos\left[\frac{\pi(2n+1)q}{2N}\right],$$

$0 \le m \le\ M-1$ and $0 \le n \le\ N-1$

where $\alpha(p) = \begin{cases} \frac{1}{\sqrt{M}}, p = 0 \\ \sqrt{\frac{2}{M}},\ 1 \le p \le\ M-1 \end{cases}$ and $\alpha(q) = \begin{cases} \frac{1}{\sqrt{N}}, q = 0 \\ \sqrt{\frac{2}{N}},\ 1 \le q \le\ N-1 \end{cases}$

2. Implementation Details of DCT Image Compression

A MATLAB program named **cen501IC.m** has been created which accepts a 512 X 512 RAW Image from the Command Window.



Steps involved in Image Compression is as follows:

2.1. Accept the RAW Image as input

2.2. Perform **Level Translation** by subtracting 128 from each and every pixel of the image. This is done since DCT is designed to work on pixel values ranging from -128 to 127.

2.3. Image segmentation carried out to divide the image into non-overlapping 8 X 8 pixel blocks. 2-D DCT is applied on the image to obtain the **DCT co-efficient matrix**. This is achieved through the custom 2-D DCT Matlab function: **fDct2** which distributes the frequencies of the image. The low frequencies reside in the upper-left corner of the spectrum, while the high frequencies are in the lower right, as illustrated in figure 2.1. The DC component is at position [0, 0], the upper-leftmost value. Rest of the components are known as AC Coefficient.

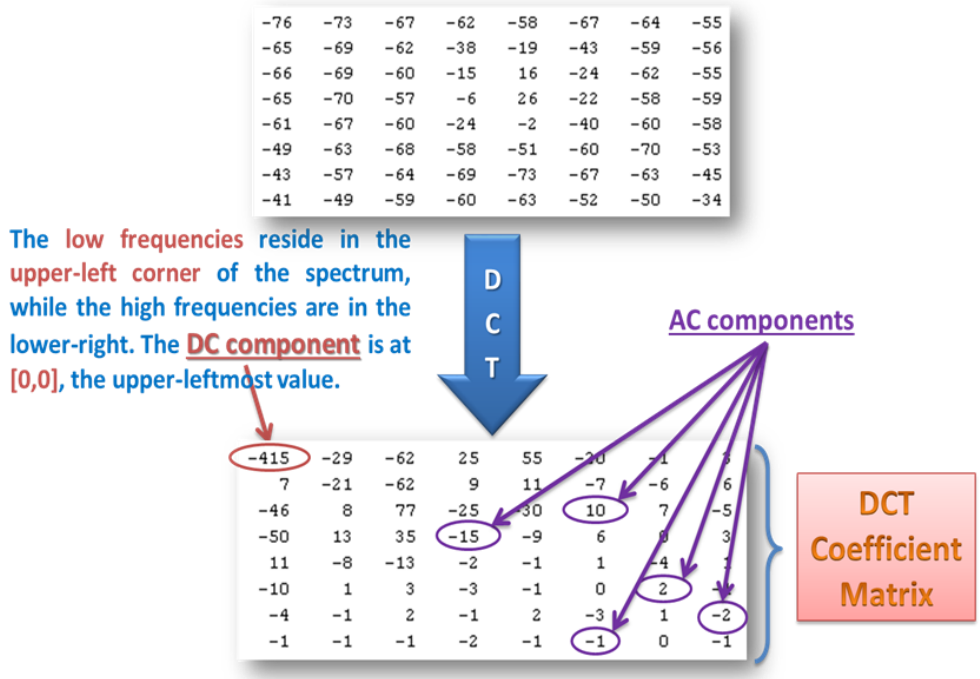


Figure 2.1: Application of DCT in level translated matrix.

2.4. Perform masking on the DCT co-efficient matrix in order to obtain as many numbers of zeros as possible without losing much image quality. Here, we are using three different types of masks in order to analyse the image compression results for varying amounts of zeros.

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Mask1 **Mask2** **Mask3**

The bottom lower one quarter of the mask 1, half of mask 2 and three quarter of mask 3 have been set to zero. The more important lower frequency components are retained as "1" in the mask. This is done in view of the fact that the human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This allows reducing the amount of information in high frequency components by setting them to zero. The Matlab custom function: **fMask** is used to obtain the masked image in each of the three scenarios by multiplying each element of the mask with corresponding 8X8 elements in the DCT co-efficient matrix.

2.5. The Matlab custom function: **fcalMeanSdStepsize** is used to compute the Mean and Standard Deviation of the pixels of the masked image by grouping the similar frequencies of the 8X8 blocks together. Here, the input matrix of dimension 512 X 512 is translated to a 64 X 4096 matrix where the row corresponds to similar frequencies and columns correspond to the 64 elements of each 8 X 8 blocks.

$$\begin{bmatrix} DC1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \begin{bmatrix} DC2 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \dots \begin{bmatrix} DC64 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$

$$\begin{bmatrix} DC65 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \begin{bmatrix} DC66 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \dots \begin{bmatrix} DC128 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\begin{bmatrix} DC449 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix} \quad \dots \quad \dots \quad \begin{bmatrix} DC512 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 32 & 33 & 34 & 35 & 36 & 37 & 38 & 39 \\ 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 \\ 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 \end{bmatrix}$$

Masked Image 1 (512 X 512)

$$\begin{bmatrix} DC1 & \dots & \dots & \cdots & \cdots & \cdots & \dots & \dots & DC4096 \\ 1 & 1 & 1 & \cdots & \cdots & \cdots & \dots & \dots & 1 \\ 2 & 2 & 2 & \cdots & \cdots & \cdots & \dots & \dots & 2 \\ 3 & 3 & 3 & \cdots & \cdots & \cdots & \dots & \dots & 3 \\ 4 & 4 & 4 & \cdots & \cdots & \cdots & \dots & \dots & 4 \\ \dots & \dots & \dots & \cdots & \cdots & \cdots & \dots & \dots & \dots \\ 61 & 61 & 61 & \cdots & \cdots & \cdots & 61 & 61 & 61 \\ 62 & 62 & 62 & \cdots & \cdots & \cdots & 62 & 62 & 62 \\ 63 & 63 & 63 & \cdots & \cdots & \cdots & 63 & 63 & 63 \end{bmatrix}$$

Dimensions (64 X 4096)

$$Mean(i) = \frac{\sum_{j=0}^{4096} Pixel(i,j)}{4096}$$

$$Standard\ Deviation(i) = Stanadard\ Deviation\ of\ Pixels\ in\ (i)th\ row$$

where i = 1 to 64 (= number of rows).

Since, a 10-bit uniform scalar quantizer is used, for each of the 64 elements in the 8 X 8 block, the step size is calculated as follows:

$$Step\ size(i) = \frac{4 * Standard\ Deviation(i)}{2\text{^}10}$$

2.6. Carry out normalization and quantization by using the custom Matlab function: **fNormQuantize**

$$Normalized\ pixel = \frac{Pixel - Mean}{Standard\ Deviation}$$

$$Quantized\ pixel = \frac{Normalized\ pixel}{Step\ size}$$

2.7. The custom Matlab function: **fCompress** compresses the image by considering only the non-zero elements of the image which then can be stored or transmitted over a medium.

2.8. The custom Matlab function: **fDecompress** decompresses the image by carrying out de-quantization and de-normalization.

$$Decompressed\ pixel = (Quantized\ pixel * SD * Step\ size) + Mean$$

2.9. The custom Matlab function: **fIDct2** performs 2-D Inverse DCT after which 128 is added to each and every pixel in order to recover the final image.

Image Analysis

- The custom Matlab function: **fCalTotAvgBits** calculates the Total number of bits and Average number of bits/pixel based on the below formulas:

$$Total\ number\ of\ bits = Number\ of\ nonzero\ pixels * 8$$

$$Average\ number\ of\ bits/pixel = \frac{Total\ number\ of\ bits}{512 * 512}$$

- The custom Matlab function: **fCalPsnr** calculates the peak signal-to-noise ratio (PSNR) via the mean square error (MSE) as follows:

$$MSE = \frac{1}{M * N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}[I(i,j) - K(i,j)]^2$$

$$PSNR = 10 * \log_{10}\left[\frac{MAX_I^2}{MSE}\right]$$

Here, M and N are dimension of the image and MAX_I is the maximum pixel value of the original image.

- The custom Matlab function: **fDft2** calculates the 2-D DFT magnitude spectrum of the reconstructed image by using 1-D FFT.

MATLAB Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   Topic: 2-D Discrete Cosine Transform Image Compression
%   CEN 501 - Fall 2014
%   Arizona State University
%   by: SOURAV SAMANTA
%   ASU Id: 1207860455
%   Email: ssamant4@asu.edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function cen501IC(enterCompletePath_withinSingleQuotes)
clc
close all

%enterCompletePath_withinSingleQuotes = 'E:\My Stuff\A S U\CEN
501\Content\Project\lena.raw';
%Input the 512 X 512 .RAW image
rawImage = fopen(enterCompletePath_withinSingleQuotes,'r');
orgImage = fread(rawImage);
fclose(rawImage);
orgImage = reshape(orgImage,512,512);
orgImage = orgImage.';
```

```
%Perform Level Translation (Pixel = Pixel - 128)
lvlTransImage = orgImage - 128;

%Divide the Image into 8 X 8 blocks and apply custom 2-D DCT 'fDct2'
%function to obtain DCT Coefficient Matrix
fun = @(block_struct) fDct2(block_struct.data);
dctCoeffMat = blockproc(lvlTransImage,[8 8],fun);

%Mask-1 (Setting lower one-quarter of the Mask to zero)
mask1 = [1 1 1 1 1 1 1 1
         1 1 1 1 1 1 1 1
         1 1 1 1 1 1 1 1
         1 1 1 1 1 1 1 0
         1 1 1 1 1 1 0 0
         1 1 1 1 0 0 0 0
         1 1 1 1 0 0 0 0
         1 1 1 0 0 0 0 0];

%Mask-2 (Setting lower one-half of the Mask to zero)
mask2 = [1 1 1 1 1 1 1 1
         1 1 1 1 1 1 1 0
         1 1 1 1 1 0 0 0
         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 1 0 0 0 0 0 0
         1 0 0 0 0 0 0 0];

%Mask-3 (Setting three-fourth of the Mask to zero)
mask3 = [1 1 1 1 1 0 0 0
         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
         1 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0
         0 0 0 0 0 0 0 0];

%Perfom Masking using 'fMask' function
maskedImage1 = fMask(dctCoeffMat,mask1);
maskedImage2 = fMask(dctCoeffMat,mask2);
maskedImage3 = fMask(dctCoeffMat,mask3);

%Compress Masked Images using 'fCompress' function
[comImage1,mean8x8a,std8x8a,step8x8a] = fCompress(maskedImage1);
[comImage2,mean8x8b,std8x8b,step8x8b] = fCompress(maskedImage2);
[comImage3,mean8x8c,std8x8c,step8x8c] = fCompress(maskedImage3);

%Decompress Images using 'fDecompress' function
recImage1 = fDecompress(comImage1,mean8x8a,std8x8a,step8x8a);
recImage2 = fDecompress(comImage2,mean8x8b,std8x8b,step8x8b);
recImage3 = fDecompress(comImage3,mean8x8c,std8x8c,step8x8c);

%Calculate the Magnitude Spectrum of Images by 2-D DFT using 'fDft2'
%function
dft2OrgImage = fDft2(orgImage);
```

```
dft2RecImage1 = fDft2(recImage1);
dft2RecImage2 = fDft2(recImage2);
dft2RecImage3 = fDft2(recImage3);

%Calculate Total no. of bits to code the Image & Average number of
%bits/pixel
[totBits1,avgBitsPerPixel1] = fCalTotAvgBits(comImage1);
[totBits2,avgBitsPerPixel2] = fCalTotAvgBits(comImage2);
[totBits3,avgBitsPerPixel3] = fCalTotAvgBits(comImage3);

%Calculate the PSNR of the Images
psnrVal1 = fCalPsnr(recImage1,orgImage);
psnrVal2 = fCalPsnr(recImage2,orgImage);
psnrVal3 = fCalPsnr(recImage3,orgImage);

%Display Data
set(gcf,'Position',get(0,'Screensize'));

subplot(2,4,1);
subimage(mat2gray(orgImage));
t1 = title({'Original Image';['Total no. of Bits =
',num2str(512*512*8)];['Average no. of bits/pixel = ',num2str(8)]});
set(t1,'FontSize',10);
axis off;
subplot(2,4,2);
subimage(mat2gray(recImage1));
t2 = title({'Reconstructed Image using One-Quarter Zero mask';['PSNR =
',num2str(psnrVal1),' dB & Total no. of Bits = ',num2str(totBits1)];['Average
no. of bits/pixel = ',num2str(avgBitsPerPixel1)]});
set(t2,'FontSize',9);
axis off;
subplot(2,4,3);
subimage(mat2gray(recImage2));
t3 = title({'Reconstructed Image using Half Zero mask';['PSNR =
',num2str(psnrVal2),' dB & Total no. of Bits = ',num2str(totBits2)];['Average
no. of bits/pixel = ',num2str(avgBitsPerPixel2)]});
set(t3,'FontSize',9);
axis off;
subplot(2,4,4);
subimage(mat2gray(recImage3));
t4 = title({'Reconstructed Image using Three-Quarter Zero mask';['PSNR =
',num2str(psnrVal3),' dB & Total no. of Bits = ',num2str(totBits3)];['Average
no. of bits/pixel = ',num2str(avgBitsPerPixel3)]});
set(t4,'FontSize',9);
axis off;
subplot(2,4,5);
imagesc(dft2OrgImage);
colormap(gray);
title({'Magnitude Spectrum of';'Original Image'});
subplot(2,4,6);
imagesc(dft2RecImage1);
colormap(gray);
title({'Magnitude Spectrum of Reconstructed Image';'using One-Quarter Zero
mask'});
subplot(2,4,7);
imagesc(dft2RecImage2);
colormap(gray);
```

```
title({'Magnitude Spectrum of Reconstructed Image';'using Half Zero mask'});
subplot(2,4,8);
imagesc(dft2RecImage3);
colormap(gray);
title({'Magnitude Spectrum of Reconstructed Image';'using Three-Quarter Zero
mask'});

end

%Function for Masking
function Y = fMask(X,mask)

    fun = @(block_struct) block_struct.data .* mask;
    Y = blockproc(X,[8 8],fun);

end


%Function to Compress an Image
function [Y,mean8x8,std8x8,step8x8] = fCompress(X)

    %Calculate Mean, Standard Deviation & Step-size
    [mean8x8,std8x8,step8x8] = fcalMeanSdStepsize(X);

    %Normalize & Quantize
    fun = @(block_struct)
fNormQuantize(block_struct.data,mean8x8,std8x8,step8x8);
    Y = blockproc(X,[8 8],fun);

end


%Function to Decompress an Image
function Y = fDecompress(X,mean8x8,std8x8,step8x8)

    %De-normalize & De-quantize
    fun1 = @(block_struct) ((block_struct.data .* std8x8 .* step8x8) +
mean8x8);
    temp1Y = blockproc(X,[8 8],fun1);

    %Perform 2-D Inverse DCT using custom 'fIDct2' function
    fun2 = @(block_struct) fIDct2(block_struct.data);
    temp2Y = blockproc(temp1Y,[8 8],fun2);

    %Level Translation + 128
    Y = temp2Y + 128;

end

%2-D DCT function
function Y = fDct2(X)

    %Determine the dimensions of the Image
    [M,N] = size(X);
    Y = zeros(size(X));

    for p = 0:M-1
```

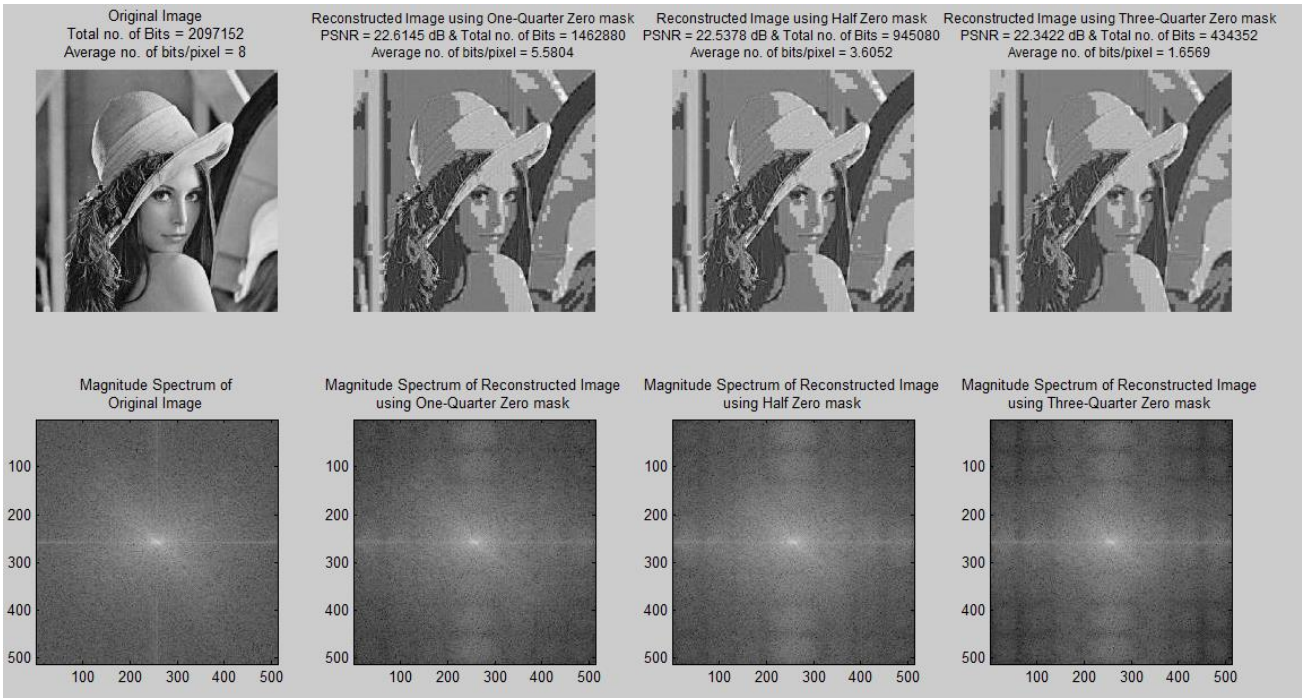
```
        if (p == 0)
            Ap = 1/sqrt(M);
        else
            Ap = sqrt(2/M);
        end
        for q = 0:N-1
            sum = 0;
            if (q == 0)
                Aq = 1/sqrt(N);
            else
                Aq = sqrt(2/N);
            end
            for m = 1:M
                for n = 1:N
                    sum = sum + (X(m,n) * cos(pi*(2*m-1)*p/(2*M)) *
cos(pi*(2*n-1)*q/(2*N)));
                end
            end
            Y(p+1,q+1) = Ap * Aq * sum;
        end
    end
end


%2-D Inverse DCT function
function Y = fIDct2(X)

    %Determine the dimensions of the Image
    [M,N] = size(X);
    Y = zeros(size(X));

    for m = 0:M-1
        for n = 0:N-1
            sum = 0;
            for p = 0:M-1
                if (p == 0)
                    Ap = 1/sqrt(M);
                else
                    Ap = sqrt(2/M);
                end
                for q = 0:N-1
                    if (q == 0)
                        Aq = 1/sqrt(N);
                    else
                        Aq = sqrt(2/N);
                    end
                    sum = sum + (Ap * Aq * X(p+1,q+1) *
cos(pi*(2*m+1)*p/(2*M)) * cos(pi*(2*n+1)*q/(2*N)));
                end
            end
            Y(m+1,n+1) = sum;
        end
    end
end

%Function to calculate Mean, Standard Deviation & Step-size
function [mean8x8,std8x8,step8x8] = fcalMeanSdStepsize(X)
```

```
    [M,N] = size(X);
    m = 8;      %No. of rows in the block (=8 for 8X8 block)
    n = 8;      %No. of columns in the block (=8 for 8X8 block)
    if (M == N)
        %Re-arrange the frequency components of the Image by grouping the
        %equivalent frequencies together i.e. DC component (lowest
        %frequency) from each block arranged in the 1st row followed by
        %AC components (high frequencies) in the 2nd row and so on.
        %Resulting matrix is 64 X 4096
        for i = 0:(m*n-1)
            fun = @(block_struct) reshape(block_struct.data',[64 1]);
            tempY = blockproc(X(8*i+1:8*i+8,:),[8 8],fun);
            if (i==0)
                Y = tempY;
            else
                Y = horzcat(Y,tempY);
            end
        end

        %Mean, SD & Step-size for each of the 64 elements in the 8X8 block
        mean_x = zeros(64,1);
        std_x = zeros(64,1);
        step_x = zeros(64,1);

        for i = 1:m*n
            mean_x(i,1) = mean(Y(i,:));     %Calculate Mean of 1-D array
            std_x(i,1) = std(Y(i,:));       %Calculate Standard Deviation of
1-D array
            %Calculate Step-size using 10 bits
            step_x(i,1) = 4*std_x(i,1)/(2^10);
        end
        mean8x8 = reshape(mean_x,[8 8])';   %Mean formatted into 8X8 block
        std8x8 = reshape(std_x,[8 8])';     %SD formatted into 8X8 block
        step8x8 = reshape(step_x,[8 8])';   %Step-size formatted into 8X8
block
    else
        disp('Error: Dimensions mis-match');
    end
end

%Function to Normalize and Quantize
function Y = fNormQuantize(X,mean8x8,std8x8,step8x8)

    Y = zeros(8,8);
    for i = 1:8
        for j = 1:8
           if (std8x8(i,j) == 0)
               Y(i,j) = 0;
           else
               Y(i,j) = round((X(i,j)-
mean8x8(i,j))/(step8x8(i,j)*std8x8(i,j)));
           end
        end
    end
end
```

```
%Function to compute 2-D DFT
function Y = fDft2(X)

    %Determine the dimensions of the Image
    [M,N] = size(X);
    Y = zeros(M,N);

    %Compute the 1-D FFT of the Image
    for i = 1:M
        Y(i,:) = fft(X(i,:));
    end

    %Re-compute the 1-D FFT of the Image
    for j = 1:N
        Y(:,j) = fft(Y(:,j));
    end

    %Shift the zero-frequency component to center of spectrum
    tempY = fftshift(Y);

    %Obtain the Magnitude Spectrum
    Y = log(1+abs(tempY));
end

%Function to calculate Total no. of bits to code the Image & Average number
of
%bits/pixel
function [totBits,avgBitsPerPixel] = fCalTotAvgBits(X)

    [M,N] = size(X);
    noOfNonZeroElements = nnz(X);
    totBits = noOfNonZeroElements * 8;
    avgBitsPerPixel = totBits / (M*N);

end

%Function to calculate PSNR
function psnrValue = fCalPsnr(noisyImage,orgImage)

    %Determine the dimensions of the Noisy Image
    [M,N] = size(noisyImage);

    %Compute the difference between Original Image & Noisy Image
    difImage = orgImage - noisyImage;

    %Compute the Mean Square Error
    difImage = difImage .* difImage;
    sumInt = sum(sum(difImage));
    mseVal = sumInt/(M*N);

    %Finding out the maximum value of Pixel in Original Image
    maxPixelOrgImage = max(max(orgImage));

    %Calculate PSNR
    psnrValue = 10*log10((maxPixelOrgImage*maxPixelOrgImage)/mseVal);

end
```

3. Results

Input Image: 512X512 8-bit-per-pixel, raster-scanned "lena.raw" image



The upper four images indicate the Original and Reconstructed images while the lower ones indicate their corresponding 2-D DFT magnitude spectrum. They are obtained after proper shifting of FFT output to ensure that they are centred on the zero frequency (or DC). Since, we have two independent variables for 2-D signals, we have two frequency axes (horizontal frequency for x and vertical frequency for y). Thus, with the Fourier transform, we move from a series of amplitudes over space to amplitudes over frequency.

- **Original Image (uncompressed)**
  Total number of bits = 2097152
  Average number of bits/pixel = 8
- **Reconstructed Image 1** (Triangular lower one-quarter zero mask used)
  PSNR = 22.6145 dB
  Total number of bits required to encode = 1462880
  Average number of bits/pixel = 5.5804
- **Reconstructed Image 2** (Triangular lower half zero mask used)
  PSNR = 22.5378 dB
  Total number of bits required to encode = 945080

Average number of bits/pixel = 3.6052

- **Reconstructed Image 3** (Triangular lower three-quarter zero mask used)
  PSNR = 22.3422 dB
  Total number of bits required to encode = 434352
  Average number of bits/pixel = 1.6569

Inferences from Magnitude Spectrum of the Lena Image

- Most of the "**lena**" image is smooth (major changes occur around the edges). The number of edges is very small as compared to smooth regions of the image. This is reflected in the magnitude spectrum where the center point corresponds to the zero frequency. The region around it is very bright. It indicates that the image is rich in low frequencies (that is it is largely smooth).
- As we move away from the center, the brightness keeps on decreasing. This reflects the fact that there are fewer and fewer high frequencies in the image.
- Along the horizontal, vertical and secondary diagonal, we see a streak of lines which shows that there are major changes along horizontal, vertical and primary diagonal in the image respectively.
- The Fourier magnitude spectrum is symmetric about the origin which corroborates the fact that the magnitude spectrum of a real signal is even.

Inferences from the results of the experiment

- The PSNR value decreases as we move from the reconstructed images 1 to 3. This is because, more number of high frequency components have been made to zero thereby increasing the distortion rate. Hence, the image quality of the reconstructed image 3 is poor compared to the reconstructed image 1.
- The total number of bits required to encode the compressed image 1 is more than the 3rd one since there are lesser number of zeros in the image.

$$Total\ number\ of\ bits\ \propto \frac{1}{Compression\ ratio}$$

- Also, average number of bits/pixel is more in case of the 1st reconstructed image 1 than the 3rd one.
- The 2-D DFT magnitude spectrum of the reconstructed image 3 shows more number of streak of lines along the horizontal, vertical and secondary diagonal as compared to the 1st reconstructed image since more number of high frequency components have been removed from the image.
- Compression ratio is maximum in case of reconstructed image 3 while minimum in case of 1. Thus, we can conclude

$$Image\ Quality\ \propto \frac{1}{Compression\ ratio}$$

4. Conclusions

Discrete Cosine Transform is a widely used compressed technique which is used in signal and image processing, but as it involves a lossy algorithm it leads to blocking artefacts and hence it may not be suitable for suitable for medical applications which require the exact original image.

DCT can still be used for lossy data compression, because it has strong "energy compaction" property. Since, most of the information tends to be concentrated in a few low frequency components of the DCT, the majority of the high frequency ones can be neglected. This would mean a slight degradation of the image quality without much appreciable change visible to the human eye.

In this project, we have seen that the quality of reconstructed images increases with the increase in better quality mask (mask with lesser number of zeros), while the compression ratio decreases.

Also, there are other compression techniques (such as wavelet compression) available which provide a better way to compress and regenerate the image as the compression ratio and quality of the image is better than the DCT compression. There is no need to divide the input coding into 2-D blocks. It provides higher flexibility as various types of wavelet functions can be freely chosen.