Image compression

Rahul Patel (121040)
rahul.patel@iet.ahduni.edu.in

Shashwat Sanghavi (121049)
shashwat.sanghavi@iet.ahduni.edu.in

Institute of Engineering & Technology, Ahmedabad University

October 20, 2015

1 Introduction [1, 2, 3]

Internet is one of the most important medium for data transfer in present age. Utility of the Internet has evolved since it's perception. In it's infancy it was used by applications which consumed minimal data as they were text based. As a matter of contrast, it wouldn't be a hyperbole to state that in major present day applications usage of multimedia data is significantly higher than text. This fact is bolstered by the fact that 90 percentage of world's data is generated in the past two years. Hence to utilize the medium efficiently it is of utmost importance that the data transferred is as small as possible. To achieve this goal various compression model have been devised. A compression model takes the raw input data and represents it in such a form that the space required to store the modified data is less than the original data. Which implies better utilization of resources like storage space and bandwidth while transferring this data. Apart from the Internet there are many applications where data usage is extremely crucial. Hence, it has been researchers focus point since years to develop better compression models.

For an example, let us consider an RGB image of size $1980 * 1080$ pixels. The size occupied by this image will be $1980 * 1080 * 8$(bits for each pixel) $*3$(RGB layers) $= 6.11$

MBs. For storing multiple images over a small storage device or to transfer this image over Internet, it is extremely important compress the image to minimize the size with minimum loss of data.

Taking into consideration the above mentioned points one cannot deny the significance of such models in better utilization of resources. Hence, as a part of this project we tried to study JPEG compression model which is considered to be a milestone in image compression. We also simulated the model, results of which are discussed below.

2 Compression model

Compression model can be classified as lossy compression and lossless compression based on their ability to obtain the original data from the modified data.

1. **Lossy compression**

   In lossy compression the original data cannot be reconstructed completely from the modified data. There is an information loss in the reconstructed data as compared to the original data. This kind of compression is generally used by applications in which data loss does not alter the perceived information content by human beings. Generally, lossy compression gives higher compression ratio as compared to lossless compression. Lossy compression is most commonly used to compress multimedia data like audio, video, and images, especially in applications such as streaming media and internet telephony.

2. **Lossless compression**

   In lossless compression the original data can be reconstructed completely from the modified data. This kind of compression is used for highly information sensitive data. Typical application of lossless compression is text compression, as one would like to reconstruct the original message without altering any letter or word.

This report presents a detailed study and simulated results of JPEG compression model which is a form of lossy image compression. Image compression is mostly lossy as small amount of data loss does not alter the perceived information content of the image. JPEG compression is one of the widely used compression model to as a good amount of compression is obtained without much perceived data loss.

3 JPEG model [4, 5, 6]

Below is the diagram showing overview of the steps followed in JPEG compression. The output of RLE is also subjected to Huffman coding to further reduce the size. Huffman coding is not implemented in this project as it will be beyond the scope of this project.

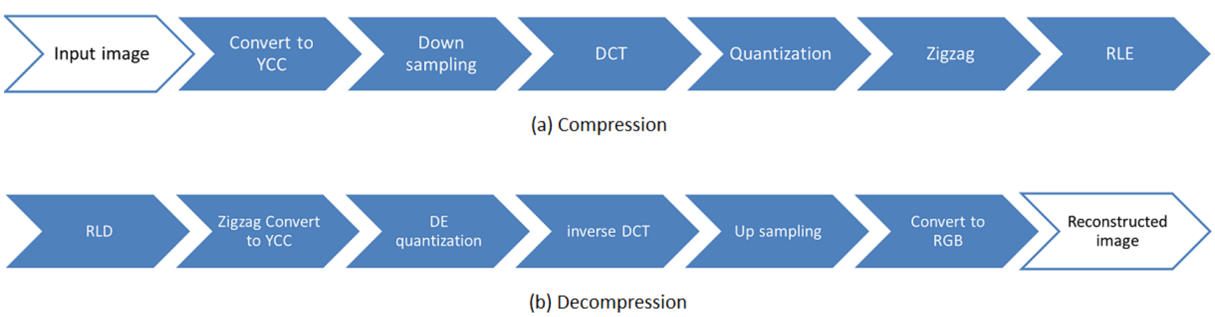


Figure 1: (a) Block diagram of techniques implemented during JPEG compression (b) Block diagram of techniques implemented during JPEG decompression to reconstruct image

3.1 Compression

1. **Convert input image to YCbCr**: First of all the color space of the input image I_{in} is converted from RGB to YCbCr I_{ycc}. The major reason for converting the image to YCbCr is the ease of operation on the chrominance and luminance component of the image. It exploits the fact that human eye is more sensitive to intensity changes

rather than color changes. Thus, the chrominance components are down sampled to half of its size as changes in the chrominance are not identified easily.

2. **Divide image into small blocks**: Each of the three channel in I_{ycc} is converted in to 8x8 sub images I_{block}. Here 8x8 size is obtained empirically. As Figure 2 shows, increasing the size of block also improves image quality. But as the size of block is increased the DCT calculation becomes computationally expensive. As observed in the image 2, image quality for 8x8 block is sufficiently similar to original image.

Figure 2: Applying DCT over left most image and reconstructing image with 25% of the coefficients.(a)original image (b) 2x2 block size (c) 4x4 block size (d) 8x8 block size

3. **Apply DCT** JPEG compression typically uses type-II discrete cosine transform (DCT-II). The major reason for using DCT-II is it's strong energy compaction property. Hence we apply DCT to I_{block} which gives us I_{dct}. DCT transforms the image from spatial domain to frequency domain. In frequency domain it represents an image in terms of 64(8*8) fundamental frequency components. DCT for any matrix can be calculated by equation 1. Inverse of this transform can be calculated by equation 2 which is known as inverse DCT(IDCT). IDCT is used to convert frequency domain data to spacial domain.

$$T(u,v) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} f(x,y)r(x,y,u,v) \tag{1}$$

$$f(x,y) = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u,v)s(x,y,u,v) \tag{2}$$

Here,

$$r(x,y,u,v) = s(x,y,u,v) = \alpha(u)\alpha(v)cos[(2x+1)\frac{u\pi}{2n}]cos[(2y+1)\frac{v\pi}{2n}] \tag{3}$$

Figure 3 shows the 2-Dimensional fundamental frequencies from JPEG DCT. Any image can be represented as linear combination of below shown blocks. In the image, going from left to right shows the increase in horizontal frequency where as going down from the top shows increase in vertical frequency. Thus the top left corner corresponds to the DC component(summation of all intensity levels) of the image while the bottom right corner corresponds to the highest frequency component of the image.

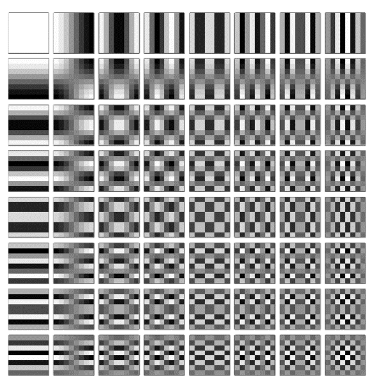


Figure 3: 2-Dimensional frequencies from JPEG DCT

4. **Quantization** Quantization is performed on I_{dct} to get I_{quant}. The quantization matrices are based on the quality factor which lies between 1 to 100. The quality of the image improves with the increase in quality factor. The matrix obtained after quantization is sparsified because of the frequency components having less weight are discarded. These frequency components majorly correspond to higher frequency.

As mentioned above, human eye is more sensitive to luminance rather than chrominance, removal of high frequency data will affect the perceived information of the image.

5. **Zigzag traversal**

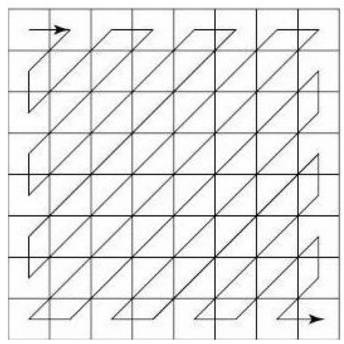


Figure 4: Zigzag traversal

The quantized data I_{quant} which is in the form of 8x8 matrix is subjected to zigzag traversal to give I_{zz}. It converts the 8x8 matrix block to a 1x64 vector. Figure 4 shows the zigzag traversal on an 8x8 block. It can be inferred from the Figure 4 that most of the zeros will be at the end of the vector as it corresponds to high frequency components.

6. **Run length encoding**

14	1	0	0	0	0	0	0	0	0

Input vector of 1x10 dimension which goes as an input to RLE

14	1	1	1	0	8

Output of RLE

Figure 5: Performing RLE on an input vector of dimension 1x10

The output after zigzag traversal I_{zz} is subjected to run length encoding(RLE) to give I_{rle}. This output is written to a file which is used during the decompression to reconstruct the image. As shown in Figure 5 the output of RLE is smaller in length as compared to the input. The output of RLE is in the form of (value, repetition). For eg. 14 is the value and 1 is times 14 is repeated.

3.2 Decompression

For decompression the steps followed in compression are reversed to reconstruct the image.

4 Results

In JPEG compression, the amount of compression one can achieve depends on tpan he Q-factor. Results shown in Table 1 and 2 highlight this fact. It showcases a set of images with different Q-factor. For the given input image we can see that for Q-factor equal to 45 and above, the reconstructed image looks almost similar. Also, the size of these images is significantly smaller than the size of input image. Variation in size of compressed image and the compression ratio w.r.t to the Q-factor is shown in Figure 7 and 8. A compression ratio of 10:1 (input image size: output image size) is obtained for Q-factor equal to 5. A compression ratio of 5:1 is obtained for Q-factor equal to 45.

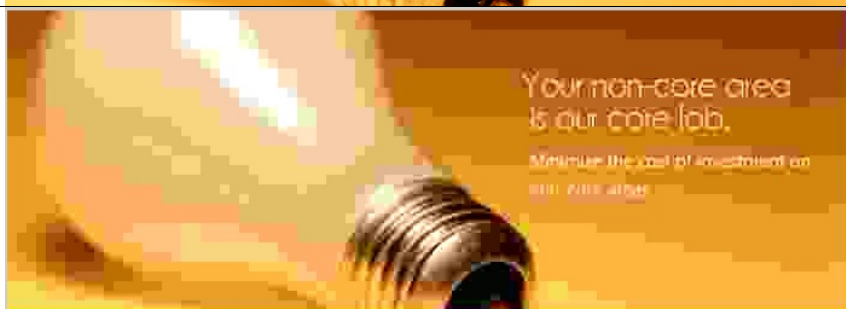
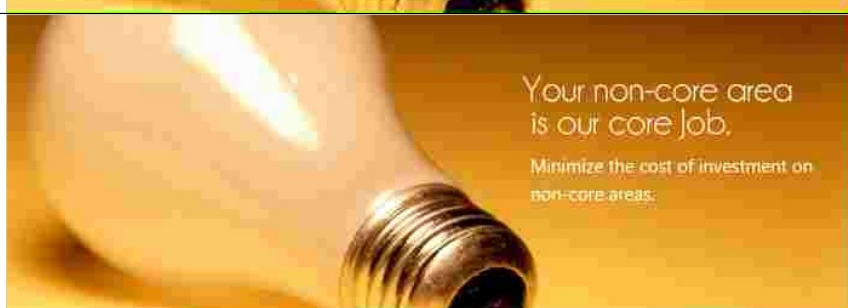
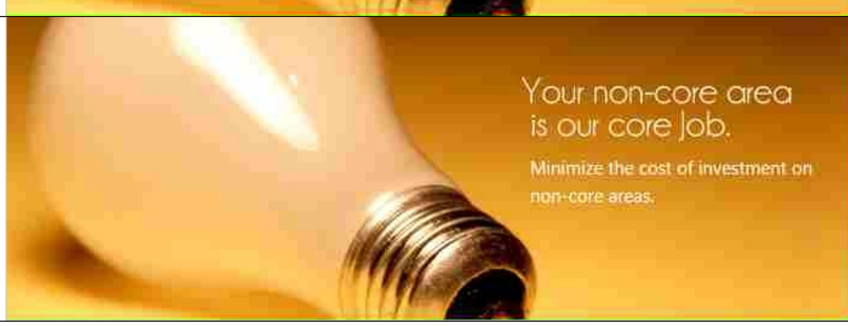
Image	Q-factor	Size
	Input Image	511 KB
	5	51.3 KB
	10	63.3 KB
	15	71.5 KB
	20	78.7 KB

Table 1: Result of compressed images with different Q-factors.

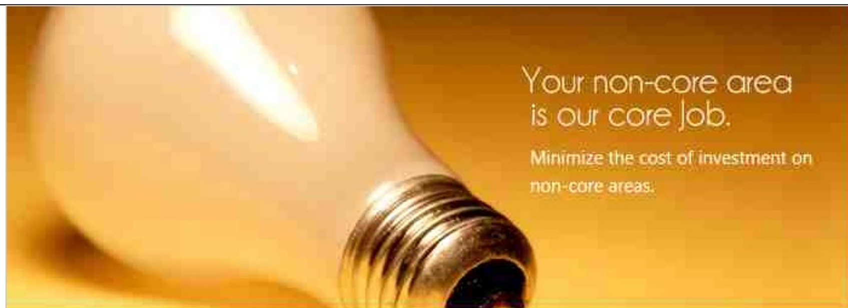
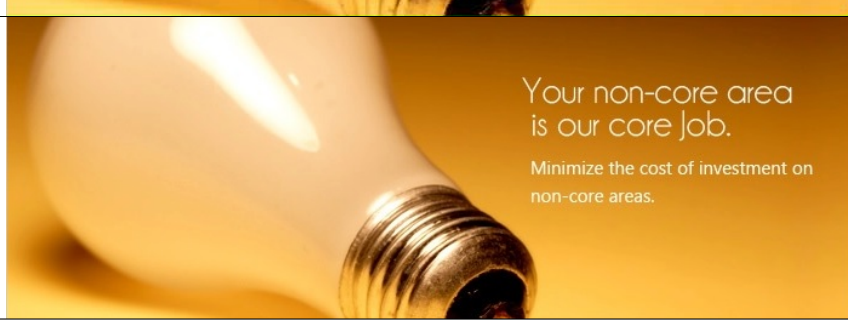
<table>
<tr><th>Image</th><th>Q-factor</th><th>Size</th></tr>
<tr><td>
</td><td>25</td><td>85.2 KB</td></tr>
<tr><td>
</td><td>35</td><td>98.8 KB</td></tr>
<tr><td>
</td><td>45</td><td>109 KB</td></tr>
<tr><td>
</td><td>55</td><td>118 KB</td></tr>
<tr><td>
</td><td>95</td><td>269 KB</td></tr>
</table>

Table 2: Result of compressed images with different Q-factors.

We also calculated the RMSE using equation 4. Figure 6 shows how RMSE varies w.r.t Q-factor. As expected the RMSE(root mean square error) decreases as the Q-factor increases.

$$RMSE = \sqrt{\frac{1}{p}\sum_{i=0}^{p}(Input_i - Reconstructed_i)^2} \quad (4)$$

P = Total number of pixels in the image i.e. Summation of pixels in three channels.

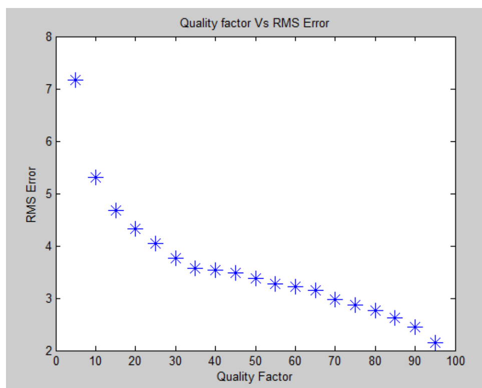


Figure 6: Graph showing change in the RMS error for different Q-factor. With increase in Q-factor RMS error decreases

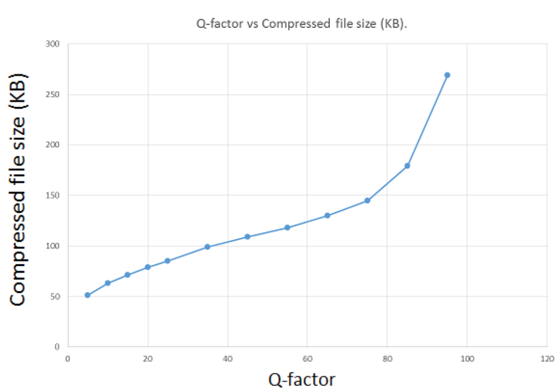


Figure 7: Q-factor Vs. output file size

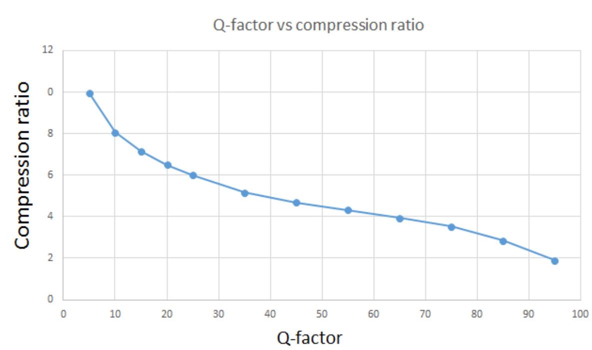


Figure 8: Q-factor Vs. compression ratio

5 Conclusion & future work

Compression techniques have become a necessity in today's world to optimally utilize the resources like bandwidth, storage space, etc. This project presented simulation of one such technique called JPEG compression and decompression. It exploits the fact that human eye is less sensitive to high intensity variation and color variation in an image. Hence, we obtain a compressed image by removing such information. The compressed image will contain the same perceived information as the input image. The technique is well explained and implemented in the above sections. Results shown in Table 1 and 2 and Figure 6 helps to conclude that the given technique can be useful in image compression without losing the perceived information content of the image.

To obtain better compression results one can apply Huffman coding to the output generated after RLE.

6 Matlab Code

6.1 Codebase

Table 3 contains the list of files along with its functionality, developed for the simulation of JPEG compression and decompression. These functionality can be easily mapped to the techniques shown in Figure 1.

Filename	Usage
writeImage.m	Reads the input image and writes the data of the R, G and B channels separately in binary form
readImage.m	Read the R, G and B channels of image written in binary by writeImage.m
main.m	Initialize the quality factor and quantization matrices for chrominance and luminance. Also works as handle to call the compress.m and decompress.m.
compress.m	It simulates JPEG compression. It performs various operation like DCT transform, quantization, zigzag traversal and RLE on 8x8 block.
decompress.m	It simulates JPEG decompression. It performs steps performed by compress.m in reverse order.
zigzag_1.m	It does zigzag encoding and decoding during compress and decompress
rle.m	It is used for run length encoding on the output of the zigzag.m
unrle.m	It is used for run length decoding on the output of rle.m

Table 3: Table containing names of files and their usage in the JPEG compression and decompression

6.2 writeImage.m

```
% This functions takes an RGB image matrix as a input and stores
    that matrixs into three binary files

function []=writeImage(imMat)
    % Open three binary files to write matrices of Red Green and
        Blue channel
    fred=fopen('iR.bin','w');
    fgreen=fopen('iG.bin','w');
    fblue=fopen('iB.bin','w');

    % Seperate three channels of an image
    iR=imMat(:,:,1);
    iG=imMat(:,:,2);
    iB=imMat(:,:,3);
```

```

    % Write matrices in the binary file
    fwrite(fred,iR);
    fwrite(fgreen,iG);
    fwrite(fblue,iB);
    fclose(fred);
    fclose(fgreen);
    fclose(fblue);
end
```

6.3 readImage.m

```
% This function takes binary files as a input and reads
    matricies contained by those file. These matrices would be
    converted into image matrix.
function [i]=readImage(iR,iG,iB)

    % Open binary files which contains R,G and B matrices of an
        image
    fred=fopen(iR);
    fgreen=fopen(iG);
    fblue=fopen(iB);

    % Read matrices from the file and reshape the matrix with
        dimension 250*698
    iR1=fread(fred,[250 698]);
    iG1=fread(fgreen,[250 698]);
    iB1=fread(fblue,[250 698]);

    % combine matrices to obtain an image
    i(:,:,1)=iR1;
    i(:,:,2)=iG1;
    i(:,:,3)=iB1;
    i=uint8(i);
end
```

6.4 main.m

```
clear all;
q_factor=100;
height=698;
width=250;

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% PRE-PROCESSING %%%%%%%%%%%%%%%%%%%%%%%%%%%%
if q_factor < 50
    q_scale = floor(5000 / q_factor);
else
    q_scale = 200 - 2 * q_factor;
end
% Initialization of quantization matrices for chrominance and luminance
% Quant luminance
Q_y = [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];
% Quant chrominance
Q_c = [ 17 18 24 47 99 99 99 99 ;18 21 26 66 99 99 99 99 ;
    24 26 56 99 99 99 99 99; 47 66 99 99 99 99 99 99;
    99 99 99 99 99 99 99 99; 99 99 99 99 99 99 99 99;
    99 99 99 99 99 99 99 99; 99 99 99 99 99 99 99 99 ] ;

% Scale Quant luminance and chrominance
Q_y = round(Q_y .* (q_scale / 100));
Q_c = round(Q_c .* (q_scale / 100));


% call matlab code for compression
compress
figure;

% call matlab code for decompression
decompress
```

6.5 compress.m

```
close all;
%Read Image
RGB = readImage('iR.bin','iG.bin','iB.bin');
imshow(RGB);
% Convert to YCbCr
ycc = rgb2ycbcr(RGB);
% Downsampling using bilinear transformation
y = ycc(:,:,1);
Cb = ycc(:,:,2);
Cr = ycc(:,:,3);
```

```
Cb_down = imresize(Cb, 0.5, 'bilinear');
Cr_down = imresize(Cr, 0.5, 'bilinear');

% Convert the luminance height and width to multiple of 8
if rem(size(y,1),8) ~= 0
    y = [y; zeros(8-rem(size(y,1),8), size(y,2))];
end
if rem(size(y,2),8) ~= 0
    y = [y zeros(size(y,1), 8-rem(size(y,2),8))];
end

% Convert the chrominance height and width to multiple of 8
if rem(size(Cb_down,1),8) ~= 0
    Cb_down = [Cb_down; zeros(8-rem(size(Cb_down,1),8), size(Cb_down,2))];
    Cr_down = [Cr_down; zeros(8-rem(size(Cr_down,1),8), size(Cr_down,2))];
end
if rem(size(Cb_down,2),8) ~= 0
    Cb_down = [Cb_down zeros(size(Cb_down,1), 8-rem(size(Cb_down,2),8))];
    Cr_down = [Cr_down zeros(size(Cr_down,1), 8-rem(size(Cr_down,2),8))];
end

%%%%%%%%%%%%%%%%%%%%%%% COMPRESSION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Block wise (8x8) DCT of chrominance and luminance
dct = @(block_struct) dct2(block_struct.data);
y_dct = blockproc(y, [8 8], dct);
Cb_down_dct = blockproc(Cb_down, [8 8], dct);
Cr_down_dct = blockproc(Cr_down, [8 8], dct);

% Quantization of chrominance and luminance
quant_y = @(block_struct) round(block_struct.data ./ Q_y);
quant_c = @(block_struct) round(block_struct.data ./ Q_c);
y_dct_quant = blockproc(y_dct, [8 8], quant_y);
Cb_down_dct_quant = blockproc(Cb_down_dct, [8 8], quant_c);
Cr_down_dct_quant = blockproc(Cr_down_dct, [8 8], quant_c);
xyz=1;
%Zigzag encoding
fh = fopen('y.txt','w');
```

```
for i = 1:8:size(y_dct_quant,1)
    for j = 1:8:size(y_dct_quant,2)
        block = y_dct_quant(i:i+7, j:j+7);
        straight = zigzag_1(block);
       %Apply RLE on the vector
       %int2str(uint8(straight))
        rle(straight,fh);
        xyz=xyz+1;
    end
end
fprintf(fh,'%c','.');
fclose(fh);

 %Zigzag encoding for chrominance
 fcb = fopen('cb.txt','w');
 fcr = fopen('cr.txt','w');
for i = 1:8:size(Cr_down_dct_quant,1)
    for j = 1:8:size(Cr_down_dct_quant,2)
       %for Cr
        block = Cr_down_dct_quant(i:i+7, j:j+7);
        straight = zigzag_1(block);
       %Apply RLE on the vector
        rle(straight,fcr);
       %for Cb
        block = Cb_down_dct_quant(i:i+7, j:j+7);
        straight = zigzag_1(block);
       %Apply RLE on the vector
        rle(straight,fcb);
    end
end
fprintf(fcb,'%c','.');
fclose(fcb);
fprintf(fcr,'%c','.');
fclose(fcr);
```

6.6 decompress.m

```
yName='y.txt';
cbName='cb.txt';
crName='cr.txt';


straighty=unrle(yName);
y_dct_quant=0;
```

```
i=1;
j=i;
k=1;
while (i<width)
    while (j<height)
         block=zigzag_1(straighty(k,:));
         y_dct_quant(i:i+7,j:j+7)=block;
         j=j+8;
         k=k+1;
    end
    j=1;
    i=i+8;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%get Cb
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
straightcb=unrle(cbName);
Cb_down_dct_quant=0;
i=1;
j=i;
k=1;
while (i<=width/2)
    while (j<=height/2)
         block=zigzag_1(straightcb(k,:));
         Cb_down_dct_quant(i:i+7,j:j+7)=block;
        j=j+8;
         k=k+1;
    end
    j=1;
    i=i+8;
end

%%%%%%%%%%%%%%%%%%%%get Cr%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
straightcr=unrle(crName);
Cr_down_dct_quant=0;
i=1;
j=i;
k=1;
while (i<=width/2)
    while (j<=height/2)
         block=zigzag_1(straightcr(k,:));
         Cr_down_dct_quant(i:i+7,j:j+7)=block;
         j=j+8;
```

```
            k=k+1;
        end
        j=1;
        i=i+8;
end


%%%%%%%%%%%%%%%%%%%%%%%% DECOMPRESSION %%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % De-quantization
    de_quant_y = @(block_struct) block_struct.data .* Q_y;
    de_quant_c = @(block_struct) block_struct.data .* Q_c;
    y_dct = blockproc(y_dct_quant, [8 8], de_quant_y);
    Cb_down_dct = blockproc(Cb_down_dct_quant, [8 8], de_quant_c);
    Cr_down_dct = blockproc(Cr_down_dct_quant, [8 8], de_quant_c);

    % Block wise (8x8) iDCT of chrominance and luminance
    idct = @(block_struct) idct2(block_struct.data);
    y_idct = blockproc(y_dct, [8 8], idct);
    Cb_down_idct = blockproc(Cb_down_dct, [8 8], idct);
    Cr_down_idct = blockproc(Cr_down_dct, [8 8], idct);

    % Upsampling Chrominance using bilinear transform
    Cb_up = imresize(Cb_down_idct, 2, 'bilinear');
    Cr_up = imresize(Cr_down_idct, 2, 'bilinear');

    %Reconstructing the chrominance and luminance of the same size
    y_reconstruct = y_idct(1:width, 1:height);
    Cb_up_reconstruct = Cb_up(1:width, 1:height);
    Cr_up_reconstruct = Cr_up(1:width, 1:height);

    % Reconstruct image similar to original ycc image
    ycc_reconstruct = zeros([width,height,3]);
    ycc_reconstruct(:,:,1) = y_reconstruct;
    ycc_reconstruct(:,:,2) = Cb_up_reconstruct;
    ycc_reconstruct(:,:,3) = Cr_up_reconstruct;
    ycc_reconstruct = uint8(ycc_reconstruct);

    % Reconstruct image similar to original RGB image
    RGB_reconstruct = uint8(ycbcr2rgb(ycc_reconstruct));
```

```

    imshow(RGB_reconstruct);
```

6.7 rle.m

```
% Function for doing RLE.
% Input: Vector
% Output: Char1 freq1 char2 freq2 ...,
% Sample
% Input: [1 1 1 0 0 1 0 1]
% Output: 1 3 0 2 1 1 0 1 1 1 , (in a file named data.txt)

function [] = rle(straight,fh)
    % count the frequency of each character
    i=1;
    while(i <= size(straight,2))
        curr = straight(i);
        fprintf(fh,'%s ',int2str(curr));
        j=i+1;
        while(j <= size(straight,2))
            if curr == straight(j)
                j = j + 1;
            else
                count = j - i;
                i = j;
                break;
            end
        end
        if j>size(straight,2)
            count = j-i;
            i = j;
        end
        fprintf(fh,'%s ', int2str(count));
    end
    fprintf(fh,'%c',',');
end
```

6.8 unrle.m

```
function straight=unrle(filename)
    f=fopen(filename,'r');
    row=0;
    while(1)
        x=fscanf(f,'%d');
```

```
        if length(x)>=1
            values=0;
            freq=0;
            row=row+1;
            temp=x;
            i=1;
            j=1;
            k=1;
            while (i<=length(x))
                if(mod(i,2) == 1)
                    values(j)= x(i);
                    j=j+1;
                else
                    freq(k)=x(i);
                    k=k+1;
                end
                i=i+1;
            end
            i=1;
            j=1;
            while(i <= length(freq))
                straight(row,j:j+freq(i)-1)=values(i);
                j=j+freq(i);
                i=i+1;
            end
        else
            x=fscanf(f,'%c',1);
            if x==','
    %               continue;
            elseif x=='.'
                break;
            end
        end
    end
    fclose(f);
end
```

6.9 zigzag.m

```
% function output = zigzag (input)
% The function zigzag takes a matrix (8x8) or a vector as an input
% argument.
% Based on the input if performs zigzag encoding or decoding
```

```matlab

function output = zigzag_1(input)
    % Init indices
    straight_index = 2;
    x_index = 1;
    y_index = 2;
    flag = 1;
    % Matrix to vector (Encoding)
    if size(input,1) == 8 && size(input,2) == 8
        output = zeros(1,64);
        output(1) = input(1,1);
        input_flag = 1;
    % Vector to matrix (Decoding)
    elseif size(input,1) == 1 && size(input,2) == 64
        output = zeros(8,8);
        output(1,1) = input(1);
        input_flag = 2;
    end
    while straight_index < 65
        % Check matrix to vector or vice versa
        if input_flag == 1
            output(straight_index) = input(x_index,y_index);
        elseif input_flag == 2
            output(x_index,y_index) = input(straight_index);
        end
        % Cross Traverse down
        if flag == 1
            x_index = x_index + 1;
            y_index = y_index - 1;
        % Cross Traverse up
        elseif flag == 2
            x_index = x_index - 1;
            y_index = y_index + 1;
        end
        straight_index = straight_index + 1;
        % Boundary conditions for zig-zag encoding on 8x8 block
        % Current position is in the bottom right corner of cube
        if(x_index > 8 && y_index < 1)
            x_index = 8;
            y_index = 2;
            flag = 2;
        % Current position is in the bottom of cube
        elseif(y_index < 1)
```

```
            y_index = y_index + 1;
            flag = 2;
        % Current position is in the left side of cube
        elseif(x_index < 1)
            x_index = x_index + 1;
            flag = 1;
        % Current position is in the top of cube
        elseif(y_index > 8)
            y_index = y_index - 1;
            x_index = x_index + 2;
            flag = 1;
        % Current position is in the right side of cube
        elseif(x_index > 8)
            x_index = x_index - 1;
            y_index = y_index + 2;
            flag = 2;
        end
    end
end
```

References

[1] Big Data, for better or worse: 90% of world's data generated over last two years, SINTEF. May 22, 2013.
http://www.sciencedaily.com/releases/2013/05/130522085217.htm

[2] Human eye sensitivity and photometric quantities,
http://www.ecse.rpi.edu/∼ schubert/Light-Emitting-Diodes-dot-org/Sample-Chapter.pdf

[3] EYE INTENSITY RESPONSE, CONTRAST SENSITIVITY,
http://www.telescope-optics.net/eye_intensity_response.htm

[4] Dwivedi, Harsh Vardhan. Design of JPEG Compressor. Diss. National Institute of Technology Rourkela, 2009.
ethesis.nitrkl.ac.in/1090/1/Thesis.pdf

[5] Andrew B. Lewis, University of Cambridge, Computer Laboratory. *JPEG tutorial*.
https://www.cl.cam.ac.uk/teaching/1011/R08/jpeg/acs10-jpeg.pdf

[6] JPEG - Cardiff School of Computer Science & Informatics.
https://www.cs.cf.ac.uk/Dave/Multimedia/PDF/11_JPEG.pdf