Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3551 - DIGITAL SIGNAL PROCESSING

APPLICATION OF 2D-DCT FOR IMAGE COMPRESSION

MANIMOHAN T. *200377M*

This report is submitted as Assignment of module EN3551
18^{th} October 2023

ABSTRACT

This assignment report focuses on the application of 2D-DCT for image compression, where the Discrete Cosine Transform is used to compress digital images. The report provides a theoretical background on 2D-DCT, explaining its properties, orthogonality, and energy preservation. It also discusses the quantization and coding process for compression and the subsequent decompression steps.

The assignment involves the compression of various images with different quality levels. The performance is evaluated in terms of the percentage of zeros, peak signal-to-noise ratio (PSNR), and visual quality. Through this assignment, students gain a practical understanding of the principles of image compression using 2D-DCT and the impact of different quality levels on compression results. This practical experience enhances their knowledge of digital signal processing and image compression techniques.

1 DETECTING HARMONICS IN NOISY DATA AND SIGNAL INTERPOLATION USING DFT

1.1 Procedure

1.1.1 2-D DCT - Application to Image Compression

Suppose the image to be compressed has size $M \times N$, where M and N are multiples of 8 and represented with 8 bits. The image is first divided into 8×8 blocks. The steps described below are applied to each of these blocks. Let B be a representative 8×8 data block.

Step 1: Level-Off and 2-D DCT

Since DCT gives a lower DC coefficient when pixel values are in the range -128 to 127 rather than in the range 0 to 255, B is leveled off by subtracting 128 from each entry. For illustration purposes, we take the $(16, 18)$-th block of the 256×256 image Lena as matrix B.

$$B = \begin{bmatrix} 125 & 134 & 137 & 139 & 138 & 138 & 141 & 142 \\ 113 & 119 & 126 & 134 & 139 & 141 & 144 & 149 \\ 80 & 95 & 103 & 106 & 114 & 127 & 141 & 147 \\ 63 & 65 & 53 & 62 & 75 & 86 & 108 & 130 \\ 93 & 80 & 60 & 33 & 35 & 35 & 52 & 69 \\ 126 & 108 & 88 & 74 & 53 & 45 & 35 & 32 \\ 130 & 116 & 90 & 96 & 62 & 63 & 55 & 49 \\ 115 & 80 & 61 & 65 & 68 & 88 & 68 & 75 \end{bmatrix}$$

The modified data block after subtracting 128 from B is given by,

$$\tilde{B} = \begin{bmatrix} -3 & 6 & 9 & 11 & 10 & 10 & 13 & 14 \\ -15 & -9 & -2 & 6 & 11 & 13 & 16 & 21 \\ -48 & -33 & -25 & -22 & -14 & -1 & 13 & 19 \\ -65 & -63 & -75 & -66 & -53 & -42 & -20 & 2 \\ -35 & -48 & -68 & -95 & -93 & -93 & -76 & -59 \\ -2 & -20 & -40 & -54 & -75 & -83 & -93 & -96 \\ 2 & -12 & -38 & -32 & -66 & -65 & -73 & -79 \\ -13 & -48 & -67 & -63 & -60 & -40 & -60 & -53 \end{bmatrix}$$

Applying 2-D DCT to $\tilde{B}$, we obtain,

$$C = T_8 B \tilde{T}_8 = \begin{bmatrix} -272.3750 & 17.1771 & 46.7784 & 4.2270 & 6.1250 & -0.5799 & 0.2421 & -8.4813 \\ 182.5146 & -109.5361 & -28.0398 & -25.1231 & -8.9567 & -7.0036 & -6.1703 & 10.3445 \\ 117.4897 & 19.1429 & -30.8911 & 15.7065 & 1.7309 & 6.7882 & 5.4116 & -8.5788 \\ -23.4612 & 97.7162 & -0.0872 & -7.0795 & -1.0461 & 2.6980 & -2.9351 & 2.5446 \\ -48.3750 & -35.2958 & 27.0407 & 6.4060 & 4.1250 & -7.3615 & 3.5470 & 4.3781 \\ 15.8386 & -7.1745 & -7.9234 & -6.7518 & -0.5166 & 3.8019 & -8.3849 & -7.4654 \\ 0.4477 & 1.3348 & -5.5884 & 3.4787 & -4.6404 & -0.7361 & 4.6411 & 2.6707 \\ -3.6673 & 9.8948 & 6.2377 & -7.3148 & -7.0342 & 1.0065 & -0.6727 & 2.3138 \end{bmatrix}$$

Alternatively, one may apply MATLAB function `dct2` to $\tilde{B}$ to obtain C.

Step 2: Quantization

A critical step in image compression is quantization and coding of the DCT coefficients in C. The quantization may be achieved by converting matrix $C = \{c_{i,j}\}$ to matrix $S = \{s_{i,j}\}$ determined by,

$$s_{i,j} = \text{round}\left(\frac{c_{i,j}}{q_{i,j}}\right)$$

where $Q = \{q_{i,j}\}$ is a quantization matrix that may be selected according to a desired quality level of the compression. In the JPEG standard, subjective experiments involving the human visual system have produced their own quantization matrices. For example, with a quality level 50 (on a 1-to-100 scale) we have,

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Quantization matrices of other quality levels can be obtained by multiplying the above standard quantization matrix by a scaling factor τ where,

$$\tau = \begin{cases} \frac{100-\text{quality level}}{50}, & \text{quality level} > 50 \\ \frac{50}{\text{quality level}}, & \text{quality level} < 50 \end{cases}$$

The scaled quantization matrix is then rounded and clipped to integer values between 0 and 255. With C obtained in Step 1 and $Q = Q_{50}$, the matrix S in Eq. 10 is computed as

$$S = \begin{bmatrix} -17 & 2 & 5 & 0 & 0 & 0 & 0 & 0 \\ 15 & -9 & -2 & -1 & 0 & 0 & 0 & 0 \\ 8 & 1 & -2 & 1 & 0 & 0 & 0 & 0 \\ -2 & 6 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Evidently, if a quantization matrix with a reduced quality level is used, S will contain more zeros, thus a higher compression can be achieved at the cost of reduced image quality.

Step 3: Coding

The quantized matrix S is then converted by an encoder to a stream of binary data like $\{1001101, \ldots\}$. Detailed coverage of the coding process is beyond the scope of this experiment. Instead, we mention the following.

* The DC coefficients of the 8×8 blocks are encoded by a differential pulse-code modulation (DPCM) coding (also referred to as DC prediction).
* Because the location of the retained coefficients varies from block to block, the quantized AC coefficients are zigzag scanned and ordered into (run, level) pairs, where "level" means the value of a nonzero coefficient, and "run" means the number of zero coefficients preceding it.
* These (run, level) pairs are entropy coded, that is, longer codes for less frequent pairs and vice versa.

Step 4: Decompression

This is a step to be carried out at the receiver to reconstruct the image. It consists of the following three simple operations.

* Pointwise multiplication of matrix S with the quantization matrix Q to get an image block $R = Q \circ S$. In our example, the Q matrix was taken to be Q_{50} that yields,

$$R = \begin{bmatrix} -272 & 22 & 50 & 0 & 0 & 0 & 0 & 0 \\ 180 & -108 & -28 & -19 & 0 & 0 & 0 & 0 \\ 112 & 13 & -32 & 24 & 0 & 0 & 0 & 0 \\ -28 & 102 & 0 & 0 & 0 & 0 & 0 & 0 \\ -54 & -44 & 37 & 0 & 0 & 0 & 0 & 0 \\ 24 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

* Apply 2-D inverse DCT to matrix R. Note that this can be done by using the MATLAB command `idct2`. Alternatively, it can be done using $D_8^T R D_8$. This gives

$$E = D_8^T R D_8 =$$

$$\begin{bmatrix} 2.7181 & 2.1228 & 2.2693 & 4.2600 & 7.6484 & 10.8548 & 12.7203 & 13.3406 \\ -18.1034 & -10.6760 & -0.3337 & 7.7581 & 11.5839 & 12.5496 & 12.8140 & 13.1153 \\ -43.2294 & -34.5413 & -23.6264 & -15.8552 & -9.7946 & -0.4317 & 12.7343 & 23.0141 \\ -57.9809 & -58.9924 & -63.0283 & -67.8737 & -64.7907 & -46.4675 & -17.7381 & 4.4189 \\ -40.7650 & -50.9610 & -69.2674 & -88.8143 & -98.8380 & -92.2600 & -73.6202 & -57.6914 \\ 0.1073 & -12.7805 & -34.0432 & -57.0831 & -76.3805 & -89.4098 & -96.5154 & -99.3926 \\ 6.4904 & -8.6805 & -28.5866 & -43.2078 & -52.0062 & -61.1538 & -73.3190 & -83.0164 \\ -25.3942 & -44.5619 & -65.1582 & -70.4424 & -60.8203 & -50.0267 & -47.8854 & -50.9752 \end{bmatrix}$$

* Finally, the effect of the "level-off" operation in Step 1 is taken into account by adding 128 to the entries of the matrix E obtained above. This yields the reconstructed image block that mimics the image block B as,

$$B = E + 128 =$$

$$\begin{bmatrix} 130.7181 & 130.1228 & 130.2693 & 132.2600 & 135.6484 & 138.8548 & 140.7203 & 141.3406 \\ 109.8966 & 117.3240 & 127.6663 & 135.7581 & 139.5839 & 140.5496 & 140.8140 & 141.1153 \\ 84.7706 & 93.4587 & 104.3736 & 112.1448 & 117.2054 & 127.5683 & 139.7343 & 151.0141 \\ 70.0191 & 69.0076 & 64.9717 & 60.1263 & 63.2093 & 81.5325 & 110.2619 & 132.4189 \\ 87.2350 & 77.0390 & 58.7326 & 39.1857 & 29.1620 & 35.7400 & 54.3798 & 70.3086 \\ 128.1073 & 115.2195 & 93.9568 & 70.9069 & 51.6195 & 38.5902 & 31.4846 & 28.6074 \\ 134.4904 & 119.3195 & 99.4134 & 84.7922 & 75.9938 & 66.8462 & 54.6810 & 44.9836 \\ 102.6058 & 83.4381 & 62.8418 & 57.5576 & 67.1797 & 77.9733 & 80.1146 & 77.0248 \end{bmatrix}$$

On comparing the block $B^{\tilde{}}$ with block B (given in Step 1), we see $B^{\tilde{}}$ approximates B quite well. For an image of size $M \times N$, the number of blocks is $M/8 \times N/8$, and the steps described above are applied to each of these blocks for a selected quality level. We count the number of zeros contained in the S matrices (see Step 2) in order to compute the compression factor because the total number of zeros indicates how the image has been compressed. As expected, the use of reduced quality levels leads to a higher percentage of total zeros at the cost of a degraded image.

1.1.2 Peak Signal to Noise Ratio (PSNR)

The PSNR is defined as:

$$\text{PSNR} = 20 \log_{10} \left(\frac{\psi_{\max}}{\sigma_e} \right)$$

where $\psi_{\max}$ denotes the maximum light intensity of the original image and σ_e^2 denotes the mean squared error. For an 8-bit digital image, $\psi_{\max} = 255$. Furthermore, σ_e^2 can be computed in MATLAB applying the command `mean` twice (over the two dimensions) on a squared error matrix. The error matrix is given by $I_{\text{rec}} - I_{\text{original}}$, where I_{rec} and I_{original} are the reconstructed and original images.

1.1.3 Tasks

1. Download 3 images assigned to your index number. Select one additional image of your choice, which is not in the provided set of images.

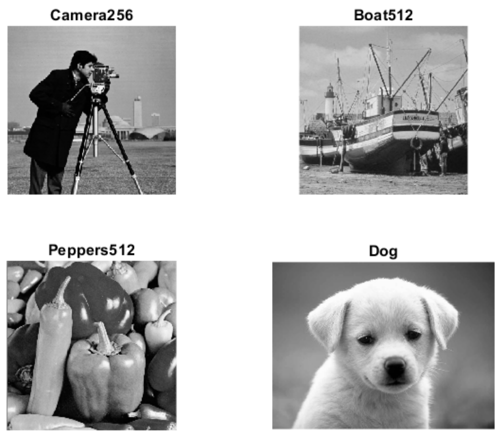

Figure 1 — Load the image

2. Compress each of these images using the DCT-based method described above with three quality levels. The quality levels are also assigned based on your index number.

Compress and observe image 1

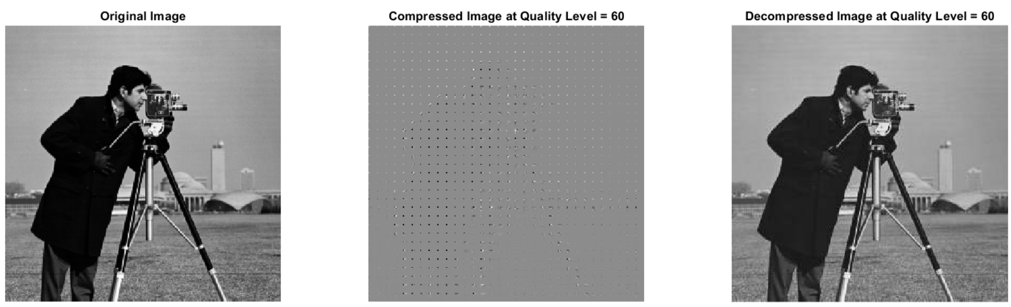

Figure 2 — Compress and observe image 1 at quality level 60

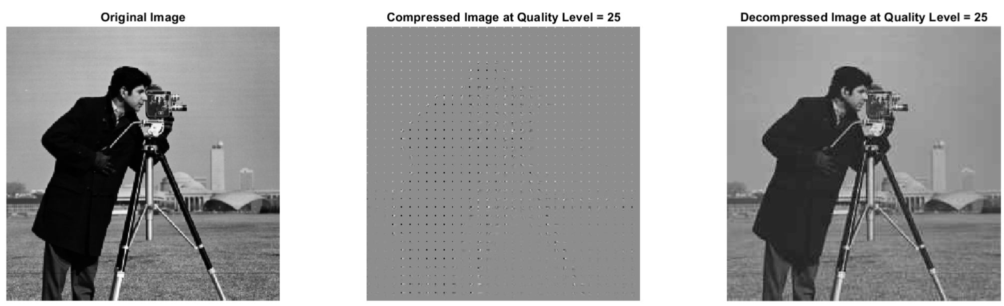

Figure 3 — Compress and observe image 1 at quality level 25

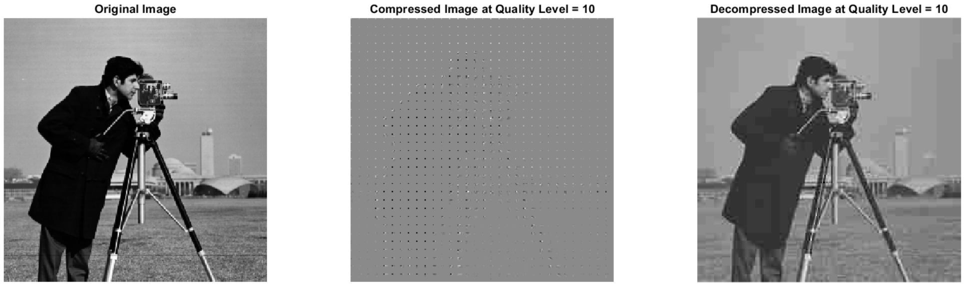

Figure 4 — Compress and observe image 1 at quality level 10

Compress and observe image 2

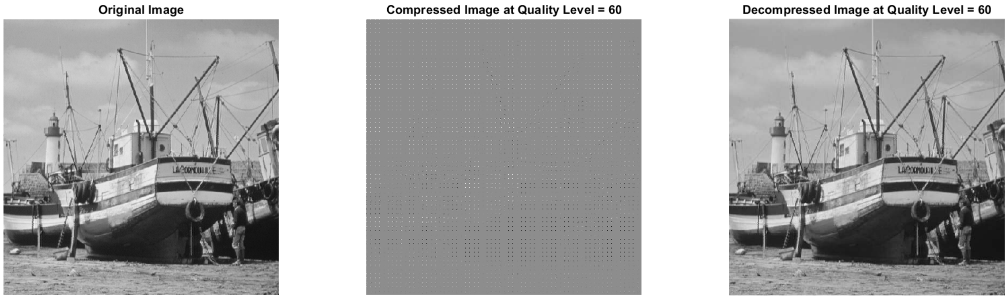

Figure 5 — Compress and observe image 2 at quality level 60

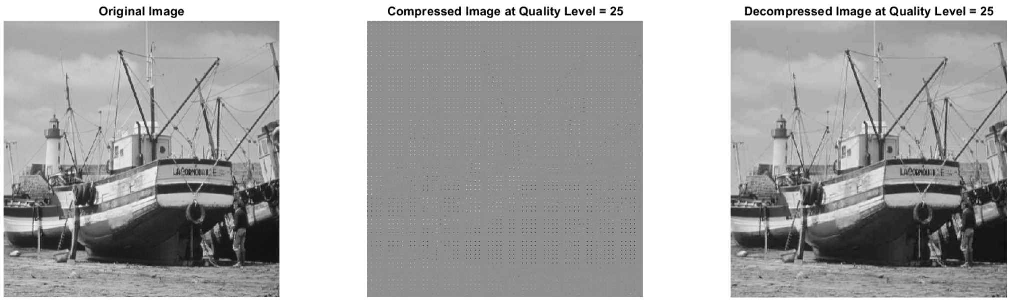

Figure 6 — Compress and observe image 2 at quality level 25

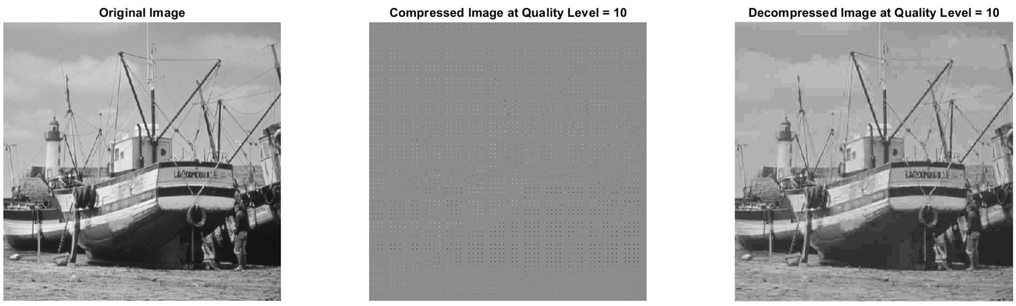

Figure 7 — Compress and observe image 2 at quality level 10

Compress and observe image 3

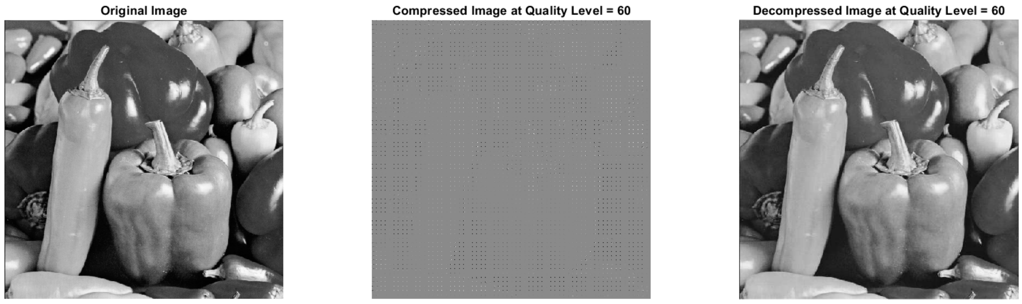

Figure 8 — Compress and observe image 3 at quality level 60

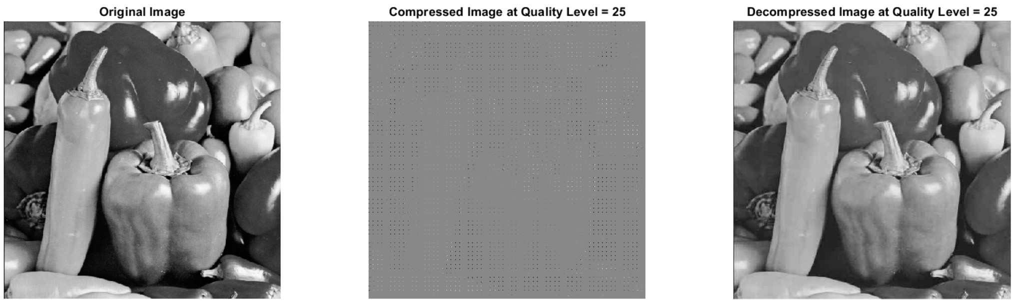

Figure 9 — Compress and observe image 3 at quality level 25

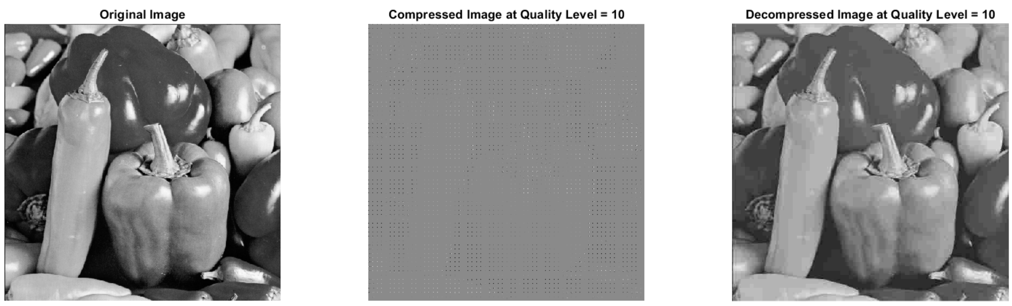

Figure 10 — Compress and observe image 3 at quality level 10

Compress and observe image 4

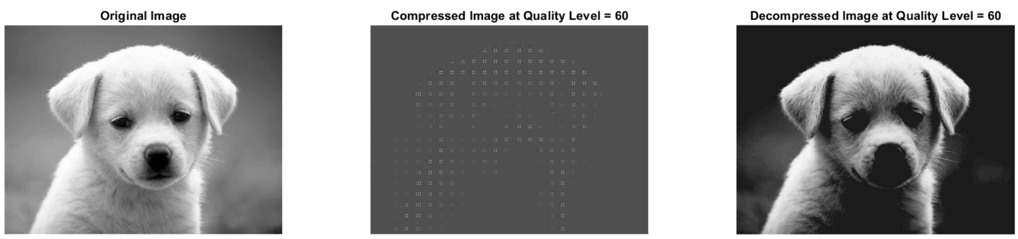

Figure 11 — Compress and observe image 4 at quality level 60

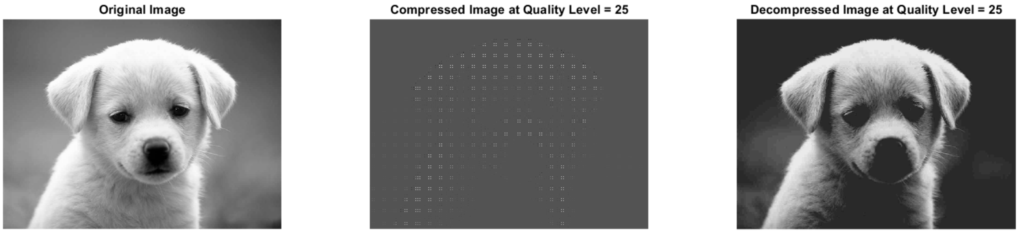

Figure 12 — Compress and observe image 4 at quality level 25

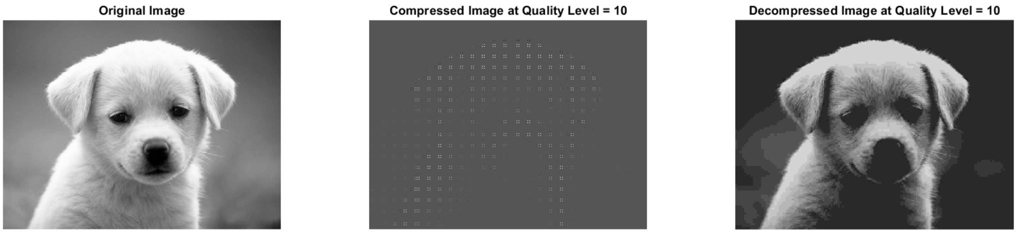

Figure 13 — Compress and observe image 4 at quality level 10

3. Observe the results in terms of:

- Percentage of zeros
- Quality in terms of peak signal-to-noise ratio (PSNR)
- Visual quality of the compressed images as related to the quality level.

Compress and observe image 1

```
When Quality Level = 60
Percentage of Zeros = 83.1192%
Peak Signal-to-Noise Ratio(PSNR) = 32.5619dB

When Quality Level = 25
Percentage of Zeros = 90.6219%
Peak Signal-to-Noise Ratio(PSNR) = 29.1581dB

When Quality Level = 10
Percentage of Zeros = 95.0058%
Peak Signal-to-Noise Ratio(PSNR) = 26.2428dB
```

Figure 14 — Compress and observe the facts of image 1

Compress and observe image 2

```
When Quality Level = 60
Percentage of Zeros = 85.1223%
Peak Signal-to-Noise Ratio(PSNR) = 41.5961dB

When Quality Level = 25
Percentage of Zeros = 91.3395%
Peak Signal-to-Noise Ratio(PSNR) = 38.2472dB

When Quality Level = 10
Percentage of Zeros = 95.3625%
Peak Signal-to-Noise Ratio(PSNR) = 34.8973dB
```

Figure 15 — Compress and observe the facts of image 2

Compress and observe image 3

```
When Quality Level = 60
Percentage of Zeros = 87.1059%
Peak Signal-to-Noise Ratio(PSNR) = 41.4033dB

When Quality Level = 25
Percentage of Zeros = 93.0378%
Peak Signal-to-Noise Ratio(PSNR) = 39.1156dB

When Quality Level = 10
Percentage of Zeros = 95.9991%
Peak Signal-to-Noise Ratio(PSNR) = 36.153dB
```

Figure 16 — Compress and observe the facts of image 3

Compress and observe image 4

```
When Quality Level = 60
Percentage of Zeros = 96.3632%
Peak Signal-to-Noise Ratio(PSNR) = 30.3488dB

When Quality Level = 25
Percentage of Zeros = 97.3148%
Peak Signal-to-Noise Ratio(PSNR) = 30.3305dB

When Quality Level = 10
Percentage of Zeros = 98.6434%
Peak Signal-to-Noise Ratio(PSNR) = 30.2499dB
```

Figure 17 — Compress and observe the facts of image 4

For all 4 images, the following 3 observations can be seen when the quality level decreases from 60 to 25 to 10

a) **An increase in the percentage of zeros** can be observed when reducing the quality level. This phenomenon can be explained using the following equations. When the quality level is low, the scaling factor τ becomes large. For example, at quality level 90, τ is 0.2, whereas at quality level 10, τ becomes 5. A large τ results in a large quantization matrix Q, which, in turn, leads to a smaller scaled quantization matrix S. Consequently, when employing a quantization matrix Q with a lower quality level, the scaled quantization matrix S contains more zeros, achieving higher compression at the expense of reduced image quality. This expected behavior of the percentage of zeros can be observed in the experimental results obtained from all four images.

b) **A decrease in Peak Signal-to-Noise Ratio (PSNR)** can be noted as the quality level decreases. This can be explained using the following equations. When the quality level decreases, the deviation between the reconstructed image and the original image increases. As a result, the mean squared error value increases, leading to a decrease in PSNR, as indicated by the equation. This anticipated behavior of PSNR is evident in the experimental results obtained from all four images.

c) **Visual quality experiences a reduction** as the quality level decreases. However, the relationship between visual quality and the quality value doesn't appear to follow a linear pattern. This can be elaborated as follows.

- The relationship between quality level and perceived visual quality is **non-linear**. Decreasing the quality level suggests increased compression and a potential decrease in visual quality. However, the extent of perceived quality reduction is influenced by various factors and may not strictly correlate with the numerical change in quality level. For instance, reducing the quality value from 60 to 25 doesn't necessarily result in an equivalent reduction in perceived visual quality. This non-linear relationship is evident in the results from Task 2, where it's observed that a decrease from 60 to 25 doesn't significantly degrade perceived visual quality.
- The **threshold effect** is noteworthy. At very high-quality levels, slight reductions in quality may not be noticeable to the human eye. However, as the quality level decreases, there's a point where even a minor reduction can lead to a significant loss in visual quality. This phenomenon is referred to as the "threshold effect."
- **Diminishing Returns**, As the quality level continues to decrease, visual quality degrades more rapidly. Beyond a certain point, further reductions may lead to a disproportionately large drop in visual quality.
- **Visual quality is subjectively perceived** and can vary from person to person. Some individuals may be more sensitive to compression artifacts, while others may be less discerning.

4. Some images are more difficult to compress than others in the sense that the visual (subjective) quality cannot be maintained as easily as others for a given compression ratio. Observe the compression results obtained in task (2) from this perspective and explain why?

   Some images present a greater challenge in terms of compression while preserving visual (subjective) quality compared to others. When examining the results from Task 2, it's essential to understand the reasons behind this phenomenon.

   The difficulty of effectively compressing an image while retaining visual quality is influenced by several key factors: image content, complexity, and frequency distribution. Images featuring intricate textures, patterns, and high-frequency details tend to be more demanding to compress without introducing noticeable artifacts.

   Image Content: Natural images, such as landscapes and portraits, are characterized by intricate details, subtle color variations, and complex textures. This complexity makes it more challenging to achieve compression without compromising visual quality compared to artificial images like text or diagrams.

   Image Complexity: Images with a high degree of randomness or entropy, such as those with considerable noise or fine textures, pose greater difficulties for effective compression.

The compression algorithm struggles to identify redundant patterns or predictable areas within these images.

Frequency Distribution: Images with a broad spectrum of frequencies, including high-frequency components like sharp edges and fine details, also present challenges in compression. High-frequency information is crucial for preserving image sharpness and clarity but is more susceptible to compression artifacts such as ringing and blurring.

In general, images with simpler content, lower complexity, and a narrower frequency distribution are more amenable to compression without compromising visual quality. For instance, images featuring large areas of uniform color, smooth gradients, or simple geometric shapes typically lend themselves well to compression.

These factors, as described, become evident upon reviewing the outcomes from Task 2.

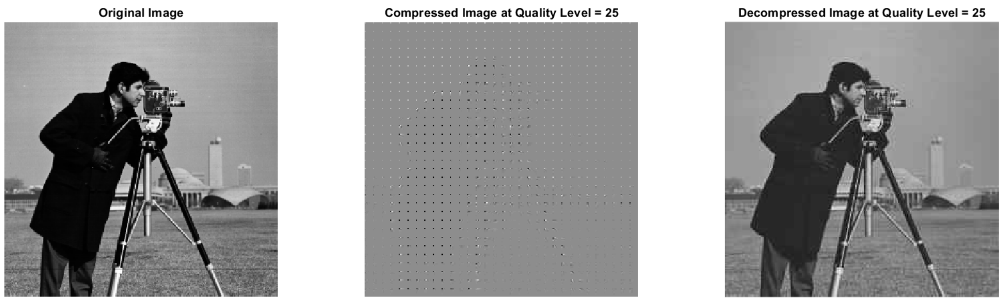

Figure 18 — Compress and observe image 1 at quality level 25

Considering the image in question, it's evident that the quality of smooth regions like the person's coat and the sky remains relatively consistent. However, there is a noticeable loss of fine details in the grass's texture, and the fine lines of the tripod have become less distinct.

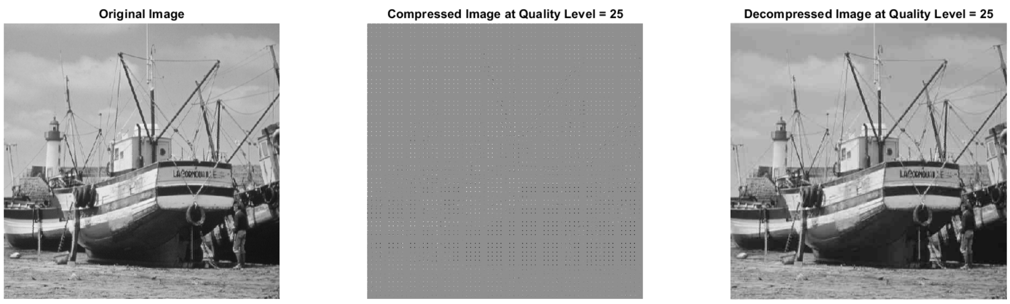

Figure 19 — Compress and observe image 2 at quality level 25

In the image above, the compression has had a more pronounced effect on fine details like the ship's masts and halyards, in contrast to the relatively smooth areas like the sky and the bottom of the ship.

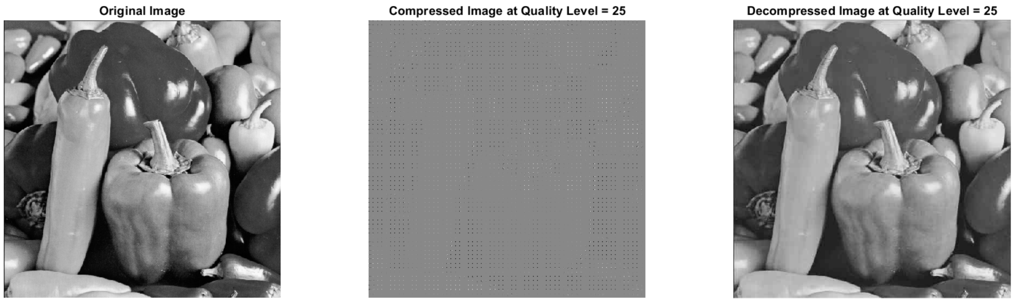

Figure 20 — Compress and observe image 3 quality level 25

This image predominantly consists of fine textures and lacks substantial smooth regions. Consequently, it has suffered significant degradation as a result of compression. This degradation is particularly conspicuous at the edges of the peppers.

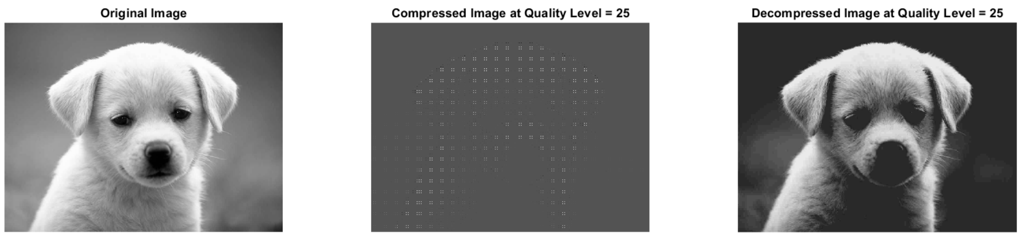

Figure 21 — Compress and observe image 4 quality level 25

In the image depicted above, there is a noticeable preservation of smooth regions, like the solid portions, during the compression process. However, the compression appears to have caused significant degradation in the sharpness of the edges. Additionally, numerous areas within the image containing intricate parts have experienced substantial deterioration as a result of the compression.

1.1.4 Useful MATLAB Commands

- `plot`: plots a function
- `fft`: calculates the DFT
- `dct2`: returns the two-dimensional discrete cosine transform of the input
- `idct2`: returns the two-dimensional inverse discrete cosine transform of the input
- `load`: load contents from a file into a MATLAB workspace.
- `norm`: returns the norm of a vector input.

1.1.5 References

1. Alan Oppenheim, Ronald Schafer, "Discrete Time Signal Processing," Prentice Hall Signal Processing (3rd edition), 2009.
2. S.A. Khayam, "The discrete cosine transform (DCT): Theory and Application," Report, Dept of ECE, Michigan State University, March 2003.
3. K. Cabeen and P. Gent, "Image compression and the discrete cosine transform," College of the Redwoods.
4. A related demo by Berkeley EECS: JPEG DCT Demo

Github Link :- Assignment-02

2.1 Application of 2D-DCT for Image Compression

1. Download 3 images assigned to your index number. You have to select one more image of your choice, which is not in the provided set of images.

```
% Load the images
load('camera256.mat');  % Replace with your image file names
load('boat512.mat');
load('peppers512.mat');
image = imread("dog.jpg");
% Display the images
figure;
subplot(2, 2, 1);
imshow(camera256, []);
title('Camera256');

subplot(2, 2, 2);
imshow(boat512, []);
title('Boat512');

subplot(2, 2, 3);
imshow(peppers512, []);
title('Peppers512');

subplot(2, 2, 4);
imshow(image, []);
title('Dog');
```

Listing 2.1 — Load images

2. Compress each of these images by the DCT-based method described above with three quality levels. The quality levels are also assigned based on your index number.

3. Observe the results in terms of:

- Percentage of zeros
- Quality in terms of peak signal-to-noise ratio (PSNR)
- Visual quality of the compressed images as related to the quality level.

```
clc;
clear;
close all;
```

```matlab

% Loading the required image
load("camera256.mat")
image = camera256;


[rows, cols, ~] = size(image);

% Determine the number of 8x8 blocks needed along each dimension
numBlocksRows = ceil(rows/8);
numBlocksCols = ceil(cols/8);

% Make a copy of the original image before doing operations using it
original_image = image;

% Divide the image into 8x8 blocks
image = mat2cell(image, 8 * ones(1, numBlocksRows), 8 * ones(1,
    numBlocksCols));

% Subtract 128 from every single element
image = cellfun(@(x) x - 128, image, 'UniformOutput', false);

% Apply DCT-2 to each block
image = cellfun(@dct2, image, 'UniformOutput', false);

% Define the quality levels for compression.
qualityLevels = [60, 25, 10];
numQualityLevels = length(qualityLevels);

% Loop through each quality level.
for level = 1:numQualityLevels
    % Calculate the quality factor for this level.
    qualitylevel = qualityLevels(level);
    % Defining Quantization Matrix
    quantizationMatrix = createJPEGQuantizationMatrix(qualitylevel);

    % Quantize each block
    quantizedmatrix = cellfun(@(x)
        quantizeDCTCoefficients(x,quantizationMatrix),image,'UniformOutput'
    ,false);

    % Combine quantizedmatrix into a single image
    quantizedmatrix = cell2mat(quantizedmatrix);
    quantizedmatrix_copy = quantizedmatrix;

```

```
    % Converting quantized matrix into a cell array
    quantizedmatrix = mat2cell(quantizedmatrix,8*ones(1,numBlocksRows),8*ones(1,
    numBlocksCols));

    % Encoding each block
    encodedmatrix = cellfun(@encodeDCTCoefficients,quantizedmatrix,'UniformOutput'
    ,false);

    % Pointwise multiplication of quantizationMatrix and quantizedmatrix
    dequantizedmatrix = cellfun(@(x) x.*quantizationMatrix,quantizedmatrix,'UniformOutput',false);

    % Apply inverse DCT-2 to each block
    level_offed_matrix = cellfun(@idct2,dequantizedmatrix,'UniformOutput',false);

    % Adding 128 to every single element
    decompressed_blocks = cellfun(@(x) x+128,level_offed_matrix,'UniformOutput',false);

    % Combine all blocks into a single image
    decompressed_image = cell2mat(decompressed_blocks);

    % Calculating Zero Percentage in the dequantizedmatrix
    dequantizedmatrix = cell2mat(dequantizedmatrix);
    zero_percentage = (nnz(dequantizedmatrix==0)/numel(dequantizedmatrix))*100;

    % Calculating the maximum light intensity of the original image using antilog of one side of the image
    max_intensity = 2^(log2(rows))-1;


    % Calculating Mean Squared Error by iteratively using mean() over the two axes of a squared error matrix
    % First, ensure that both original_image and decompressed_image have the same data type (e.g., double)
    original_image = double(original_image);
    decompressed_image = double(decompressed_image);

    % Now calculate the Mean Squared Error
```

```matlab
    mean_squared_error = mean(mean((original_image -
       decompressed_image).^2));

    % Calculating Peak Signal-to-Noise Ratio
    peak_signal_to_noise_ratio = 10 * log10((max_intensity^2) /
       mean_squared_error);

    % Display the calculated performance metrics
    temp = "When Quality Level = " + num2str(qualitylevel);
    disp(temp);
    pm1 = "Percentage of Zeros = " + num2str(zero_percentage)+"%";
    disp(pm1);
    pm2 = "Peak Signal-to-Noise Ratio(PSNR) = " +
       num2str(peak_signal_to_noise_ratio)+"dB";
    disp(pm2);
    disp(" ");
    % Displaying Images
    myfigure = figure();
    myfigure.WindowState = 'maximized';

    subplot(1, 3, 1);
    imshow(original_image, []);
    title('Original Image');

    subplot(1, 3, 2);
    imshow(cell2mat(quantizedmatrix), []);
    title_string = "Compressed Image at Quality Level = " +
       num2str(qualitylevel);
    title(title_string);

    subplot(1, 3, 3);
    imshow(decompressed_image, []);
    title_string = "Decompressed Image at Quality Level = " +
       num2str(qualitylevel);
    title(title_string);
end



function quantizationMatrix = createJPEGQuantizationMatrix(quality)
    % Define the standard JPEG quantization matrix for quality level 50
    standardQuantizationMatrix = [
        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
    ];

    % Quality should have values between 1 and 100
    if quality <= 0
        quality = 1;
    elseif quality > 100
        quality = 100;
    end

    % Adjust Quantization Matrix based on the desired quality level
    if quality < 50
        scaleFactor = 50 / quality;
    else
        scaleFactor = 2 - (quality / 50);
    end

    quantizationMatrix = round(standardQuantizationMatrix *
       scaleFactor);

    % Ensure that no values are less than 1
    quantizationMatrix(quantizationMatrix < 1) = 1;
end

% Define a function to quantize the matrix with quantizationMatrix
function quantizedmatrix = quantizeDCTCoefficients(matrix,
   quantizationMatrix)
    % Initialize the quantized matrix
    quantizedmatrix = round(matrix ./ quantizationMatrix);
end

% Define a function to encode [0,0] coefficient using differential PCM
   and encode AC coefficients with (run,level) pairs
function encodedmatrix = encodeDCTCoefficients(matrix)
    % Initialize the encoded matrix
    encodedmatrix = zeros(1, 64);

    % Encode the DC coefficient
    encodedmatrix(1) = matrix(1, 1);

    % Encode the AC coefficients
```

```
    index = 2;
    for i = 1:8
        for j = 1:8
            if i == 1 && j == 1
                continue;
            end
            if matrix(i, j) == 0
                index = index + 1;
            else
                encodedmatrix(index) = matrix(i, j);
                index = index + 1;
            end
        end
    end
end
```

Listing 2.2 — compress and observe image 1 (”camera512”)

```
clc;
clear;
close all;

% Loading the required image
load("boat512.mat")
image = boat512;


[rows, cols, ~] = size(image);

% Determine the number of 8x8 blocks needed along each dimension
numBlocksRows = ceil(rows/8);
numBlocksCols = ceil(cols/8);

% Make a copy of the original image before doing operations using it
original_image = image;

% Divide the image into 8x8 blocks
image = mat2cell(image, 8 * ones(1, numBlocksRows), 8 * ones(1,
    numBlocksCols));

% Subtract 128 from every single element
image = cellfun(@(x) x - 128, image, 'UniformOutput', false);

% Apply DCT-2 to each block
image = cellfun(@dct2, image, 'UniformOutput', false);

```

```
% Define the quality levels for compression.
qualityLevels = [60, 25, 10];
numQualityLevels = length(qualityLevels);

% Loop through each quality level.
for level = 1:numQualityLevels
    % Calculate the quality factor for this level.
    qualitylevel = qualityLevels(level);
    % Defining Quantization Matrix
    quantizationMatrix = createJPEGQuantizationMatrix(qualitylevel);

    % Quantize each block
    quantizedmatrix = cellfun(@(x) quantizeDCTCoefficients(x,quantizationMatrix),image,
    'UniformOutput',false);

    % Combine quantizedmatrix into a single image
    quantizedmatrix = cell2mat(quantizedmatrix);
    quantizedmatrix_copy = quantizedmatrix;

    % Converting quantized matrix into a cell array
    quantizedmatrix = mat2cell(quantizedmatrix,8*ones(1,numBlocksRows),8*ones(1,
    numBlocksCols));

    % Encoding each block
    encodedmatrix = cellfun(@encodeDCTCoefficients,quantizedmatrix,'UniformOutput'
    ,false);

    % Pointwise multiplication of quantizationMatrix and quantizedmatrix
    dequantizedmatrix = cellfun(@(x) x.*quantizationMatrix,quantizedmatrix,'UniformOutput',false);

    % Apply inverse DCT-2 to each block
    level_offed_matrix = cellfun(@idct2,dequantizedmatrix,'UniformOutput',false);

    % Adding 128 to every single element
    decompressed_blocks = cellfun(@(x) x+128,level_offed_matrix,'UniformOutput',false);

    % Combine all blocks into a single image
    decompressed_image = cell2mat(decompressed_blocks);
```

```
% Calculating Zero Percentage in the dequantizedmatrix
dequantizedmatrix = cell2mat(dequantizedmatrix);
zero_percentage =
    (nnz(dequantizedmatrix==0)/numel(dequantizedmatrix))*100;

% Calculating the maximum light intensity of the original image
    using antilog of one side of the image
max_intensity = 2^(log2(rows))-1;


% Calculating Mean Squared Error by iteratively using mean() over
    the two axes of a squared error matrix
% First, ensure that both original_image and decompressed_image
    have the same data type (e.g., double)
original_image = double(original_image);
decompressed_image = double(decompressed_image);

% Now calculate the Mean Squared Error
mean_squared_error = mean(mean((original_image -
    decompressed_image).^2));

% Calculating Peak Signal-to-Noise Ratio
peak_signal_to_noise_ratio = 10 * log10((max_intensity^2) /
    mean_squared_error);

% Display the calculated performance metrics
temp = "When Quality Level = " + num2str(qualitylevel);
disp(temp);
pm1 = "Percentage of Zeros = " + num2str(zero_percentage)+"%";
disp(pm1);
pm2 = "Peak Signal-to-Noise Ratio(PSNR) = " +
    num2str(peak_signal_to_noise_ratio)+"dB";
disp(pm2);
disp(" ");
% Displaying Images
myfigure = figure();
myfigure.WindowState = 'maximized';

subplot(1, 3, 1);
imshow(original_image, []);
title('Original Image');

subplot(1, 3, 2);
imshow(cell2mat(quantizedmatrix), []);
```

```
    title_string = "Compressed Image at Quality Level = " +
        num2str(qualitylevel);
    title(title_string);

    subplot(1, 3, 3);
    imshow(decompressed_image, []);
    title_string = "Decompressed Image at Quality Level = " +
        num2str(qualitylevel);
    title(title_string);
end



function quantizationMatrix = createJPEGQuantizationMatrix(quality)
    % Define the standard JPEG quantization matrix for quality level 50
    standardQuantizationMatrix = [
        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
    ];

    % Quality should have values between 1 and 100
    if quality <= 0
        quality = 1;
    elseif quality > 100
        quality = 100;
    end

    % Adjust Quantization Matrix based on the desired quality level
    if quality < 50
        scaleFactor = 50 / quality;
    else
        scaleFactor = 2 - (quality / 50);
    end

    quantizationMatrix = round(standardQuantizationMatrix *
        scaleFactor);

    % Ensure that no values are less than 1
    quantizationMatrix(quantizationMatrix < 1) = 1;
```

```
end

% Define a function to quantize the matrix with quantizationMatrix
function quantizedmatrix = quantizeDCTCoefficients(matrix,
    quantizationMatrix)
    % Initialize the quantized matrix
    quantizedmatrix = round(matrix ./ quantizationMatrix);
end

% Define a function to encode [0,0] coefficient using differential PCM
    and encode AC coefficients with (run,level) pairs
function encodedmatrix = encodeDCTCoefficients(matrix)
    % Initialize the encoded matrix
    encodedmatrix = zeros(1, 64);

    % Encode the DC coefficient
    encodedmatrix(1) = matrix(1, 1);

    % Encode the AC coefficients
    index = 2;
    for i = 1:8
        for j = 1:8
            if i == 1 && j == 1
                continue;
            end
            if matrix(i, j) == 0
                index = index + 1;
            else
                encodedmatrix(index) = matrix(i, j);
                index = index + 1;
            end
        end
    end
end
```

Listing 2.3 — compress and observe image 2 ("boat512")

```
clc;
clear;
close all;

% Loading the required image
load("peppers512.mat")
image = peppers512;

[rows, cols, ~] = size(image);
```

```
% Determine the number of 8x8 blocks needed along each dimension
numBlocksRows = ceil(rows/8);
numBlocksCols = ceil(cols/8);

% Make a copy of the original image before doing operations using it
original_image = image;

% Divide the image into 8x8 blocks
image = mat2cell(image, 8 * ones(1, numBlocksRows), 8 * ones(1,
    numBlocksCols));

% Subtract 128 from every single element
image = cellfun(@(x) x - 128, image, 'UniformOutput', false);

% Apply DCT-2 to each block
image = cellfun(@dct2, image, 'UniformOutput', false);

% Define the quality levels for compression.
qualityLevels = [60, 25, 10];
numQualityLevels = length(qualityLevels);

% Loop through each quality level.
for level = 1:numQualityLevels
    % Calculate the quality factor for this level.
    qualitylevel = qualityLevels(level);
    % Defining Quantization Matrix
    quantizationMatrix = createJPEGQuantizationMatrix(qualitylevel);

    % Quantize each block
    quantizedmatrix = cellfun(@(x)
        quantizeDCTCoefficients(x,quantizationMatrix),image,
    'UniformOutput',false);

    % Combine quantizedmatrix into a single image
    quantizedmatrix = cell2mat(quantizedmatrix);
    quantizedmatrix_copy = quantizedmatrix;

    % Converting quantized matrix into a cell array
    quantizedmatrix =
        mat2cell(quantizedmatrix,8*ones(1,numBlocksRows),8*ones(1,
    numBlocksCols));

    % Encoding each block
```

```
    encodedmatrix = cellfun(@encodeDCTCoefficients,quantizedmatrix,'UniformOutput'
    ,false);

    % Pointwise multiplication of quantizationMatrix and quantizedmatrix
    dequantizedmatrix = cellfun(@(x) x.*quantizationMatrix,quantizedmatrix,'UniformOutput',false);

    % Apply inverse DCT-2 to each block
    level_offed_matrix = cellfun(@idct2,dequantizedmatrix,'UniformOutput',false);

    % Adding 128 to every single element
    decompressed_blocks = cellfun(@(x) x+128,level_offed_matrix,'UniformOutput',false);

    % Combine all blocks into a single image
    decompressed_image = cell2mat(decompressed_blocks);

    % Calculating Zero Percentage in the dequantizedmatrix
    dequantizedmatrix = cell2mat(dequantizedmatrix);
    zero_percentage = (nnz(dequantizedmatrix==0)/numel(dequantizedmatrix))*100;

    % Calculating the maximum light intensity of the original image using antilog of one side of the image
    max_intensity = 2^(log2(rows))-1;


    % Calculating Mean Squared Error by iteratively using mean() over the two axes of a squared error matrix
    % First, ensure that both original_image and decompressed_image have the same data type (e.g., double)
    original_image = double(original_image);
    decompressed_image = double(decompressed_image);

    % Now calculate the Mean Squared Error
    mean_squared_error = mean(mean((original_image - decompressed_image).^2));

    % Calculating Peak Signal-to-Noise Ratio
    peak_signal_to_noise_ratio = 10 * log10((max_intensity^2) / mean_squared_error);

```

```
    % Display the calculated performance metrics
    temp = "When Quality Level = " + num2str(qualitylevel);
    disp(temp);
    pm1 = "Percentage of Zeros = " + num2str(zero_percentage)+"%";
    disp(pm1);
    pm2 = "Peak Signal-to-Noise Ratio(PSNR) = " +
       num2str(peak_signal_to_noise_ratio)+"dB";
    disp(pm2);
    disp(" ");
    % Displaying Images
    myfigure = figure();
    myfigure.WindowState = 'maximized';

    subplot(1, 3, 1);
    imshow(original_image, []);
    title('Original Image');

    subplot(1, 3, 2);
    imshow(cell2mat(quantizedmatrix), []);
    title_string = "Compressed Image at Quality Level = " +
       num2str(qualitylevel);
    title(title_string);

    subplot(1, 3, 3);
    imshow(decompressed_image, []);
    title_string = "Decompressed Image at Quality Level = " +
       num2str(qualitylevel);
    title(title_string);
end



function quantizationMatrix = createJPEGQuantizationMatrix(quality)
    % Define the standard JPEG quantization matrix for quality level 50
    standardQuantizationMatrix = [
        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
    ];

```

```
    % Quality should have values between 1 and 100
    if quality <= 0
        quality = 1;
    elseif quality > 100
        quality = 100;
    end

    % Adjust Quantization Matrix based on the desired quality level
    if quality < 50
        scaleFactor = 50 / quality;
    else
        scaleFactor = 2 - (quality / 50);
    end

    quantizationMatrix = round(standardQuantizationMatrix * scaleFactor);

    % Ensure that no values are less than 1
    quantizationMatrix(quantizationMatrix < 1) = 1;
end

% Define a function to quantize the matrix with quantizationMatrix
function quantizedmatrix = quantizeDCTCoefficients(matrix, quantizationMatrix)
    % Initialize the quantized matrix
    quantizedmatrix = round(matrix ./ quantizationMatrix);
end

% Define a function to encode [0,0] coefficient using differential PCM and encode AC coefficients with (run,level) pairs
function encodedmatrix = encodeDCTCoefficients(matrix)
    % Initialize the encoded matrix
    encodedmatrix = zeros(1, 64);

    % Encode the DC coefficient
    encodedmatrix(1) = matrix(1, 1);

    % Encode the AC coefficients
    index = 2;
    for i = 1:8
        for j = 1:8
            if i == 1 && j == 1
                continue;
            end
            if matrix(i, j) == 0
```

```
                    index = index + 1;
               else
                    encodedmatrix(index) = matrix(i, j);
                    index = index + 1;
               end
          end
     end
end
```

Listing 2.4 — compress and observe image 3 (”peppers512”)

```
clc;
clear;
close all;

% Loading the required image
image = imread("dog.jpg");
[rows, cols, ~] = size(image);

% Determine the number of 8x8 blocks needed along each dimension
numBlocksRows = ceil(rows/8);
numBlocksCols = ceil(cols/8);

% Make a copy of the original image before doing operations using it
original_image = image;

% Divide the image into 8x8 blocks
image = mat2cell(image, 8 * ones(1, numBlocksRows), 8 * ones(1,
    numBlocksCols));

% Subtract 128 from every single element
image = cellfun(@(x) x - 128, image, 'UniformOutput', false);

% Apply DCT-2 to each block
image = cellfun(@dct2, image, 'UniformOutput', false);

% Define the quality levels for compression.
qualityLevels = [60, 25, 10];
numQualityLevels = length(qualityLevels);

% Loop through each quality level.
for level = 1:numQualityLevels
     % Calculate the quality factor for this level.
     qualitylevel = qualityLevels(level);
     % Defining Quantization Matrix
     quantizationMatrix = createJPEGQuantizationMatrix(qualitylevel);
```

```

% Quantize each block
quantizedmatrix = cellfun(@(x) quantizeDCTCoefficients(x,quantizationMatrix),image,
'UniformOutput',false);

% Combine quantizedmatrix into a single image
quantizedmatrix = cell2mat(quantizedmatrix);
quantizedmatrix_copy = quantizedmatrix;

% Converting quantized matrix into a cell array
quantizedmatrix = mat2cell(quantizedmatrix,8*ones(1,numBlocksRows),8*ones(1,
numBlocksCols));

% Encoding each block
encodedmatrix = cellfun(@encodeDCTCoefficients,quantizedmatrix,'UniformOutput'
,false);

% Pointwise multiplication of quantizationMatrix and quantizedmatrix
dequantizedmatrix = cellfun(@(x) x.*quantizationMatrix,quantizedmatrix,'UniformOutput',false);

% Apply inverse DCT-2 to each block
level_offed_matrix = cellfun(@idct2,dequantizedmatrix,'UniformOutput',false);

% Adding 128 to every single element
decompressed_blocks = cellfun(@(x) x+128,level_offed_matrix,'UniformOutput',false);

% Combine all blocks into a single image
decompressed_image = cell2mat(decompressed_blocks);

% Calculating Zero Percentage in the dequantizedmatrix
dequantizedmatrix = cell2mat(dequantizedmatrix);
zero_percentage = (nnz(dequantizedmatrix==0)/numel(dequantizedmatrix))*100;

% Calculating the maximum light intensity of the original image using antilog of one side of the image
max_intensity = 2^(log2(rows))-1;

```

```

    % Calculating Mean Squared Error by iteratively using mean() over
       the two axes of a squared error matrix
    % First, ensure that both original_image and decompressed_image
       have the same data type (e.g., double)
    original_image = double(original_image);
    decompressed_image = double(decompressed_image);

    % Now calculate the Mean Squared Error
    mean_squared_error = mean(mean((original_image -
       decompressed_image).^2));

    % Calculating Peak Signal-to-Noise Ratio
    peak_signal_to_noise_ratio = 10 * log10((max_intensity^2) /
       mean_squared_error);

    % Display the calculated performance metrics
    temp = "When Quality Level = " + num2str(qualitylevel);
    disp(temp);
    pm1 = "Percentage of Zeros = " + num2str(zero_percentage)+"%";
    disp(pm1);
    pm2 = "Peak Signal-to-Noise Ratio(PSNR) = " +
       num2str(peak_signal_to_noise_ratio)+"dB";
    disp(pm2);
    disp(" ");
    % Displaying Images
    myfigure = figure();
    myfigure.WindowState = 'maximized';

    subplot(1, 3, 1);
    imshow(original_image, []);
    title('Original Image');

    subplot(1, 3, 2);
    imshow(cell2mat(quantizedmatrix), []);
    title_string = "Compressed Image at Quality Level = " +
       num2str(qualitylevel);
    title(title_string);

    subplot(1, 3, 3);
    imshow(decompressed_image, []);
    title_string = "Decompressed Image at Quality Level = " +
       num2str(qualitylevel);
    title(title_string);
end
```

```
function quantizationMatrix = createJPEGQuantizationMatrix(quality)
    % Define the standard JPEG quantization matrix for quality level 50
    standardQuantizationMatrix = [
        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
    ];

    % Quality should have values between 1 and 100
    if quality <= 0
        quality = 1;
    elseif quality > 100
        quality = 100;
    end

    % Adjust Quantization Matrix based on the desired quality level
    if quality < 50
        scaleFactor = 50 / quality;
    else
        scaleFactor = 2 - (quality / 50);
    end

    quantizationMatrix = round(standardQuantizationMatrix *
       scaleFactor);

    % Ensure that no values are less than 1
    quantizationMatrix(quantizationMatrix < 1) = 1;
end

% Define a function to quantize the matrix with quantizationMatrix
function quantizedmatrix = quantizeDCTCoefficients(matrix,
   quantizationMatrix)
    % Initialize the quantized matrix
    quantizedmatrix = round(matrix ./ quantizationMatrix);
end
```

```
% Define a function to encode [0,0] coefficient using differential PCM
    and encode AC coefficients with (run,level) pairs
function encodedmatrix = encodeDCTCoefficients(matrix)
    % Initialize the encoded matrix
    encodedmatrix = zeros(1, 64);

    % Encode the DC coefficient
    encodedmatrix(1) = matrix(1, 1);

    % Encode the AC coefficients
    index = 2;
    for i = 1:8
        for j = 1:8
            if i == 1 && j == 1
                continue;
            end
            if matrix(i, j) == 0
                index = index + 1;
            else
                encodedmatrix(index) = matrix(i, j);
                index = index + 1;
            end
        end
    end
end
```

Listing 2.5 — compress and observe image 4(”dog”)

***