Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN3551 - Digital Signal Processing

Application of 2D-DCT for image compression

Name	Index Number
Kariyawasam K.K.D.	200289U

Task 01

For this image compression assignment used images are as follow.

(a) *camera256*

(b) *boat512*

(c) *goldhill512*

(d) *columbia*

Figure 1: *Set of images used*

Task 02

In this task, we applied a Discrete Cosine Transform (DCT)-based image compression method to a set of four images. We executed the compression process using three distinct quality levels 80, 40 and 15. The quality levels affected the compression outcome, resulting in varying levels of image degradation. The Peak Signal-to-Noise Ratio (PSNR) was calculated and by analyzing PSNR values, we evaluated the trade-off between image quality and compression level for each image.

Step 01: Level off and 2-D DCT

In the first step the image was divided into 8x8 blocks and then pixel values are adjusted within the range -128 to 127 by subtracting 128 from each pixel's value. This process is call level-off and this was done due to DCT gives a lower DC coefficient when pixel values are in the range -128 to 127 rather than in the range 0 to 255. For an example consider first 8x8 block of "camera256" image.

$$B = \begin{bmatrix} 156 & 159 & 158 & 155 & 158 & 156 & 159 & 158 \\ 160 & 154 & 157 & 158 & 157 & 159 & 158 & 158 \\ 156 & 159 & 158 & 155 & 158 & 156 & 159 & 158 \\ 160 & 154 & 157 & 158 & 157 & 159 & 158 & 158 \\ 156 & 153 & 155 & 159 & 159 & 155 & 156 & 155 \\ 155 & 155 & 155 & 157 & 156 & 159 & 152 & 158 \\ 156 & 153 & 157 & 156 & 153 & 155 & 154 & 155 \\ 159 & 159 & 156 & 158 & 156 & 159 & 157 & 161 \end{bmatrix}$$

After levelled off

$$\hat{B} = \begin{bmatrix} 28 & 31 & 30 & 27 & 30 & 28 & 31 & 30 \\ 32 & 26 & 29 & 30 & 29 & 31 & 30 & 30 \\ 28 & 31 & 30 & 27 & 30 & 28 & 31 & 30 \\ 32 & 26 & 29 & 30 & 29 & 31 & 30 & 30 \\ 28 & 25 & 27 & 31 & 31 & 27 & 28 & 27 \\ 27 & 27 & 27 & 29 & 28 & 31 & 24 & 30 \\ 28 & 25 & 29 & 28 & 25 & 27 & 26 & 27 \\ 31 & 31 & 28 & 30 & 28 & 31 & 29 & 33 \end{bmatrix}$$

Then 2-D DCT applied on each block using the Matlab function **dct2**.

$$C = \begin{bmatrix} 230.8750 & -1.9711 & 0.6581 & 0.4709 & 2.3750 & 1.0406 & 2.5687 & -0.4977 \\ 3.8203 & -0.7165 & 0.5280 & 0.7221 & -1.9537 & 0.2029 & -2.2978 & 4.0017 \\ 2.0439 & 0.5134 & 3.3650 & -1.4406 & -2.3705 & -1.5856 & -0.4848 & -0.6546 \\ -4.9979 & 0.0819 & -3.1677 & -0.4627 & 0.9110 & -0.9090 & 0.5341 & -0.2885 \\ 3.3750 & -0.4348 & 0.3218 & 0.6203 & 0.8750 & 0.2899 & -2.1628 & 0.5289 \\ -2.0790 & 0.9224 & 1.1297 & -1.4267 & -2.5590 & 0.0485 & 0.3766 & 1.4228 \\ 1.9947 & -0.7280 & 1.0152 & -1.3862 & -2.1300 & -5.4382 & -2.1150 & -0.7537 \\ -2.8577 & 0.9839 & -1.2675 & -2.0439 & -1.2717 & -0.4118 & -3.8042 & 3.1307 \end{bmatrix}$$

The following code snippet was used in this step. Note that full code can be found in the appendix.

```matlab
% Dividing to 8x8 blocks
B = image(i:i+7, j:j+7);
% Level off by substracting 128 from each entry
B_hat = B - 128;
% Apply DCT
C = dct2(B_hat);
```

Step 02: Quantization

The quantization is achieved by converting the C matrix to S matrix by dividing coefficients in C matrix by corresponding coefficient in quantization matrix and rounded to integer values. Quantization matrix is calculated as below.

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

Scaling factors for quality levels 80, 40 and 15 are 0.4, 1.25 and 3.33 respectively. To obtain the quantization matrices corresponding to above quality levels Q_{50} was multiplied by those quality factors.

The S matrix correspond to the above C matrix with quality level 80 is as follows.

$$S = \begin{bmatrix} 36 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Step 03: Decompression

In this step the image is reconstructed for that first pointwise multiplication of matrix S and matrix Q was computed using **R = Q.*S;** this Matlab code line.

$$R = \begin{bmatrix} 230.4000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4.8000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6.4000 & 0 & 0 & 0 & 0 & 0 \\ -5.6000 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Then 2-D inverse DCT was applied to R using Matlab function **idct2**. Finally, 128 was added to each entry of E matrix to compensate the effect of level off. F matrix correspond to above R matrix is shown below.

$$F = \begin{bmatrix} 158.1748 & 157.3748 & 156.2434 & 155.4434 & 155.4434 & 156.2434 & 157.3748 & 158.1748 \\ 158.2643 & 157.9330 & 157.4643 & 157.1330 & 157.1330 & 157.4643 & 157.9330 & 158.2643 \\ 157.6767 & 158.0080 & 158.4767 & 158.8080 & 158.8080 & 158.4767 & 158.0080 & 157.6767 \\ 156.1498 & 156.9498 & 158.0812 & 158.8812 & 158.8812 & 158.0812 & 156.9498 & 156.1498 \\ 154.7188 & 155.5188 & 156.6502 & 157.4502 & 157.4502 & 156.6502 & 155.5188 & 154.7188 \\ 154.7920 & 155.1233 & 155.5920 & 155.9233 & 155.9233 & 155.5920 & 155.1233 & 154.7920 \\ 156.4670 & 156.1357 & 155.6670 & 155.3357 & 155.3357 & 155.6670 & 156.1357 & 156.4670 \\ 158.1566 & 157.3566 & 156.2252 & 155.4252 & 155.4252 & 156.2252 & 157.3566 & 158.1566 \end{bmatrix}$$

When comparing the matrix B and matrix F we can see they are approximately equal.

Results obtained

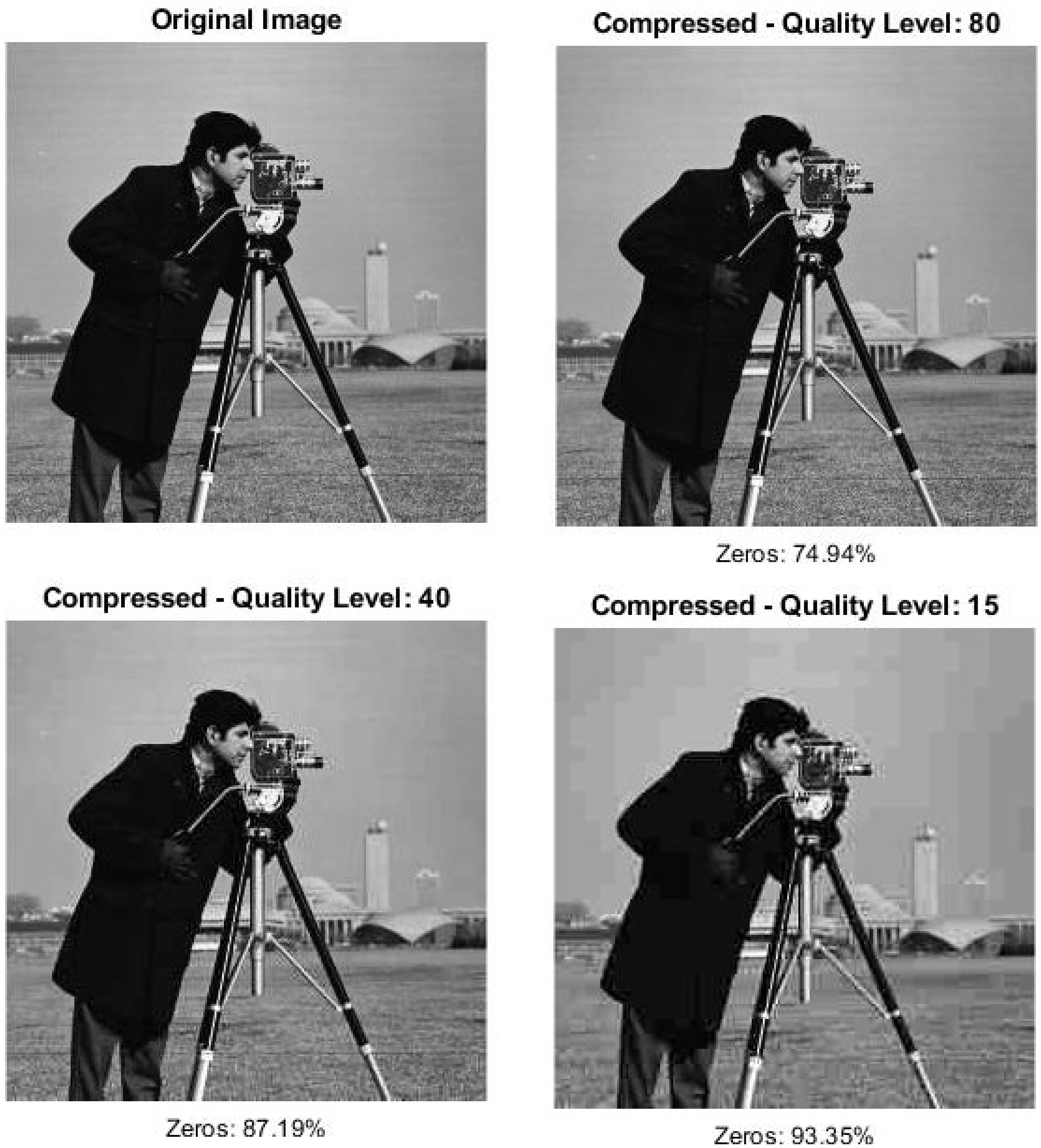

Figure 2: *camera256 image*

PSNR values

Quality level 80 : 35.8017
Quality level 40 : 30.7854
Quality level 15 : 27.5293

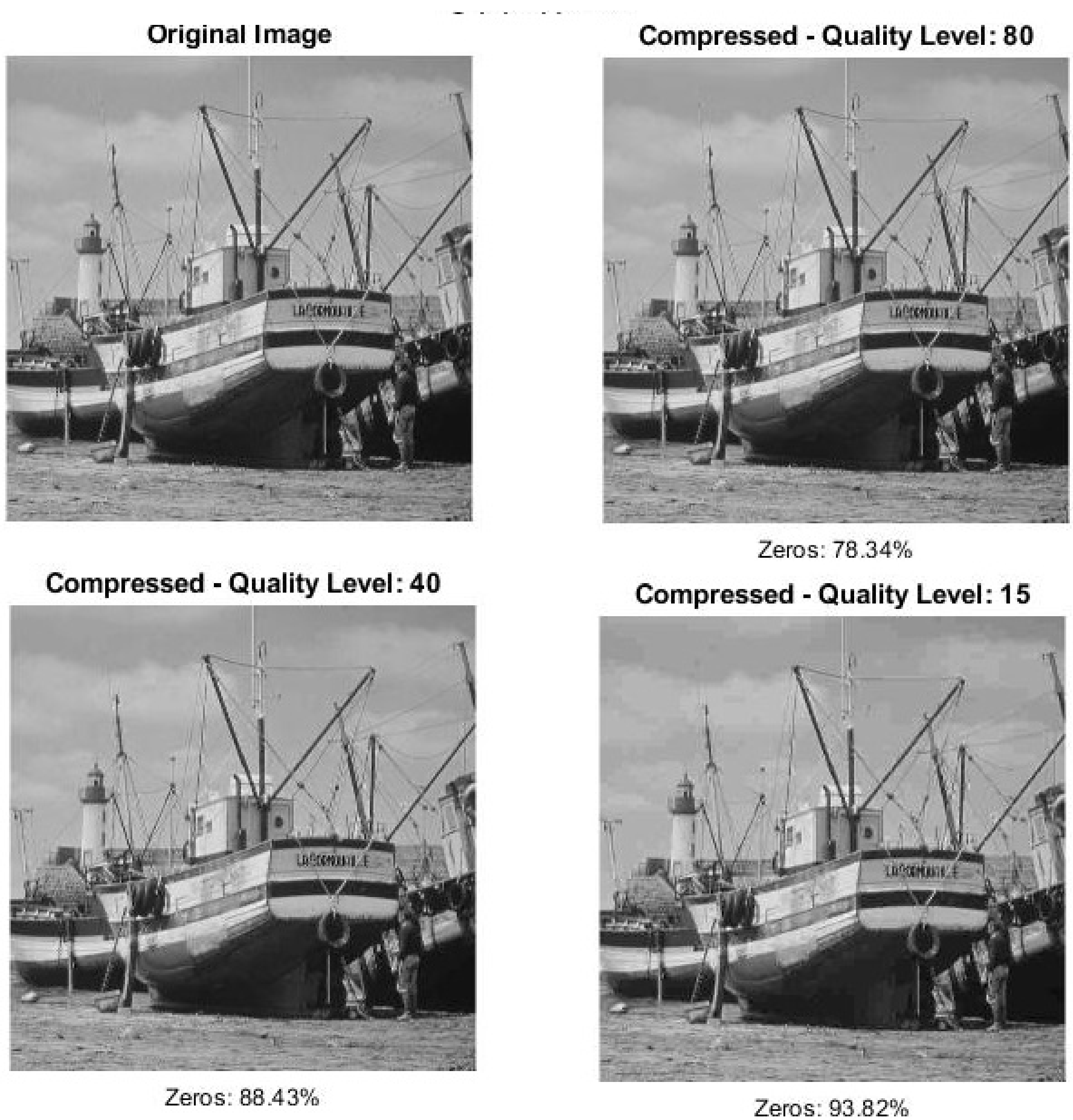

Figure 3: *boat512 image*

PSNR values

Quality level 80 : 38.1985
Quality level 40 : 33.8941
Quality level 15 : 30.3580

Figure 4: *goldhill512 image*

PSNR values

Quality level 80 : 36.5395
Quality level 40 : 32.9299
Quality level 15 : 29.9526

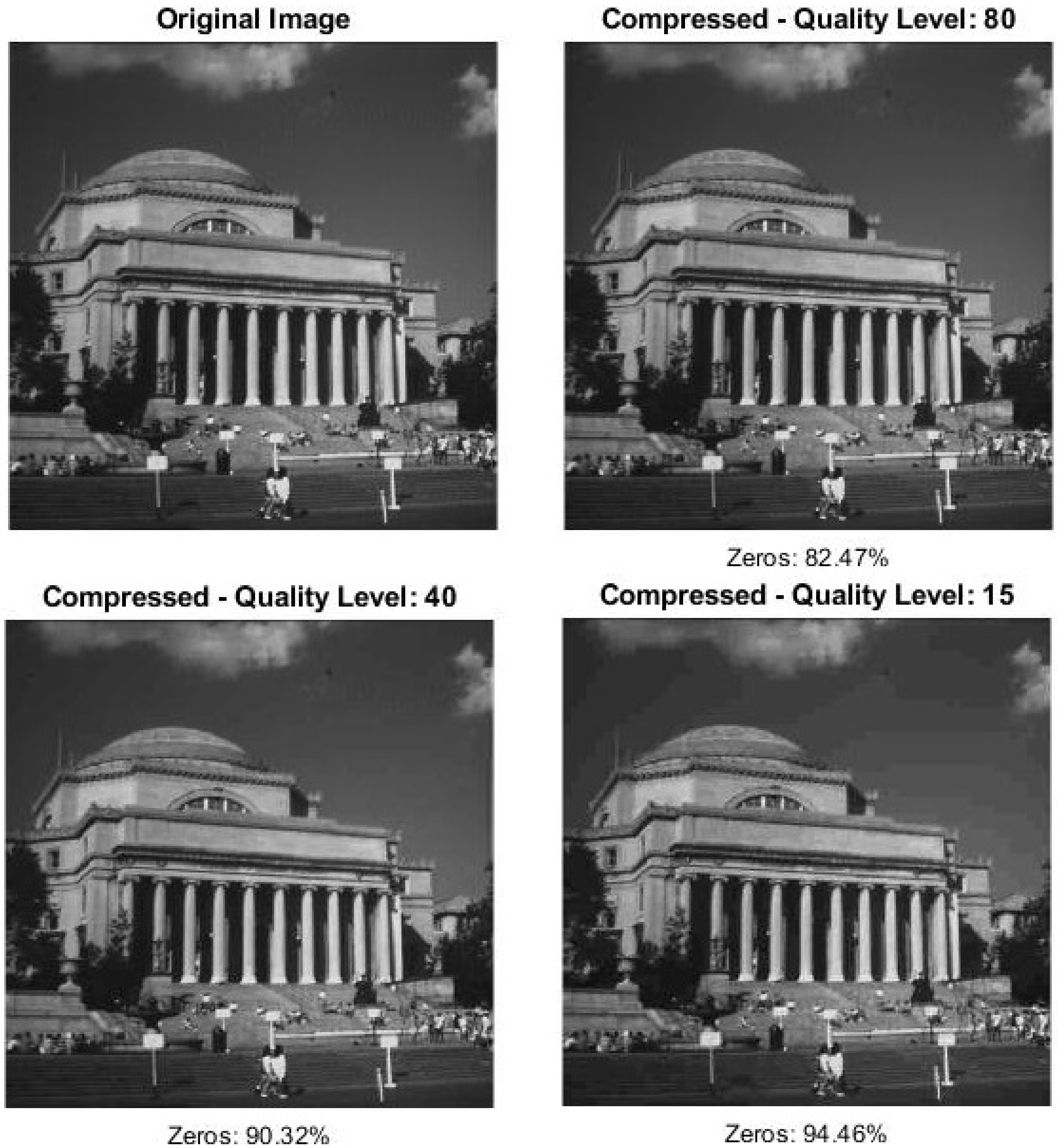

Figure 5: *columbia image*

PSNR values

Quality level 80 : 41.5308
Quality level 40 : 37.1680
Quality level 15 : 32.8086

Task 03

Percentage of zeros

Table 1: *Percentage of zeros table*

Image	Quality 80	Quality 40	Quality 15
camera256	74.94%	87.19%	93.35%
boat512	78.34%	88.43%	93.82%
goldhill512	73.55%	86.78%	93.80%
columbia	82.47%	90.32%	94.46%

According to the results obtained, as the quality level decreases (from 80 to 40 and 15), the percentage of zeros in the compressed images increases for all the test images. This is an expected outcome, as lower quality levels lead to stronger quantization, causing more coefficients to be rounded to zero. The trade-off is that more aggressive compression results in reduced image quality.

Different images show different percentages at the same quality levels. For instance, "columbia" image has a higher percentage of zeros compared to other images, which means that it is more compressible without a significant loss in perceived quality. On the other hand, "goldhill512" and "camera256" exhibit slightly lower sparsity at all quality levels, indicating that they may harder to compress.

Quality in terms of peak signal-to-noise ratio

The PSNR is defined as below,

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

The code snippet implemented to calculate the PSNR value.

```
max_intensity = 255;

% Calculate the PSNR value
psnr = 20 * log10(max_intensity / sqrt(mse));
psnr_values(:,k) = psnr;
```

Table 2: *PSNR value table*

Image	Quality 80	Quality 40	Quality 15
camera256	35.8017	30.7854	27.5293
boat512	38.1985	33.8941	30.3580
goldhill512	36.5395	32.9299	29.9526
columbia	41.5308	37.1680	32.8086

As expected, the PSNR values decrease as the quality level decreases. Higher quality levels result in higher PSNR values, indicating better image quality, while lower quality levels result in lower PSNR values, indicating more significant compression artifacts.

Visual quality of the compressed images as related to the quality level.

Using a higher quality level, such as 80, results in better visual quality. High-quality compression retains fine details, textures, and color variations.

A moderate quality level, such as 40, strikes a balance between compression and visual quality. It offers a reasonable level of compression while maintaining acceptable visual fidelity. Intermediate quality levels may lead to some loss of fine details and minor compression artifacts.

Choosing a lower quality level, such as 15, results in a significant reduction in visual quality than higher quality levels. Lower quality levels often lead to the loss of fine details, sharpness, and color accuracy.

Task 04

There can be several reasons why 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.

- Images with complex content, intricate patterns, and a wide range of colors and details are often more challenging to compress without visible artifacts.
- Images that contain fine details, subtle gradients, and textures are more susceptible to compression artifacts. These details are often the first to be compromised in lossy compression.
- Images with geometric shapes, sharp edges, or high-frequency components can exhibit artifacts in the form of blockiness or ringing effects when compressed at lower quality levels.
- Images containing human faces are particularly sensitive to compression. Compression artifacts in facial features can be quite noticeable and negatively impact the perceived quality of the image.

A Appendix

A.1 Matlab code

```matlab
% Load the given data
load("Data/SampleImages/camera256.mat");
load("Data/SampleImages/boat512.mat");
load("Data/SampleImages/goldhill512.mat");

% Load the 4 th image
imageMatrix = imread("columbia.tif");
imageMatrix = double(imageMatrix);

quality_levels = [80, 40, 15];

psnr_cam = compression(camera256, quality_levels);
psnr_boat = compression(boat512, quality_levels);
psnr_ghill = compression(goldhill512, quality_levels);
psnr_choice = compression(imageMatrix, quality_levels);

display(psnr_cam);
display(psnr_boat);
display(psnr_ghill);
display(psnr_choice);

function psnr_values = compression(image, quality_levels)
    n = length(quality_levels);
    psnr_values = zeros(1,n);

    figure;
    % Plot the first image in the first subplot
    subaxis(2, 2, 1, 'Spacing', 0.1, 'Padding', 0.02, 'Margin', 0.01);
    imshow(uint8(image));
    title('Original Image');

    % Iterate with every quality level
    for k = 1:n
        quality_level = quality_levels(k);  % Take the quality level
        [M, N] = size(image);
        block_size = 8;              % Define the block size

        % Quantization matrix for quality level 50
        Q_50 = [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];

        % Calculate the quality factor
        if quality_level > 50
            tau = (100 - quality_level)/50;
        else
            tau = 50/quality_level;
```

```
        end

        % Quantization matrix
        Q = tau*Q_50;
        numZeros = 0;

        recon_image = zeros(M,N);   % Create a matrix to store reconstructed image

        for i = 1:block_size:M
            for j = 1:block_size:N
                % Dividing to 8x8 blocks
                B = image(i:i+7, j:j+7);
                % Level off by substracting 128 from each entry
                B_hat = B - 128;
                % Apply DCT
                C = dct2(B_hat);
                % Perform quantization
                S = round(C ./ Q);
                % Calculate number of zeros
                numZeros = numZeros + sum(S(:) == 0);
                % Decompression
                R = Q.*S;
                % Calculate inverse 2-D DCT
                E = idct2(R);
                F = E + 128;
                recon_image(i:i+block_size-1, j:j+block_size-1) = F;
            end
        end
        zero = (numZeros/(M*N))*100;
        % Calculate the error matrix
        error_mat = recon_image - image;
        % Calculate the squared error matrix
        squared_error = (error_mat).^2;
        % Calculate the mean squared error (MSE)
        mse = mean(squared_error(:));

        max_intensity = 255;

        % Calculate the PSNR value
        psnr = 20 * log10(max_intensity / sqrt(mse));
        psnr_values(:,k) = psnr;

        % Plotting the images
        subaxis(2, 2, k + 1, 'Spacing', 0.1, 'Padding', 0.02, 'Margin', 0.01);
        combined_title = sprintf('Compressed - Quality Level: %d', quality_level);
        imshow(uint8(recon_image));
        title(combined_title);
        text(0.5, -0.05, sprintf('Zeros: %.2f%%', zero), 'Units', 'normalized', '
            HorizontalAlignment', 'center');
    end
end
```