Assignment 02

submitted for

EN3551 - Digital Signal Processing

Department of Electronic and Telecommunication Engineering
University of Moratuwa

Udugamasooriya P. H. J.
220658U
Progress on GitHub

26 October 2025

Question 03 The relevant images and quality levels are `cameraman.mat`, `Monarch.mat`, `Parrot.mat`, and 80%, 30% and 15%. We compress and decompress each image and compare with the original image at each of the quality levels given.

Note that we skip the process of coding and decoding, as we are only interested in the effects of compression on the visual quality of the image.

Results obtained for each image are as follows:

1. `cameraman.mat`

Figure 1: `cameraman.mat` at various compression levels

Quality Level	80%	30%	15%
Percentage of Zeros	74.9786%	89.386%	93.3502%
PSNR (dB)	35.7235	29.6951	27.4598

Comments on visual quality:

- 80% compression
  - virtually indistinguishable from the original

  - slight reduction in contrast levels, e.g., the coat is grayer than in the original
- 30% compression
  - slight blockiness is visible, especially towards the upper right
  - some ringing is visible at the edges; there is a slight halo around the cameraman and camera
  - the grass on the ground appears blurry
- 15% compression
  - significant blockiness is visible
  - ringing effects are visible—there is a blurry halo around the cameraman and camera
  - the grass on the ground is significantly more blurry
  - there is a clear reduction in contrast levels

2. `Monarch.mat`

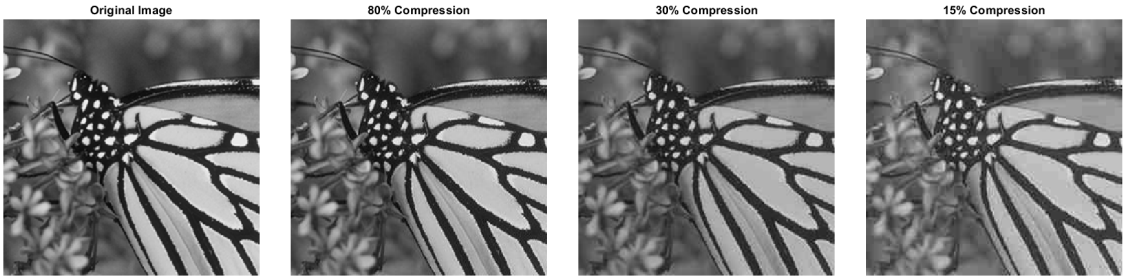

Figure 2: `Monarch.mat` at various compression levels

Quality Level	80%	30%	15%
Percentage of Zeros	71.8948%	85.7666%	90.2161%
PSNR (dB)	35.7556	30.0778	27.725

Comments on visual quality:
- 80% compression
  - virtually indistinguishable from the original
  - slight reduction in contrast levels, e.g., the black lines in the wings seem a lighter shade of gray and the brighter, white regions seem more dull
- 30% compression
  - blockiness is visible
  - further reduction in contrast levels
- 15% compression
  - significant blockiness is visible
  - the outlines around the white regions on the wings are no longer sharp and have blurred out

3. `Parrot.mat`

Figure 3: `Parrot.mat` at various compression levels

Quality Level	80%	30%	15%
Percentage of Zeros	80.5573%	91.2155%	94.1544%
PSNR (dB)	38.2348	32.9118	30.4194

Comments on visual quality:

- 80% compression
  - very similar to the original
  - fine details of the feathers have diminished
- 30% compression
  - further blurring of the details of feathers
  - some blockiness visible
- 15% compression
  - significant blockiness is visible
  - significant loss of detail of the feathers; the body seems made up of plain colors
  - slight halo visible near the back of the neck

4. Image of Choice - `pisa.jpg`

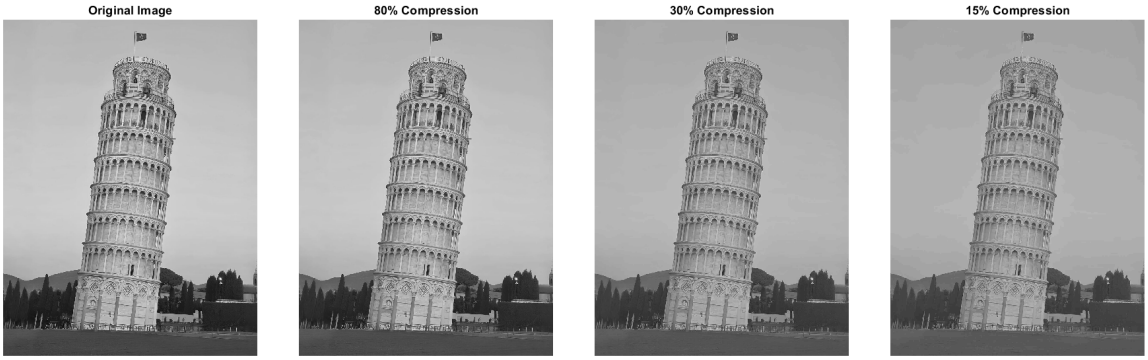

Figure 4: `pisa.jpg` at various compression levels

Quality Level	80%	30%	15%
Percentage of Zeros	87.8315%	93.783%	95.6683%
PSNR (dB)	42.1905	34.3926	31.6468

Comments on visual quality:

- 80% compression
  - virtually indistinguishable from the original
- 30% compression
  - walls seem smoothed out
  - fine details of carvings and engravings are not visible and are blurry
- 15% compression
  - further loss of fine details due to blurring out
  - bands visible in the sky instead of a smooth transition between colors
  - some blockiness visible in the grass

We wish to also make the following general comments:

- The quantity σ_e is a measure of the average variability/error between the original image and reconstructed image.

  The PSNR measures how σ_e compares with the maximum intensity level ψ_{max} in the image.

  If $\frac{\psi_{\text{max}}}{\sigma_e}$ is high, then we expect the intensity levels in the reconstructed image to remain very close to those of the original image, but to deviate more significantly otherwise.

  In each of the images, the PSNR can be seen to decrease with decreasing quality level, as expected.
- In each of the cases, even with a quality level as high as 80%, we see that the algorithm has managed to reduce more than 70% of the entries in the matrix to zeros, giving a very significant compression at negligible loss of visual quality.

Question 04 We will compare the compressed images over the quality level 15%.

Across the three given images, it seems that `Monarch.mat` shows the most degradation, followed by `cameraman.mat`, and `Parrot.mat` shows the least degradation.

Further, it seems that the image of personal choice, `pisa.jpg`, shows even less degradation than `Parrot.mat`.

Among the first three images, `Monarch.mat` has the most activity, i.e., high frequency content, compared to the other two. Further, `cameraman.mat` has some sharp edges whereas `Parrot.mat` mostly has smoother edges.

This explains the disparity among their compressed versions. The effect of the DCT is to suppress high frequency content from an image, so images with a lot of such detail will undergo heavier degradation for the same quality level.

As sharp edges also contribute high frequency content to an image, images with such edges will also tend to degrade more than images with softer edges. We expect to see some ringing around such edges in the reconstruction due to the suppressed high frequencies.

We note that even though an image might contain quickly changing features/textures, if they are concentrated to small regions of the image, or if the features are small compared to the main subject of the image, the loss in such detail appears unnoticeable.

For example, the detail of the parrot's feathers, while constituted of high frequencies, remains mostly unnoticeable even in the original image, unless one inspects very closely. However, high frequency content is spread all throughout the image of the butterfly, due to the rapidly varying patterns on its wings.

Hence, the loss of detail in the butterfly's wings is more apparent in the compressed image than the loss of detail in the parrot's feathers.

Another observation made is the appearance of bands in the compressed image when the original contains smooth gradients. This is a consequence of the choice of independently quantizing 8×8 blocks of the image.

The 8×8 blocks restrict the frequencies that can be used to represent the variation within a block to those of cosines taking the form

$$\cos\left(\frac{(2n+1)k\pi}{16}\right).$$

However, when the image size is much larger than 8×8, it could contain gradients with far lower frequencies than could be represented by such cosines.

As a result, it is possible that such blocks be approximated solely by the average pixel value within them. This could lead to the formation of bands.

These effects are visible in the background of `cameraman.mat` and `pisa.jpg`, and more significantly so in `pisa.jpg` due to its greater height.

A Code Snippets

```
clc;
clear;
close all;

% We will use DCT to compress 8 x 8 blocks of the images
block_size = 8;

% Prepare image of choice
img_of_choice = rgb2gray(imread('pisa.jpg'));
```

```
% Array to store names of images being used
img_names = ["Monarch.mat", "Parrots.mat", "cameraman.mat"];
num_imgs = size(img_names, 2);

% Array with quality levels to be used for each of the images
quality_levels = [80, 30, 15];
num_quality_levels = size(quality_levels, 2);

% Load each image and perform the relevant tasks
for i = 0:num_imgs
        if i == 0
                img = img_of_choice;
                disp("Image of Choice");
        else
                img = load(img_names(i));
                fn = fieldnames(img);
                img = img.(fn{1});

                disp("Image: " + img_names(i));
        end

        figure;
        t = tiledlayout(1, num_quality_levels+1,'TileSpacing', 'compact', 'Padding', '
            compact');

        nexttile;
        imshow(img, [], 'Border', 'tight');
        title("Original Image");

        for quality_level = quality_levels
                disp("At " + quality_level + "% Compression");

                % Compress and decompress
                [compressed, num_zeros, num_zeros_percent] = compress_img(img, block_size
                    , quality_level);
                % Typically there would be a coding and decoding step here, but we
                % skip this step
                decompressed = decompress_img(compressed, block_size, quality_level);

                % Display the reconstructed image
                nexttile;
                imshow(decompressed, [], 'Border', 'tight');
                title(quality_level + "% Compression");

                % Compute the PSNR
                psnr_ = psnr(img, decompressed);
                disp("PSNR = " + psnr_);

                % Display the number of zeros
                disp("Number of zeros = " + num_zeros + " (" + num_zeros_percent + "%)");
        end
end
```

Listing 1: Main Code

```
function [compressed_img, num_zeros, num_zeros_percent] = compress_img(img, block_size,
    quality_level)

% Start by converting the image to type double
img = double(img);

% Extract dimensions of the image
[img_h, img_w] = size(img);

% Construct an empty matrix of the same size as the original image to store
% the compressed image
compressed_img = zeros(size(img));

% Variable to store the number of zeros in the compressed image
num_zeros = 0;

% Obtain the quantization matrix to be used in the compression
Q = get_quantization_matrix(block_size, quality_level);

% Loop over block_size x block_size blocks of the image
for row = 1:block_size:img_h
       for col = 1:block_size:img_w
              % Extract the block_size x block_size block from the image
              B = img(row:row+(block_size - 1), col:col+(block_size - 1));

              % Level-off by subtracting 128
              B = B - 128.0;

              % Apply the 2D DCT
              C = dct2(B);

              % Quantize the DCT coefficients
              [S, block_num_zeros] = quantize(C, Q);

              % Replace the relevant block in the target image with the quantized
              % DCT coefficients
              compressed_img(row:row+(block_size - 1), col:col+(block_size - 1)) = S;

              % Keep a running count of the number of zeros
              num_zeros = num_zeros + block_num_zeros;
       end
end

% Find the number of zeros as a percentage
num_zeros_percent = (num_zeros / (img_h * img_w)) * 100;

end
```

Listing 2: Image Compression

```
function [decompressed_img] = decompress_img(img, block_size, quality_level)

% Start by converting the image type to double
```

```
img = double(img);

% Obtain the image dimensions
[img_h, img_w] = size(img);

% Construct an empty matrix to store the decompressed image
decompressed_img = zeros(size(img));

% Obtain the quantization matrix used corresponding to the specified
% quality level
Q = get_quantization_matrix(block_size, quality_level);

% Loop over block_size x block_size blocks of the image
for row = 1:block_size:img_h
        for col = 1:block_size:img_w
        % Extract the relevant block from the image
        S = img(row:row+(block_size - 1), col:col+(block_size - 1));

        % Estimate the matrix that was quantized to obtain the block above
        R = Q .* S;

        % Obtain the inverse DCT
        E = idct2(R);

        % Replace the corresponding block in the target image with the IDCT
        % and add 128 to undo the leveling off
        decompressed_img(row:row+(block_size - 1), col:col+(block_size - 1)) = E + 128.0;
        end
end

end
```

Listing 3: Image Decompression

```
function [Q] = get_quantization_matrix(block_size, quality_level)
% Note that the size of Q depends on the block size being used. Here, we
% hardcode for a block of size 8.

% Provided quantization matrix for quality level 50 and block size 8
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
];

% Scaling factor for a specified quality level
if quality_level < 50
        tau = 50 / quality_level;
else
```

```
        tau = (100 - quality_level) / 50;
end

% Round the entries of the scaled Q and clip off values as described
Q = round(tau * Q_50);
Q = min(max(Q, 0), 255);

end
```

Listing 4: Generation of Quantization Matrix

```
function [S, num_zeros] = quantize(C, Q)

S = round(C ./ Q);
num_zeros = sum(sum(S == 0));

end
```

Listing 5: Quantization

```
function [result] = psnr(original_img, reconstructed_img)
% Convert both images to type double
original_img = double(original_img);
reconstructed_img = double(reconstructed_img);

% Maximum intensity level in the original image
psi_max = max(original_img(:));

% Obtain the error matrix
error = original_img - reconstructed_img;

% Obtain the root mean squared error
mean_squared_error = mean(mean(error .^ 2));
root_mean_squared_error = sqrt(mean_squared_error);

result = 20 * log10(psi_max / root_mean_squared_error);
end
```

Listing 6: PSNR Computation