**08.06.2023**

# CmpE362 Project: Digital Signal Processing Application Image Compression


**Volkan Öztürk & Arda Arslan**
**2019400033 2020400078**

**Computer Engineering Department**
**Engineering Faculty**
**Bogazici University**

**This documentation includes the following sections:**

## 1) Introduction

In this project we examine different methods for image compression. We choose two methods from the literature: Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT). We analyze the trade-off between image quality and size. Finally compare efficiency of two methods.

## 2) Image Compression Methods

### 2.1) Discrete Cosine Transform (DCT):

The formal explanation of DCT can be found here.

#### 2.1.1) Explanation of our code:

```
% Reading the image
originalImage = imread('IMG_1jpeg');

% Converting the image to double precision image
originalImage = im2double(originalImage);

% Spliting the image into RGB channels
redChannel = originalImage(:, :, 1);
greenChannel = originalImage(:, :, 2);
blueChannel = originalImage(:, :, 3);
```

We first read the image and convert to double. Then by indexing the third dimension with values 1, 2, and 3, we extract the red, green, and blue channels, respectively.

```
% Performing DCT on each color channel separately
dctRedChannel = dct2(redChannel);
dctGreenChannel = dct2(greenChannel);
dctBlueChannel = dct2(blueChannel);
```

This code segment applies the Discrete Cosine Transform (DCT) to each color channel separately. The DCT is a mathematical transformation that converts the image from the spatial domain to the frequency domain. The dct2 function (available in Image Processing Toolbox of Matlab) is used to perform the DCT on each color channel.

```
% Setting a threshold to remove high-frequency coefficients
% Higher the threshold lower the image size with lower quality
% Threshold coefficient values: 0.01, 0.0005, 0.000001
t_coeff1 = 0.001;
t_coeff2 = 0.0005;
t_coeff3 = 0.000001;
threshold = t_coeff3 * max([abs(dctRedChannel(:)); abs(dctGreenChannel(:)); abs(dctBlueChannel(:))]);

% Appling compression by zeroing out coefficients below the threshold
dctRedChannel(abs(dctRedChannel) < threshold) = 0;
dctGreenChannel(abs(dctGreenChannel) < threshold) = 0;
dctBlueChannel(abs(dctBlueChannel) < threshold) = 0;
```

This code calculates the threshold value based on the maximum absolute value among the DCT coefficients of all color channels. It multiplies the maximum value by desired coefficient to obtain the threshold. The threshold is used to determine which coefficients to keep or discard in the compression process. Changing the threshold has a trade-off between quality over size. Then we apply the compression by setting the DCT coefficients below the threshold to zero. By doing this, the high-frequency components (which generally represent fine details) are eliminated or reduced, resulting in compression.

```
% Performing inverse DCT to retrieve the compressed image
compressedRedChannel = idct2(dctRedChannel);
compressedGreenChannel = idct2(dctGreenChannel);
compressedBlueChannel = idct2(dctBlueChannel);
```

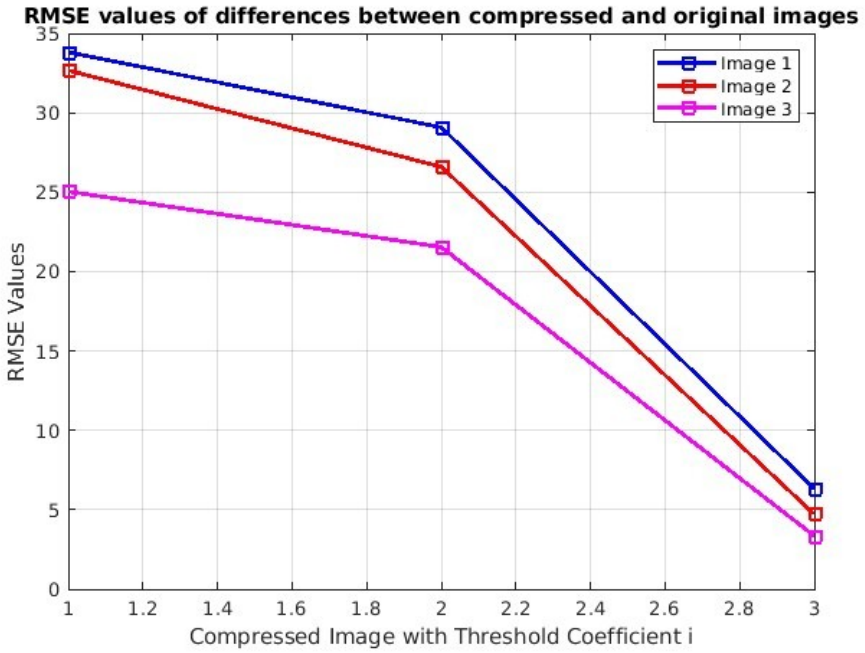
We then change the domain from frequency domain to spatial domain by using inverse DCT transform.

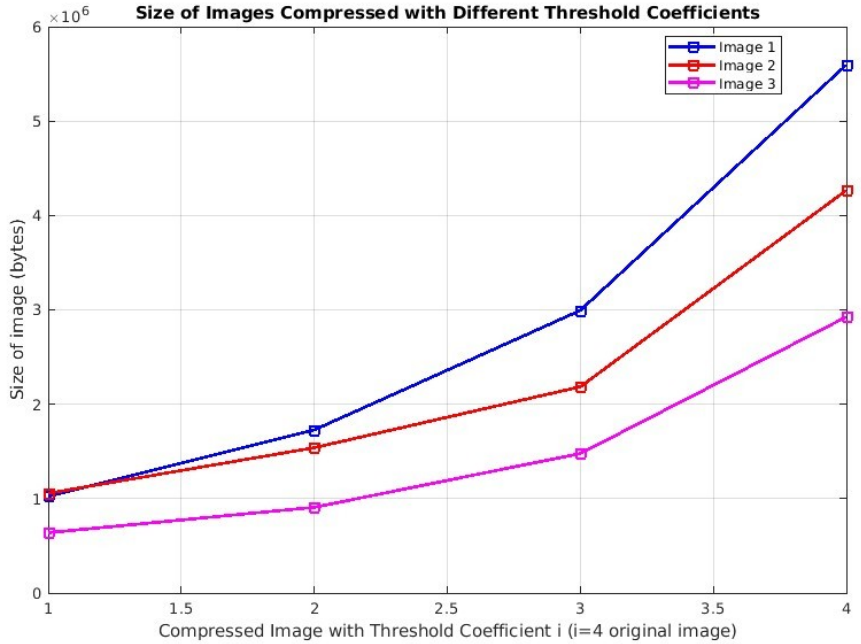
```
% Combining the color channels into the compressed image
compressedImage = cat(3, compressedRedChannel, compressedGreenChannel, compressedBlueChannel);

% Save the compressed image
imwrite(compressedImage, 'IMG_dct_compressed_1.jpg');
```

Finally we combine RGB channels and write the image.

## 2.1.2) Results



Example Image 1

Original image

Image after compression with t_coeff = 0.01

Image after compression with t_coeff = 0.0005

Image after compression with t_coeff = 0.000001

Example Image 2

Original image

Image after compression with t_coeff = 0.01

Image after compression with t_coeff = 0.0005

Image after compression with t_coeff = 0.000001

Example Image 3

Original image

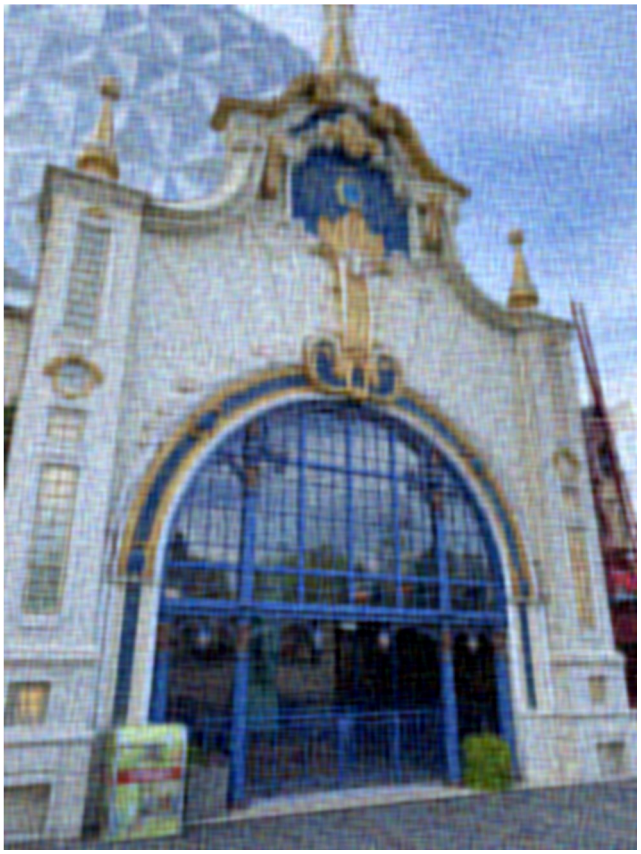


Image after compression with t_coeff = 0.01

Image after compression with t_coeff = 0.0005

Image after compression with t_coeff = 0.000001

## 2.2) Discrete Wavelet Transform (DWT):

The formal explanation of DWT can be found here.

### 2.2.1) Explanation of our code:

```
% Reading the image
originalImage = imread('IMG_3.jpeg');

% Converting image to double precision
originalImage = im2double(originalImage);
```

The image is read and converted to double

```
% Defining the wavelet and level of decomposition
wavelet = 'sym8';
level = 3; % Number of decomposition levels
```

First we define wavelet and its number of decomposition levels. Symlets are a variant of Daubechies wavelets. They are similar in shape to Daubechies wavelets but have slightly better frequency characteristics. Symlets are denoted as symN, where N represents the number of vanishing moments.

```
% Performing DWT on each color channel
compressedImage = zeros(size(originalImage));
for i = 1:3
    channel = originalImage(:,:,i);
    [cA, cH, cV, cD] = dwt2(channel, wavelet);

    % Defining the compression ratio
    compressionRatio1 = 0.1;
    compressionRatio2 = 0.5;
    compressionRatio3 = 0.9;

    % Determining the number of coefficients to keep
    coefficientsToKeep = round(compressionRatio3 * numel(cA));

    % Converting the wavelet coefficients matrices to vectors
    cA_vector = cA(:);
    cH_vector = cH(:);
    cV_vector = cV(:);
    cD_vector = cD(:);

    % Sorting the coefficients in descending order of magnitude
    [~, indices] = sort(abs(cA_vector), 'descend');

    % Setting the least significant coefficients to zero
    cA_vector(indices(coefficientsToKeep+1:end)) = 0;

    % Reshaping the vector to the original wavelet coefficients matrices
    compressed_cA = reshape(cA_vector, size(cA));
    compressed_cH = reshape(cH_vector, size(cH));
    compressed_cV = reshape(cV_vector, size(cV));
    compressed_cD = reshape(cD_vector, size(cD));

    % Reconstructing the compressed channel using inverse DWT
    compressedChannel = idwt2(compressed_cA, compressed_cH, compressed_cV, compressed_cD, wavelet);

    % Updating the compressed image with the compressed channel
    compressedImage(:,:,i) = compressedChannel;
end
```

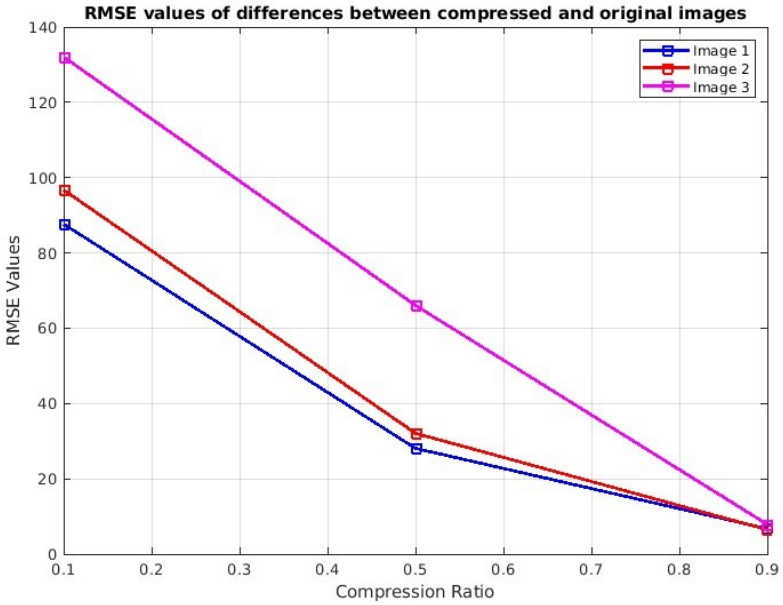
We construct a for loop to make same operations (body) for each channel of the RGB image. Explanation of loop body:

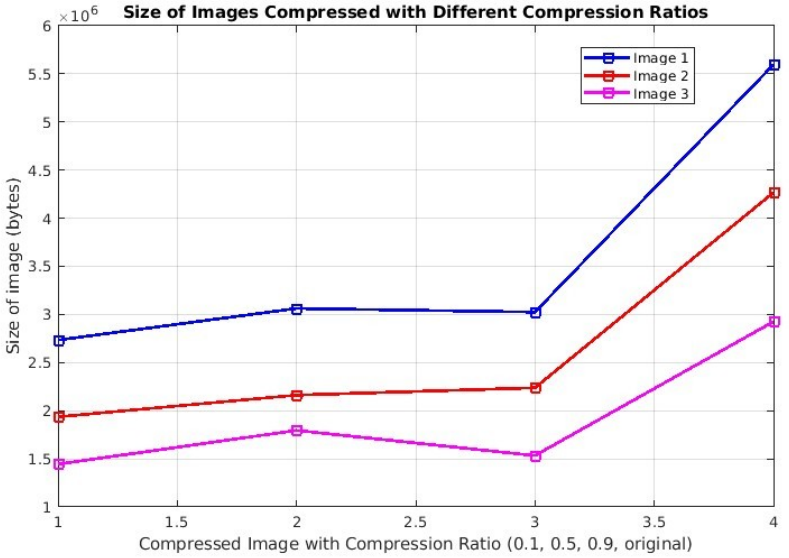
We first retrieve the channel. Then used dwt2 function of MATLAB available in Image Processing Toolbox to make discrete wavelet transform. The image is decomposed into four sets of wavelet coefficients: approximation coefficients (cA), horizontal detail coefficients (cH), vertical detail coefficients (cV), and diagonal detail coefficients (cD). The dwt2 function performs the 2D DWT using the specified wavelet. Then we determine the coefficient ratio according to our needs (trade-off between image size & quality). Then we determine the number of coefficients to keep using coefficient ratio that we set (This is about precision). Afterwards we effectively delete (set zero to least significant ones) that number of coefficients of the sorted vectors of wavelet coefficients. Finally we reshape the wavelet coefficient vectors to matrices and used inverse discrete wavelet transform by idwt2 that is available within Image Processing Toolbox and we construct the resulting image by placing the channel accordingly.

```
imwrite(compressedImage, "IMG_3_dwt_compressed_ratio3.jpg")
```

Finally we write the image.

2.2.2) Results


RMSE values of differences between compressed and original images
Image 1
Image 2
Image 3
RMSE Values
Compression Ratio
0
20
40
60
80
100
120
140
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9


Size of Images Compressed with Different Compression Ratios
×10^6
Image 1
Image 2
Image 3
Size of image (bytes)
Compressed Image with Compression Ratio (0.1, 0.5, 0.9, original)
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
6

Example Image 1

Original image

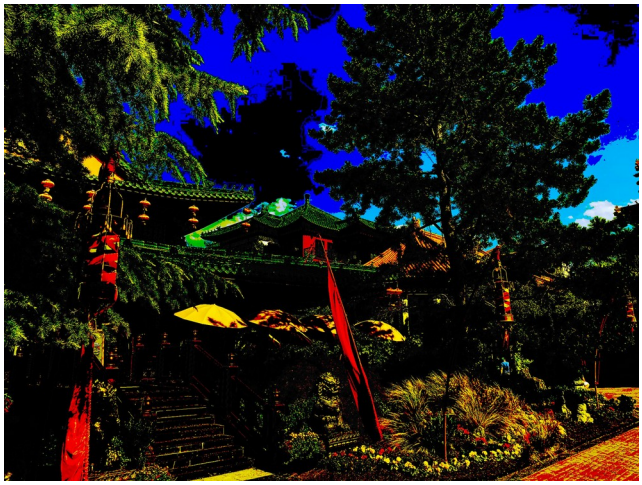


Image after compression with ratio= 0.1

Image after compression with ratio= 0.5

Image after compression with ratio= 0.9

Example Image 2

Original image

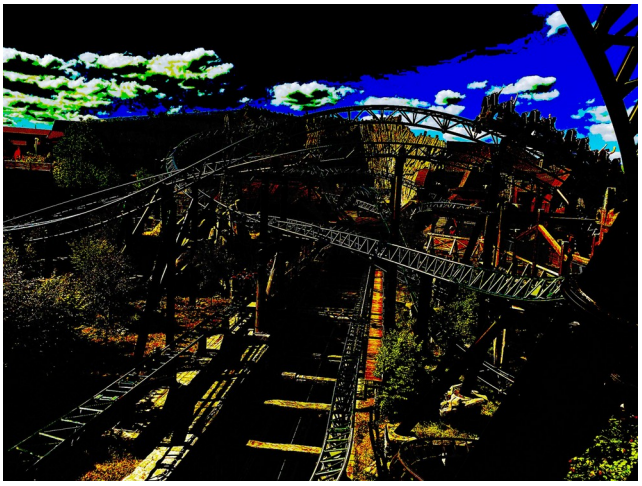


Image after compression with ratio= 0.1

Image after compression with ratio= 0.5

Image after compression with ratio= 0.9

Example Image 3

Original image

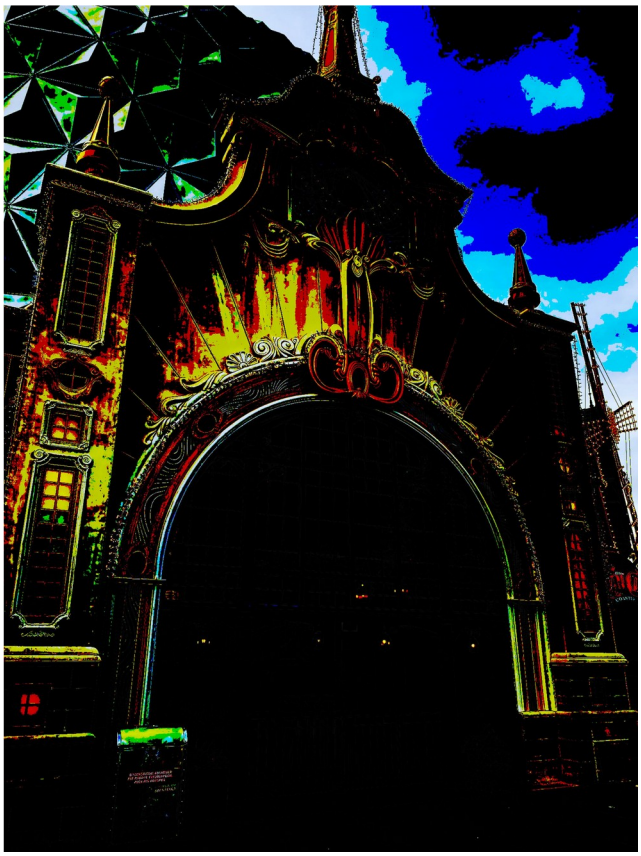


Image after compression with ratio= 0.1

Image after compression with ratio= 0.5

Image after compression with ratio= 0.9

3) Conclusion

We successfully implemented two image processing methods from literature. For both of the methods (Discrete Cosine Transform & Discrete Wavelet Transform ) as well as other methods from literature; it is evident that there is a trade-off between image size and its quality. From the figures that we have plotted in the result sections, we observed this trade-off clearly. Higher the image size, higher the quality (We compared quality with using RMSE values.).

To compare the two different methods we implemented (Discrete Cosine Transform & Discrete Wavelet Transform ), DCT performs better over DWT. Because it is evident that DWT causes bigger RMSE values with respect to size ratio (compressed / original) compared to DCT.