EQ2330 Image and Video Processing

EQ2330 Image and Video Processing, Project 2

LO, King Lam
kllo@kth.se

TSANG, Chin Yung Virginia
cyvtsang@kth.se

CHIU, Lok Ying
lychiu@kth.se

December 7, 2022

Summary

In this project, we evaluate the performances of two transform-based image compression algorithms- the discrete cosine transform (DCT) and the fast wavelet transform (FWT). Evaluation is done by measuring the degradation due to the quantization of transform coefficients. Here we implement the two transforms, quantize the coefficients and then measure the performances of using different bit-rates in encoding quantized transform coefficients.

1 Introduction

Today's image compression methods fall into two main categories: lossless and lossy image compression. Both the Discrete cosine transform (DCT) and the Discrete wavelet transform (DWT) belong to lossy compression, in which they undergo a step called quantization to discard less important information.

DCT works similarly to discrete Fourier transform where it transforms the image into the frequency domain. The transform separates the image into parts of different frequencies and expresses them in sums of varying frequency cosine functions. DWT is based on small wave functions called wavelets with varying frequency and limited duration. These wavelets offer the advantage of simultaneous localization in the time and frequency domain.

The reconstructed image with these two image coders contains some distortion due to information loss during the process. However, as we will soon look into, the distortion level can be adjusted in the compression stage and is measured by the PSNR curve.

2 System Description

2.1 DCT-based Image Compression

The DCT represents an image as a sum of sinusoids of varying magnitudes and frequencies. The transformation is separable and orthonormal, therefore it can be written as $y = AxA^T$. The inverse of the DCT transform is $x = A^T yA$. x is a block with $M \times M$ size from an image, and A is an $M \times M$ transform matrix with elements

$$\alpha_{ik} = \alpha_i \cos\left(\frac{(2k+1)i\pi}{2M}\right) \text{ for } i,k = 0,1,\ldots,M-1, \text{with } \alpha_0 = \sqrt{\frac{1}{M}}\ \alpha_i = \sqrt{\frac{2}{M}}\ \forall i > 0 \quad (1)$$

We use $M = 8$ as the size of the image block.

2.1.1 Uniform Quantizer

A uniform mid-thread quantizer without threshold characteristics can be expressed as

$$y = q\lfloor\frac{x}{q}\rceil \quad (2)$$

where q represents the step size, x the input and y the quantized input

2.1.2 Distortion and Bit-rate Estimation

We use the Peak Signal to Noise Ratio (PSNR) to evaluate the quality of the reconstructed image. A high PSNR implies a good quality of the reconstructed image. To compute the PSNR, we first need to calculate the mean squared error (MSE) between the original image and the reconstructed image, which is defined as follows:

$$MSE = \frac{\sum_{M,N}[I_1(m,n) - I_2(m,n)]^2}{M * N} \tag{3}$$

where M and N are the numbers of rows and columns of the images, and I(m, n) represents the intensity value of the (m, n)th pixel. Then we use it to compute the PSNR. The PSNR defined for 8-bit images is as follows:

$$PSNR = 10 \log_{10} \frac{255^2}{MSE} \tag{4}$$

We encode each of the 64 coefficients in a block by using variable length code (VLC). To calculate the bit-rate required to encode them, we use the following formula to compute the entropy of each block:

$$H(X) = -\sum_{x \epsilon X} p(x) \log_2 p(x) \tag{5}$$

where p(x) is the probability of occurrence of x. Then we take the mean of all the blocks' entropies as the bit-rate.

2.2 FWT-based Image Compression

2.2.1 The Two-Band Filter Bank

We used the direct implementation method to implement the Two-Band Filter Bank. The flow chart of the implementation is shown in Figure 1. The left-hand side of Figure 1 is the flow of the 1D two-band analysis filter bank function, and the right-hand side is the flow of the 1D two-band synthesis filter bank function. In both of the functions, padding will be added before the convolution and it will be removed after the convolution as well. It is to avoid the border effects of the convolution operation.

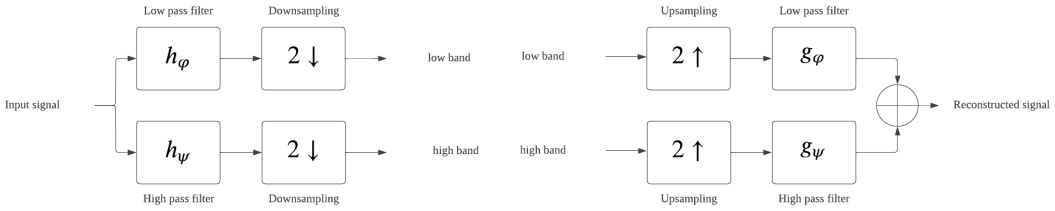

Figure 1: Flow chart of 1D Two-Band Filter Bank

2.2.2 The FWT

From Section 7.5 of the textbook[1], to implement a 2D FWT function, we first apply the analysis filter along the horizontal direction. Then, for each of the coefficients we get from the analysis filter, we apply the analysis filter again along the vertical direction. As a result, we will have the approximation, horizontal, vertical, and diagonal coefficients matrices. Daubechies 8-tap filter is used in our implementation.

To achieve scaling, we apply the 2D FWT function to the approximation coefficients matrix recursively. The approximation coefficient matrix would become the new input image in the next iteration, as shown in Figure 2. Figure 3 depicted the reconstructed image as the approximation coefficients matrix in the next iteration of inverse 2D FWT.

2.2.3 Uniform Quantizer

The uniform quantizer for FWT is the same as the one for DCT. Please refer to Section 2.1.1

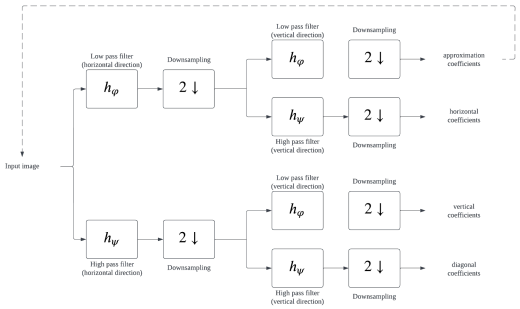

Figure 2: Flow chart of 2D FWT

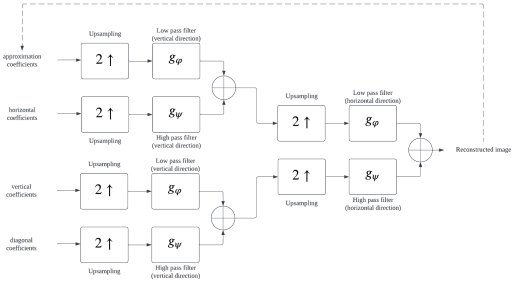

Figure 3: Flow chart of 2D iFWT

2.2.4 Distortion and Bit-Rate Estimation

To calculate the PSNR for the FWT, equation 4 is used. And to calculate the bit-rate for VLC, equation 5 is used.

3 Results

3.1 DCT-based Image Compression

To implement an 8×8 DCT transform following equation 1, we use matrix A as shown below:

$$A = \begin{pmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & 0.1913 & 0.4619 & 0.4619 & 0.1913 & 0.1913 & 0.4619 \\ 0.4157 & 0.0975 & 0.4904 & 0.2778 & 0.2778 & 0.4904 & 0.0975 & 0.4157 \\ 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.2778 & 0.4904 & 0.0975 & 0.4157 & 0.4157 & 0.0975 & 0.4904 & 0.2778 \\ 0.1913 & 0.4619 & 0.4619 & 0.1913 & 0.1913 & 0.4619 & 0.4619 & 0.1913 \\ 0.0975 & 0.2778 & 0.4157 & 0.4904 & 0.4904 & 0.4157 & 0.2778 & 0.0975 \end{pmatrix} \tag{6}$$

Using the DCT algorithm on an 8×8 image block from $boats512x512.tif$ we find that applying DCT followed by inverse DCT results in a perfectly reconstructed image as the original. This is illustrated from figure 4 to 6.

Figure 4: A 8×8 block from the boat image

Figure 5: The DCT transform of figure 4

Figure 6: The inverse DCT transform of figure 5

We quantized the coefficients with step-size = 4 by a uniform mid-tread quantizer without threshold characteristic. Figure 7 shows the quantizer function.

The distortion of the reconstructed image from the original one after quantizing DCT coefficients is 0.0833. We take quantization step size =1. Using the same step size we have distortion between the original and the quantized DCT coefficients equals to 0.0833. With the perfect reconstruction of the DCT algorithm, the same extent of distortion of the DCT coefficients passes on to distortion between images. Refer to figure 8. The blue curve in the plot shows how bit-rate relates to PSNR when using the DCT algorithm. To reduce the bit rate we use a larger quantization step.

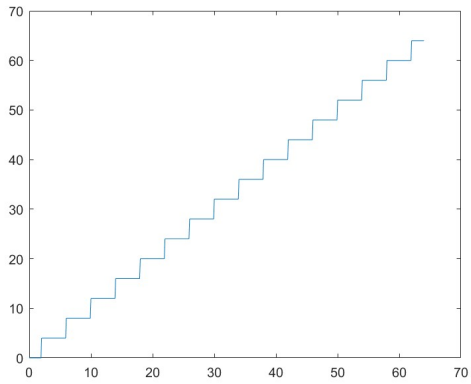

Figure 7: Quantizer function with step-size = 4

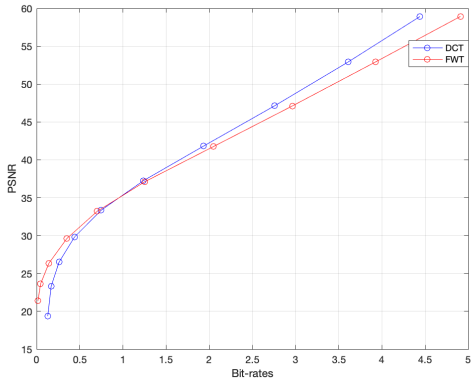

Figure 8: Relationship between Bit-rate and PSNR in DCT and FWT

For each decreased bit rate the PSNR decreases by 6 dB. This means a reduction of the bit-rate will decrease the image quality.

3.2 FWT-based Image Compression

3.2.1 The Two-Band Filter Bank and The FWT

We passed ***harbour512x512.tif*** to the 2D FWT function implemented by following the flow in Figure ... Figure 9 shows the wavelet coefficients for scale 4.

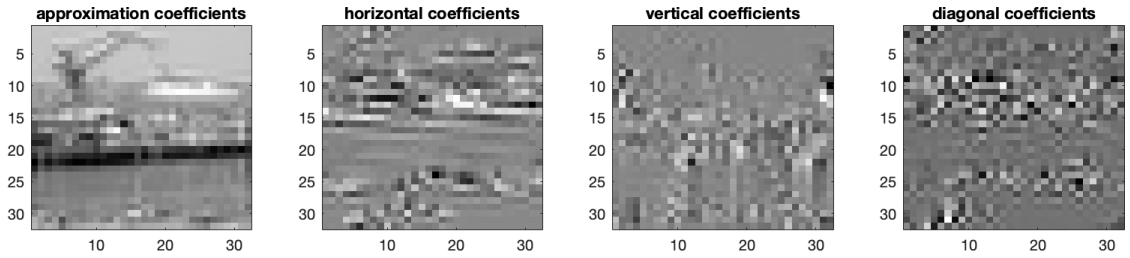

Figure 9: wavelet coefficients for scale 4 of the image ***harbour512x512.tif***

As 2D FWT needed to achieve perfect reconstruction, we tried to reconstruct the image. In Figure 10, the left-hand side is the original image and the right-hand side is the reconstructed image. There is no difference between them.

3.2.2 Uniform Quantizer and Distortion and Bit-Rate Estimation

We quantize all of the wavelet coefficients we get from the 2D FWT function. The step size is set to 1. Then we reconstruct the image with the quantized coefficients. The distortion between the original image and the reconstructed image is 0.0832. And the distortion between every original wavelet coefficients and its quantized coefficients is roughly around 0.083. The weighted average of the distortion between every original wavelet coefficients and its quantized coefficients is 0.0832, which is the same as the distortion between the original image and the reconstructed image. This is because the FWT can reconstruct the image perfectly, so the distortion in the quantized coefficients will be preserved in the reconstructed image.

In figure 8, the red curve shows the relationship between bit-rate and PSNR when FWT is used. To lower the bit-rate, quantization with a larger step size is used. We can see that when the bit-rate is decreased by 1 bit, the PSNR is decreased by 6 dB. It showed the reduction of the bit-rate will reduce the quality of the image.

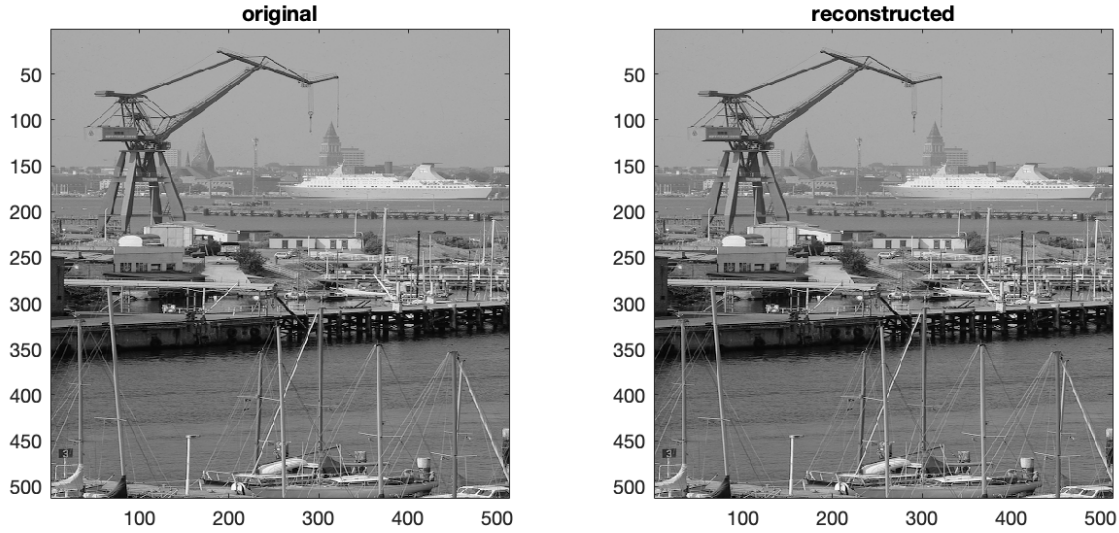

Figure 10: original and reconstructed image of ***harbour512x512.tif***

4 Conclusions

In this project, we implemented two image compression algorithms which are the discrete cosine transform and the discrete wavelet transform. By applying the transforms then their respective inverses, we showed that both algorithms perfectly reconstruct the original input image. If we quantize the transform coefficients in the algorithms, we can reduce the bit-rate required to encode the image, hence the effect of compressing an image. However, it is worth mentioning that quantization removes details of the image and therefore makes the compression lossy. The DCT and the FWT perform differently under different bit-rates in image compression. Using a low bit-rate between 0 to 1 we have the FWT compressing the image better. With a high bit-rate greater than 1 the DCT performs better than the FWT.

Appendix

Who Did What

We equally spilt the work.

MatLab code

4.0.1 bitrate.m

```
function entropy = bitrate(img) % calculate the bit-rate of a given matrix
    [M, N] = size(img);
    [pixel_counts, ~] = groupcounts(img(:)); % find the counting of all levels
    pixel_probas = nonzeros(pixel_counts(:)./(M*N));
    log_pixel_probas = log2(pixel_probas);
    entropy = -sum(pixel_probas.*log_pixel_probas);
end
```

4.0.2 dct2matrixA.m

```
function [A] = dct2matrixA(M)
A = zeros(M);
for i=0:M-1
    for j=0:M-1
        if i==0
            alpha = sqrt(1/M);
```

```
        else
            alpha = sqrt(2/M);
        end
        A(i+1,j+1) = alpha*cos((2*j+1)*i*pi/(2*M));
    end
end
```

4.0.3 fwt2d.m

```
function [LL, LH, HL, HH] = fwt2d(img, LPF)
    img_length = length(img);
    L = zeros([img_length,img_length/2]);
    H = zeros([img_length,img_length/2]);
    % first work on the horizontal direction
    for x = 1:img_length
        [L(x,:),H(x,:)] = fwt_analysis(img(x,:),LPF);
    end
    LL = zeros([img_length/2,img_length/2]);
    LH = zeros([img_length/2,img_length/2]);
    HL = zeros([img_length/2,img_length/2]);
    HH = zeros([img_length/2,img_length/2]);
    % then work on the vertical direction
    for y = 1:img_length/2
        [LL(:,y),LH(:,y)] = fwt_analysis(L(:,y).',LPF);
        [HL(:,y),HH(:,y)] = fwt_analysis(H(:,y).',LPF);
    end
end
```

4.0.4 fwt2d_scale.m

```
function [LLs, LHs, HLs, HHs] = fwt2d_scale(img, scale)
    % Daubechies 8-tap filter is used
    [ LPF, ~, ~, ~] = wfilters("db8");
    curr_LL = img;
    LLs = cell(1,scale);
    LHs = cell(1,scale);
    HLs = cell(1,scale);
    HHs = cell(1,scale);

    % achieve scaling by recusively apply the filter
    for i = 1:scale
        [LLs{i}, LHs{i}, HLs{i}, HHs{i}] = fwt2d(curr_LL, LPF);
        curr_LL = LLs{i};
    end
end
```

4.0.5 fwt_analysis.m

```
function [cA,cD] = fwt_analysis(signal, LPF)
    % find the coresponing high pass filter
    HPF = wrev(qmf(wrev(LPF)));

    % add the periodic extension to the input signal to prevent border
    % effect
    padded = padarray(signal, [0,length(LPF)-1], "circular");

    output_lp = conv(padded, LPF, "full");
    output_hp = conv(padded, HPF, "full");
```

```
    %remove padding
    output_lp = output_lp((length(LPF)-1)*2:(length(LPF)-1)*2+length(signal)-1);
    output_hp = output_hp((length(HPF)-1)*2:(length(HPF)-1)*2+length(signal)-1);

    cA = downsample(output_lp, 2);
    cD = downsample(output_hp, 2);

end
```

4.0.6 fwt_psnr_bitrate.m

```
function [weighted_avg_bitrates, avg_PSNRs] = fwt_psnr_bitrate()

imgs{1} = 255*im2double(imread("../images/boats512x512.tif"));
imgs{2} = 255*im2double(imread("../images/harbour512x512.tif"));
imgs{3} = 255*im2double(imread("../images/peppers512x512.tif"));

[~, total_imgs] = size(imgs);

scale = 4;

step_range = [2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9];

[~, total_step_range] = size(step_range);

bitrates = zeros(10,4,scale);
weighted_avg_bitrates = zeros(10,1);

PSNRs = zeros(10, 3);

for i = 1:total_step_range
    concated_coffs = cell(4,scale);
    for j = 1:total_imgs
        % apply fwt to each image
        [LLs, LHs, HLs, HHs] = fwt2d_scale(imgs{j}, scale);
        % quantize all of the coefficients
        [LLs, LHs, HLs, HHs] = quantize_all(LLs, LHs, HLs, HHs, scale, step_range(i));
        % reconstruct the image
        img_re = ifwt2d_scale(LLs, LHs, HLs, HHs, scale);
        % calculate the PSNR
        PSNRs(i,j) = psnr_8bits(imgs{j}, img_re);

        % merge all coefficients in each of the subband
        if j == 1
            for k = 1:scale
                concated_coffs{1,k} = LLs{k}(:);
                concated_coffs{2,k} = LHs{k}(:);
                concated_coffs{3,k} = HLs{k}(:);
                concated_coffs{4,k} = HHs{k}(:);
            end

        else
            for k = 1:scale
                concated_coffs{1,k} = [concated_coffs{1,k};LLs{k}(:)];
                concated_coffs{2,k} = [concated_coffs{2,k};LHs{k}(:)];
                concated_coffs{3,k} = [concated_coffs{3,k};HLs{k}(:)];
                concated_coffs{4,k} = [concated_coffs{4,k};HHs{k}(:)];
            end
```

```
        end

    end
    % calculate the bit rate of each subband
    for j = 1:4
        for k = 1: scale
            bitrates(i,j,k)= bitrate(concated_coffs{j,k});
        end
    end
    % calculate the weighted average of the bit-rate
    for j = 2:4
        for k = 1:scale
            weighted_avg_bitrates(i) = weighted_avg_bitrates(i) + bitrates(i,j,k)*1/(4^k);
        end
    end
    weighted_avg_bitrates(i) = weighted_avg_bitrates(i) + bitrates(i,1,scale) * 1/(4^k);

end

avg_PSNRs = mean(PSNRs, 2).';

end
```

4.0.7 fwt_synthesis.m

```
function output = fwt_synthesis(cA, cD, LPF)
    % find the coresponing high pass filter
    HPF = qmf(LPF);

    upsampled_cA = upsample(cA ,2);
    upsampled_cD = upsample(cD ,2);

    % add the periodic extension to the input signal to prevent border
    % effect
    padded_cA = padarray(upsampled_cA, [0,length(LPF)-1], "circular");
    padded_cD = padarray(upsampled_cD, [0,length(HPF)-1], "circular");

    output1 = conv(padded_cA, LPF, "full");
    output2 = conv(padded_cD, HPF, "full");

    %remove padding
    output1 = output1(length(LPF)+1:end - (length(LPF)-1)*2+1);
    output2 = output2(length(HPF)+1:end - (length(HPF)-1)*2+1);

    output = output1 + output2;
end
```

4.0.8 get_dct2.m

```
%dct2 function
function [y] = get_dct2(x,M)
A=dct2matrixA(M);
y = A*x*A'; %x is the signal block
plot_matrix(y);
end
```

4.0.9 get_indct2.m

```
%inverse dct2 function
function [x] = get_indct2(y,M)
A = dct2matrixA(M);
x = A'*y*A;
plot_matrix(x);
end
```

4.0.10 ifwt2d.m

```
function output = ifwt2d(LL, LH, HL, HH, LPF)
    img_length = length(LL);
    L = zeros([img_length*2,img_length]);
    H = zeros([img_length*2,img_length]);
    % first work on the vertical direction
    for y = 1:img_length
        L(:,y) = fwt_synthesis(LL(:,y).', LH(:,y).', LPF);
        H(:,y) = fwt_synthesis(HL(:,y).', HH(:,y).', LPF);
    end
    output = zeros([img_length*2,img_length*2]);
    % then work on the horizontal direction
    for x = 1:img_length*2
        output(x,:) = fwt_synthesis(L(x,:), H(x,:), LPF);
    end

end
```

4.0.11 ifwt2d_scale.m

```
function output = ifwt2d_scale(LLs, LHs, HLs, HHs, scale)
    % Daubechies 8-tap filter is used
    [ ~, ~, LPF, ~] = wfilters("db8");
    output = LLs{scale};
    % achieve scaling by recusively apply the filter
    for i = scale:-1:1
        output = ifwt2d(output, LHs{i}, HLs{i}, HHs{i}, LPF);
    end

end
```

4.0.12 mse.m

```
function output = mse(img1,img2)
    [M,N] = size(img1);
    output = sum((img1(:)-img2(:)).^2)/(M*N);
end
```

4.0.13 plot_matrix.m

```
function plot_matrix(A)
imagesc(A);
axis square;
axis off;
colormap(gray);
end
```

4.0.14 psnr_8bits.m

```
function output = psnr_8bits(img1,img2)
```

```
    output = 10 * log10((255^2)/mse(img1,img2));
end
```

4.0.15 q1.m

```
clc;
clear;

%import photos
boat = double(imread("../images/boats512x512.tif"));
harbours = double(imread("../images/harbour512x512.tif"));
peppers = double(imread("../images/peppers512x512.tif"));


%DCT of block size 8x8
M = 8;
A = dct2matrixA(M); % matrix A in dct2
disp(A);
%use boat 8x8 to test the function
boat8x8 = imresize(boat,[8 8]);
boat8x8 = double(boat8x8);
figure(3)
subplot(1,3,1);
plot_matrix(boat8x8);

%dct2 transform
subplot(1,3,2);
y = get_dct2(boat8x8,M);

%inverse dct2 transfosrm
subplot(1,3,3);
get_indct2(y,M);

% quantizer
% x: 0-64
% y: function output
figure(1);
arr = 0:0.1:64;
output = quantizer(arr, 4);
subplot(1,1,1);
plot(arr, output);

[boat_M, boat_N] = size(boat);
img_rc = zeros(boat_M, boat_N);
mse_q = 0;
total_coff = 0;

for i = 1:boat_M/M
    for j = 1:boat_N/M
        img_block = boat((i-1)*M+1:M*i,(j-1)*M+1:M*j); % index of the row: (i-1)*M+1:M*i, index of
        img_dct = dct2(img_block); % DCT
        img_q = quantizer(img_dct, 1); % quantizer
        mse_q = mse_q + mse(img_dct, img_q);
        total_coff = total_coff+1;
        img_rc((i-1)*M+1:M*i,(j-1)*M+1:M*j) = A'*img_q*A; % reconstructed image
    end
end

mse_q = mse_q/total_coff; % the mse between original and the quantized dct coeffs
d = mse(boat, img_rc); % d = the mse between original and the reconstructed image
```

```
step_range = [2^0, 2^1, 2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9];
[~, total_step_range] = size(step_range);
bitrates = zeros(10,64,64);
PSNRs = zeros(10, 3);

total_coff = 64;

for i = 1:total_step_range
    b_rc = zeros(size(boat));
    h_rc = zeros(size(harbours));
    p_rc = zeros(size(peppers));
    for j = 1:total_coff
        for k = 1:total_coff
            b_block = boat((j-1)*M+1:M*j,(k-1)*M+1:M*k);
            h_block = harbours((j-1)*M+1:M*j,(k-1)*M+1:M*k);
            p_block = peppers((j-1)*M+1:M*j,(k-1)*M+1:M*k);

            b_dct = dct2(b_block); % DCT
            h_dct = dct2(h_block);
            p_dct = dct2(p_block);

            b_q = quantizer(b_dct, step_range(i)); % quantizer
            h_q = quantizer(h_dct, step_range(i));
            p_q = quantizer(p_dct, step_range(i));

            merged = [b_q(:), h_q(:), p_q(:)];
%             disp("i,j,k:"+i+ " "+ j+ " "+k);
%             test = bitrate(merged);
            bitrates(i,j,k) = bitrate(merged);

            b_rc((j-1)*M+1:M*j,(k-1)*M+1:M*k) = A'*b_q*A; % reconstructed image
            h_rc((j-1)*M+1:M*j,(k-1)*M+1:M*k) = A'*h_q*A;
            p_rc((j-1)*M+1:M*j,(k-1)*M+1:M*k) = A'*p_q*A;

        end
    end
    PSNRs(i,1) = psnr_8bits(boat, b_rc);
    PSNRs(i,2) = psnr_8bits(harbours, h_rc);
    PSNRs(i,3) = psnr_8bits(peppers, p_rc);

end

avg_PSNRs = mean(PSNRs,2);
avg_bitrates = mean(bitrates,2);
avg_bitrates = mean(avg_bitrates, 3);

[fwt_avg_bitrates, fwt_avg_PSNRs]=fwt_psnr_bitrate();

figure(2);

plot(avg_bitrates, avg_PSNRs, '-bo');
hold on;
plot(fwt_avg_bitrates, fwt_avg_PSNRs, '-ro');

xlabel("Bit-rates");
ylabel("PSNR");
```

```
legend("DCT", "FWT");
grid on;
```

4.0.16 q2.m

```
clc;
clear;

load coeffs.mat

img = imread("../images/harbour512x512.tif");

scale = 4;

% perform fwt in harbour
[LLs, LHs, HLs, HHs] = fwt2d_scale(img, scale);

%plot all of the coefficients in scale 4
figure(1);
subplot(1, 4, 1);
imagesc(LLs{scale});
title("approximation coefficients");
subplot(1, 4, 2);
imagesc(LHs{scale});
title("horizontal coefficients");
subplot(1, 4, 3);
imagesc(HLs{scale});
title("vertical coefficients");
subplot(1, 4, 4);
imagesc(HHs{scale});
title("diagonal coefficients");
colormap gray(256);

step = 1;

%quantize all coefficients
[qLLs, qLHs, qHLs, qHHs] = quantize_all(LLs, LHs, HLs, HHs, scale, step);
% recontruct the image
img_re = ifwt2d_scale(qLLs, qLHs, qHLs, qHHs, scale);

% plot the image
figure(2);
subplot(1, 2, 1);
imagesc(img);
title("original");
subplot(1, 2, 2);
imagesc(img_re);
title("reconstructed");
colormap gray(256);

img = 255*im2double(img);

% calculate the mse between original image and reconstructed image
ori_re_mse = mse(img, img_re);

disp("d between original and reconstructed: " + ori_re_mse);

% calculate the mse between each of the coefficients
LLs_mse = zeros(1,4);
LHs_mse = zeros(1,4);
```

```
HLs_mse = zeros(1,4);
HHs_mse = zeros(1,4);

for i = 1:scale
    LLs_mse(i) = mse(LLs{i},qLLs{i});
    LHs_mse(i) = mse(LHs{i},qLHs{i});
    HLs_mse(i) = mse(HLs{i},qHLs{i});
    HHs_mse(i) = mse(HHs{i},qHHs{i});
end

% take the weighted average
overall_cof_mean = 0;

for i = 1:scale
    overall_cof_mean = overall_cof_mean + LHs_mse(i)*(1/(4^i));
    overall_cof_mean = overall_cof_mean + HLs_mse(i)*(1/(4^i));
    overall_cof_mean = overall_cof_mean + HHs_mse(i)*(1/(4^i));
end

overall_cof_mean = overall_cof_mean + LLs_mse(scale)*(1/(4^scale));
```

4.0.17 quantize_all.m

```
function [LLs, LHs, HLs, HHs] = quantize_all(LLs, LHs, HLs, HHs, scale, step)
    % quantize all coefficients
    for i = 1:scale
        LLs{i} = quantizer(LLs{i}, step);
        LHs{i} = quantizer(LHs{i}, step);
        HLs{i} = quantizer(HLs{i}, step);
        HHs{i} = quantizer(HHs{i}, step);
    end
end
```

4.0.18 quantizer.m

```
function output = quantizer(input, step)
    output = step* round(input/step);
end
```

References

[1] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, 2nd ed., 2002