Report for EQ2330 Image and Video Processing

EQ2330 Image and Video Processing, Project 2

Zhenghao Li
zhenghl@kth.se

Letao Feng
letao@kth.se

Heng Zhao
hengzhao@kth.se

December 6, 2023

Summary

In this project, we assess the effectiveness of two image compression algorithms based on transforms: the discrete cosine transform (DCT) and the fast wavelet transform (FWT). The evaluation involves quantizing the coefficients obtained from these transforms and measuring the impact of quantization on image quality. Our implementation includes the transformation processes, coefficient quantization, and performance measurement across various bit-rates used in encoding the quantized transform coefficients

1 Introduction

Contemporary image compression techniques can be broadly categorized into two types: lossless and lossy compression. The Discrete Cosine Transform (DCT) and the Discrete Wavelet Transform (DWT) are examples of lossy compression methods, involving a quantization step to eliminate less critical information.

The DCT operates in a manner akin to the discrete Fourier transform, shifting the image into the frequency domain. This transformation dissects the image into segments of varying frequencies, representing them as sums of cosine functions with different frequencies. On the other hand, the DWT relies on small wave functions known as wavelets, characterized by diverse frequencies and limited durations.

The reconstructed image resulting from these compression techniques inevitably exhibits some distortion due to the loss of information during the process. However, as we will soon explore, the degree of distortion can be adjusted during the compression stage and is quantified by the PSNR curve.

2 System Description

2.1 DCT-based Image Compression

In this section, we use DCT-II, which is a separable orthonormal transformation. The DCT transform of a signal block of size M × M can be expressed by $y = AxA^T$, where x is the M × M signal block, and A is the M × M transform matrix containing elements:

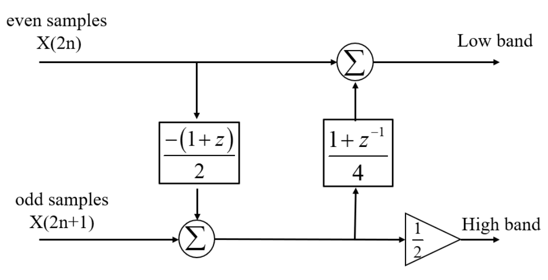
$$a_{ik} = a_i \cos(\frac{(2k+1)i\pi}{2M}) \tag{1}$$

where $i, k = 0, 1, 2, ...M-1$, $a_0 = \sqrt{\frac{1}{M}}$, $a_i = \sqrt{\frac{2}{M}}, \forall i > 0$.

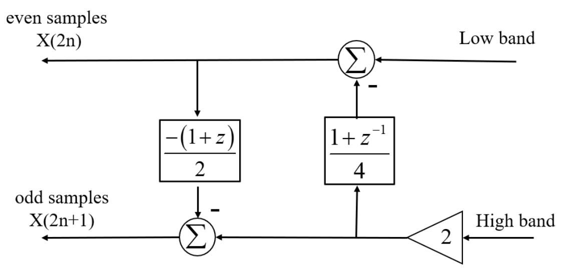
The inverse DCT (IDCT) can be implemented by $x = A^T yA$.

2.2 FWT-based Image Compression

For FWT implementation, we use the 5/3 wavelet filter bank as shown in figure 1a. There are three steps in its implementation: split, predict and update.
1. Decomposition is to divide the data into two parts, the even sequence and the odd sequence.
2. The prediction is to use the decomposed even sequence to predict the odd sequence. The obtained prediction error is the high-frequency component of the transformation.
3. The update is to update the even sequence by the prediction error to obtain the transformed low frequency



(a) Analysis filters



(b) Synthesis filters

Figure 1: FWT and iFWT structures

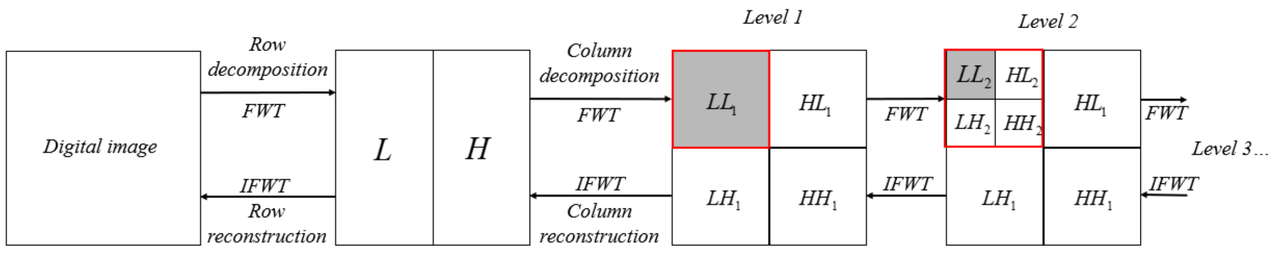


Figure 2: Analysis filters

component.
The corresponding synthesis step is the inverse transform of the analysis step.

The lifting equations of the analysis filter bank and synthesis filter bank are shown as flows:

$$\begin{cases} h(n) = \frac{X(2n+1)}{2} - \frac{X(2n)+X(2n+2)}{4} \\ l(n) = X(2n) + \frac{h(n)+h(n-1)}{2} \end{cases} \tag{2}$$

$$\begin{cases} X(2n) = l(n) - \frac{h(n)+h(n-1)}{2} \\ X(2n+1) = 2h(n) + \frac{X(2n)+X(2n+2)}{2} \end{cases} \tag{3}$$

where $x(2n)$ is the even samples and $x(2n+1)$ is the odd samples. High band $h(n)$ denotes the high-frequency output coefficients and low band$l(n)$ denotes the low-frequency output coefficients.

A biorthogonal wavelet allows for perfect reconstruction, which is always achieved with a lifting implementation. We implement the 5/3 wavelet with a lifting structure, it is a biorthogonal wavelet. Note that here we use the circular shift function circshift(A,K,M) in Matlab to handle the border effect when we filtering, where A represents the matrix to be shifted. K represents the number of shifts. M is used to decide whether to shift rows or columns.

We implement a two-dimensional wavelet transform by processing rows and columns successively. As shown in figure 2. Firstly we implement the lifting scheme for every column and obtain the low band and high band. Then we implement the lifting scheme to every row for both low and high bands, thus obtaining four subbands that represent the approximate subband, horizontal detail subband, vertical detail subband, and diagonal detail subband respectively. The approximate sub-band retains the main information of the image, while the detail sub-band contains the high-frequency information of the image, providing a richer detailed description.

2.3 Quantization and analysis

2.3.1 Uniform Quantizer

Quantization plays a key role in optimizing the video quality-bit rate trade-off. It can be divided into uniform quantization and non-uniform quantization. In this section, we will use a uniform mid-tread quantizer to implement the quantization of the coefficients.

A uniform quantizer divides the range of input values into equal intervals, assigning a representative value to each interval. It can simplify the quantization process but introduce quantization errors. Its mathematical expression is:

$$Q(x) = \lfloor \frac{x}{\Delta} \rceil \times \Delta \tag{4}$$

where x denotes the input coefficient, $\lfloor \cdot \rceil$ denotes the round operator, Δ denotes the step-size.

Figure 3 shows the uniform quantizer function when the step-size $\Delta = 4$.

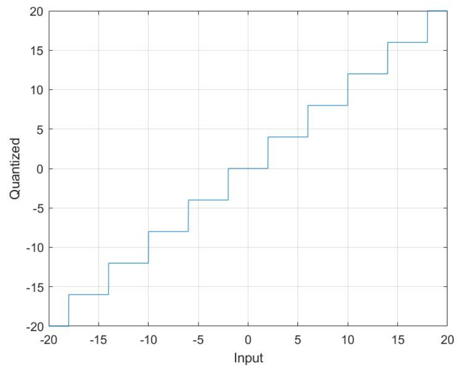


Figure 3: Quantizer function

2.3.2 Distortion and Bit-Rate Estimation

For 8-bit images, the PSNR is calculated as follows

$$PSNR = 10\log_{10}(\frac{255}{d^2}) \quad [dB] \tag{5}$$

where d is the mean squared error (MSE) between the original and the reconstructed images.

To calculate the MSE between the original and the reconstructed images, we have:

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{i=0}^{n-1}[I(i,j) - R(i,j)]^2 \tag{6}$$

where $I(i,j)$ denotes the original image, $R(i,j)$ denotes the reconstructed image.

Based on Shannon Theorem: the use of distortion-free optimal source coding allows the code word length used for each source symbol to be as small as possible, but its lower bound is the entropy of the original source. It gives the critical value of data compression under lossless conditions. Therefore, in this section, we can assume that the ideal code word length for variable length coding (VLC) is the entropy. Also because we use a special code for each of the 64 coefficients in a block, the final bit rate should be the average of these entropies. The entropy can be calculated as follows.

$$H(p) = -\sum_{i=0}^{255} p(i)\log_{10} p(i) \tag{7}$$

where $p(i)$ denotes the probability of the grey level i.

3 Results

3.1 DCT-based Image Compression

3.1.1 Blockwise 8×8 DCT

According to the formula 1, the Matrix A is as follows:

$$A = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix} \tag{8}$$

3.1.2 Distortion and Bit-Rate Estimation

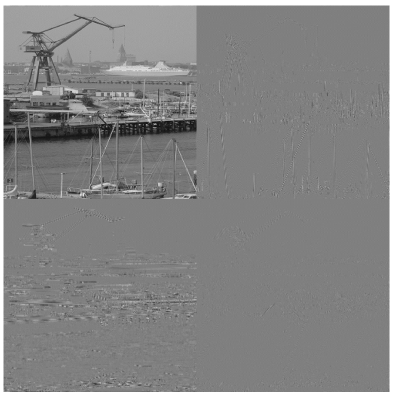
After comparing d with the mean squared error between the original and the quantized DCT coefficients, we find that they have the same value. This is because the DCT transform is a separable orthonormal transformation. The DCT transform itself does not introduce more distortion or preserve more bits.

To prove this conclusion, we calculate d and the MSE between the original and the quantized DCT coefficients in the image "pepper", both of which have a value of 0.0832.

3.2 FWT-based Image Compression

For this part, a two-dimensional FWT and inverse FWT are implemented over a image using the lifting structure and the 5/3 wavelet. The algorithm can be used to perform an arbitrary scale of FWT over an image whose length and width are two to the power of an arbitrary number. Figure 4 shows the effect of 5/3 2-D wavelet and the reconstructed image without quantization. It also can be seen in the image 4a the high frequency components in vertical, horizontal direction and both combined respectively.

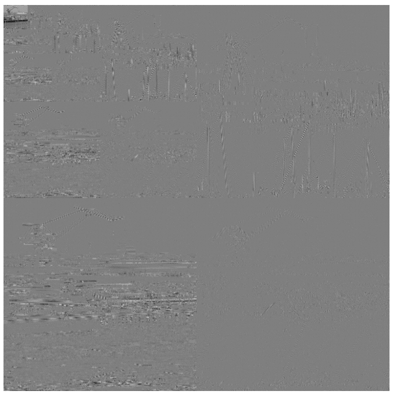
Afterwards, the coefficients of scale-4 wavelet are calculated before quantization and iFWT to reconstruct


(a) scale-1 FWT

(b) reconstructed image

Figure 4: 2-D FWT and iFWT

the image. Figure 5 shows the FWT coefficients after a scale-4 FWT and the reconstructed image when the coefficients are quantized with a step-size of 16. It is clear that after compression i.e. quantized, the image reconstructed would lose some detail information and the quality would be worsened.


(a) scale-4 FWT

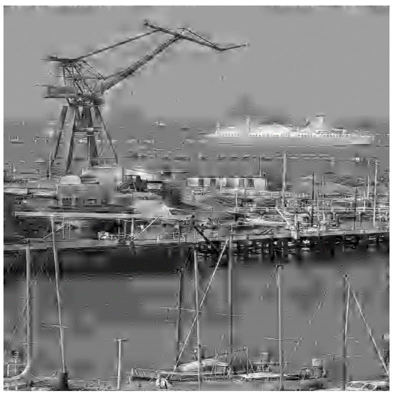

(b) reconstructed image (quantization step size 16)

Figure 5: 2-D FWT and iFWT

3.3 Distortion and Bit-Rate Estimation

A comparison of the mean square error of quantizer and the average distortion are performed in this part of the two aforementioned transform methods, as is shown in figure 6.
While they are equal when DCT are used to transform, there lie some differences when a 5/3 wavelet is used to

transform before quantization. The comparison of the average distortion and the quantization error are shown in the following table. Clearly, it is not either equal or similar as it should be. The reason to this is that the lifting wavelet structure we used to implement is not unitary transform, so there can ba a change in scale between the error introduced when quantizing and the final distortion. This can be proved by a calculate of the power. The average power of the initial image was 17440. However, the power of wavelet coefficients before quantization is 118.0275. The power change when performing the transform implies that our wavelet is not a unitary transform, which might be the explanation of the great difference. Also, the 5/3 wavelet is not an orthogonal transform, so there can be slight difference even if they are normalized.

Table 1: Average distortion d and Quatization error for the images (step-size from 2^0 to 2^9)

Average distortion d	0.7929	2.8733	9.7446	31.7854	94.1993
Quatization error	0.0797	0.2802	0.9275	2.8974	8.2289
Average distortion d	252.7804	689.6827	1094.8553	8378.7602	17440.2716
Quatization error	20.0909	40.3440	53.6614	93.5120	118.0275

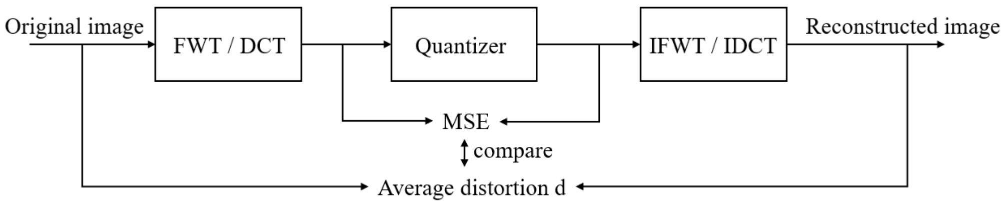


Figure 6: the comparison between two MSEs

A rate-PSNR curve is shown in figure 7. By comparison, it can be inferred that in this case DCT has a better performance than FWT. However, this is a rather preliminary conclusion since FWT is not normalized, which results in a amplification of the quantization error. This leads to bad performance of FWT comparing to DCT. Hence, to derive a more mature result, a normalized FWT structure is needed.

By studying the two rate-PSNR curves, we can observe that, in this case, the DCT transform performs better than the FWT transform. This is rather a premature conclusion to make. However, despite all above, it is still clear that for each decreased bit-rate, the PSNR decrease by 6dB as it should be, which implies that a reduction of the bit-rate decreases the image quality.

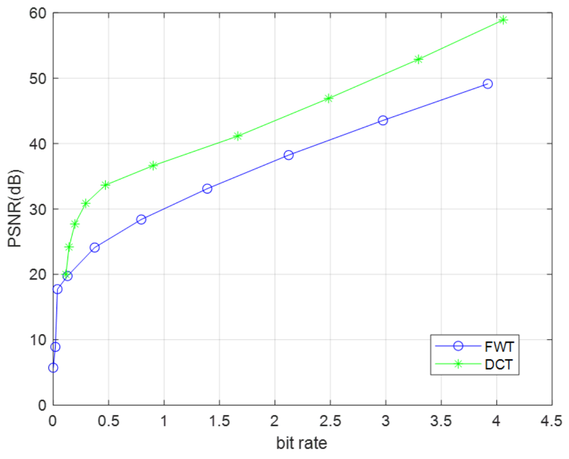


Figure 7: the curve of PSNR and bit-rate

4 Conclusions

In this project, we evaluate the achievable performance of two transform based image compression algorithms, DCT and FWT. We study the DCT-based and FWT-based image compression respectively. After implementing the quantization of the coefficients, we calculate the PSNR and the bit rate required to encode the coefficients. Based on the results, we find that both of the methods can reconstruct images perfectly.

Appendix

Who Did What

	contribution
Letao Feng	33.3...%
Heng Zhao	33.3...%
Zhenghao Li	33.3...%

MatLab code

Include the well documented MatLab code that you have used.

```
\textbf{1.DCT}

M=8;
A=zeros(M, M);
for i=0:M-1
    for k=0:M-1
        if i==0
            alpha=sqrt(1/M);
        else
            alpha=sqrt(2/M);
        end
        A(i+1,k+1)=alpha*cos(((2*k+1)*i*pi)/(2*M));
    end
end

% Display the DCT matrix A
disp('DCT Matrix A:');
disp(A);


image=double(imread("images\peppers512x512.tif"));
Blocks=zeros(8,8,64*64);
for i=1:64
    for k=1:64
        Blocks(:,:,64*(k-1)+i)=image((k-1)*8+1:(k-1)*8+8,(i-1)*8+1:(i-1)*8+8);
    end
end
DCT_co=zeros(8,8,64*64);
for i=1:4096
    DCT_co(:,:,i)=A*Blocks(:,:,i)*A';
end

Blocks_recstr=zeros(8,8,64*64);
en=zeros(1,4096);
PSNR=zeros(1,10);
bit_rate=zeros(1,10);
for l=0:9
    DCT_co_qtzd=midtread(DCT_co,2^l);
    for i=1:4096
        Blocks_recstr(:,:,i)=A'*DCT_co_qtzd(:,:,i)*A;
        en(i)=entr(DCT_co_qtzd(:,:,i));
    end
    d=sum(sum(sum((Blocks-Blocks_recstr).^2)))/(512*512);
    PSNR(l+1)=10*log10(255^2/d);
    bit_rate(l+1)=mean(en);
end
```

```
plot(bit_rate,PSNR,'ob-');
grid on

\textbf{2.quantization}

function quant = midtread(x,steplen)
% uniform mid-tread quantizer function
quant = round(x/steplen)*steplen;
end

\textbf{3.entropy}

function y = entr(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);
    y = -sum(pixel_probas.*log_pixel_probas);
end

\textbf{4.FWT(main)}

clear
clc
close all
image=double(imread('harbour512x512.tif'));
fwt_coeff=fwt_M_scale(image,4);
imshow(uint8(fwt_coeff))

%BITRATE
PSNR=zeros(1,10);
bit_rate=zeros(1,10);
dd=zeros(1,10);

for l=0:9
    coeff_qtzd=midtread(fwt_coeff,2^l);
    % figure
    % imshow(coeff_qtzd);
    image2=ifwt(coeff_qtzd,4);
    figure
    imshow(uint8(image2))

    d=sum((image-image2).^2,"all")/(512*512);
    dd(l+1)=d;
    PSNR(l+1)=10*log10(255^2/d);

    bit_rate(l+1)=entr(coeff_qtzd);
end
figure
plot(bit_rate,PSNR,'ob-');
grid on

coeff_qtzd=midtread(fwt_coeff,2^0);
image2=ifwt(coeff_qtzd,4);
e=uint8(image2-image);
figure
imshow(e)

\textbf{5.FWT}
```

```
function fwt_coeff = fwt_M_scale(image,I)
% do a M-scale FWT
J=I;
[M,N]=size(image);
while J>0
    A=image(1:M/(2^(I-J)),1:N/(2^(I-J)));
    A=FWT(A);
    image(1:M/(2^(I-J)),1:N/(2^(I-J)))=A;
    J=J-1;
end
fwt_coeff=image;
end

function [output_image]=FWT(f)
%column
x2n_column=f(:,1:2:end);
x2nplus1_column=f(:,2:2:end);

high_col=1/2*(x2nplus1_column - 0.5*x2n_column - 0.5*circshift(x2n_column,-1,2));
low_col=1*(x2n_column + 0.25*2*high_col + 0.25*2*circshift(high_col,1,2));

%row
x2n_row1=low_col(1:2:end,:);
x2nplus1_row1=low_col(2:2:end,:);

high21=1/2*(x2nplus1_row1 - 0.5*x2n_row1 - 0.5*circshift(x2n_row1,-1,1));
low11=1*(x2n_row1 + 0.25*2*high21 + 0.25*2*circshift(high21,1,1));

x2n_row2=high_col(1:2:end,:);
x2nplus1_row2=high_col(2:2:end,:);
high22=1/2*(x2nplus1_row2 - 0.5*x2n_row2 - 0.5*circshift(x2n_row2,-1,1));
low12=1*(x2n_row2 + 0.25*2*high22 + 0.25*2*circshift(high22,1,1));

out1=[low11,low12];
out2=[high21,high22];
output_image=[out1;out2];
end

\textbf{6. IFWT}

function image = ifwt(fwt_coeff,I)
% do a M-scale iFWT
J=I;
[M,N]=size(fwt_coeff);
while J>0
    LL=fwt_coeff(1:M/(2^J),1:N/(2^J));
    LH=fwt_coeff(M/(2^J)+1:M/(2^(J-1)),1:N/(2^J));%-128;
    HL=fwt_coeff(1:M/(2^J),N/(2^J)+1:N/(2^(J-1)));%-128;
    HH=fwt_coeff(M/(2^J)+1:M/(2^(J-1)),N/(2^J)+1:N/(2^(J-1)));%-128;
    recon_image=IFWT(LL,HL,LH,HH);
    fwt_coeff(1:M/(2^(J-1)),1:N/(2^(J-1)))=recon_image;
    J=J-1;
end
image=fwt_coeff;

end
```

```
function[recon_image]=IFWT(LL,HL,LH,HH)
x2n_L=1*LL-0.25*2*LH-0.25*2*circshift(LH,1,1);
x2nplus1_L=2*LH+0.5*x2n_L+0.5*circshift(x2n_L,-1,1);
nColumns = size(x2n_L,2);
x1 = [x2n_L,x2nplus1_L]';
x1 = reshape(x1(:),nColumns,[])';


x2n_R=1*HL-0.25*2*HH-0.25*2*circshift(HH,1,1);
x2nplus1_R=2*HH+0.5*x2n_R+0.5*circshift(x2n_R,-1,1);
nColumns = size(x2n_R,2);
x2 = [x2n_R,x2nplus1_R]';
x2 = reshape(x2(:),nColumns,[])';

x2n=1*x1-0.25*2*x2-0.25*2*circshift(x2,1,2);
x2nplus1=2*x2+0.5*x2n+0.5*circshift(x2n,-1,2);

[nRows,nClosB] = size(x2n);
nClosA = size(x2nplus1,2);
x = zeros(nRows,nClosA+nClosB);
x(:,1:2:end) = x2n;
x(:,2:2:end) = x2nplus1;
recon_image=x;

end
```

References

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