Project 2: Image Compression

EQ2330 Image and Video Processing, Project 2

Ata Yavuzyilmaz
atay@kth.se

Tom Diener
tdiener@kth.se

December 16, 2022

Summary

The project evaluates the application of two different transforms, namely the discrete cosine transform and the fast wavelet transform, to be used in an image compression algorithm. Due to necessary quantization of the transform coefficients, the image reconstruction is accompanied by a loss of quality. The measurements of rate-PSNR curves for a certain data set of images show that a higher compression can be reached with a less precise quantization. Nevertheless, this results in a poorer quality for the reconstructed image. Therefore, a trade-off between image quality and compression has to be found.

1 Introduction

The aim of this project is to evaluate achievable performance of two transform based image compression algorithms. The transforms are the discrete cosine transform (DCT) and the fast wavelet transform (FWT). The first one was used in the JPEG standard until the year 2000, until it was replaced by the discrete wavelet transform which became part of the new standard JPEG2000.

The idea of compression due to using a transform is to represent the whole image with a number of coefficients which can be coded more efficiently. Those coefficients can be calculated based on the transform algorithm. The transforms itself would allow to reconstruct the image perfectly, but due to limited precision in reality, the transform coefficients representing the image have to be quantized. That leads to distortions in the reconstructed image. If different degrees of quantization are chosen, namely the step size is altered, the distortion can be evaluated in context with the respective coding rate.

2 DCT-based Image Compression

2.1 Blockwise 8 x 8 DCT

Since it would be computationally to costly to calculate the DCT for the whole image, instead a blockwise-processing approach is chosen. Therefore, the image is split into adjacent blocks of size of 8 x 8 and the DCT is calculated for each of those blocks individually with 1 using DCT-II.

$$a_{ik} = \alpha_i cos(\frac{(2k+1)i\pi}{2M}) \text{ for } i,k = 0,1,...,M-1 \tag{1}$$

with

$$\alpha_0 = \sqrt{\frac{1}{M}} \tag{2}$$

$$\alpha_i = \sqrt{\frac{2}{M}}, \quad \forall i > 0 \tag{3}$$

Due to the fact that the DCT is a separable orthonormal transform, the DCT transform of a signal block of size 8×8 can be calculated by $y = AxA^T$ and accordingly the inverse DCT transform is $x = A^T y A$, where x is the 8×8 signal block and A is the 8×8 transform matrix containing the elements

$$A = \begin{bmatrix} 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 & 0.3536 \\ 0.4904 & 0.4157 & 0.2778 & 0.0975 & -0.0975 & -0.2778 & -0.4157 & -0.4904 \\ 0.4619 & 0.1913 & -0.1913 & -0.4619 & -0.4619 & -0.1913 & 0.1913 & 0.4619 \\ 0.4157 & -0.0975 & -0.4904 & -0.2778 & 0.2778 & 0.4904 & 0.0975 & -0.4157 \\ 0.3536 & -0.3536 & -0.3536 & 0.3536 & 0.3536 & -0.3536 & -0.3536 & 0.3536 \\ 0.2778 & -0.4904 & 0.0975 & 0.4157 & -0.4157 & -0.0975 & 0.4904 & -0.2778 \\ 0.1913 & -0.4619 & 0.4619 & -0.1913 & -0.1913 & 0.4619 & -0.4619 & 0.1913 \\ 0.0975 & -0.2778 & 0.4157 & -0.4904 & 0.4904 & -0.4157 & 0.2778 & -0.0975 \end{bmatrix} \tag{4}$$

2.2 Quantization with uniform quantizer

In equation 4, the transform coefficients are given, which need to be quantized as described in Section 1. In this project, a uniform mid-tread quantizer without threshold characteristic with the same step-size for all coefficients was used.

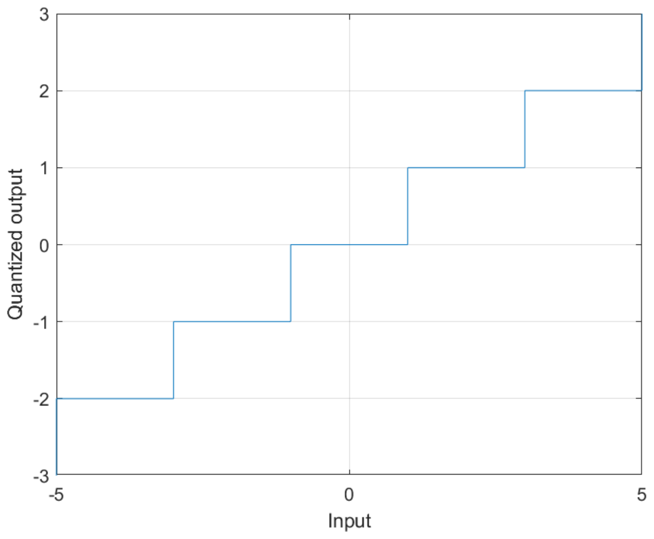


Figure 1: Plot of the used uniform mid-tread quantizer.

2.3 Distortion and Bit-Rate Estimation

The mean squared error (MSE) between the original and the reconstructed image is introduced as the distortion d. This difference results from quantizing the DCT coefficients. It can be shown mathematically and computationally that the MSE between the quantized coefficients and the original ones, and the distortion have the same value since the transform preserves the MSE. This distortion can be influenced by the step size of the quantizer where a smaller step size gives a higher precision and consequently a lower distortion.

To measure the quality of the reconstructed images, the PSNR value is useful. This value can be calculated as

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

based on the distortion.

To estimate the coding bit rate, it is assumed that each coefficient of a block is encoded by an individual variable length code (VLC) with the ideal code word length. The latter is estimated by the lower bound, the entropy of each coefficient. Huffman codes as an example for VLCs are optimal prefix-codes that come as close as possible to the minimum value of rate which is equal to the entropy [2]. Therefore, in this case the entropy can be used to estimate the bit rate of the encoding. Since each coefficient is encoded individually, the entropy of the VLC can be calculated by using the statistics for the values of the same coefficient for each block of all three images in the given data set. To obtain a representative estimated bit rate for one coefficient, the entropies for all 64 coefficients of each block can be averaged.

To measure the rate-PSNR curve of the DCT on the data set, the rate is determined for different quantization steps sizes 2^i, for $i = 0, 1, ..., 9$, which produce different distortions and thus also different PSNRs. The results of those measurements are discussed in Section 4, in Figure 3a.

3 FWT-based Image Compression

With a 2-band FWT transform, it is able to obtain the low frequency and high frequency parts of the signal separately. For the FWT transform, the direct implementation method is employed, where upsampling, downsampling, and filtering operations are involved. For this, the functions for 1-dimensional FWT are written first, and then it is modified for 2-dimensional FWT.

3.1 The Two-Band Filter Bank

The two-band filter bank has 2 different filters. One of these filters (which is the prototype scaling vector h_ϕ) is used to get the low frequency parts of the signal, and the other filter (the wavelet vector h_ψ) is used to get the high frequency parts of the signal. For some biorthogonal wavelet filters such as Daubechies, the wavelet vector can be obtained from the prototype scaling vector from the formula [1]

$$h_\psi(n) = (-1)^n h_\psi(N - 1 - n), \quad n = 0, 1, ...N - 1. \tag{6}$$

For the 1-D FWT, the input vector is first periodically extended so that there are no bordering effects when the convolution is performed. After convolving the replicated signal with the analysis filters, which are prototype scaling vector and the wavelet vector, the two obtained signals are downsampled by 2. The FWT is now complete. The result is two vectors with half the size of the input vector, which are low and high frequency parts of the signal.

The 1-D inverse FWT is also similar. The low and high frequency parts of the signal are first upsampled by 2. Then, the vectors are convolved with the low-band and high-band synthesis filters. The low-band synthesis filter is the inversed version of the prototype scaling vector, and the high-band synthesis filter is obtained from the low-band synthesis filter, using the formula 6. The resulting signals are added and the inverse FWT is complete. Note that, because of our implementation, we had to shift the output signal by 1 to get the same signal as the input signal in the spatial domain.

3.2 The 2-D FWT

The 2-D FWT implementation is different than the 1-D case. The FWT is applied to the input image twice, in different directions. First, 1-D FWT is applied to the each row of the image. For an image with size 512x512, the result is two matrices with size 512x256. The first matrix is the low frequency part, the second matrix is the high frequency part of the image. Then, 1-D FWT is applied again to the columns of these two matrices. The result are four matrices with size 256x256. The matrices represent Low-Low, High-Low, Low-High, and High-High parts of the image, depending on the direction.

After applying the 2-D FWT to the image *harbour* 4 times consecutively, the scale 4 FWT coefficients are shown in Figure 2.

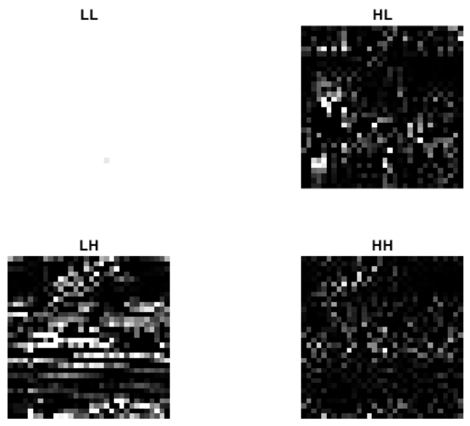


Figure 2: Scale 4 FWT coefficients of the image *harbour*.

These coefficients are quantized with the quantizer described in Section 2.2.

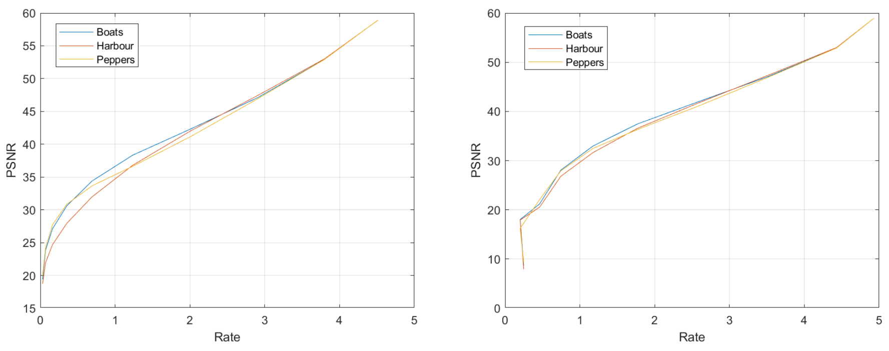
3.3 Distortion and Bit-Rate Estimation

After quantizing the FWT coefficients and reconstructing the original image, the distortion d of the images is calculated, which is the mean squared error between the original image and reconstructed image. Then, it is compared with the mean squared error d_q between the original FWT and quantized coefficients.

For this part, 2-D FWT is applied once to the images. It is found that the distortion d is equal to the average of the mean squared error between the FWT coefficients and their quantized versions. Since a biorthogonal filter is used, when the FWT coefficients are quantized, the overall MSE should not change. If there were no quantizations, the image would be completely restored. However, due to the biorthogonal filtering, the average MSE for the coefficients is equal to the MSE of the image.

$$d = (d_{qLL} + d_{qHL} + d_{qLH} + d_{qHH})/4 \tag{7}$$

To draw the PSNR per bitrate graph, it is needed to estimate the bitrate of the image, similar to Section 2.3, the bitrate is equal to the entropy. Since each subband is encoded individually for the VLC, the entropy is calculated for each subband. Also, for each image in the dataset, each subband is encoded with the same VLC. Therefore the entropies of each subband of all images are combined. The resulting four entropies are averaged. The PSNR is calculated



(a) PSNR-rate of DCT (b) PSNR-rate of FWT

Figure 3: PSNR-rate curves

using the Formula 5. The process is iterated for each quantization step 2^i, for $i = 0, 1, ..., 9$. The plot is given in Section 4, in Figure 3b.

4 Results

On the basis of the aforementioned quantization and the explained procedure to estimate the bit-rate, the rate-PSNR curves in Figure 3 for the data set were measured for both DCT and FWT. Both show the same behaviour of an increasing PSNR for a higher bit-rate.

As mentioned before, a lower step size implies a less precise quantization and thus a higher distortion. Since PSNR and distortion are inversely correlated (see Equation 5), a decrease of the step size reduces the PSNR as well.

With those conclusions, Figure 3 can be interpreted as higher quality is reached for investing more bits for quantizing the data. On the other hand it says that a less precise quantization, resulting in higher distortion and lower PSNR, leads to higher compression because less values need to be encoded. For practical usage of the compression algorithm, a trade-off between compression and image quality has to be found. One option could be to set a threshold for the PSNR to guarantee a certain quality and then choose the transform with the better compression result for the particular image.

5 Conclusions

Two transform based image compression algorithms are implemented in this project, and the compression results are compared using the PSNR-rate curves. Higher quantization step size results in a more aggressive compression, since more coefficient values are rounded to the same value. However, quantization destroys image information. Higher quantization step sizes destroy more information from the image, hence the distortion of the image increases. In other words, a more aggressive compression results in a more distorted image.

From the PSNR-rate curves, we can see that for the same rate, PSNR of DCT is higher than FWT. For the DCT, each coefficient of the transformed 8x8 block is encoded together with the other blocks. For a 512x512 image, this results in 4096 coefficients. For FWT, each subband is encoded individually. But the encoded blocks have a size of 256x256. This means that we encode more values for each subband, and this results in a higher entropy and a higher rate. Therefore, to encode the image with same quality, FWT needs higher rate than DCT.

Appendix

MatLab code

Code for part 2:

```
%% 8x8 DCT block
% calculate the A matrix
M = 8; % Block size
A = zeros(M,M);

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

% write functions for blockwise operation
dct = @(block_struct) A * block_struct.data * A';
inverse_dct = @(block_struct) A' * block_struct.data * A;
ent = @(block_struct) reshape(transpose(block_struct.data), 1, []);
% prepare sorting the coefficients for entropy calculation

% print A values
disp(A)

%% apply DCT to an image
img = double(imread("images\airfield512x512.tif"));
dct_result = blockproc(img, [8 8], dct);

%% uniform quantizer, unlimited number of quantization steps
stepsize = 2;
quantized = stepsize * round(dct_result/stepsize);

syms q(x)
q(x) = stepsize * round(x/stepsize);
figure
fplot(q)
grid on
xlabel("Input")
ylabel("Quantized output")

%% reconstruct the image and calculate the MSE
reconstructed = blockproc(quantized, [8 8], inverse_dct);
d = immse(reconstructed, img);
d_quantizer = immse(quantized, dct_result);
% distortion of the images and DCT coefficients are equal.

%% calculate rate and PSNR and plot the graph
% get images in the dataset
```

```
img_boat = double(imread("images\boats512x512.tif"));
img_peppers = double(imread("images\peppers512x512.tif"));
img_harbour = double(imread("images\harbour512x512.tif"));
% perform DCT to images
dct_boat = blockproc(img_boat, [8 8], dct);
dct_peppers = blockproc(img_peppers, [8 8], dct);
dct_harbour = blockproc(img_harbour, [8 8], dct);

% quantize the images with different step sizes
% distortion for different step sizes
d_b = zeros(1,10);
d_p = zeros(1,10);
d_h = zeros(1,10);
entro = zeros(1,10);

% calculate distortion and bite rate for step sizes 2^0 to 2^9
for i = 0:9
    stepsize =  2^i;

    quantized_b = stepsize * round(dct_boat/stepsize);
    quantized_p = stepsize * round(dct_peppers/stepsize);
    quantized_h = stepsize * round(dct_harbour/stepsize);

    % calculate the distortion as mse of the image
    d_b(i+1) = immse(quantized_b, dct_boat);
    d_p(i+1) = immse(quantized_p, dct_peppers);
    d_h(i+1) = immse(quantized_h, dct_harbour);

    [n,m] = size(quantized_h);
    sz = n*m;

    % transform coefficients of each block into a matrix of 64 rows
    ent_b = blockproc(quantized_b, [8 8], ent);
    ent_b = reshape(ent_b', 64, sz/64)';
    ent_p = blockproc(quantized_p, [8 8], ent);
    ent_p = reshape(ent_p', 64, sz/64)';
    ent_h = blockproc(quantized_h, [8 8], ent);
    ent_h = reshape(ent_h', 64, sz/64)';
    en_all = [ent_b; ent_p; ent_h];
    % sum over the entropies of all coefficient to get the total entropy of
    % a block
    for m = 1:64
        entro(i+1) = entro(i+1) + entropy(mat2gray(en_all(:,m)))/64;
    end
end

% calculate PSNR
p_b = 10*log10(255^2./d_b);
p_h = 10*log10(255^2./d_h);
p_p = 10*log10(255^2./d_p);

% plot the figures
figure
plot(d_b, entro)
```

```
hold on
grid on
plot(d_h,entro)
plot(d_p,entro)
legend(["Boats", "Harbour", "Peppers"])
xlabel("Distortion")
ylabel("Rate")

figure
plot(entro, p_b)
hold on
grid on
plot(entro, p_h)
plot(entro, p_p)
legend(["Boats", "Harbour", "Peppers"])
ylabel("PSNR")
xlabel("Rate")
```

Code for part 3:

```
clear
%% apply FWT with Daubechies 8-tap filter to image harbour
% https://wavelets.pybytes.com/wavelet/db8/
db8 = [-0.00011747678400228192
0.0006754494059985568
-0.0003917403729959771
-0.00487035299301066
0.008746094047015655
0.013981027917015516
-0.04408825393106472
-0.01736930100202211
0.128747426620186
0.00047248457399797254
-0.2840155429624281
-0.015829105256023893
0.5853546836548691
0.6756307362980128
0.3128715909144659
0.05441584224308161
];

img = double(imread("images\harbour512x512.tif"));

[yll, yhl, ylh, yhh] = fwt2d(img, db8);

i = 1:length(db8);
inv_db8 = db8(length(db8)-i+1);

result = inv_fwt2d(yll, yhl, ylh, yhh, inv_db8);

% figure
% imshow(uint8(img_reconst))

% compare the two images if they are equal, to test if our FWT
% implementation works.
```

```
diff = immse(result, img);
disp(diff)

%% scale 4 2D-FWT implementation
% To compute the wavelet coefficients at scale 4, we need 4 filter banks.

% filter bank 1
[yll1, yhl1, ylh1, yhh1] = fwt2d(img, db8);

% filter bank 2 to the Low Low part of the FWT
[yll2, yhl2, ylh2, yhh2] = fwt2d(yll1, db8);

% filter bank 3
[yll3, yhl3, ylh3, yhh3] = fwt2d(yll2, db8);

% filter bank 4
[yll4, yhl4, ylh4, yhh4] = fwt2d(yll3, db8);

% plot the results
figure
tiledlayout(2,2)

nexttile
imshow(uint8(yll4))
title("LL")

nexttile
imshow(uint8(yhl4))
title("HL")

nexttile
imshow(uint8(ylh4))
title("LH")

nexttile
imshow(uint8(yhh4))
title("HH")

%% Uniform Quantizer
stepsize = 2;

yll4_q = stepsize * round(yll4/stepsize);
yhl4_q = stepsize * round(yhl4/stepsize);
ylh4_q = stepsize * round(ylh4/stepsize);
yhh4_q = stepsize * round(yhh4/stepsize);

% % plot the quantized results
% figure
% tiledlayout(2,2)
% title("Quantized Coefficients")
%
% nexttile
% imshow(uint8(yll4_q))
% title("LL")
```

```
%
% nexttile
% imshow(uint8(yhl4_q))
% title("HL")
%
% nexttile
% imshow(uint8(ylh4_q))
% title("LH")
%
% nexttile
% imshow(uint8(yhh4_q))
% title("HH")

%% Distortion and Bit-Rate Estimation

% get images in the dataset
img_boat = double(imread("images\boats512x512.tif"));
img_peppers = double(imread("images\peppers512x512.tif"));
img_harbour = double(imread("images\harbour512x512.tif"));

% apply 2d FWT to images
[b_ll, b_hl, b_lh, b_hh] = fwt2d(img_boat, db8);
[p_ll, p_hl, p_lh, p_hh] = fwt2d(img_peppers, db8);
[h_ll, h_hl, h_lh, h_hh] = fwt2d(img_harbour, db8);

d_b = zeros(1,10);
d_p = zeros(1,10);
d_h = zeros(1,10);

% e_b = zeros(1,10);
% e_p = zeros(1,10);
% e_h = zeros(1,10);
entro = zeros(1,10);
for i = 0:9
    % apply quantization
    stepsize = 2^i;
    q_b_ll = stepsize * round(b_ll/stepsize);
    q_b_hl = stepsize * round(b_hl/stepsize);
    q_b_lh = stepsize * round(b_lh/stepsize);
    q_b_hh = stepsize * round(b_hh/stepsize);

    q_p_ll = stepsize * round(p_ll/stepsize);
    q_p_hl = stepsize * round(p_hl/stepsize);
    q_p_lh = stepsize * round(p_lh/stepsize);
    q_p_hh = stepsize * round(p_hh/stepsize);

    q_h_ll = stepsize * round(h_ll/stepsize);
    q_h_hl = stepsize * round(h_hl/stepsize);
    q_h_lh = stepsize * round(h_lh/stepsize);
    q_h_hh = stepsize * round(h_hh/stepsize);

    % reconstruct images
    b_recons = inv_fwt2d(q_b_ll, q_b_hl, q_b_lh, q_b_hh, inv_db8);
    p_recons = inv_fwt2d(q_p_ll, q_p_hl, q_p_lh, q_p_hh, inv_db8);

    h_recons = inv_fwt2d(q_h_ll, q_h_hl, q_h_lh, q_h_hh, inv_db8);

    % calculate distortion
    d_b(i+1) = immse(b_recons, img_boat);
    d_p(i+1) = immse(p_recons, img_peppers);
    d_h(i+1) = immse(h_recons, img_harbour);

    % calculate the entropy (bit-rate estimation)
    entro(i+1) = (entropy(mat2gray([q_b_ll q_p_ll q_h_ll]))
    + entropy(mat2gray([q_b_hl q_p_hl q_h_hl]))
    + entropy(mat2gray([q_b_lh q_p_lh q_h_lh]))
    + entropy(mat2gray([q_b_hh q_p_hh q_h_hh])))/4;
end

% compare d and difference between wavelet coefficients and their quantized
% versions

q_dist = (immse(b_ll, q_b_ll) + immse(b_hl, q_b_hl) + immse(b_lh, q_b_lh)
+ immse(b_hh, q_b_hh))/4;
disp(d_b(end))
disp(q_dist)

% The distortion between the original and reconstructed image is equal to
% the average of the distortion of the wavelet coefficients and their
% quantized versions.

% calculate PSNR
p_b = 10*log10(255^2./d_b);
p_h = 10*log10(255^2./d_h);
p_p = 10*log10(255^2./d_p);

% plot the figures
figure
plot(d_b, entro)
hold on
grid on
plot(d_h,entro)
plot(d_p,entro)
legend(["Boats", "Harbour", "Peppers"])
xlabel("Distortion")
ylabel("Rate")

figure
plot(entro, p_b)
hold on
grid on
plot(entro, p_h)
plot(entro, p_p)
legend(["Boats", "Harbour", "Peppers"])
ylabel("PSNR")
xlabel("Rate")
```

```
% FWT filter bank with direct implementation
function [y] = fwt(x, h)
    x = reshape(x, [length(x),1]);
    % find wavelet vector from scaling vector
    N = length(h);
    hw = zeros(N,1);
    for i = 1:N
        hw(i) = (-1)^i * h(N-i+1);
    end

    %disp(hw)
    % FWT direct implementation

    % extend the input signal
    x_ext = [x; x; x];

    %disp(size(x_ext))
    % apply the filters
    y0 = conv(x_ext, h, "same");
    y0 = y0(length(x)+1:end-length(x));
    y0 = downsample(y0, 2);

    y1 = conv(x_ext, hw, 'same');
    y1 = y1(length(x)+1:end-length(x));
    y1 = downsample(y1, 2);

    y = [y0; y1];
end

% inverse FWT
function [x] = inv_fwt(y, h)
    y = reshape(y, [length(y),1]);
    % find wavelet vector from scaling vector
    N = length(h);
    hw = zeros(N,1);
    for i = 0:N-1
        hw(i+1) = (-1)^i * h(N-i);
    end
    %disp(y)
    % divide the high and low band
    y0 = y(1:length(y)/2);
    y1 = y(length(y)/2+1:end);

    % upsample the signals
    y0 = upsample(y0, 2);
    y1 = upsample(y1, 2);

    % extend the input signals
    y0_ext = [y0; y0; y0];
    y1_ext = [y1; y1; y1];

    % apply the filters
    x0 = conv(y0_ext, h, 'same');
    x0 = x0(length(y)+1:end-length(y));
    x1 = conv(y1_ext, hw, 'same');
    x1 = x1(length(y)+1:end-length(y));

    % merge the outputs
    x = x0 + x1;
    x = circshift(x,1);
end

function [yll, yhl, ylh, yhh] = fwt2d(x, h)
    yl1 = zeros(length(x), length(x)/2);
    yh1 = zeros(length(x), length(x)/2);

    yll = zeros(length(x)/2, length(x)/2);
    yhl = zeros(length(x)/2, length(x)/2);
    ylh = zeros(length(x)/2, length(x)/2);
    yhh = zeros(length(x)/2, length(x)/2);

    for i = 1:length(x)
        y1 = fwt(x(i,:), h);
        yl1(i,:) = y1(1:length(y1)/2);
        yh1(i,:) = y1(length(y1)/2+1:end);
    end

    for j = 1:size(yh1,2)
        yl2 = fwt(yl1(:,j), h);
        yh2 = fwt(yh1(:,j), h);

        yll(:,j) = yl2(1:length(yl2)/2);
        ylh(:,j) = yl2(length(yl2)/2+1:end);
        yhl(:,j) = yh2(1:length(yh2)/2);
        yhh(:,j) = yh2(length(yh2)/2+1:end);
    end

end

function [x] = inv_fwt2d(yll, yhl, ylh, yhh, h)
    yl = zeros(2*length(yll),length(yll));
    yh = zeros(2*length(yll),length(yll));

    for i = 1:size(yll, 2)
        yl(:,i) = inv_fwt([yll(:,i); ylh(:,i)], h);
        yh(:,i) = inv_fwt([yhl(:,i); yhh(:,i)], h);
    end

    x = zeros(2*length(yll),2*length(yll));
    for j = 1:length(yl)
        x(j,:) = inv_fwt([yl(j,:) yh(j,:)], h);
    end
end
```

References

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

[2] EQ2330 Lecture Slides, *Compression Fundamentals*, Available: https://people.kth.se/ mflierl/EQ2330/09-CompressionFundamentals.pdf