Advanced Image Analysis
Wavelet Transformation

Masters in Computer Vision

UNIVERSITE DE BOURGOGNE

Centre Universitaire Condorcet - UB, Le Creusot

07 January 2018

Submitted By:
Mohit Kumar Ahuja

Supervisor:
Dr. Philippe Carre

Index

1. Introduction

The wavelet transform is similar to the Fourier transform (or much more to the windowed Fourier transform) with a completely different merit function. The **main difference** is this:

- Fourier transform decomposes the signal into sines and cosines, i.e. the functions localized in Fourier space; in contrary the wavelet transform uses functions that are localized in both the real and Fourier space. Generally, the wavelet transform can be expressed by the following equation:

$$F(a,b) = \int_{-\infty}^{\infty} f(x)\, \psi^*_{(a,b)}(x)\, \mathrm{d}x$$

* is the complex conjugate symbol and function ψ is some function**.**

As it is seen, the Wavelet transform is in fact an infinite set of various transforms, depending on the merit function used for its computation. This is the main reason, why we can hear the term "wavelet transform" in very different situations and applications. There are also many ways how to sort the types of the wavelet transforms. Here we show only the division based on the wavelet orthogonality. We can use *orthogonal wavelets* for discrete wavelet transform development and *non-orthogonal wavelets* for continuous wavelet transform development. These two transforms have the following properties:

1. The discrete wavelet transform returns a data vector of the same length as the input is. Usually, even in this vector many data are almost zero. This corresponds to the fact that it decomposes into a set of wavelets (functions) that are orthogonal to its translations and scaling. Therefore we decompose such a signal to a same or lower number of the wavelet coefficient spectrum as is the number of signal data points. Such a wavelet spectrum is very good for signal processing and compression, for example, as we get no redundant information here.

2. The continuous wavelet transform in contrary returns an array one dimension larger than the input data. For a 1D data we obtain an image of the time-frequency plane. We can easily see the signal frequencies evolution during the duration of the signal and compare the spectrum with other signals spectra. As here is used the non-orthogonal set of wavelets, data are highly correlated, so big redundancy is seen here. This helps to see the results in a more humane form.

2. Problem Statement

This problem investigates the wavelet transform and its application to image denoising.

1. *Develop a Matlab function for computing the J-level wavelet transform of an NxN image (assume N is a power of 2).*

   Your function can repeatedly use the instruction that I wrote and discussed in class (down sampling, filtering, computation of the high pass-filter), or you can start from scratch. Your function should take three arguments: an input image, the number of levels J you wish to compute and the analysis low pass filter. Your function should output an array of NxN wavelet coefficients (in the arrangement discussed in class).

2. *Develop a Matlab function for computing the inverse J-level wavelet transform of an NxN array of wavelet coefficients.*

   Your function could repeatedly use the instruction that I wrote and discussed in class (upsampling, filtering, …). Your function should take three arguments: an input array of wavelet coefficients, the number of levels J in that array and the **analysis** low-pass filter. Your function should output an image reconstructed from the input coefficients.

3. *Test your forward and inverse transform code by applying it to the Lena image with the Daubechies D4 filter:*

   h0 = [0.48296 0.83652 0.22414 -0.12941];
   You can test it with other (grayscale) images if you want, too.

4. *Image Denoising: analyse of the denoising performance of the wavelet transform.*

   Add a small amount of Gaussian white noise (with variance varying from 2 to 20), compute the transforms, set all coefficients to zero except those whose magnitude is larger than 3s (you can also try factors other than 3), and reconstruct an estimate of the original image by applying the corresponding inverse transform. Note that we have used the Hard threshold

$$T^{hard}\left(d_{l,k}\right)=\begin{cases} d_{j,k} \text{ if } \left|d_{l,k}\right| \geq \lambda \\ 0 \text{ if } \left|d_{l,k}\right| < \lambda \end{cases}$$

   Compare the Denoising performance of the Soft Threshold of the wavelet transform with the Hard Threshold of the wavelet transform using the test image(s).

$$T^{soft}\left(d_{j,k}\right)=\begin{cases} sign\left(d_{j,k}\right)\left(\left|d_{j,k}\right|-\lambda\right) \text{if } \left|d_{j,k}\right| \geq \lambda \\ 0 \text{ if } \left|d_{j,k}\right| < \lambda \end{cases}$$

3. Decomposition

There are two main steps in Wavelet Transformation which are:

1. Decomposition
2. Reconstruction

So, in the decomposition part we decompose the signal using two Filters:

1. High Pass Filter – G0
2. Low Pass Filter – H0

The schematic diagram is shown in figure 1. Where you can see the decomposition process of a signal. We apply high-pass filter (HPF) and low-pass filter (LPF) then we do the down sampling and we obtain the 1st scale and if we want to extend it we can furthermore extend the same till the level we want.

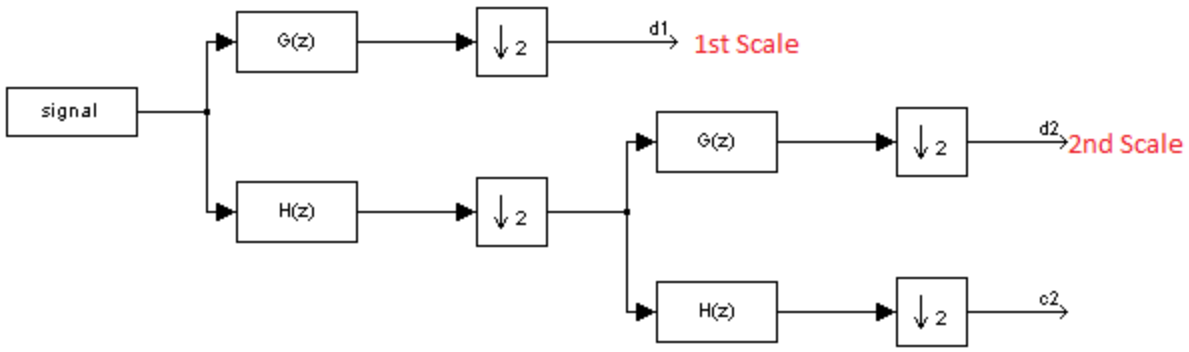

Figure 1: Schematic diagram of 2D wavelet transform

3.1 Matlab Function

I made a Matlab function named "`Compute_Wavelet_Coefficient`" in which we first compute the High-Pass Filter using the Low-Pass Filter. The low-pass filter used in this code is:

$$H0 = [0.48296 \quad 0.83652 \quad 0.22414 \quad -0.12941]$$

And the rest of the code will run in a loop which depends on the number of J-Levels which we will define in the starting. The Filter will be applied for Horizontal part as well as for Vertical part so in my code, First I applied the filter for horizontal part and then for vertical part.

```
%% Filtering the horizontal part
for i = 1:size(I,1)       % For Rows
  % Low-pass filter
  % Downsampling
  % High-pass filter
  % Downsampling
end
```

And the same has been done for the vertical part, down is the code for the vertical part for your reference.

```
%% Filtering the vertical part
for j = 1:size(I,2)    % For Columns
  % Low-pass Filter
  % Downsampling
  % High-pass Filter
  % Downsampling
end
```

As soon as we get the coefficients, we will start making an array of coefficients. The number of coefficients will depend on the J-Level we will give. The new coefficients will be concatenated to the previous ones.

```
Wavelet_Coefficients(1:Image_rows, 1:Image_columns) = Vertical_Part;
```

The schematic of the Decomposition can be seen in figure 2. Where you will see the 4 wavelets out of which one will represent the scaling, second will represent the horizontal elements, third will represent the vertical elements and the last one will represent the diagonal elements of the image or signal as shown in figure 4.

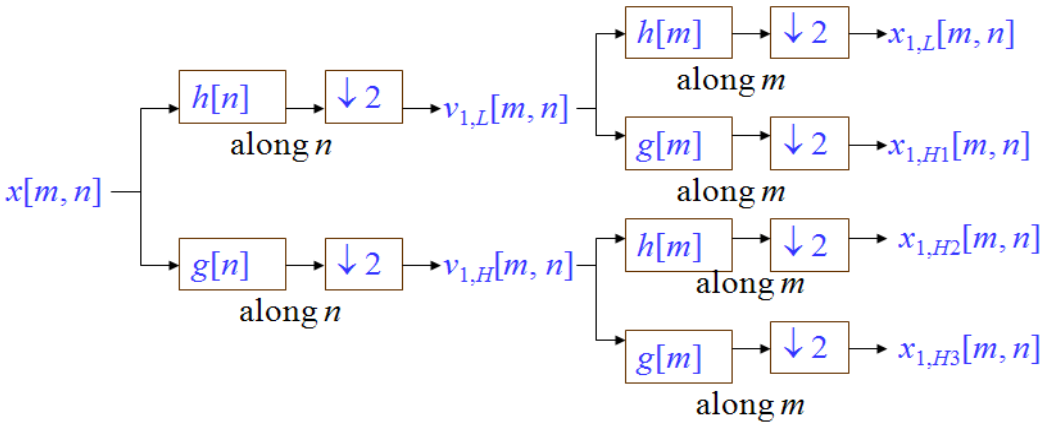

Figure 2: Schematic diagram of 2D wavelet transform

Figure 3: Original Lena Image

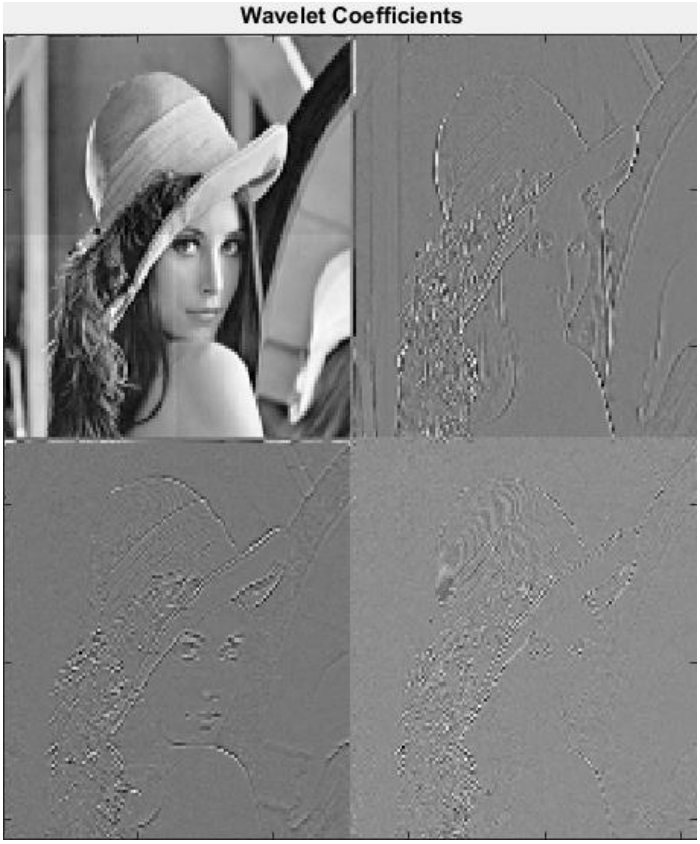

Figure 4: Wavelet Coefficients of Lena

3.2 Low-Pass Filter

The most basic of filtering operations is called "low-pass". A low-pass filter, also called a "blurring" or "smoothing" filter, averages out rapid changes in intensity. The simplest low-pass filter just calculates the average of a pixel and all of its eight immediate neighbors. The result replaces the original value of the pixel. The process is repeated for every pixel in the image.

This low-pass filtered image looks a lot blurrier. But why would you want a blurrier image? Often images can be noisy – no matter how good the camera is, it always adds an amount of “snow” into the image. The statistical nature of light itself also contributes noise into the image.

In our code we are using the Low-Pass Filter given by the professor which is:

$$H0 = [0.48296 \quad 0.83652 \quad 0.22414 \quad -0.12941]$$

```
% Low-pass filtering in rows
Image_Row_LPF = pconv(H0,Image_Row);
% Low-pass Filtering in columns
Image_Columns_LPF = pconv(H0,Image_Columns')';
```

Example of High Pass Filter is shown below in figure 5:

Figure 5: Before and After applying LPF to an Image

3.3 High-Pass Filter

High-pass filter is used in digital image processing to perform image modifications, enhancements, noise reduction, etc., using designs done in either the spatial domain or the frequency domain. The unsharp masking, or sharpening, operation used in image editing software is a high-boost filter, a generalization of high-pass. A high-pass filter, if the imaging software does not provide one, can be achieved by duplicating the image, applying a Gaussian blur to the dupe, inverting the colors, and blending the images using a low opacity for the dupe.

In our code we are using the Low-Pass Filter to compute the high-pass filter. Example of High Pass Filter is shown below in figure 6:

```
% High-pass filtering in rows
Image_Row_HPF = pconv(G0,Image_Row);
% High-pass Filtering in columns
Image_Columns_HPF = pconv(G0,Image_Columns')';
```

Figure 6: Left half of the Image is normal and right half has applied HPF

3.4 Down sampling

Down sampling is a process where the size of the image is compressed or reduced from its original size. For example, suppose you have an image with dimensions AxB, and you want to shrink it to the dimensions of CxD, assuming that A>C and B>D. The most straight forward way to do this is to discard entire columns/rows of data. There should be some sort of pattern and reason to this. One naive way is to simply discard pixels at the edges of the image. For the sample image shown below, we delete the first (A-C) columns, and the first (B-D) rows. The result is less than desirable. We've effectively cropped out a large portion of the original image. We have used a similar technique in our case it is shown below:

```
% Downsampling for Horizontal Part for LPF
Horizontal_Part(i,1:Image_columns/2)=Image_Row_LPF(1:2:length(Image_Row_LPF));
% Downsampling for Horizontal Part for HPF
Horizontal_Part(i,Image_columns/2+1:Image_columns)=Image_Row_HPF(1:2:length(Image_Row_HPF));
% Downsampling for Vertical Part for LPF
Vertical_Part(1:Image_rows/2,j) = Image_Columns_LPF(1:2:length(Image_Columns_LPF));
% Downsampling for Vertical Part  for HPF
Vertical_Part(Image_rows/2+1:Image_rows,j)=Image_Columns_HPF(1:2:length(Image_Columns_HPF));
```

3.5 Results

The results after Wavelet Decomposition can be seen in figure 7. In figure 7, it is clearly shown that which wavelet of the output corresponds to which part of the image.

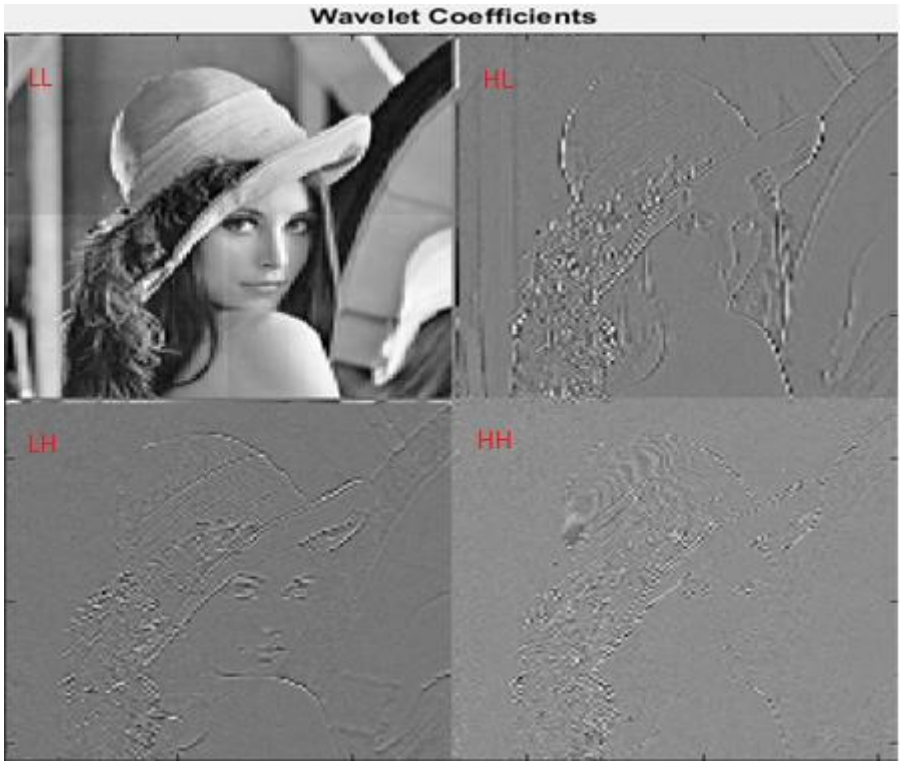

Figure 7: Lena image after wavelet decomposition

Where the different parts of the output wavelets shows:

1. LL : Details or the scale.
2. HL : Variation along X-axis.
3. LH : Variation along Y-axis.
4. HH : Variation along the Diagonals.

I also used another image for decomposition and the results are shown below:

Figure 7: Wavelet decomposition to 1st level

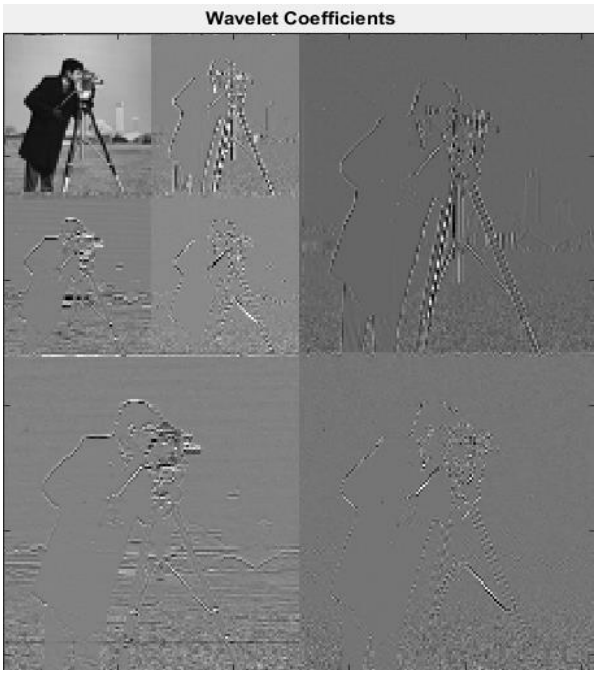

Figure 8: Wavelet decomposition to 2nd level

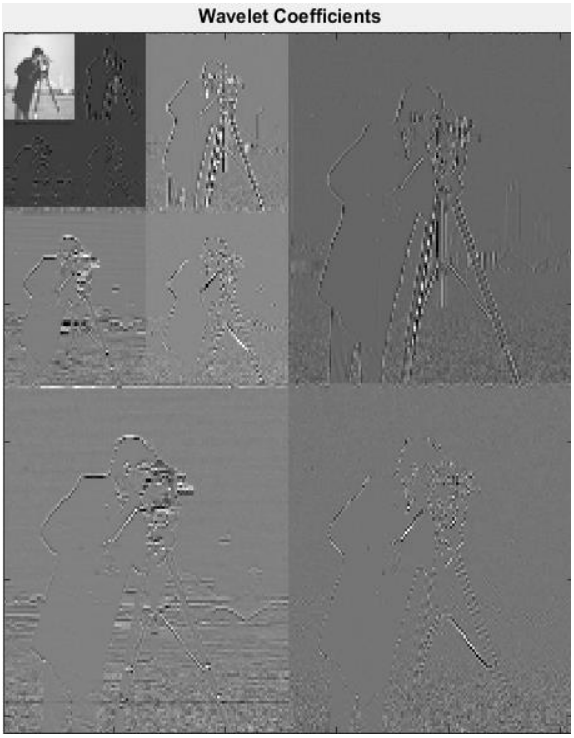

Figure 9: Wavelet decomposition to 3rd level

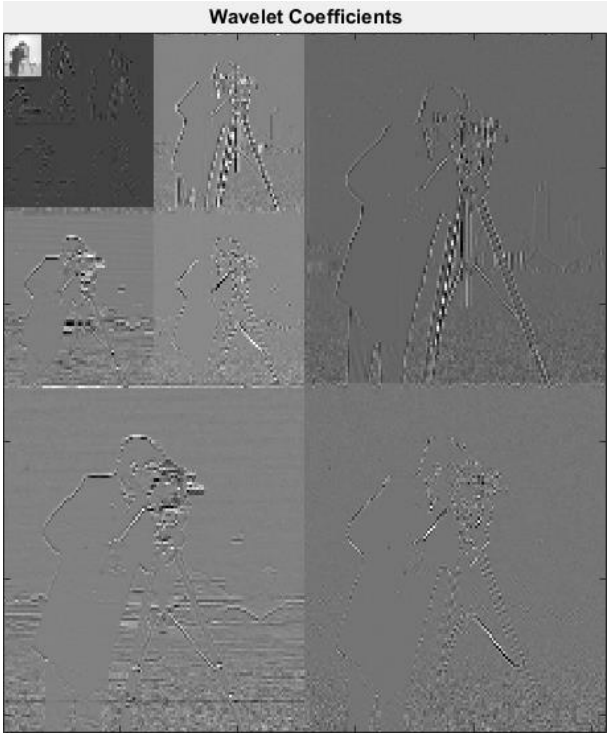

Figure 10: Wavelet decomposition to 4th level

4. Reconstruction

As explained in the decomposition part that there are two main steps in Wavelet Transformation which are:

1. Decomposition
2. Reconstruction

So, the reconstruction part also consists of two Filters:

1. High Pass Filter – G0
2. Low Pass Filter – H0

The Reconstruction of a signal is completely the inverse of the Decomposition of a signal as it uses the wavelet coefficients and recreate the original image. We do up-sampling and then apply high-pass filter (HPF) and low-pass filter (LPF) to obtain the original image. The process will be repeated till the J-Level defined while decomposition.

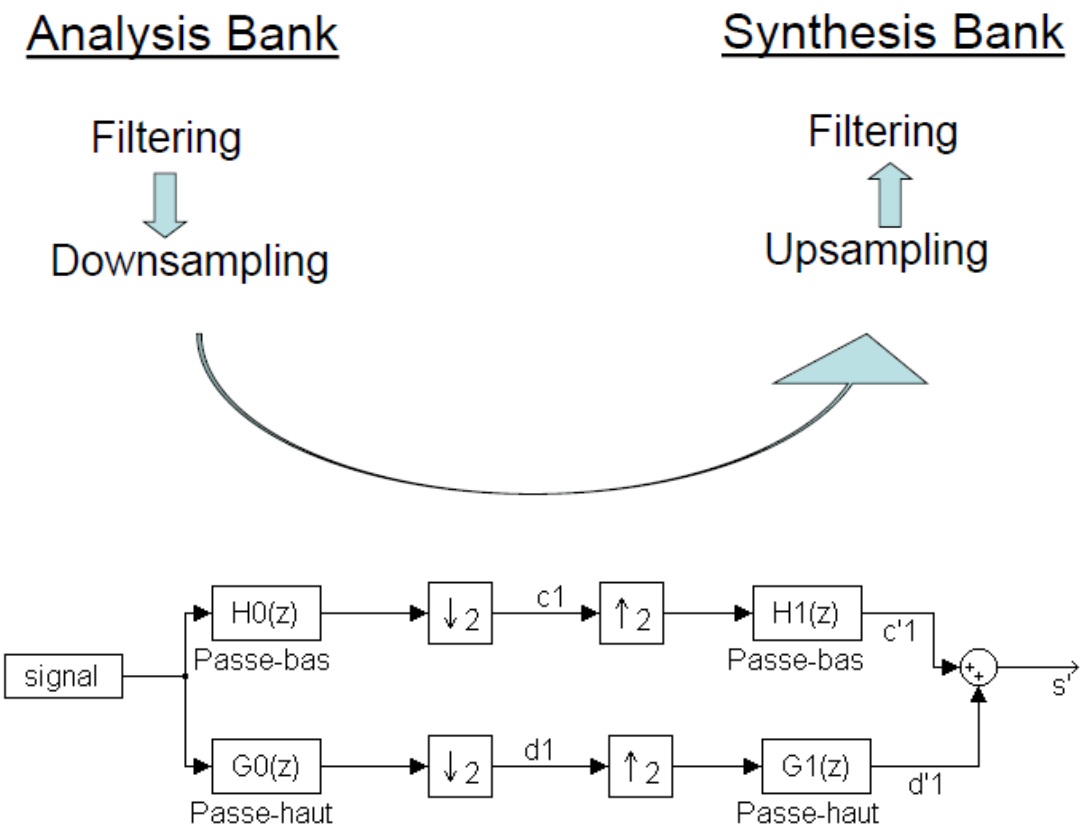

Figure 11: Schematic of the Reconstruction process

4.1 Matlab Function

I made a Matlab function for reconstruction "`Reconstruct_Using_Wavelet_Coefficient`". Which will take the wavelet coefficients as an Input and will return the

reconstructed image. I first compute the High-Pass Filter using the Low-Pass Filter. The low-pass filter used in this code is:

$$H0 = [0.48296 \quad 0.83652 \quad 0.22414 \quad -0.12941]$$

And the rest of the code will run in a loop which depends on the number of J-Levels which we will define in the Decomposition part. The Filter will be applied for Vertical part as well as for Horizontal part so in my code, First I applied the filter for Vertical part and then for Horizontal part.

```
%% Reconstruction of the Vertical part
   for j = 1:Image_columns
      % Upsampling
      % Inverse Low-pass Filtering
      % Upsampling
      % Inverse High-pass Filtering
      % Reconstruct Vertical part
   end
```

And the same has been done for the horizontal part, down is the code for the horizontal part for your reference.

```
%% Reconstruction of the Horizontal part
   for i = 1:Image_rows
      % Upsampling
      % Inverse Lowpass filtering
      % Upsampling
      % Inverse Highpass filtering
      % Reconstruct Horizontal part
   end
```

As soon as we get the reconstructed horizontal and vertical part, we will start

```
Wavelet_Coefficients(1:Image_rows, 1:Image_columns) = Horizontal;
```

And in the end of the loop when the horizonatal part will be completely reconstructed,

```
Reconstructed_Image = Wavelet_Coefficients;
```

So, we obtain the original Image from wavelet coefficients.

4.2 HPF/LPF

The HPF and the LPF used in reconstruction is the same as used in the Decomposition. Still we will have a short description of the filters used during reconstruction process:

```
% In Vertical Section
  % Inverse Low-pass Filtering
  Wave_Coeff_Column_LPF = pconv(H0,fliplr(Wave_Coeff_Column_LPF'))';
  % fliplr is the reverse process
  % Inverse High-pass Filtering
  Wave_Coeff_Column_HPF = pconv(G0,fliplr(Wave_Coeff_Column_HPF'))';
```

```
% In Horizontal Section
  % Inverse Lowpass filtering
  Wave_Coeff_Row_LPF = pconv(H0,fliplr(Wave_Coeff_Row_LPF));
  % Inverse Highpass filtering
  Wave_Coeff_Row_HPF = pconv(G0,fliplr(Wave_Coeff_Row_HPF));
```

*pconv is a function used for periodic convolution

4.3 Upsampling

Upsampling can refer to the entire process of increasing the sampling rate of a signal, or it can refer to just one step of the process, the other step being interpolation. Complementary to decimation, which decreases sampling rate, it is a specific case of sample rate conversion in a multi-rate digital signal processing system. When upsampling is performed on a sequence of samples of a *signal* or other continuous function, it produces an approximation of the sequence that would have been obtained by sampling the signal at a higher rate (or density, as in the case of a photograph).

In our case, we used upsampling for increasing the size of the image to make it as the same size as of the original image.

```
% Upsampling for columns during LPF
  Wave_Coeff_Column_LPF = zeros(Image_rows, 1); % Build array of zeros
   Wave_Coeff_Column_LPF(1:2:length(Wave_Coeff_Column_LPF)) =
   Wave_Coeff_Column(1:Image_rows/2); % Fill 1/2 with values from
   Coefficients
% Upsampling for columns during HPF
  Wave_Coeff_Column_HPF=zeros(Image_rows, 1);    % Build array of zeros
   Wave_Coeff_Column_HPF(1:2:length(Wave_Coeff_Column_HPF)) =
   Wave_Coeff_Column(Image_rows/2+1:Image_rows);
% Upsampling for Rows during LPF
  Wave_Coeff_Row_LPF = zeros(1, Image_columns);  % Build array of zeros
   Wave_Coeff_Row_LPF(1:2:length(Wave_Coeff_Row_LPF)) = Image_Row(1,
   1:Image_columns/2); % Fill 1/2 with values from Coefficients
% Upsampling for Rows during HPF
  Wave_Coeff_Row_HPF = zeros(1, Image_columns);  % Build array of zeros
  Wave_Coeff_Row_HPF(1:2:length(Wave_Coeff_Row_HPF)) = Image_Row(1,
   Image_columns/2+1:Image_columns);
```

4.4 Results

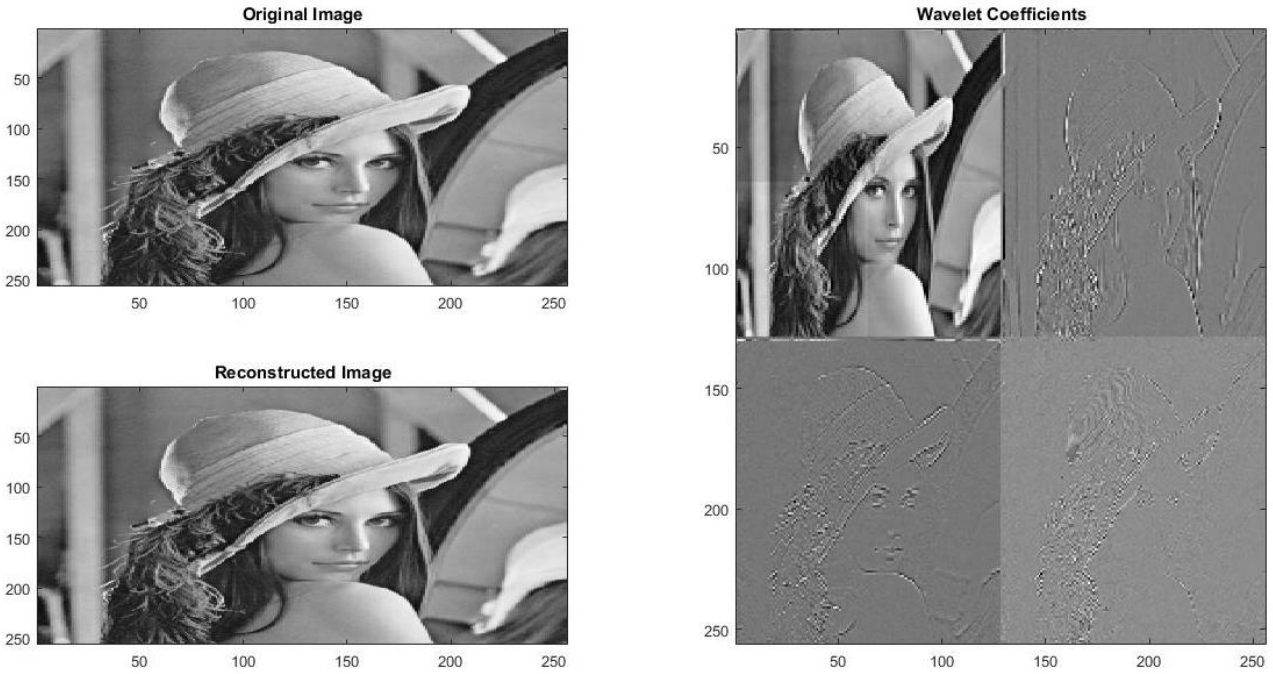

Figure 12: Reconstruction of Image from 1st level

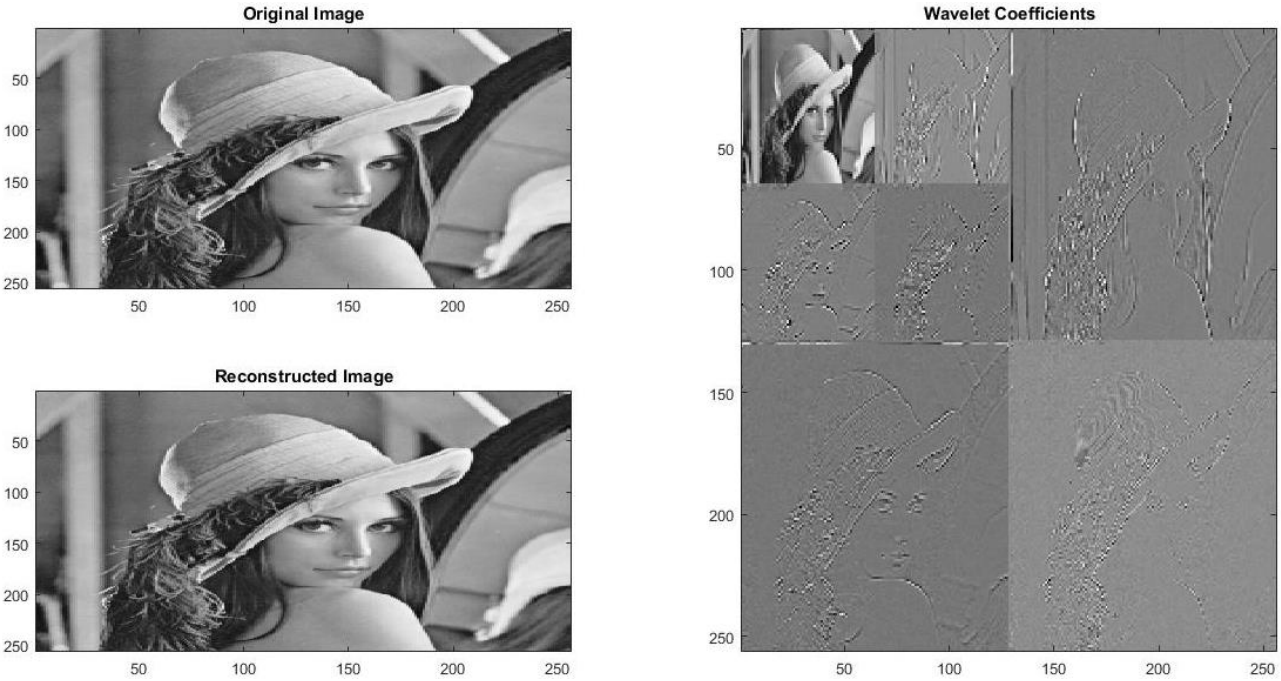

Figure 13: Reconstruction of Image from 2nd level

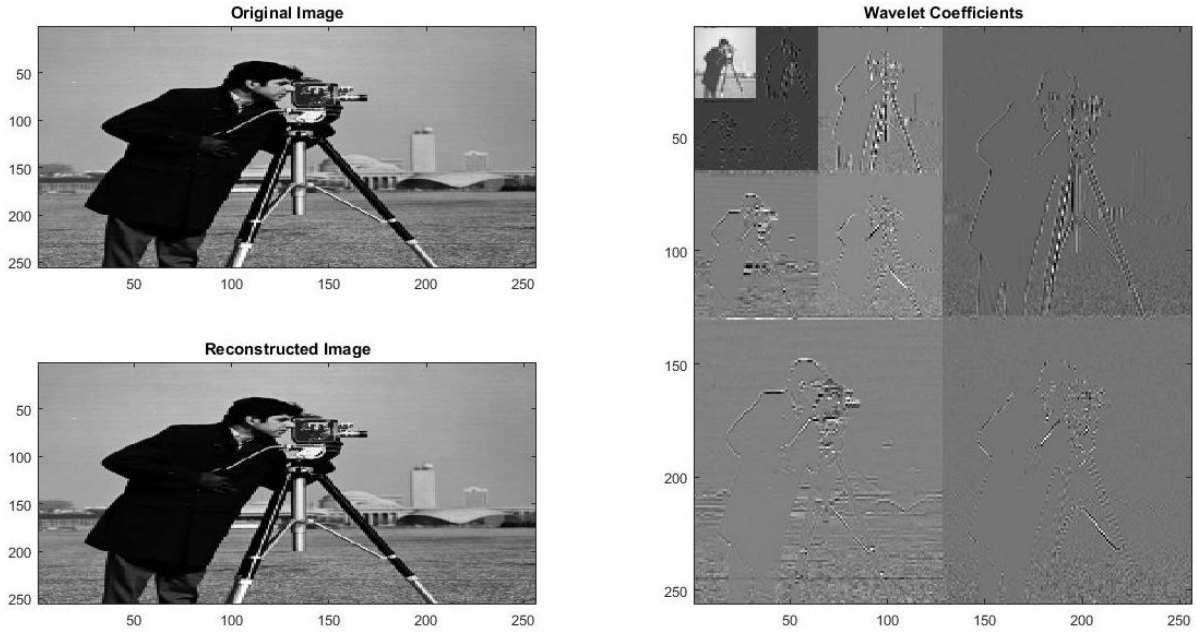

Figure 14: Reconstruction of Image from 3rd level

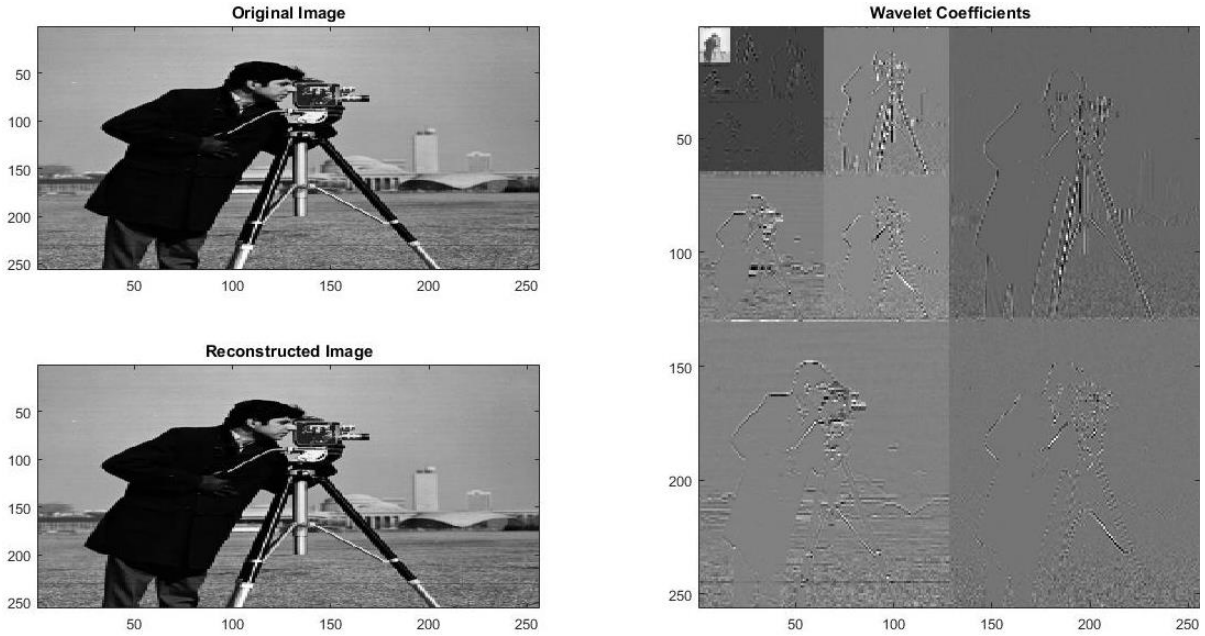

Figure 15: Reconstruction of Image from 4th level

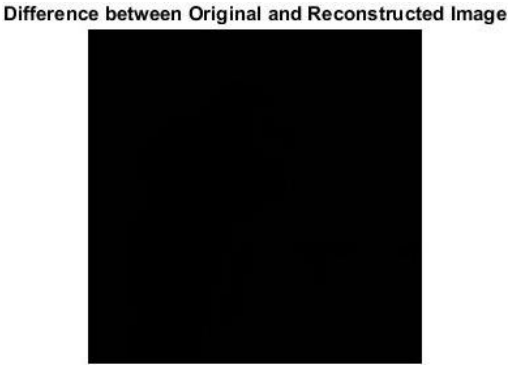

Figure 16: Difference between the reconstructed image and the original image

5. Denoising

Wavelet transforms enable us to represent signals with a high degree of sparsity. This is the principle behind a non-linear wavelet based signal estimation technique known as wavelet denoising. We first added Gaussian White Noise to the Image;

```
Noisy_Image = I+randn(size(I,1),size(I,1))*15;
*Where I is the Image
```

5.1 Decomposition / Reconstruction

We will use the same decomposition function as used previously, from which we will get the wavelets and then we will estimate the noise level from the coefficients of 1st scale. We will apply two types of thresholding on the wavelets:

1. Hard thresholding

```
Wavelet_Coefficients_Noise_2=Wavelet_Coefficients_Noise.*(abs(Wavelet
_Coefficients_Noise)>=threshold);
```

2. Soft Thresholding

```
Wavelet_Coefficients_Noise_2=(sign(Wavelet_Coefficients_Noise).*(abs(
Wavelet_Coefficients_Noise)-
threshold)).*((abs(Wavelet_Coefficients_Noise)>=threshold));
```

For Hard thresholding, we used;

$$T^{hard}\left(d_{l,k}\right)=\begin{cases} d_{j,k} \text{ if } \left|d_{l,k}\right| \geq \lambda \\ 0 \text{ if } \left|d_{l,k}\right| < \lambda \end{cases}$$

And foor soft thresholding, we used;

$$T^{soft}\left(d_{j,k}\right)=\begin{cases} sign\left(d_{j,k}\right)\left(\left|d_{j,k}\right|-\lambda\right) \text{if } \left|d_{j,k}\right| \geq \lambda \\ 0 \text{ if } \left|d_{j,k}\right| < \lambda \end{cases}$$

The difference between both hard thresholding and soft thresholding can be seen in the results. We set all coefficients to zero except those whose magnitude is larger than 3s (we can also try factors other than 3), and reconstruct an estimate of the original image by applying the corresponding inverse transform After we apply any of the thresholding to the coefficients we will reconstruct the original image and we will see that the noise level has reduced. As it is clearly seen in the resulting figure 17.

5.2 Results

1. Using Hard Thresholding

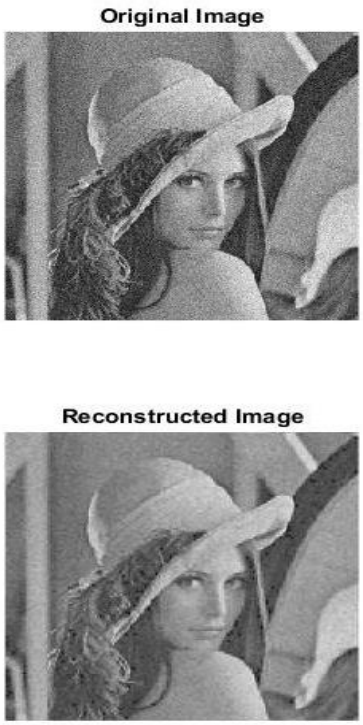

Figure 17: Lena Denoising on 1st level

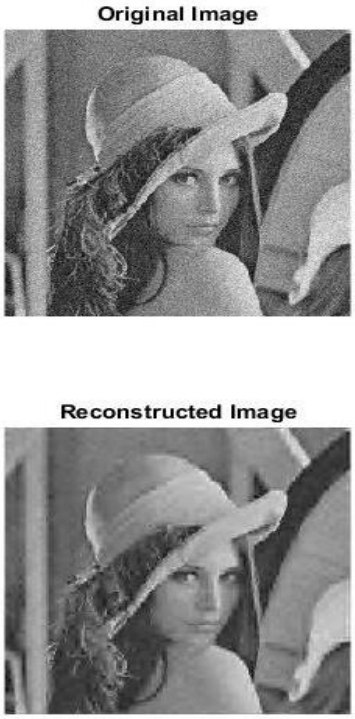

Figure 18: Lena Denoising on 2nd level

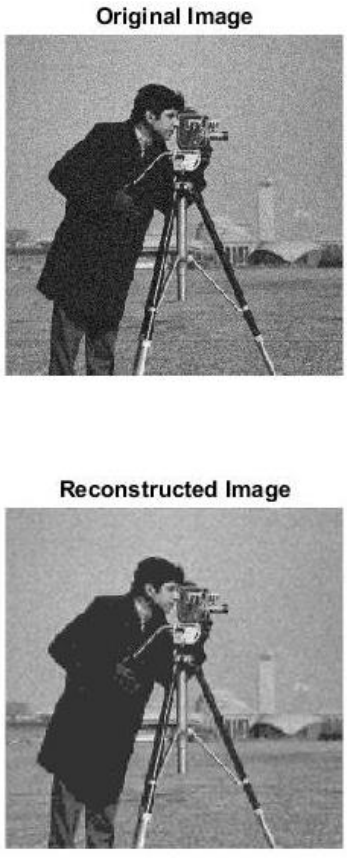

Figure 19: Cameraman Image Denoising on 1st level

Figure 20: Cameraman Image Denoising on 2nd level

2. Using Soft Thresholding

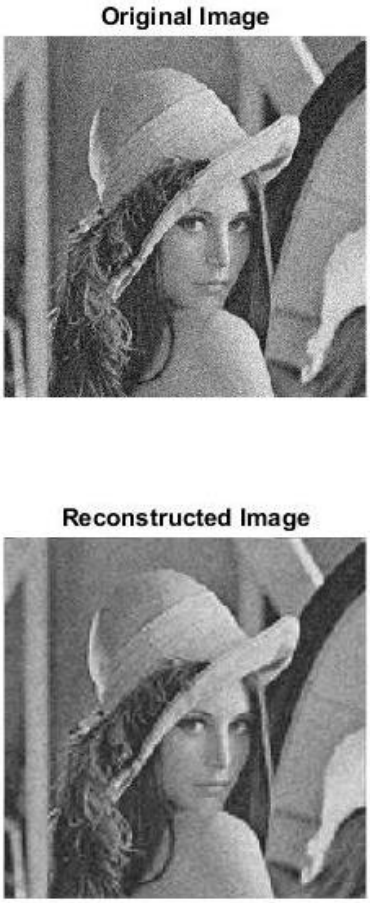

Figure 21: Lena Denoising on 1st level

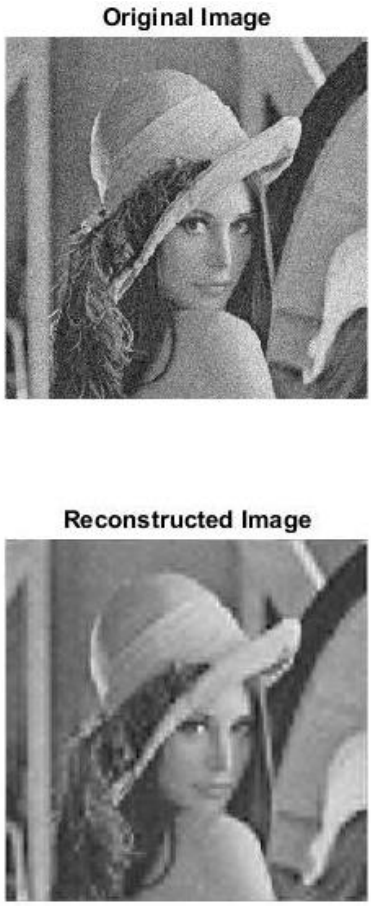

Figure 22: Lena Denoising on 2nd level

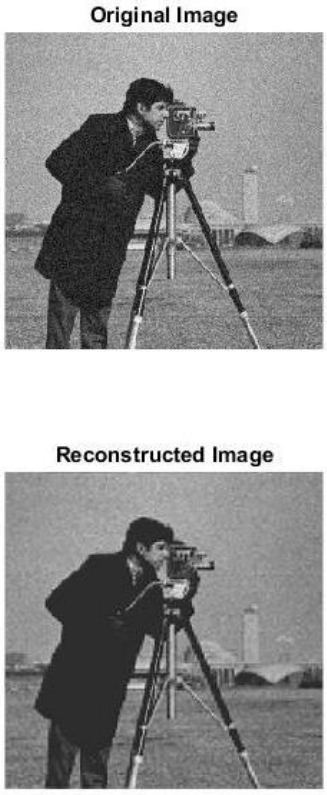

Figure 23: Cameraman Image Denoising on 1st level

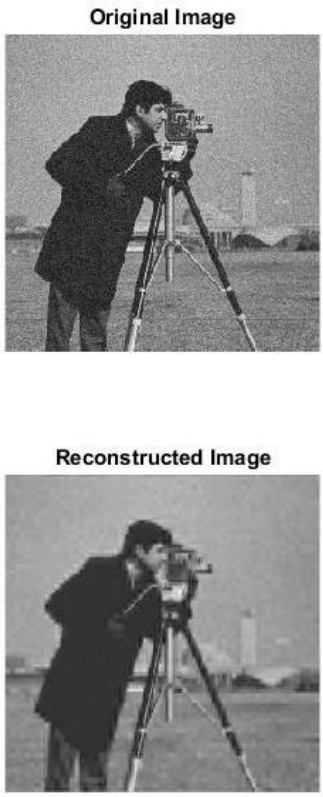

Figure 24: Cameraman Image Denoising on 2nd level

6. Conclusion

From the abstract idea in approximation, multiresolution theory, we generalize the Fourier transform and start the journey of wavelets. The discrete wavelet is more useful in realization. We often use 2D wavelets to do image compression. The continuous wavelet transform analyze the continuous-time signal in a different perspective. By the advantage of multiresolution, we can locate time and frequency more accurately. The wavelet design is more complicated in mathematics but the design procedure completes the existence of the wavelets. The application chapter mentions the nowadays JPEG and JPEG2000 standard.

The wavelets bring us to a new vision of signal processing. It tactically avoids the problem that Fourier analysis encounters. Its implementation is simple. We need some designed filters to do the task. Although the implementation is quite easy, the filter design includes lots of mathematical originality and this is the research topic. Once we establish wavelets with more ideal properties, lots of difficult problems might be solved.