

Digital Image Processing

Computer Homework 2

Due date: 1404/02/07



K. N. Toosi
University of Technology

Application of **Wavelet Transform** to Image Processing

1. Instructions

- a) Submit the assignment as a **Jupyter Notebook** (.ipynb) containing:
 - i. Executed results of the code (no empty cells).
 - ii. Explanations in Markdown cells for each step of the assignment.
- b) Important Notes:
 - i. Attach all necessary files (including sample images) to ensure the notebook runs directly.
 - ii. Keep each part of the code in separate cells for clarity.
 - iii. Add proper documentation and comments in both Markdown and code cells for a better score.
 - iv. Both Python and MATLAB can be submitted, but Python submissions will be awarded 10% more points.
 - v. Only functions for **1D processing** can be used. Thus, it is **not allowed** to use functions for 2D analysis like `dwt2/idwt2` as part of the exercise that you implement them yourself.

2. Preliminaries

What is a Wavelet?

Wavelets are mathematical functions that are localized in both time and frequency, meaning they focus on a specific point rather than spreading across the entire signal. This makes them useful for overcoming the limitations of the Fourier Transform, which provides frequency information but lacks precise temporal details. According to Heisenberg's Uncertainty Principle, there is a trade-off between frequency and time resolution—high frequency resolution results in poor time resolution and vice versa. The Discrete Wavelet Transform (DWT) is a type of wavelet transform where wavelets are sampled at discrete intervals. Unlike the Fourier Transform, DWT offers better temporal resolution, allowing it to capture both frequency and time-based location information simultaneously.

One-Level Discrete Wavelet Transform (DWT)

The DWT of a signal $x[n]$ is computed by passing it through a series of filters. First, the signal is convolved with a **low-pass filter** $g[n]$, which captures the smooth (approximate) components:

$$y_{\text{low}}[n] = (x * g)[n] = \sum_{k=-\infty}^{\infty} x[k]g[n - k]$$

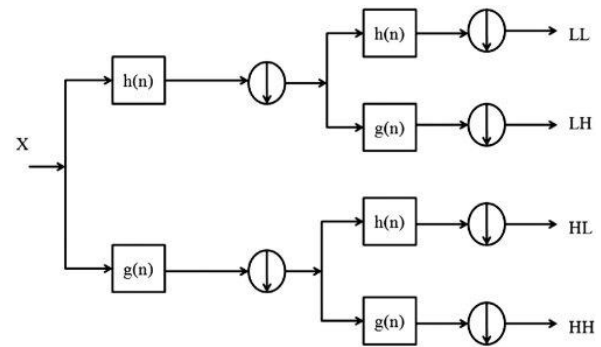
The signal is also decomposed simultaneously using a **high-pass filter** $h[n]$. The outputs give the detail coefficients (from the high-pass filter):

$$y_{\text{high}}[n] = (x * h)[n] = \sum_{k=-\infty}^{\infty} x[k]h[n - k]$$

For 2D images, the filtering process is applied separately in horizontal and vertical directions, resulting in four sub-bands:

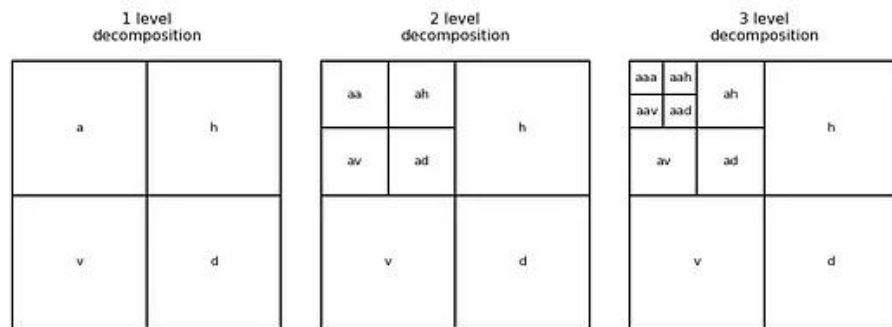
- ◆ LL (Low-Low) → Approximation (smooth regions)
- ◆ LH (Low-High) → Horizontal edges
- ◆ HL (High-Low) → Vertical edges
- ◆ HH (High-High) → Diagonal edges

[1]



This process can be repeated on the **LL sub-band** for multi-level wavelet decomposition, breaking the image into finer and coarser details.

Result will be like this:

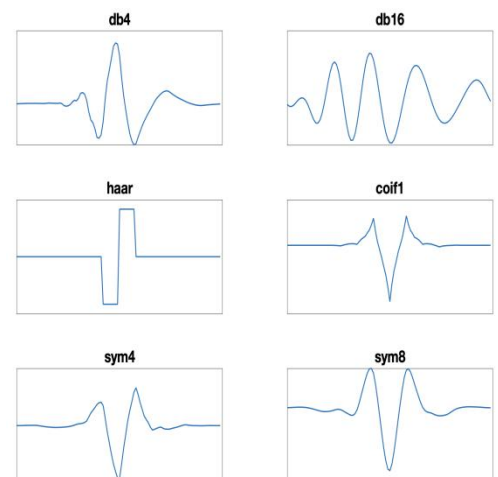


[2]

Different Wavelet Filters

1. **Haar Wavelet:** The simplest wavelet, often used for fast and basic analysis. It acts like a step function, making it ideal for detecting edges.
2. **Daubechies Wavelets:** More advanced, smoother wavelets that preserve signal characteristics better, commonly used in image compression.
3. **Coiflet Wavelets:** Designed for better symmetry and high vanishing moments, useful in denoising applications.
4. **Symlet Wavelets:** A symmetric version of Daubechies wavelets with improved reconstruction properties.

[3]



Before starting the assignment, install and import the required libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import pywt
```

3. Tasks & Implementation

Task 1: 2D Wavelet Sub band Coding

- Step 1: Select a test images and display them.

1. Load image ([Khayyam_Mausoleum.jpg](#), [milad-tower.jpg](#))
2. Convert it to grayscale (if not already).



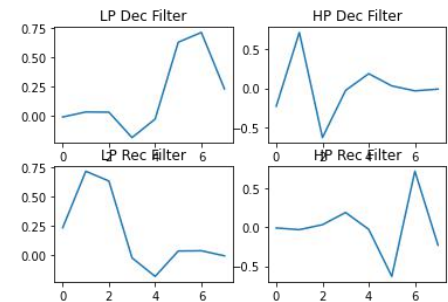
[4]



[3]

- Step 2: Select and plot filter.

Plot the low/high **decomposition** and **reconstruction** filters for Daubechies 4-tap (db4) wavelet transform. You can obtain the Daubechies coefficients using the function **wfilters** in MATLAB or **pywt.families** in PYTHON. This is the wavelet transform you should use for all the remaining tasks.

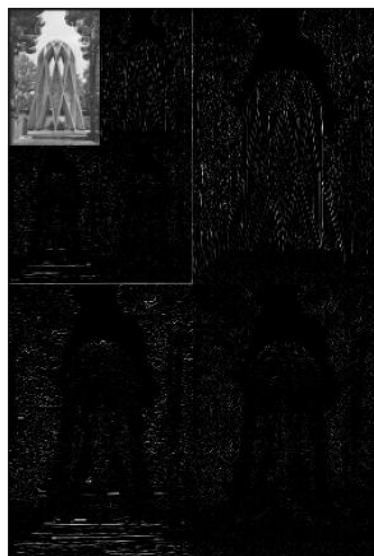


Code Hint	
To presents all filters	<pre>for family in pywt.families(): print("Family: " + family + ", " .join(pywt.wavelist(family)))</pre>
To select the db4 filter	<pre>wavelet = pywt.Wavelet('db4') dec_lo, dec_hi = np.array(wavelet.dec_lo), np.array(wavelet.dec_hi) rec_lo, rec_hi = np.array(wavelet.rec_lo), np.array(wavelet.rec_hi)</pre>

- Step 3: Implement a 2D wavelet and inverse 2D wavelet transforms using 1D wavelet transforms. It should have the a function and following interface:

dec_image = wvt2D (input_image, dec_lowpass_filter, dec_highpass_filter, level)
rec_image = iwvt2D (dec_image, rec_lowpass_filter, rec_highpass_filter, level)

2-Level Decommission

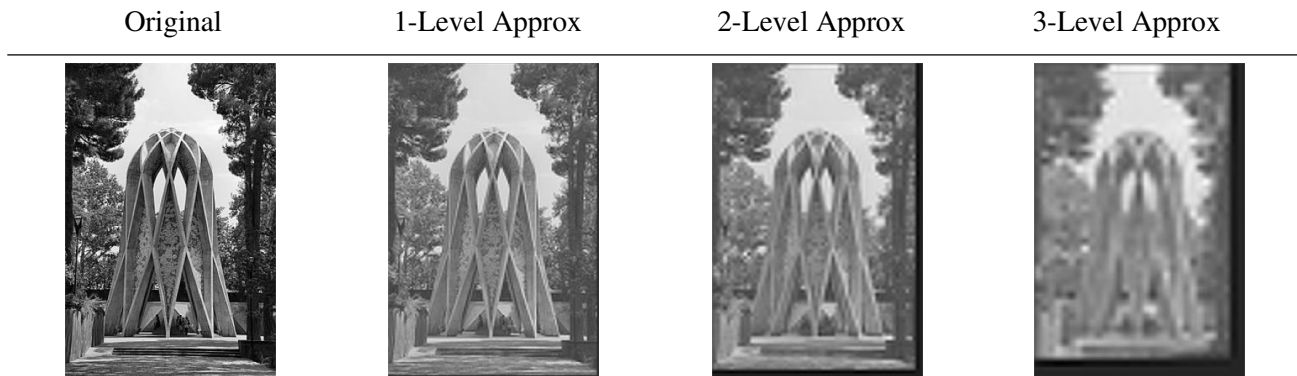


2-Level Re-Construction



Notice : To show better the detail coefficients you may add a DC value (e.g. 128) to each coefficient. To test, run a 3 level decomposition and reconstruction of the selected test image. Verify how well is the image reconstructed by computing the MAE and SSIM between the original and the reconstructed image.

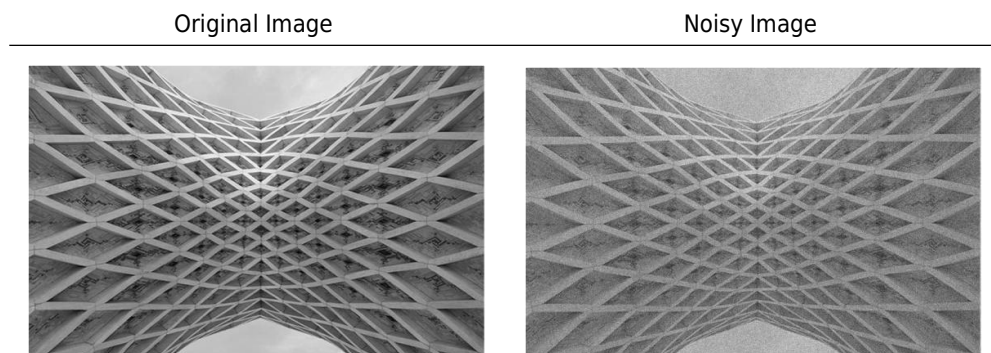
- Step 4: Decompose and reconstruct the image using the approximation coefficients only. Perform this for 3 progressively larger levels. Visualize the original image and the three reconstructed images, side by side.



- Step 5: Repeat the steps for Haar and Sym4 and compare MSE, SSIM value (10% extra point)

Task 2: 2D Wavelet Denoising

- Step 1: Add Gaussian noise ($\mu = 0$, $\sigma = 10$) to the original test image ([Azadi_Tower.jpg](#) [5]). Visualize both images (the original and its noisy version) side by side. Compute peak signal to noise ratio (PSNR) between the two images.


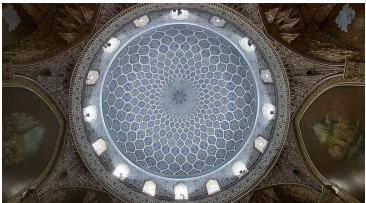


- Step 2: Answer Following Questions:
 1. Where to Apply the Threshold (LL , LH, HL, HH) ?
 2. How much is the **Optimal Level** of Decomposition?
- Step 2: Perform Wavelet decomposition and thresholding the detail coefficients in the following way:
 1. Select a threshold **VisuShrink** ($T = \sigma\sqrt{2\log(n)}$), where n is the number of the detail coefficients (total number of pixels in the image (Width \times Height) and σ is an estimate of the noise level.
 2. How to Estimate σ ?
 3. Set all detail coefficients with their absolute value $< T$ to zero.

4. Reconstruct the image.

- Step 3: Compute the PSNR for different decomposition levels and σ values, i.e., for decomposition levels {1,2,3} and values of $\sigma = \{1,10,100\}$.

Task 3: Wavelet-based edge detection

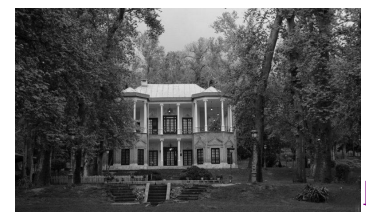
- Step 1 : Load and display the [Saadabad_Palace.jpg](#) and [Marmar_Palace.jpg](#) images and **convert to grayscale**.  [6]
- Step 2: Perform 1-level, 2-level, and 3-level wavelet decomposition for both images and display the resulting coefficients utilizing the 'db4' wavelet.  [7]
- Step 3: Reconstruct both images firstly using LH, HL, and HH components, and secondly using only LH and HL components (Hint: you can zero the LL (and HH) component(s) to eliminate their effect). Perform the reconstructions using 1, 2, and 3-level decomposition components, respectively. (Note: There will be 6 series of results to illustrate)
- Step 4: Explain how this approach could lead to edge detection. What is the difference in reconstructed images with and without eliminating the HH component?
- Step 5: Argue for which image this wavelet-based edge detection algorithm performs better and write the possible reasons. Specifically, in the case of the "[Saadabad_Palace.jpg](#)" image, which edges are detected better (The windows or the columns itself)? Explain why.
- Step 6: Repeat the parts 'b' and 'c' with different Daubechies wavelets (for example: 'db3', 'db2', and 'db1'), and interpret the results.

Task 4 (+10 points): Wavelet-Based Image Compression

Image compression is essential for reducing **storage space** and **transmission time** while maintaining acceptable image quality. One powerful technique is Wavelet-Based Compression, which decomposes an image into different frequency components and selectively removes less significant details. In this assignment, you'll explore how wavelet transform can be applied to grayscale images to reduce file size while preserving essential image features.

Let's break down the steps:

1. Load the grayscale image [Niavaran_Palace.jpg](#).
2. Apply 2D Discrete Wavelet Transform (DWT) to break the image into low-frequency (important details) and high-frequency (fine details, noise) component. (Filters : db1, db4, haar), (Decompose Level: 2, 3, 4)
3. Sort the wavelet coefficients based on magnitude
4. Threshold the coefficients, keeping only the most important ones and setting others to zero.
5. Reconstruct the image using the remaining coefficients.
6. Compare original vs. compressed image quality (SSIM) and file size .



Code Hint	
To take the compression parameter	<pre># Parameters for wavelet compression n = 4 # Number of decomposition levels w = 'db1' # Wavelet type keep = 1 # Fraction of coefficients to keep</pre>
To sort and threshold	<pre># Sort coefficients by magnitude Csort = np.sort(np.abs(coeff_arr.reshape(-1))) # Compute threshold to keep only a fraction of coefficients thresh = Csort[int(np.floor((1 - keep) * len(Csort)))] # Apply thresholding (keep only large coefficients) ind = np.abs(coeff_arr) > thresh Cfilt = coeff_arr * ind # Set small coefficients to zero</pre>

Task 5 (+20 points): Instagram's Progressive Image Loading

Have you ever noticed how images load on Instagram when your internet is slow? Instead of waiting for the full high-quality image to appear, Instagram first shows a blurry, low-quality version and then gradually improves the resolution. This method ensures that users can see something immediately rather than waiting for the full image to load.

This idea is similar to **wavelet-based image compression**, where we can reconstruct an image step by step, keeping only the most important details first and refining it later.

1. Load the Color Image [Niavaran_Palace_Color.jpg](#).
2. Separate the image into Red, Green, and Blue (R, G, B) channels.
3. Perform Discrete Wavelet Transform (DWT) on each channel independently.
4. Use sorting and indexing to keep a percentage of the largest coefficients.
5. Reconstruct Each Channel
6. Combine the R, G, and B channels to obtain the final compressed image.
7. Save and Compare the Results.
8. Show the result of each compression rate side by side/



[8]

Download Step 1



Download Step 2





Best of Luck

For further information, do not hesitate to contact me at a.najafi@email.kntu.ac.ir.

4. Appendix

a) **Mean Squared Error (MSE)** The average squared difference between the value observed in a statistical study and the values predicted from a model. When comparing observations with predicted values, it is necessary to square the differences as some data values will be greater than the prediction (and so their differences will be positive) and others will be less (and so their differences will be negative). Given that observations are as likely to be greater than the predicted values as they are to be less, the differences would add to zero. Squaring these differences eliminates this situation [8].

$$\text{MSE} = \overset{\text{Mean}}{\frac{1}{n}} \sum_{i=1}^n \overset{\text{Error}}{(Y_i - \hat{Y}_i)} \overset{\text{Squared}}{^2}$$

[9]

b) **Structural Similarity Index (SSIM)** Is a perceptual metric that quantifies image quality degradation caused by processing such as data compression or by losses in data transmission. SSIM actually measures the perceptual difference between two similar images. It cannot judge which of the two is better: that must be inferred from knowing which is the “original” and which has been subjected to additional processing such as data compression [10].

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Where :

$$\mu_x = \frac{1}{N} \sum_{i=1}^N x_i, \quad \mu_y = \frac{1}{N} \sum_{i=1}^N y_i$$

$$\sigma_x^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2, \quad \sigma_y^2 = \frac{1}{N-1} \sum_{i=1}^N (y_i - \mu_y)^2$$

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$$

c) **Peak Signal-to-Noise Ratio (PSNR)** is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. Because many signals have a very wide dynamic range, (ratio between the largest and smallest possible values of a changeable quantity) the PSNR is usually expressed in terms of the logarithmic decibel scale.

$$PSNR = 10 \log_{10} \left(\frac{L^2}{MSE} \right)$$

$L = \text{maximum possible pixel value (e.g., 255 for an 8-bit image)}.$