Introduction:

This project aims to implement several algorithms, starting from reading a grayscale image in PGM format to filtering and denoising a noisy image.

In the first phase, an algorithm is developed from scratch to read a PGM file and convert it into a matrix, which will be used as input for subsequent algorithms. Next, a wavelet decomposition algorithm using Haar filtering is designed to extract the coefficients of the image. This is followed by the implementation of the reverse process—wavelet reconstruction—to recreate the original image from its frequency coefficients.

Finally, a denoising algorithm leveraging the developed wavelet decomposition and reconstruction processes is created. This algorithm is applied to reduce noise in the noisy image, and its efficiency is evaluated by comparing the denoised image to the original.

Task 1 :

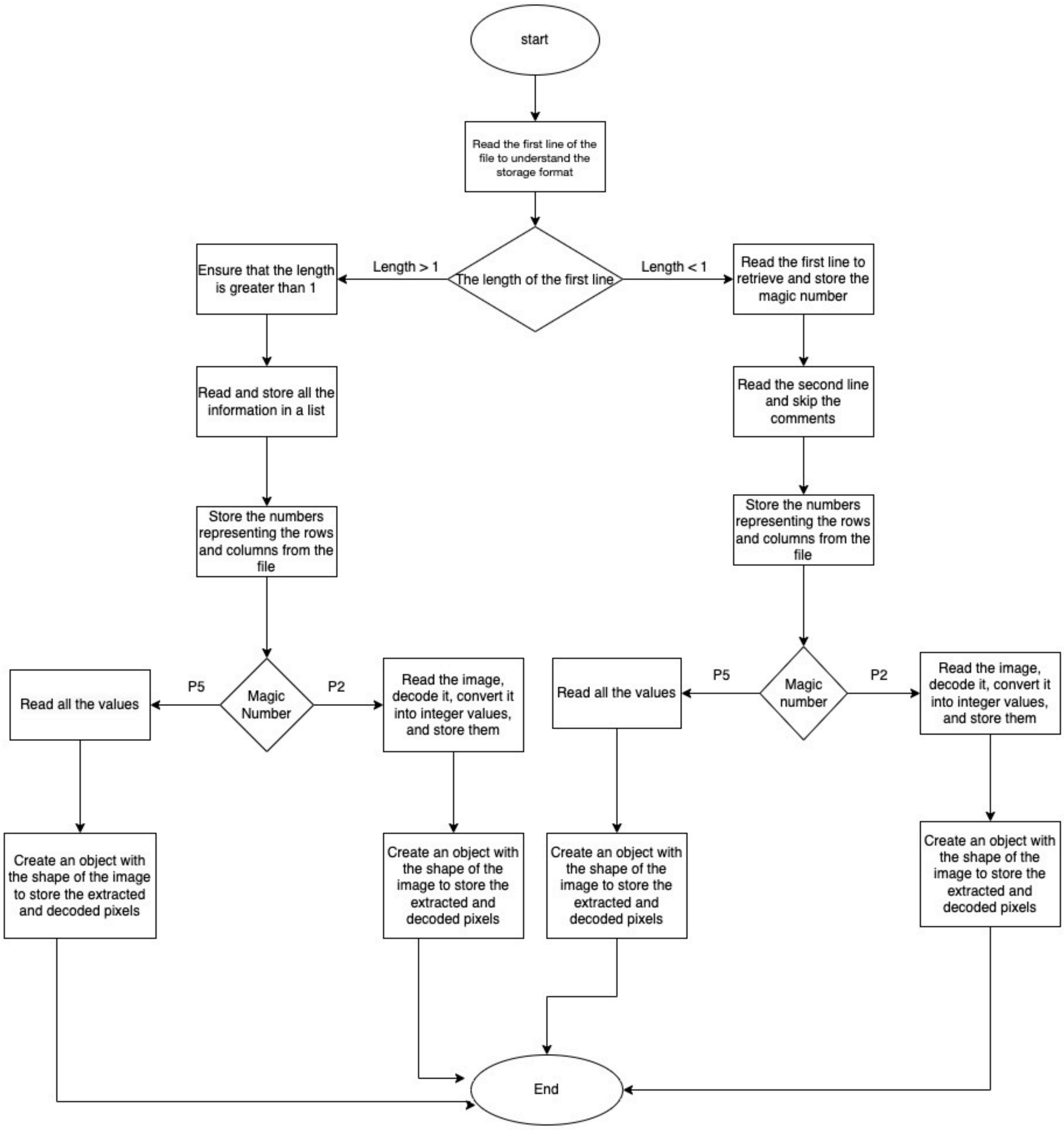
Reading the PGM file:

The dataset contains images in PGM format, commonly used for storing grayscale images. This format includes several lines that provide details about the image. The first line, referred to as the magic number, specifies the file type and can be either 'P2' or 'P5'. 'P2' indicates the image is stored in ASCII format, while 'P5' signifies binary format. The subsequent lines typically include comments, the image dimensions (number of rows and columns), the maximum pixel value, and the actual image data.

To facilitate image processing, an algorithm was developed to extract and interpret this information, converting the image into a NumPy array. The algorithm begins by reading the file using Python's open() function in binary mode, indicated by the 'rb' parameter—ideal for handling PGM files. The readline() function is employed to process the file line by line, and the split() function is used to parse each line into a list, using whitespace as the default delimiter.

At the outset, the algorithm checks the file structure to handle potential deviations from the standard format. For instance, some files might consolidate all the image information into a single line rather than distributing it across multiple lines. The flowchart below outlines the remaining steps of the process.

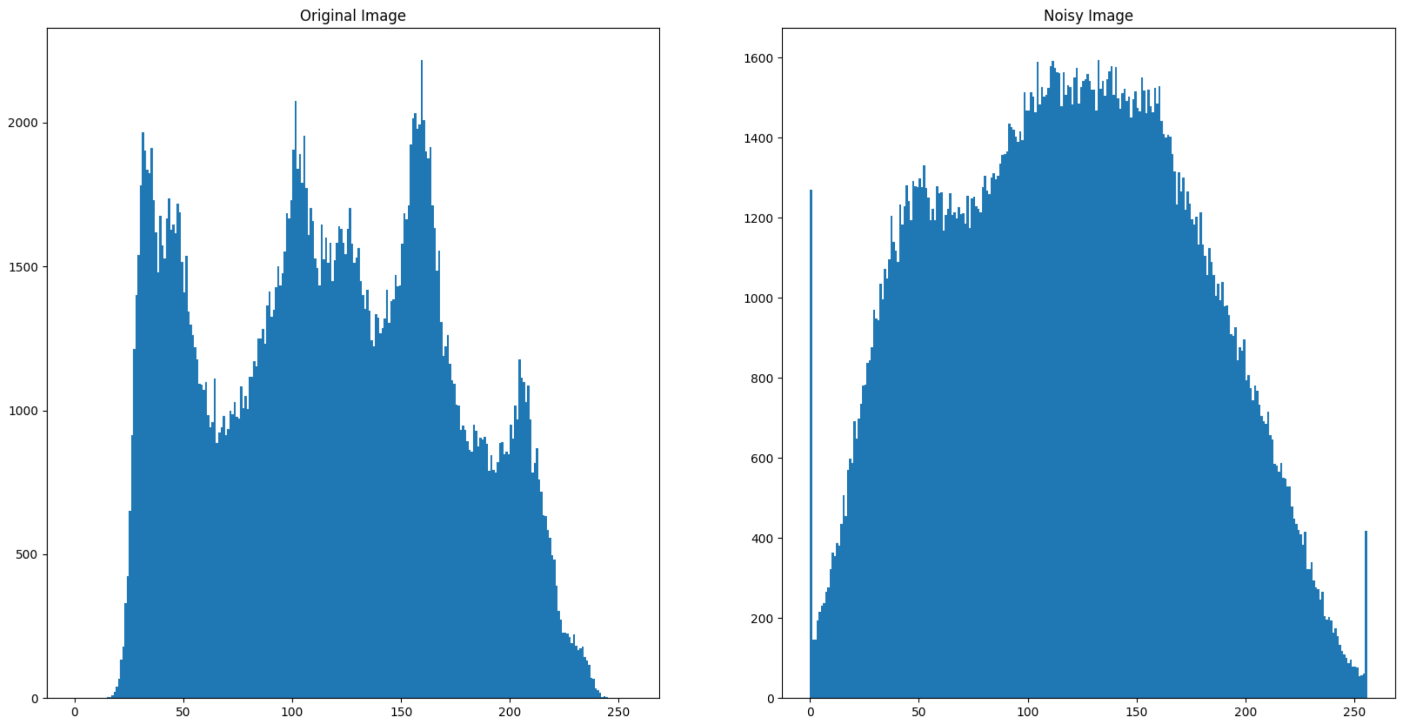
1-The flowchart of the reading PGM file Algorithm


start
Read the first line of the file to understand the storage format
The length of the first line
Length > 1
Length < 1
Ensure that the length is greater than 1
Read and store all the information in a list
Store the numbers representing the rows and columns from the file
Read the first line to retrieve and store the magic number
Read the second line and skip the comments
Store the numbers representing the rows and columns from the file
Magic Number
P5
P2
Read all the values
Read the image, decode it, convert it into integer values, and store them
Magic number
P5
P2
Read all the values
Read the image, decode it, convert it into integer values, and store them
Create an object with the shape of the image to store the extracted and decoded pixels
Create an object with the shape of the image to store the extracted and decoded pixels
Create an object with the shape of the image to store the extracted and decoded pixels
Create an object with the shape of the image to store the extracted and decoded pixels
End

Data Preprocessing:

With the algorithm for reading PGM files successfully implemented, images can now be read and stored in variables. Two variables were designated to hold the original and noisy images separately. During the preprocessing phase, analyzing the histogram of the noisy image can be particularly insightful. The histogram may reveal the presence of noise and even provide clues about its type. For instance, a high frequency of intensity values at 0 and 255 often indicates salt-and-pepper noise.

2-The histograms of original and noisy images



To enable accurate mathematical operations on the images, they must first be converted into floating-point numbers, normalized to values between 0 and 1. This conversion ensures numerical stability and compatibility for subsequent image processing steps.

Task 2:

Wavelet Decomposition

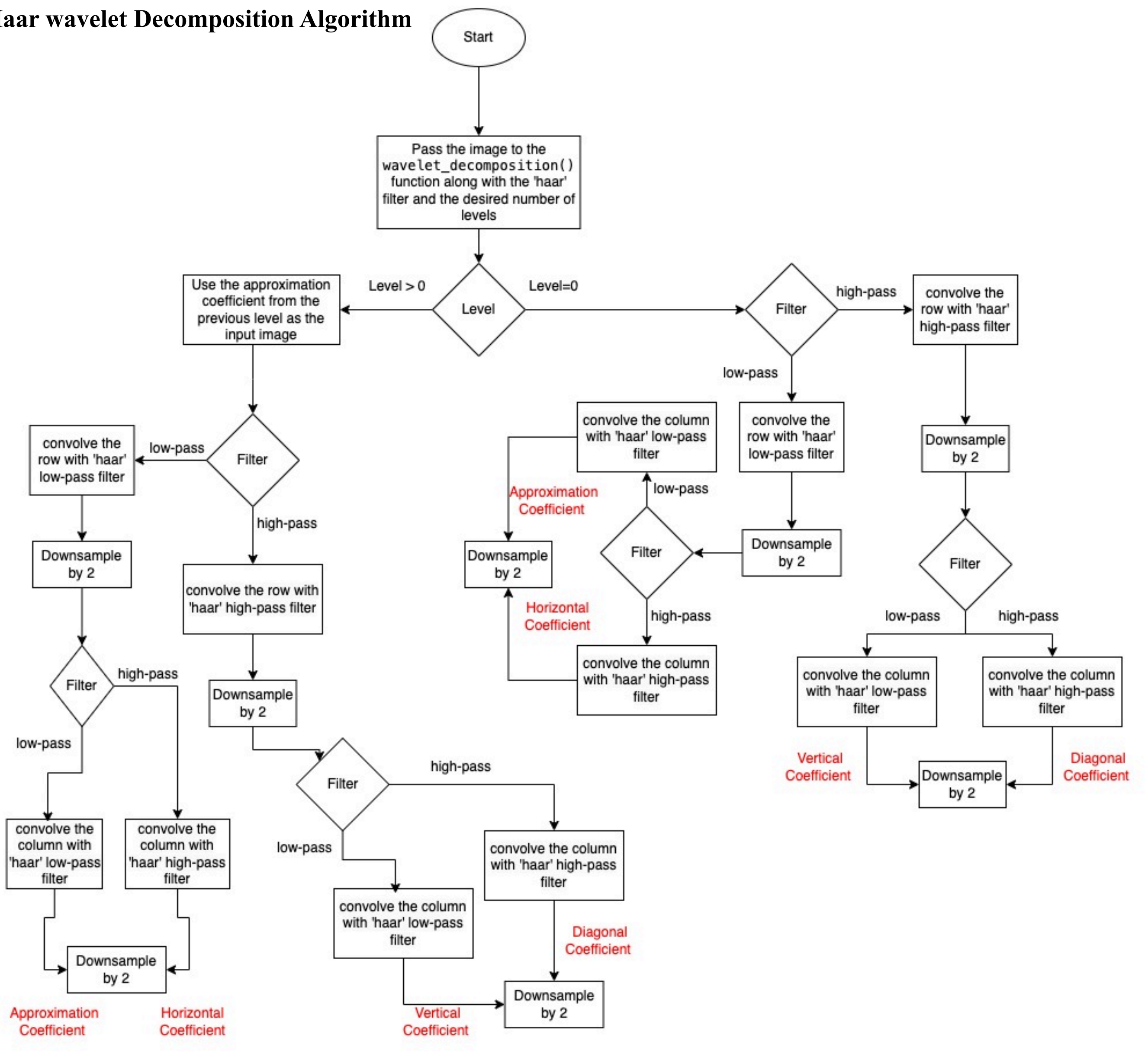
Wavelet decomposition transforms an image into a set of coefficients, effectively separating it into distinct frequency bands across multiple resolutions.To perform Haar wavelet transform, the Haar wavelet formula must first be defined. The Haar wavelet is represented using the following filters:

- **Low-pass filter [1/ √2, 1/ √2] :** This filter averages two adjacent values in the signal, preserving the low-frequency components (approximations).
- **High-pass filter [1/ √2, - 1/ √2] :** This filter calculates the difference between two adjacent values in the signal, extracting the high-frequency components (details).

These filters are applied iteratively through convolution with the image matrix during the wavelet transform, decomposing the image into approximation and detail coefficients at multiple levels.

The flowchart below illustrates the steps of wavelet decomposition

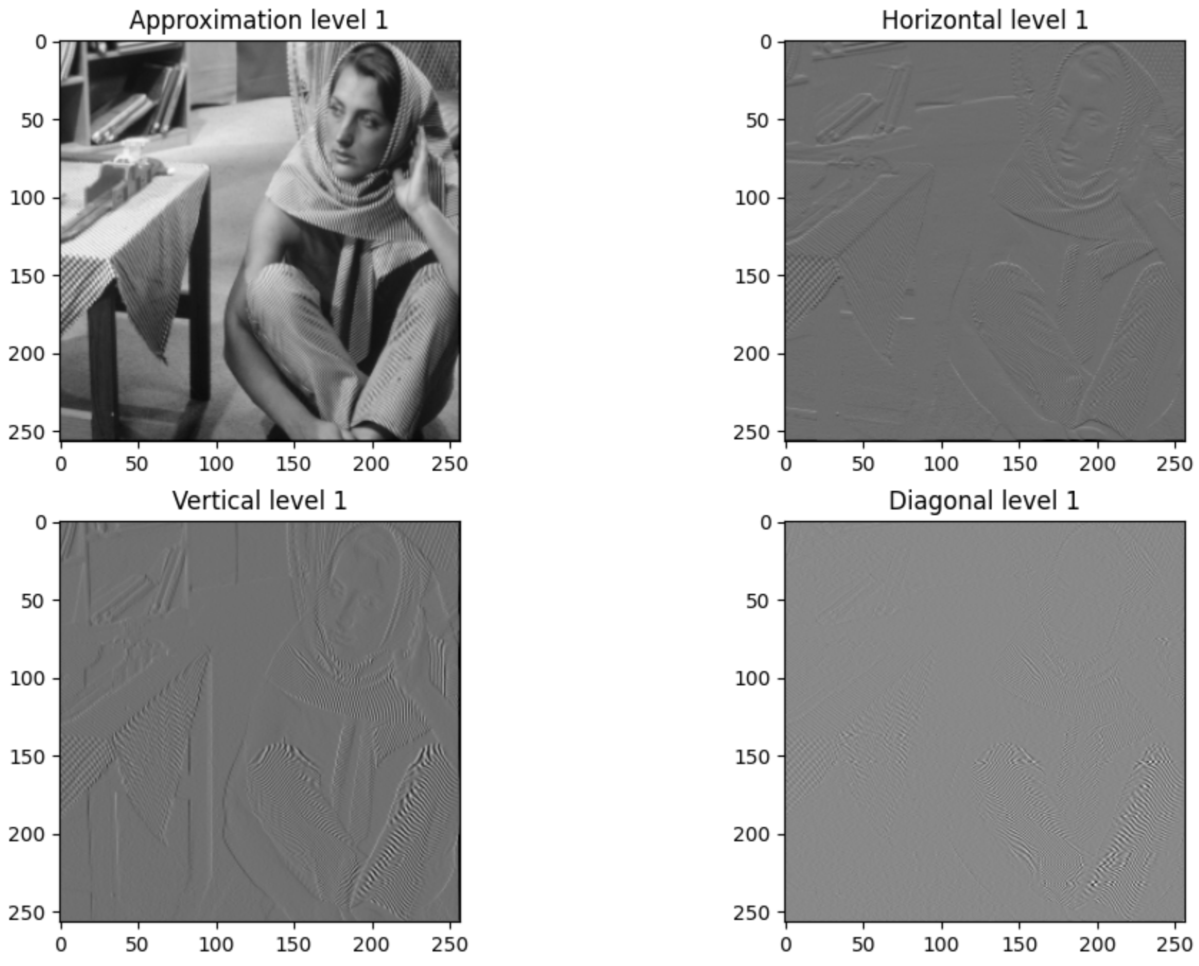
3-Haar wavelet Decomposition Algorithm



Start
Pass the image to the wavelet_decomposition() function along with the 'haar' filter and the desired number of levels
Level
Level > 0
Level=0
Use the approximation coefficient from the previous level as the input image
Filter
low-pass
high-pass
convolve the row with 'haar' low-pass filter
convolve the row with 'haar' high-pass filter
Downsample by 2
convolve the column with 'haar' low-pass filter
convolve the column with 'haar' high-pass filter
Approximation Coefficient
Horizontal Coefficient
Vertical Coefficient
Diagonal Coefficient

For downsampling, the image matrix was reduced by half at each step of applying the Haar wavelet, keeping every second value to reduce the resolution (using Python's list slicing feature) *Input image matrix: [x1,x2,x3,x4,x5,x6] after downsampling :[x1,x3,x5]*

The image below represents the wavelet decomposition coefficients at level 1(*Image:barbara*).

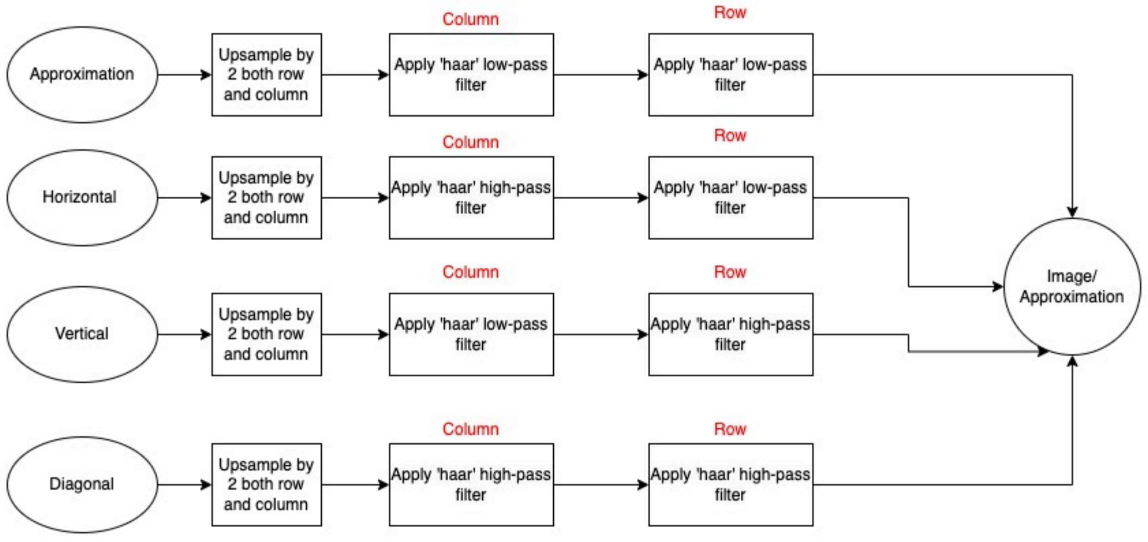


Wavelet Reconstruction:

In the reconstruction phase, the inverse process needs to be performed, which involves the following steps:

1. Upsample the coefficients by a factor of 2.
2. Convolve them with the Haar low-pass and high-pass filters.
3. Combine the outputs of the convolutions to reconstruct the image or an approximation of the previous level.

4- Haar Wavelet Reconstruction Algorithm


Approximation
Upsample by 2 both row and column
Column
Apply 'haar' low-pass filter
Row
Apply 'haar' low-pass filter
Horizontal
Upsample by 2 both row and column
Column
Apply 'haar' high-pass filter
Row
Apply 'haar' low-pass filter
Vertical
Upsample by 2 both row and column
Column
Apply 'haar' low-pass filter
Row
Apply 'haar' high-pass filter
Diagonal
Upsample by 2 both row and column
Column
Apply 'haar' high-pass filter
Row
Apply 'haar' high-pass filter
Image/ Approximation

For upsampling, zero interpolation (inserting zeros between values) was performed. First, a matrix with the size of the image (coefficients) is created, and then pixel values are inserted between zeros to restore the original resolution.

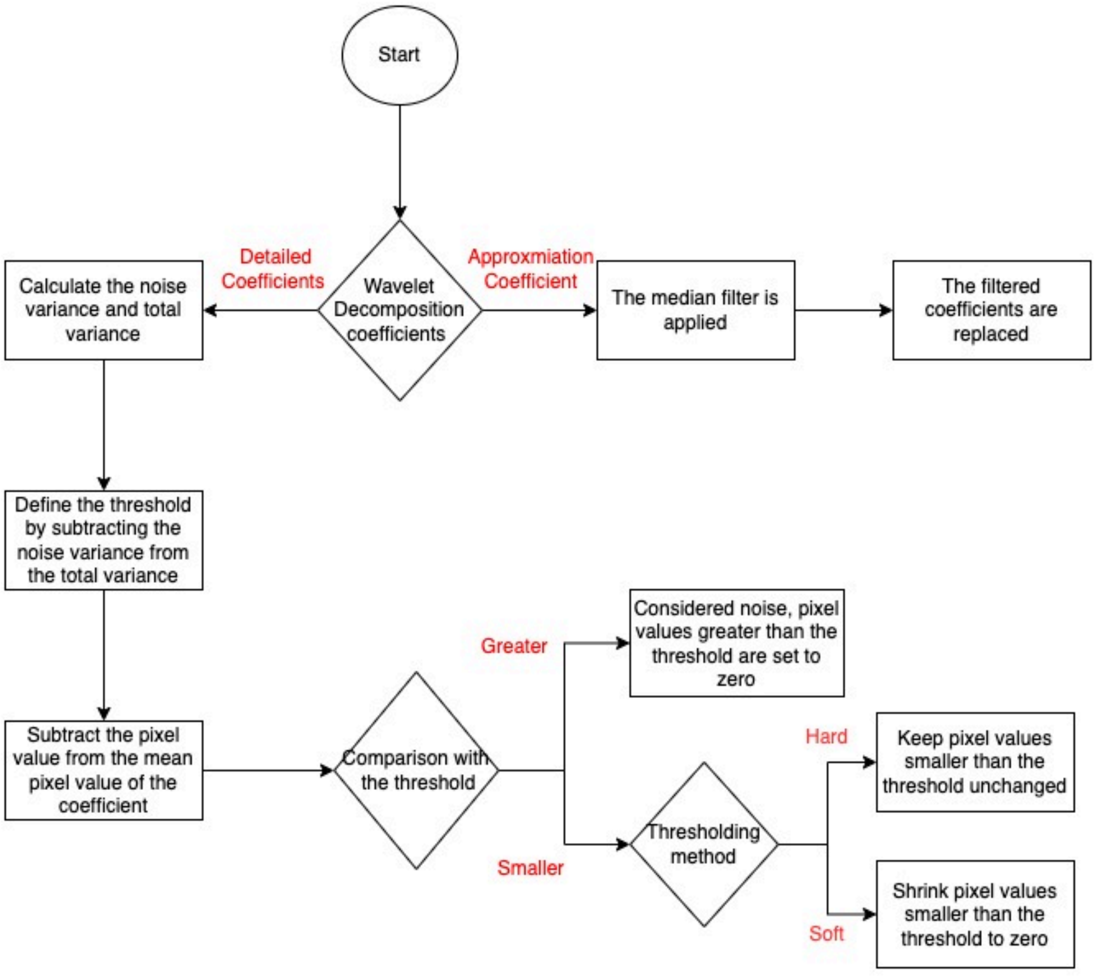
Input image matrix : [x1,x2,x3,x4] after upsampling [0,x1,0,x2,0,x3,0,x4]

By evaluating the performance through the calculation of the mean squared error (MSE) between the reconstructed image and the original, it was found that the accuracy of the reconstruction algorithm was satisfactory.

Task3:

To reduce the noise in the image using the decomposition and reconstruction wavelet transform algorithm, after obtaining the coefficients of the noisy image at three levels, and before the reconstruction process is performed, the noise in each coefficient (both low-frequency and high-frequency) must be minimized. For this purpose, the low-frequency coefficient (approximation) is passed through a median filter to reduce salt-and-pepper noise from the pixels. The high-frequency coefficients (detail coefficients) are then passed through a thresholding function, which calculates the variance of the noise and the total variance of each coefficient. By subtracting the noise variance from the total variance, a threshold is defined. The standard deviation of each pixel value is calculated by subtracting its value from the mean pixel value of the coefficient. This is then compared with the defined threshold. If the standard deviation of the pixel value is greater than the threshold, it is considered noise, and the pixel value is set to zero. If the standard deviation is smaller than the threshold, based on the chosen method ('hard thresholding' or 'soft thresholding'), the pixel value is either retained at its original value (hard) or shrunk toward zero (soft). The flowchart below illustrates the thresholding process.

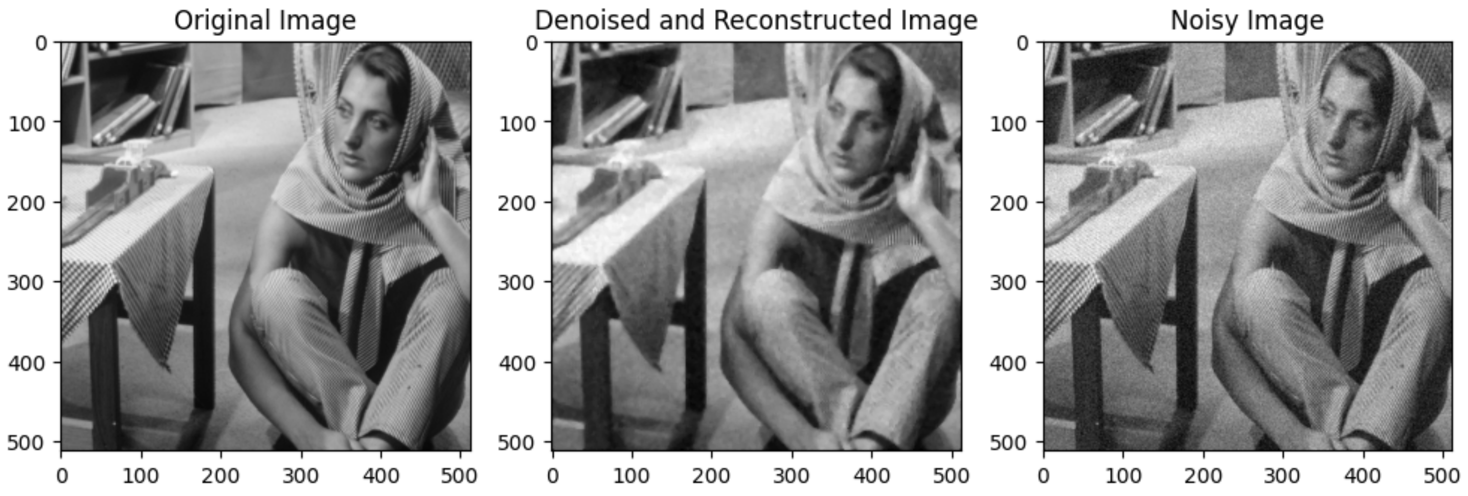
5- The flow chart of the Denoising Algorithm


Start
Wavelet Decomposition coefficients
Detailed Coefficients
Approxmiation Coefficient
Calculate the noise variance and total variance
The median filter is applied
The filtered coefficients are replaced
Define the threshold by subtracting the noise variance from the total variance
Subtract the pixel value from the mean pixel value of the coefficient
Comparison with the threshold
Greater
Considered noise, pixel values greater than the threshold are set to zero
Smaller
Thresholding method
Hard
Keep pixel values smaller than the threshold unchanged
Soft
Shrink pixel values smaller than the threshold to zero

Evaluation:

Evaluation was performed using two methods: MSE and SSIM. Both metrics show that the noise has been significantly reduced. The plots below display one of the images (Barbara) after reconstruction and denoising.

6- The comparison between images



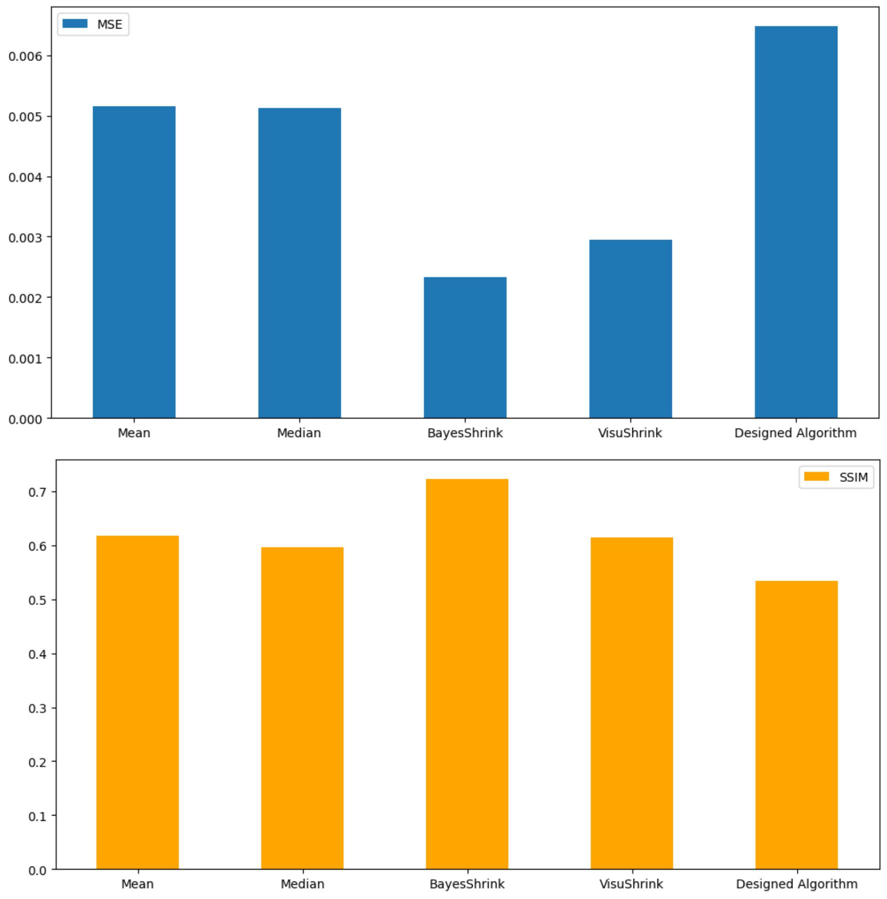
7- Evaluation table

Method	MSE	SSIM
Mean	0.00515211686423191	0.617614733738291
Median	0.00512004454838959	0.596434513067026
BayesShrink	0.00233636313867008	0.722275133190908
VisuShrink	0.00294203836210023	0.614477331733075
Designed Algorithm	0.00634372645183418	0.56926051541264

Conclusion :

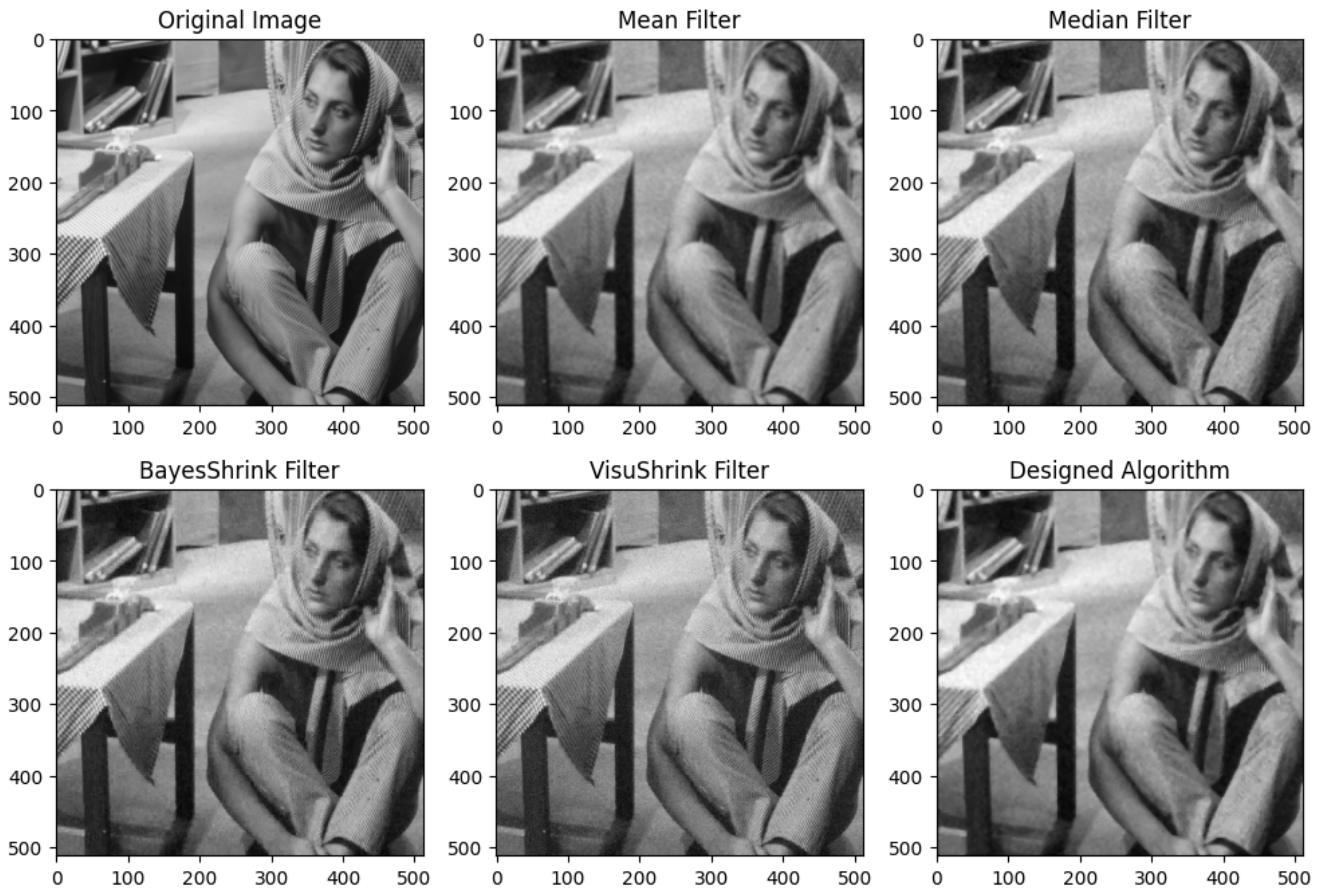
By comparing the performance of the designed algorithm with the Mean, Median filter, and VisuShrink or BayesShrink methods using the same evaluation metrics, it can be concluded that the BayesShrink and VisuShrink methods produce better results, offering higher accuracy and greater similarity to the original image(Barbara image). The table below presents a comparison of the results, and the denoised images generated by each method are displayed below the table.

8- Comparison bar charts of MSE and SSIM



The image below displays the original image alongside the denoised image (*Barbara*), achieved using various denoising methods.

9-The comparison of various image denoising methods



References:

1-*Gonzalez, Rafael C., Woods, Richard E*,”Digital image processing : global edition” ,United Kingdom, Pearson Education Canada, 2017

2-*Ravishankar Chityala, Sridevi Pudipeddi* “Image Processing and Acquisition using Python Second Edition” CRC Press 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742

3-*Maximiliano Contieri*, ”Clean Code Cookbook Recipes to Improve the Design” ,1005 Gravenstein Highway North, Sebastopol, CA 95472,O’Reilly Media, Inc.