IMAGE DENOISING WITH WAVELET TRANSFORMATION

INTRODUCTION

Images are an important way of relaying information. With the process of obtaining, processing or storing images there can be introduction of corruption or noise in the images. This can alter the quality of the image and sometimes it can also corrupt the file, making it unviewable.

Denoising is the process of removing noise from an image. This can be done in various ways, and one of the methods is to bread down the image to frequency components, remove the noise and then reconstruct the image to view an image of better quality.

1: READING PGM IMAGES

In order to denoise images, we have to first read the image and convert it into a NumPy array which makes it easier for further computations. To read PGM (Netpbm grayscale image format) images, the following steps have to be followed.

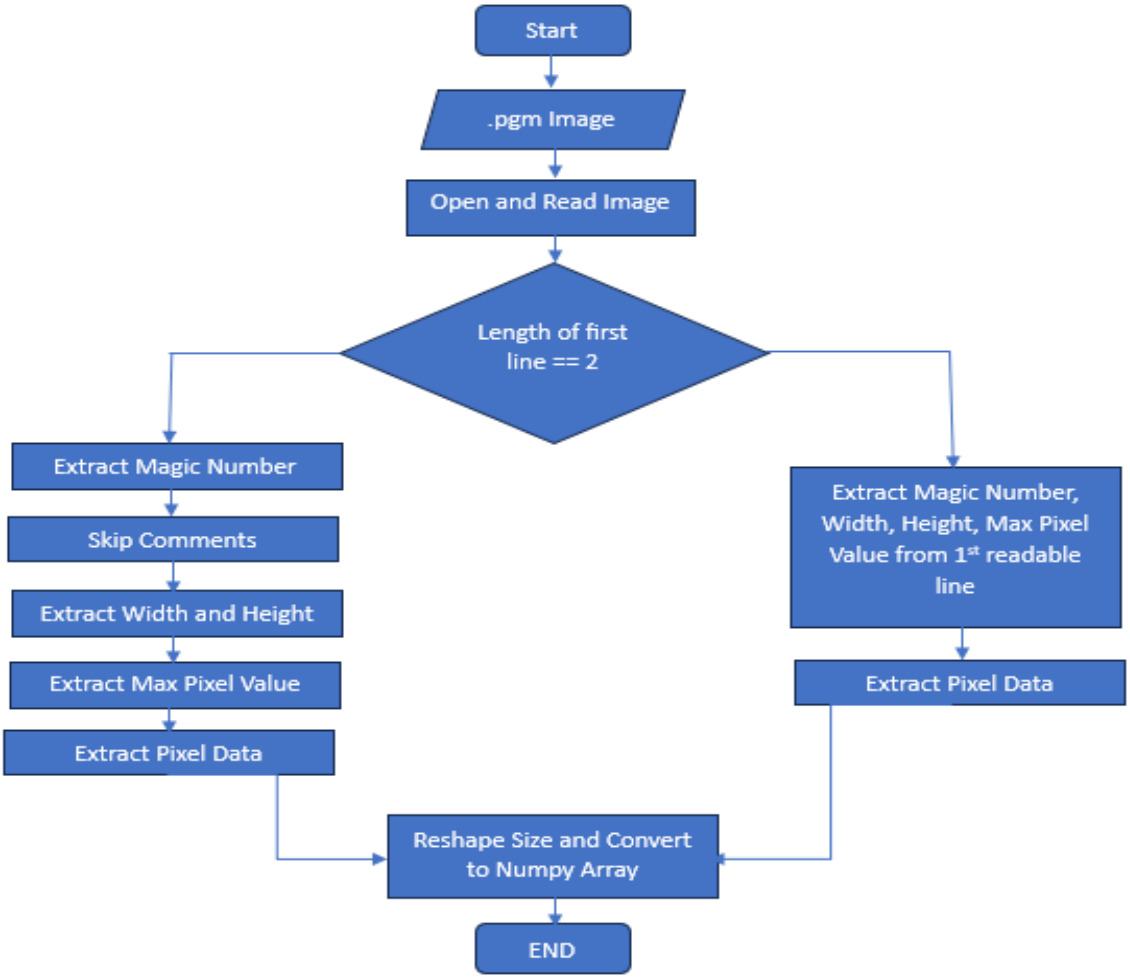

Fig 1.1: Flow of pgm_image_read function

A PGM file will have the following information:

1. Magic Number
2. A comment starting with # (Not mandatory in the file)
3. Width and Height
4. Maximum Pixel Value
5. Pixel Data

The first 4 comprise of the header data. Following it, we will have rows and columns of pixel data. If, while reading the file, the first line is not the magic number, we can assume that the image has noise. For a noisy image, the first readable line will have the magic number, width, height and maximum pixel value in the same line which is different to a normal image where they are presented in different lines.

After obtaining the pixel data, it is converted into a NumPy array from bytes. This is reshaped based on the width and height from the header information and the final NumPy array is the output of the function.

This array can be fed to an matplotlib imshow() function, which will display the image.

```
In [33]:
# Image using new function
pixel_array = pgm_image_read(r'Dataset\noisy_1\kiel.pgm' )
pgm_image = plt.imshow(pixel_array,cmap = 'gray')
plt.show()
```

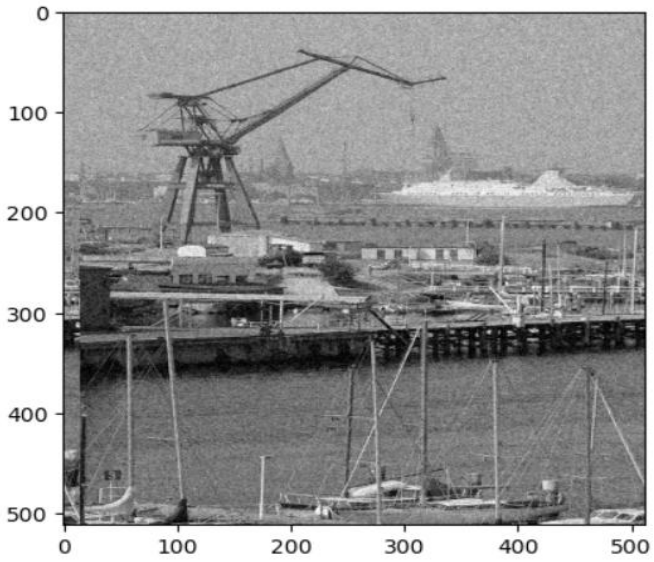

Fig 1.2: Output using pgm_image_read() and imshow()

2: WAVELET DECOMPOSITION

Wavelet Decomposition is popular frequency decomposition technique used to convert the image into frequency components in image processing. There are two outputs to this process:

- **The Low frequency component (Approximation)**: It is the overall structure of the image and is mostly smooth and contains the majority of the image's data.. Most common use is when we convert an image to a thumbnail, we can make out what the thumbnail represents, though it is not as detailed as the original image. It is also known as the **LL component** in a wavelet decomposition.
- **The High frequency components (Details):** These are the finer details of the image like the edges and textures. Noise is usually present here. There are three types of high frequency components:

    1. **LH –** Horizontal Component
    2. **HL –** Vertical Component
    3. **HH –** Diagonal Component

2.1 HAAR WAVELET TRANSFORM

FORWARD DISCRETE WAVELET TRANFORM:

For Wavelet Decomposition, Haar Wavelet transform uses **Averaging** for Approximation and **Differencing** for High Frequency components.

1. For a 1D array [8,4,6,5,9,7,8,3] :

   A. Create an array of 0's with the length of the array – [0,0,0,0,0,0,0,0]
   B. Create pairs – (8,4),(6,5),(9,7),(8,3)
   C. Perform averaging and differencing on the pairs. Populate Averaging in the 1st half of the zero array and differencing in the 2nd half.
      I. Averaging ((a+b)/2) : [6,5.5,8,5.5]
      II. Differencing ((a-b)/2) : [2,0.5,1,2.5]
   D. The final array – [6,5.5,8,5.5, 2,0.5,1,2.5]

2. For a 2D array, this transformation is applied row-wise first. Then it is applied Column-wise on the row-wise transformed 2D array.

3. The 2D array is then divided into 4 quarters.

   From the results of this array,

   i. The Top Left Quarter – LL (Approximation) component
   ii. The Top Right Quarter – LH (Horizontal) component
   iii. The Bottom Left Quarter – HL (Vertical) component
   iv. The Bottom Right Quarter – HH (Diagonal) component

4. For the next level of Decomposition, the Low Frequency component is treated as the main image and it is again split into 4 frequency components.

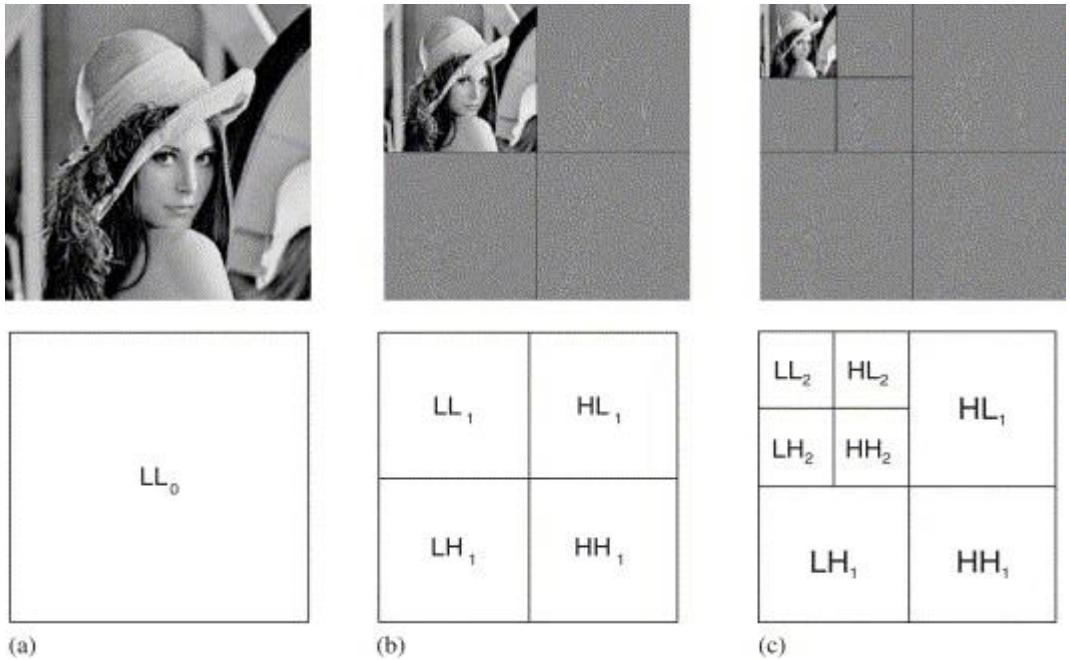

Fig. 2.1. Example DWT with corresponding notation: (a) original image, (b) one level, and (c) two level DWT. Importance prioritisation in JPEG 2000 for improved interpretability – Anthony Nguyen et al.**[1]**

The output of the Haar Wavelet FDWT function is as shown below for a 3-level decomposition:

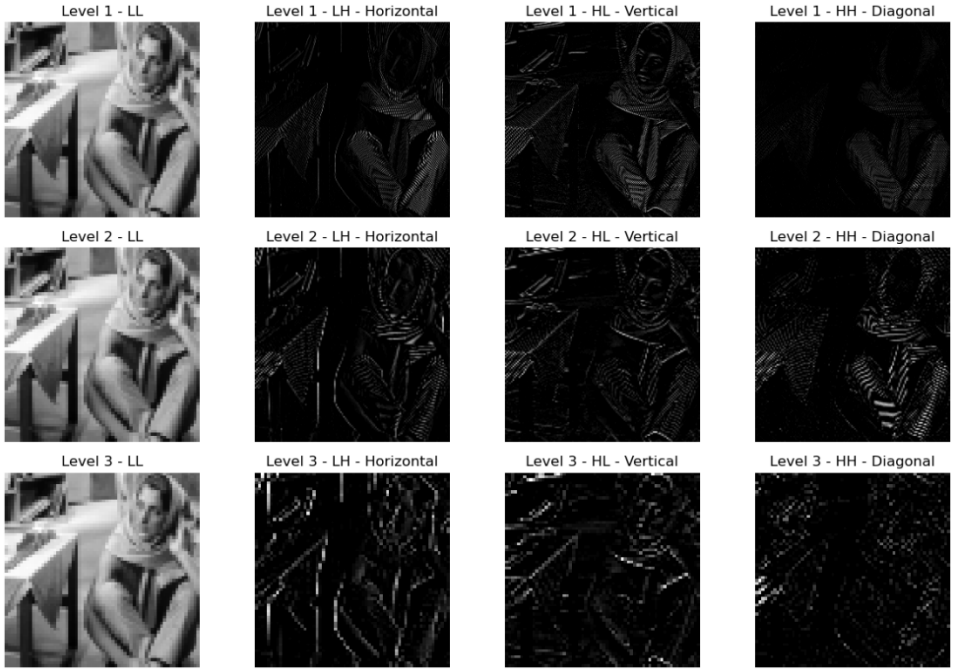

Fig. 2.2 – Haar Wavelet FDWT 3-level decomposition

INVERSE DISCRETE WAVELET TRANSFORM:

Similar to the FDWT function, the IDWT function creates a sub-function for 1D array which is then applied to on the 2D array row-wise and column-wise.

1. For 1D Array: [6,5.5,8,5.5, 2,0.5,1,2.5] (transformed array)

    A. Create an array of 0's with the length of the array: [0, 0, 0, 0, 0, 0, 0, 0]

    B. Create pairs – (6, 5.5), (8, 5.5), (2, 0.5), (1, 2.5)

    C. Perform Reconstructing (Averaging and Differencing) for the pairs. Populate the Averaging in the 1st half of the zero array and Differencing in the 2nd half:

    1. Reconstruct Averaging (a = (LL + difference)): [8, 6, 9, 8]
    2. Reconstruct Differencing (b = (LL - difference)): [4, 5, 7, 3]

    D. The final array (reconstructed): [8, 4, 6, 5, 9, 7, 8, 3]

2. For 2D Array (Reconstruction Process)
    A. Get the 2D Array into 4 Quarters:
        - Top Left – LL (Approximation) component
        - Top Right – LH (Horizontal) component
        - Bottom Left – HL (Vertical) component
        - Bottom Right – HH (Diagonal) component
    B. Apply IDWT Column-wise First
    C. Apply IDWT Row-wise on the column-wise transformed 2D array

This will result in the reconstructed image which should be very similar to the original image. To test the performance of the function, MSE is calculated between the original image and the reconstructed image as shown in Fig 2.3. The MSE is very good with a rounded-up value of 0.

```
Mean Square Error (MSE): 8.613523752410593e-33
```

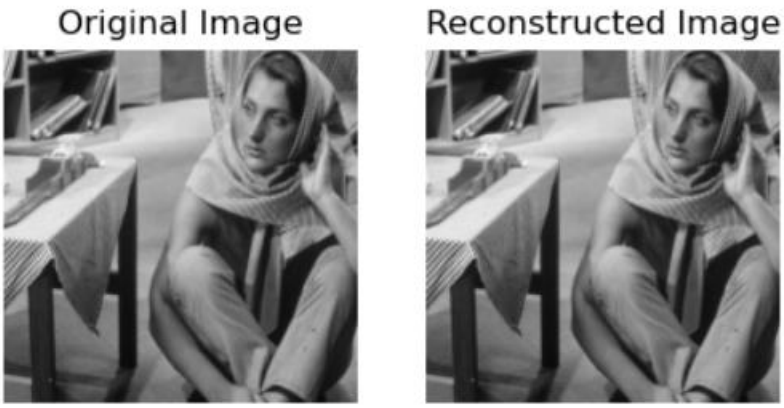

Fig 2.3 Comparison between original and reconstructed image

3: IMAGE DENOISING

Image Denoising is the process of removing noise from the image. Here, Wavelet thresholding[2] has been used to remove the noise from the image. The process is as follows:

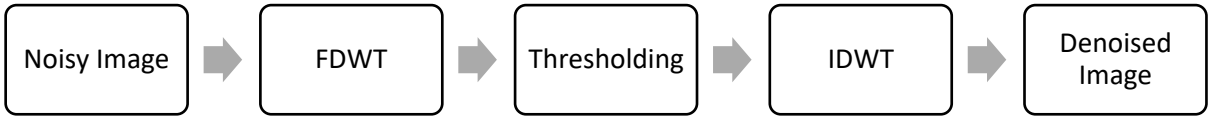

Fig 3.1 Image Denoising process

Thresholding is applied on the FDWT details (high frequency components) of the noisy image. Hard Thresholding keeps coefficients greater than threshold and sets the others to zero. Soft Thresholding shrinks the lesser coefficients towards zero by threshold.

After Thresholding, the thresholded values are provided to the IDWT function which reconstructs the image. Fig 3.2 is the result of soft thresholding.

```
Mean Square Error (MSE): 0.02926648170320704
Structural Similarity Index (SSIM): 0.575868289780329
```

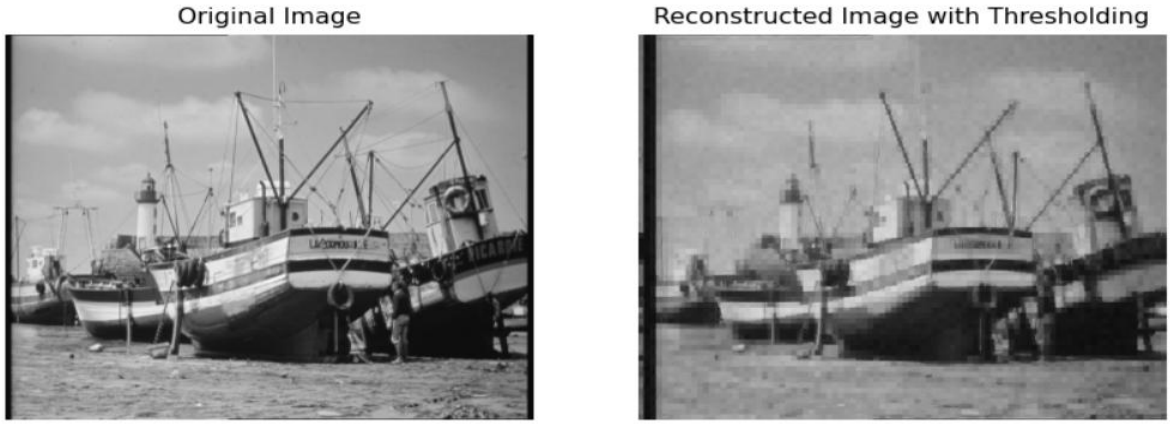

Fig 3.2 – Original and reconstructed noisy image comparison using wavelet thresholding

CONCLUSION:

Image Denoising is an important part of image processing and is used often for deep learning and AI models for advanced uses like face recognition, restoring images capturing history, etc.

Haar Wavelet is simple but also effective for discrete wavelet transformation as seen in this report. Wavelet thresholding has also provided good results as seen in Fig 3.2.

References:

[1] Importance prioritisation in JPEG 2000 for improved interpretability – Anthony Nguyen et al.
[2] 2D-Discrete Wavelet Transformation and its applications in Digital Image Processing using MATLAB - Koushik Chandrasekaran