Improving Fingerprint Recognition Using Image Processing and Machine Learning Techniques

by

Andres Bolivar Rojas Bustos

A dissertation submitted in partial fulfillment of the requirements for the degree of:

MASTER OF SCIENCE IN ELECTRONICS

in the

National Institute for Astrophysics, Optics and Electronics

August 2021

Santa Maria Tonantzintla, Puebla

Under the supervision of:

Dr. Gordana Jovanovic Dolecek, INAOE

Abstract

This thesis presents a fingerprint recognition system based on ensemble classifiers with linear discriminants, a type of supervised learning algorithm that has not been reported so far for fingerprint recognition systems.

First, a graphical application composed of several simple and interesting examples on the main topics of the image processing area is presented. This was used as a foundation to understand and apply the fundamentals of image processing in the recognition system. Subsequently, important concepts about biometrics are introduced, specifically about fingerprints and why they are used for recognition systems. Then there is a brief description of the fingerprint image databases used in this work. Next, the preprocessing algorithm is described, which applies several image processing techniques such as enhancement, binarization, segmentation, etc., in order to eliminate noise and improve the appearance of images. The different types of transformations applied to the enhanced images are also reported in order to calculate representative features that will be used to train supervised learning algorithms. These algorithms are briefly described comparing their advantages and disadvantages.

After, the results obtained by the recognition system that has been implemented in MATLAB and Python are presented. Finally, the conclusions of this research are reported, as well as future work that could improve the general results obtained.

Resumen

En este trabajo se presenta un sistema de reconocimiento de huellas digitales basado en clasificadores de conjunto con discriminantes lineales, una clase de algoritmos de aprendizaje supervisado que no ha sido reportado hasta el momento para este tipo de sistemas de reconocimiento.

Primero se presenta una aplicación grafica compuesta de varios ejemplos simples e interesantes sobre los temas principales del área de procesamiento de imágenes, Esta se utilizó como base para comprender y aplicar los fundamentos de esta área en el sistema de reconocimiento. Posteriormente se introduce conceptos importantes sobre biometría, específicamente sobre huellas digitales y porque se usan para sistemas de reconocimiento. Luego se tiene una breve descripción de las bases de datos de imágenes de huellas digitales utilizadas en este trabajo. A continuación, se describe el algoritmo de preprocesamiento implementado, el cual aplica varias técnicas de procesamiento de imágenes como mejora, binarización, segmentación, etc., con el fin de eliminar el ruido y mejorar el aspecto de las imágenes. También se reportan los diferentes tipos de transformaciones aplicadas a las imágenes mejoradas con el objetivo de calcular características representativas que serán utilizadas para entrenar a los algoritmos de aprendizaje supervisado. Estos algoritmos son descritos brevemente comparando sus ventajas y desventajas.

Posteriormente se presentan los resultados obtenidos por el sistema de reconocimiento que ha sido implementado en MATLAB y Python. Finalmente, se reportan las conclusiones de esta investigación, así como el trabajo futuro que podría mejorar los resultados generales obtenidos.

CONTENTS

Preface

Fingerprint recognition is of vital importance in many areas of industry, academia, security, banking, personal identification, etc. There is a wide variety of fingerprint recognition systems on the market, and each has its advantages and disadvantages. In this thesis, the main objective is to explore new types of fingerprint classification based on machine learning algorithms. We started the development of this work by creating a graphical approach to the fundamentals of image processing (IP). This application exemplifies many basic concepts of the image processing area, which are necessary to implement or design any system that works with images, as it is a fingerprint recognition system in this case.

The Image Processing Toolbox extends MATLAB's ability to handle IP problems by providing a collection of specific functions that create an environment suitable for solving digital image processing problems. Other toolboxes used to supplement the Image Processing Toolbox in this work include Wavelet, and the Statistics and Machine Learning Toolboxes.

The design and evaluation of a fingerprint recognition system are also implemented in MATLAB with the novelty of using machine learning models, specifically ensemble classifiers, obtaining promising results for some public fingerprint databases used in the current work.

The biometrics behind fingerprints and relevant concepts regarding their application and morphology are presented in Chapter 2. To evaluate the performance of the system designed in this work, it was decided to use several public fingerprint databases, which are described in Chapter 3. Chapter 4 presents the preprocessing of fingerprint images covering many image processing techniques like enhancement, binarization, etc. Another important section is feature extraction, presented in Chapter 5, where the types of transformations applied to fingerprints are listed and explained to obtain representative features of them, thus creating a training set that will serve as input to a classifier. Chapter 6 briefly defines the machine learning algorithms evaluated in this paper.

After having defined all the necessary components for the classification process, the results obtained from the system implemented in MATLAB are presented in Chapter 7. From another point of view, we decided to implement an analogous system in Python, which is evaluated in Chapter 8 reporting its corresponding results. Finally, the conclusions of this research and the future work are exposed in Chapter 9.

CHAPTER 1 . HANDS-ON APPROACH TO IMAGE PROCESSING FUNDAMENTALS

1.1. MOTIVATION

In this work, a tool has been developed to learn basic image processing (IP) techniques in a friendly way to provide a stable foundation to develop a fingerprint recognition system.

Since we are very familiar with MATLAB, the application has been created using this software in conjunction with the Graphical User Interface Development Environment (GUIDE). This MATLAB-based tool is used as a helpful fundament to understand and apply several image processing techniques, which later will be used to develop a preprocessing algorithm to enhance fingerprint images.

The tool includes the following topics of IP: Presentation of Images (histogram, noise, and frequency representation of images), Image Enhancement, Image Restoration, Image Compression, Morphological Image Processing, and Image Segmentation.

1.2. GENERAL DESCRIPTION

The following figure shows the main GUI for this platform indicating the general content. The following groups of demo programs (modules) are included: Presentation of Images, Image Enhancement, Image Restoration, Image Compression, Morphological Image Processing, and Image Segmentation. Clicking on each module opens the corresponding menu. The more detailed content of each menu is presented below.

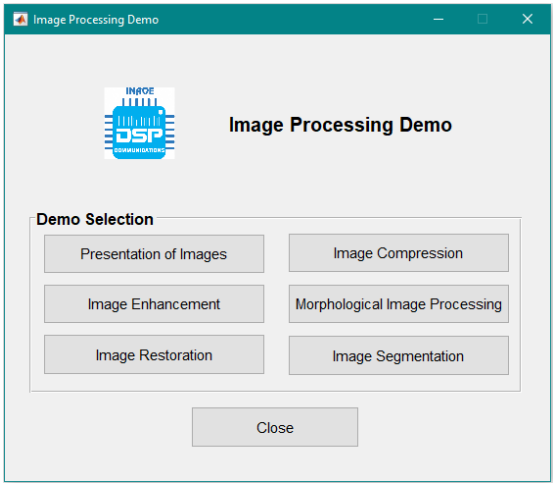

Figure 1.1. Main GUI for the proposed platform.

Presentation of Images: The Presentation of Images module includes examples for histogram of an image, noise in images, and frequency domain representation of images.

Image Enhancement: The Enhancement module includes 4 subsections each with two examples:

- Histogram Processing: Histogram Equalization and Contrast-limited adaptative histogram equalization (CLAHE).
- Spatial Filtering: Laplacian filter (Linear) and Median filter (Non-linear).
- Fuzzy Techniques: Contrast Enhancement and Edge Detection.
- Frequency Filtering: Filters in the Frequency Domain and Notch filters (Moire Pattern).

Image Restoration: The Restoration module includes examples for Wiener Filtering and Lucy-Richardson Algorithm.

Image Compression: The Compression module includes examples for Compression by Quantization, and JPEG and JPEG 2000 Compression.

Morphological Image Processing: The Morphological Processing module presents examples for Smoothing using openings and closings and Compensating for a nonuniform background.

Image Segmentation: The Segmentation module includes examples for Local vs Global Thresholding and Segmentation using gradients and the watershed transform.

All demo presentations are primarily based on [1]. Clicking on each module in the menu opens the corresponding demo presentation. The typical configuration of the demo presentations is shown in the following figure, presenting different sections that appear in the GUI.

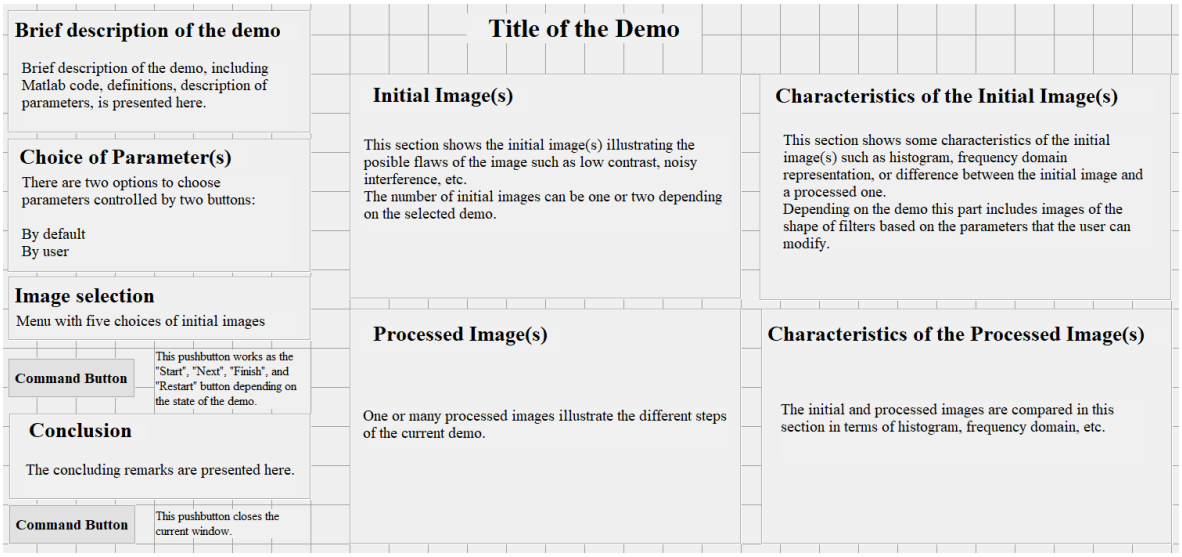

Figure 1.2. General configuration of each demo in the platform.

At the top left of the window is a brief description of the current demo that provides a brief introduction to the IP technique used in this example and explanations of the parameters used. The name of the implemented MATLAB function is also presented.

The Choice of parameters section provides the choice of default parameters or user input.

Another feature of the platform is the ability to select images for each demo. This platform provides a menu with five different images that can be selected and analyzed.

There are two command buttons. The first button functions as the "Start", "Next", "Finish", and "Restart" button depending on the state of the demo. The second command button serves as a “Return to the Previous Menu” that closes the current window allowing the user to return to the main menu.

There are different main sections including the initial image(s), processed image(s), and characteristics of each, in the center of the window. The user can view, step by step, the processed images and the effect that the parameters have on each procedure.

A conclusion on the IP technique(s) used in the current example is provided at the bottom left.

1.3. DETAILED DESCRIPTION

This section describes in detail all the examples of this platform.

1.3.1. General operation of each demo

All demos use a “Start” button that changes to “Next” after pressing it. Depending on the number of cases for each demo, this button will eventually change to the “Finish” state. This last window will be shown in the next descriptions for the rest of the demos, to reduce the number of figures that are all similar in their functionality, but each one with a different conclusion. As an example, the first appearance for the Histogram of an Image demo is presented in Figure 1.3. The “Number of bins” parameter can be changed before pressing the “Start” button if desired. After pressing the button, this will change to “Next” as can be seen in Figure 1.4. This figure presents an overexposed image (high-intensity gray levels for the pixels), this can be noticed in the corresponding histogram also presented in Figure 1.4. By pressing the “Next” button in this GUI, the following case will appear, as can be seen in Figure 1.5. This image has a low contrast showing low gray level intensity values, which justifies the dark appearance of the image. This characteristic is visible on the histogram, composed mainly of low levels. By pressing the “Next” button one more time, the GUI will change as shown in Figure 1.6. This is the final case in this demo, where all gray levels in the image span the entire range of the histogram. A conclusion text will appear in the GUI, giving a final inference on image histograms. The button will change to “Finish” and when pressed, the final GUI displays all the above cases as a summary for a final contrast and histogram comparison. This GUI is presented in Figure 1.8. The button will change to “Restart”, indicating a reset of the example to the original window presented in Figure 1.3.

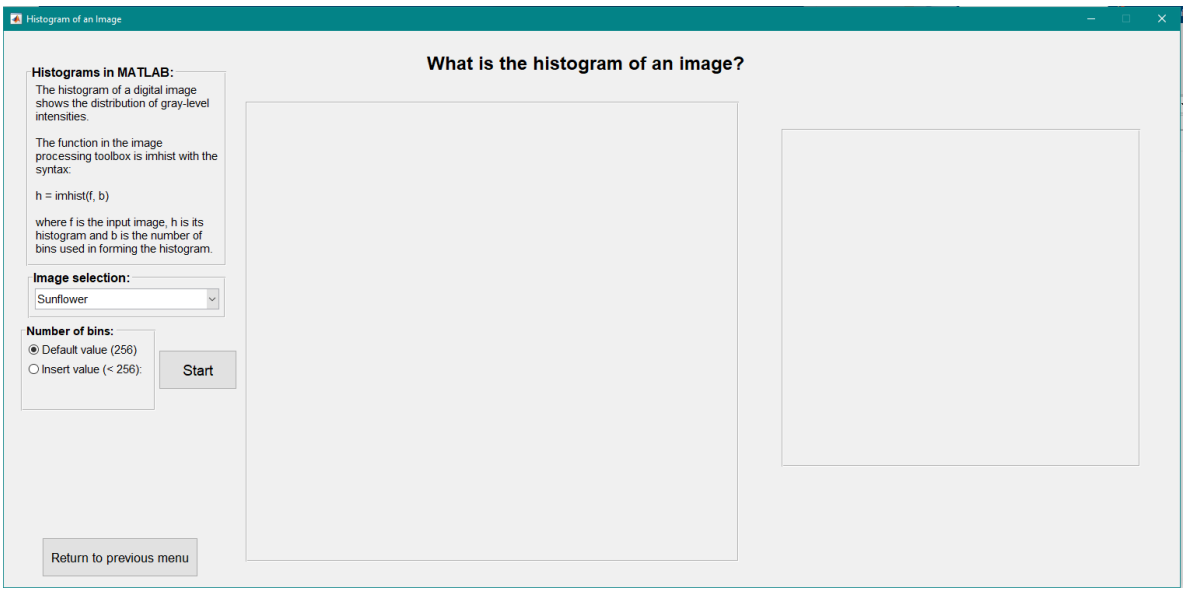

Figure 1.3. First appearance of the Histogram of an Image demo.

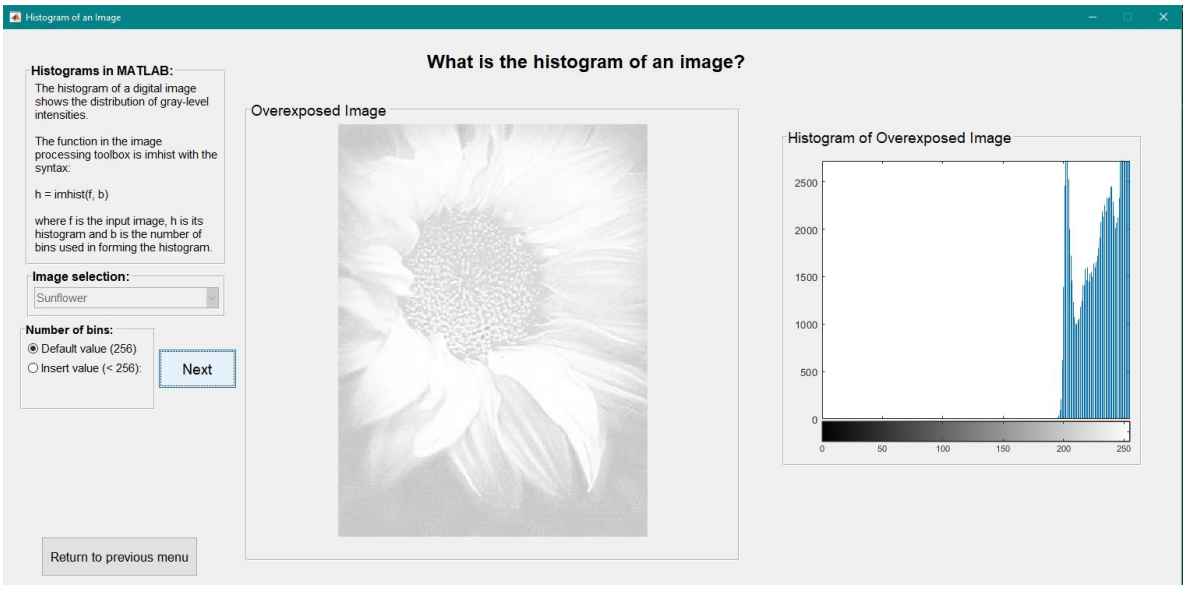

Figure 1.4. Overexposed image in the Histogram of an Image demo.

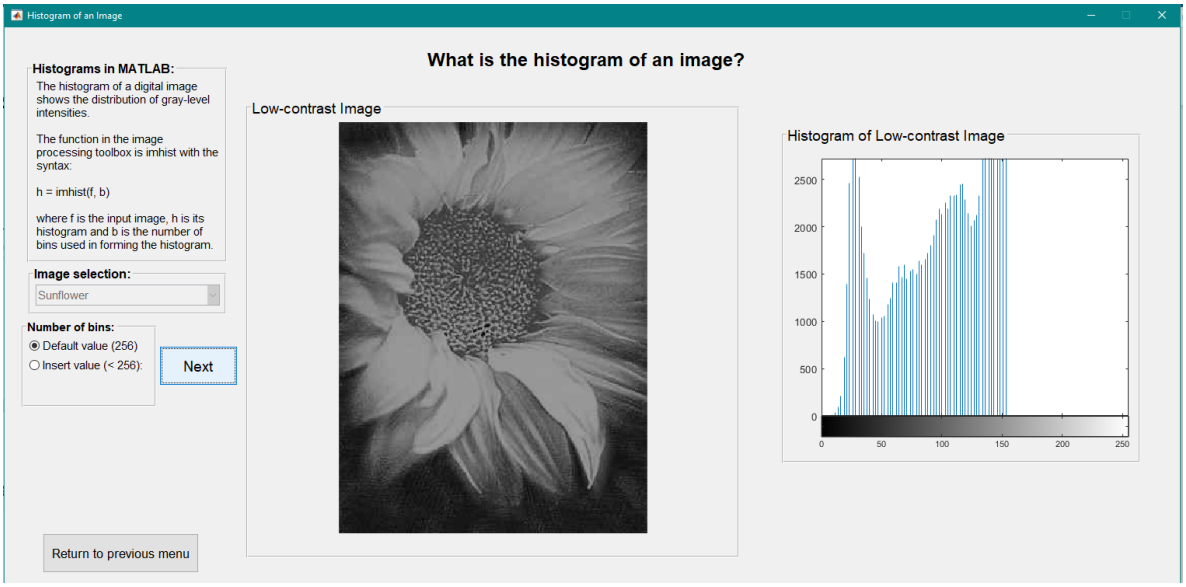

Figure 1.5. Low contrast image in the Histogram of an Image demo.

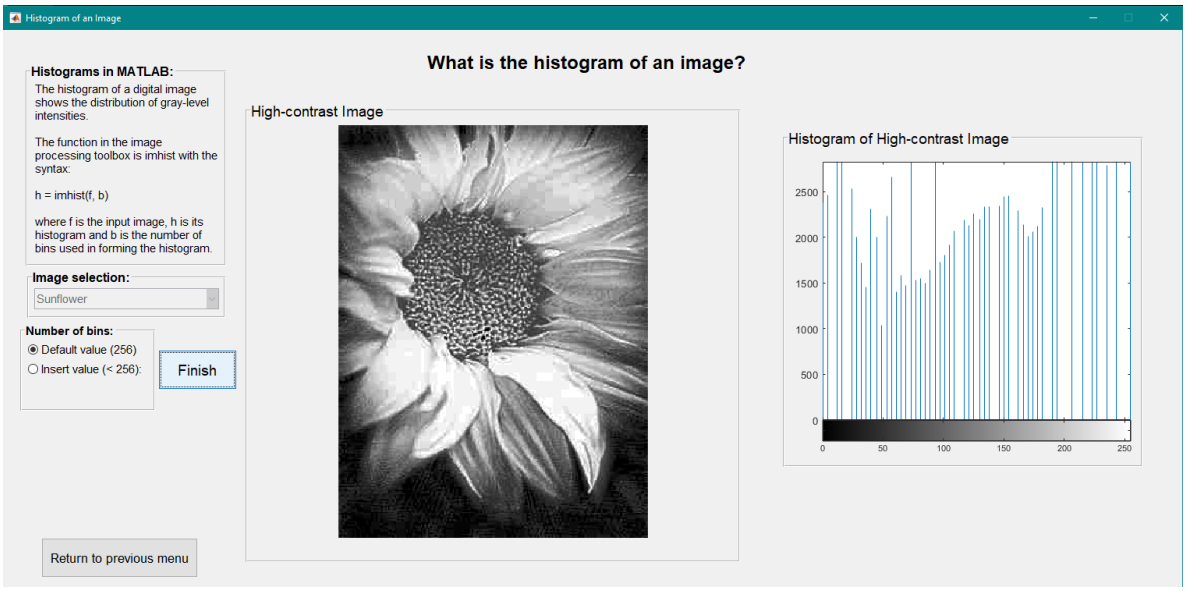

Figure 1.6. High contrast image in the Histogram of an Image demo.

1.3.2. Presentation of Images Module

Figure 1.7 presents the Presentation of Images menu showing three demos that serve as an introduction to important concepts such as histograms, noise, or frequency domain representation of images. Each menu has a button to return to the main menu, closing the current window and going back to the main GUI.

The three demos included in this module are Histogram of an Image, Noise in Images, and Frequency Domain Representation.

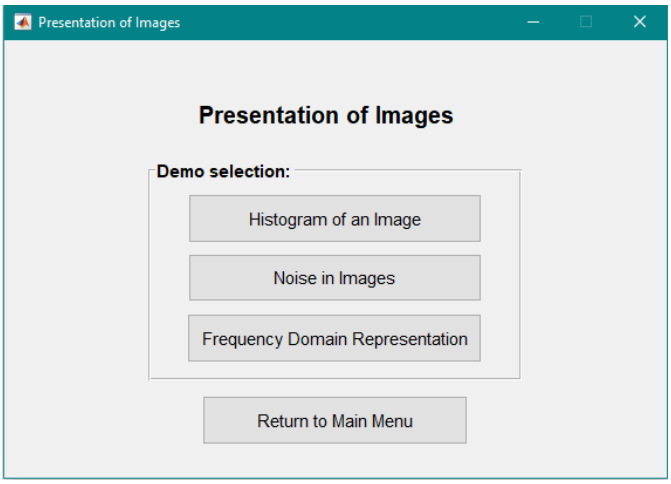

Figure 1.7. Presentation of Images GUI.

1.3.2.1. Histogram of an Image

This demo introduces image histograms and their use in image processing. This example deals with three different versions of the same image and their respective histograms, the idea is to analyze the general distribution or shape of the histogram and its correspondence with the contrast of each image. Figure 1.8 presents the comparison of the three versions of this demo using the “Sunflower” image. The contrast change between the three images is noticeable, which translates into the general distribution of each histogram. Another example with the “Cheetah” image is presented in Figure 1.9.

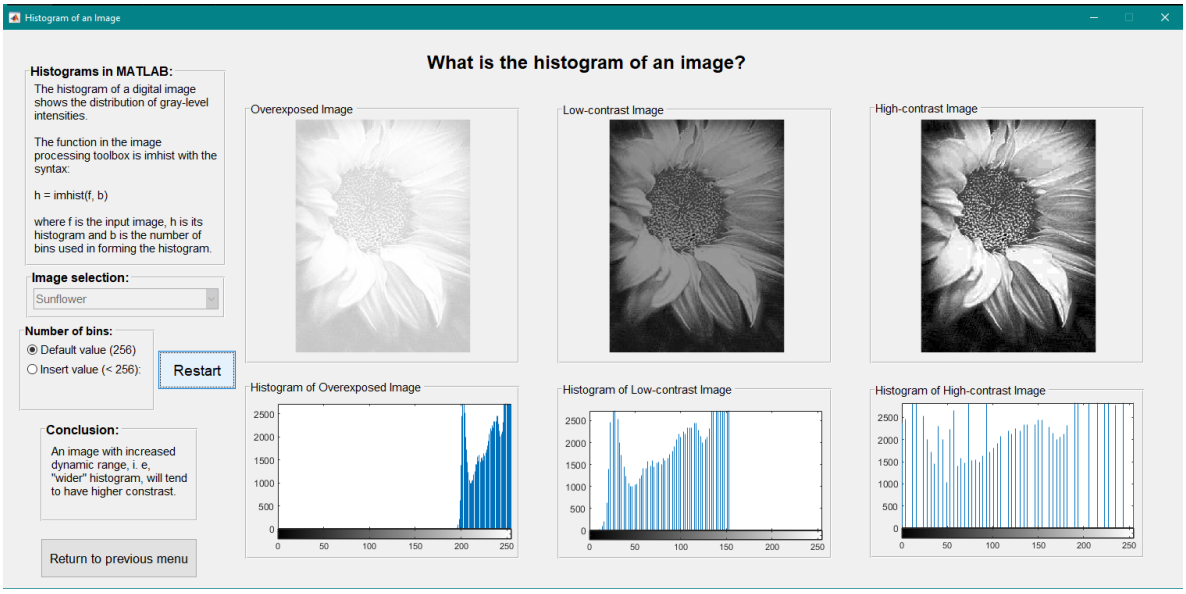

Figure 1.8. Histogram of an Image GUI, "Sunflower" example.

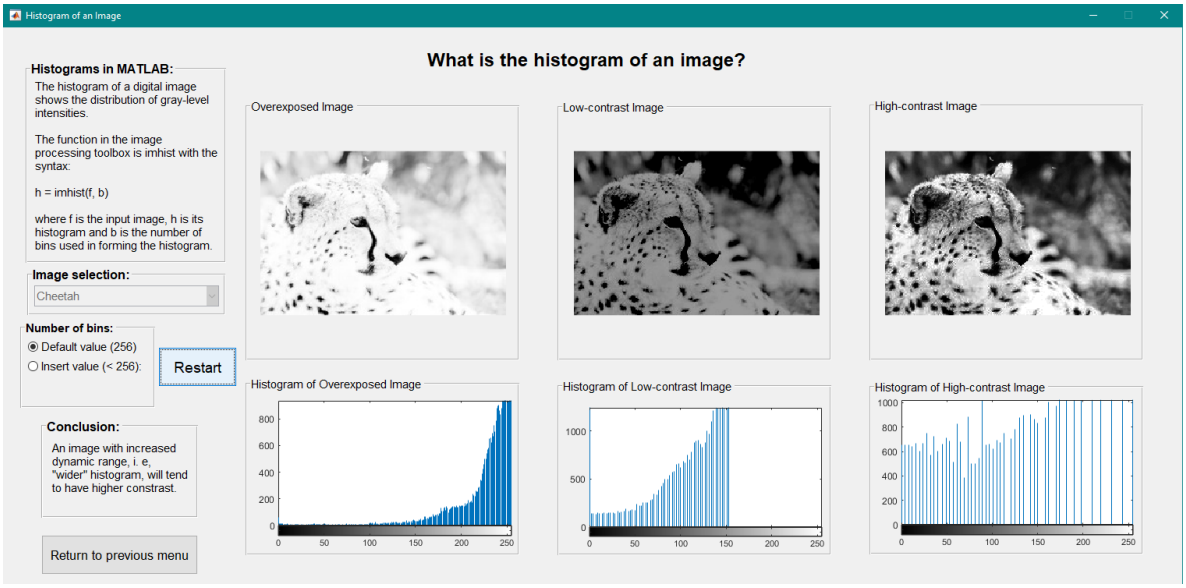

Figure 1.9. Histogram of an Image GUI, "Cheetah" example.

1.3.2.2. Noise in Images

This demo presents an introduction to some types of noises commonly found in corrupted images. These noises can be caused by interference with the imaging process or defective sensors. Figure 1.10 shows an example of this demo using the “Cat” image with the default values applied for each type of noise. The idea is to add artificial noise to an image and analyze the result for each type of noise. The four types of noises and their respective parameters are summarized in the following table:

Table 1.1 Different types of noises included in this demo.

Type of Noise	Noise Density	Mean	Variance	No parameters
Salt & Pepper	X			
Gaussian		X	X	
Poisson				X
Speckle			X	

The three parameters can be changed during the demo operation. Another example using the “Flowers” image is presented in Figure 1.11.

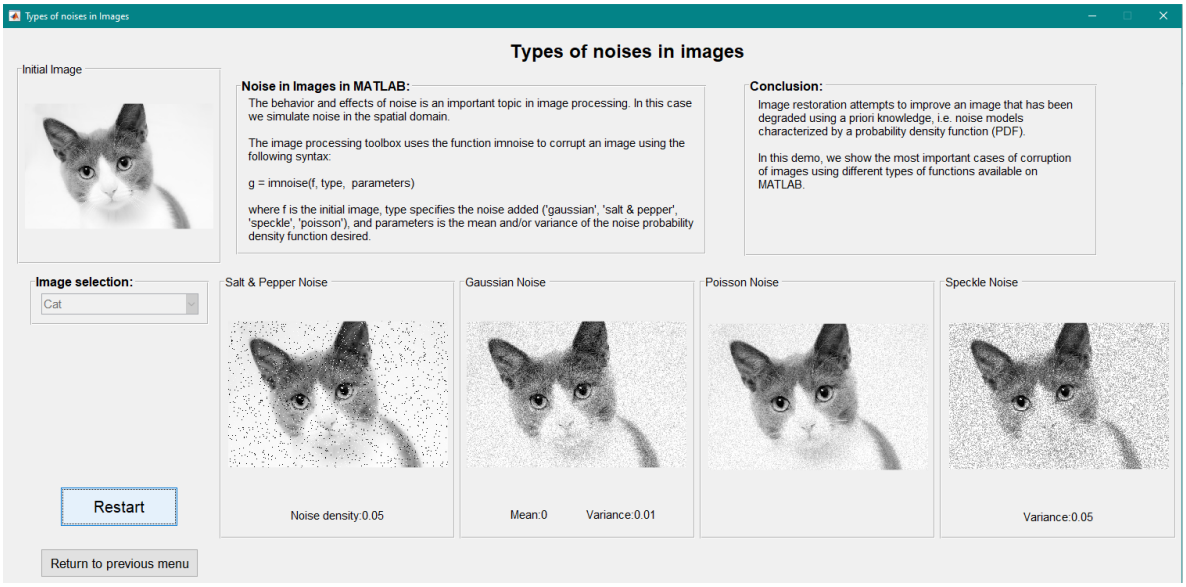

Figure 1.10. Noises in Images GUI, "Cat" example.

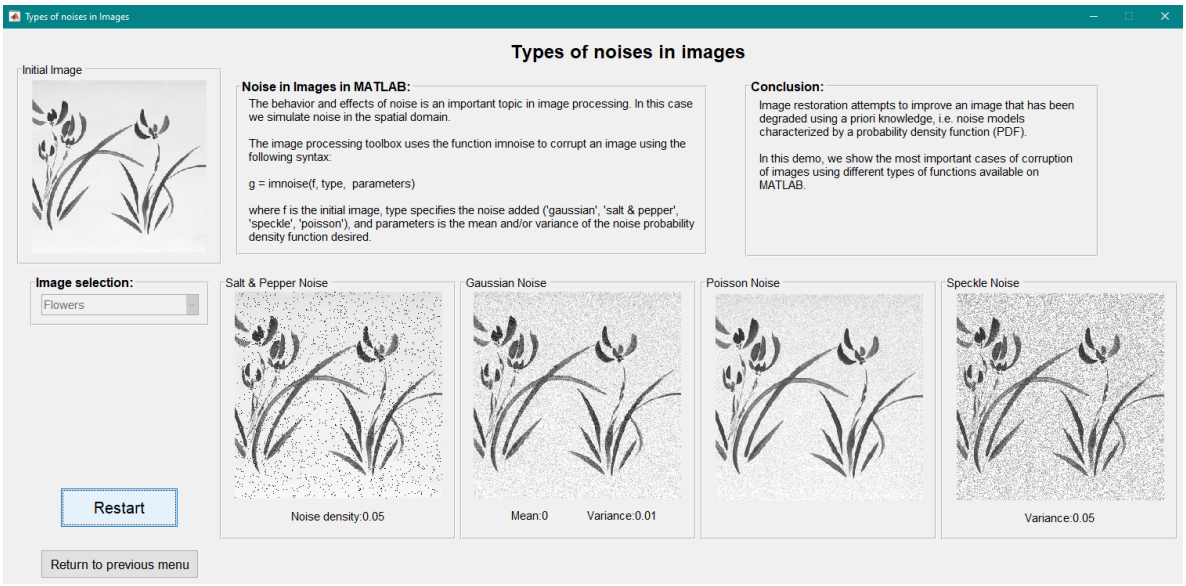

Figure 1.11. Noises in Images GUI, "Flowers" example.

1.3.2.3. Frequency Domain Representation

This demo introduces the Fast Fourier Transform (FFT) for images. Each example features two different images, including a normal image and a Moire pattern image. The respective magnitude and phase responses are presented, including a logarithmic magnitude transformation for better resolution and contrast of this response when the range of values is too wide (this explains why the Original Spectrum image is mostly dark). The parameters included in this demo are P and Q, the number of rows and columns of the FFT, respectively (this transformation is a matrix of size $P \times Q$). Figure 1.12 shows the final window of this demo for the example “First Case: Bulbs - Second Case: Lady”.

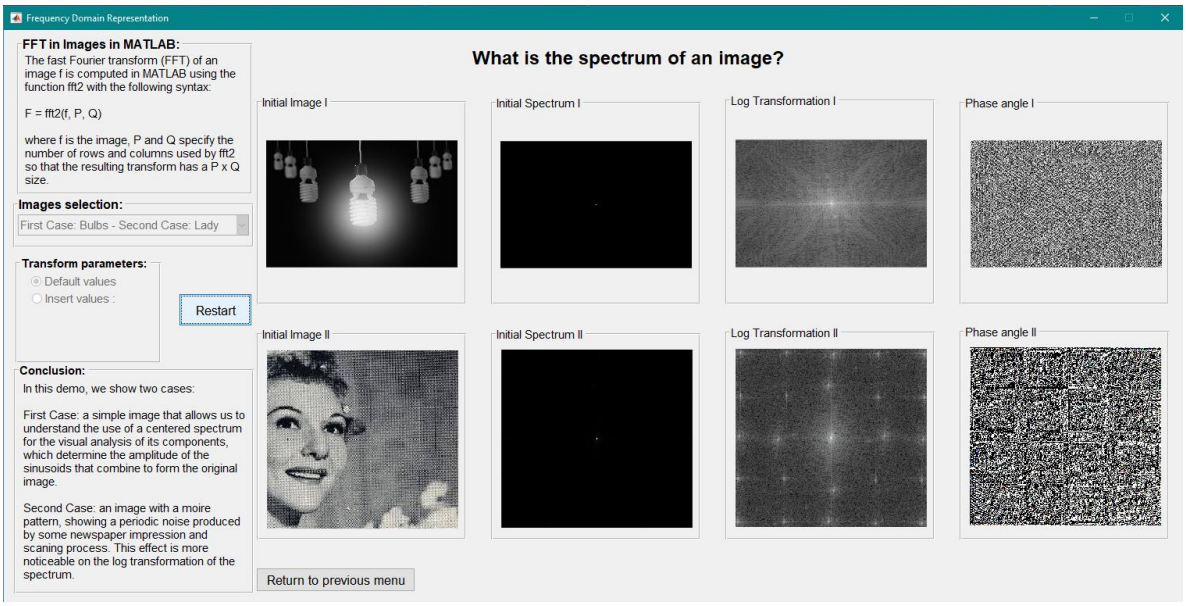

Figure 1.12. Frequency Domain Representation GUI, "First Case: Bulbs - Second Case: Lady" example.

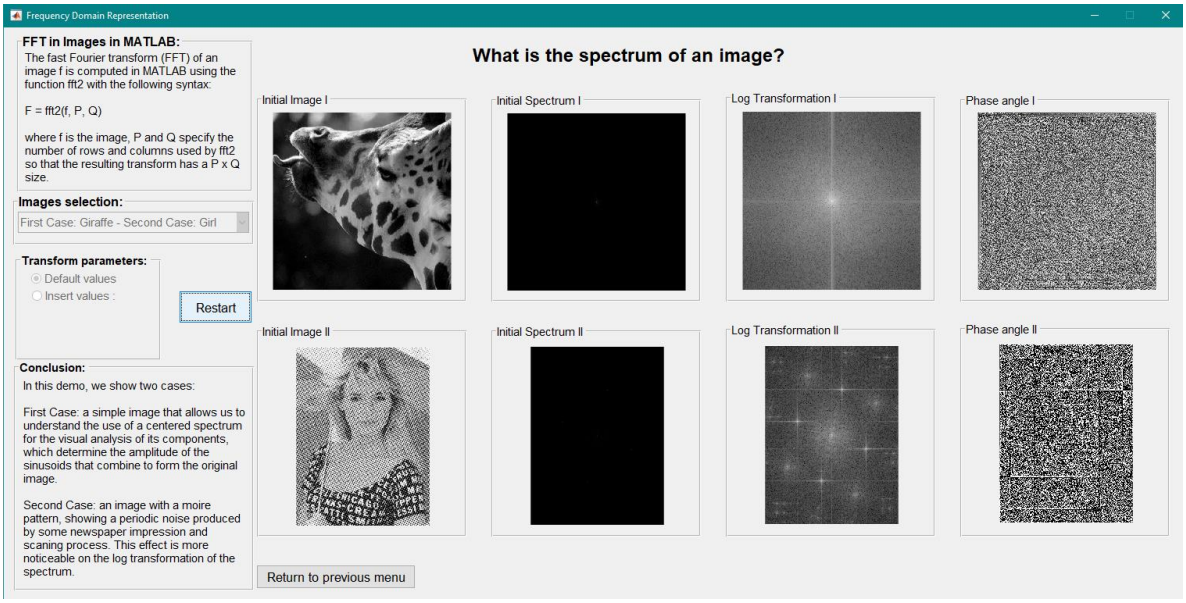

Figure 1.13. Frequency Domain Representation GUI, "First Case: Giraffe - Second Case: Girl" example.

The previous image presents another example that compares a “Giraffe” image, completely free of noise, with a “Girl” image corrupted by a Moire pattern. The difference is noticeable in the Log Transformation images that show the distribution of high-intensity peaks in the spectrum of the Moire pattern image (white dots in the image area), strictly related to the periodic noise introduced by this effect. This problem occurs mainly when scanning newspaper pictures or images.

1.3.3. Image Enhancement Module

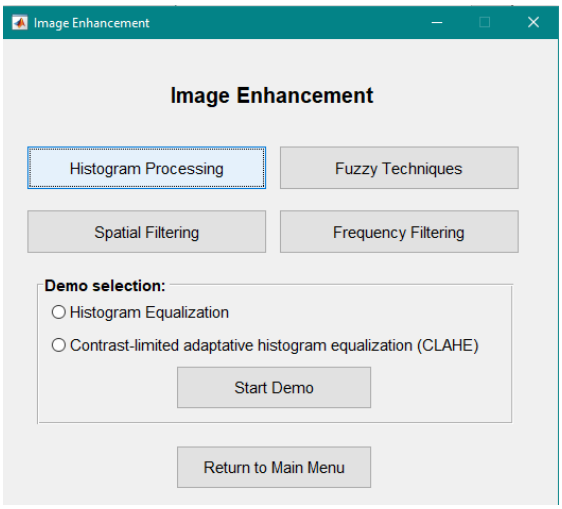

Figure 1.14. Image Enhancement GUI.

The image enhancement module implements four different IP enhancement techniques. In the Demo selection section, there are two options for each technique, these options will change according to the corresponding button. For example, Figure 1.14 shows the Histogram Equalization and Contrast-limited adaptative histogram equalization (CLAHE) options, which belong to the Histogram Processing button. Using the Spatial Filtering button, the options will change to Laplacian filter (Linear) and Median filter (Non-linear) as shown in Figure 1.15.

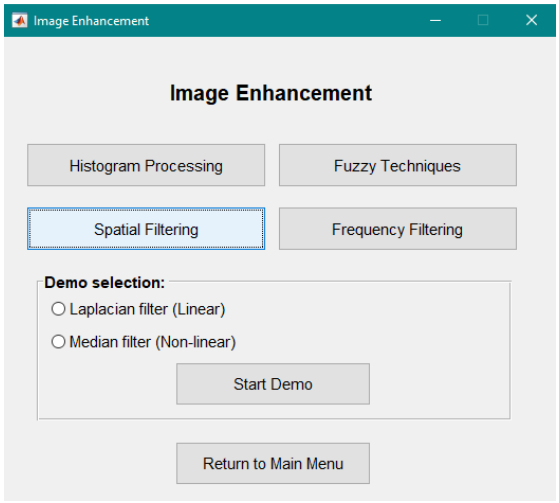

Figure 1.15. Image Enhancement GUI after pressing the Spatial Filtering button.

The same behavior applies to the Fuzzy Techniques and Frequency Filtering buttons. In total, there are 8 different demos for this image enhancement module, detailed in the following section.

1.3.3.1. Histogram Processing

1.3.3.1.1. Histogram Equalization

The first demo presents an example of histogram equalization. The “gray levels” parameter specifies the number of bins for the histogram calculation, this value has a direct impact on the number of gray levels for the equalized image. Figure 1.16 shows this demo in its final step for the “Cheetah” example. A comparison between the original image and the equalized one concludes that the latter shows a better contrast due to the equalization of the histogram. This process produces a “broader” histogram, an indicator of an image with more variety of gray-level intensity pixels.

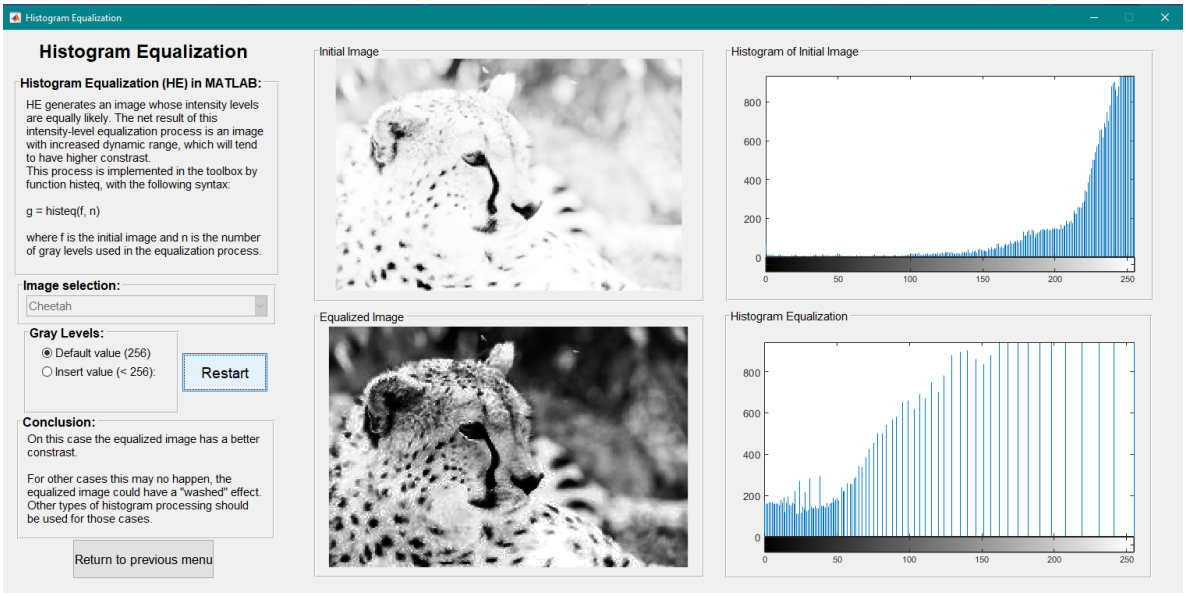

Figure 1.16. Histogram Equalization GUI, "Cheetah" example.

Another example with a “Boat” image demonstrates how the equalization process works. The equalized image presents more contrast by enhancing the darkest and lightest areas. This example is presented in the following figure.

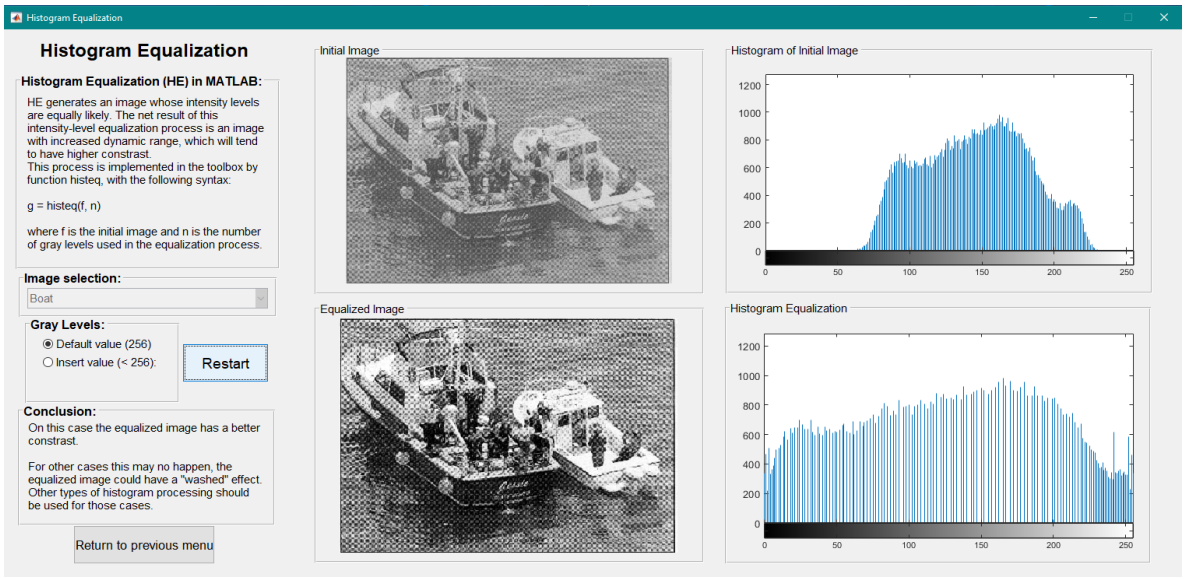

Figure 1.17. Histogram Equalization, "Boat" example.

1.3.3.1.2. Contrast-limited adaptative histogram equalization (CLAHE)

This technique consists of processing small regions of the image, called tiles, using histogram specifications for each tile individually. Contrast, especially in areas of homogeneous intensity, can be limited to avoid amplifying noise. The user must enter a value for the “Clip Limit” parameter, which specifies a contrast enhancement limit. Higher numbers result in greater contrast; however, it can produce an overexposed or “washed out” appearance. Figure 1.18 presents an example for this demo using the “Ben” image with a user-entered value of 0.03 for Clip Limit.

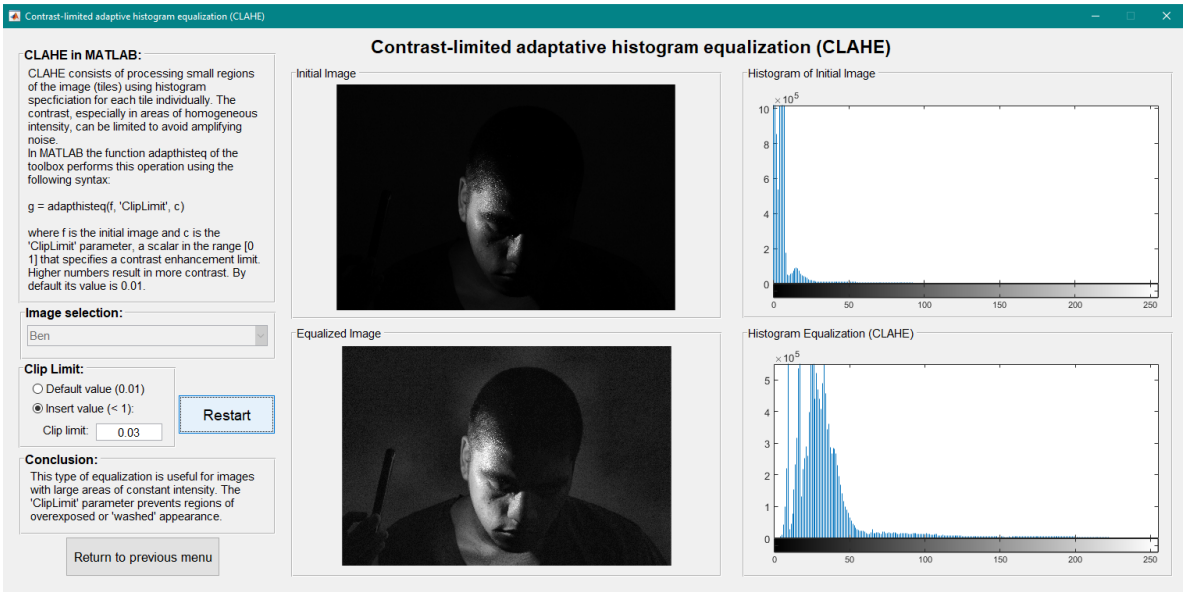

Figure 1.18. CLAHE GUI, "Ben" example.

Another example with the “Kid” image is shown in Figure 1.19 using the default value for Clip Limit. The enhancement is slightly softer in comparison with the first example of the “Ben” image because the Clip Limit is 0.01 by default.

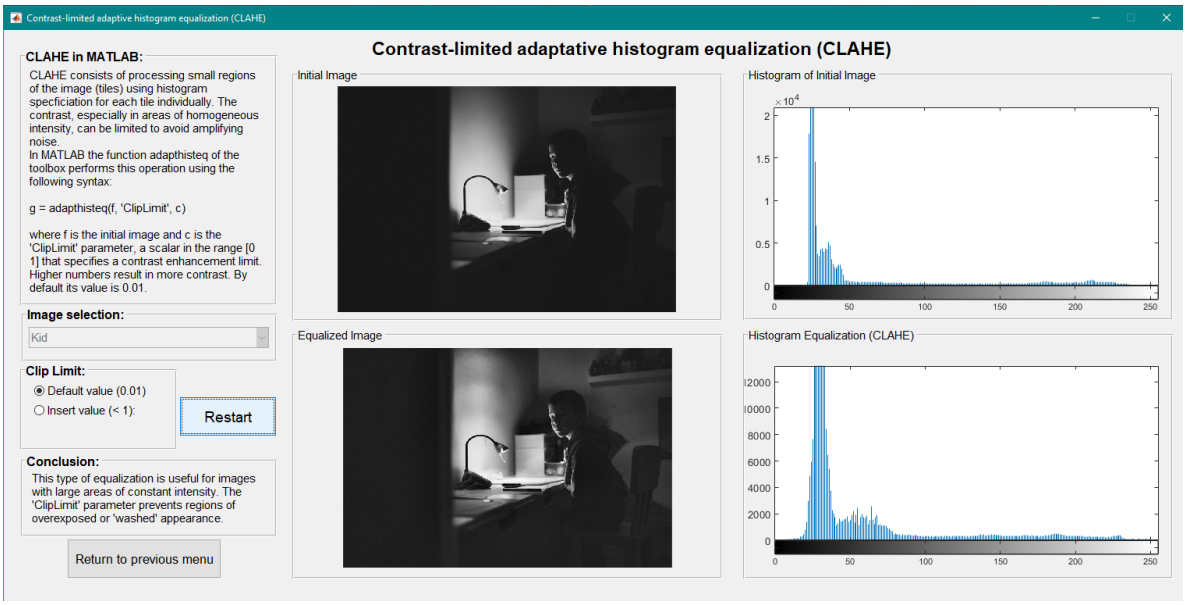

Figure 1.19. CLAHE GUI, "Kid" example.

1.3.3.2. Spatial Filtering

1.3.3.2.1. Laplacian filter (Linear)

The spatial filtering demos use two approaches. The first is a linear filter, specifically a Laplacian filter. This is a derivative operator with the ability to sharp an image by zeroing constant areas and magnifying details. Figure 1.20 presents the window GUI for this demo using the “Hands” image. The filtered version has sharper details compared to the original. A better result is presented for comparison, this image was obtained using a mask defined with a matrix. The filtered version was produced with the *fspecial* function in MATLAB.

The Laplacian filter implements the following spatial mask:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

The “better result” utilizes a slightly different spatial mask:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

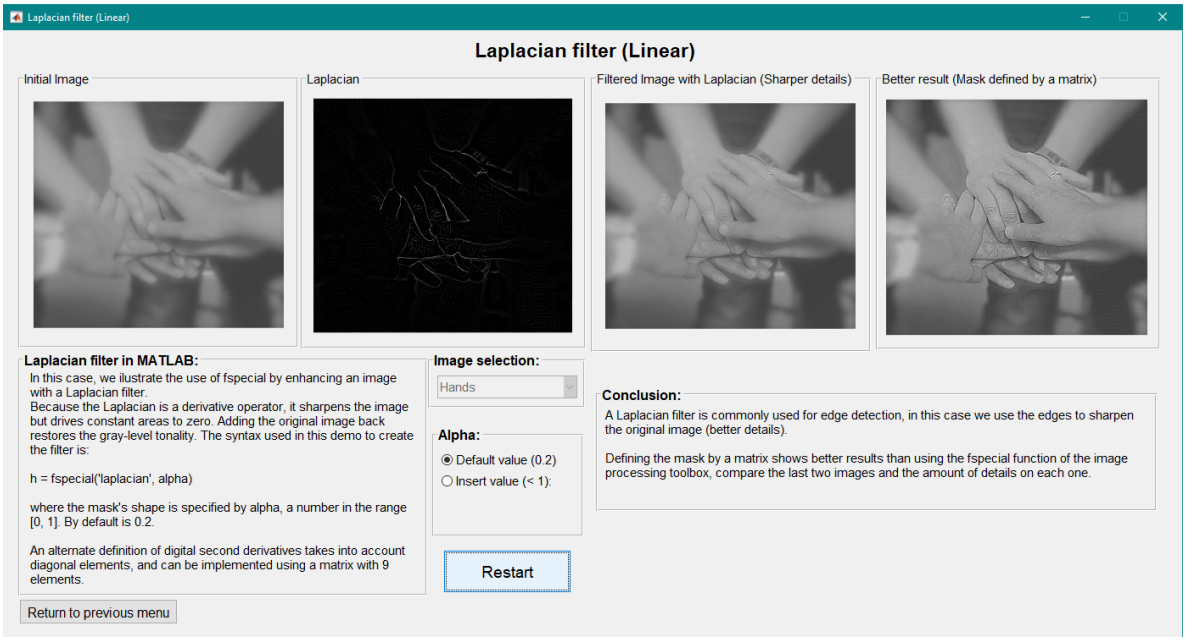

Figure 1.20. Laplacian filter GUI, "Hands" example.

A similar example is presented in Figure 1.21 using a “Van Gogh” image with multiple strikes that are highlighted as the spatial filtering increases.

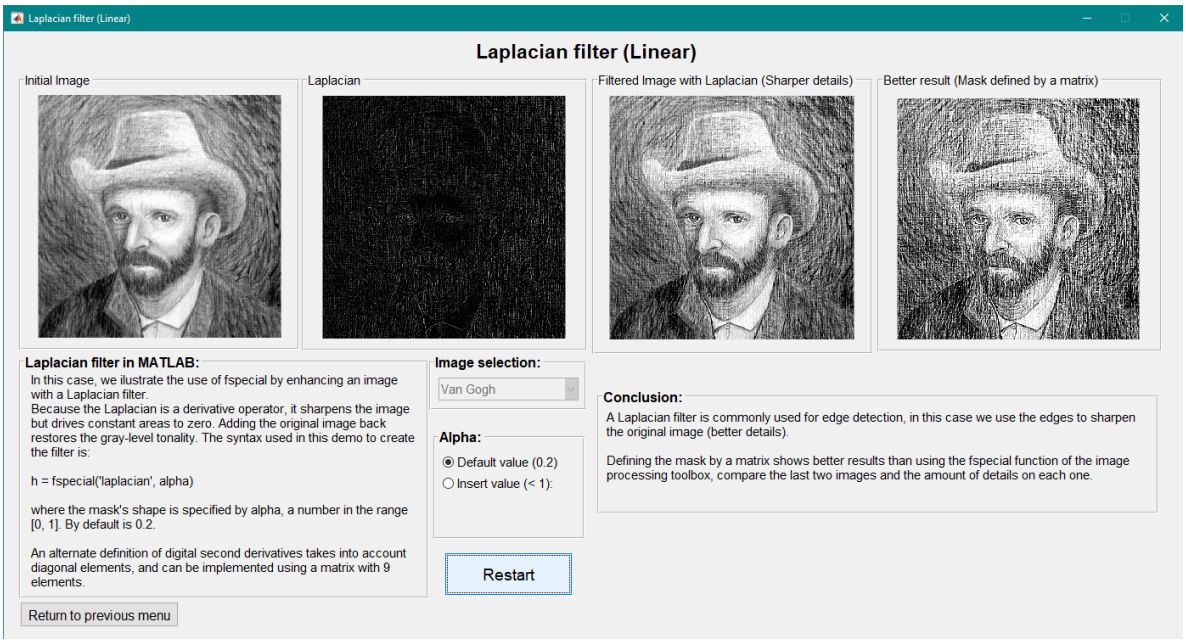

Figure 1.21. Laplacian filter GUI, "Van Gogh" example.

1.3.3.2.2. Median filter (Non-linear)

The second approach in the spatial filtering module is a non-linear technique using a median filter. This type of filter has practical importance in MATLAB, for this reason, it has its function called *medfilt2*. In this demo, the user can modify the noise density added to the image. Figure 1.22 presents an example of this demo with the “Tropical” image. A comparison between the two filtered images allows us to understand the importance of padding the border of the image.

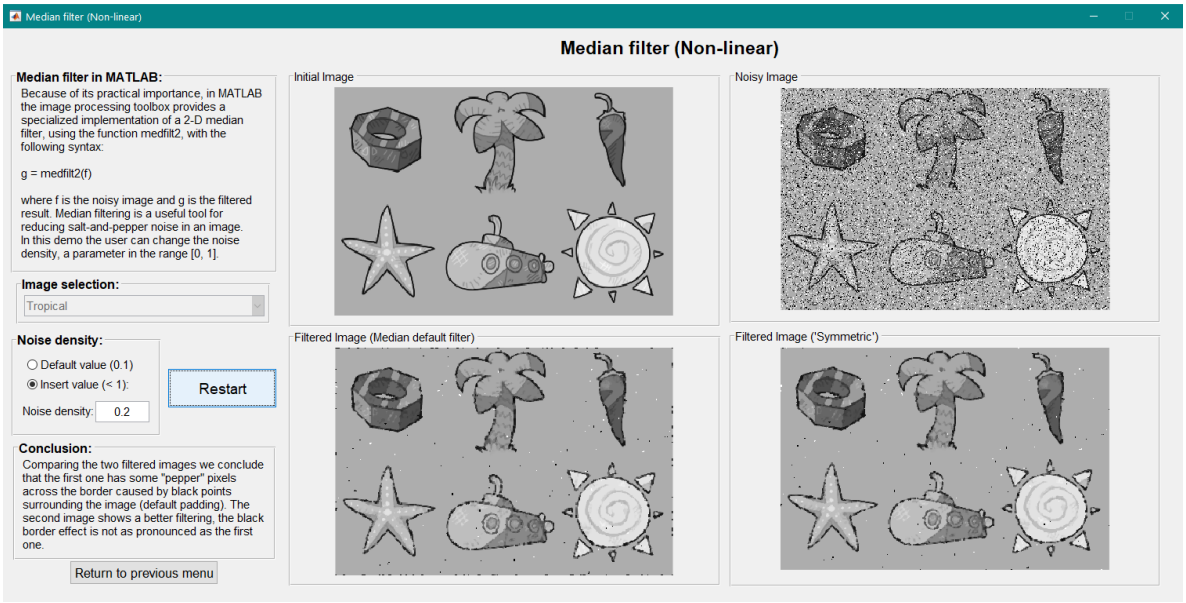

Figure 1.22. Median filter GUI, "Tropical" example.

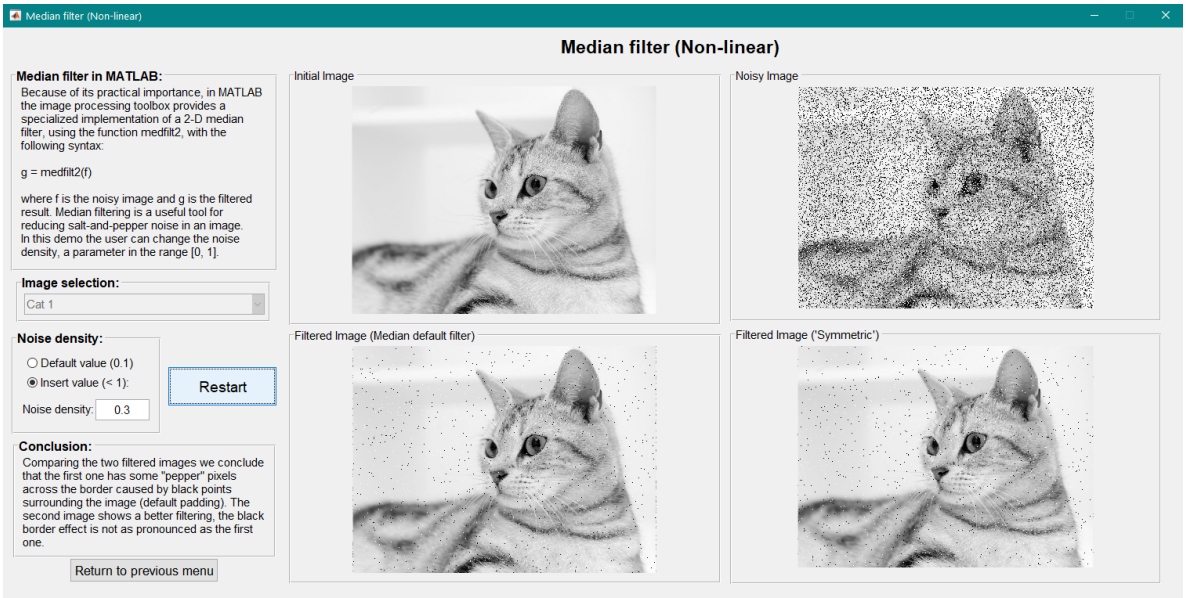

Figure 1.23. Median filter GUI, "Cat 1" example.

Several “pepper” pixels are still present in the filtered result for the first case, in the second image this problem is avoided. A symmetric padding was configured for the second case, this procedure handles the border problems inherent in spatial filtering. The “symmetric” approach extends the image by mirror-reflecting it across the border [1].

Figure 1.23 presents another example with a higher noise density, which results in a noisier image. However, the median filter is capable of recovering the initial image, which in this case is called “Cat 1”. The median filter is considered non-linear because its response is based on ordering (ranking) the pixels contained in an image neighborhood and then replacing the value of the center pixel in the neighborhood with the value determined by the ranking result [1].

1.3.3.3. Fuzzy Techniques

1.3.3.3.1. Contrast Enhancement

The fuzzy techniques used in this demo provide better results for some images compared to the histogram equalization technique. This demo presents a comparison between these two approaches. The fuzzy variables are the intensities of the pixels, which are handled by membership functions that create darker and brighter intensities, increasing the separation of dark and light on the grayscale. Figure 1.24 shows an example (“Street” image) of contrast enhancement using fuzzy techniques, the middle image shows a washed appearance, however, the Fuzzy Logic Image has a better appearance while enhancing contrast at the same time. This image has rich gray tonality, which is desirable instead of an “overexposed” appearance such as the regular histogram equalization result.

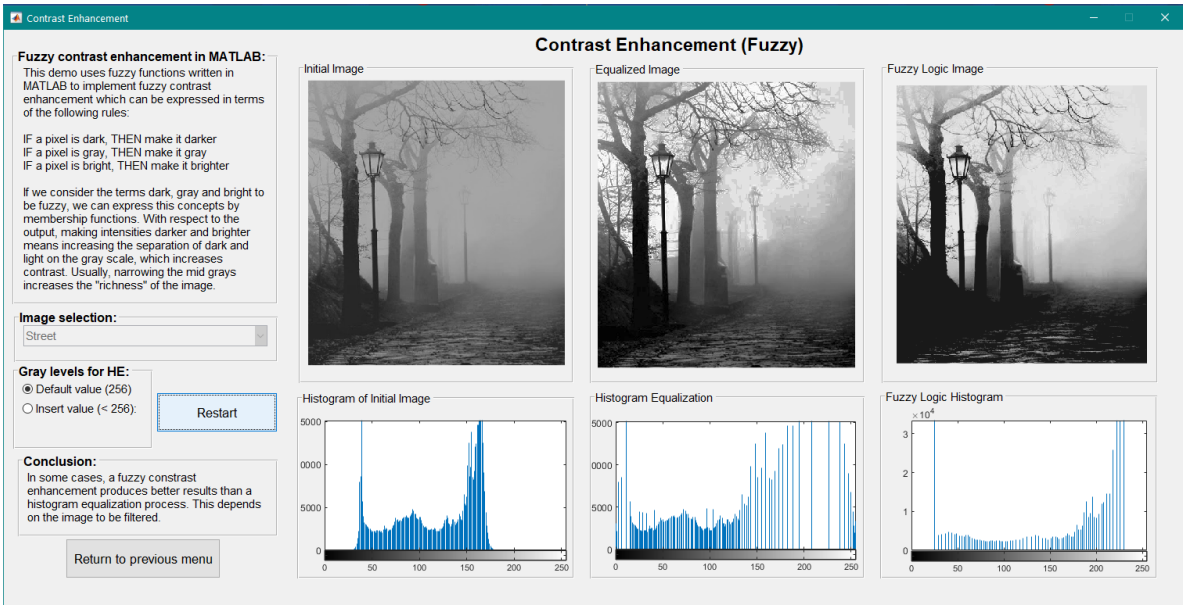

Figure 1.24. Fuzzy Contrast Enhancement GUI, "Street" example.

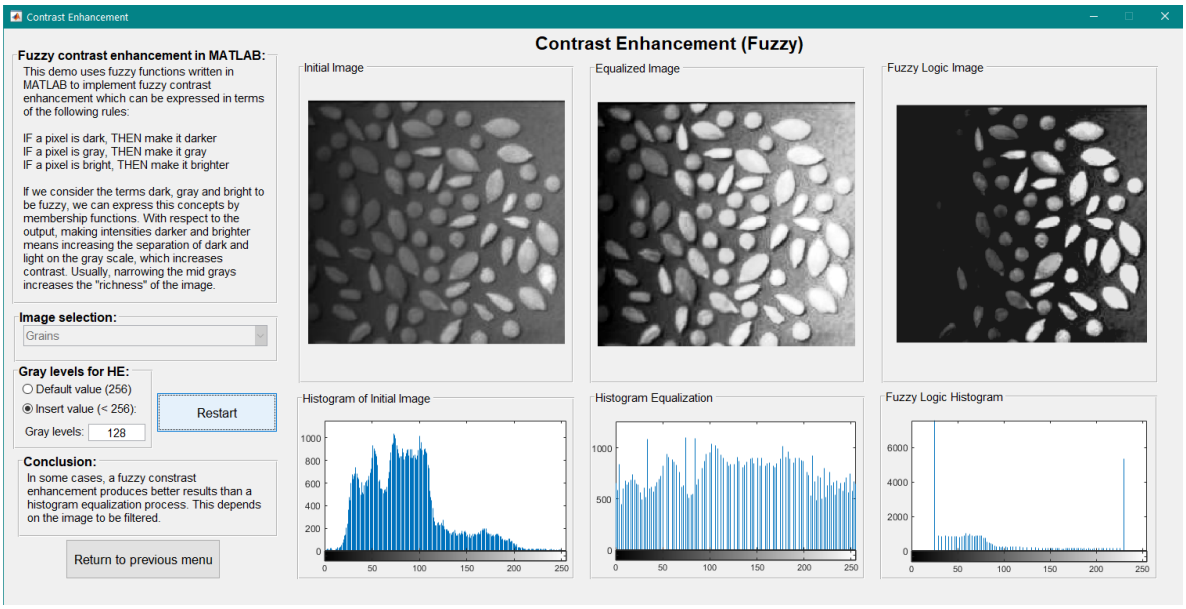

Figure 1.25. Fuzzy Contrast Enhancement GUI, "Grains" example.

The other example in this demo presents a “Grains” image in Figure 1.25, this case demonstrates a different type of use for the fuzzy contrast enhancement. For this specific image, the algorithm is not capable of producing a successful enhancement like the equalized version. However, another use for this technique is to remove foreground elements (such as some dark grains) that have similar gray levels to the background, a different approach to image segmentation.

1.3.3.3.2. Edge Detection

An interesting application of fuzzy techniques is edge detection. This method is not used as often as the Laplacian, because its results are dependent on fuzzy membership functions, which can change for different images. However, in this demo, the images are predefined meaning that the performance of the technique is not affected by the different images. When using fuzzy techniques, the basic approach is to define fuzzy neighborhood properties based on uniform regions and intensity differences. The following figures present examples of this demo using the “Tropical” and “Leaf” images, respectively.

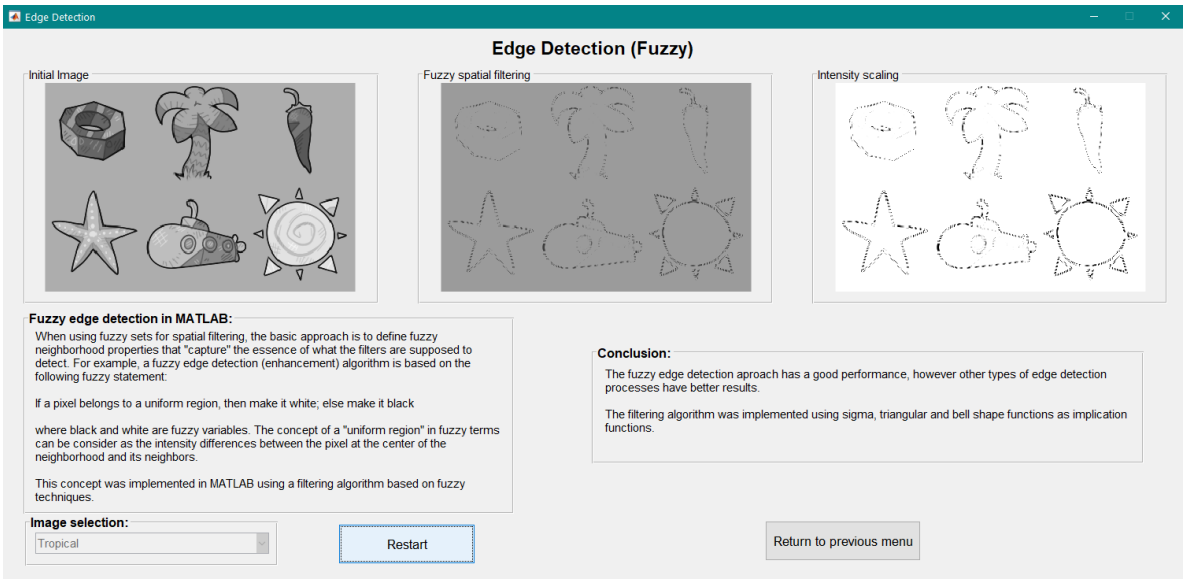

Figure 1.26. Fuzzy Edge Detection GUI, "Tropical" example.

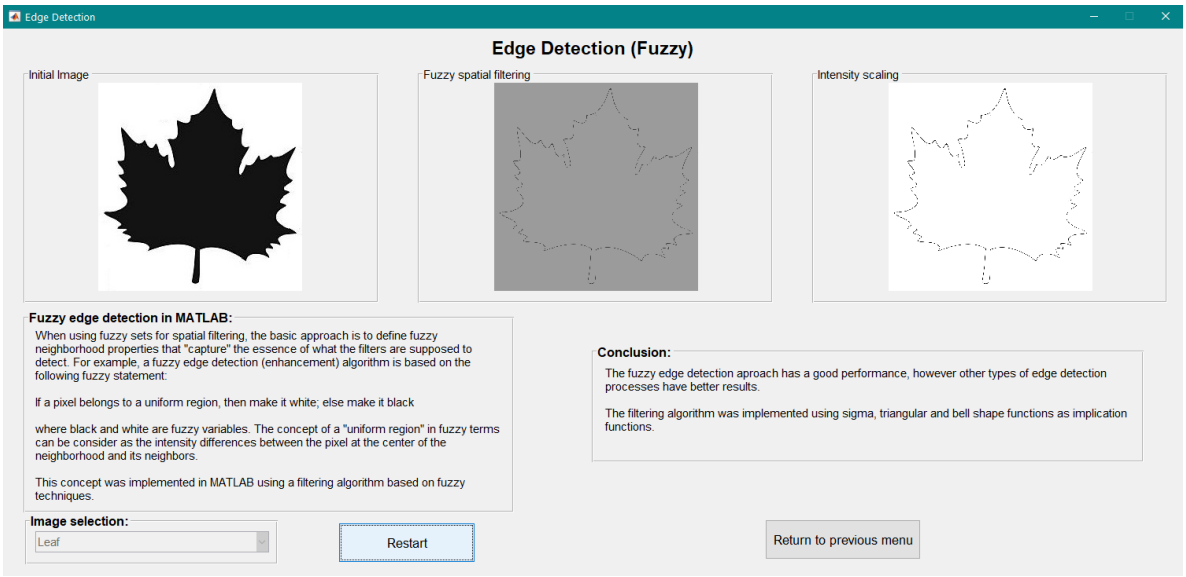

Figure 1.27. Fuzzy Edge Detection GUI, "Leaf" example.

1.3.3.4. Frequency Filtering

1.3.3.4.1. Filters in the Frequency Domain

This demo implements filter design directly in the frequency domain. The main parameters include the spectral distance “D0” a positive number that specifies the diameter of the passband, for low pass filters; or reject band, for high pass filters. Each example features two initial images, one for low pass filtering which blurs the details, and a high pass filter that extracts the high-frequency details (basically made up by edges). The following figures show two examples (“Elephants” and “Dog”) of the general operation of this demo using the default distance value of 50 pixels.

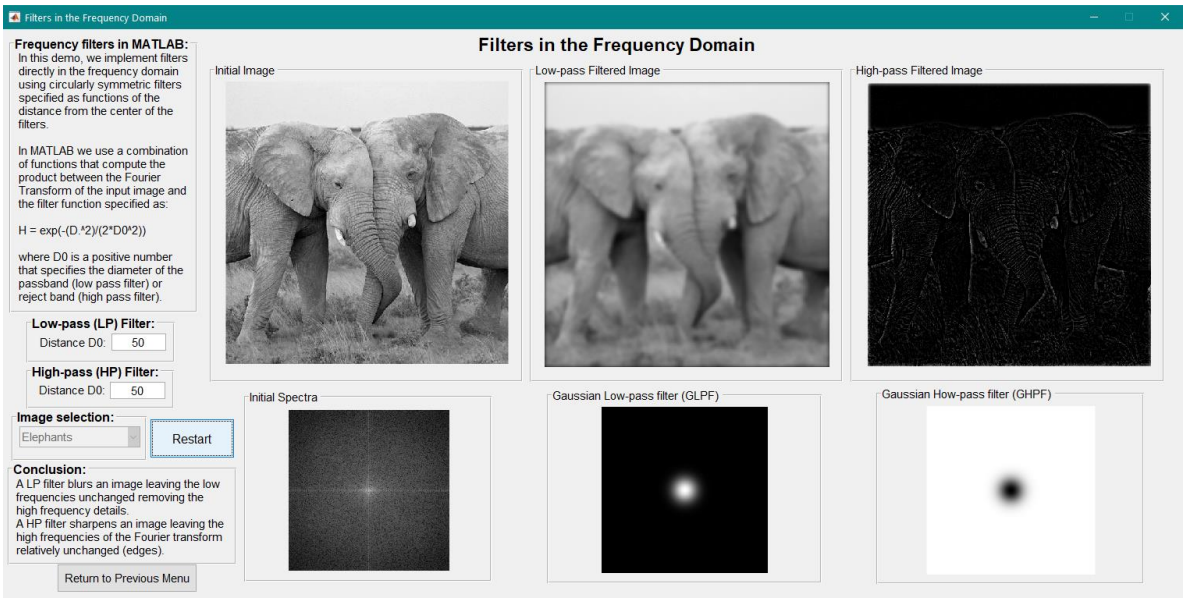

Figure 1.28. Filters in the Frequency Domain GUI, "Elephants" example.

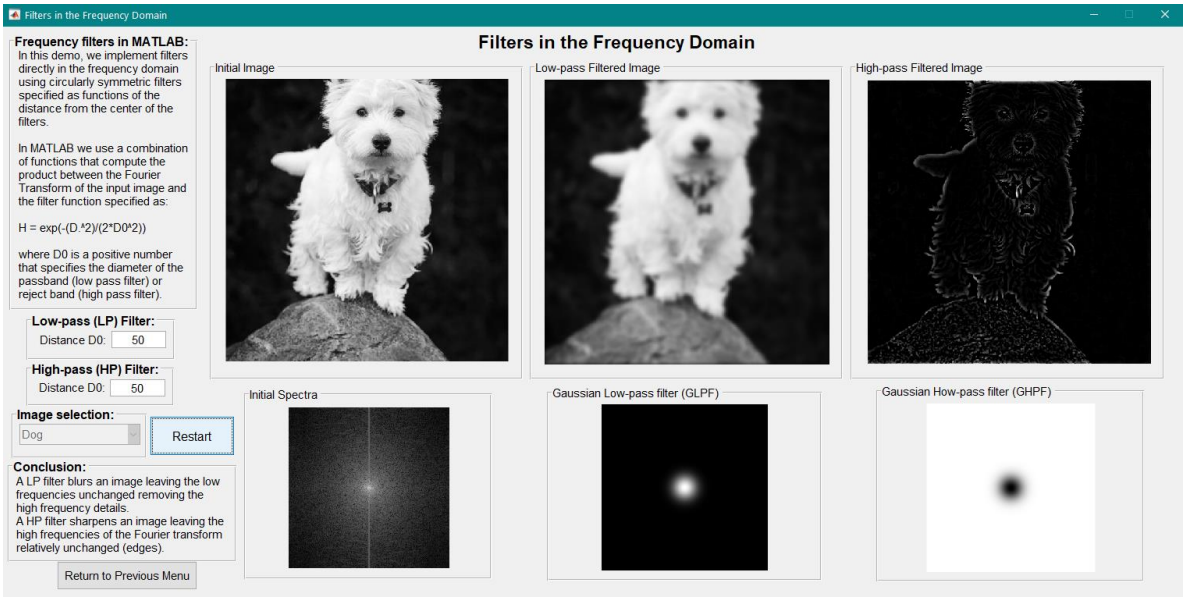

Figure 1.29. Filters in the Frequency Domain GUI, "Dog" example.

1.3.3.4.2. Notch Filters (Moire Pattern)

A Moire Pattern is a periodic noise that is added to an image in a scanning process generally of newspapers. This demo shows an example of how to eliminate this noise using notch filters, the user has the option to modify the radius of the “notches” and analyze the effect that these filters have on the general appearance of the recovered image. The notch filters have gaussian shapes. Figure 1.30 presents an example using the “Lanterns” image with the default value for the notch radius of 20 pixels.

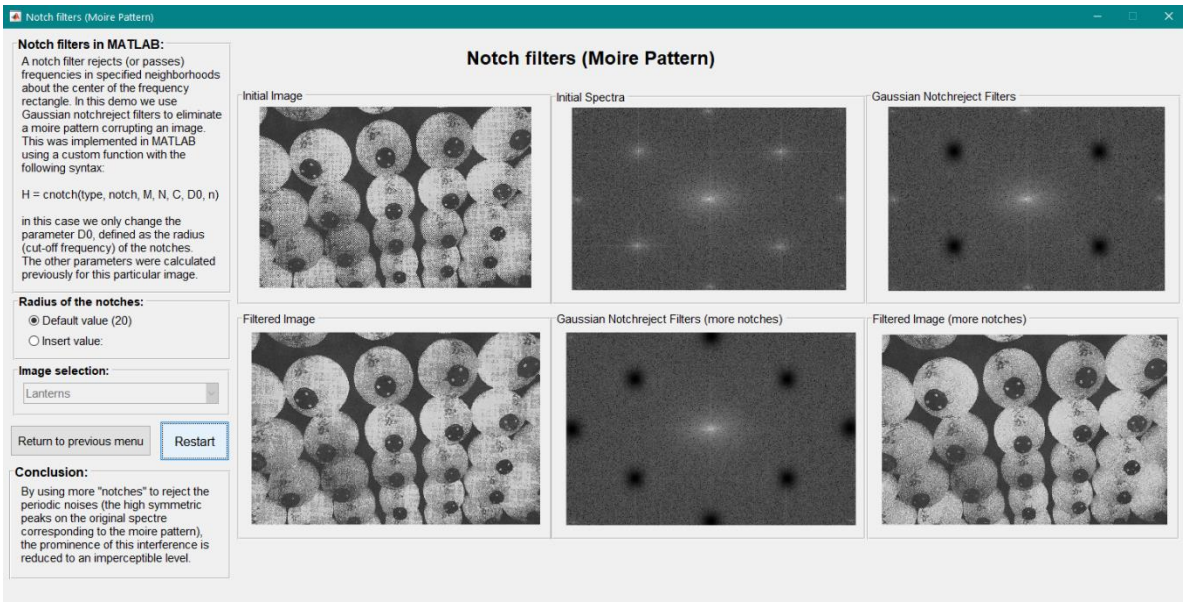

Figure 1.30. Notch Filters GUI, "Lanterns" example.

This demo compares two filtered results using a smaller and a larger number of notches. In general, by using more notches, the recovered image will look better as can be seen in the previous figure, and the example of the “Asian girl” image in the following figure.

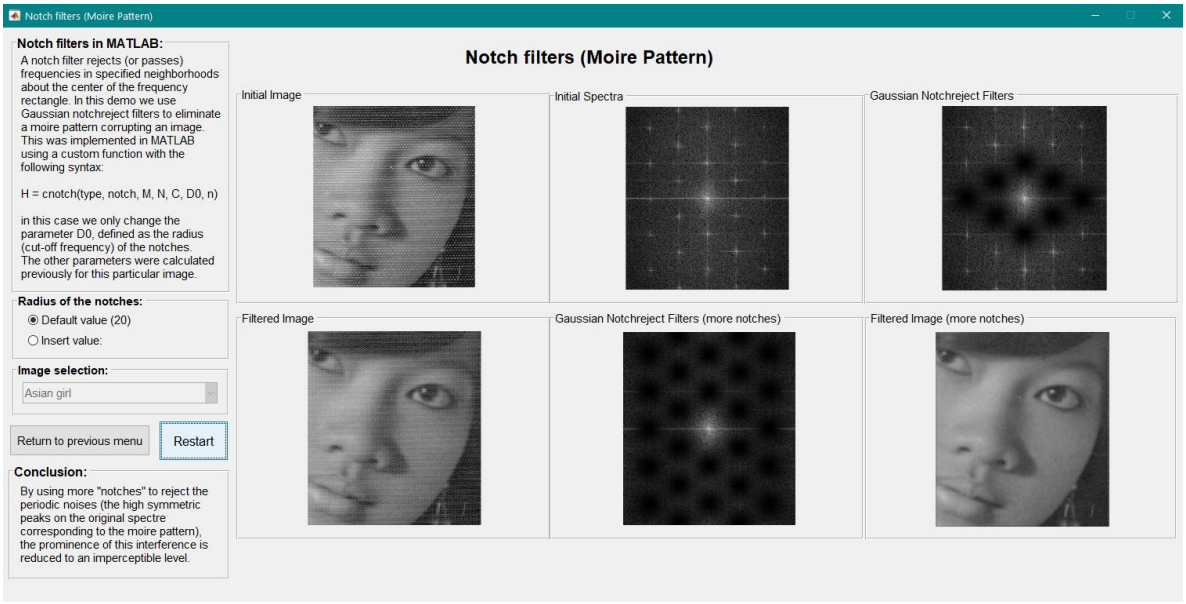

Figure 1.31. Notch Filters GUI, "Asian girl" example.

1.3.4. Image Restoration Module

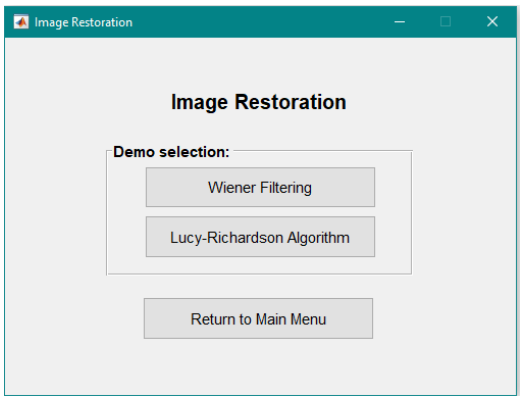

Figure 1.32. Image Restoration GUI.

The application implements two demos for image restoration. Figure 1.32 presents both options, including the Wiener Filtering and Lucy-Richardson Algorithm, two well-known algorithms used in image restoration.

1.3.4.1. Wiener Filtering

Figure 1.33 shows a comparison between the initial image and its degraded version, in this case for the “Checkerboard” image. The user controls the linear motion used to create a PSF (Point Spread Function) that corrupts the original image and the Gaussian noise that finally produces the degraded version. This demo compares several restoration techniques including Inverse filtering, Wiener filtering using a constant ratio, and Wiener filtering with autocorrelation functions.

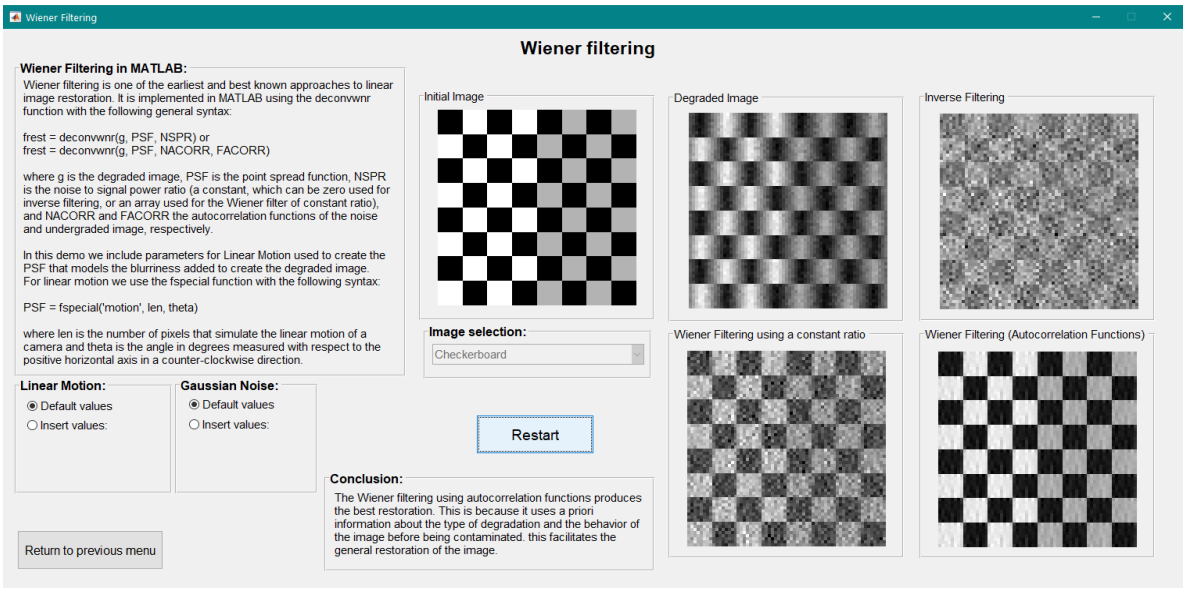

Figure 1.33. Wiener Filtering GUI, "Checkerboard" example.

This demo is useful for analyzing noise induced by linear motion where the user modifies the number of pixels translated and the angle at which the motion occurs. The following figure presents an example of this type of degradation in the “Star” image with the restored versions concluding that the autocorrelation functions approach produces the best results.

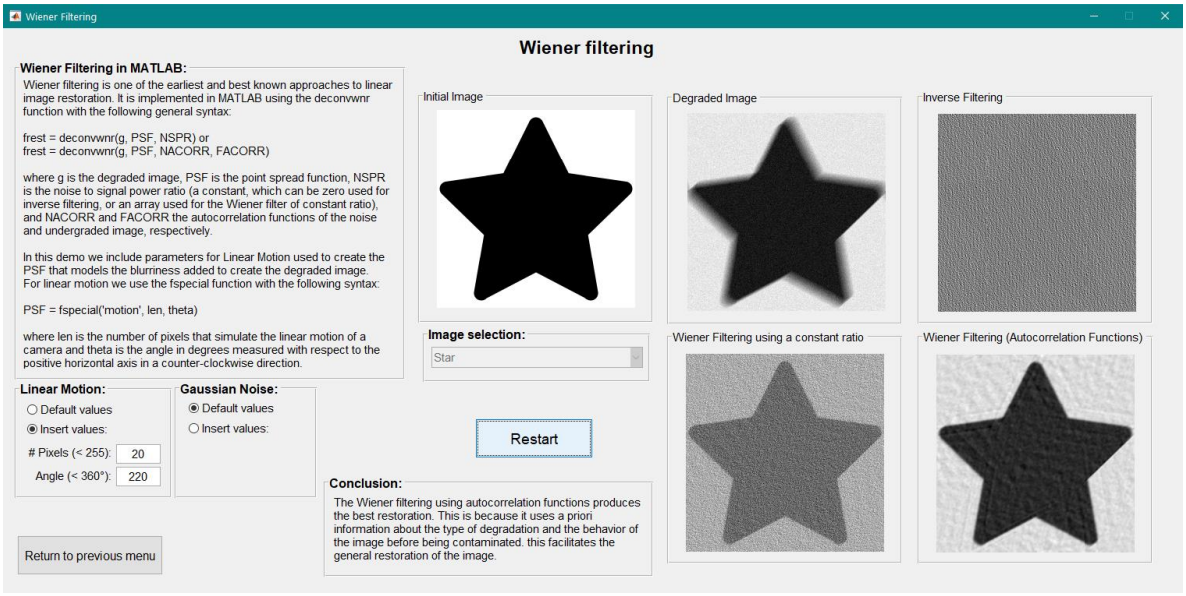

Figure 1.34. Wiener Filtering GUI, "Star" example.

1.3.4.2. Lucy-Richardson Algorithm

The Lucy-Richardson (L-R) algorithm is a nonlinear and iterative restoration tool. This demo provides better results than a linear method, however being an iterative algorithm, it is necessary to define the number of iterations. This decision depends on several outputs (trial and error) accepting the best result. Figure 1.35 presents the restoration results using the same “Checkerboard” image. For this specific example, 10 iterations of the algorithm are sufficient to produce an acceptable restoration.

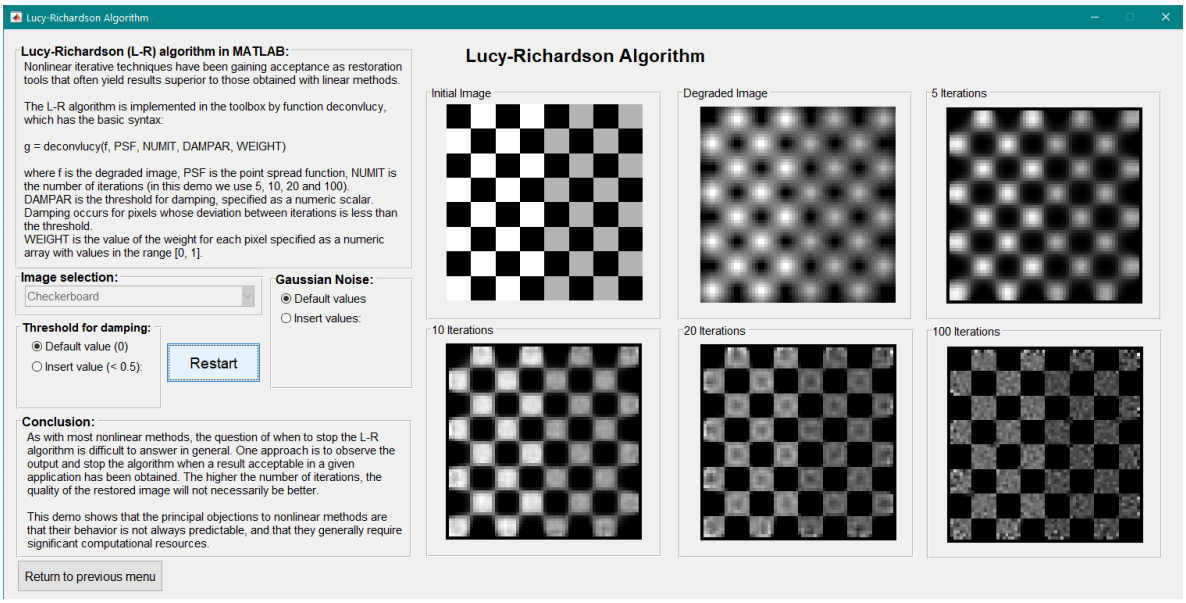

Figure 1.35. Lucy-Richardson Algorithm GUI, "Checkerboard" example.

Another example using the “Leaf” image demonstrates the variability of restored versions based on the number of iterations. For this case, the best-restored image corresponds to 5 iterations because the dark area of the leaf is not corrupted by gray pixels like the other restored versions with a higher number of iterations. This example is presented in Figure 1.36.

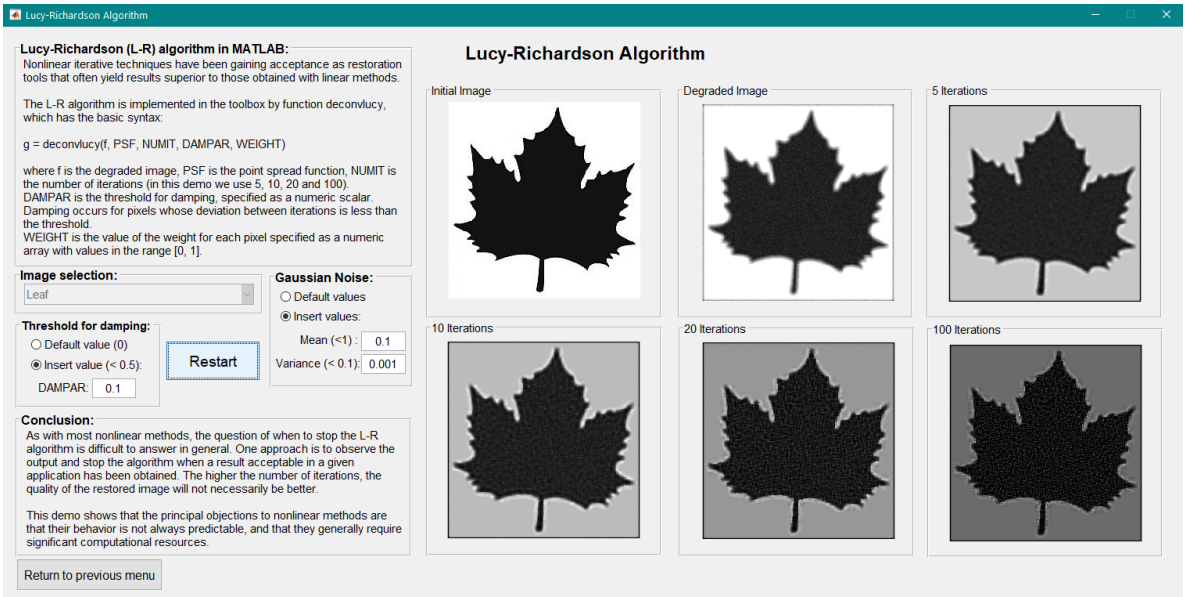

Figure 1.36. Lucy-Richardson Algorithm GUI, "Leaf" example.

1.3.5. Image Compression Module

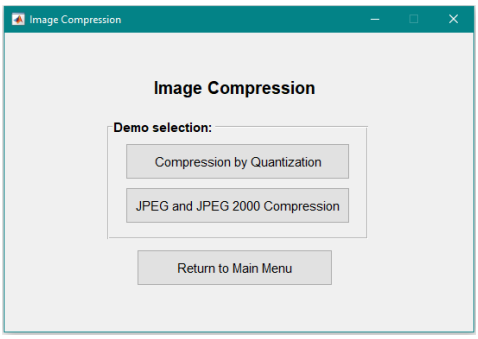

Figure 1.37. Image Compression GUI.

For image compression, the application offers two demos. Figure 1.37 presents both options, including Compression by Quantization and, JPEG and JPEG 2000 Compression, the latter being a comparison between the most known compression standard for images.

1.3.5.1. Compression by quantization

An image has psychovisual redundancy and it is desirable to reduce this information as it is not essential for normal visual processing. This removal is called quantization because it creates a reduced number of discrete levels for gray level intensities in an image, this process is not reversible. This demo shows an example of this procedure using portrait images and their quantized versions for comparison purposes. Figure 1.38 shows the operation for the “Marilyn Monroe” image with 2 bits used in compression by two methods: Uniform quantization and IGS (Improved Gray Scale) quantization. The original image has 8-bit intensity levels. The difference between the quality of the compressed images is remarkable.

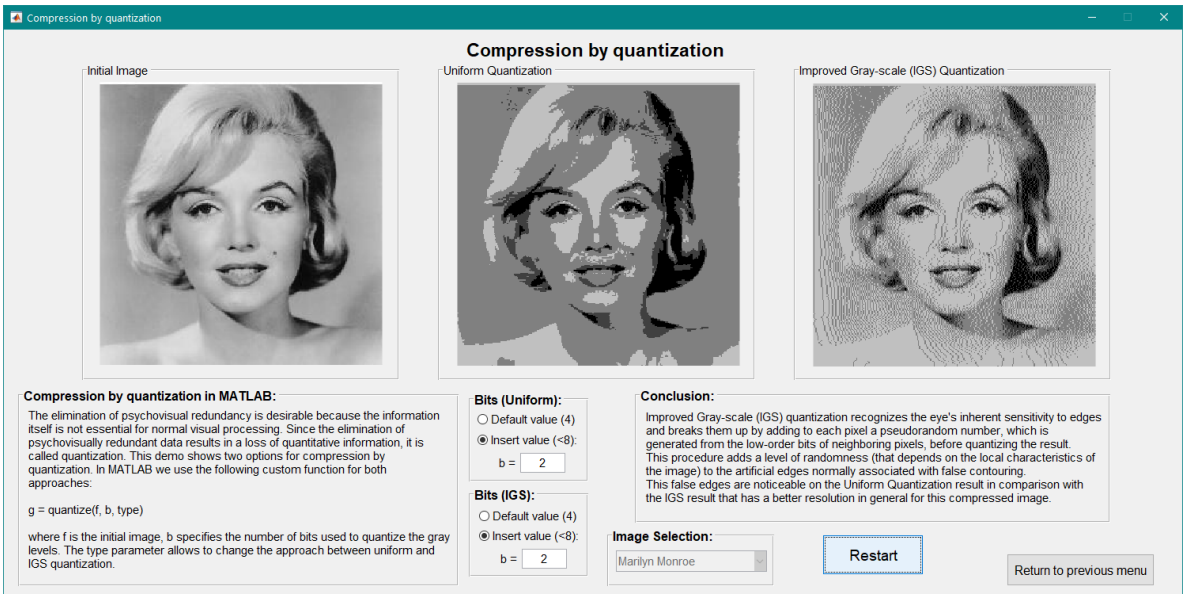

Figure 1.38. Compression by quantization GUI, “Marilyn Monroe” example.

The following figure shows another example with the image called “Audrey Hepburn” with 2 bits. This demo confirms the advantage of using the IGS approach, as it recognizes the eye’s interest sensitivity to edges and divides them by adding a pseudorandom number to each pixel, generated from the lower-order bits of neighboring pixels. False edges are more noticeable in the uniform quantization result, as can be seen in Figures 1.38 and 1.39.

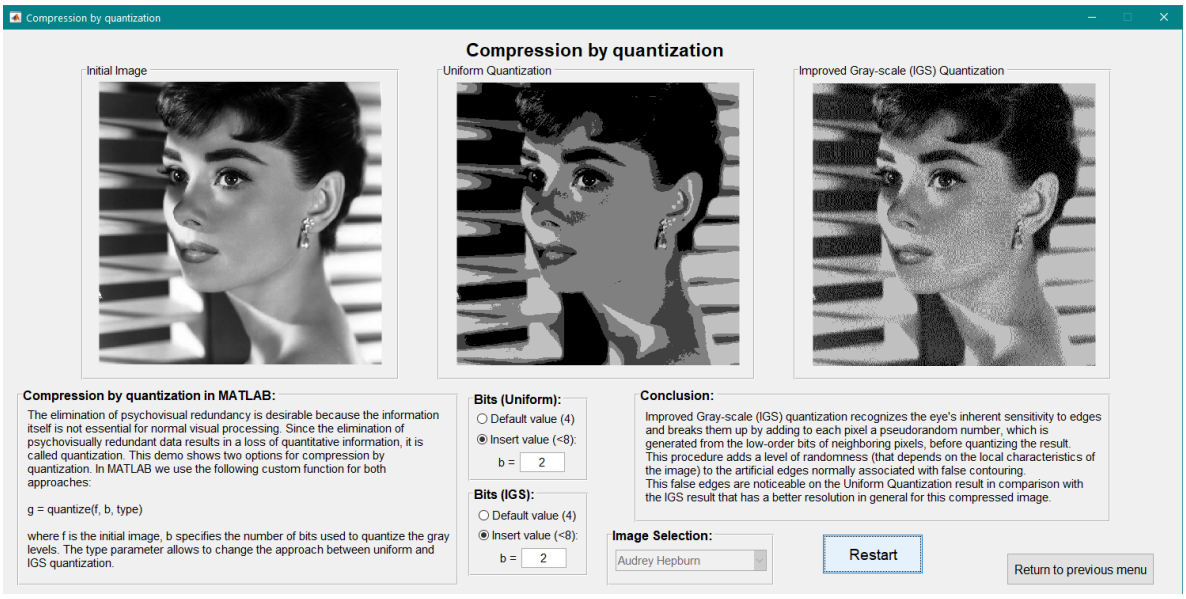

Figure 1.39. Compression by quantization GUI, “Audrey Hepburn” example.

1.3.5.2. JPEG and JPEG 2000 Compression

Joint Photographic Experts Group (JPEG) is the most popular compression standard used in multiple files including images. This demo applies a version of the original JPEG compression with the difference of not including Huffman coding. The demo features a portrait and its compressed version using both JPEG standards. This demo provides a comparison between both results and the improvement of JPEG 2000 over the original due to the implementation of the Wavelet transform in this method. Figures 1.40 and 1.41 show the operation of this demo for the “Boy” and “Dog” images, respectively. The differences between the two approaches are more noticeable in the Zoom for Details images (square regions in the JPEG version). The parameters have a direct impact on the quality of the compressed image.

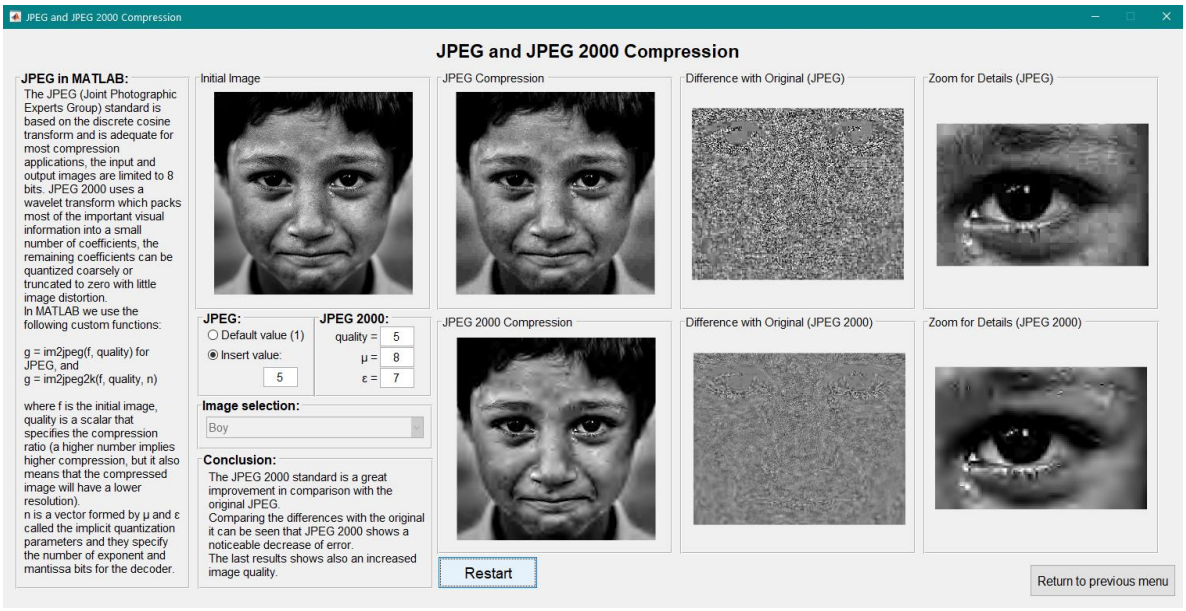

Figure 1.40. JPEG and JPEG 2000 Compression GUI, "Boy" example.

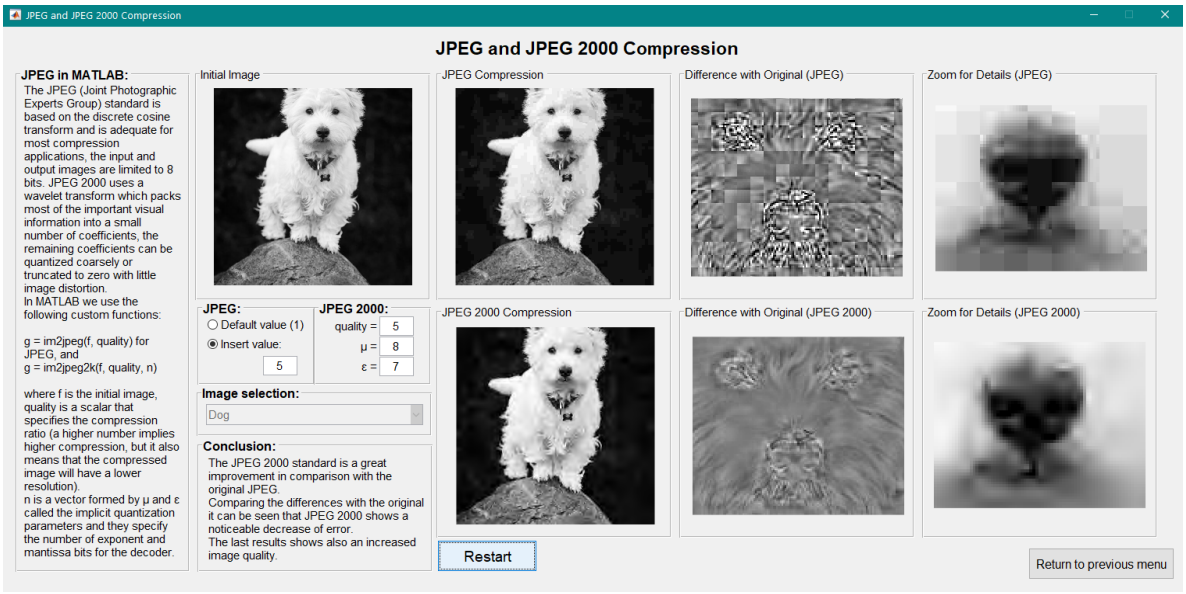

Figure 1.41. JPEG and JPEG Compression GUI, "Dog" example.

1.3.6. Morphological Image Processing Module

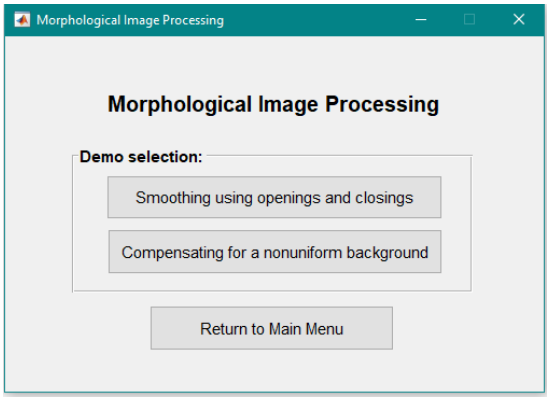

Figure 1.42. Morphological Image Processing GUI.

The morphological image processing module contains two demos. Figure 1.42 presents both options, including Smoothing using openings and closings and Compensating for a nonuniform background. Both approaches are common examples of methods used in morphological image processing, such as openings and closings. The combination of these methods provides important tools for IP.

1.3.6.1. Smoothing using openings and closings

This demo explains the definition of openings and closings and applies these techniques to smooth dark streaks in an image. The GUI provides all the steps required to perform the smoothing process. This type of processing is common for many IP

applications. Figure 1.43 shows an example using the “Plugs” image. The final processed image is free of wood grain (they appear as dark streaks on the wood dowels).

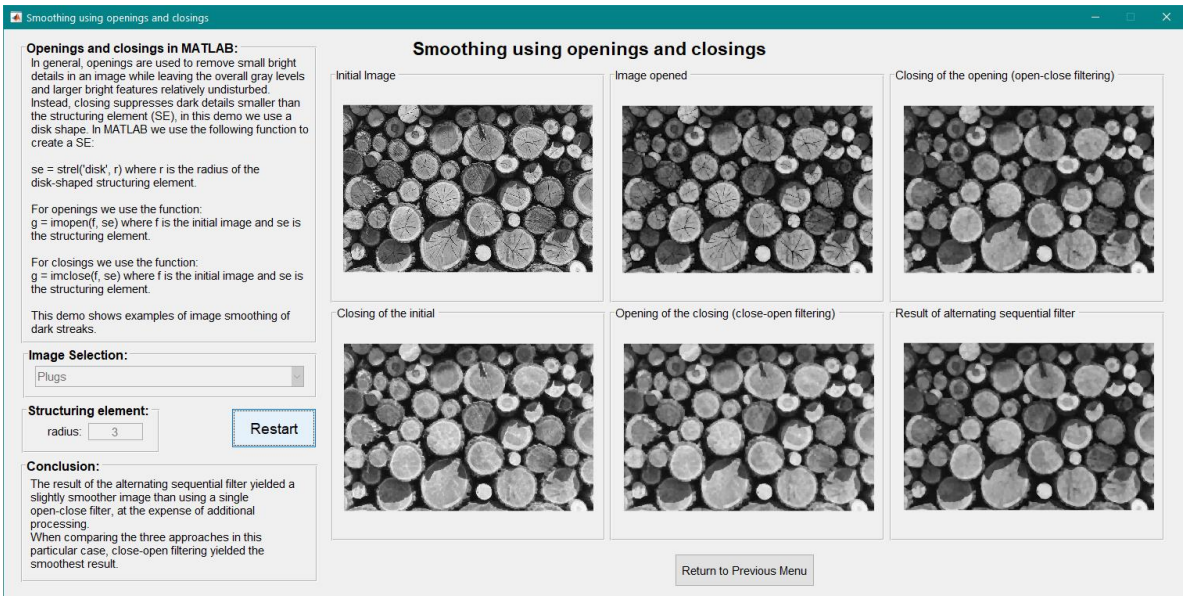

Figure 1.43. Smoothing using openings and closings GUI, "Plugs" example.

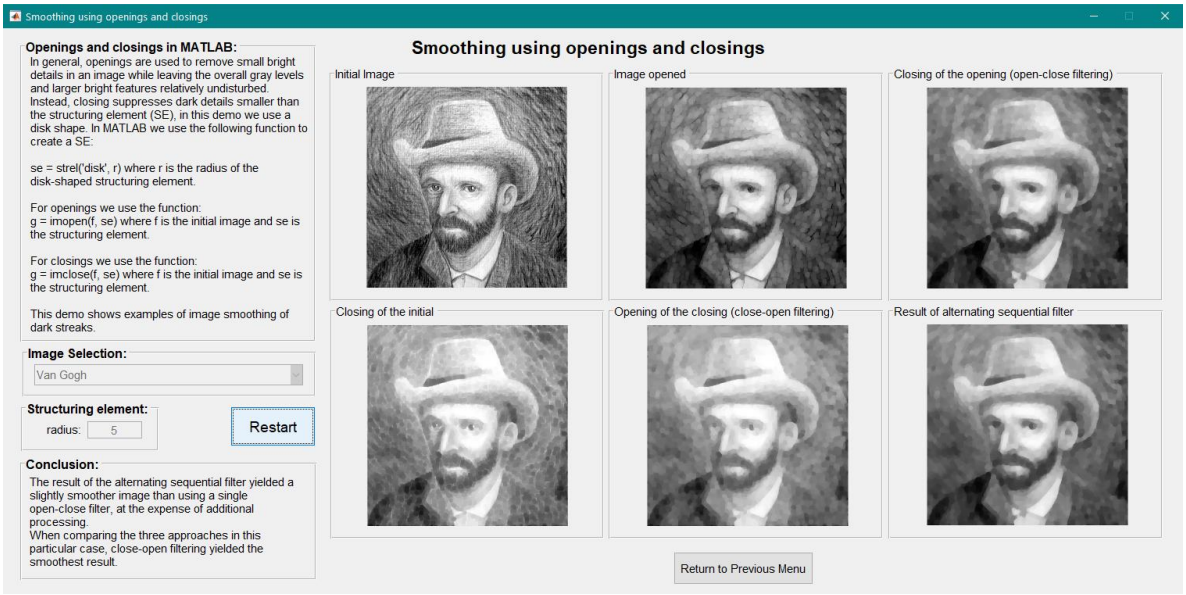

Figure 1.44. Smoothing using openings and closings GUI, "Van Gogh" example.

Figure 1.44 shows another example of this demo. The initial “Van Gogh” image features dark, hard strokes around the portrait. This image is a good example of how smoothing using morphological operators removes such strokes, as the result of alternating sequential filter demonstrates.

1.3.6.2. Compensating for a nonuniform background

Figure 1.45 shows an example (“Grains” image) of segmentation using morphological operators to compensate for a nonuniform background. This type of image needs preprocessing to obtain a suitable segmentation, the radius of the structuring element is the parameter that the user applies to change the segmented results. This compensation applies openings and closings to enhance foreground details over the background, providing a better template image for segmentation, which in this demo is a simple threshold. A morphological reconstruction removes the white areas in the image that are smaller than the structuring element, this approach improves the segmentation results.

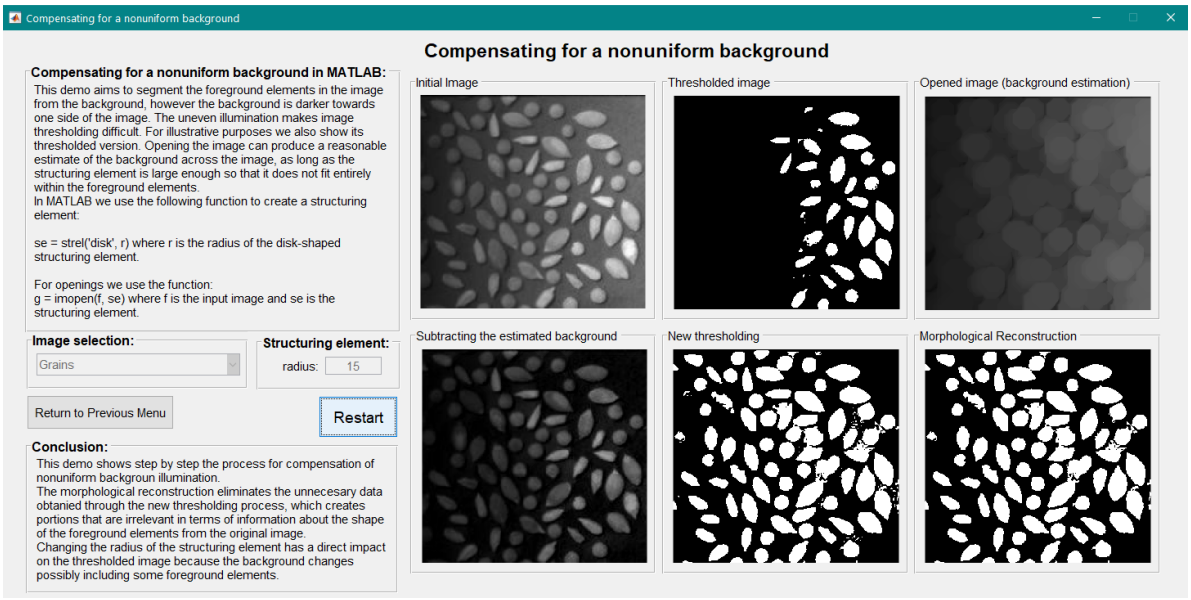

Figure 1.45. Compensating for a nonuniform background GUI, "Grains" example.

The second example in this demo utilizes the “Vessels” image that shows another case of a nonuniform background image. Figure 1.46 presents this image. This effect makes it difficult to segment the elements in the foreground from the background. This demo works by finding a background estimate and subtracting this image from the original, this operation creates a template image that can be better segmented (by thresholding in this case).

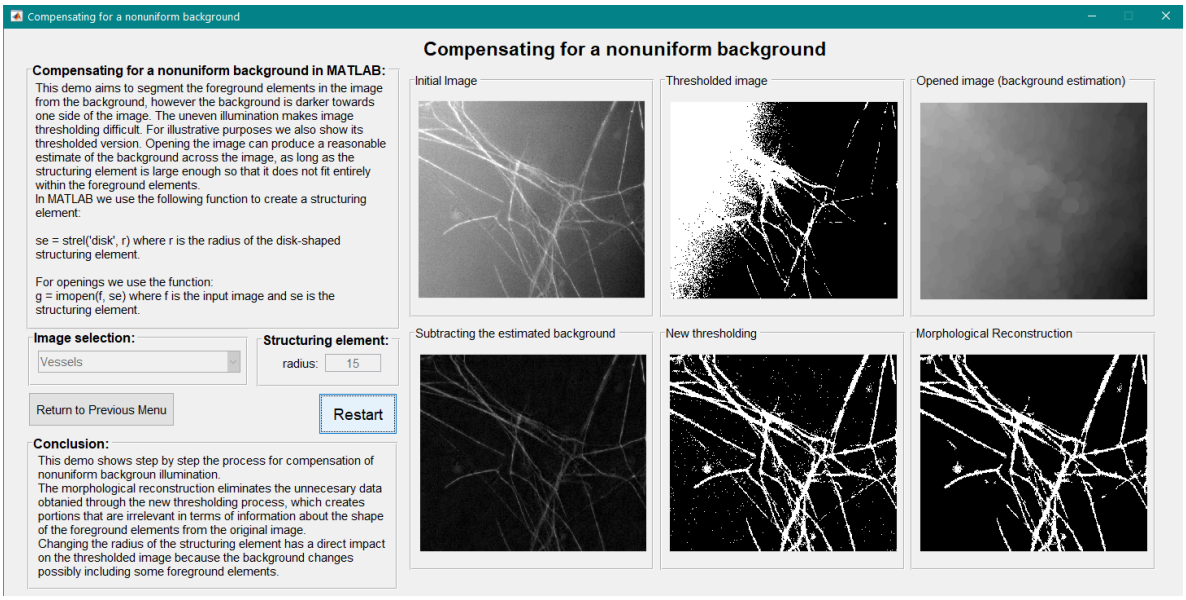

Figure 1.46. Compensating for a nonuniform background GUI, "Vessels" example.

1.3.7. Image Segmentation Module

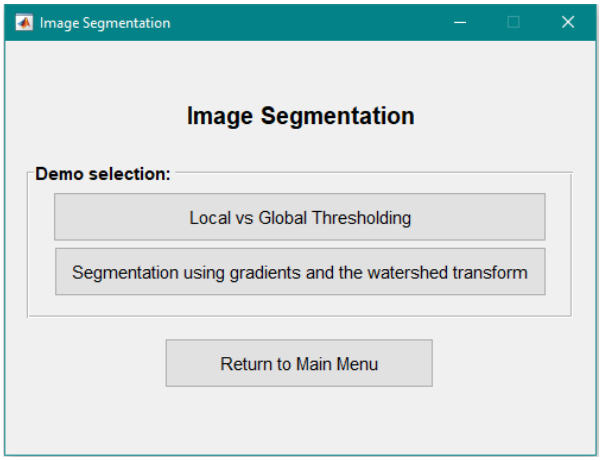

Figure 1.47. Image Segmentation GUI.

The image segmentation module offers two demos. Figure 1.47 presents both options including Local vs Global Thresholding and Segmentation using gradients and the watershed transform.

1.3.7.1. Local vs Global Thresholding

Thresholding is one of the most widely used segmentation techniques, this demo shows two approaches of this method. Global thresholding techniques often fail when the background illumination is highly uneven (as discussed in the previous section). To solve

this problem, another approach in which there is more than one dominant object intensity (in which global thresholding also presents difficulties) is to use variable or local thresholding. As the name suggests, this type of technique computes a threshold value at each point in the image based on one or more properties of the pixels in its neighborhood. The main parameters for local thresholding are nonnegative scalars that create a proportional relationship between the local mean and the standard deviation, with the threshold value indicating whether a pixel is assigned a value of 1 or 0. The user can modify these constants in this demo. Figure 1.48 presents the comparison between both thresholding approaches with the "Quantum" image. The comparison shows the advantages of using a local approach versus a global threshold based on the Otsu method. This global approach maximizes the variation between classes between two sets of intensity levels. Since the images in this demo have three differentiated intensity levels, Otsu's method fails to segment the highest intensities from their surroundings, which is not the case for local thresholding.

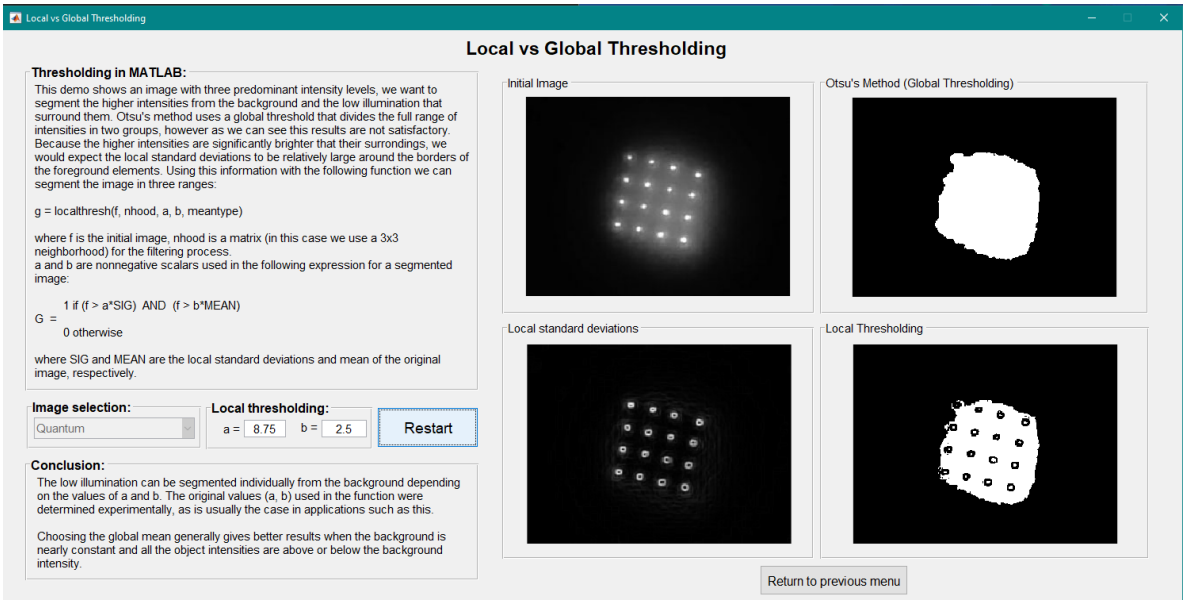

Figure 1.48. Local vs Thresholding GUI, "Quantum" example.

The following figure presents another example of this demo using the "Cells" image. In this case, the constants a and b have different values. These values were found by trial and error, which is common in applications like the one presented in this demo.

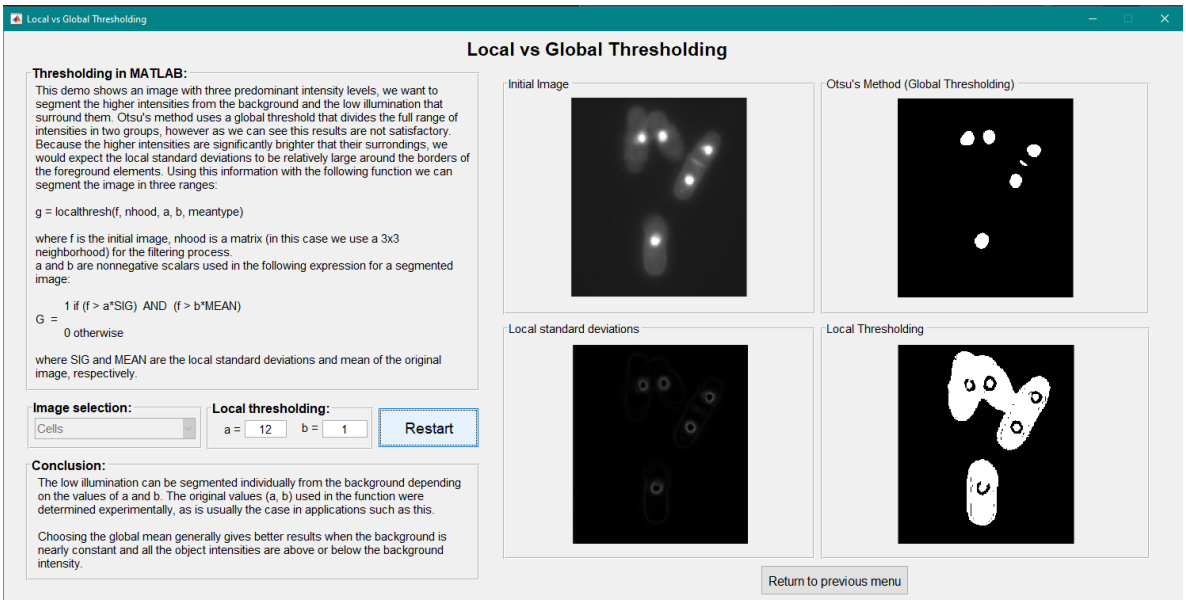

Figure 1.49. Local vs Thresholding GUI, "Cells" example.

1.3.7.2. Segmentation using gradients and the watershed transform

In geography, a watershed is a ridge that divides areas drained by different river systems. The watershed transform applies these ideas to gray-scale image processing to solve a variety of image segmentation problems [1]. This demo presents an introduction to the watershed transform and the application of gradient magnitude for segmentation. The magnitude of the gradient is often used to preprocess a gray-scale image before using the watershed transform for segmentation. The gradient magnitude image has high pixel values along object edges and low pixel values elsewhere. Then, the watershed transform would result in watershed ridge lines along the edges of the objects. It is easier to segment a gradient image than its original version, which is demonstrated in this demo. Figure 1.50 shows this demo with the “Particles 1” image using the default values for the “length” parameters of the structuring elements for opening and closing.

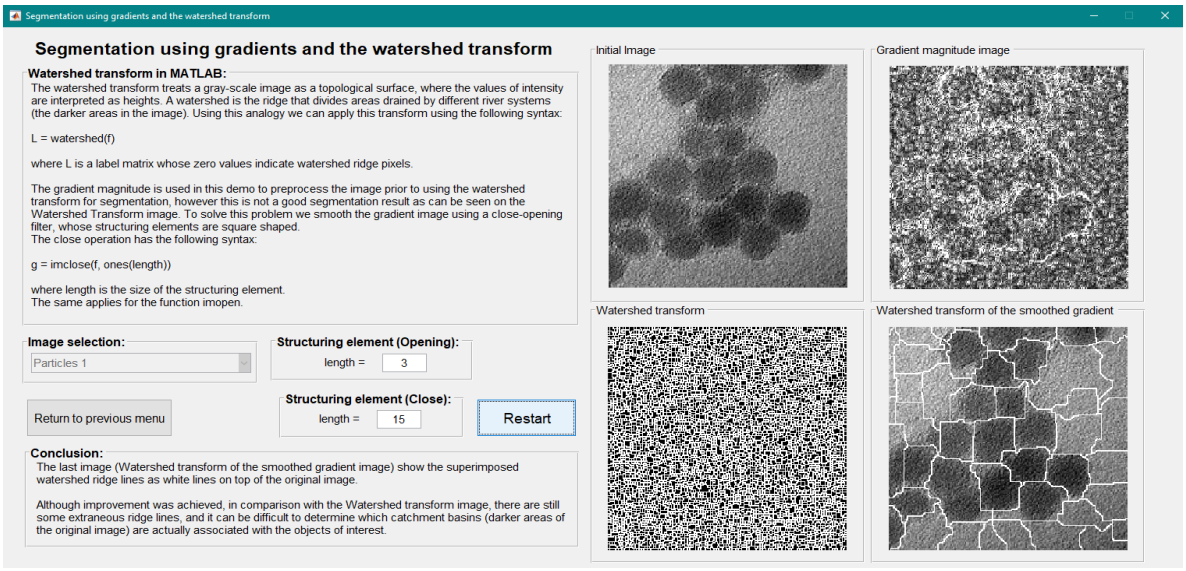

Figure 1.50. Segmentation using gradients and the watershed transform GUI, "Particles 1" example.

The Watershed transform image in the previous figure represents an example of over-segmentation. This is not a good result because there are too many watershed ridges that do not correspond to the boundaries of the objects of interest (“Particles”). To solve this problem, the gradient image was smoothed before calculating the watershed transform using a close-opening approach. The last image represents the smoothed version with some improvement, however, there are still some extraneous ridge lines. This behavior can be further refined by using a limited number of segmented regions. Another example of this demo with the “Particles 2” image is presented below.

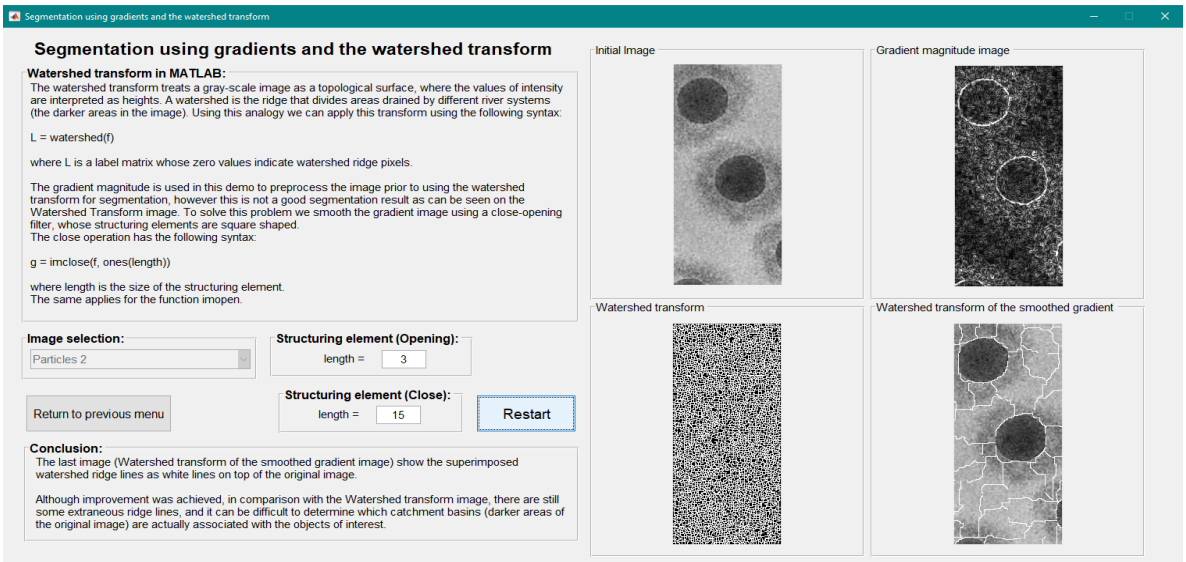

Figure 1.51. Segmentation using gradients and the watershed transform GUI, "Particles 2" example.

CHAPTER 2 . BIOMETRICS

Biometric recognition is the automatic recognition of individuals based on their biological and behavioral characteristics. These types of tools are helpful to establish the confidence that we are dealing with individuals who are already known (or not known) meaning that they belong to a group with certain rights or privileges [2].

Biometric recognition is promoted as a technology that can help to identify criminals, provide better control of access to physical facilities, patient tracking in medical informatics, personalization of social services, banking, and many more applications [2].

The main biometric characteristics used for identification are fingerprint, iris, face, voice, hand geometry, retina, handwriting, and gait [3].

A biometric characteristic requires two important attributes for unambiguous recognition: uniqueness and stability over time. The fingerprint is one of the biometrics that fulfills both requirements, the other two are the face and iris [4].

Fingerprints are accessible and they do not provide more information than necessary, such as an individual's race or health, and they perform relatively well [3].

2.1. MOTIVATION

There are many reasons to use biometrics, including improving the convenience and efficiency of routine access transactions, reducing fraud, improving public safety and national security [2].

The fingerprint has more discriminators than any other biometric feature currently in use. Fingerprints remain unchanged for life. The fingerprints of even identical twins are different [5]. One of the main reasons for the popularity of fingerprints is the relatively low price of fingerprint sensors [3]. Today our own devices have finger-imaging sensors that provide better ways to protect them from intruders, another security measure in addition to the typical password, or even face recognition.

2.2. FINGERPRINT IMAGES

Most of the early research work on fingerprint recognition was based on minutiae points (ridge ending and bifurcation) of the fingerprint which does not work well for poor quality images. Nowadays, researchers explore non-minutiae representations of fingerprints by considering fingerprint images as oriented textures that combine the global and local information present in a fingerprint [5].

Figure 2.1 presents an example of global and local features in a fingerprint image.

Figure 2.1. Global and local features of a fingerprint image *[5]*

In this work, the main focus is on non-minutiae techniques, which implies that there is no need to analyze every local feature in the fingerprint, but rather texture-based features are calculated. A complete image of a fingerprint cannot be used or processed every time because the memory required to perform that task can be large. This problem slows down the processing speed of the system for recognition. For this reason, only prominent features are extracted from each image, and a feature set is created [5].

2.3. FINGERPRINT RECOGNITION SYSTEM

The typical architecture of a fingerprint recognition system is presented in Figure 2.2. This system has two phases: enrollment and fingerprint matching. The first phase consists of an image sensor that captures the fingerprint image from which multiple features will be extracted, processed, and stored in a feature set or database. This process is repeated for the second phase by creating a query template that is used in the matching process to determine a similarity score between the two fingerprints. A decision is made based on the corresponding score allowing access to an enrolled subject or denying access to an intruder [5].

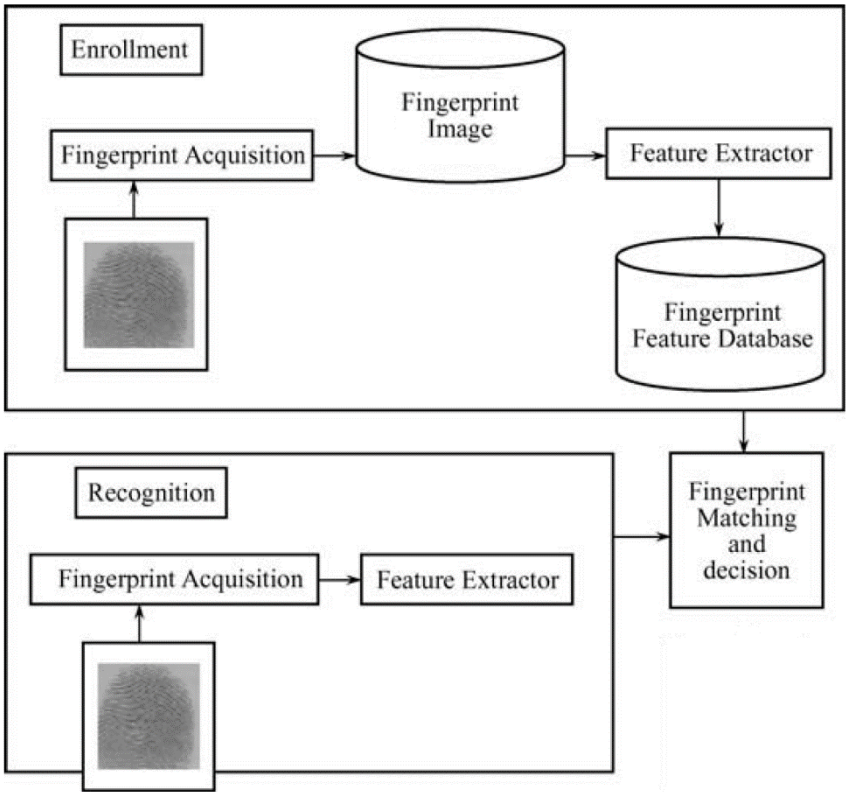

Figure 2.2. Typical architecture of a fingerprint recognition system

The current work uses public fingerprint databases that provide the images avoiding the need to find a sensor to capture the fingerprint images. The rest of the architecture was designed in MATLAB and is described in more detail in the following chapters.

CHAPTER 3 . FINGERPRINT DATABASES

Three public fingerprint databases were used in this work and a brief description is given below.

3.1. FVC2000

FVC2000 was the First International Competition for Fingerprint Verification Algorithms held in August 2000. This competition was organized by professors from various laboratories at the University of Bologna, San Jose State University, and Michigan State University. The results of 11 participants were presented at the 15th International Conference on Pattern Recognition [6]. From this database, we only used the set "B" compose with 4 subsets with 80 images each. The subsets belong to 10 different subjects with 8 impressions each. The following table presents detailed information for this set.

Table 3.1. Information for subsets in the FVC2000 database [6].

Subset	Sensor Type	Image Size
DB1	Low-cost optical sensor	300x300
DB2	Low-cost capacitive sensor	256x364
DB3	Optical sensor	448x478
DB4	Synthetic generator	240x320

The following figure presents four examples from this database.

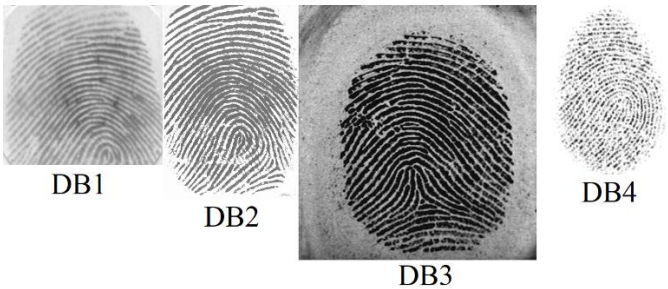

Figure 3.1. Examples of fingerprints from the FVC2000 database with one image from each subset *[6]*.

3.2. FVC2002

FVC2002 was the Second International Competition for Fingerprint Verification Algorithms held in April 2002. This competition was organized by the same professors from the FVC2000 competition. The results of 31 participants (21 industrial, 6 academic, and 4 others) were presented at the 16th International Conference on Pattern Recognition [7].

Similar to the FVC2000 database, we only use the set "B" composed of 320 images in total. The next table presents a detailed description of the subsets in this database:

Table 3.2. Information for subsets in the FVC2002 database [7].

Subset	Sensor Type	Image Size
DB1	Optical sensor	388x374 (142 Kpixels)
DB2	Optical sensor	296x560 (162 Kpixels)
DB3	Capacitive sensor	300x300 (88 Kpixels)
DB4	Synthetic generator	288x384 (108 Kpixels)

The following figure presents four examples from this database.

Figure 3.2. Examples of fingerprints from the FVC2002 database with one image from each subset *[7]*.

3.3. FVC2004

FVC2004 was the Third International Fingerprint Verification Competition and like the two previous competitions, it was organized by the same laboratories. The results of 43 participants (29 industrial, 6 academic, and 8 independent developers) were presented at the First International Conference on Biometric Authentication [8]. As in the previous two databases, we only use the set "B" with the same number of fingerprint images. The following table summarizes this database.

Table 3.3. Information for subsets in the FVC2004 database [8].

Subset	Sensor Type	Image Size
DB1	Optical sensor	640x480 (307 Kpixels)
DB2	Optical sensor	328x364 (119 Kpixels)
DB3	Thermal sweeping sensor	300x480 (144 Kpixels)
DB4	Synthetic generator	288x384 (108 Kpixels)

The following figure presents four examples from this database.

Figure 3.3. Examples of fingerprints from the FVC2004 database with one image from each subset *[8]*

CHAPTER 4 . PREPROCESSING

This chapter discusses the different image processing techniques used to enhance the ridge-valley patterns of fingerprints. First, we need to define some concepts about the characteristics of the shape of a fingerprint [3]:

- Directional field: Defined as the local orientation of the ridge-valley structures, it provides an approximate definition of the structure or shape of the fingerprint.
- Singular points: They are the discontinuities of the directional field separated into two classes: core and delta. The core is the uppermost point of the innermost curving ridge, and a delta is a point where three ridge flows meet.
- Minutiae: These are points with specific details about the ridge-valley structure, for example, the ridge endings and bifurcations. There are several other types of minutiae points, but they will not be considered as this work is primarily focused on classification using non-minutiae features.

The directional field is used in this work as an input fingerprint image enhancement method together with Gabor filters based on the fast enhancement algorithm proposed in [9]. The authors developed an algorithm capable of adaptively enhancing ridge-valley structures using both local ridge orientation and local frequency information [9] which are part of the directional field. This section presents brief definitions that will be used later to enhance fingerprints [9].

- A gray-level fingerprint image I is defined as a $M \times N$ matrix, where $I(i,j)$ represents the intensity of the current pixel located in the *ith* row and *jth* column. The mean and variance of this image are defined as:

$$M(I) = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} I(i,j)\,, \tag{4.1}$$

$$VAR(I) = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left(I(i,j) - M(I)\right)^2 . \quad (4.2)$$

- An orientation image O is defined as a $M \times N$ image, where $O(i,j)$ represents the local ridge orientation at pixel (i,j). The local ridge orientation is generally specified for a block of pixels rather than one. This implies that the fingerprint image is divided into $w \times w$ non-overlapping blocks and a single local ridge orientation is defined for each block. When analyzing a fingerprint, we can notice that the local ridge orientations of 90° and 270° are the same, since the ridges oriented at these angles in a local neighborhood cannot be differentiated from each other.
- A frequency image F is a $M \times N$ image, where $F(i,j)$ represents the local ridge frequency defined as the frequency of the ridge-valley structure in a local neighborhood along the normal direction to the local ridge orientation. This image is specified in blocks as the orientation image.
- A region mask R is a $M \times N$ image, where $R(i,j)$ indicates the category of a pixel. There are two possible categories:

    - Recoverable pixel: Where ridges and valleys are corrupted by a small amount of noise, this could be in the form of scars, creases, smudges, etc. However, neighboring regions or pixels can provide information about the true ridge-valley structures. These pixels are labeled with a value of 1.
    - Unrecoverable pixel: Where ridges and valleys are corrupted with a significant amount of noise and distortion making the structures not visible. The neighboring regions are not capable to provide information to recover the true ridge-valley structures. These pixels are labeled with a value of 0.

The following diagram presents the main steps in the enhancement algorithm used in this work:

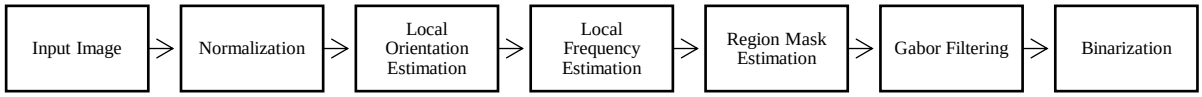

Figure 4.1. Flow diagram of the enhancement algorithm implemented *[9]*.

These steps are detailed in the following sections.

4.1. NORMALIZATION

Let $I(i,j)$ denote the gray-level value of the pixel (i,j), M and VAR the mean and variance of I, respectively, and $N(i,j)$ denote the normalized gray-value in the pixel (i,j) defined as:

$$N(i,j) = \begin{cases} M_0 + \sqrt{\dfrac{VAR_0(I(i,j)-M)^2}{VAR}}, & if\ I(i,j) > M \\ M_0 - \sqrt{\dfrac{VAR_0(I(i,j)-M)^2}{VAR}}, & otherwise \end{cases}, \tag{4.3}$$

where M_0 and VAR_0 are the desired mean and variance values, respectively. The normalization is a pixel-wise operation, and its objective is to clarify ridge-valley structures by reducing variations in gray-level values throughout the image, facilitating the next steps in preprocessing.

4.2. LOCAL ORIENTATION ESTIMATION

An orientation image provides information about a fingerprint in terms of oriented texture, an intrinsic property defined by invariant coordinates of ridges and valleys in a local neighborhood. Based on [9], and starting with a normalized image N, the following steps were implemented to create an orientation image applying a least mean square estimation algorithm:

1. Divide the normalized image N into blocks of size $w \times w$. The size has a direct impact on the estimation since a higher value will cover a bigger area of the image producing a wider estimation, not local as needed. In this case, it was implemented a size of $w = 16$ as recommended by the authors.
2. Calculate the gradients $\partial_x(i,j)$ and $\partial_y(i,j)$ for each pixel (i,j). In this case, the 5-tap 1st derivatives were implemented using coefficients given by Farid and Simoncelli [10].
3. Estimate the local orientation of each block centered at pixel (i,j) using the following equations:

$$V_x(i,j) = \sum_{u=i-\frac{w}{2}}^{i+\frac{w}{2}} \sum_{v=j-\frac{w}{2}}^{j+\frac{w}{2}} 2\partial_x(u,v)\partial_y(u,v), \tag{4.4}$$

$$V_y(i,j) = \sum_{u=i-\frac{w}{2}}^{i+\frac{w}{2}} \sum_{v=j-\frac{w}{2}}^{j+\frac{w}{2}} (\partial_x^2(u,v) - \partial_y^2(u,v)), \tag{4.5}$$

$$\theta(i,j) = \frac{1}{2}\tan^{-1}\left(\frac{V_y(i,j)}{V_x(i,j)}\right), \tag{4.6}$$

where $\theta(i,j)$ is the least square estimate of the local ridge orientation at the block centered in (i,j). In the mathematical sense, this value represents the orthogonal direction to the dominant direction of the Fourier spectrum of the $w \times w$ window.

4. The local ridge orientation varies slowly in a local neighborhood where singular points do not appear, this implies that a lowpass filter can be used to modify a corrupted local ridge orientation altered by the presence of noise. Before applying a low pass filter, the orientation image must be converted to a continuous vector field defined as:

$$\Phi_x(i,j) = \cos\big(2\theta(i,j)\big), \tag{4.7}$$

$$\Phi_y(i,j) = \sin\big(2\theta(i,j)\big). \tag{4.8}$$

Using these values, the lowpass filtering is performed as:

$$\Phi'_x(i,j) = \sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} \sum_{v=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} W(u,v)\Phi_x(i-uw, j-vw), \tag{4.9}$$

$$\Phi'_y(i,j) = \sum_{u=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} \sum_{v=-\frac{w_\Phi}{2}}^{\frac{w_\Phi}{2}} W(u,v)\Phi_y(i-uw, j-vw), \tag{4.10}$$

where W is a two-dimensional lowpass Gaussian filter and $w_\Phi \times w_\Phi$ is the size of this filter, in this case, 5×5.

5. Finally, the local ridge orientation at (i,j) is calculated as:

$$O(i,j) = \frac{1}{2}\tan\left(\frac{\Phi'_y(i,j)}{\Phi'_x(i,j)}\right). \tag{4.11}$$

4.3. LOCAL FREQUENCY ESTIMATION

The gray level intensities along ridges and valleys create a local neighborhood where no minutiae or singular points appear. This can be modeled as a sine-shaped wave along the normal direction to the local ridge orientation. This indicates that the local ridge frequency is another intrinsic property of a fingerprint image. If N is the normalized image and O is the orientation image, the following steps calculate the local ridge frequency estimation:

1. Divide N in blocks of size $w \times w$, in this case, this value was set to $w = 16$.
2. For each block centered on (i, j), calculate an oriented window of size $l \times w$ (32×16) defined in the ridge coordinates system.
3. For each block centered on (i, j), calculate the x-signature, $X[0], X[1], \dots, X[l-1]$, of the ridges and valleys within the oriented window, where:

$$X[k] = \frac{1}{w} \sum_{d=0}^{w-1} N(u, v), \qquad k = 0,1, \dots, l-1, \tag{4.12}$$

$$u = i + \left(d - \frac{w}{2}\right) \cos O(i, j) + \left(k - \frac{l}{2}\right) \sin O(i, j). \tag{4.13}$$

$$v = j + \left(d - \frac{w}{2}\right) \sin O(i, j) + \left(\frac{l}{2} - k\right) \cos O(i, j). \tag{4.14}$$

   If there are no singular points or minutiae within the oriented window, the x-signature forms a discrete sinusoidal-shaped wave with the same frequency as the ridge-valley structure in the oriented window. Hence, the frequency of the ridges and valleys can be estimated from this x-signature. If $T(i, j)$ is the average number of pixels between two consecutive peaks in the x-signature, the frequency $\Omega(i, j)$ is calculated as $\Omega(i, j) = 1/T(i, j)$. If there are no consecutive peaks in the x-signature, the frequency is assigned a value of -1 to differentiate it from valid frequency values.
4. If minutiae and/or singular points appear in a block, the x-signature does not form a well-defined sinusoidal-shaped wave. This also happens for ridge-valley structures that are corrupted by any kind of noise. The frequency values for these blocks must be interpolated from the frequency of the neighboring blocks that have valid values. The x-signature behaves as an indicator of the presence of minutiae or singular points depending on the shape of the wave that this technique produces.

5. The distances between the ridges can change slowly in a local neighborhood which means that a lowpass filter can be used to remove outliers in the interpolated version. The following equation applies this filter to the interpolated version of the frequency called Ω' to create the local ridge frequency:

$$F(i,j) = \sum_{u=-\frac{w_\Omega}{2}}^{\frac{w_l}{2}} \sum_{v=-\frac{w_\Omega}{2}}^{\frac{w_l}{2}} W_l(u,v)\Omega'(i-uw, j-vw), \tag{4.15}$$

where W_l is a two-dimensional lowpass filter and $w_l = 7$ is the size of the filter.

4.4. REGION MASK ESTIMATION

A fingerprint image has pixels or blocks that could be in a recoverable or unrecoverable region. This classification process can be done by evaluating the wave shape created by local ridges and valleys. The algorithm applied in this work uses three characteristics to describe the sinusoidal-shape wave:

- Amplitude, denoted by α
- Frequency, denoted by β
- Variance, denoted by γ

If $X[1], X[2], \ldots, X[l]$ is the x-signature of a block centered on (i,j), the three characteristics are computed as:

$$\alpha = H_p - D_p, \tag{4.16}$$

where H_p is the average height of the peaks, and D_p is the average depth of the valleys.

$$\beta = \frac{1}{T(i,j)}, \tag{4.17}$$

where $T(i,j)$ is the average number of pixels between two consecutive peaks.

$$\gamma = \frac{1}{l}\sum_{i=1}^{l}\left(X[i] - \frac{1}{l}\sum_{i=1}^{l} X[i]\right)^2. \tag{4.18}$$

These features create three-dimensional patterns that are classified using a 1NN classifier into two clusters: recoverable and unrecoverable. If a block centered on (i, j) is recoverable, then $R(i, j) = 1$, else $R(i, j) = 0$.

4.5. GABOR FILTERING

An interesting characteristic of fingerprints is their inherent configuration of parallel ridges and valleys, which have a well-defined frequency and orientation providing useful information that helps eliminate unwanted noise. Since the sinusoidal-shape waves from ridge-valley structures vary slowly in a local orientation, a well-tuned bandpass filter at a specific frequency and orientation can remove noise while preserving true ridge-valley structures.

Gabor filters are suitable for use as bandpass filters because they have frequency-selective and orientation-selective properties, giving them optimal joint resolution in the spatial and frequency domains [11]. These characteristics help to eliminate undesired noise while preserving true ridges and valleys. The even-symmetric Gabor filter has the general form:

$$h(x, y: \phi, f) = e^{-\frac{1}{2}\left[\frac{(x\cos\phi)^2}{\sigma_x^2} + \frac{(y\sin\phi)^2}{\sigma_y^2}\right]} \cos(2\pi f x \cos\phi), \tag{4.19}$$

where ϕ is the orientation of the Gabor filter, f is the frequency of the sinusoidal plane wave, and σ_x and σ_y are the space constants of the Gaussian envelope along the x and y axes, respectively.

To apply Gabor filters to an image, three parameters must be specified:

- The frequency of the sinusoidal plane wave
- The filter orientation
- The standard deviations of the Gaussian envelope

The first parameter corresponds to the local ridge frequency and the second parameter is the local ridge orientation. The third parameter involves a trade-off of values, since the higher these values, the more resistant to noise the filters, but this is more likely to create spurious ridges and valleys. In contrast, the smaller the values, the filters will not create spurious ridge-valley structures but will be less effective at removing noise. In this work, both values were implemented as 0.5.

Using the estimated images, G as the normalized fingerprint, O as the orientation image, F as the frequency image, and R as the region mask, the enhanced image E is calculated as:

$$E(i,j) = \begin{cases} 255, & if\ R(i,j) = 0 \\ \sum_{u=-\frac{w_g}{2}}^{\frac{w_g}{2}} \sum_{v=-\frac{w_g}{2}}^{\frac{w_g}{2}} h\big(u, v: O(i,j), F(i,j)\big) G(i-u, j-v), & otherwise \end{cases} \quad (4.20)$$

where $w_g = 11$ is the size of the Gabor filters.

4.6. BINARIZATION

This last step is implemented using a threshold in the enhanced image E using the following criteria:

$$B(i,j) = \begin{cases} 1, & if\ E(i,j) \geq 0 \\ 0, & otherwise \end{cases}, \quad (4.21)$$

where $B(i,j)$ is the binarized version of the fingerprint, concluding the preprocessing of the input images.

The concepts and procedures discussed in this chapter were implemented in MATLAB based on the functions developed by Dr. Peter Kovesi [12].

4.7. EXAMPLES

The following figures present several examples of the preprocessed fingerprint images, one for each subset in the three databases:

4.7.1. FVC2000

The fingerprint preprocessing is presented from left to right: original, normalized, orientation field, Gabor filtering, and binarized images; using one impression per subset from the FVC2000 database [6]:

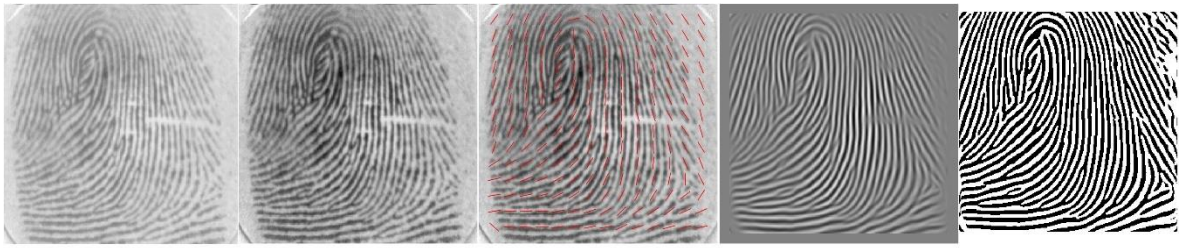

Figure 4.2. DB1 subset from the FVC2000 database.

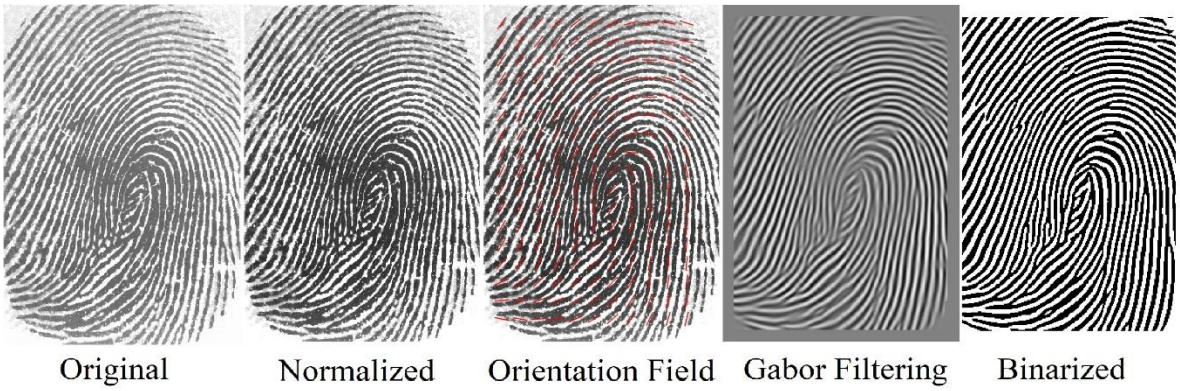

Figure 4.3. DB2 subset from the FVC2000 database.

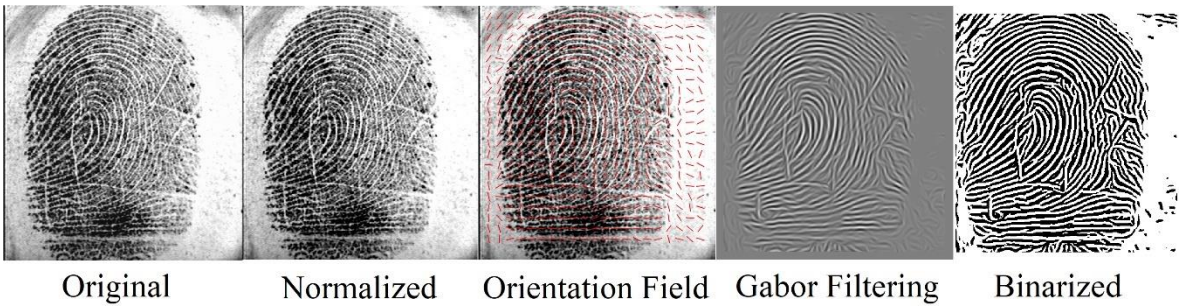

Figure 4.4. DB3 subset from the FVC2000 database.

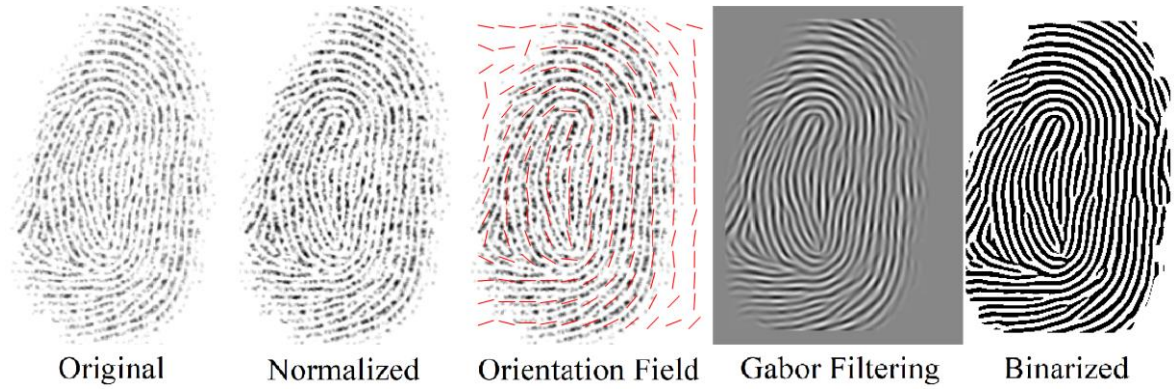

Figure 4.5. DB4 subset from the FVC2000 database.

4.7.2. FVC2002

Fingerprint preprocessing using one impression per subset from the FVC2002 database [7]:

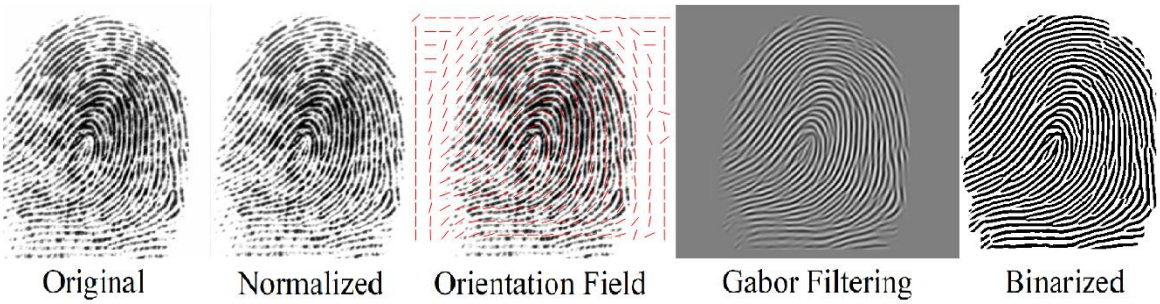

Figure 4.6. DB1 subset from the FVC2002 database.

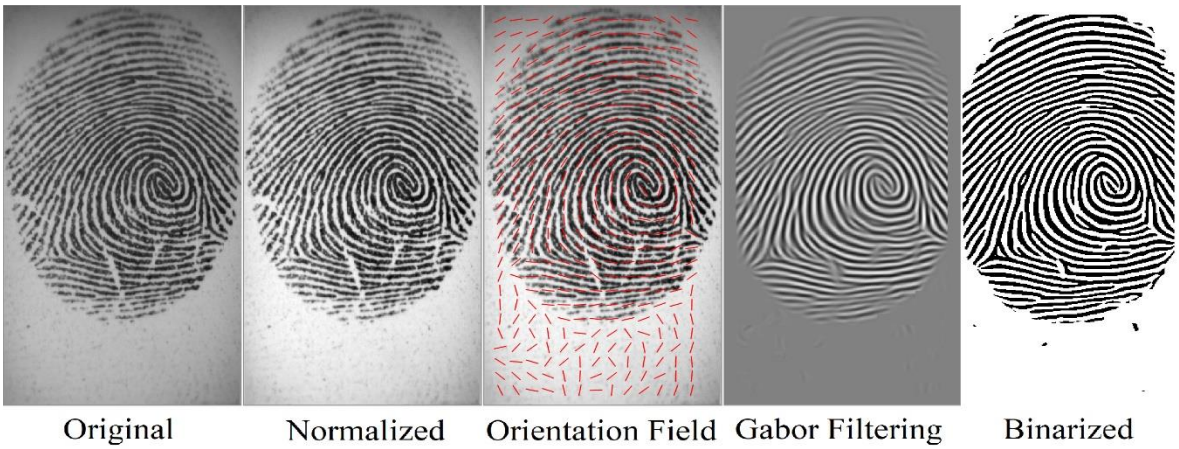

Figure 4.7. DB2 subset from the FVC2002 database.

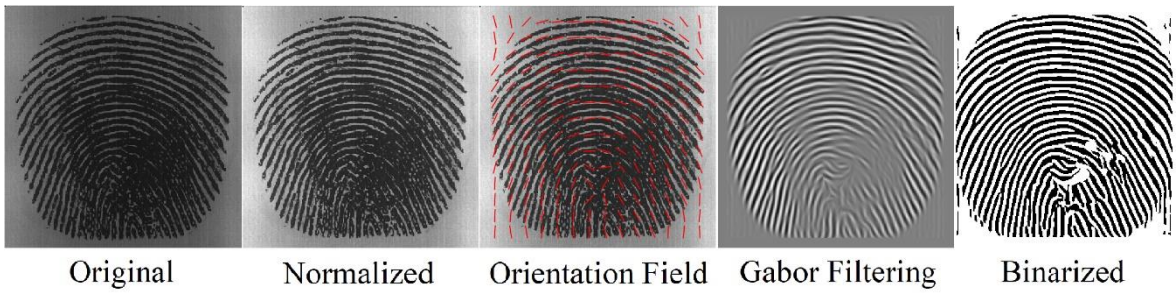

Figure 4.8. DB3 subset from the FVC2002 database.

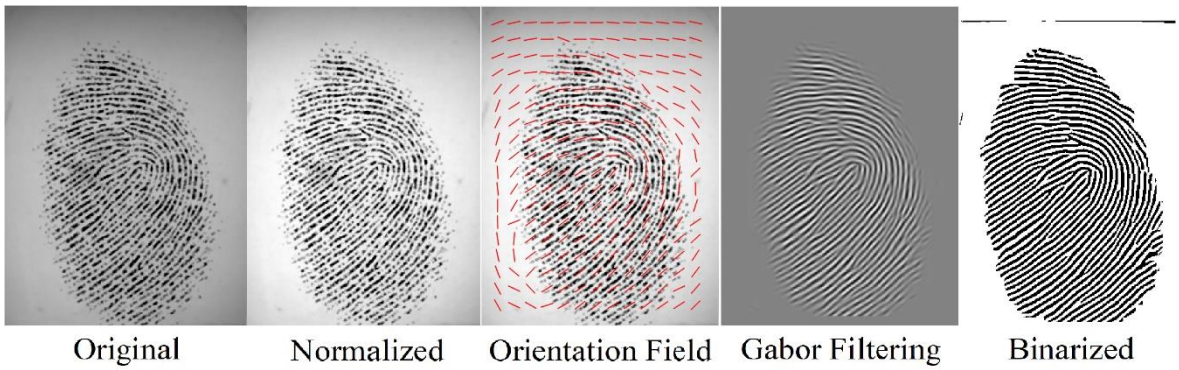

Figure 4.9. DB4 subset from the FVC2002 database.

4.7.3. FVC2004

Fingerprint preprocessing using one impression per subset from the FVC2004 database [8]:

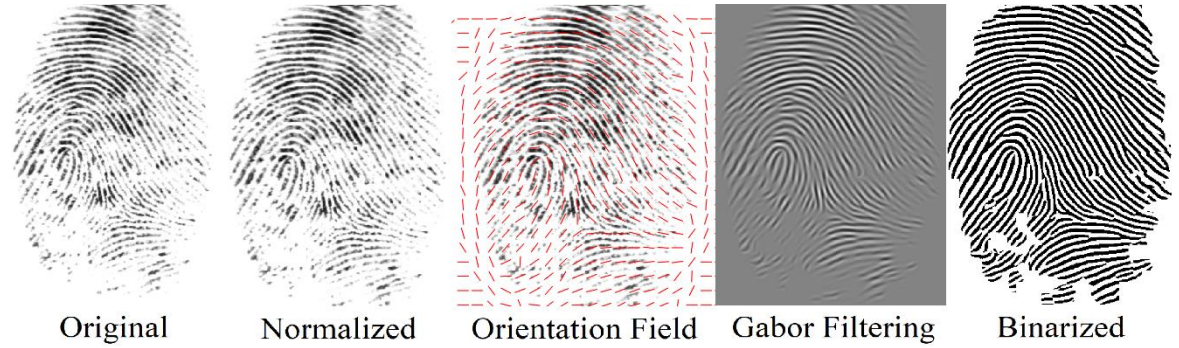

Figure 4.10. DB1 subset from the FVC2004 database.

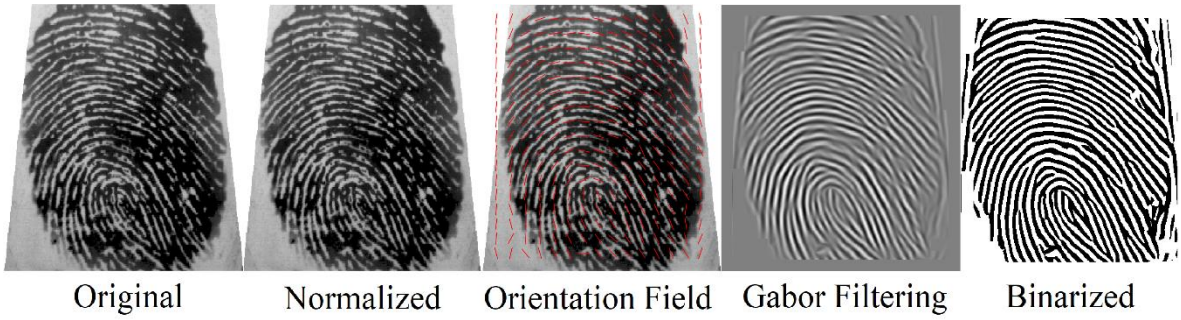

Figure 4.11. DB2 subset from the FVC2004 database.

Figure 4.12. DB3 subset from the FVC2004 database.

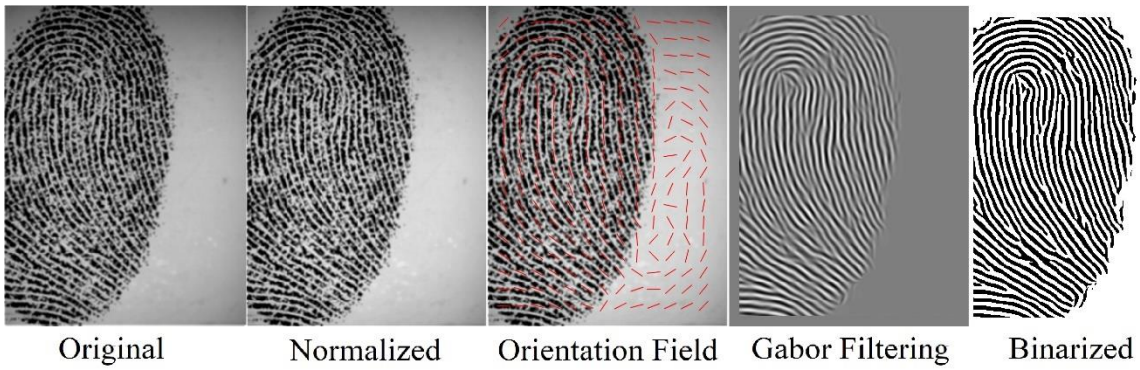

Figure 4.13. DB4 subset from the FVC2004 database.

CHAPTER 5 . FEATURE EXTRACTION OF FINGERPRINTS

In the fingerprint recognition literature, minutiae points have always been used as features for fingerprint classification. However, nowadays we have other approaches, in this work we will focus on texture-based features, specifically using the Discrete Wavelet Transform (DWT) in combination with various transformations detailed in the following sections. This procedure creates a hybrid fingerprint recognition system that uses a significant component of the rich discriminatory information available in the texture of fingerprint images that contains local and global information [5].

5.1. DISCRETE WAVELET TRANSFORM

The Fourier Transform applied to an image provides information regarding the image's frequency attributes. In contrast, the Discrete Wavelet Transform (DWT) provides powerful insight into the spatial and frequency characteristics of an image [1]. This advantage of the DWT provides a highly intuitive framework for the representation of fingerprint images in terms of their texture components. The Fast Wavelet Transform (FWT) is an iterative computational approach to the DWT using filter banks as can be seen in the following figure.

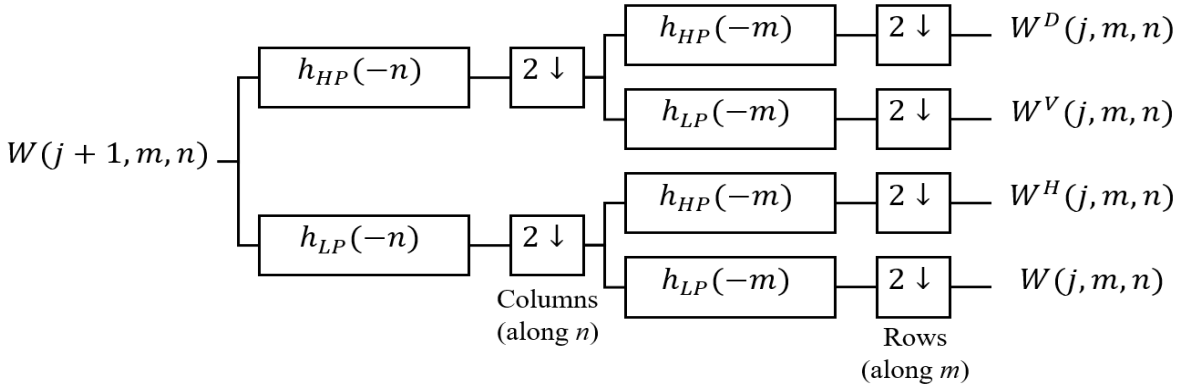

Figure 5.1. 2-D FWT filter bank *[1]*.

where $W(j,m,n)$ and $\{W^i(j,m,n)\ for\ i = H,V,D\}$ are the DWT coefficients at scale j. The blocks that contain the time-reversed vectors $h_{LP}(-n)$ and $h_{HP}(-m)$ are low pass and high pass decomposition filters, respectively. The blocks with a 2 and a down arrow represent downsampling. The input to the filter bank is decomposed into four lower resolution components. The W coefficients are created via two low pass filters and are called approximation coefficients, and $\{W^i\ for\ i = H,V,D\}$ are the horizontal, vertical, and diagonal detail coefficients, respectively. The output $W(j,m,n)$ can be used as a subsequent input $W(j+1,m,n)$ to the block diagram to create even lower resolution components. The image $f(x,y)$ is the highest resolution representation available and serves as the input for the first iteration. The three transform domain variables involved are the scale j, and the horizontal and vertical translation, n and m [1].

The Wavelet Toolbox is a package of functions and applications for analyzing and synthesizing signals and images. The toolbox includes algorithms for continuous and discrete Wavelet analysis [13]. In MATLAB, the DWT is implemented through the Wavelet Toolbox, specifically using the *wavedec2* function with the following syntax:

```
[c, s] = wavedec2(f, j, wname)
```

where f is the preprocessed image, j is the number of scales to be calculated, and *wname* specifies the name of the FWT filter family name. In this case, the DWT was applied twice to extract features, the following table details the parameters used for both cases.

Table 5.1. Parameters of the DWT computed in this work.

References	**Features**	**j**	**wname**
[14]	GLCM	1	‘db1’
[15, 16]	Statistics, Wavelet-Bands Selection Features	5	‘db12’

Both DWTs were calculated to obtain the Wavelet representation of the preprocessed fingerprint that will then be used to produce the feature specified in the table.

The *wavedec2* outputs provide information about the Wavelet decomposition vector in c, and a bookkeeping matrix s, containing the number of coefficients per level and orientation [17].

5.2. GRAY LEVEL CO-OCCURRENCE MATRIX

Texture analysis characterizes the spatial variation of the pattern in an image based on mathematical models and procedures [18]. The approach applied in this work is based on one of the earliest and most widely used methods known as the Gray Level Co-

occurrence Matrix (GLCM). The GLCM is a square matrix that provides certain properties about the spatial distribution of gray levels in the texture of an image [18]. This matrix shows how often a reference pixel value with intensity i occurs in a specific relationship with another neighboring pixel with intensity j. In other words, each element (i, j) of the GLCM is the number of occurrences of the pixel pair at a distance d relative to each other [18]. This spatial relationship can be defined in multiple forms with different offsets and angles. For an image I of size $M \times N$, the elements of the corresponding GLCM for a displacement vector $d = (d_x, d_y)$ are defined as:

$$GLCM = \sum_{x=1}^{M} \sum_{y=1}^{N} \begin{cases} 1, & if\ I(x,y) = i\ \ and\ \ I(x+d_x, y+d_y) = j \\ 0, & otherwise \end{cases}, \tag{5.1}$$

The following figure illustrates the four spatial relationships used in this work, considering one neighboring pixel $d = 1$ along with the four possible directions [0 1] for 0°, [-1 1] for 45°, [-1 0] for 90°, and [-1 -1] for 135° [14].

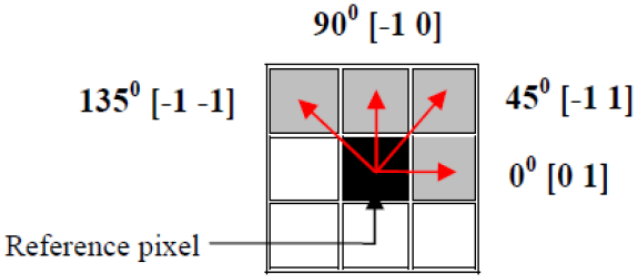

Figure 5.2. Gray level co-occurrence matrix directions *[18]*.

Each element of the GLCM is the number of times that two pixels with gray values i and j are neighbors in distance d and direction θ [18]. A regular histogram does not carry information about the relative position of the pixels with respect to each other, that is why for texture measurements, the GLCM is mainly used since it incorporates in the texture analysis not only the distribution of intensities but also the relative position of pixels in an image [1]. The number of possible intensity levels in the original image determines the size of the GLCM. In this case, we work with 8-bit images, which means that there are 256 possible levels [1].

In MATLAB we apply the *graycomatrix* function from the Image Processing Toolbox to compute the co-occurrence matrices. The syntax used was:

```
GLCM = graycomatrix(A, 'NumLevels', 256, 'Offset', offsets)
```

where *A* is the approximation image obtained from the first DWT using the *appcoef2* function from the Wavelet Toolbox. This matrix is a representation of the approximation coefficients for the first scale calculated from the preprocessed fingerprint image. The other two parameters specify the number of levels applied in the GLCM calculation and the vector of offsets that provides the four directions detailed previously. The number of levels was specified as 256 for the best possible representation, as this is the highest possible value for an 8-bit image.

The GLCM is computed on the approximation image because the Wavelet transform decomposed the original image into these lower frequency coefficients ignoring the noise signals that are related to the higher frequencies, which are present in the detail coefficients [14]. Since this approximation has a lower resolution, it provides a compressed representation of the fingerprint image, allowing to ignore several extra details that are not relevant to the texture information for this specific application.

So far, we have only calculated the GLCM, but we need to compute the texture descriptors that will be a part of the feature set used for fingerprint recognition. To accomplish this objective, we apply the *graycoprops* function to generate four descriptors using the following syntax:

```
stats = graycoprops(GLCM, 'all')
```

where *'all'* specifies that all available descriptors must be calculated including contrast, correlation, energy, and homogeneity; whose equations are presented below:

$$Contrast = \sum_{i,j} |i-j|^2 \, GLCM_{i,j}, \tag{5.2}$$

$$Correlation = \sum_{i,j} \frac{(i-\mu i)(j-\mu j) GLCM_{i,j}}{\sigma_j \sigma_j}, \tag{5.3}$$

$$Energy = \sum_{i,j} GLCM_{i,j}^2, \tag{5.4}$$

$$Homogeneity = \sum_{i,j} \frac{GLCM_{i,j}}{1+|i-j|}, \tag{5.5}$$

where the first-order statistics μ and σ are the mean and variance, respectively [14]. In this case, because we use four different orientations, the total number of features per fingerprint is 16 as texture descriptors. For each orientation, the GLCM is calculated applying its respective offset, which means that equations (5.2) – (5.5) need to be applied four times, because the GLCM variable will have four different values, one for each offset.

5.3. SPATIAL DOMAIN

In the spatial domain, five different features are calculated including the pixel density, the mean of standard deviations (μ^{σ}), the standard deviation of the means (σ^{μ}), the mean of the variances (μ^{D}), and the standard deviation of the variances (σ^{D}) [19]. The pixel density is calculated using a binarized fingerprint obtained from the preprocessing. The other four features were calculated in a normalized version of the original fingerprint [19]. For an image of size $M \times N$ with pixel values $\{p_{i,j} | i = 1,2,\dots,M; j = 1,2,\dots,N\}$, the pixel density is defined as:

$$Pixel\ density = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} p_{ij} = 1. \tag{5.6}$$

The mean of the standard deviations is defined as:

$$\mu^{\sigma} = \frac{1}{N}\sum_{j=1}^{N}\sqrt{\frac{\sum_{i=1}^{M}\left(p_{i,j} - \mu^{j*}\right)^2}{M}}, \tag{5.7}$$

where

$$\mu^{j*} = \frac{1}{N}\sum_{i=1}^{M} p_{i,j} \mid j = 1,2,\dots,M. \tag{5.8}$$

The standard deviation of the means is:

$$\sigma^{\mu} = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left[\left(\frac{1}{M}\sum_{i=1}^{M} p_{i,j}\right) - \mu^{ij*}\right]^2}, \tag{5.9}$$

where

$$\mu^{ij*} = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} p_{i,j}. \tag{5.10}$$

The mean of the variances is:

$$\mu^{D} = \frac{1}{N}\sum_{j=1}^{N}\left[\frac{1}{M}\sum_{i=1}^{M}\left|p_{i,j} - \mu^{j*}\right|\right]. \tag{5.11}$$

Finally, the standard deviation of the variances is defined as:

$$\sigma^D = \sqrt{\frac{\sum_{j=1}^{N}\left[\frac{\sum_{i=1}^{M}\left|p_{i,j} - \mu^{j*}\right|}{M} - \mu^D\right]^2}{N}}. \tag{5.12}$$

In MATLAB, the previous equations were implemented using native functions including *bwarea*, *mean*, *std*, and *var*. A brief description of each function is presented below.

The *bwarea* function estimates the area of a binary image by counting the total number of white pixels (intensity value of 1). The syntax is:

```
total = bwarea(BW)
```

where BW is the binary image and total is the number of white pixels. This value is divided by the product of the number of rows and columns of the image, obtaining the pixel density feature.

The other statistical functions have the following syntax:

```
M = mean(A)
S = std(A, 1)
V = var(A, 1)
```

where it is self-explanatory that the M, S, and V variables are the mean, standard deviation, and variance, respectively. The parameter A is the vector or matrix to apply the current statistics. Since the default computation in MATLAB for the *std* and *var* functions utilizes $N-1$ or $M-1$, it was necessary to include the parameter 1 which specifies that the calculation of the standard deviation and the variance is normalized by the number of observations N or M to have correspondence with equations (5.7) to (5.12) [20, 21]. From this section, 5 features of each fingerprint have been calculated.

5.4. FOURIER DOMAIN

The Discrete Fourier Transform (DFT) is one of the most important image processing tools used to create a frequency representation of an image by decomposing it into its sine and cosine components [5]. In this case, because we use two-dimensional

matrices, the DFT must be implemented in 2-D. If $f(x,y)$ is the normalized fingerprint image of size $M \times N$, the 2-D DFT is defined by:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y)\, e^{-j2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}, \tag{5.13}$$

for $u = 0,1,2,\dots,M-1$ and $v = 0,1,2,\dots,N-1$. The Fourier or frequency domain is the coordinate system spanned by $F(u,v)$ with u and v as (frequency) variables [1]. In this work, these frequency variables provide different information about fingerprints and therefore are used as features for recognition. Specifically, the features derived from the resulting DFT are $\mu^{\sigma F}, \sigma^{\mu F}, \mu^{DF}, \sigma^{DF}$, where F stands for the 2-D frequency transformation. These features were already defined for the spatial domain subsection in the equations (5.7), (5.9), (5.11), and (5.12), but instead of using $p_{i,j}$ its analog Fourier variable $F(u,v)$ is applied.

In MATLAB, the Fast Fourier Transform (FFT) in two dimensions is applied to the input normalized fingerprints as a way to calculate the DWT efficiently. The *fft2* function has the following basic syntax:

```
F = fft2(f)
```

where f is the input image, and F is the 2-D Fourier transform using the FFT algorithm. The output transform is the same size as the input image [22].

A total of 4 new features from each fingerprint are calculated in the frequency domain.

5.5. DISCRETE COSINE TRANSFORM

The Discrete Cosine Transform (DCT) is another transformation in frequency, but this one only uses the cosine terms, i.e., the real coefficients [19]. If $f(x,y)$ is the enhanced fingerprint image of size $M \times N$, the 2-D DCT is defined as:

$$DCT(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\left(\frac{(2x+1)u\pi}{2M}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right), \tag{5.14}$$

for $u = 0,1,2,\dots,M-1$ and $v = 0,1,2,\dots,N-1$. The DCT is a real, orthogonal, fast, and separable transform [5]. From this transformation, four features were computed from the resulting DCT called $\mu^{\sigma DCT}, \sigma^{\mu DCT}, \mu^{DDCT}, \sigma^{DDCT}$, where DCT stands for the 2-D DCT

transformation. These features were calculated according to equations (5.7), (5.9), (5.11), and (5.12) using the $DCT(u,v)$ instead of $p_{i,j}$.

In MATLAB, the 2-D DCT is implemented by the *dct2* function with the syntax:

```
DCT = dct2(f)
```

where f is the input image, and DCT is the 2-D DCT. This matrix is the same size as the input [23].

From the DCT representation, 4 features are calculated for each fingerprint image.

5.6. STATISTIC MEASURES ON THE DWT

Based on a trial-and-error method, the number of scales or decompositions applied to the second DWT in this work is $j = 5$ which creates a total of $3j + 1 = 3(5) + 1 = 16$ lower resolution images. The Wavelet mother applied, in this case, was 'db12', also found based on trial-and-error. For each of these images, excluding the approximation image, seven statistical measures were computed including maximum, mean, standard deviation, Euclidian norm, variance, skewness, and kurtosis [15, 19].

5.6.1. Maximum

In MATLAB, native functions were applied to compute these features. The *max* function provides the maximum elements of a vector or matrix D with the following syntax [24]:

```
M = max(D)
```

In this case, since we use matrices, the *max* function needs to be applied twice, first to the columns and then to the rows of the detail image.

5.6.2. Euclidean norm

For an image of size $M \times N$ with pixel values $\{p_{i,j} | i = 1,2,\dots,M; j = 1,2,\dots,N\}$ the Euclidean norm is defined by the following equation:

$$n = \|D\| = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N}\left|p_{i,j}\right|^2}, \tag{5.15}$$

the square root of the sum of all the squares in the detail image. The *norm* function returns the Euclidean norm, the following syntax was used:

```
n = norm(D)
```

where D is the detail image, and n is the Euclidean norm, also called the vector magnitude [25].

5.6.3. Skewness

Skewness is a measure of the asymmetry of the data around the sample mean. If the skewness is positive, the data spread out more to the right of the mean, if it is negative, the data is extended further to the left. By logic, this implies that, for example, for a normal or Gaussian distribution (or any perfectly symmetric distribution) the skewness is zero [26]. The skewness of a distribution is defined as:

$$s = \frac{E(x-\mu)^3}{\sigma^3}, \tag{5.16}$$

where μ is the mean of x, σ is the standard deviation of x, and $E(t)$ is the expected value of t. The *skewness* function in MATLAB, which belongs to the Statistics and Machine Learning Toolbox, computes a sample version of this population value. By default, MATLAB uses a biased skewness and applies the following equation:

$$s_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^3}{\left(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2}\right)^3}, \tag{5.17}$$

where s_1 is the sample skewness of the input vector x [27]. The *skewness* function has the following syntax:

```
y = skewness(D)
```

where y is the sample skewness of D. Since we use images (matrices), we apply the *skewness* function twice to produce only one value as the current feature for the input detail image.

5.6.4. Kurtosis

Kurtosis is a measure of how outlier-prone a distribution is. In other words, it measures how prone a distribution is to have atypical values. By definition, the kurtosis of the normal distribution is 3 [28]. This means that distributions that are more prone to

outliers than the normal distribution will have kurtosis greater than 3, and distributions that are less outlier-prone have a value less than 3 [28]. The kurtosis of a distribution is defined as:

$$k = \frac{E(x-\mu)^4}{\sigma^4}, \tag{5.18}$$

where μ and σ are the mean and standard deviation of x, respectively; and $E(t)$ is the expected value of t. The *kurtosis* function in MATLAB, which is also part of the Statistics and Machine Learning Toolbox, calculates a sample version of this population value. By default, MATLAB uses a biased kurtosis and applies the following equation:

$$k_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)^2}, \tag{5.19}$$

where k_1 is sample kurtosis of the input vector x [28]. The *kurtosis* function has the following syntax:

```
k = kurtosis(D)
```

where k is the sample kurtosis of D. In the same way, as for the *skewness* function, the kurtosis was calculated first for columns and then for the rows to obtain a single value per detail image.

From this sub-section, we have a total of 105 new features per fingerprint. This value is obtained by computing the seven statistics for each level of decomposition and each of the detail coefficients (horizontal, vertical, diagonal). The total is calculated in the following equation:

$$\begin{aligned} &7\ features * 3\ detail\ images * 5\ levels\ of\ decomposition \\ &= 105\ features. \end{aligned} \tag{5.20}$$

5.7. WAVELET-BANDS SELECTION FEATURES

Based on the work reported in [16], another set of features has been calculated with the objective of improving recognition rates. In this method, new Wavelet features called Wavelet-Band Selection Features (WBSF) are extracted from the five decomposition levels (the second DWT applied in this work). An example of this Wavelet decomposition is presented in the following figure using a fingerprint from the FVC2000 database [6].

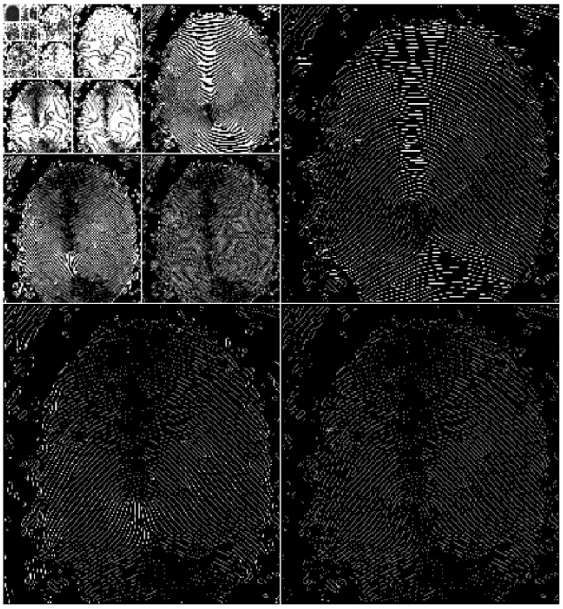

Figure 5.3. Example of 5-decomposition levels of the 2-D DWT applied to a fingerprint *[6]*.

The idea is to divide the Wavelet bands shown in the previous figure into sub-bands as can be seen in Figure 5.4. These features provide information about the fingerprint image in both horizontal and vertical directions. The added features are the shaded cells shown in the following figure. The colored lines represent the original five-level divisions analogous to Figure 5.3.

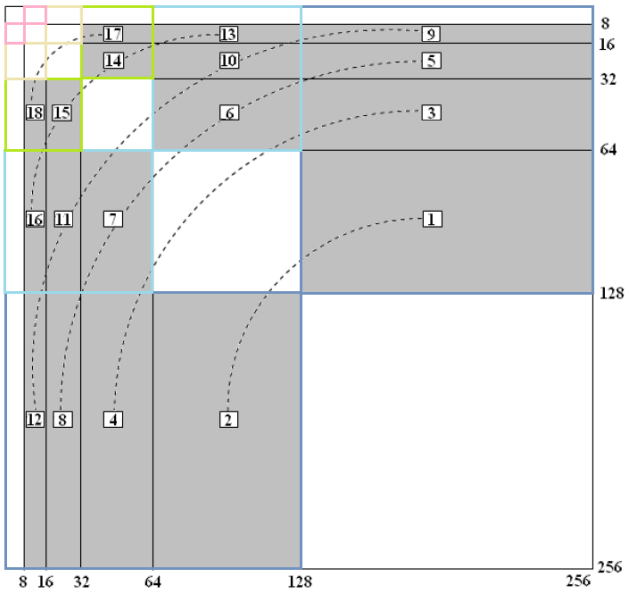

Figure 5.4. Wavelet channel decomposition (5-levels) indicating the number of new features calculated *[16]*.

The length specified in the previous figure represents an image of size 256×256, however, these limits can be generalized by taking the height of each level and divide it by 4, 3, and 2 in the case of the first, second, and third level of decomposition, respectively. The last two resolution levels (four and five) and diagonal detail images are not considered in this procedure [16]. A total of 18 new sub-bands were calculated for each fingerprint from the same DWT applied for the statistics features. From these sub-bands, two statistics were calculated, the mean and the standard deviation [16]. These two statistics create a total of 36 new features for each fingerprint.

5.8. TOTAL NUMBER OF FEATURES

The following table summarizes the total number of features calculated per fingerprint image in this work. The feature set has a total of 170 features to be used with supervised learning algorithms for fingerprint recognition.

Table 5.2. Total number of features.

Method	**Features**	**# of features per fingerprint**
2-D DWT (1 scale, 'db1'), GLCM	Contrast, correlation, energy, homogeneity	16
Preprocessed image (normalization, binarization)	Pixel density, $\mu^{\sigma}, \sigma^{\mu}, \mu^{D}, \sigma^{D}$	5
2-D FFT	$\mu^{\sigma F}, \sigma^{\mu F}, \mu^{DF}, \sigma^{DF}$	4
2-D DCT	$\mu^{\sigma DCT}, \sigma^{\mu DCT}, \mu^{DDCT}, \sigma^{DDCT}$	4
2-D DWT (5 scales, 'db12'), statistics measures	Maximum, mean, standard deviation, Euclidean norm, variance, skewness, kurtosis	105
2-D DWT (5 scales, 'db12'), WBSF	Mean, standard deviation	36
	TOTAL	170

CHAPTER 6 . MACHINE LEARNING ALGORITHMS

Machine learning is an engineering approach whose main objective is to study, design, and improve mathematical models which can be trained (once or continuously) with context-related data, to infer the future and make decisions without complete knowledge of all the influencing elements (external factors) [29]. A software entity receives information from the environment and chooses the best action to achieve a specific goal by adopting a statistical learning approach, trying to determine the correct probability, and use it to compute an action (value or decision) that is most likely to have success (the smallest possible error) [29]. Machine learning teaches computers to learn from experience, in other words, algorithms use computational models to "learn" information from data without relying on a specific equation used as a model [30]. Machine learning algorithms find natural patterns in data, and this helps to make decisions and predictions. This behavior has been applied in various real-world applications, including computational finance, computer vision and image processing, computational biology, energy production, natural language processing, etc. [30]

The machine learning approach applied in this work is supervised learning. This makes sense because there are public fingerprint databases available online, these databases act as a supervised scenario providing a training set made up of pairs (input and expected output) [29]. From the information provided by the fingerprint images, which indicate the corresponding subject to who they belong, a supervised machine learning algorithm corrects its parameters to reduce the magnitude of a global loss function, after each iteration, if the algorithm is flexible enough and data elements are coherent the overall accuracy increases, and the difference between the predicted and expected value approaches zero [29].

However, in a supervised scenario, the goal is to train a system that has to work with never-before-seen samples. This means that the model must have a generalization ability and avoid the undesired problem called overfitting [29]. This problem is avoided in this work by using 10-fold cross-validation. This overfitting could lead to overlearning due

to an excessive capacity to correctly predict only the samples used for training, while the error for the remainder is always very high [29].

For fingerprint recognition, the primary goal of a supervised machine learning algorithm is to classify an input fingerprint image into a discrete number of possible outcomes (subjects) or categories. In this work, we use a system labeled: “Subject-One”, “Subject-Two”, and so on; to indicate the correspondence of a query input fingerprint to the predicted outcome computed by machine learning algorithms.

In terms of which algorithm to use for a certain application, there are dozens of supervised machine learning algorithms, each with a different approach to learning [30]. The right algorithm is usually found through trial and error, and it also depends on the size and type of data.

6.1. SUPERVISED MACHINE LEARNING ALGORITHMS

In this work, multiple supervised algorithms have been tested using a native application called Classification Learner. The application performs automated training to find the best type of classification model among a group of supervised machine learning models including decision trees, discriminant analysis, Support Vector Machines (SVM), nearest neighbors, naive Bayes, and ensemble classifiers [31]. A description of each classification model is presented below.

6.1.1. *k*-Nearest Neighbors

The *k*-Nearest Neighbors (*k*-NN) algorithm is one of the simplest classifiers because the model only needs to store the training set for it to work [32]. The *k*-NN algorithm categorizes the outputs according to the class defined by their “nearest neighbors” in the dataset. This implies that *k*-NN assumes that the outputs near each other are similar [30]. The two parameters of a *k*-NN algorithm are the number of neighbors and the distance measure. There are multiple variations for distance metrics, including Euclidean, city block, cosine, and Chebyshev [30]. An advantage of *k*-NN is the reasonable performance obtained without many adjustments, however, this algorithm has high costs in terms of memory and prediction speed. This explains why it is not used very often in practice [32]. The following figure shows a representation of this algorithm.

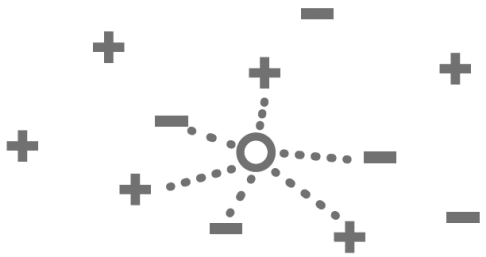

Figure 6.1. *k*-NN algorithm *[30]*.

6.1.2. Support Vector Machines

Support Vector Machines (SVM) classify data by finding a hyperplane (linear boundary) that separates all data points in one class from another. The best hyperplane is the one with the largest margin between the two classes when the data is linearly separable [30]. If the data cannot be separated linearly, a loss function is used to penalize the points on the wrong side of the hyperplane. Often, this type of algorithm uses a kernel transform to convert separable non-linear data into higher dimensions where a linear decision boundary can be found [30]. The largest margin between the two classes can also be thought of as the hyperplane that maximizes the distance between the classes using a limited number of samples called support vectors, which are closer to the separation margin [29]. Although SVMs are usually applied for two classes, these algorithms can be used for multiclass classification with a technique called error-correcting output codes [30]. This type of classifier is useful for non-linear separable high-dimensional data and when the system needs a simple easy-to-interpret and accurate classifier [30]. The representation of how an SVM works is presented below, the hyperplane separates the data points of two different classes.

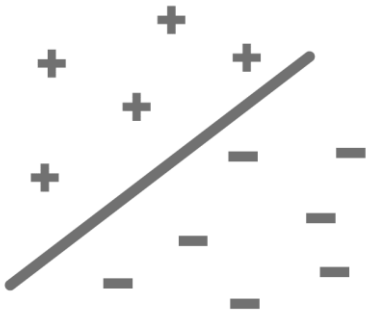

Figure 6.2. Support Vector Machine *[30]*.

6.1.3. Naive Bayes

Naive Bayes is a family of powerful and easy-to-train classifiers that determine the probability of an outcome given a set of conditions using Bayes' theorem [29]. This type of

classifier assumes that the presence of a particular feature is unrelated to the presence of another [30]. It works by classifying new data by calculating the probability that it belongs to a particular class and choosing the highest value. These classifiers are suitable for small datasets with many parameters and are easy to interpret [30].

Considering two probabilistic events A and B the Bayes' theorem is defined by the following expression:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{6.1}$$

where $P(B)$ is the marginal probability of B, and $P(A|B)$ and $P(B|A)$ are conditional probabilities. This is one of the fundamental equations used in statistical learning [29] and provides a weight between a prior and a posterior probability considering equation (6.1) in the general discrete case as:

$$P(A|B) = \frac{P(B|A)P(A)}{\sum_i P(B|A_i)P(A_i)}, \tag{6.2}$$

since the denominator is a normalization factor, this equation can be expressed as a proportionality relationship [29]:

$$P(A|B) \propto P(B|A)P(A). \tag{6.3}$$

Replacing the normalization factor with α, equation (6.3) becomes:

$$P(A|B) = \alpha P(B|A)P(A). \tag{6.4}$$

Considering the case when there are more concurrent conditions (more realistic approach for real-life problems):

$$P(A|C_1 \cap C_2 \cap \ldots \cap C_n). \tag{6.5}$$

The problem becomes extremely complex if the joint probability is considered; instead, it is easier to consider the impact of individual factors, but the problem can become intractable expressed as:

$$P(A|C_1 \cap C_2 \cap \ldots \cap C_n) = \alpha P(C_1 \cap C_2 \cap \ldots \cap C_n|A)P(A). \tag{6.6}$$

Therefore, a common assumption of conditional independence allows simplifying the previous expression as:

$$P(A|C_1 \cap C_2 \cap \ldots \cap C_n) = \alpha P(C_1|A)P(C_2|A) \ldots P(C_n|A)P(A). \quad (6.7)$$

It is easier to compute the singles $P(C_i|A)$ and multiply them, instead of finding the joint probability [29].

The adjective "naive" has been attributed not because these algorithms are limited or less efficient, but because of the fundamental assumption of the previous equation. This naive condition implies the conditional independence of causes (events, features, parameters, etc.). In many contexts, this assumption can be difficult to accept because the probability of one particular feature could be strictly correlated with another [29]. However, under particular conditions (which are not so rare to happen), the different dependencies clear one another, causing that the Naive Bayes classifier succeeds in achieving high performances even if the conditional independence is not assured [29].

In general, naive Bayes models are very fast to train and to predict, also they work well with high-dimensional sparse data and are relatively robust [32]. The following figure shows a graphic representation of how these algorithms work.

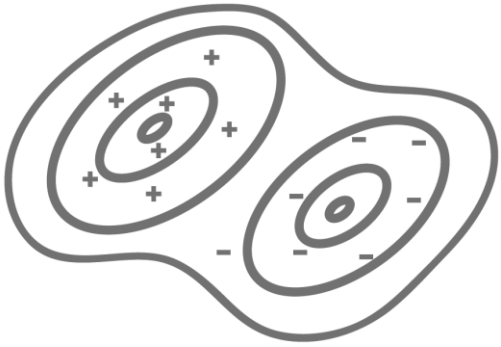

Figure 6.3. Naive Bayes *[30]*.

6.1.4. Discriminant Analysis

The discriminant analysis finds linear combinations of features to classify them; it assumes that the different classes generate data based on Gaussian distributions [30]. The training process consists of finding parameters for a Gaussian distribution that describes each class, this distribution acts as boundaries and can be a linear or quadratic function [30]. These boundaries determine the class of new data. These models are simple and easy to interpret, they can also improve memory use in the training process and are fast models to predict [30]. The following figure presents a quadratic discriminant analysis as an example of how this algorithm works.

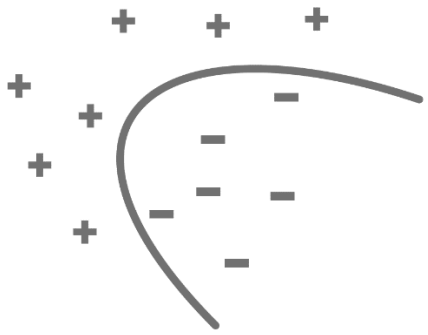

Figure 6.4. Discriminant Analysis (quadratic version) *[30]*.

The Linear Discriminant Analysis (LDA) finds the projection hyperplane that minimizes the variance between classes and maximizes the distance between the projected means of the classes. These two objectives can be achieved by solving an eigenvalue problem with the corresponding eigenvector that defines the hyperplane of interest. This characteristic enables fast and massive processing of data samples. The intuition behind the method is to determine a subspace of lower dimension compared to the original data, in which the data points of the original problem are "separable". This behavior is represented in Figure 6.5 where a two-dimensional set of samples is projected on a line (one dimension), this line is chosen so that the projection maximizes the "separability" of the projected samples. The separability is defined in terms of the statistical measures: mean and variance. This hyperplane can be used for classification, dimensionality reduction, and interpretation of given features [33].

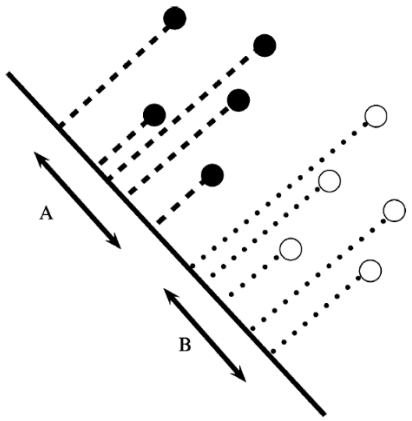

Figure 6.5. How LDA works, example of two-dimensional data samples projected on a line (lower dimension) *[33]*.

6.1.5. Decision Trees

The decision tree is one of the most popular classification models because it can predict responses by following the decisions in a tree, a path from the root (beginning) to a leaf node [30]. The tree consists of branching conditions in which the value of a predictor is

compared to a trained weight. The total number of branches and weight values are determined during the training process. In other words, a decision tree learns a hierarchy of if/else questions, leading to a decision [32]. Advantages of this model include ease of interpretation, fast to fit, and minimal memory usage [30]. Compared to other algorithms, a decision tree classifier can be easily visualized and understood by nonexperts (at least for smaller trees) and is invariable to scaling of the data [32]. However, even when using techniques to manage the number of nodes, such as pre-pruning, these algorithms tend to overfit and provide poor generalization performance, this is why in most applications, an ensemble of decision trees is used instead of a single tree [32]. The following figure presents a decision tree to represent the main operation of this type of model.

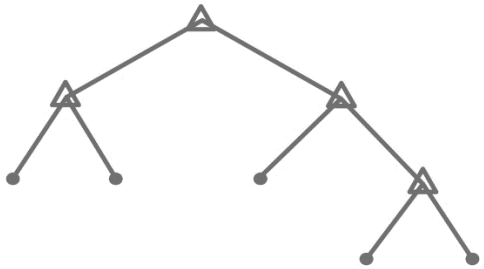

Figure 6.6. Decision Tree *[30]*.

6.1.6. Ensemble Classifiers

As the name suggests, ensemble classifiers apply several "weaker" models combined into one "stronger" ensemble [30]. The types of ensembles most used in machine learning are bagging and boosting, we also talk about the random subspace method. They all modify the training data set, build classifiers on these modified training sets, and then combine them into a final decision rule by simple or weighted majority voting [34]. Below is a description of each type of ensemble.

6.1.6.1. Bagging

It was proposed by [35] and is based on a combination of the concepts: bootstrapping and aggregating. Bootstrapping means selecting a random set of samples from the input data with replacement [34].

When doing a bootstrap replicate as follows:

$$X^b = \left(X_1^b, X_2^b, \dots, X_n^b\right), \tag{6.8}$$

the training set $X = (X_1, X_2, \dots, X_n)$ can prevent or avoid less misleading training objects in the bootstrap set. Consequently, a classifier with this new set may have performed better

[34]. Aggregating means combining classifiers. In general, when taking a bootstrap sample from the training set, approximately $1/e \approx 37\%$ of the objects are not presented in the sample, which means that possible 'outliers' in the training set sometimes do not appear in the bootstrap sample. This creates better classifiers (with a smaller classification error in the training data set) when using the bootstrap samples instead of the original training set. Therefore, they will be more decisive than other bootstrap versions in the final judgment. Aggregating classifiers in bagging can sometimes provide better performance than individual classifiers. Bagging can incorporate the benefits of both approaches (bootstrapping and aggregating) by combining the advantages of the individual classifiers in the final solution [34]. According to [36], bagging performance improves if used with an unstable learner, which can be defined as a learner causing significant changes by disrupting the training set.

For example, a bagged decision tree consists of trees trained independently on bootstrapped data from the input data [30]. These types of ensembles are also called Random Forests. The idea behind random forests is that each tree could do a relatively good job of predicting, but most likely overfitting some of the data. By implementing many trees, each working well in different ways, the amount of overfitting can be reduced by averaging the results. The name random comes from the randomness injected into the tree building to ensure that each tree is different [32]. The following figure is a representation of a bagged decision tree.

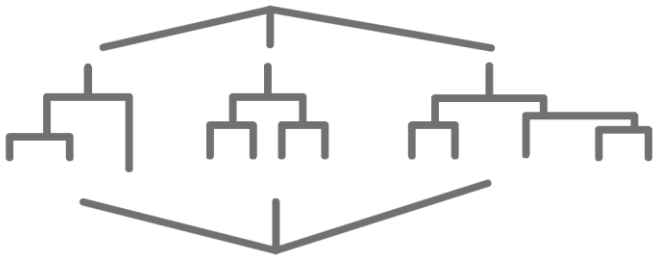

Figure 6.7. Bagged decision tree *[30]*.

6.1.6.2. Boosting

First proposed in [37], boosting is another technique to combine weak learners who perform poorly on a classifier with a classification rule that provides better performance [34]. Unlike bagging, which uses random subsets of data independent of other steps in the algorithm; the classifiers and training sets in boosting are obtained in a sequential deterministic way. Boosting creates a strong learner by iteratively adding "weak" learners and adjusting the weight of each by focusing on misclassified examples [30]. At each step of boosting, the training data is reweighed such that incorrectly classified objects get larger

weights in a new, modified training set. This implies that it maximizes the margins between training objects [34].

6.1.6.3. Random Subspace

The Random Subspace Method (RSM) was proposed by [38], it is a technique that modifies training data in feature space. For each training object $X_i(i = 1,2, \dots, n)$ in the training sample set $X = (X_1, X_2, \dots, X_n)$ defined as a p-dimensional vector $X_i = (x_{i1}, x_{i2}, \dots, x_{ip})$ described by p features, the RSM randomly selects r features $r < p$ from the p-dimensional data set X. This is considered as the r-dimensional random subspace of the original p-dimensional feature space. The modified training set $\hat{X}^b = (\hat{X}_1^b, \hat{X}_2^b, \dots, \hat{X}_n^b)$ consists of r-dimensional training objects $\hat{X}_i^b = (x_{i1}^b, x_{i2}^b, \dots, x_{ir}^b)(i = 1,2, \dots, n)$ where r components $x_{ij}^b(j = 1,2, \dots, r)$ are randomly selected from p components $x_{ij}(j = 1,2, \dots, p)$ of the training vector X_i (the selection is the same for each training vector). The classifiers are constructed on the random subspaces $\hat{X}^b$ and combined by simple majority voting in the final decision rule [34].

Another advantage of this method is that it can benefit from the use of random subspaces for both constructing and aggregating the classifiers. If the number of training objects is relatively small compared to the data dimensionality, building classifiers in random subspaces can probably solve the small sample size problem. The subspace dimensionality is smaller than in the original feature space, while the number of training objects remains the same. Therefore, the relative size of the training sample increases. When the data has many redundant features, it is possible to obtain better classifiers in random subspaces than in the original feature space. The combined decision of such classifiers may be superior to a single classifier constructed on the original training set in the complete feature space [34].

6.2. COMPARISON OF SUPERVISED MACHINE LEARNING ALGORITHMS

The following table summarizes and compares the main characteristics for each supervised machine learning algorithm included in this chapter.

Table 6.1. Comparison of supervised machine learning algorithms [39].

Algorithm	Prediction Speed	Training Speed	Memory Usage	Description
k-Nearest Neighbors	Moderate	Fast	Medium	Easy interpretation and implementation
Linear SVM	Fast	Fast	Small	Good for small linear problems
Non-linear SVM	Slow	Slow	Medium	Good for high-dimensional data
Naive Bayes	Fast	Fast	Medium	Assumes that the Bayes' theorem is satisfied
Discriminant Analysis	Fast	Fast	Small	Simple and easy to interpret
Decision Tree	Fast	Fast	Small	Prone to overfitting
Ensemble Classifier	Moderate	Slow	Varies	High accuracy for small or medium datasets

CHAPTER 7 . FINGERPRINT RECOGNITION SYSTEM IN MATLAB

From the input fingerprint databases [6, 7, 8], the preprocessing applied to these images allows to extract features from different fingerprints characteristics using texture descriptors, statistic measures, and different transformations (DWT, FFT, DCT) creating a set of 170 features for each fingerprint image complied in a table in a columns wise manner. Each row is labeled with a categorical variable indicating the number of the subject to which the current fingerprint belongs. This procedure has been performed for each subset (four for each database) in the three main databases.

Several supervised algorithms have been tested using a native application called Classification Learner, which is part of the Statistics and Machine Learning Toolbox in MATLAB. This application trains models to classify data using various classifiers. Provides the ability to explore input data, select features, specify validation schemes, train models, and evaluate results. The application performs automated training that helps to find the best type of classification model among the following models: decision trees, discriminant analysis, Support Vector Machines (SVM), nearest neighbors, naive Bayes, and ensemble classifiers [31].

7.1. CLASSIFICATION LEARNER

This application has a graphical user interface that allows for simple iterative training and evaluation of the supervised machine learning models. In this case, all the algorithms available in the MATLAB R2019b version were trained to obtain several results and evaluate each one. The first step is to create a new session by loading the corresponding table for each subset. A table of size 80x75 is charged in the application, the actual size of the feature table is 80x171, this value differs from the data shown in the figure because MATLAB counts multiple columns in the table as one, for example, the CONTRASTS feature is four columns, but the software counts it as one. The following figure presents the new session window.

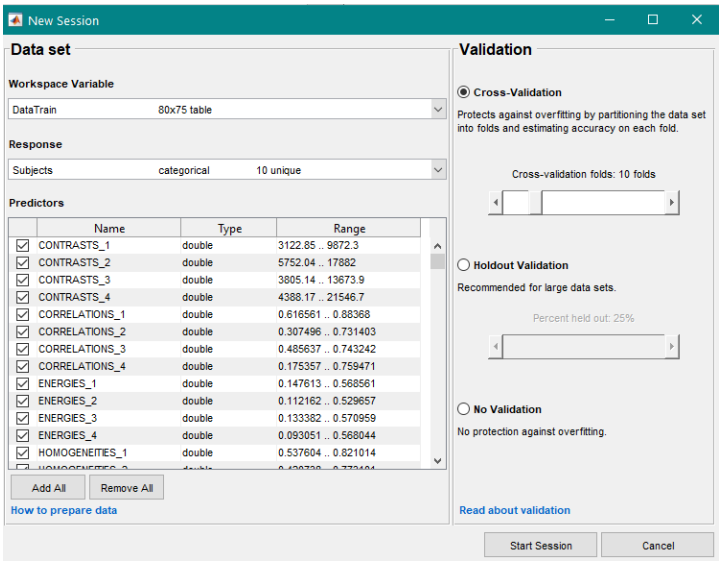

Figure 7.1. New session in the classification learner app.

An important characteristic that must be defined in this step is the Validation option. In this work, it was decided to use Cross-Validation, since it protects against overfitting and generalizes the reported results by averaging the accuracies obtained in the folds. 10-fold cross-validation was implemented since it is the most frequently used approach in the literature. The Predictors section in the previous window allows choosing the predictors (features) that should be used for the classification application, in this case, all the available features were selected. After clicking the Start Session button, the next window will present a scatter plot of the first two features in the set of predictors, as can be seen in the following figure.

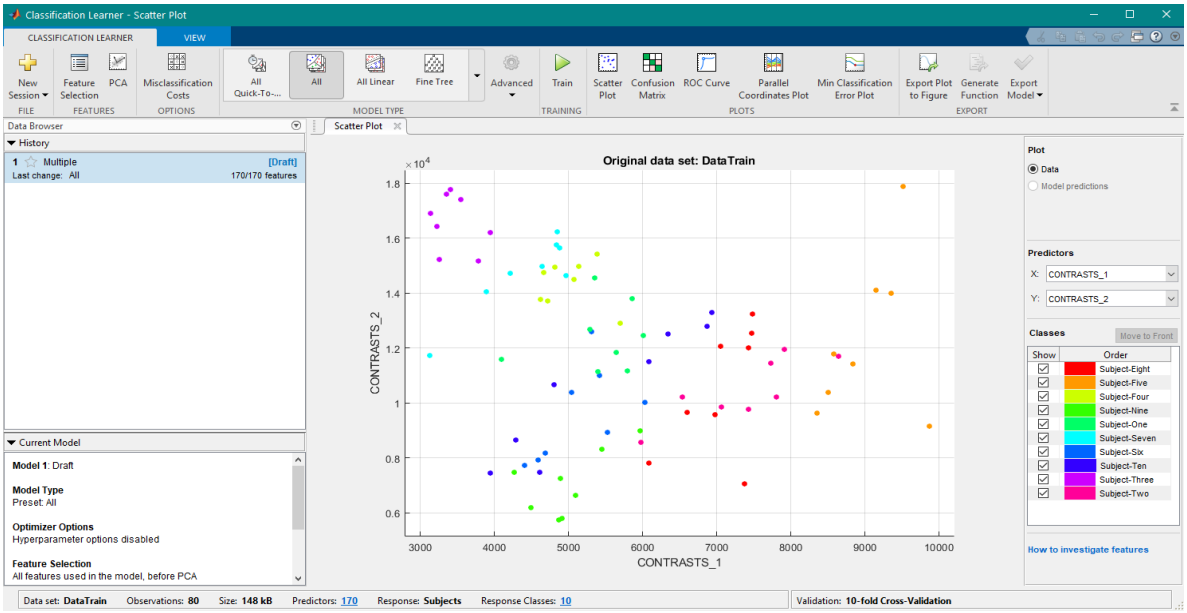

Figure 7.2. Scatter plot of the first two predictors.

On the Model Type section, the option “All” was selected, which means that all available models will be trained. Another interesting characteristic of this application is the possibility to visualize the current predictors and the expected output, on the right side of the window in the Classes section these data samples can be turned on and off so that the main clusters can be easily visualized. In the lower part of the window the main configuration parameters are reported, including the number of observations (80, since we have subsets with 10 different subjects each one with 8 fingerprint impressions), the number of predictors (170, the same number of extracted features per fingerprint), and the response classes (10, because there are only 10 possible subjects). The last step is to press the Train button and the training process will start, a total of 24 training models are executed sequentially in the application. The next section reports the main results for each database.

The following flowchart presents a summary of how the Classification Learner application works [40]:

Figure 7.3. Classification Learner application flowchart *[40]*.

This flowchart represents the common workflow for training classification models or classifiers in this application [40]. For the evaluation of classifiers, in this work, the confusion matrix and accuracy value were used.

7.2. RESULTS FOR THE FVC2000 DATABASE

The FVC2000 database [6] has four subsets called DB1, DB2, DB3, and DB4. The following images report the best accuracy obtained for the DB1_FVC2000, DB2_FVC2000, DB3_FVC2000, and DB4_FVC2000 subsets, respectively.

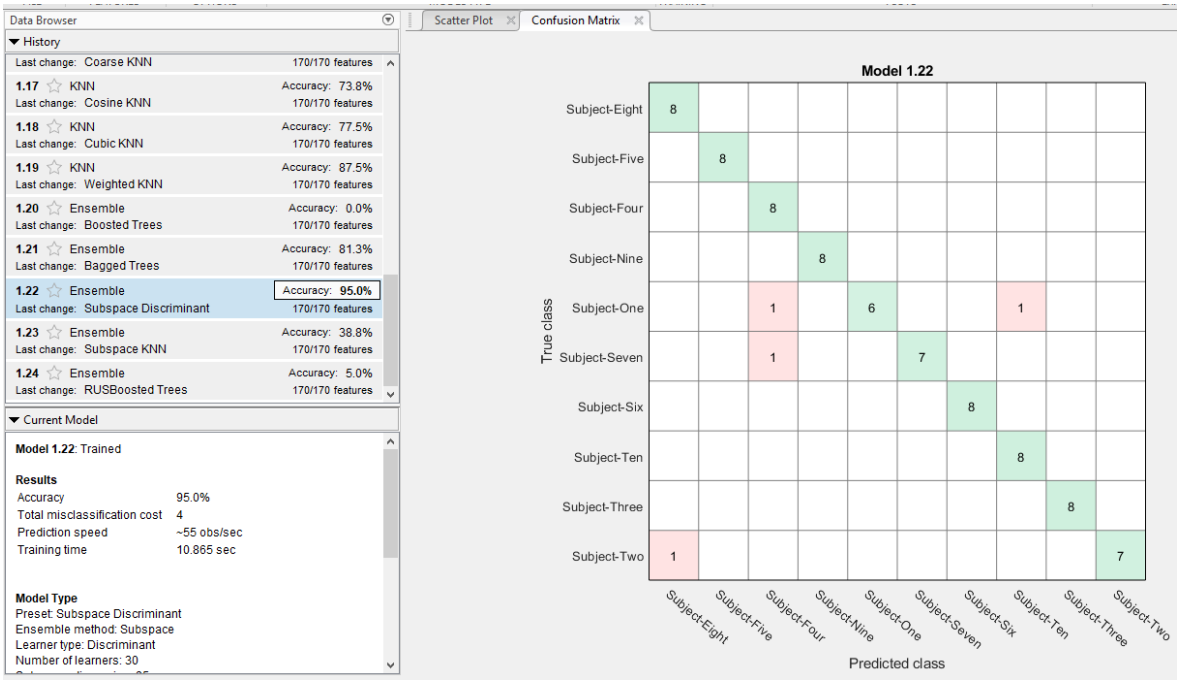

Figure 7.4. Highest accuracy for the DB1_FVC2000 subset.

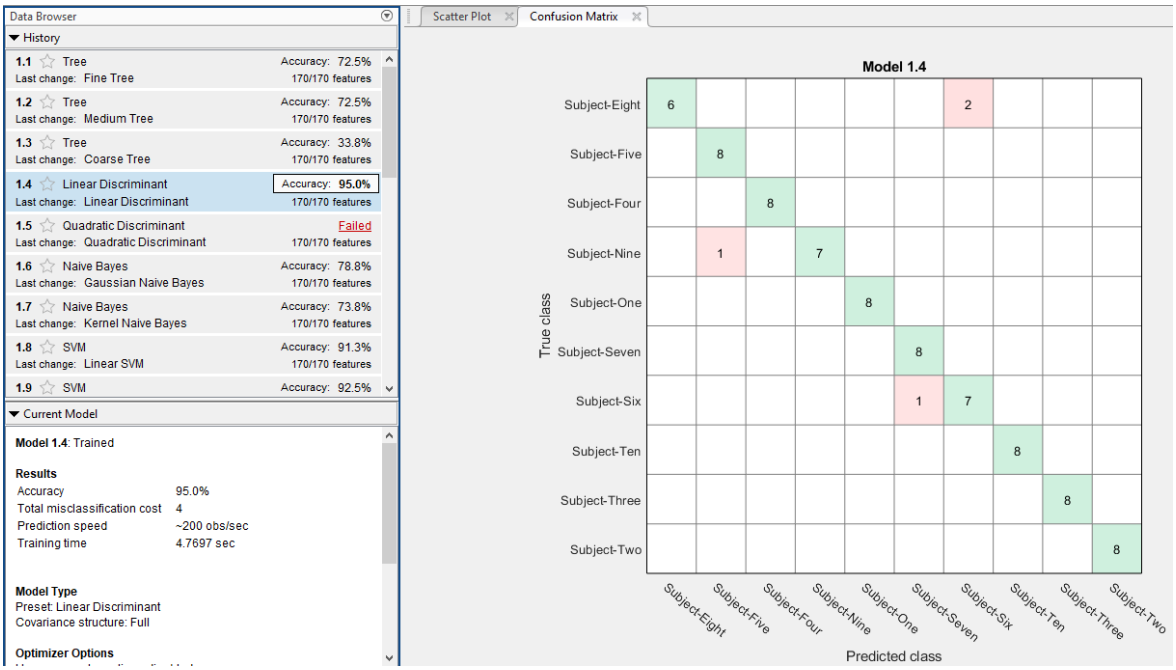

Figure 7.5. Highest accuracy for the DB2_FVC2000 subset.

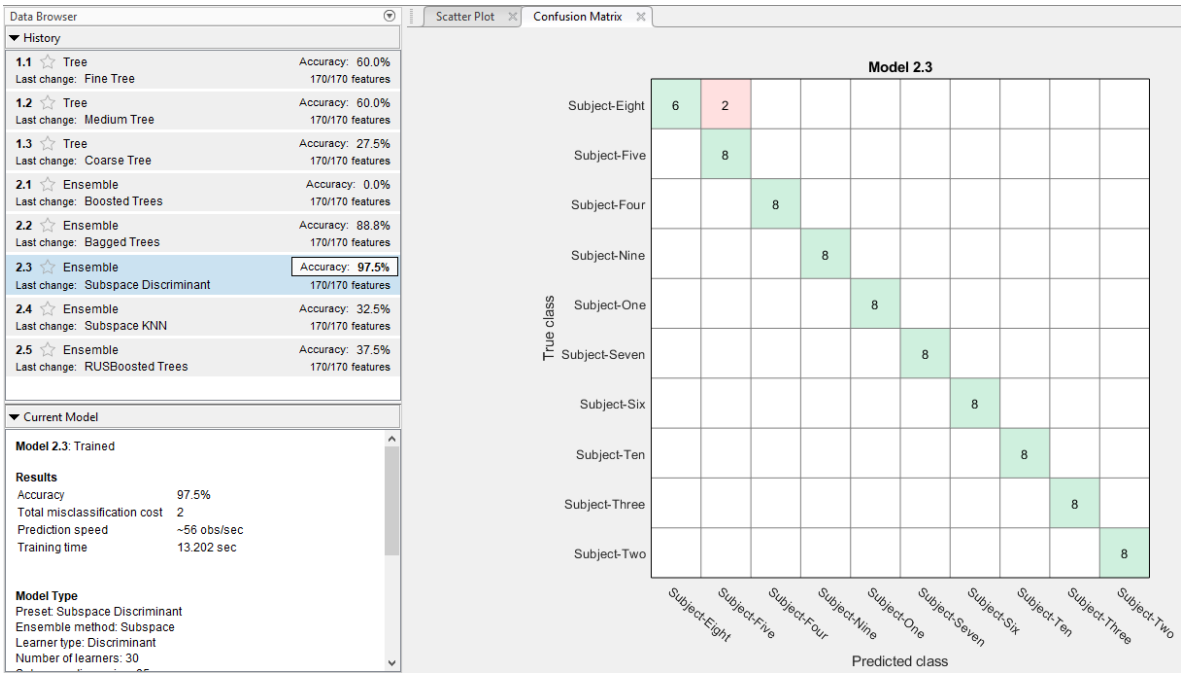

Figure 7.6. Highest accuracy for the DB3_FVC2000 subset.

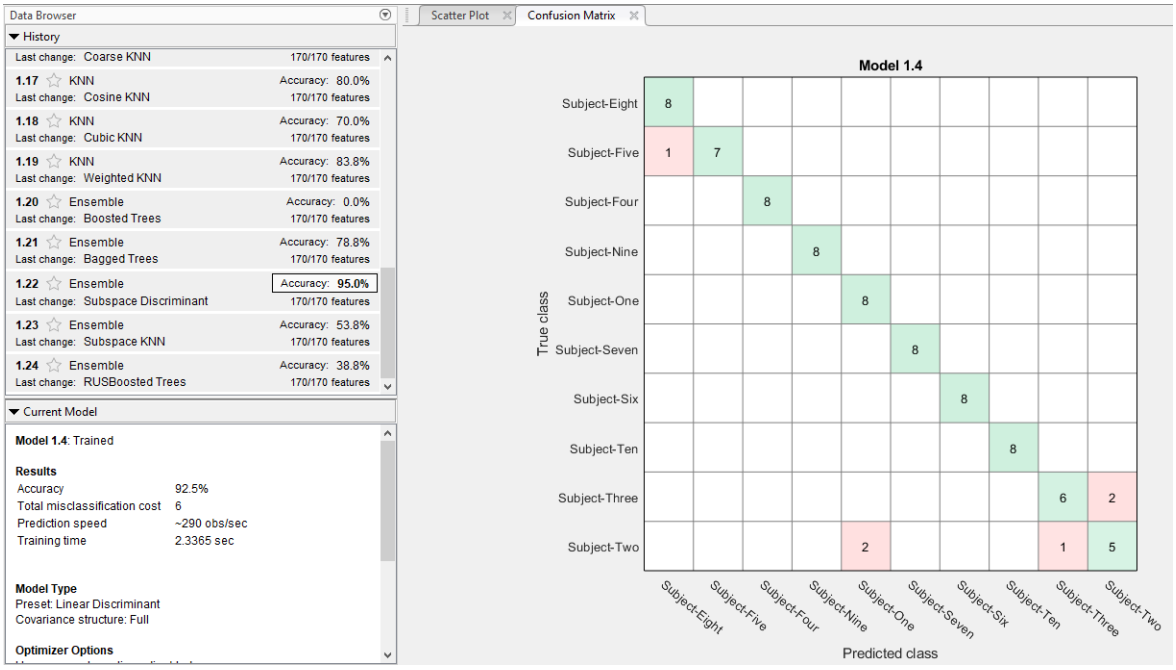

Figure 7.7. Highest accuracy for the DB4_FVC2000 subset.

These results show a high accuracy for recognition in the four subsets. The confusion matrix is also included for each case showing the correct classification in green and the misclassifications in red.

7.3. RESULTS FOR THE FVC2002 DATABASE

The FVC2002 database [7] has four subsets called DB1, DB2, DB3, and DB4. The following images report the best accuracy obtained for the DB1_FVC2002, DB2_FVC2002, DB3_FVC2002, and DB4_FVC2002 subsets, respectively.

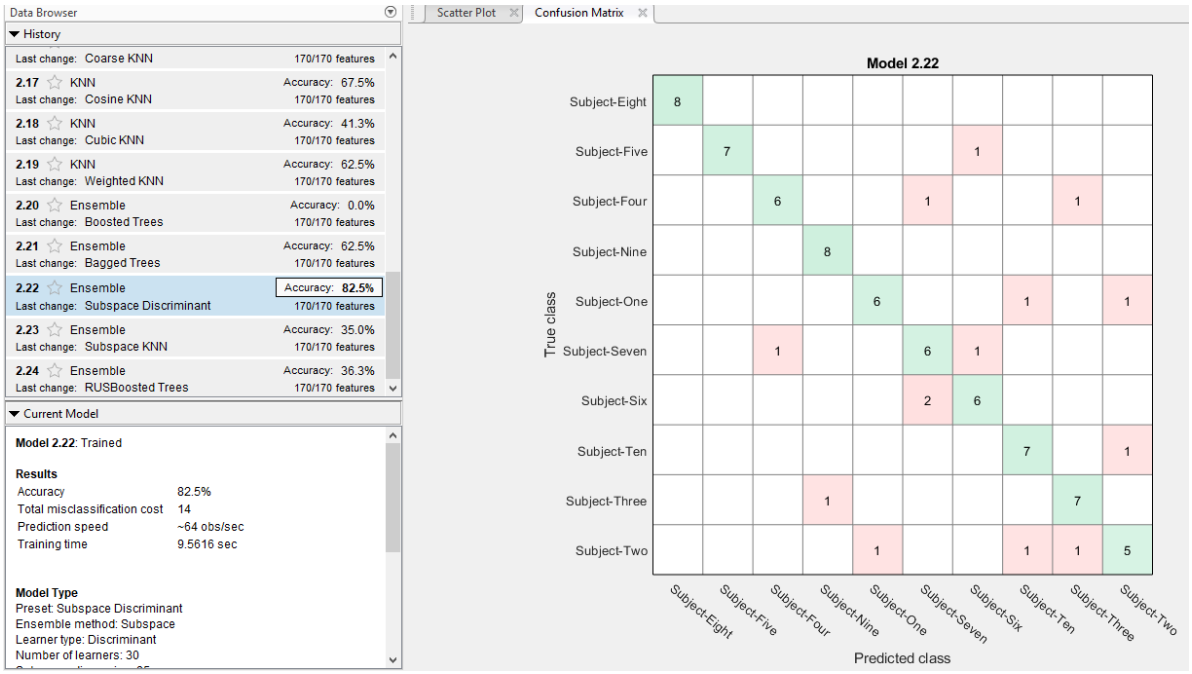

Figure 7.8. Highest accuracy for the DB1_FVC2002 subset.

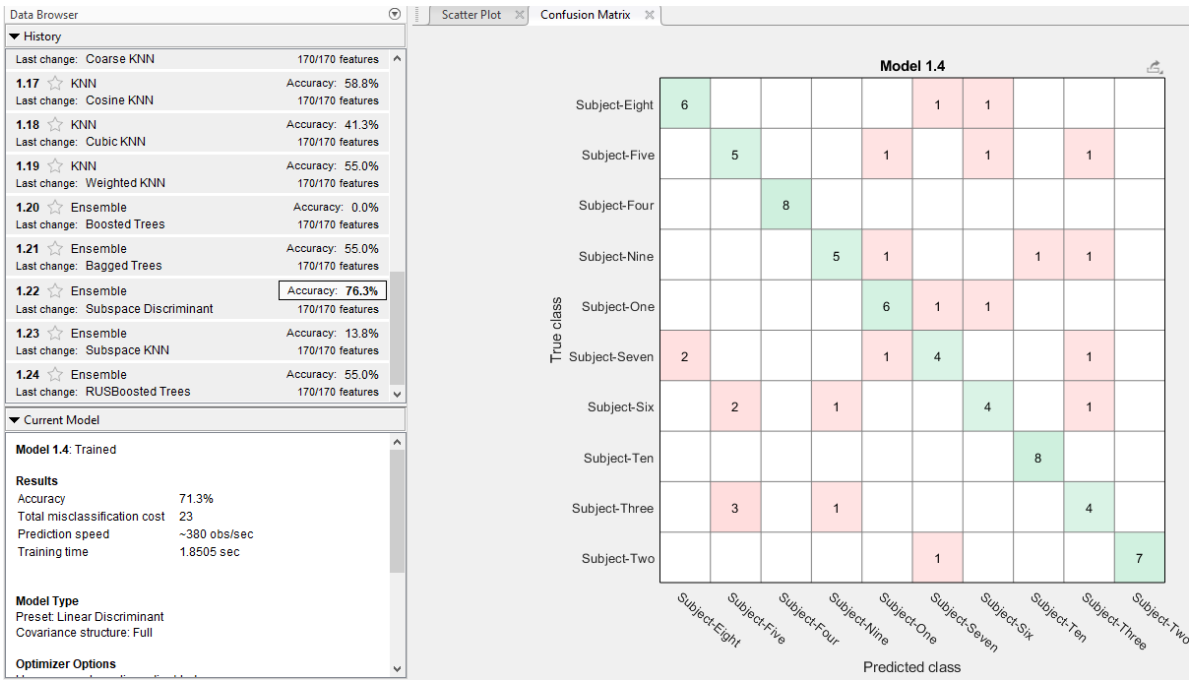

Figure 7.9. Highest accuracy for the DB2_FVC2002 subset.

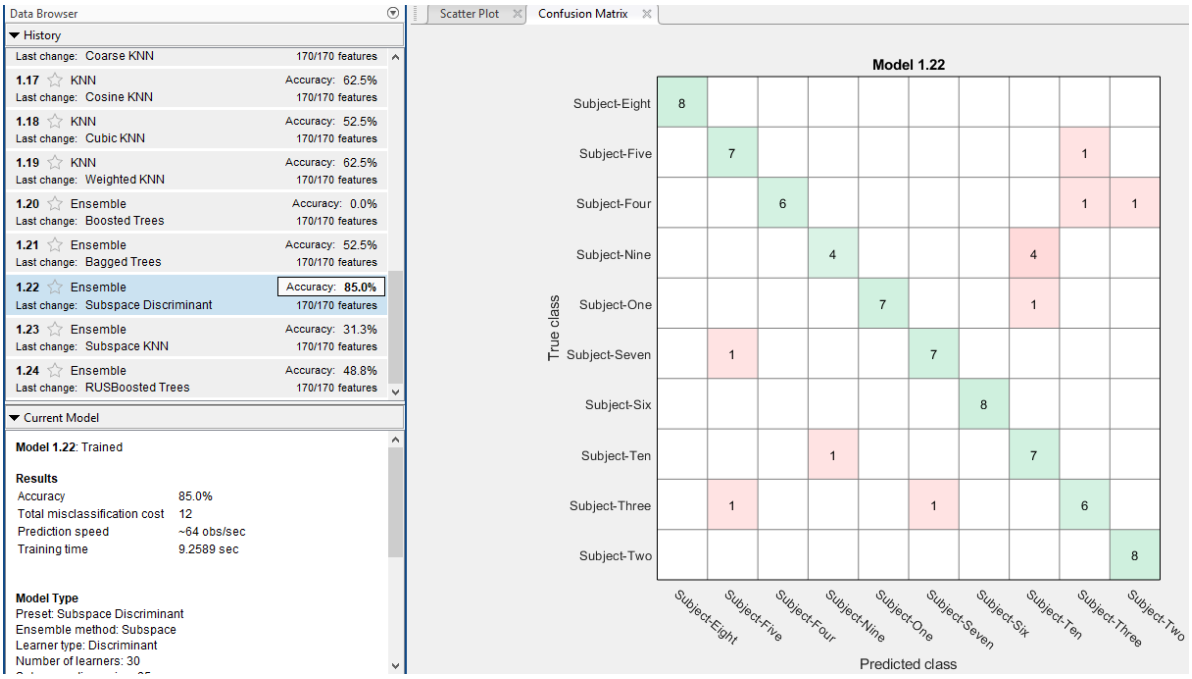

Figure 7.10. Highest accuracy for the DB3_FVC2002 subset.

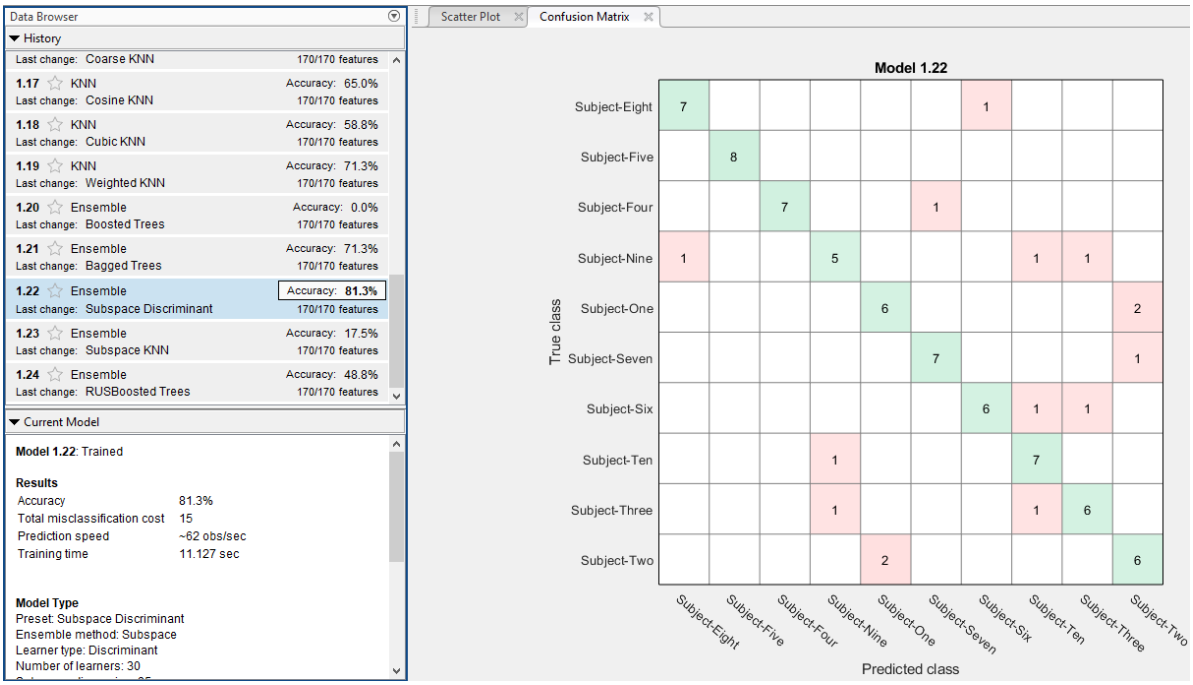

Figure 7.11. Highest accuracy for the DB4_FVC2002 subset.

These results do not show accuracy values as high as for the previous database, the FV2002 database is significantly more difficult to recognize.

7.4. RESULTS FOR THE FVC2004 DATABASE

The FVC2004 database [8] has four subsets called DB1, DB2, DB3, and DB4. The following images report the best accuracy obtained for the DB1_FVC2004, DB2_FVC2004, DB3_FVC2004, and DB4_FVC2004 subsets, respectively.

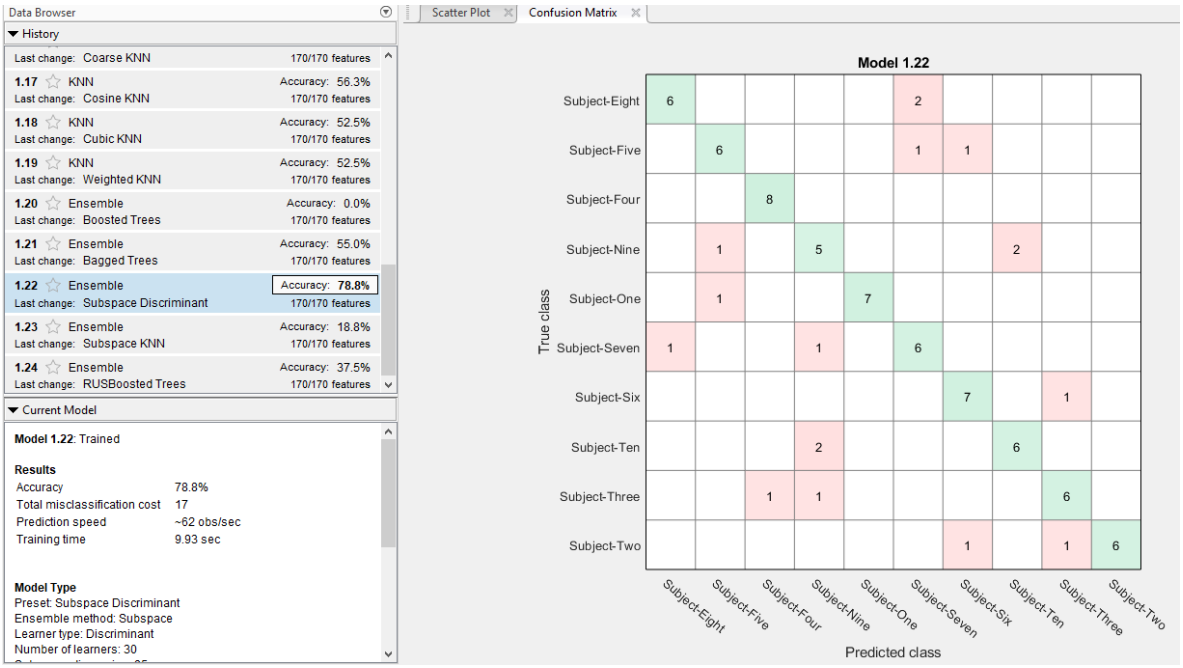

Figure 7.12. Highest accuracy for the DB1_FVC2004 subset.

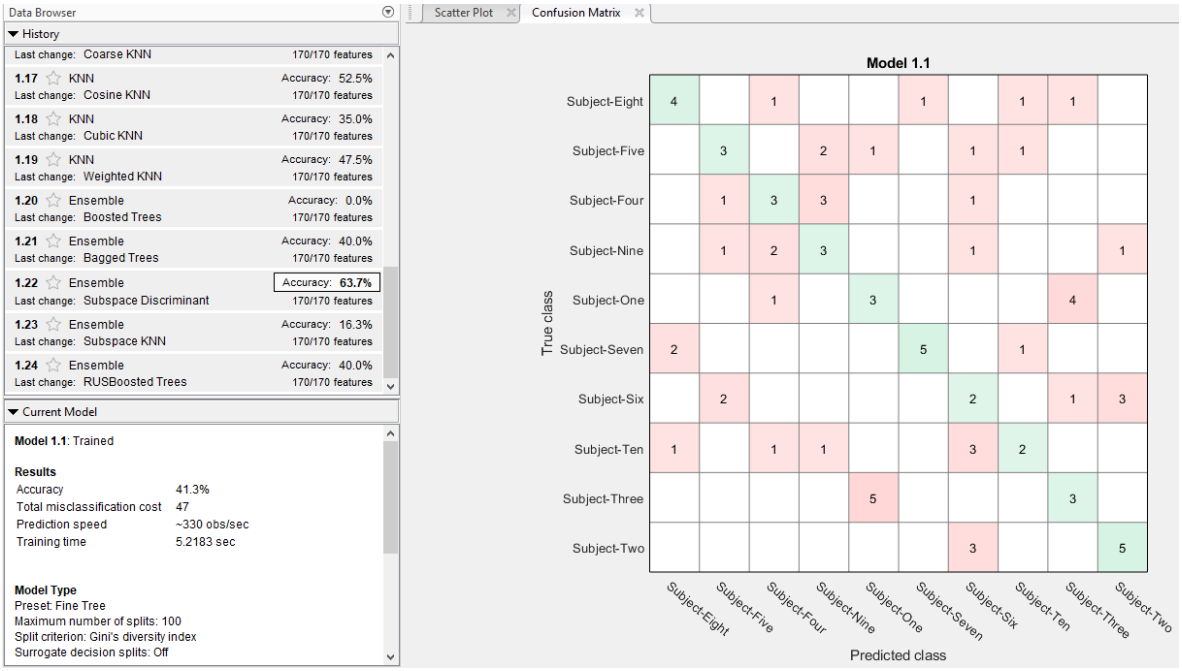

Figure 7.13. Highest accuracy for the DB2_FVC2004 subset.

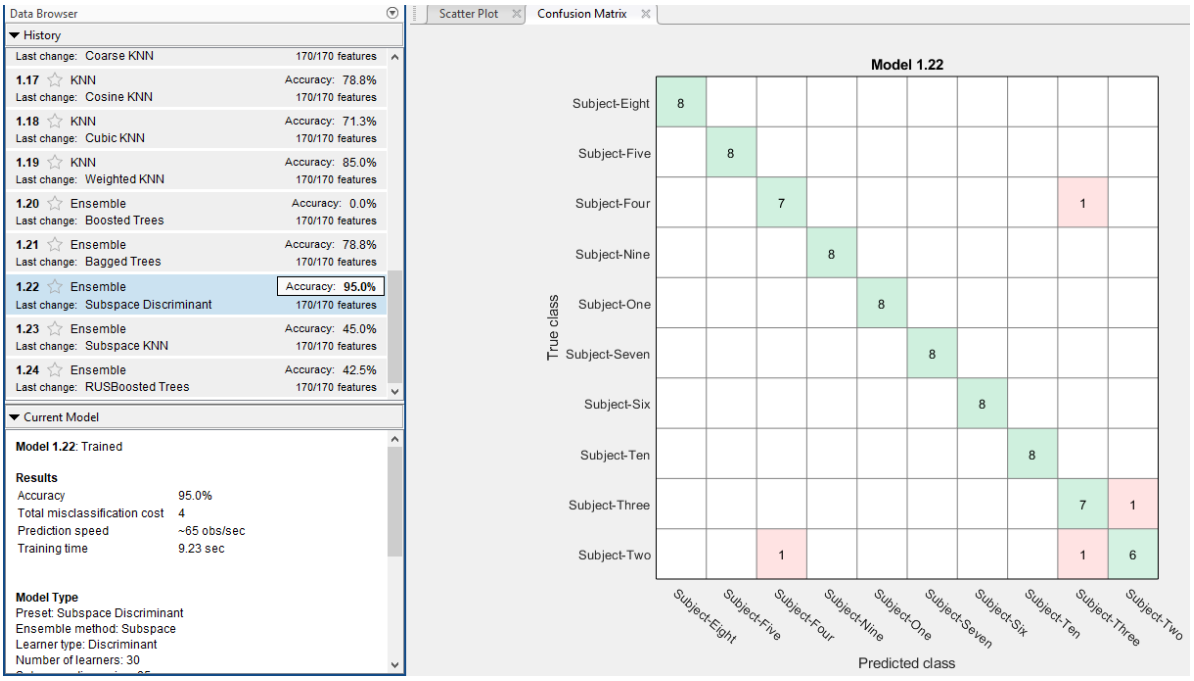

Figure 7.14. Highest accuracy for the DB3_FVC2004 subset.

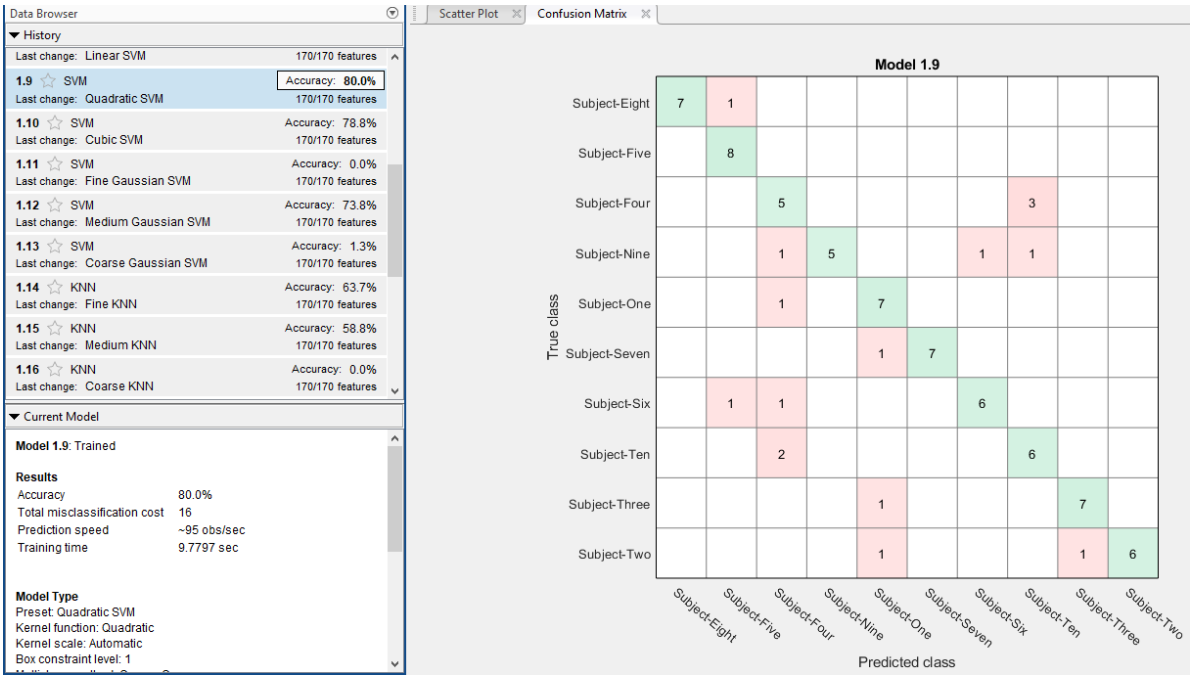

Figure 7.15. Highest accuracy for the DB4_FVC2004 subset.

In contrast with the results of FVC2002, for some subsets in FVC2004, the accuracy is higher, however, for others, it is lower.

7.5. DISCUSSION

The following table summarizes the highest accuracy values obtained for the twelve subsets analyzed in this work.

Table 7.1. Results of the Classification Process.

Database	Model Type	Accuracy
DB1_FVC2000	Ensemble: Subspace Discriminant	95 %
DB2_FVC2000	Linear Discriminant	95 %
DB3_FVC2000	Ensemble: Subspace Discriminant	97.5 %
DB4_FVC2000	Ensemble: Subspace Discriminant	95 %
DB1_FVC2002	Ensemble: Subspace Discriminant	82.5 %
DB2_FVC2002	Ensemble: Subspace Discriminant	76.3 %
DB3_FVC2002	Ensemble: Subspace Discriminant	85 %
DB4_FVC2002	Ensemble: Subspace Discriminant	81.3 %
DB1_FVC2004	Ensemble: Subspace Discriminant	78.8 %
DB2_FVC2004	Ensemble: Subspace Discriminant	63.7 %
DB3_FVC2004	Ensemble: Subspace Discriminant	95 %
DB4_FVC2004	SVM: Quadratic SVM	80 %

Ensemble: Subspace Discriminant is the classifier model that produces the highest accuracy values for ten of the twelve subsets used for fingerprint recognition. In MATLAB, the term Subspace indicates the use of random subspace ensembles, and Discriminant implies that this ensemble improves the accuracy of discriminant analysis classifiers. The function *fitcensemble* was applied and its parameters were automatically calculated by the Classification Learner application based on some characteristics of the input table data such as the number of predictors, etc. The *fitcensemble* syntax for ensemble subspace discriminant classifiers is [41]:

```
classificationEnsemble=fitcensemble(predictors,response,'Method','
Subspace','NumLearningCycles',30,'Learners','discriminant','NPredT
oSample',85,'ClassNames',labels)
```

where *predictors* is the input table, *response* is the final column of the input table (the target data or the expected response from the classification), '*Method'* specifies the current ensemble approach (in this case is the subspace), '*NumLearningCycles'* is the number of ensemble learning cycles, in this case, we specified 30 cycles. The '*Learners'* parameter specifies the type of learners that will be used in the ensemble, in this case, is discriminant. '*NPredToSample'* is the number of predictors to sample for each random subspace learner, in this case, we specified 85 because it is half the total number of features (170). Finally, '*ClassNames'* specifies the cell array with information about the names of the expected responses (Subject-One, Subject-Two, etc.) [41].

This approach was analyzed since it is the model that produces the highest accuracy values. The range of accuracy values goes from 63.7% to 97.5%, this indicates that the subsets in each database have several differences that can be better understood by analyzing their corresponding scatter plots. Using the application, it is possible to have a graphical representation of how each subset behaves for the first two features which correspond to the first two contrasts. This process could be done with any other feature in the subsets, but always comparing the same two features selected for all subsets. The scatter plots for each subset are presented below.

7.5.1. FVC2000

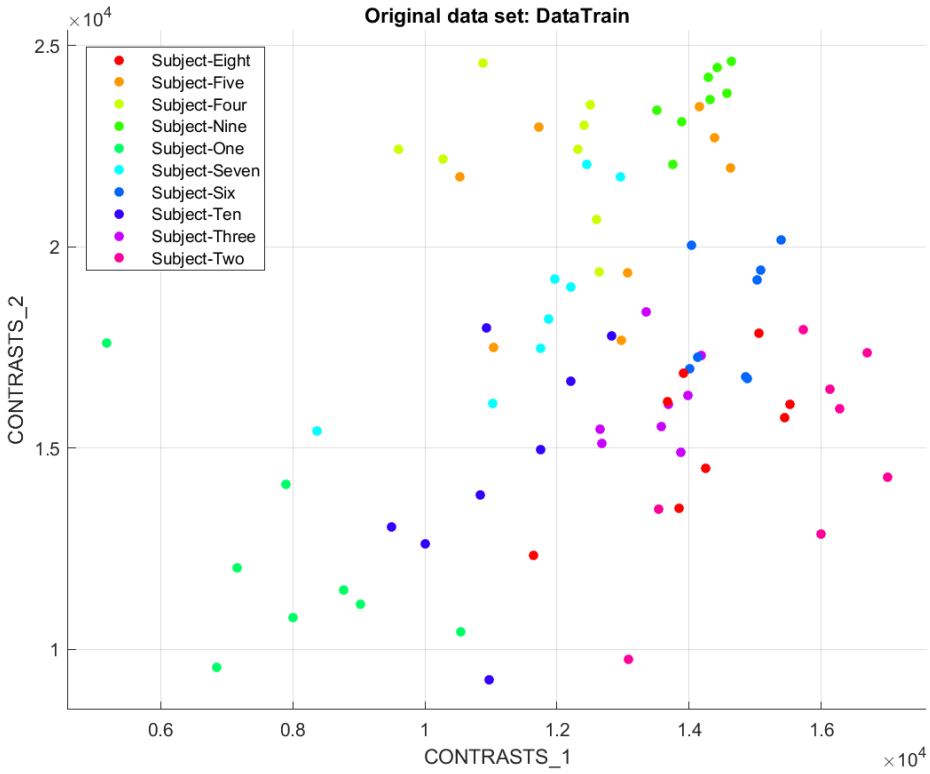

Figure 7.16. Scatter plot for the DB1_FVC2000 subset.

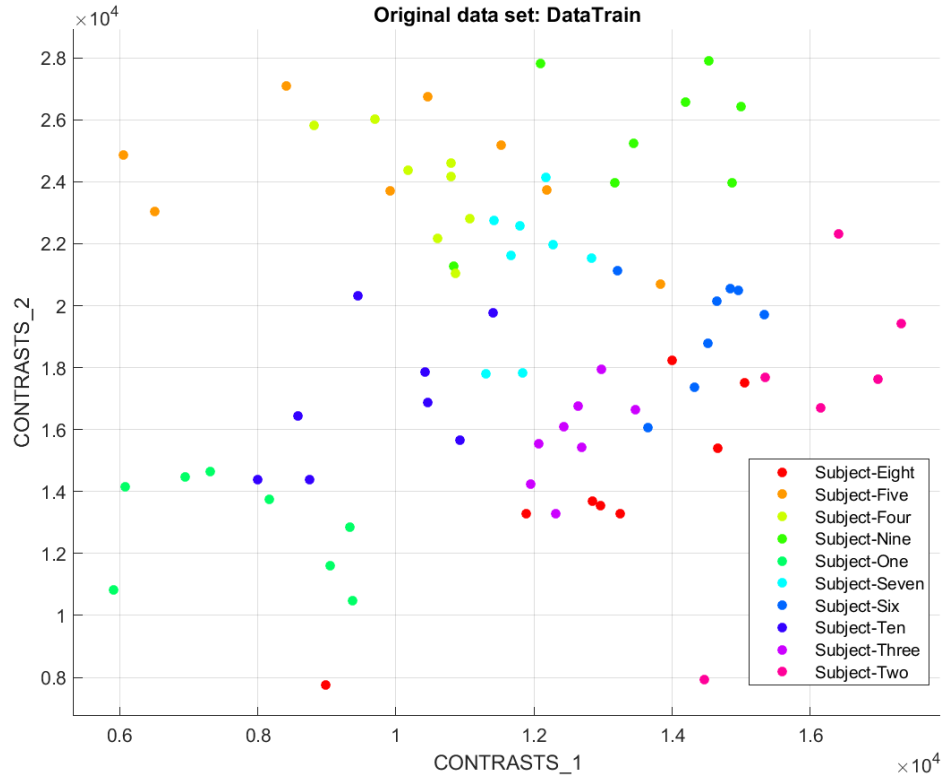

Figure 7.17. Scatter plot for the DB2_FVC2000 subset.

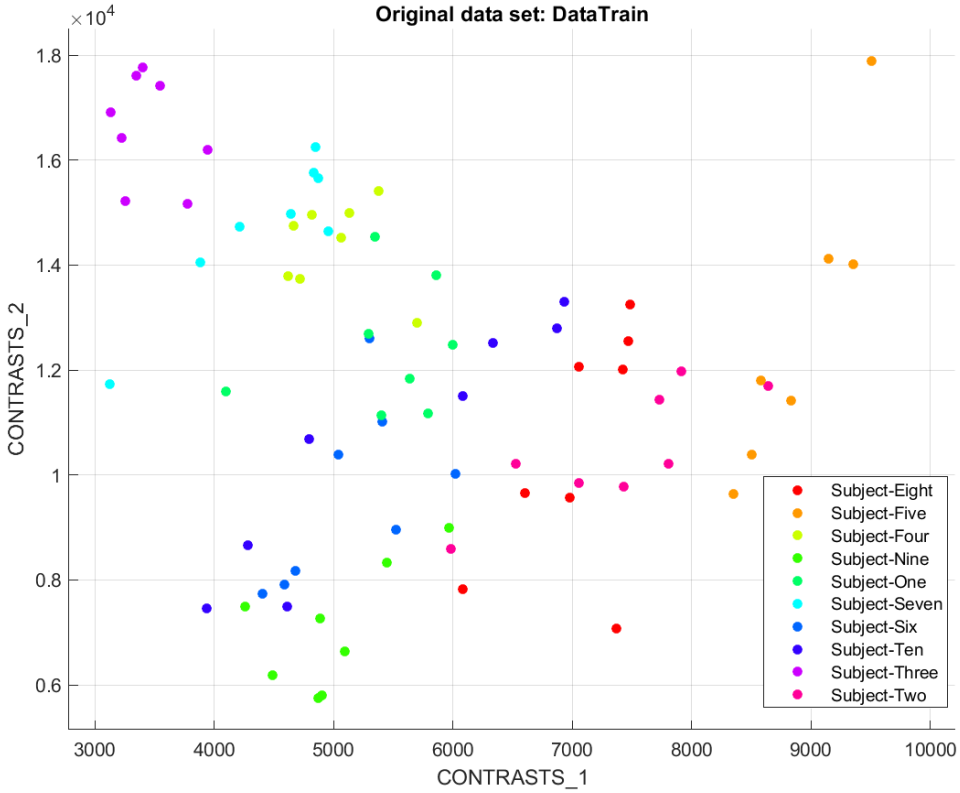

Figure 7.18. Scatter plot for the DB3_FVC2000 subset.

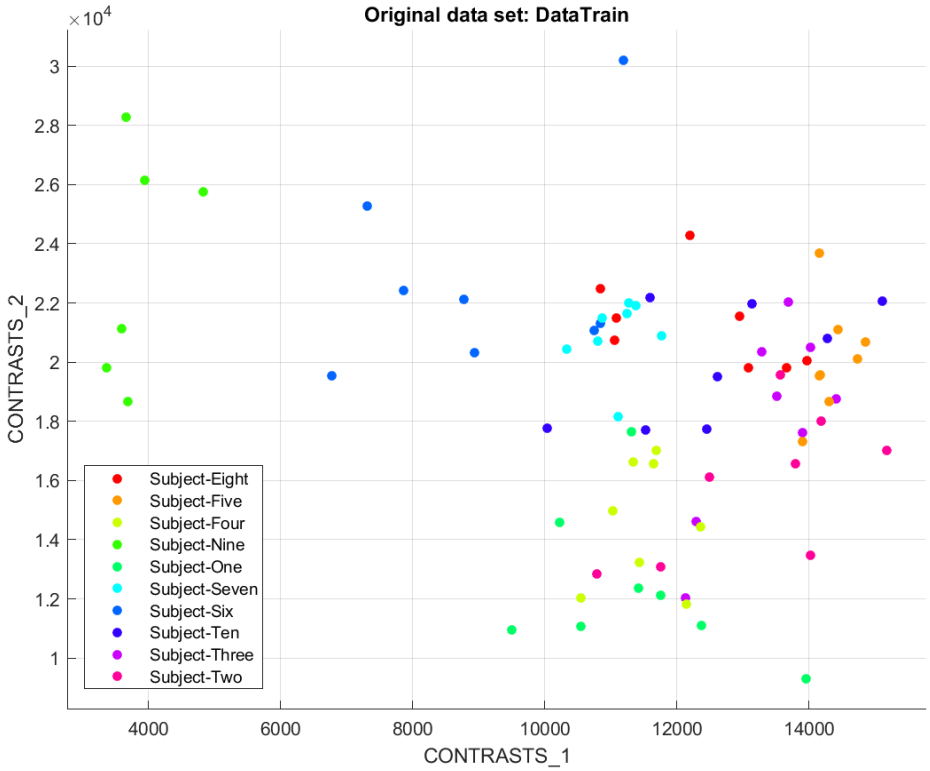

Figure 7.19. Scatter plot for the DB4_FVC2000 subset.

7.5.2. FVC2002

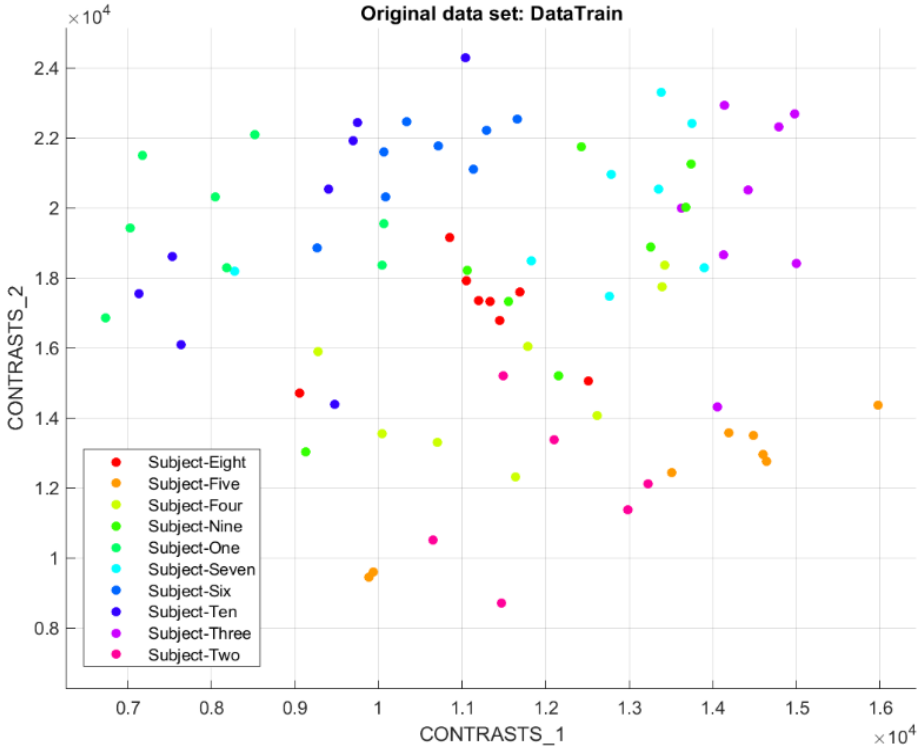

Figure 7.20. Scatter plot for the DB1_FVC2002 subset.

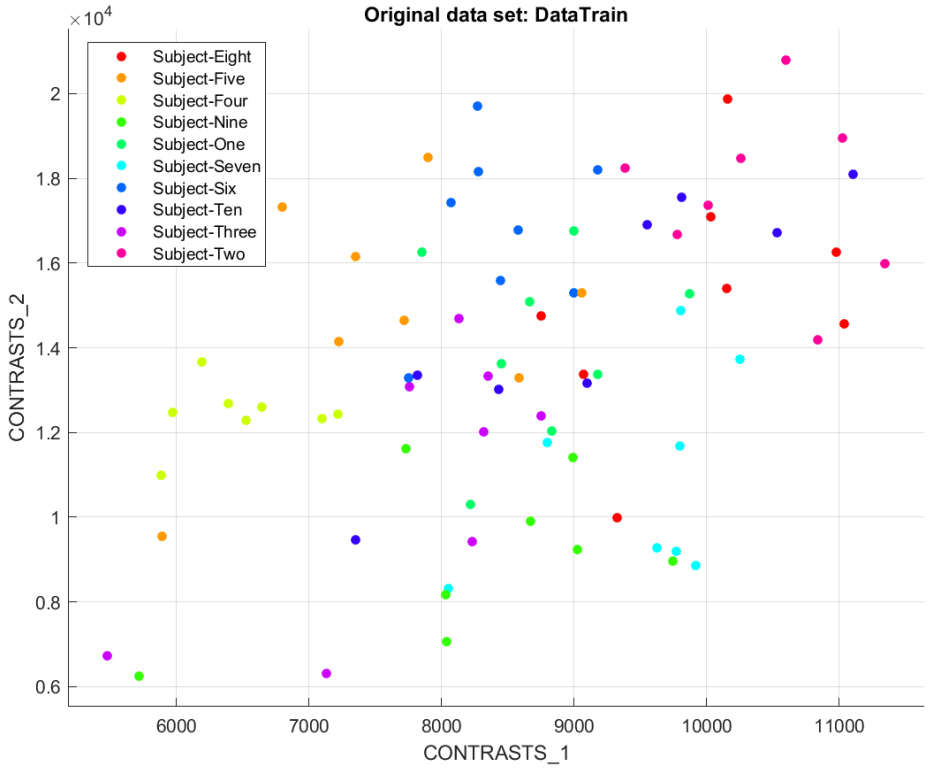

Figure 7.21. Scatter plot for the DB2_FVC2002 subset.

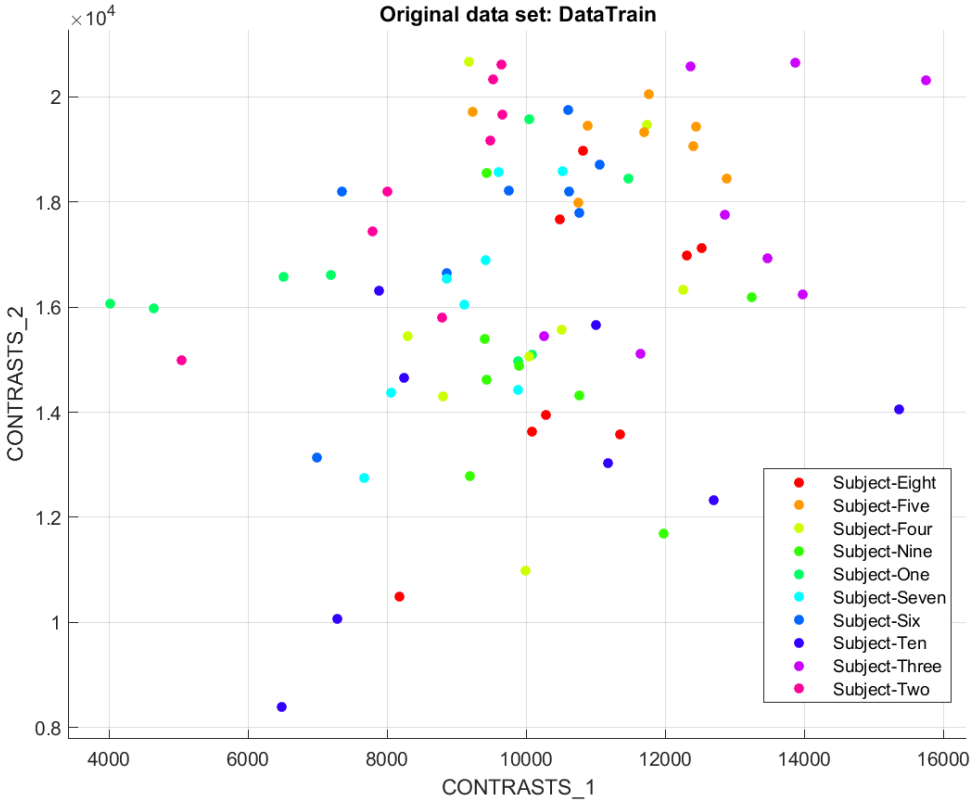

Figure 7.22. Scatter plot for the DB3_FVC2002 subset.

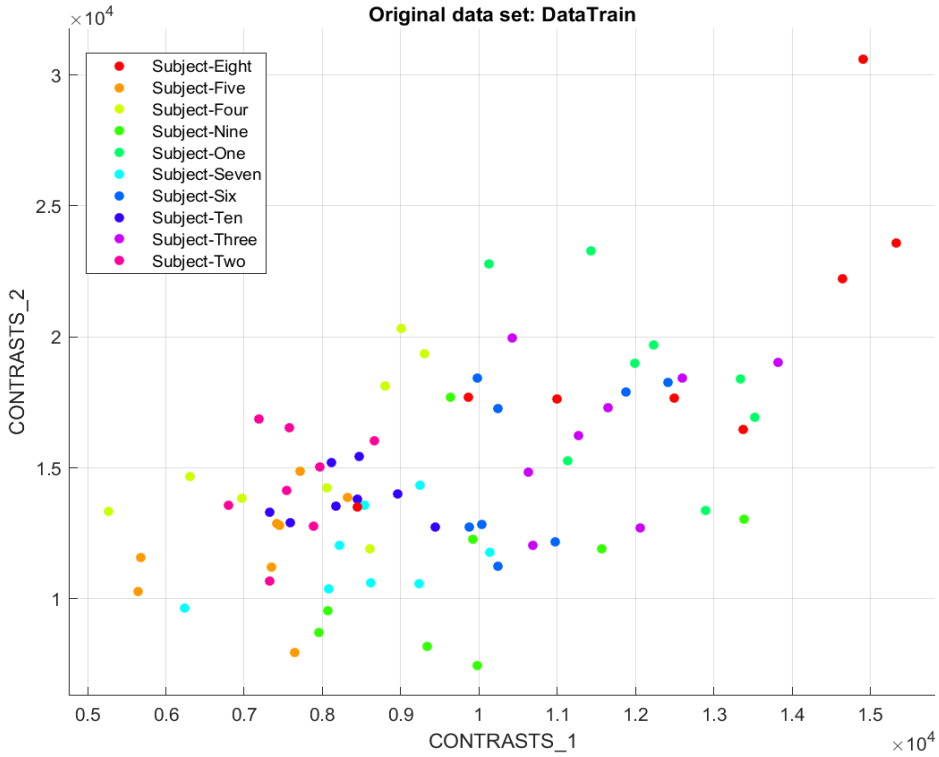

Figure 7.23. Scatter plot for the DB4_FVC2002 subset.

7.5.3. FVC2004

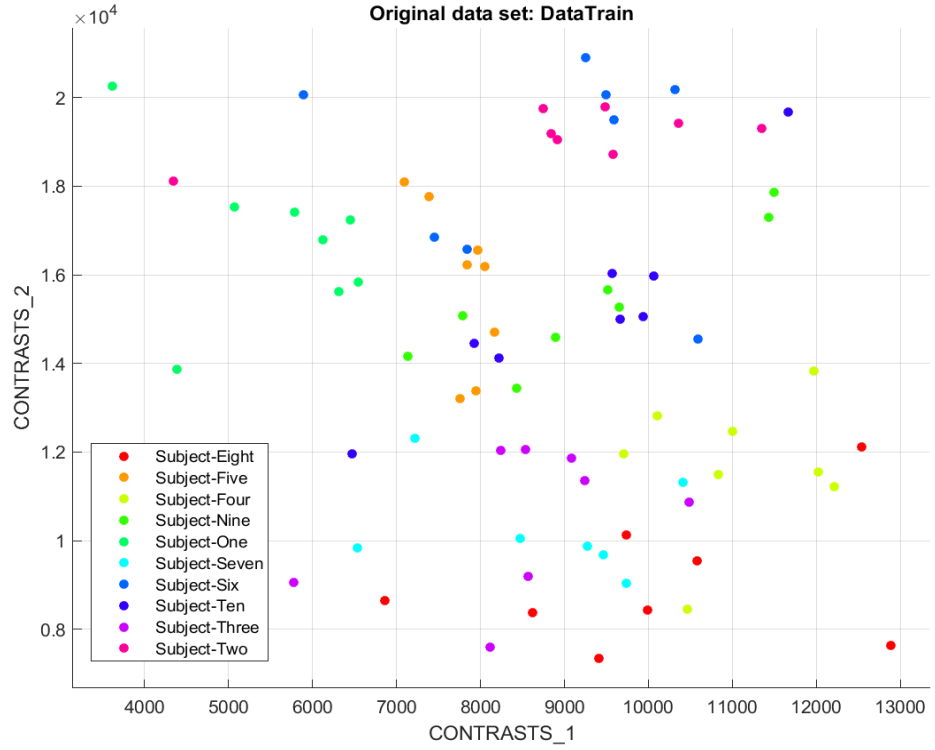

Figure 7.24. Scatter plot for the DB1_FVC2004 subset.

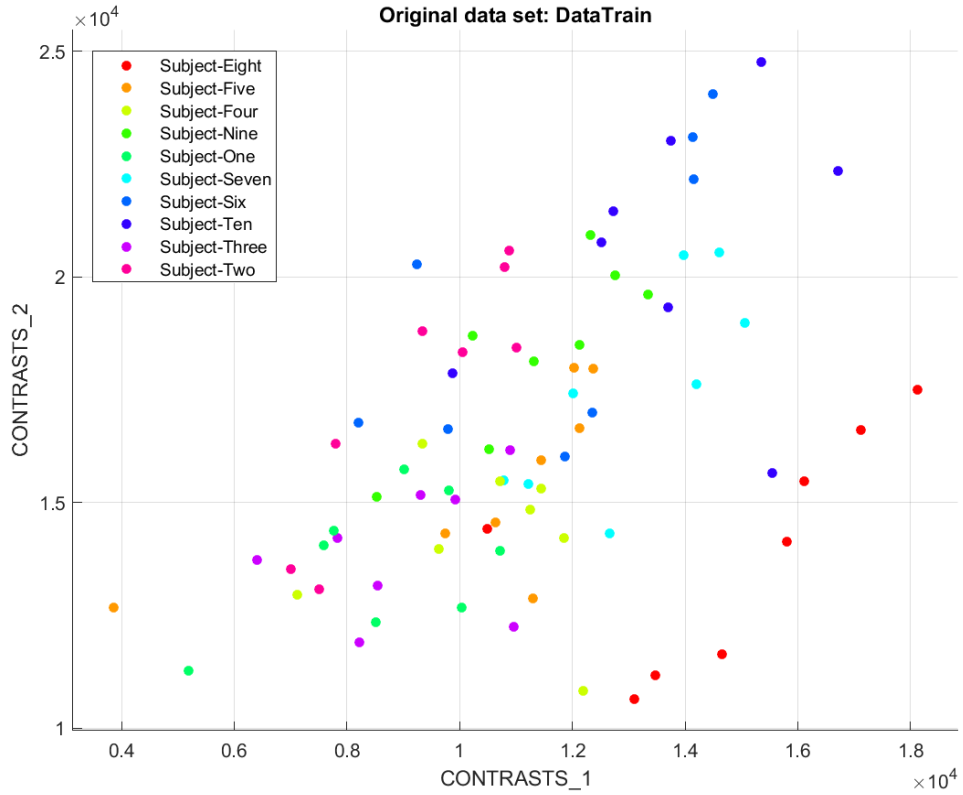

Figure 7.25. Scatter plot for the DB2_FVC2004 subset.

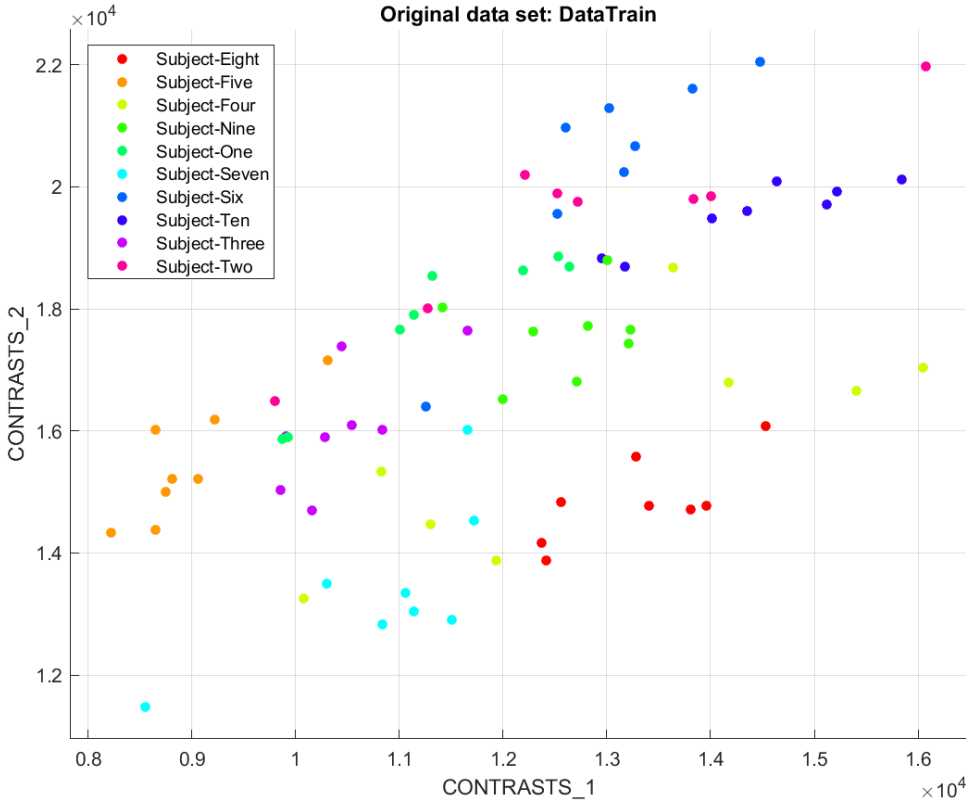

Figure 7.26. Scatter plot for the DB3_FVC2004 subset.

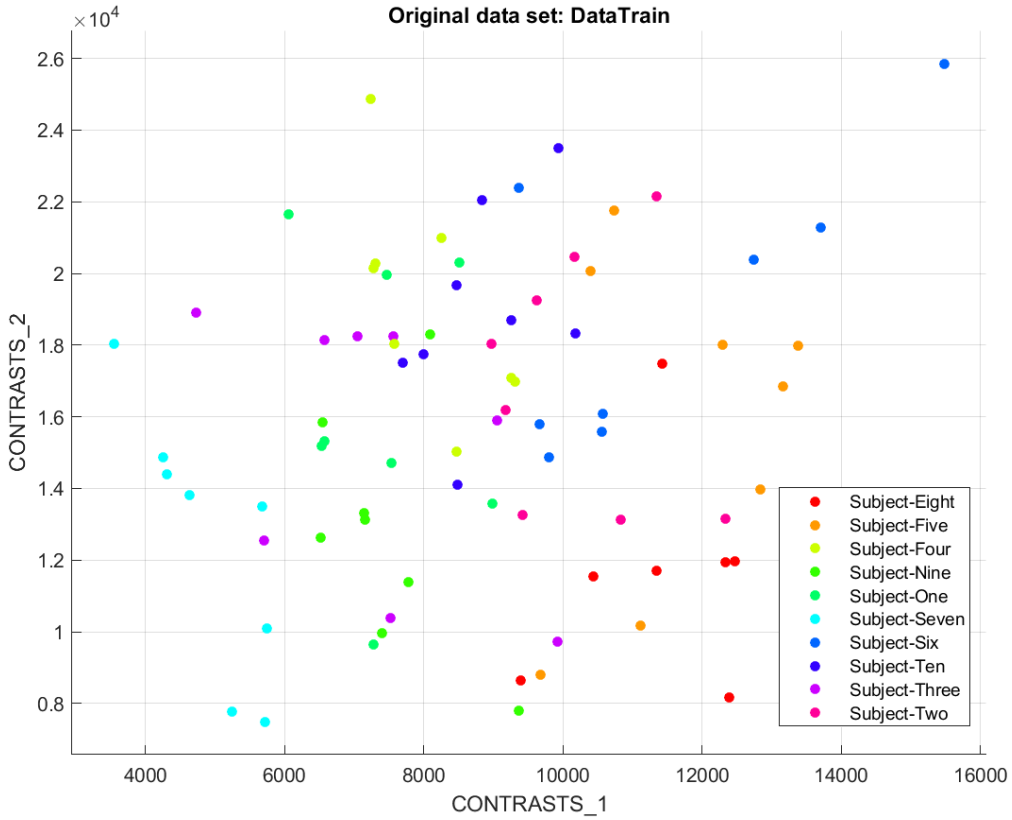

Figure 7.27. Scatter plot for the DB4_FVC2004 subset.

According to the scatter plots above, the database with notable divisions between classes (Subjects) is the FVC2000. This corresponds to the high accuracy values calculated for the subsets belonging to this database, the accuracies for this database are among the highest of 95%, 95%, 97.5%, and 95% for the DB1, DB2, DB3, and DB4 subsets, respectively. In contrast, the subsets with lower values have scatter plots with crowded data samples in which it is difficult to predict a division between classes, specifically, the DB2_FVC2004 has almost all the classes in the middle part of the plot, this translates into a more difficult classification process, which in the current system has achieved a 63.7% of accuracy.

7.5.4. NOISE COMPARISON

Another comparison could be made in terms of the original fingerprint images from each database. There is a significant difference between the noise added to the FVC2002 and FVC2004 databases compared to the FVC2000 database. The same preprocessing was performed on all three databases and the following images allow the original fingerprints to be compared with the enhanced ones, focusing on the noisiest images in each subset.

7.5.4.1. FVC2000

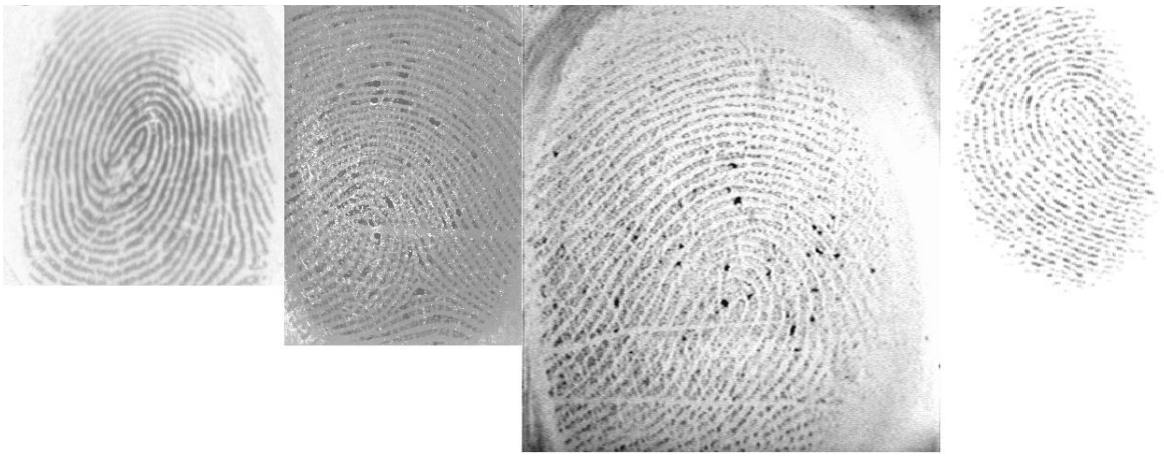

Figure 7.28. Original fingerprints from the FVC2000 database.

Figure 7.29. Enhanced versions of some FVC2000 database fingerprints.

The fingerprints included in the previous figures are in the group of the noisiest from the FVC2000 database, this is surprising due to the fact that at a glance these images have an acceptable quality (a nonexpert person could easily determine the ridge-valley structures for each fingerprint and reconstruct the missing spaces that these images present).

7.5.4.2. FVC2002

Figure 7.30. Original fingerprints from the FVC2002 database.

Figure 7.31. Enhanced versions of some FVC2002 database fingerprints.

The noisiest fingerprints from the FVC2002 database included in the previous images have more difficulty in terms of reconstruction of the ridge-valley patterns than the fingerprints from the FVC2000 database. This can be demonstrated by realizing that the preprocessing was not able to reconstruct some areas in the images, these sections have a high level of noise, and the algorithm fills these spaces with white pixels so that they can be combined with the background of the fingerprint image.

7.5.4.3. FVC2004

Figure 7.32. Original fingerprint images from the FVC2004 database.

Figure 7.33. Enhanced versions of the fingerprints from the FVC2004 database.

The images from the FVC2004 database reported in the previous images are among the noisiest of all subsets (FVC2000, FVC2002, and FVC2004). The preprocessing implemented in this work has more difficulty reconstructing various sections from these images, including white and dark areas within the fingerprint, oily smushes, and even some frames around the fingerprints.

7.5.5. COMPARISON OF ACCURACIES

The preprocessing enhancement algorithm although it can be improved, has been shown to have high and acceptable performance for several subsets in the three databases included in this work. The following figure presents a final comparison between boxplots for the validation accuracies for all subsets used in this work.

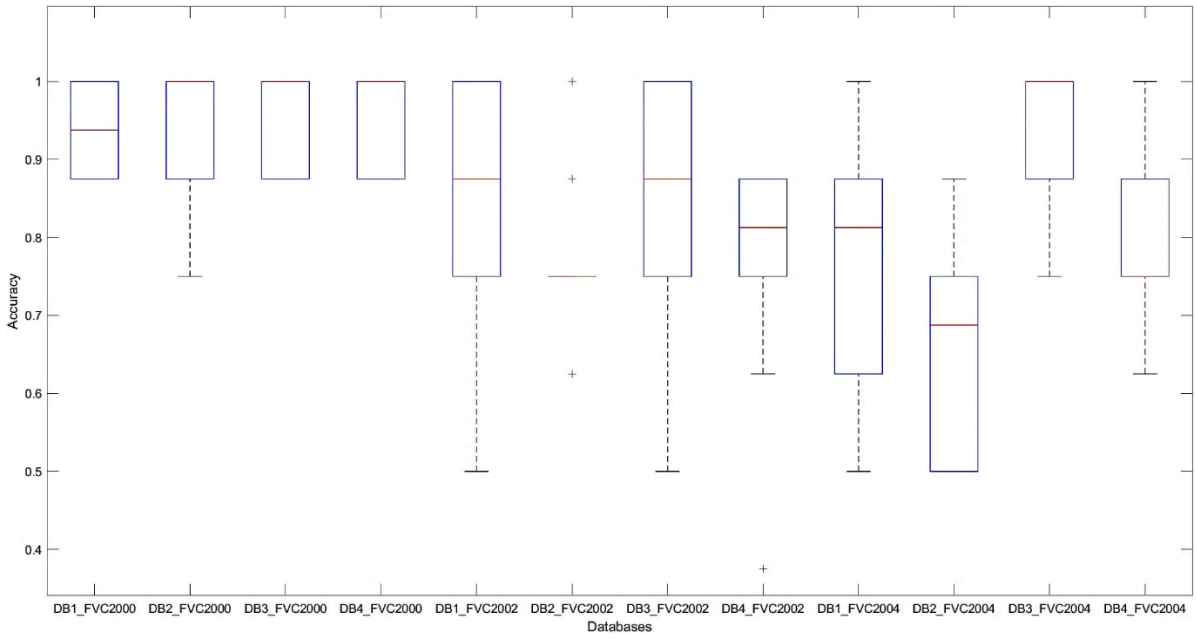

Figure 7.34. Boxplots of the accuracy values computed for each database utilized in this work.

The boxplots for the FVC2000 database (first four boxplots) have a higher distribution for the quartiles as can be noticed in the figure above. This behavior is consistent with the average accuracy values reported, which are among the highest for all 12 subsets. On the contrary, the remaining 8 subsets have wider quartiles, but almost for every classifier the fingerprint recognition system achieves a 100% accuracy for at least one-fold in the cross-validation implemented.

7.5.6. COMPARISON WITH STATE OF THE ART

The “Accuracy” metric is used to compare the performance of the classifiers from literature with the proposed method in this thesis. This comparison is presented in Table 7.2. The values included in the table correspond to the highest average accuracies obtained for each work. In our system, an accuracy of 97.5 % was obtained for the set “B” from the FVC2000 database subset DB3. All the works included in the table utilize the Wavelet transform at some stage of the fingerprint recognition. The number of features used by different methods in Table 7.2 varies from 17 up to over 1000. This implies that the system proposed in this work considers a moderate number of features taking 170, obtained from each fingerprint.

Table 7.2. Comparison of Results.

Work	Feature Extraction	Number of Features	Classification Model	Training Time	Accuracy
Iloanusi et al. [19] (2018)	Spatial, FFT, DCT, statistic measures	17	Manhattan distance	n/a	96.89 %
Velapure et al. [42] (2020)	Ridge contours, Gabor filter	≈ 16384	SVM	Not reported	87.5 %
Nguyen et al. [43] (2021)	Statistic measures	256	Random forest	10 h 50 min	95.8 %
Akbar et al. [15] (2014)	DWT, PCA, DCT	40	SVM	Not reported	95 %
Tang et al. [44] (2012)	Statistic measures, Shannon entropy	1152	Normalize Euclidean distance	n/a	96.84 %
Jirandeh et al. [45] (2014)	Gabor Wavelet	160	SVM	≈ 170 s	95.5 %
Abdul-Haleem et al. [46] (2014)	Energy, local ridge features, statistic measures, invariant moments	119	Absolute difference	n/a	96.87 %
Suwarno et al. [47] (2019)	Haar-like transformation	100	Hamming distance	n/a	80 %
This work	Spatial, FFT, DCT, GLCM, WBSF, statistic measures	170	Ensemble Subspace Discriminant	13.202 s	**97.5 %**

The training time is another characteristic that needs to be addressed since distance metrics are counted as instance-based learning or lazy-learning algorithms. In other words, they require zero training time because the training instance is simply stored [48]. The comparison of the reported training times with the time utilized in this work indicates an improvement of several seconds related to times reported in [43] and [45], which do not use distance metrics for classification. The proposed system also reports a slight improvement of less than 1% in comparison with the accuracy values reported by [19, 44, 46]. However, all these systems use distance metrics as the classification model, a much simpler approach in comparison with the ensemble classifier. It is well known that instance-based algorithms (distance metrics) are fast for learning but relatively sensitive to noise. Since the speed of classification is proportional to the number of predictors, they are slow for large sets of features. For this reason, we can justify a slight improvement in accuracy versus a secure outcome in recognition, provided by a much stable classifier, which in this case is an ensemble of subspace discriminant learners. It is important to notice that none of the

compared systems has accuracy as high as 97.5% using machine learning algorithms and reporting a reasonable training time, in this case, 13.202 seconds.

This demonstrates that the proposed fingerprint recognition system achieves high accuracy classification with a short time to train the ensemble subspace discriminant model, in comparison with the methods presented in the literature.

The authors in the works compared in Table 7.2 used several databases. Iloanusi, et al. [19] used the DB4 subset from the FVC2000 database, Velapure, et al. [42] applied the Fingerprint Color Image Database.v1, from MATLAB Central File Exchange. Nguyen, et al. [43] used the FVC group database, but subsets were not specified. Akbar, et al. [15] utilized the CASIA Fingerprint Image Database Version_5.0 (Dataset1), Tang, et al. [44] used the FVC2000 database, set A. Jirandeh, et al. [45] used the PolyU HRF database, Abdul-Haleem, et al. [46] the DB3 subset from the FVC2004 database, set A and Suwarno, et al. [47] used fingerprints captured by a commercial scanner, creating their own custom set. It is worth noting that although not all databases are the same, the comparison provides a good insight into the general performance of the proposed system inside the area of fingerprint recognition research.

CHAPTER 8 . FINGERPRINT RECOGNITION SYSTEM IN PYTHON

After developing a fingerprint recognition system in MATLAB, the next step in this work was to design a similar system in Python since this programing language nowadays has many applications for machine learning and image processing.

A similar preprocessing algorithm was implemented in Python 3 based on work designed by [9, 12, 49] which follows the same steps as described in previous chapters. The results of this preprocessing are the same as those calculated in MATLAB. However, feature extraction had several differences in terms of the specific functions to be used in this procedure.

8.1. LIBRARIES

The first step was to import the libraries that will be used to extract features from the preprocessed images. The libraries included are OpenCV-Python, NumPy, SciPy, scikit-learn, Pandas, PyWavelets, scikit-image, and DCTfunctions. Below is a brief description of these libraries.

8.1.1. OpenCV-Python

OpenCV-Python is a Python library designed to solve computer vision problems; it is an unofficial pre-built CPU-only OpenCV package for Python [50]. In this work, its main use was to load (read) input fingerprint images from database files.

8.1.2. NumPy

NumPy is one of the fundamental packages for scientific computing in Python. It provides functionality for multidimensional arrays (matrices included) and high-level mathematical functions such as linear algebra operations and the Fourier transform. A NumPy array is an array with the same type of data elements [32].

8.1.3. SciPy

SciPy is a collection of functions for scientific computing in Python including advanced linear algebra routines, signal processing, statistical distributions, etc. [32]

8.1.4. Scikit-learn

An open-source project that is constantly being developed and improved with a very active user community. This package contains various machine learning algorithms and is considered the most prominent Python library for machine learning. Scikit-learn is widely used in industry and academia, works well with various other scientific Python tools [32]. In scikit-learn, the NumPy array is the fundamental data structure, this implies that any data used with this package must be converted to a NumPy array. From the SciPy collection, scikit-learn extracts several functions to implement its algorithms [32].

8.1.5. Pandas

Pandas is a Python library for data management and analysis. Its basic structure is the DataFrame which can be modeled as a table, similar to an Excel spreadsheet. Pandas provides a good set of methods for modifying and operating on this table. In contrast to NumPy, Pandas allows each column to have a separate data type (for example, integers, dates, floating-point numbers, strings). A valuable characteristic of pandas is its ability to use a wide variety of database and file formats such as SQL, Excel, and comma-separated valued (CSV) files [32].

8.1.6. PyWavelets

PyWavelets is an open-source Wavelet transform software for Python. It combines a simple high-level interface with low-level C and Cython (C-extensions for Python) performance [51]. The Wavelet families included in this package are Haar, Daubechies, Symlets, Coiflets, Biorthogonal, etc. The main advantage of this software is its simplicity to handle equivalent functions to the Wavelet toolbox in MATLAB. For example, in this work, we applied the *wavedec* function to create the multilevel Wavelet decomposition in Python, which is analogous to the *wavedec2* function in MATLAB. Similar comments apply to the other functions that this software provides.

8.1.7. Scikit-image

It is a collection of algorithms for image processing in Python developed by an active international team of collaborators. This open-source library implements algorithms and utilities for use in research, education, and industry applications [52]. In this work, this

library was used to modify the fingerprint images to their normalized version and to implement the GLCM and its texture descriptors.

8.1.8. DCTfunctions

In Python, the DCT is calculated by the *dct* function from the SciPy.fftpack package. However, in this work, a custom function was created with the name DCTfunctions based on [53] which performs the fast discrete cosine transform and its inverse in two dimensions. Those functions work by wrapping the DFT function from NumPy, rather than explicitly performing the cosine and sine transformations themselves [53].

8.2. FEATURE EXTRACTION IN PYTHON

The feature extraction process follows the same workflow implemented in MATLAB. However, there were several differences to be aware of, as Python applies different libraries and functions that are not the same in MATLAB. The feature extraction script implemented in Python generates a CSV file with the set of features extracted from each subset, a total of 12 files were created in this step.

8.3. SUPERVISED LEARNING ALGORITHMS

The supervised learning algorithms available in the scikit-learn package include LDA, SVM, Nearest Neighbors, Naive Bayes, Decision Trees, Ensemble methods, etc. Since the MATLAB implementation uses machine learning algorithms for the classification process, the Python counterpart had to be taken into consideration. However, after trial-and-error the ensemble random subspace discriminant classifier was taken into consideration for all subsets since this type of classifier provides the highest accuracy values for recognition.

8.3.1. Linear Discriminant Analysis in Python

In Python the *LinearDiscriminantAnalysis* function implements the LDA algorithm, this function must be invoked from the *discriminant_analysis* package, which in turn is part of the scikit-learn library. In Python, the LDA function generates a classifier with linear decision boundaries, generated by fitting conditional class densities to the input data using Bayes' rule. The model fits a Gaussian density to each class assuming that they all share the same covariance matrix. This model can also be used to reduce the dimensionality of the input by projecting it in the most discriminative directions [54]. The syntax used in this work is:

```
classifier = LinearDiscriminantAnalysis()
```

where the default parameters have been used to implement the classifier, including:

- *solver='svd'*, where *svd* stands for singular value decomposition, meaning that the function does not compute the covariance matrix, which is recommended for data with a large number of features, such as the current fingerprint feature set.
- *shrinkage=None*, which implies that there is no shrinkage of data.
- *priors=None*, which controls the class prior probabilities, by default the class proportions are inferred from the training data.

8.3.2. Ensembles in Python

The two ensemble models in Python that are effective on a wide range of datasets for classification and regression are: random forests and gradient boosted decision trees [32]. However, as demonstrated in the previous chapter, the ensemble model composed of linear discriminant learners proves to be efficient for the current feature fingerprint dataset. For this reason, this type of classifier was implemented in Python generating interesting results. The *BaggingClassifier* function which belongs to the scikit-learn package is the heart of the classification process since it implements a bagging type ensemble. In this case, the following syntax was applied:

```
ensemble = BaggingClassifier(base_estimator =
LinearDiscriminantAnalysis(), n_estimators = 30, random_state = 7)
```

where *base_estimator* specifies the “weak” learner type, in this case, LDA, *n_estimators* indicates the number of base estimators in the ensemble, and *random_state* controls the random number generator used to resample the original dataset. However, this type of ensemble implementation does not consider the Random Subspace Ensemble approach, since one of the *BaggingClassifier* input parameters called *bootstrap* is set as True, indicating that the sampling of the predictors occurs with replacement. By setting this parameter with the following syntax, a random subspace ensemble can be implemented:

```
random_subspace_ensemble = BaggingClassifier(base_estimator =
LinearDiscriminantAnalysis(), bootstrap = False, max_features =
85, n_estimators = 30, random_state = 7)
```

where *bootstrap=False* configures the ensemble using a random subspace of *max_features* input features, arbitrarily chosen. The numbers included for several other parameters have the same values as their corresponding parameters in MATLAB.

Then, using both approaches, the average accuracy values, with 10-fold cross-validation for the twelve subsets, are reported in the following table.

Table 8.1. Results of the Classification Process

Database	**Accuracy**	
	Ensemble: Linear Discriminant	Ensemble: Subspace Discriminant
DB1_FVC2000	86.25 %	90 %
DB2_FVC2000	85 %	84.58 %
DB3_FVC2000	96.25 %	94.16 %
DB4_FVC2000	83.75 %	86.67 %
DB1_FVC2002	67.5 %	68.75 %
DB2_FVC2002	51.25 %	52.08 %
DB3_FVC2002	56.25 %	67.5 %
DB4_FVC2002	65 %	65.42 %
DB1_FVC2004	48.75 %	47.5 %
DB2_FVC2004	38.75 %	38.75 %
DB3_FVC2004	73.75 %	84.17 %
DB4_FVC2004	63.75 %	61.25 %

In general, there is a small improvement for the calculated mean accuracy values for the subspace case compared to the ensemble default case. A better representation of these values for both cases is presented in the following boxplots of the 10-fold cross-validation accuracies for fingerprint recognition in Python.

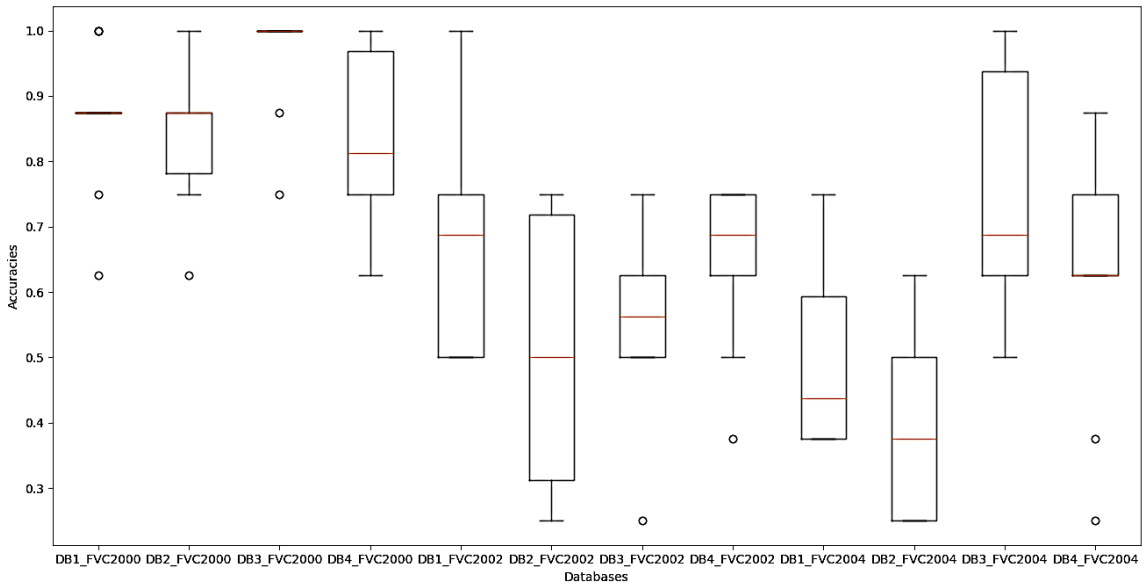

Figure 8.1. Boxplots for the default ensemble classifiers.

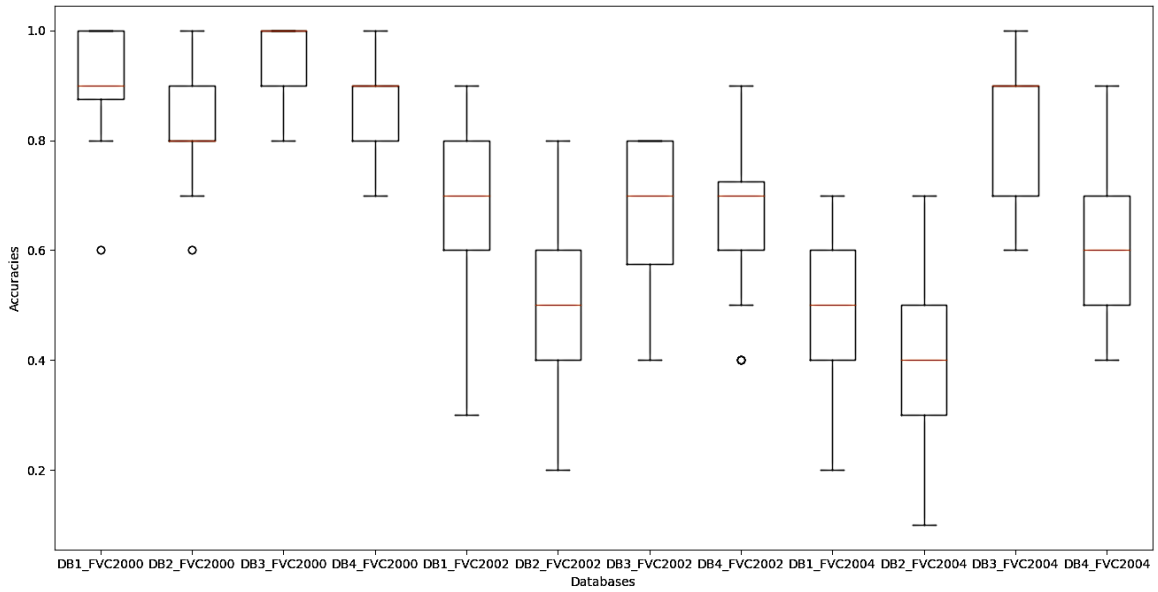

Figure 8.2. Boxplots for ensemble random subspace classifiers.

By observation, it can be said that the ensemble random subspace classifiers provide a better recognition accuracy since their boxplots have higher values for the inter-quartile range.

CHAPTER 9 . CONCLUSIONS AND FUTURE WORK

9.1. CONCLUSIONS

In this thesis, a demo application has been designed using MATLAB GUIDE creating a tool with more complete coverage of IP topics in comparison with works previously presented. A great variety of examples are included to help better understand the IP principal topics in a graphical environment.

A fingerprint recognition system has been described, with the novelty of using random subspace ensemble discriminant classifiers. The main subsystems: preprocessing, feature extraction, and classification; produce interesting accuracy values which demonstrate that this fingerprint recognition system is capable to work with several databases proving its capacity for generalization.

The machine learning approach included in this system provides better results in comparison with simpler classifiers such as decision trees, k-NN, Naive Bayes, LDA, etc. Both systems (MATLAB and Python) provide good frameworks for the design and assessment of fingerprint recognition, which is why in this thesis, both programming languages were used.

The maximum accuracy value reported for MATLAB is 97.5 % and for Python is 96.25 % which both correspond to the FVC2000 database subset DB3. This high accuracy demonstrates an acceptable performance achieved by the image processing and machine learning techniques applied in this thesis.

9.2. FUTURE WORK

The preprocessing algorithm implemented in this thesis must be improved to achieve higher accuracy recognition values. In consequence, it is planned to develop a better algorithm using different image processing techniques such as image restoration or morphological image processing, which can produce better ridge-valley structures or even recover missing sections from poor-quality fingerprints.

Additionally, another machine learning algorithms that could be implemented with this system are neural networks, which according to the literature could produce higher accuracy values, improving the performance of the proposed fingerprint recognition system.

Recently, image processing along with machine learning techniques have been used to solve problems of spectrum sensing in cognitive radio. Consequently, this would be a possible investigation direction.

LIST OF FIGURES

REFERENCES

[1] R. C. Gonzalez, R. E. Woods and S. L. Eddins, Digital Image Processing Using MATLAB, Gatesmark Publishing, 2009.

[2] National Research Council of the National Academies, "Biometric recognition: Challenges and opportunities," National Academies Press, Washington, 2010.

[3] A. M. Bazen, Fingerprint Identification: Feature Extraction, Matching, and Database Search, Twente University Press, 2002.

[4] P. Schuch, "Deep learning for fingerprint recognition systems," NTNU, 2019.

[5] S. Bharkad and M. Kokare, "Fingerprint identification — ideas, influences, and trends of new age," in *Pattern Recognition, Machine Intelligence and Biometrics*, Berlin, Springer, 2011, pp. 411-446.

[6] BioLab - University of Bologna, "FVC2000," 2000. [Online]. Available: http://bias.csr.unibo.it/fvc2000/download.asp. [Accessed 8 February 2021].

[7] BioLab - University of Bologna, "FVC2002 - Second International Fingerprint Verification Competition," 2002. [Online]. Available: http://bias.csr.unibo.it/fvc2002/download.asp. [Accessed 8 February 2021].

[8] Biometric System Lab - University of Bologna, "FVC2004 - Third International Fingerprint Verification Competition," 2003. [Online]. Available: http://bias.csr.unibo.it/fvc2004/download.asp. [Accessed 8 February 2021].

[9] L. Hong, Y. Wan and A. Jain, "Fingerprint image enhancement: Algorithm and performance evaluation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 20, no. 8, pp. 777-789, 1998.

[10] H. Farid and E. P. Simoncelli, "Differentiation of discrete multidimensional signals," *IEEE Transactions on Image Processing,* vol. 13, no. 4, pp. 496-508, 2004.

[11] A. K. Jain and F. Farrokhnia, "Unsupervised texture segmentation using gabor filters," *Pattern Recognition,* vol. 24, no. 12, pp. 1167-1186, 1991.

[12] P. Kovesi, "MATLAB and Octave Functions for Computer Vision and Image Processing," 2000. [Online]. Available: https://www.peterkovesi.com/matlabfns/#fingerprints. [Accessed 17 February 2021].

[13] MathWorks, "Wavelet Toolbox," The MathWorks, [Online]. Available:

https://www.mathworks.com/products/wavelet.html. [Accessed 17 May 2021].

[14] K. S. Jeyalakshmi and T. Kathirvalavakumar, "Haralick features from wavelet domain in recognizing fingerprints using neural network," in *International Conference on Mining Intelligence and Knowledge Exploration*, 2020.

[15] S. Akbar, A. Ahmad and M. Hayat, "Identification of fingerprint using discrete wavelet transform in conjunction with support vector machine," *IJCSI International Journal of Computer Science Issues,* vol. 11, no. 5, pp. 189-199, 2014.

[16] M. D. Al-Hassani, A. Kadhim and V. W. Samawi, "Fingerprint identification technique based on wavelet-bands selection features (WBSF)," *International Journal of Computer Engineering and Technology (IJCET),* vol. 4, no. 3, pp. 308-323, 2013.

[17] MathWorks, "2-D wavelet decomposition," The MathWorks, [Online]. Available: https://www.mathworks.com/help/wavelet/ref/wavedec2.html. [Accessed 17 May 2021].

[18] B. Pathak and D. Barooah, "Texture analysis based on the gray-level co-occurrence matrix considering possible orientations," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering,* vol. 2, no. 9, pp. 4206-4212, 2013.

[19] O. Iloanusi, N. David, C. Osuagwu and S. Olisa, "Multiple domains and transform-based features for fingerprint matching," *International Journal of Scientific and Technology Research,* vol. 7, no. 9, pp. 27-34, 2018.

[20] MathWorks, "Standard deviation," The MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/std.html. [Accessed 17 May 2021].

[21] MathWorks, "Variance," The MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/var.html#bum7s4o-1-w. [Accessed 17 May 2021].

[22] MathWorks, "2-D fast Fourier transform," The MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/fft2.html. [Accessed 17 May 2021].

[23] MathWorks, "2-D discrete cosine transform," The MathWorks, [Online]. Available: https://www.mathworks.com/help/images/ref/dct2.html. [Accessed 17 May 2021].

[24] MathWorks, "Maximum elements of an array," The MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/max.html. [Accessed 17 May 2021].

[25] MathWorks, "Vector and matrix norms," The MathWorks, [Online]. Available: https://www.mathworks.com/help/matlab/ref/norm.html. [Accessed 17 May 2021].

[26] MathWorks, "Skewness," The MathWorks, [Online]. Available: https://www.mathworks.com/help/stats/skewness.html. [Accessed 14 May 2021].

[27] D. P. Doane and L. E. Seward, "Measuring skewness: A forgotten statistic?," *Journal of Statistics Education,* vol. 19, no. 2, 2011.

[28] MathWorks, "Kurtosis," The MathWorks, [Online]. Available: https://www.mathworks.com/help/stats/kurtosis.html?s_tid=doc_ta. [Accessed 14 May 2021].

[29] G. Bonaccorso, Machine Learning Algorithms, Packt Publishing Ltd, 2017.

[30] MathWorks, "Machine Learning with MATLAB," [Online]. Available: https://www.mathworks.com/content/dam/mathworks/ebook/gated/machine-learning-ebook-all-chapters.pdf. [Accessed 18 May 2021].

[31] MathWorks, "Train models to classify data using supervised machine learning," The MathWorks, Inc., [Online]. Available: https://www.mathworks.com/help/stats/classificationlearner-app.html. [Accessed 8 February 2021].

[32] A. C. Muller and S. Guido, Introduction to machine learning with Python: a guide for data scientists, O'Reilly Media, Inc, 2016.

[33] P. Xanthopoulos, P. Pardalos and T. Trafalis, "Linear discriminant analysis," in *Robust Data Mining*, Springer, 2013, pp. 27-33.

[34] M. Skurichina and R. P. W. Duin, "Bagging, boosting and the random subspace method for linear classifiers," *Pattern Analysis & Applications*, vol. 5, no. 2, pp. 121-135, 2002.

[35] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123-140, 1996.

[36] S. Agarwal and C. R. Chowdary, "A-stacking and A-bagging: Adaptive versions of ensemble learning algorithms for spoof fingerprint detection," *Expert Systems with Applications*, vol. 146, 2020.

[37] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Machine Learning: Proc. of the Thirteenth International Conference*, 1996.

[38] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832-844, 1998.

[39] MathWorks, "Mastering Machine Learning: A Step-by-Step Guide with MATLAB," [Online]. Available: https://www.mathworks.com/campaigns/offers/mastering-machine-learning-with-matlab.html. [Accessed 10 February 2021].

[40] MathWorks, "Classification Learner App," The MathWorks, [Online]. Available: https://www.mathworks.com/help/stats/classification-learner-app.html?s_tid=CRUX_lftnav. [Accessed 21 May 2021].

[41] MathWorks, "Fit ensemble of learners for classification - MATLAB fitcensemble," The Mathworks, Inc., [Online]. Available: https://www.mathworks.com/help/stats/fitcensemble.html#bvcj_s0-1-Mdl. [Accessed 6 April 2021].

[42] A. Velapure and R. Talware, "Performance analysis of fingerprint recognition using machine learning algorithms," in *Proc. of the Third International Conference on Computational Intelligence and Informatics*, 2020.

[43] L. T. Nguyen, H. T. Nguyen, A. D. Afanasiev and T. V. Nguyen, "Automatic identification fingerprint based on machine learning method," *Journal of the Operations Research Society of China*, 2021.

[44] T. Tang, "Fingerprint recognition using wavelet domain features," in *2012 8th International Conference on Natural Computation (ICNC 2012)*, 2012.

[45] H. M. Jirandeh, H. Sadeghi and M. A. Javadi Rad, "High-resolution automated fingerprint recognition system (AFRS) based on gabor wavelet and SVM," *International Journal of Scientific and Engineering Research,* vol. 5, no. 5, pp. 166-169, 2014.

[46] M. G. Abdul-Haleem, L. E. George and H. M. Al-Bayti, "Fingerprint recognition using haar wavelet transform and local ridge attributes only," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 1, pp. 122-130, 2014.

[47] S. Suwarno and P. I. Santosa, "Simple verification of low-resolution fingerprint using non-minutiae feature," in *Journal of Physics: Conference Series*, 2019.

[48] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques," *Informatica,* vol. 31, pp. 249-268, 2007.

[49] U. Deshmukh, "GitHub - Utkarsh-Deshmukh/Fingerprint-Enhancement-Python: Using oriented gabor filters to enhance fingerprint images," 19 April 2021. [Online]. Available: https://github.com/Utkarsh-Deshmukh/Fingerprint-Enhancement-Python. [Accessed 26 April 2021].

[50] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools,* 2000.

[51] G. R. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt and A. O'Leary, "PyWavelets: A Python package for wavelet analysis," *Journal of Open Source Software,* vol. 4, no. 36, p. 1237, 2019.

[52] S. Van der Walt, J. L. Schonberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart and T. a. t. s.-i. c. Yu, "scikit-image: image processing in Python," *PeerJ,* p. e453, 2014.

[53] M. Newman, "Functions to perform fast discrete cosine and sine transforms," 24 June 2011. [Online]. Available: http://www-personal.umich.edu/~mejn/computational-physics/dcst.py. [Accessed 21 April 2021].

[54] scikit-learn developers, "sklearn.discriminant_analysis.LinearDiscriminantAnalysis," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html. [Accessed 27 April 2021].