תקציר בעברית

הפרויקט מתמקד בתכנון ופיתוח אלגוריתם לשיפור איכות תמונות טביעת אצבע, על מנת לאפשר חילוץ מדויק של תכונות מפתח הנקראות מיניושיות. זיהוי טביעות אצבע מתבסס על תבניות ייחודיות של רכסים והתפצלויות בקצות האצבעות, אך גורמים שונים כגון חוסר אחידות בלחץ, יובש, לחות, הטיה ועיוותים עלולים לפגוע באיכות התמונות המתקבלות.

כדי להתמודד עם בעיות אלו, יישמנו ב־ MATLAB תהליך עיבוד שלם. האלגוריתם מתחיל בשיפור מורפולוגי, ממשיך לבינאריזציה, ניקוי התמונה באמצעות מסכות קונבולוציה, ודילול של הרכסים, ליצירת תמונה מעודנת של טביעת האצבע. תהליך זה מאפשר זיהוי מדויק של נקודות מיניושיה, כולל סיומי רכסים והתפצלויות.

המידע שחולץ מאורגן במבנה מקיף הכולל קואורדינטות מרחביות, מדדי מרחק וניתוחים סטטיסטיים. אף על פי שמוקד הפרויקט הוא שיפור איכות התמונה וחילוץ תכונות מטביעות האצבע, אנו מציעים גם אפשרויות לאלגוריתמים להשוואה, בהתבסס על המידע המבני, ובכך מניחים בסיס להמשך פיתוח בתחום השוואת טביעות האצבע וזיהוין.

English Abstract

This project focuses on designing and developing an algorithm for enhancing fingerprint image quality to enable accurate extraction of key features known as minutiae. Fingerprint identification relies on the unique ridge and bifurcation patterns found on fingertips, but various factors such as pressure inconsistencies, dryness, moisture, tilt, and distortion can compromise the quality of the captured images.

To overcome these issues, we implemented a complete processing pipeline in MATLAB. The algorithm begins with morphological enhancement, followed by binarization, image cleaning using convolution masks and thinning to produce a refined fingerprint image. This allows for precise detection of minutiae points, including ridge endings and bifurcations.

The extracted data is structured into a comprehensive format that includes spatial coordinates, distance metrics, and statistical summaries. While the primary focus of this project is the accurate enhancement and feature extraction of fingerprint images, we also suggest potential matching algorithms based on the structured information, setting the stage for future work in fingerprint comparison and identification.

Table Of Contents

Figure List

1. Introduction

1.1 Overview of fingerprints identification systems

Fingerprint recognition is one of the most popular and effective methods for personal identification, leveraging the uniqueness of minutiae—specific points where fingerprint ridge lines end, intersect, or branch. Biometric identification systems that use fingerprint patterns are known as AFIS (Automatic Fingerprint Identification Systems), we will be extracting level 2 minutiae features.

A typical fingerprint recognition system consists of four main stages: capturing biometric data using a sensor, pre-processing the image to enhance ridge clarity and remove noise, extracting distinctive fingerprint features, and matching these features against templates stored in a database [2]

1.2 The importance of fingerprints identification

We touch things every day, a coffee cup, a car door, a computer keyboard. Each time we do, it is likely that we leave behind our unique signature in our fingerprints.

No two people have the same fingerprints. Even identical twins, with identical DNA, have different fingerprints. This uniqueness allows fingerprints to be used in a variety of ways, including background checks, biometric security, mass disaster identification, and of course, criminal investigations.

Fingerprint analysis has been used to identify suspects and solve crimes for more than 100 years, and it remains an extremely valuable tool for law enforcement. One of the most important uses for fingerprints is to help investigators link one crime scene to another involving the same individual. Fingerprint identification also assists in tracking a criminal's record, previous arrests, and convictions, supporting sentencing, probation, parole, and pardoning decisions.[3]

Beyond criminal justice, fingerprint recognition plays a vital role in civil applications such as voter registration, border control, securing access to smartphones and personal devices, financial transactions, and healthcare record management. It is increasingly integrated into

modern identity verification systems for travel documents like e-passports and national ID cards, providing a fast, secure, and reliable method to authenticate an individual's identity.[4]

1.3 Objectives and Engineering Solutions

1.3.1 Development of a Fingerprint Minutiae Extraction Algorithm

Design and implement an algorithm capable of enhancing fingerprint image quality and accurately extracting minutiae points, which include ridge endings and bifurcations. This involves applying morphological processing techniques, such as ridge thinning, to improve feature clarity and prepare the fingerprint for reliable analysis.

1.3.2 Engineering Solutions Enabled by this Work

The engineering solutions achieved through this project include improving the reliability of biometric identification in various applications, such as background checks, access control systems, mobile device security, and civil registration systems. By enhancing the accuracy and efficiency of fingerprint recognition, this work contributes to stronger authentication mechanisms for criminal investigations, border control, e-passport verification, and secure financial transactions.

2. Literature Review

2.1 Fingertip formation and features

2.1.1 Minutiae points

A fingerprint is the impression formed by the friction ridges on the surface of a fingertip. Everyone's fingerprint serves as a unique identifier and remains unchanged throughout their lifetime, provided the fingertip is not damaged by injury or deep cuts. Even identical twins possess different fingerprints, highlighting their uniqueness [5]. A fingerprint template is composed of ridge patterns and valleys across the fingertip. Ridges are defined as individual curved lines, while valleys refer to the spaces between two adjacent ridges. The distinctive arrangement of these ridges and valleys forms the unique features found in every fingerprint, as illustrated in Figure 1 and 2.

The term "minutia" refers to a pixel that holds special significance due to its unique neighbourhood and orientation. [3] Since every image is composed of pixels—basic units carrying the image's information—some pixels become particularly important because they contain critical, unique details. These combinations of pixels are referred to as minutiae points.

In fingerprint analysis, the most fundamental minutiae features are ridge endings and ridge bifurcations. A pixel with only one neighbour in a thinned fingerprint image is classified as a ridge ending, while a pixel with three neighbours is considered a ridge bifurcation.[1]

Well-defined fingerprint images are crucial both for manual annotation and for computerized extraction. Comparing different minutiae extraction algorithms is possible, and advanced automated methods can achieve sub-pixel accuracy.

It is important to note that the orientation and location of minutiae heavily depend on the image processing stages.

Figure 1 - Minutiae points

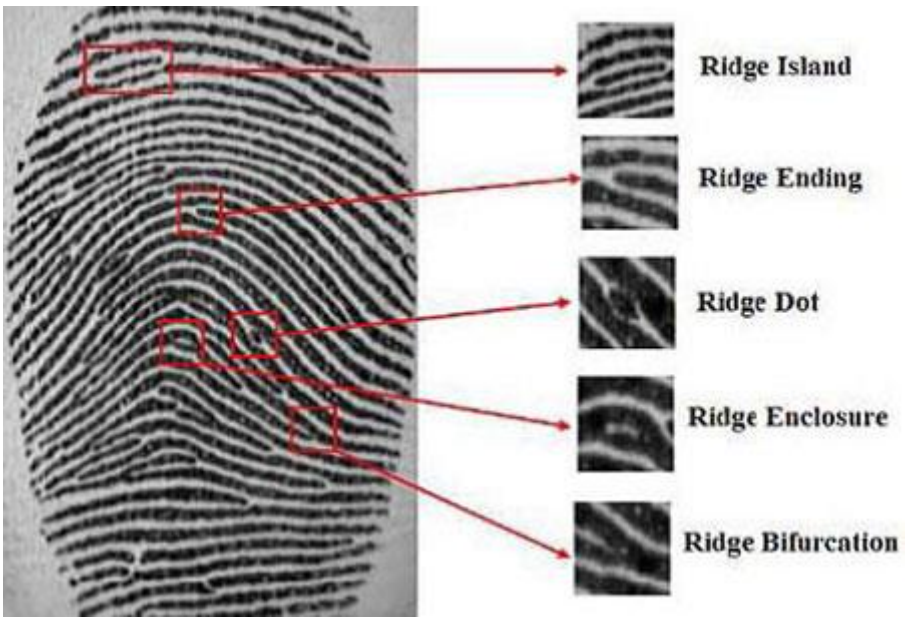

Figure 2 - classes of ridges and bifurcations

2.1.2 The five major fingertip feature classes

Fingerprints are typically categorized into five major classes based on the overall pattern of their ridges and valleys. These classes capture the broad structural flow of the fingerprint and serve as a foundation for further detailed analysis.[5][4] The five major fingerprint pattern classes are:

- **Arch** – A simple pattern where ridges flow from one side to the other without forming loops or whorls. Arches are the least common type.
- **Tented Arch** – Similar to the arch but with a sharp rise or "tent-like" spike in the center, forming a more pronounced peak.
- **Left Loop** – A pattern where ridges enter from the right, form a loop, and exit toward the left side of the finger.
- **Right Loop** – A pattern where ridges enter from the left, loop around, and exit toward the right side.
- **Whorl** – A circular or spiral pattern where ridges form complete circuits. Whorls are more complex and can include several subtypes.

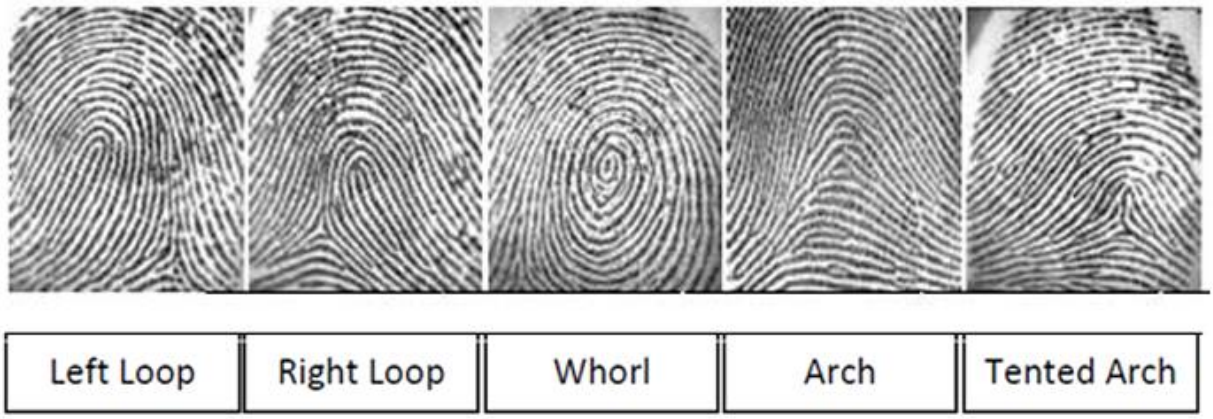

Figure 3 - The 5 major categories

2.1.3 The 3 levels of fingerprint features

Fingerprint features are categorized into three levels: [5],

Level 1 - focuses on the overall ridge flow and pattern types (arch, loop, whorl) for basic classification.

Level 2 - captures the minutiae details like ridge endings and bifurcations, which are key for unique identification.

Level 3 - examines fine structures such as ridge shapes, pores, and ridge thickness, offering high-precision matching when high-resolution images are available.

2.2 Fingerprint sample quality

Fingerprint patterns, formed by friction ridges on the fingertip, must be captured with high clarity to enable reliable feature extraction and matching. Image quality is influenced by environmental factors, user behaviour, biometric sample integrity, and system performance.[6][7] User-controllable factors, such as moisture on fingers or contamination of the scanner surface, can be mitigated to enhance image quality. In contrast, limitations like sensor resolution, algorithmic robustness, or ridge degradation due to aging or pathology are system- or subject-dependent and cannot be corrected at the user level. [6]

Provided in figure 3. Fingerprint different quality samples:
a) good quality, b) medium quality, c) poor quality. [6][7]

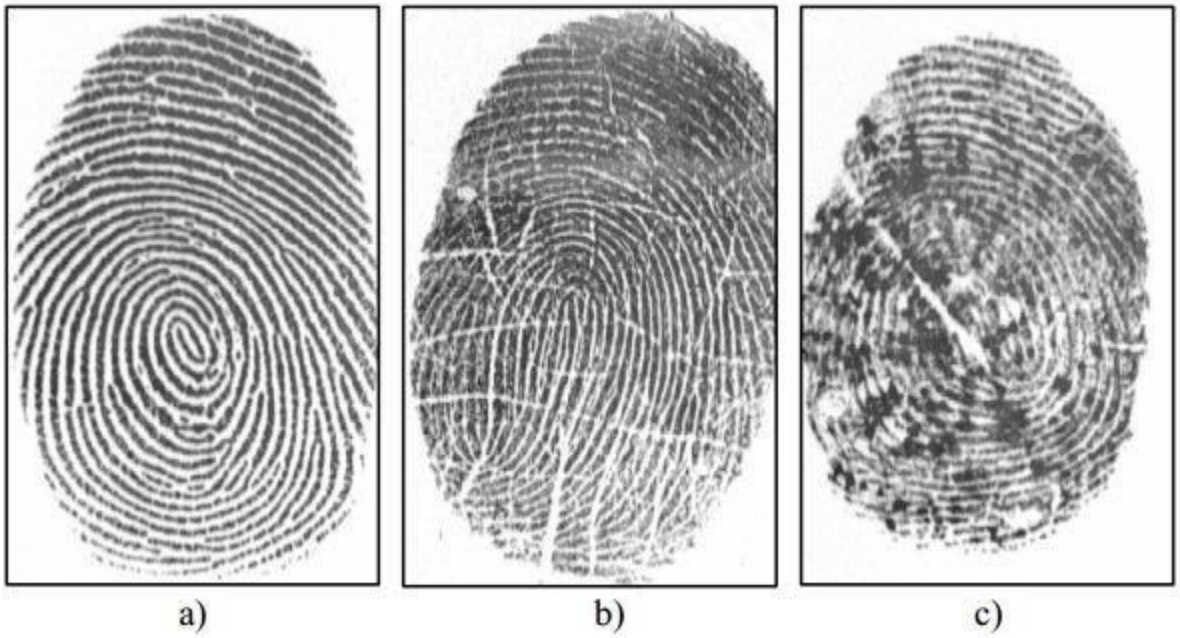

Figure 4 - Sample Quality

2.3 Morphological operations

Morphology is a broad set of image processing operations that process images based on shapes. In a morphological operation, each pixel in the image is adjusted based on the value of other pixels in its neighbourhood. By choosing the size and shape of the neighbourhood, you can construct a morphological operation that is sensitive to specific shapes in the input image.[13]

2.3.1 Morphological operation – Thinning:

It is used to decrease the ridge width to make the ridge only 1 pixel wide. It is done so that we can easily find the ridge end or bifurcates in the fingerprint,

The thinning algorithm goes as such:[12]

In the first sub iteration, delete pixel p if and only if the conditions G_1, G_2, and G_3 are all satisfied. [8][10]

In the second sub iteration, delete pixel p if and only if the conditions G_1, G_2, and G_3' are all satisfied.

Condition G_1:

$$X_{H(p)} = 1 \qquad 1$$

where

$$X_{H(p)} = \sum_{i=1}^{4} bi = 1 \qquad (3)$$

$$bi = \begin{cases} 1, \ if\ x_{2i-1} = 0\ and\ (x_{2i} = 1\ or\ x_{2i+1} = 1) \\ 0, \text{otherwise} \end{cases} \qquad (3)$$

$x_1,\ x_2, \ldots,\ x_8$ are the values of the eight neighbours of p, starting with the east neighbor and numbered in counter-clockwise order.[10]

Condition G_2:

$$2 \leq min\{\ n_1(p), n_2(p)\ \} \leq 3 \qquad (4)$$

where

$$n_1(p) = \sum_{k=1}^{4} x_{2k-1} \vee x_{2k} \tag{5}$$

$$n_2(p) = \sum_{k=1}^{4} x_{2k} \vee x_{2k+1} \tag{6}$$

Condition G_3:

$$(x_2 \vee x_3 \vee x_8') \wedge x_1 = 0 \tag{7}$$

Condition G_3':

$$(x_6 \vee x_7 \vee x_4') \wedge x_5 = 0 \tag{8}$$

The two sub iterations together make up one iteration of the thinning algorithm. When the user specifies an infinite number of iterations ($n = Inf$), the iterations are repeated until the image stops changing.

2.3.2 Morphological Process – ROI (Region of interest):

Region of Interest (ROI) extraction in fingerprint images is essential to isolate reliable ridge patterns from background noise, ensuring robust feature detection and reduced computational load. The process begins with adaptive binarization to convert the grayscale image into a binary map, highlighting ridges. This is followed by morphological thinning, which refines ridge lines to a single-pixel width for better structural analysis.[31][30]

To enhance continuity in fragmented ridge segments, morphological closing is applied—using dilation followed by erosion—which bridges small gaps. Subsequently, hole-filling eliminates small background regions enclosed within ridges. Morphological opening (erosion followed by

dilation) is then used to remove small noise clusters, ensuring that only significant ridge structures remain.

Further refinement is achieved through connected component analysis, removing small, disconnected foreground objects that likely represent noise. Finally, erosion shrinks the remaining regions slightly, trimming uncertain boundary areas and preserving only the stable core of the fingerprint.[31]

The resulting binary mask defines the ROI, allowing for background suppression in the original image and filtering out invalid minutiae. These morphological operations are tailored to the structural regularity of fingerprint patterns, offering a practical and effective segmentation approach without the need for deep learning.

2.4 Image Formats

Image Format describes how data related to the image will be stored. Data can be stored in compressed, Uncompressed, or vector format. Each format of the image has a different advantage and disadvantage. Image types such as TIFF are good for printing while JPG or PNG, are best for the web.[14] Images of BMP type preserve complex, high-quality images. Each pixel is stored individually, ensuring the image's quality and accuracy remain intact.[15][14]

TIFF(.tif, .tiff):

Tagged Image File Format this format store image data without losing any data. It does not perform any compression on images, and a high-quality image is obtained but the size of the image is also large, which is good for printing, and professional printing.

JPEG (.jpg, .jpeg):

Joint Photographic Experts Group is a loss-prone (lossy) format in which data is lost to reduce the size of the image. Due to compression, some data is lost but that loss is very less. It is a very common format and is good for digital cameras, nonprofessional prints, E-Mail, PowerPoint, etc., making it ideal for web use.

BMP (.bmp):

Bitmap files are great for icons, screengrabs, and 2D photos. They can store color image data as well as monochrome data in a variety of bit/color depths. Though most bitmap files are uncompressed and large, you can make them smaller with lossless data compression, also allows you to store complex, high-quality images. Each BMP file will store every pixel independently and so the quality and accuracy of the image are maintained.

PNG (.png):

PNG or Portable Network Graphics files are a lossless image format. It was designed to replace gif format as gif supported 256 colors unlike PNG which support 16 million colors.

2.5 Image types

RGB images represent color through three channels—red, green, and blue—each contributing to the final color seen. Grayscale images reduce this to a single channel, capturing light intensity from black to white in ranges of 0 to 255. Binary images take it a step further, simplifying the image to two values: black and white, 0 and 1, often to highlight key features [17].

2.5.1 Gray Scale

Gray Scale is a range of shades of gray without apparent color. The darkest possible shade is black, which is the total absence of transmitted or reflected light. The lightest possible shade is white, the total transmission or reflection of light at all visible wavelengths. Intermediate shades of gray are represented by equal brightness levels of the three primary colors (red, green and blue) for transmitted light, or equal amounts of the three primary pigments (cyan, magenta and yellow) for reflected light.[16]

Conversion of RGB to Gray scale given by the equation:

$$0.299 \cdot Red + 0.587 \cdot Green + 0.114 \cdot Blue = Gray\ scaled_{pixel\ value\ (i,j)} \qquad (9)$$

$$whereas, \qquad i = row, j = coloumn$$

2.5.2 RGB

RGB image can be viewed as three different images (a red scale image, a green scale image and a blue scale image) stacked on top of each other, and when fed into the red, green and blue inputs of a color monitor, it produces a color image on the screen.

2.5.3 Binary

Binary images are images that have been quantised to two values, usually denoted 0 and 1, but often with pixel values 0 and 255, representing black and white. Binary images are used in many applications since they are the simplest to process, but they are such an impoverished representation of the image information that their use is not always possible. Sometimes the output of other image processing techniques is represented in the form of a binary image, for example, the output of edge detection can be a binary image (edge points and non-edge points). [16] Binary image processing techniques can be useful for subsequent processing of these output images. Binary images are typically obtained by thresholding a grey level image. Pixels with a grey level above the threshold are set to 1 (equivalently 255), whilst the rest are set to 0. This produces a white object on a black background (or vice versa, depending on the relative grey values of the object and the background). Of course, the 'negative' of a binary image is also a binary image, simply one in which the pixel values have been reversed.[17]

2.6 Image preprocessing

2.6.1 Average Operator

The averaging operator smooths an image by replacing each pixel's value with the average of its neighbouring pixel values within a defined template.[18] In a $M \times N$ template, each pixel inside the window contributes equally.

The output at position (x, y) is given by (eq. 10):

$$O(x,y) = \frac{1}{MN} \sum_{i \in M} \sum_{j \in N} I(x+i, y+j) \tag{10}$$

$I(x+i, y+j)$ are the pixel values within the template, and M and N are the dimensions of the template.[18]

This operation reduces noise and fine details, revealing the broader structures in the image as we can see in (fig .5).

Figure 5 - Original image on the left, Averaged filtered image on the right

2.6.2 Linear Contrast Stretch Enhancement

Contrast enhancements improve the perceptibility of objects in the scene by enhancing the brightness difference between objects and their backgrounds. Contrast enhancements are typically performed as a contrast stretch followed by a tonal enhancement, although these could both be performed in one step. A contrast stretch improves the brightness differences uniformly across the dynamic range of the image, whereas tonal enhancements improve the brightness differences in the shadow (dark), midtone (grays), or highlight (bright) regions at the expense of the brightness differences in the other regions.[19]

A high-contrast image spans the full range of gray-level values; therefore, a lowcontrast image can be transformed into a high-contrast image by remapping or stretching the gray-level values such that the histogram spans the full range. The contrast stretch is often referred to as the dynamic range adjustment (DRA). The simplest contrast stretch is a linear transform that maps the lowest gray level GL_{min} in the image to zero and the highest value GL_{max} in the image to 255 (for an eight-bit image), with all other gray levels remapped linearly between zero and 255,

to produce a high-contrast image that spans the full range of gray levels. This linear transform is given by:

$$g'(x,y) = INT\left\{\frac{255}{GL_{max} - GL_{min}} \cdot [g(x,y) - GL_{min}]\right\} \tag{11}$$

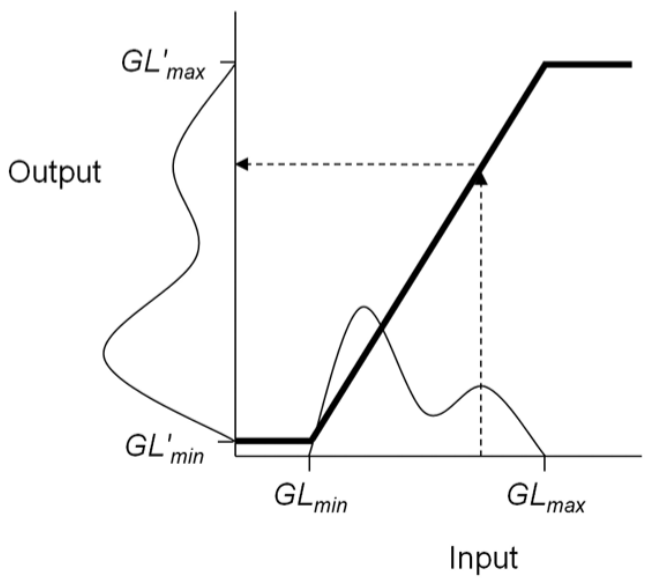

Figure 6 - A Linear transform that remaps the gray levels between GL'min and GL'max.

The linear transform for contrast enhancement spreads the gray-level values evenly over the full contrast range available; thus, the relative shape of the histogram remains unchanged but is widened to fill the range. [19] The stretching of the histogram creates evenly distributed gaps between gray-level values in the image. Note that although the linear transform will increase the contrast of the image, the steps between the populated gray-level values increase in contrast as well, which can result in visible contouring artifacts in the image (Fig. 7).

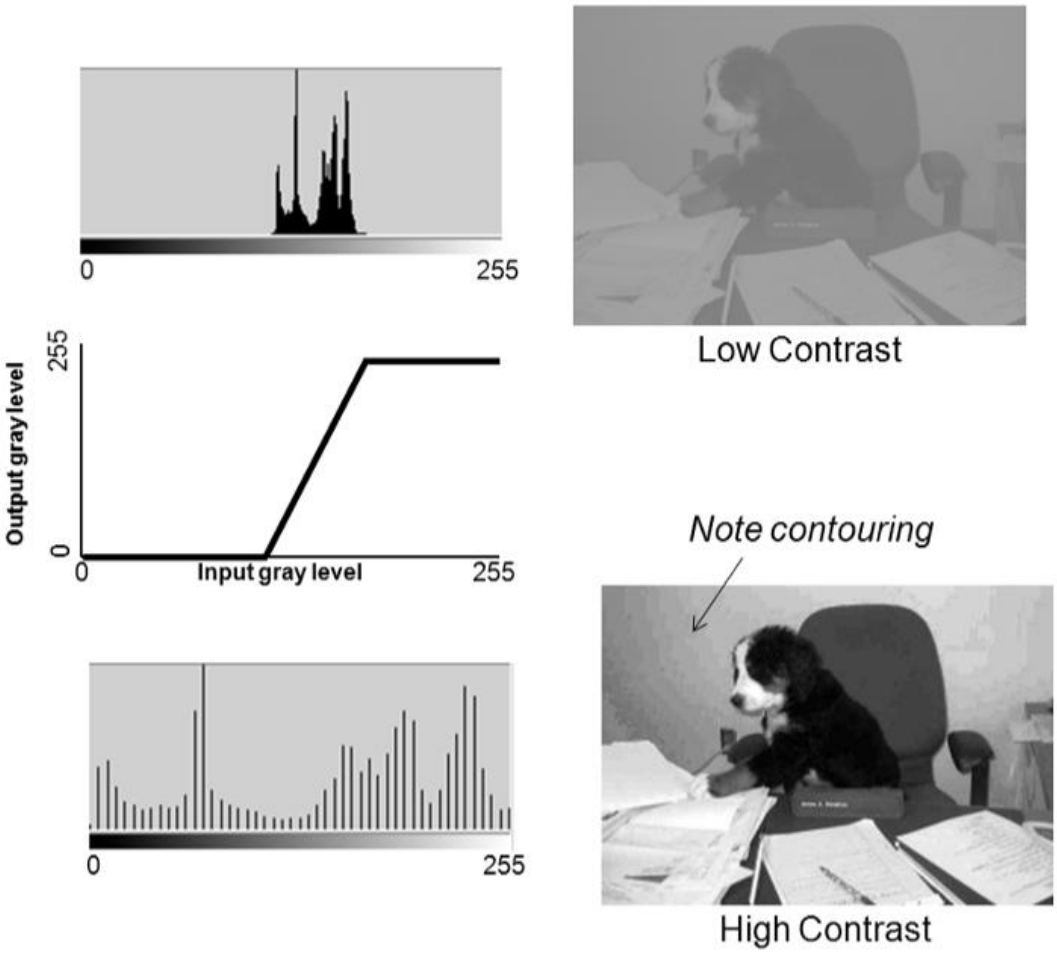

Figure 7 - Enhancement of low-contrast images can produce image artifacts.

2.6.3 Binarization

Image binarization is an important aspect of image analysis, such as scene text detection and medical image analysis. Especially in the field of document image processing, binarization has a wide range of applications as a basic method of digital image processing, including text recognition, document image segmentation, image morphological processing and feature extraction. It commonly serves as the primary stage in document analysis and recognition systems, as well as Optical Character Recognition (OCR), exerting a substantial impact on the efficacy of subsequent character segmentation and recognition.[20]

The threshold method is an image segmentation technique that relies on the grayscale value of a pixel to separate the image based on a specified threshold. This method typically involves the local thresholding method, in local Thresholding, [21][20] the image is divided into regions/segments to determine independent threshold values for each region/segment which separate the image pixels based on their size relationship to the specified threshold.

The threshold calculation formula is expressed as follows:

$$T = m + k * s \tag{12}$$

Where, m represents the average grey value of pixels in the local area, s represents the standard deviation, and k is a constant, a correction factor, which can be adjusted according to the background conditions of the image.[20] we can understand more by observing at the differences in (fig 8), the left image is the original and the image on the right is after binarization filter that had a threshold of some value, low enough to exclude the environment and keep all dark shades, example given in (fig. 8).

Figure 8 - Example of Binarized image

2.7 Convolution

2.7.1 2D convolution in image processing

Many image processing results come from a modification of one pixel with respect to its neighbours. When this modification is similar in the entire image g, it can be mathematically defined using a second image h which defines the neighbour relationships.[22] This results in a third image f. This is the so-called convolution, and it is denoted with $*$:

$$f(x,y) = g(x * h)h(x,y) = \sum_{m} \sum_{n} g(x-m, y-n)h(m,n) \tag{13}$$

Whereas: $f(x,y)$ — output, $g(x,y)$ — input, $h(m,n)$ —Kernel h, row m, column n;

Intuitively, the convolution "spreads" each pixel (m,n) in g following h and proportionally to the intensity $g(m,n)$. an example of the computing of a particular pixel is provided in (fig 9).

$$\begin{array}{|c|c|c|c|c|}
\hline
f_{1,1} & f_{1,2} & f_{1,3} & f_{1,4} & f_{1,5} \\
\hline
f_{2,1} & \boxed{f_{2,2}} & f_{2,3} & f_{2,4} & f_{2,5} \\
\hline
f_{3,1} & f_{3,2} & f_{3,3} & f_{3,4} & f_{3,5} \\
\hline
f_{4,1} & f_{4,2} & f_{4,3} & f_{4,4} & f_{4,5} \\
\hline
f_{5,1} & f_{5,2} & f_{5,3} & f_{5,4} & f_{5,5} \\
\hline
\end{array}
=
\begin{array}{|c|c|c|c|c|}
\hline
g_{1,1} & g_{1,2} & g_{1,3} & g_{1,4} & g_{1,5} \\
\hline
g_{2,1} & g_{2,2} & g_{2,3} & g_{2,4} & g_{2,5} \\
\hline
g_{3,1} & g_{3,2} & g_{3,3} & g_{3,4} & g_{3,5} \\
\hline
g_{4,1} & g_{4,2} & g_{4,3} & g_{4,4} & g_{4,5} \\
\hline
g_{5,1} & g_{5,2} & g_{5,3} & g_{5,4} & g_{5,5} \\
\hline
\end{array}
*
\begin{array}{c}
\begin{array}{ccc} -1 & 0 & +1 \end{array} \\
\begin{array}{|c|c|c|}
\hline
h_{-,-} & h_{-,0} & h_{-,+} \\
\hline
h_{0,-} & h_{0,0} & h_{0,+} \\
\hline
h_{+,-} & h_{+,0} & h_{+,+} \\
\hline
\end{array}
\end{array}
\begin{array}{c}
\\ -1 \\ 0 \\ +1
\end{array}$$

$$f_{2,2} = g_{3,3}h_{-,-} + g_{3,2}h_{-,0} + g_{3,1}h_{-,+} + g_{2,3}h_{0,-} + g_{2,2}h_{0,0} + g_{2,1}h_{0,+} + g_{1,3}h_{+,-} + g_{1,2}h_{+,0} + g_{1,1}h_{+,+}$$

Figure 9 - calculation of matrix g convolved with matrix h

2.7.2 Kernels in 2D convolution

In image convolution, involves a kernel, or matrix that is applied over the input image's pixels to generate an output image. The kernel size and values determine the effect the kernel has on the image. The dimensions of the kernel should be smaller or equal to that of the input images. The kernel typically is square shaped and has an odd kernel size out of convenience. How is the

kernel applied in a convolution? The first step is that the kernel is flipped both horizontally and vertically. This is done by convolution. [23] Using a non-flipped kernel would be doing a cross-correlation rather than a convolution. In the case of a symmetric kernel, the cross-correlation is equivalent to its convolution. Flipping or not flipping the kernel generally does not have a large impact on the resulting image visually.[22]

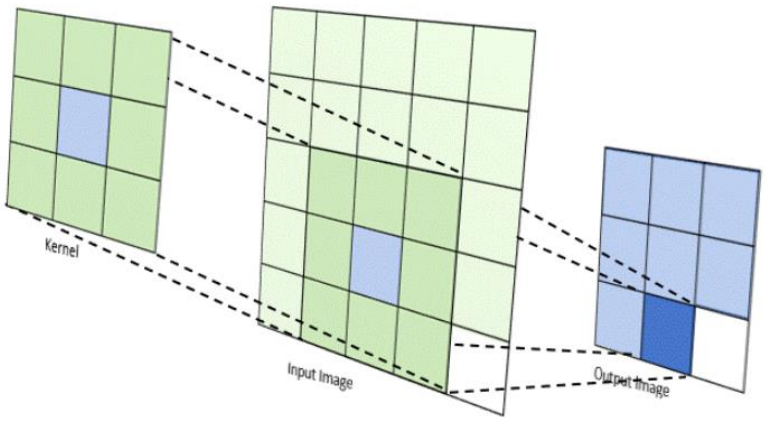

Figure 10 - Convolution of input matrix with a kernel matrix

The output image pixels are calculated by performing an element-by-element multiplication with the kernel and the covered section of the input image and then summing them up. Given an example kernel and input image, an example of the calculation is shown in (fig 10) and (fig 11) with the first pixel. In the example, I intentionally used small numbers for ease of calculation. Additionally, it's important to note that the output pixels of a convolution can yield values outside of 0–255, it may be useful to normalize the results.[23]

In 2D convolution, kernels modify images by enhancing or suppressing specific features. They are used for blurring and smoothing to reduce noise, for edge detection by highlighting intensity changes, and for sharpening to enhance fine details. Kernels can also create texture effects like embossing or reduce noise while preserving important structures. The choice of kernel determines how the image is transformed and which features become more prominent.[22]

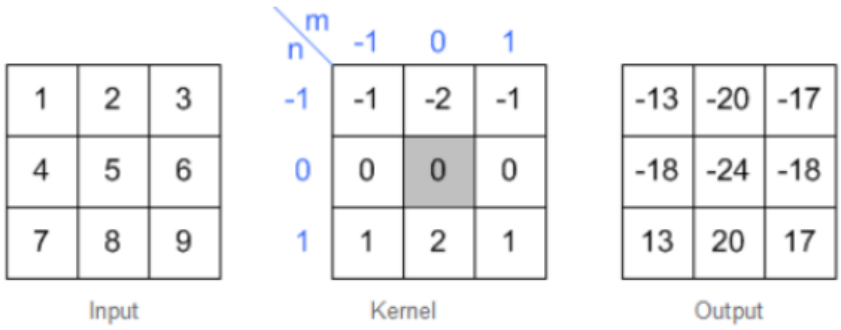

Figure 11 - Convolution Output

2.8 Breadth-First Search Algorithm

Breadth-First Search (BFS) is a fundamental graph traversal algorithm that explores a graph systematically, layer by layer. Starting from a designated source node, BFS visits all nodes at the current distance before moving to nodes farther away. It relies on a queue data structure to manage the order of exploration, ensuring that nodes are visited in the sequence they are discovered. When a node is visited, all of its unvisited neighbouring nodes are added to the queue and marked as visited to prevent repeated exploration. This method guarantees that the shortest path (in terms of the number of edges) from the source to any reachable node is found in an unweighted graph. BFS is widely used in applications such as shortest path finding, connectivity checking, and network broadcasting.[25][24]

Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion to search a graph data structure for a node that meets a set of criteria. It uses a queue to remember the next vertex to start a search, when a dead end occurs in any iteration.

Breadth First Search algorithm starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.[24]

As in the example given in (fig 12), BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.[25]

- **Rule 1** − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** − If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** − Repeat Rule 1 and Rule 2 until the queue is empty.

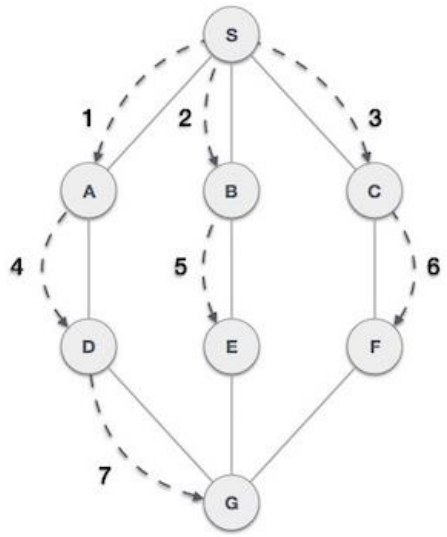

Figure 12 - BFS propagation

2.9 Image Histogram

An image histogram is a graphical representation of the number of pixels in an image as a function of their intensity.

Histograms are made up of bins, each bin representing a certain intensity value range. The histogram is computed by examining all pixels in the image and assigning each to a bin depending on the pixel intensity. [26] An image histogram is a graph of pixel intensity (on the *x*-axis) versus number of pixels (on the *y*-axis). The *x*-axis has all available grey levels, and the *y*-axis indicates the number of pixels that have a particular grey-level value. Multiple grey levels can be combined into groups in order to reduce the number of individual values on the *x*-axis, as shown in (fig 13).

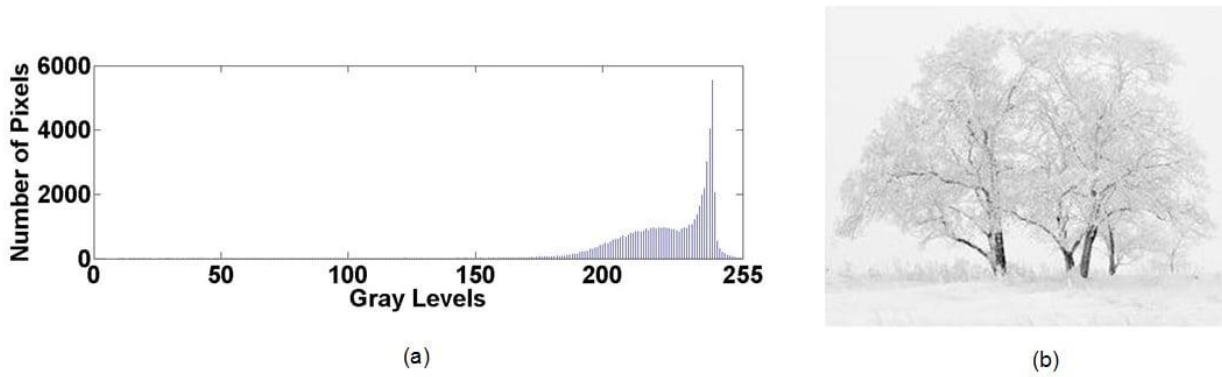

Figure 13 - Histogram of pixel intensity

2.9.1 Histogram comparison

To compare two histograms $(H_1 \; and \; H_2)$, first we have to choose a *metric* $(d(H_1, H_2))$ to express how well both histograms match, [27] Bhattacharyya distance histogram comparison given by the equation:

$$D_B(P, Q) = -ln(\sum_{i=1}^{n} \sqrt{P(i) \cdot Q(i)} \tag{14}$$

Whereas

$P(i)$ and $Q(i)$ are the normalized values (sum to 1) of the i_{th} bin in the two histograms.

The sum mentioned in eq. 14 is called the Bhattacharyya coefficient

The coefficient measures the amount of overlap between the two distributions:

- A value of **1** means perfect overlap (identical histograms).
- As the value approaches **0**, the overlap decreases.

The distance D_B increases as similarity decreases.

In (fig 14) there's an intersection of two histograms as comparison method.

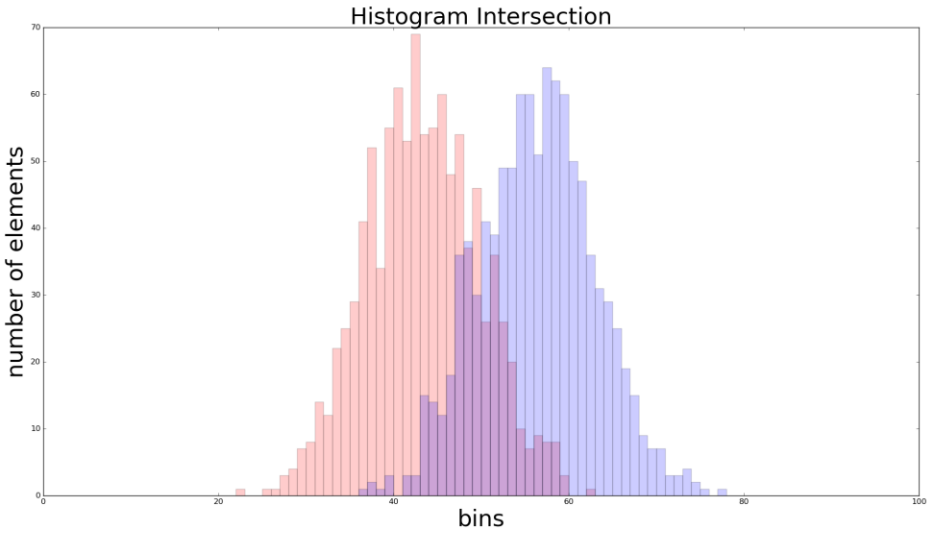

Figure 14 - Histogram matching by intersection

2.10Machine Learning and Neural Networks

2.10.1 Supervised learning

the various algorithms generate a function that maps inputs to desired outputs. One standard formulation of the supervised learning task is the classification problem the learner is required to learn (to approximate the behaviour of) a function which maps a vector into one of several classes by looking at several input-output examples of the function.[28]

The learning process in a simple machine learning model is divided into two steps: training and testing. In training process, samples in training data are taken as input in which features are learned by learning algorithm and build the learning model. In the testing process, learning model uses the execution engine to make the prediction for the test or production data.

2.10.2 Convolutional Neural Network

A CNN consists of four layers: the convolutional layer, pooling layer, fully connected layer, and nonlinearity layer. Each layer plays a distinct role in the network's function, which will be explained in detail in the following subsections [27][28][29] and shown in (fig 15).

Convolutional Layer

The convolutional layer processes an image as a set of numbers, typically represented as matrices, where each number corresponds to the light intensity of a pixel. The convolutional layer uses a kernel filter to calculate the convolution of the input image, extracting key features. The kernel filter is smaller in size than the input image but has a constant parameter value. The filter slides across the input image step by step, calculating the product of the kernel weights and the pixel values, resulting in a 2D activation map. The network learns visual features from this map. The convolution process is defined by parameters such as the kernel size, stride length, and padding. The kernel size refers to the filter's dimensions, stride length indicates how many steps the filter takes across the image, and padding refers to the extra space around the input image.

Pooling layer

The pooling layer connects two consecutive convolutional layers and helps reduce the number of parameters and computational load by down-sampling the representations. The pooling function typically either maximizes or averages values, with the maximizing function often used for optimal performance. This layer is also effective in reducing overfitting and computational weights. The pooling operation typically reduces the dimensions of an activation map, with the max-pooling function being a common approach for this purpose.

Fully Connected Layer

The third layer in a CNN is the fully connected layer, also referred to as the convolutional output layer. This layer operates similarly to a feedforward neural network and is typically located at the bottom of the network. It receives inputs from the final pooling or convolutional output layer, which is flattened into a vector before being passed on. This flattening technique allows for studying high-level non-linear combinations of features represented by the output from the previous layers.

The nonlinearity layer, or activation function, plays a crucial role in CNN layers. It applies a mathematical transformation to the filtered output, determining the final output of the network. One of the most common activation functions is the Rectified Linear Unit (ReLU), which is used for feature extraction. The activation function maps output values within a specific range, such as between -1 and 1 or 0 and 1, enabling the network to model more complex relationships. Activation functions can be divided into linear and non-linear types. Linear activation functions

produce outputs proportional to the input, which can perform simple tasks like 'yes' or 'no' decisions. In contrast, non-linear activation functions allow neural networks to model more complex mappings between inputs and outputs, essential for advanced learning and modelling tasks.

Nonlinearity Layer (Activation Function)

The nonlinearity layer, or activation function, is essential in CNNs to determine the output of a neuron after filtering. ReLU (Rectified Linear Unit) is the most common activation function, used for feature extraction. It maps the output to specific ranges like -1 to 1 or 0 to 1, enabling complex decision-making.

Activation functions fall into two categories:

1. **Linear Activation**: Outputs are proportional to input, suitable for simpler tasks but limited in complexity.
2. **Non-linear Activation**: Enables complex mappings between inputs and outputs, essential for advanced learning and modelling.

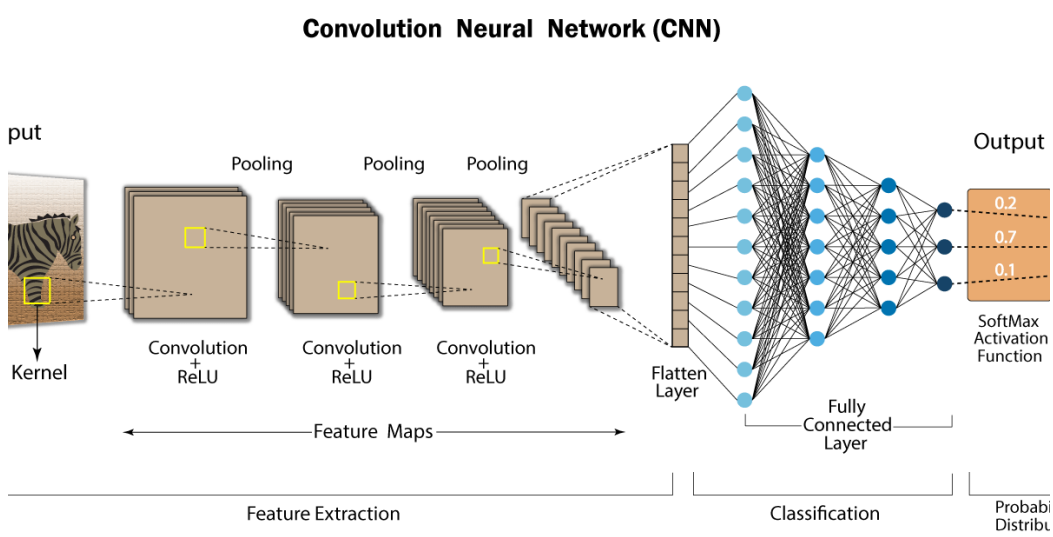

Figure 15 - Example of all four stages of a CNN

2.10.3 Training a Neural Network

Training deep networks is commonly approached as an optimization problem, where a supervised cost function is defined on the output layer relative to the target, and gradient-based optimization is used to adjust the network's weights and biases. However, deep networks often underperform compared to networks with one or two hidden layers. Two hypotheses may explain this difficulty. The first is that gradient descent can become trapped in poor local minima or plateaus of the non-convex training criterion, and the increased number of layers may lead to more such basins. To mitigate this, it has been suggested to train networks in a constructive manner, adding neurons or layers one at a time. While this approach can effectively handle complex functions, it often leads to overfitting in practical tasks. The second hypothesis posits that deep networks possess multiple basins of attraction in their parameter space, yielding solutions with low training error but varying generalization performance. Even if gradient descent finds a good minimum in terms of training error, this does not guarantee good generalization. While model selection via cross-validation can address this, if good generalization configurations are scarce, the training process may fail to find them. As discussed in this paper, unsupervised initialization can help direct the training process to favorable basins of attraction, improving performance on both the training and test sets.

2.10.4 Localization Techniques

Point localization is the process of accurately identifying specific points within an image, such as the swirl point in fingerprint images. Image regression models are particularly effective for this task, as they predict continuous values (like pixel coordinates) instead of discrete class labels.

In point localization using regression models, Convolutional Neural Networks (CNNs) are typically employed to extract spatial features from the image, enabling precise localization of specific points. These models are trained on labeled data, where the target points are known, and use regression loss functions, such as Mean Squared Error (MSE), to minimize the discrepancy between predicted and true point locations.[30]

One common approach is direct regression, where the model directly predicts the coordinates (x, y) of the target point. This method offers a straightforward and efficient solution for point localization, although it may be more susceptible to noise compared to other methods like

heatmap-based regression. Despite this, direct regression remains a popular choice due to its simplicity and faster inference time.

2.11 ZK9500 USB Fingerprint Scanner

The ZK9500 is an advanced optical fingerprint scanner developed by ZKTeco, designed primarily for desktop-based fingerprint registration. Its compact form factor and low power consumption make it particularly suitable not only for PCs but also for integration with Android tablets and mobile devices, expanding its versatility in mobile and embedded applications.[32]

What distinguishes the ZK9500 is its ability to detect and capture fingerprints rapidly with high-resolution output, which ensures clarity in the captured ridge patterns. Importantly, the device can extract fingerprint images in both RAW and BMP formats without any internal post-processing. This is a significant feature for developers and researchers, as it provides access to unprocessed biometric data—crucial for custom algorithm development, image enhancement, minutiae extraction, or machine learning pipelines where control over the raw data is essential.

ZKTeco also provides SDKs (Software Development Kits) to support a range of platforms, allowing developers to integrate the scanner easily into their own applications—whether for identification systems, time attendance, secure access, or research purposes. This combination of open data access, portability, and SDK support makes the ZK9500 a flexible and developer-friendly biometric capture solution which can be seen in fig 16.[32]

Figure 16 - ZK9500 Fingerprint Infra-red Scanner

3. Applied Methodology – Implementation

We will divide the Fingerprint Recognition System Project into 3 main parts.

Part 1 – Getting a fingerprint sample.

Part 2 – Preprocessing, Postprocessing procedure and feature extraction algorithms.

Part 3 – Fingerprint information extraction and evaluation.

Part 4 – Suggested Matching approaches.

For convenience MATLAB function names will be displayed as mathematical equations such as: $function$

3.1 Acquiring a fingerprint sample

In the initial phase of the project, fingerprint samples are acquired using the ZK9500 infrared optical fingerprint scanner developed by ZKTeco. This device is connected to a computer via USB and is controlled through ZKTeco's official Software Development Kit (SDK), which allows programmatic access to fingerprint capture functionalities.

To collect a sample, the user places the desired finger onto the scanner. The SDK is configured to capture and save the fingerprint image in BMP format to a predefined directory on the local system. Each fingerprint image is systematically named to encode identifying information about the sample. The naming convention follows the pattern:

<name>_<finger-index>_<sample-number>.bmp

For example, the filename ofir_4_1.bmp refers to the first captured sample of Ofir's fourth finger from the left. This naming structure ensures clarity and traceability when handling multiple samples from different individuals and fingers. The BMP format is chosen to preserve the raw ridge structure without any embedded processing, allowing full control over later stages of the image analysis pipeline.

Figure 17 – Example of fingerprint sample Ofir_7_1

3.2 Pre & Post processing procedure and feature extraction algorithms

3.2.1 Loading the Image

In the next stage, the fingerprint sample is loaded into MATLAB for further processing. For demonstration and testing purposes throughout the project, the sample image ofir_7_1.bmp is used as a representative input. This image is loaded using MATLAB's $imread$ function, which reads the BMP file into a matrix representing the grayscale intensity values of the fingerprint.

The image matrix is stored in a variable to be used in subsequent preprocessing steps such as ROI extraction, minutiae detection, and matching analysis. This step is essential for integrating the raw fingerprint image into the full recognition pipeline developed in this project.

3.2.2 ROI Extraction

After loading the fingerprint image in BMP format, the next step in the pipeline is to isolate the Region of Interest (ROI) from the raw grayscale fingerprint sample. This is achieved through a dedicated MATLAB function named $extract\_fingerprint\_roi$, which applies a sequence of morphological and image processing operations to suppress the background and enhance the central fingerprint pattern, which contains the most relevant features for later analysis and matching.

The function receives the fingerprint image as a grayscale matrix. The first operation is adaptive binarization, which converts the grayscale image into a binary one by applying locally adaptive thresholding. This approach is chosen over global thresholding to handle variations in

illumination and contrast across different fingerprint samples. A sensitivity value of 0.55 was empirically selected to balance noise suppression with ridge preservation.

Following binarization, thinning is applied using morphological skeletonization ($bwmorph$ with 'thin' operation), which reduces the ridge lines to single-pixel width. This simplification facilitates accurate subsequent morphological operations while preserving the essential topological features of the fingerprint.

To improve connectivity between ridges and fill small gaps, morphological closing is applied with a disk-shaped structuring element of radius 6. This operation effectively bridges small breaks between ridge structures that may have occurred during thinning or binarization.

After closing, the binary image may still contain internal voids or holes within the ridge areas. These are eliminated using the $imfill$ function with the 'holes' option, ensuring that each ridge segment is a solid region rather than a fragmented one. This is crucial for defining a consistent ROI.

Next, morphological opening is performed using a smaller structuring element (disk of radius 3) to eliminate small noise artifacts and spurious structures that may not belong to the genuine fingerprint area. This step enhances the clarity of the overall ROI by discarding isolated dots or small clusters.

Finally, the clean binary image is subjected to erosion using a disk of radius 12 to shrink the ROI and exclude noisy edges or partially captured regions near the border of the scanner. The resulting mask accurately outlines the fingerprint's high-confidence region.

The final ROI mask is then applied to the original grayscale fingerprint image, setting all background pixels to zero and retaining only the area identified as the fingerprint. The result is a pre-processed fingerprint image with the background removed and only the significant ridge area retained, which can now be passed to further modules such as minutiae detection, feature extraction, or classification, results in fig 18).

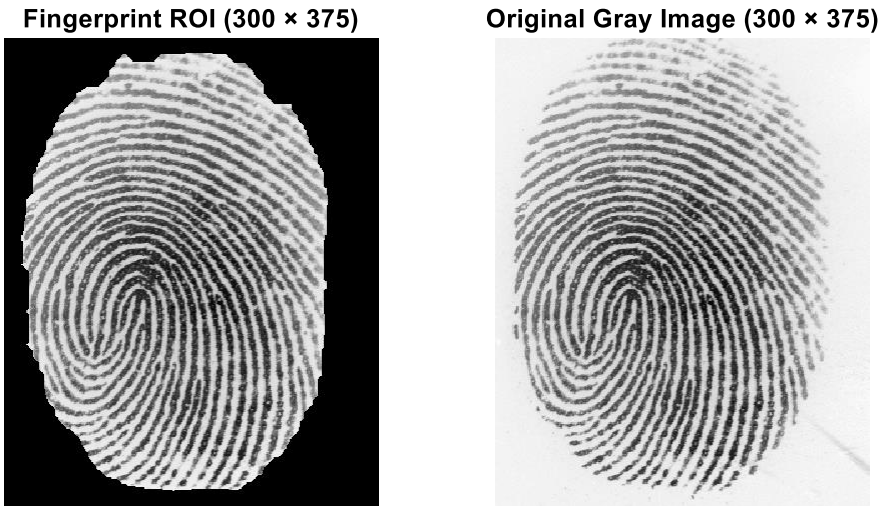

Figure 18 – Region of Interest of Gray scaled fingerprint

3.2.3 Creating Average level and Subtraction

The code processes each pixel in the grayscale image by replacing its value with the **average** value of the pixels in its local neighbourhood. The size of the neighbourhood is defined by `windowSize`, which creates a square region centered at the pixel (with `windowSize` 5 in this case, so a 11×11 window). This has the effect of suppressing local intensity variations, such as noise and sharp edges, which makes the image appear smoother.

Here's the breakdown:

- For each pixel at location (i, j) a square neighbourhood around that pixel is considered.
- The neighbourhood size is $(2 \cdot windowSize + 1) \times (2 \cdot windowSize + 1)$ centered at (i, j).
- The mean of all pixel values in this window is computed.

- The result replaces the original pixel value in the output image `After_low_Pass`.

This operation mimics the behaviour of a mean filter, a basic form of low pass filtering in image processing.

The math equations translated to code act as the following:

Let the original image be denoted as a 2D function:

$$I_{(i,j)} \quad for \quad i \in [1, R], \qquad j \in [1, C] \tag{15}$$

where R and C are the number of rows and columns of the image.

We define the output smoothed image as:

$$I_{low}(i,j) = \frac{1}{N} \cdot \sum_{m=-w}^{w} \sum_{m=-w}^{w} I(i+m, j+n) \tag{16}$$

where:

- $w = window\ Size$
- $N =$ number of valid pixels in the window (usually $(2w+1)^2$, unless edges clip the window)
- The sums are only taken over indices where $(i+m, j+n)$ are within image bounds.

To extract the high-frequency content—such as fingerprint ridge edges and fine structures—we subtract the smoothed image from the original grayscale image:

$$I_{high}(x,y) = I(x,y) - I_{low}(x,y) \tag{17}$$

As result we get the following, in (fig 19):

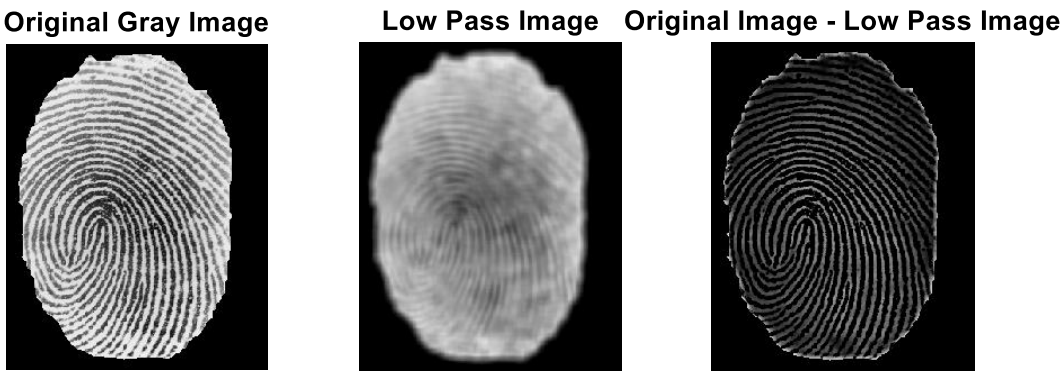

Figure 19 – Subtraction of Low pass image

3.2.4 Linear contrast stretch transformation

After extracting the high-frequency components via the subtraction of the smoothed image from the original, the resulting subtracted image may still have limited dynamic range. This can make important features appear dim or poorly contrasted, reducing their visibility for further processing. To address this, we apply contrast stretching, a linear intensity transformation that expands the pixel value range to fully utilize the 8-bit grayscale spectrum $[0,255]$.

Mathematically, let the minimum and maximum intensities in the difference image be I_{min} and I_{max}, respectively. Each pixel value $I(x,y)$ in the subtracted image is then rescaled using the formula in eq. (18):

$$I_{adjusted}(x,y) = \frac{I(x,y) - Imin}{I_{max} - I_{min}} \cdot 255 \tag{18}$$

This transformation linearly maps the lowest intensity in the image to 0 and the highest to 255, effectively amplifying the contrast between dark and bright regions as seen in fig 20.

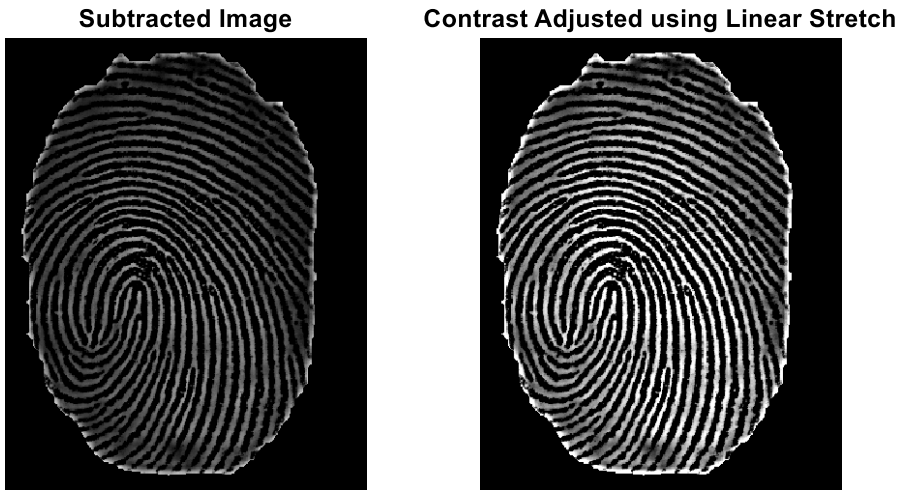

Figure 20 – Linear Contrast stretch result

3.2.5 Binarization thresholding

To isolate the fingerprint's ridge structure more effectively, we apply local thresholding using the Niblack method, which is particularly suited for non-uniform lighting and locally varying contrast, both common in fingerprint images. Unlike global thresholding techniques that apply a single threshold across the entire image, Niblack thresholding calculates a unique threshold for each pixel based on its local neighbourhood statistics.

First, the contrast-enhanced image (`ContrastedImage`) is converted to type (`double`) precision to ensure numerical stability in subsequent operations.
A square window of size $w \times w$ (with $w = 5$) is used to define the local neighbourhood around each pixel. Within each window, we compute:

1. The local mean $\mu(x, y)$ using a box filter:

$$\mu(x,y) = \frac{1}{w^2} \sum_{i,j \in N(x,y)} I(i,j) \tag{19}$$

2. The local standard deviation $\sigma(x,y)$ from the mean and mean of squares:

$$\sigma(x,y) = \sqrt{\frac{1}{w^2} \sum I(i,j)^2 - \mu(x,y)^2} \tag{20}$$

3. Niblack threshold $T(x,y)$ for each pixel is then computed using:

$$T(x,y) = \mu(x,y) + k \cdot \sigma(x,y) \tag{21}$$

where k is a user-defined constant, typically negative $(k = -0.1)$. A pixel is classified as foreground (ridge) if its intensity exceeds $T(x,y)$, resulting in a binary image as seen in fig 21, the binary image is also the end of the preprocessing part.

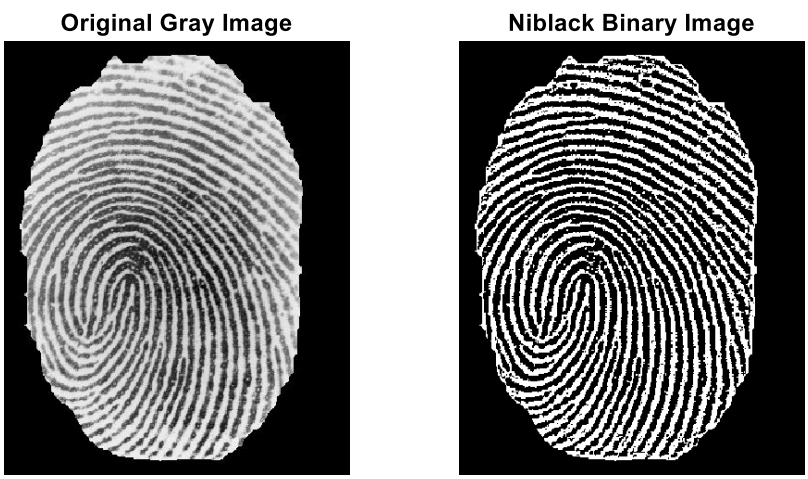

Figure 21 – Binary Image of Ofir_7_1

3.2.6 removing white and black single pixels and 2x2 white cubic pixels

To clear noise from the binarized fingerprint image, an additional post-processing step is performed to eliminate isolated black and white pixel noise**.** Ensuring both foreground and background are free of minimal, uninformative elements.

The strategy: we apply Convolution masks to detect and remove:

- Single black and white pixels (1×1)

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} \quad OR \quad \begin{matrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Black – 0, White – 1

- Small clusters of black and white pixels (2×2) squares

$$\begin{matrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} \quad OR \quad \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{matrix}$$

Black – 0, White – 1

To reuse the same convolution-based logic, the binary image is inverted, effectively converting black pixels ($value$ 0) to white ($value$ 1). This allows the same masks and conditions to be applied.

Once inverted, the same algorithm runs:

- A 3×3 convolution checks for white pixels with only a single active neighbour. If such pixels are found, they are considered isolated and removed.

- A 4×4 convolution scans for sparsely clustered blocks ($sum \leq 4$), and those small white patches are also removed, corresponding to 2×2 black clusters in the original image.

$$Mask\ 3x3\ and\ 4x4:$$

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \qquad \begin{matrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{matrix}$$

This loop is repeated twice for thorough cleanup. After processing, the image is inverted back, restoring the original polarity while maintaining the improved clarity.

Together, these post-processing routines significantly reduce the presence of noise and small-scale inconsistencies in the binary image, both from white and black regions as seen in (fig 22). This ensures a cleaner, topologically coherent binary representation of the fingerprint ridges, which is essential for precise morphological analysis and minutiae extraction in subsequent stages.

Figure 22 – 1x1 and 2x2 noised pixels cleaned

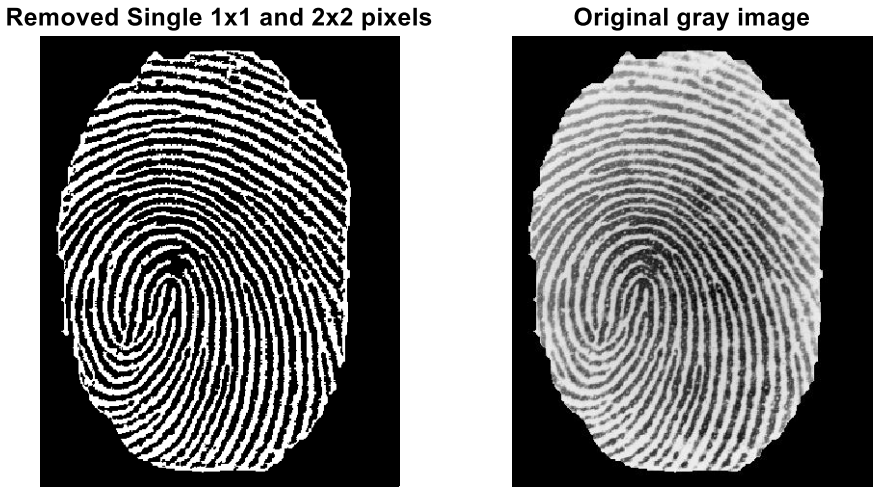

Figure 23 – Original sample vs cleaned binary image

3.2.7 Pruning edges process

To finalize the cleaning process of the binarized fingerprint image, a pruning step is applied to remove small edge branches which are thin protrusions that are loosely connected to the main fingerprint structure and could lead to false minutiae detection.

This is done in two passes using the same algorithm, by simply inverting the image between passes:

- In the first pass, the image is processed to remove black edge branches.

Black branches are defined in the middle of a matrix of $3x3$ by:

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

White branches are defined in the middle of a matrix of $3x3$ by:

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

- The binary image is convolved with a 3×3 mask to compute the number of white neighbours for each pixel. If a pixel has exactly five white neighbours (i.e., the sum in a 3×3 neighbourhood equals 5), it is considered a weakly connected branch and is transformed to its opposite.

- Then, the image is inverted (i.e., black becomes white and vice versa), and the same procedure is applied again. This second pass removes white edge branches by identifying pixels that now represent weak white protrusions in the inverted space. These are also pruned using the same neighbourhood condition.

By alternating between the original and inverted images with the same pruning rule, both black and white spurious branches are effectively removed, resulting in a cleaner, more coherent fingerprint structure as seen in (fig 24).

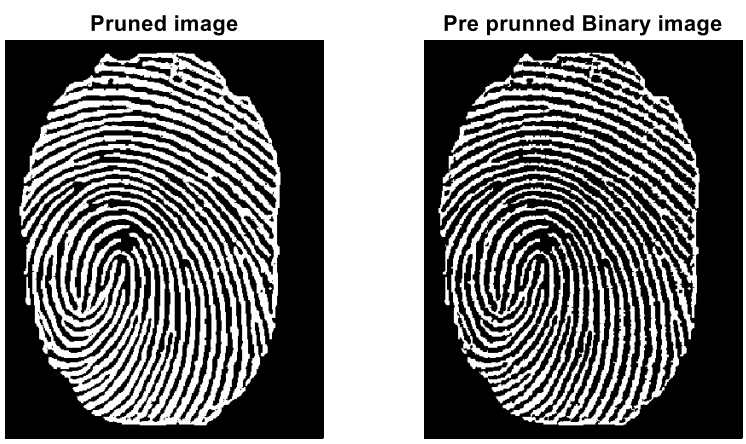

Figure 24 – Pruned image (left) Pre pruned (right)

3.2.8 Pre-Thinning process

To finalize the cleaning of the binarized fingerprint image before thinning, the pruned image is going through stages **3.2.6** and **3.2.7** again for better noise cancelation, the pre-thinned image is given in (fig 25).

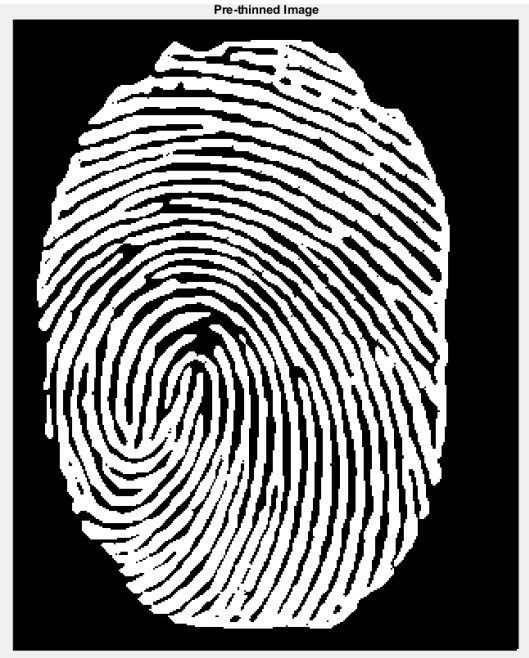

Figure 25 – Pre-thinned Image

3.2.9 Thinning Process

The Thinning process from **2.3.1** applied on the pre-thinned image, from the thinning stage we receive the thinned image in (fig 26)

Figure 26 – Thinned (not finished) image

3.2.10 Removing Branches Created by the Thinning process

The image has been successfully thinned, but this introduces a new issue known as *spurs*—small, unwanted branches that remain after thinning. To clean these spurs from the thinned image, we apply additional image processing techniques, specifically aimed at removing these unnecessary extensions. This is achieved by repeatedly shortening the spurs over five iterations, effectively reducing the length of minor ridge branches. Since the final algorithm used is not affected by these small cutoffs, this approach provides a practical and efficient solution.

This is done by scanning each pixel's surroundings and identifying patterns that match these unwanted features, such as a single black pixel surrounded by white ones. Once identified, these spurs are erased.

The process is repeated several times to gradually shorten and eliminate these small branches. This helps clean the thinned fingerprint skeleton and prepares it for further analysis, without significantly affecting the meaningful ridge paths as seen in fig 27.

Figure 27 – thinned image without spurs

3.2.11 Removing False Minutiae

After applying the thinning operation, an additional cleaning step is performed to remove unwanted structures or artifacts that often remain attached to the main ridge skeleton. These artifacts can distort the final result or introduce false minutiae points. The approach here is a template-based post-processing that uses structural masks to identify and eliminate certain ridge patterns.

A set of six 3×3 binary masks is defined. These masks are handcrafted to detect small artifacts and spurs (unwanted branches) or extra lines that typically attach to the main ridge structure in predictable shapes.

The masks represent specific binary configurations of pixels like these:

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{matrix} \qquad \begin{matrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

For each mask:

- A 3×3 sliding window moves over the image.
- If the exact pattern in the mask is found in the current window (i.e., a spur or branch is detected), the center pixel is removed by setting it to 0.
- This loop is performed over all pixels (excluding the border) and for all six masks.

Mathematically, this process checks if:

$$window(i,j) = M_k \Rightarrow set\ pixel(i,j) = 0 \tag{22}$$

for every pixel (i,j) and for each structural mask M_k , where $k = 1,2,\dots,6$

Removal ensures that only specific undesired structures are eliminated; while preserving the integrity of the genuine ridge lines, we therefore get fig 28.

Figure 28 – Spurs Completely removed

3.2.12 Detecting Bifurcations

The detection is based on matching small 3×3 local neighborhoods against a set of 13 predefined patterns, each representing a common bifurcation structure — where a single ridge splits into three branches.

The method involves sliding a 3×3 window across the image and checking if the pattern within the window matches any of the bifurcation templates. If a match is found, the center pixel of that window is marked as a bifurcation point.

The result is a new binary image, the bifurcation map, highlighting all detected bifurcation locations with red pixels.

The Bifurcation masks:

$Mask\ 1$	$Mask\ 2$	$Mask\ 3$	$Mask\ 4$
[1 0 0]	[1 0 1]	[0 0 1]	[0 1 0]
[0 1 1]	[0 1 0]	[1 1 0]	[0 1 0]
[1 0 0]	[0 1 0]	[0 0 1]	[1 0 1]

$Mask\ 5$	$Mask\ 6$	$Mask\ 7$	$Mask\ 8$
[1 0 0]	[0 0 1]	[0 1 0]	[0 1 0]
[0 1 1]	[1 1 0]	[1 1 0]	[0 1 1]
[0 1 0]	[0 1 0]	[0 0 1]	[1 0 0]

$Mask\ 9$	$Mask\ 10$	$Mask\ 11$	$Mask\ 12$
[0 0 1]	[1 0 0]	[1 0 1]	[1 0 1]
[0 1 0]	[0 1 0]	[0 1 0]	[0 1 0]

[1 0 1] [1 0 1] [1 0 0] [0 0 1]

$Mask$ 13

[0 0 1]

[0 1 0]

[1 0 1]

These masks cover various orientations and configurations of bifurcation points that may appear in a skeletonized fingerprint image.

After running the convolution with the predefined masks we achieve (fig 29):

Detected Bifurcation Points on Thinned Image

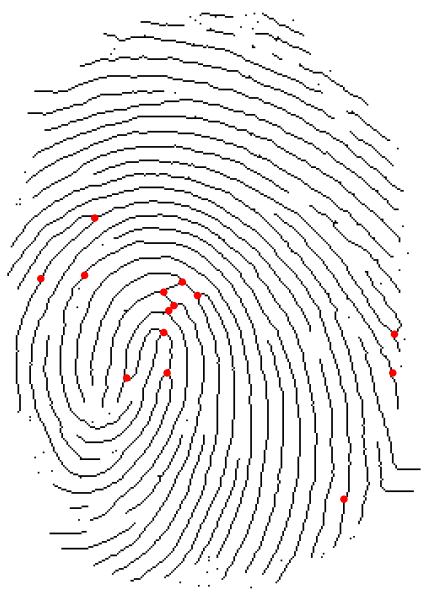

Figure 29 – Bifurcations Detected in Thinned Image

The image with bifurcation points marked demonstrates a high-quality detection outcome. The marked bifurcations closely correspond to true ridge-splitting locations in the fingerprint pattern.

In most cases, especially when using standard thinning and detection techniques, the proportion of true minutiae points out of all detected minutiae is around 80%. However, in this case, due to careful preprocessing, cleaning, and thinning, the detection reached an accuracy of approximately 90% true bifurcations, meaning that the vast majority of detected points are indeed real minutiae. This indicates a notably robust and effective detection pipeline additionally we applied the extraction of ROI function once again to filter out the edges caused by the first application in order to have a clearer fingerprint image.

3.2.13 Detecting Ridges

The detection is performed using a sliding 3×3 window that examines each pixel and its immediate neighbourhood. The algorithm specifically looks for small patterns (masks) that represent two connected pixels, which form part of a ridge segment. These patterns cover horizontal, vertical, and diagonal orientations to ensure all possible local directions are considered.

Examples of 2 connected pixels in a $3x3$ mask:

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{matrix} \qquad \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

After applying the mask on the thinned image, we get fig 30.

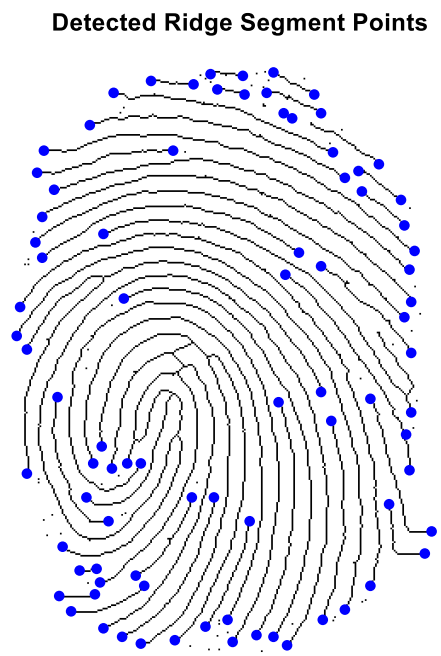

Figure 30 – Ridge Detected in Thinned Image

3.2.14 Bifurcation and Ridges Minutiae Data

By combining both ridge endings and bifurcations detected in the thinned fingerprint image, we obtained the result shown in (fig. 31). This unified representation captures all relevant minutiae points. In the next stages, we will export this information using the suggested format to enable effective matching and identification.

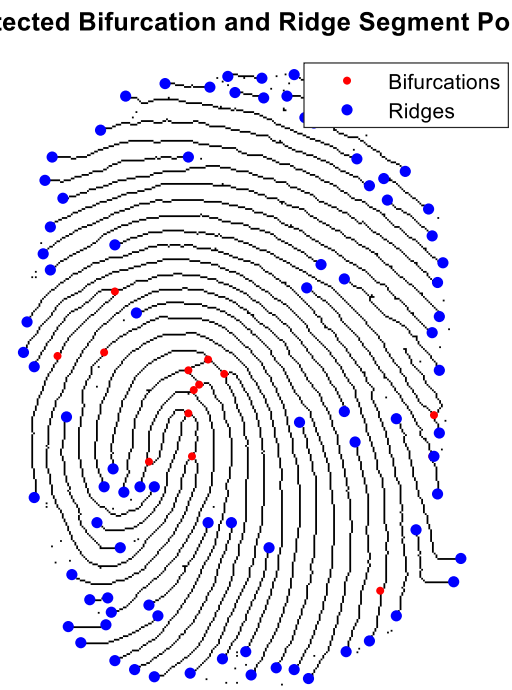

Figure 31 – Bifurcation and Ridges detected as Minutiae points in the fingerprint successfully

In (fig. 32), we overlaid the minutiae data on the original grayscale sample to better visualize the results. The extracted data appears to be highly accurate. While some noise is present outside the fingerprint area, it can be easily removed or cropped out without affecting the core information.

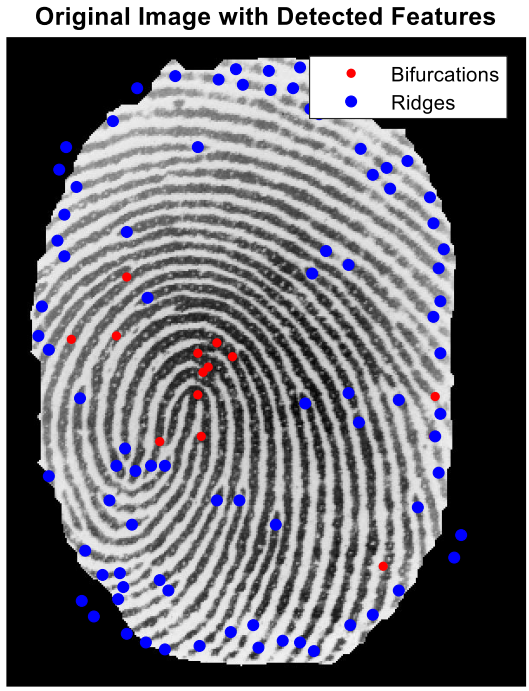

Figure 32 – Original Sample with detected features

3.2.15 Fingerprint information extraction

In this last stage, the goal is to extract the information of detected features, such as bifurcations and ridge locations for later suggested matching algorithms.

First, the map size is defined as a 2D array with 375 rows and 300 columns, representing the dimensions of the final binary map. This map acts as a "canvas" where the locations of the detected features will be marked.

The coordinates of the bifurcation points and ridge segments are provided as pairs of row and column indices. The task is to translate these coordinates into binary values on the map. For each feature, the corresponding coordinates are set to 1, while all other positions in the map remain 0. This process results in two binary maps: one for bifurcations, where 1s represent the locations of

bifurcation points, and another for ridges, where 1s represent the locations of detected ridge segments. All other areas in both maps are filled with 0. These binary maps are then ready for further use, such as visualization or analysis in subsequent stages of the process.

To counter the problem of tilting in fingerprint recognition—where fingerprints may be captured at unknown angles or localized differently across the scanner—we focus on extracting information that remains invariant under such conditions. One such invariant feature is the minimal distance between each pair of minutiae points. These distances are calculated using a Breadth-First Search (BFS) algorithm as explained in **2.8**. The algorithm begins by scanning the grid for the first '1' value (representing minutiae point) and then searches for its nearest neighbour by examining the 8-connected neighbours. If no neighbour is found immediately, the search expands to neighbours of already visited points, continuing outward until another '1' is located.

Once the minimal distances for each minutiae point are obtained, they are stored in a vector called *distances*. Initially, the algorithm generates a vector using only ridge points, then a separate vector for bifurcation points, and finally a combined vector using both ridges and bifurcations. This results in three distinct vectors, each representing an identity signature for the fingerprint.

However, due to potential variations and interferences between different scans, these vectors may not be directly comparable. To address this, we normalize each distance vector into a histogram that captures the statistical behaviour of the fingerprint. Each histogram reflects key features such as the mean, standard deviation, maximum bin count, total number of points, and the frequency of occurrence for each distance range. These normalized histograms serve as robust, compact ID representations suitable for matching algorithms.

To enable structured analysis and external use, the data should also be exported as a CSV file with the following six columns:

x	y	type	Distance ridges	Distance bifurcations	Distance combined
column	row	ridge/bifurcation	distance to nearest ridge	distance to nearest bifurcation	combined distance value

Each row in the CSV represents one detected minutiae point along with all six corresponding attributes.

Additionally, we can create a histogram of the combined distances of bifurcations and ridges over their occurrences and ridges and bifurcations separately. These histograms (displayed in fig 33. 34. And 35.) can be used to generate an identity in the form of a distribution profile, summarizing the fingerprint pattern. By calculating the mean and standard error of the histograms and representing the bin values (with their corresponding occurrences) as a vector, we can construct an ID vector. This ID vector can then be used in later suggested matching algorithms as a compact and descriptive feature representation.

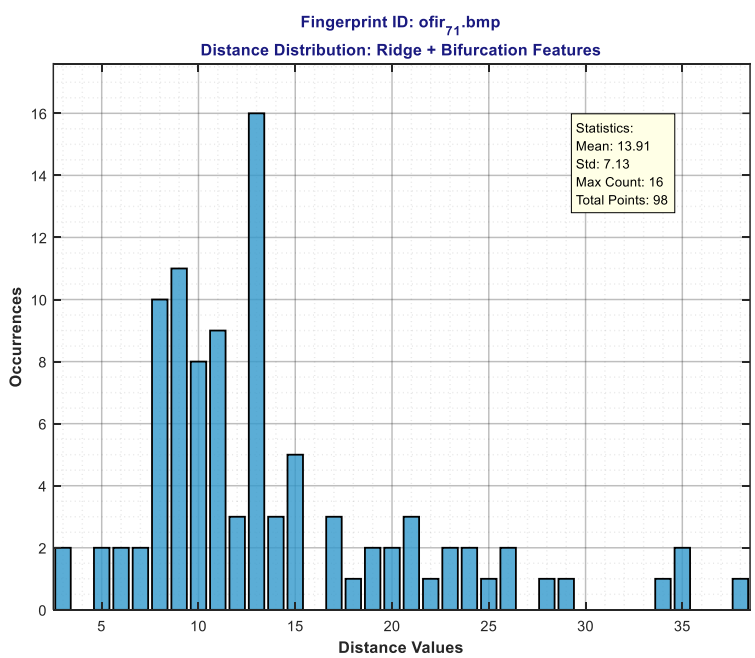

Figure 33 – Histogram representation of detected ridges and bifurcations combined

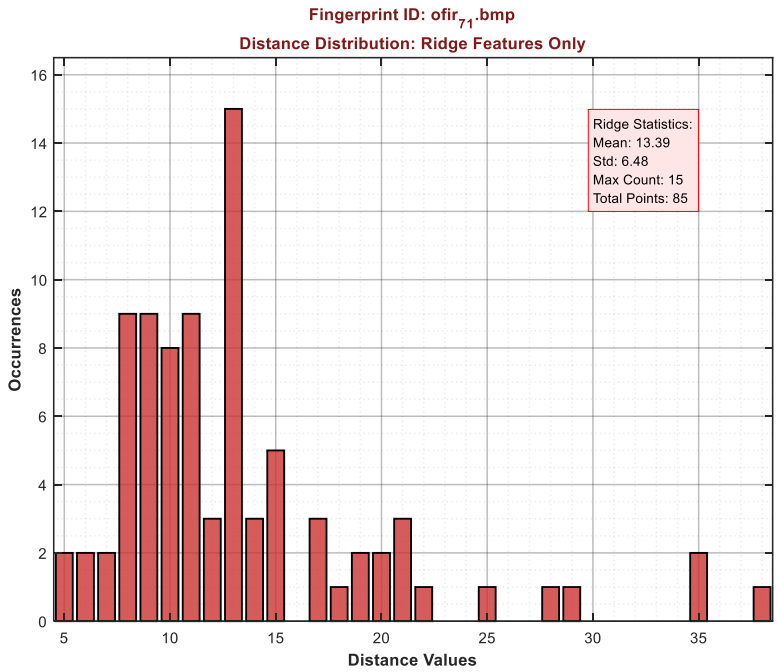

Figure 34 – Histogram representation of detected ridges

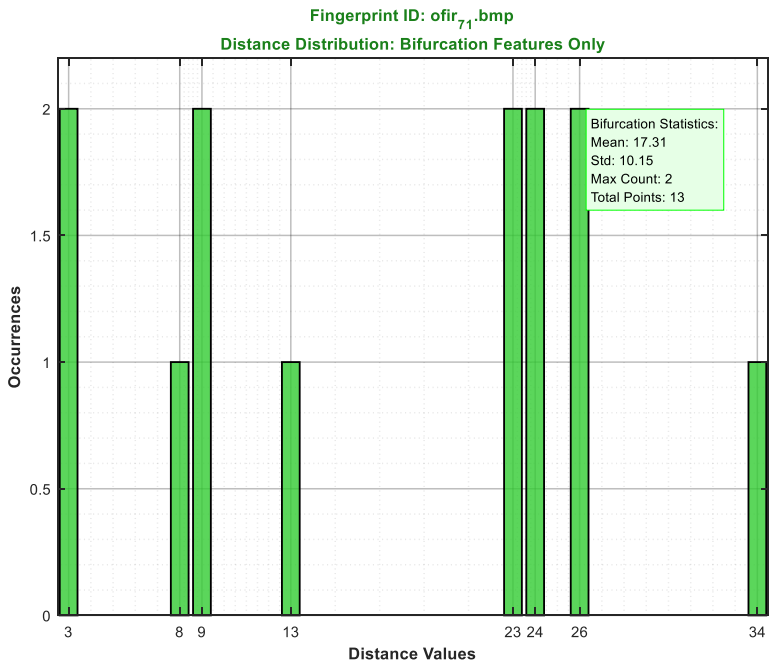

Figure 35 – Histogram representation of detected bifurcations

Example of CSV file containing the mentioned information:

Type 3 – Bifurcation, Type 1 – Ridge ending

Dist_ridge, dist_bif and dist_combined should be taken as a vector of column.

X,y and type reflect the type detected at x,y coordinates.

X	Y	Type	Dist_ridge	Dist_bif	Dist_combined
14	107	3	8	46	8
15	257	3	17	13	21
21	77	3	11	13	17
24	64	3	8	30	11
32	326	3	9	34	8
35	335	3	12	26	9
38	175	3	9	26	12
58	351	3	16	9	9
64	173	3	28	8	16
70	20	3	14	9	28
70	139	3	10	3	14
73	359	3	14	3	10
89	234	3	36	13	14
90	364	3	10	23	25
111	183	3	7	24	13
111	207	3	8	23	10
113	231	3	7	24	7
114	194	3	10	74	8
117	191	3	29	57	7
122	177	3	14	NaN	10
131	185	3	9	NaN	29
218	306	3	38	NaN	14
247	231	3	8	NaN	9
248	208	3	6	NaN	13

270	301	3	8	NaN	38
273	288	3	6	NaN	8
19	173	1	25	NaN	6
21	156	1	36	NaN	8
22	107	1	18	NaN	6
25	181	1	13	NaN	9
30	118	1	10	NaN	9
34	103	1	13	NaN	28
34	127	1	9	NaN	18
41	87	1	13	NaN	13
43	209	1	9	NaN	10
46	297	1	13	NaN	13
56	311	1	13	NaN	9
60	268	1	11	NaN	13
62	49	1	11	NaN	9
64	248	1	21	NaN	28
65	325	1	10	NaN	13
66	310	1	12	NaN	13
68	318	1	8	NaN	11
69	238	1	25	NaN	11
70	113	1	5	NaN	21
73	282	1	13	NaN	10
75	251	1	8	NaN	9
82	151	1	5	NaN	8
84	248	1	13	NaN	19
89	314	1	11	NaN	25
92	248	1	11	NaN	4
94	320	1	13	NaN	4
98	23	1	17	NaN	8
111	64	1	13	NaN	8

112	352	1	17	NaN	8
122	268	1	15	NaN	5
123	25	1	8	NaN	8
130	344	1	13	NaN	7
133	19	1	8	NaN	7
135	268	1	10	NaN	3
137	28	1	21	NaN	11
143	340	1	15	NaN	3
146	353	1	12	NaN	11
152	20	1	20	NaN	13
153	31	1	15	NaN	17
156	282	1	15	NaN	13
160	349	1	9	NaN	17
166	30	1	13	NaN	15
170	350	1	11	NaN	13
173	212	1	20	NaN	23
176	42	1	9	NaN	8
177	137	1	21	NaN	13
178	355	1	13	NaN	8
181	45	1	11	NaN	10
185	124	1	NaN	NaN	21
194	31	1	NaN	NaN	15
198	42	1	NaN	NaN	16
198	132	1	NaN	NaN	17
198	206	1	NaN	NaN	12
199	340	1	NaN	NaN	11
204	223	1	NaN	NaN	20
205	65	1	NaN	NaN	11
212	80	1	NaN	NaN	15
212	334	1	NaN	NaN	9

220	76	1	NaN	NaN	13
222	88	1	NaN	NaN	11
227	210	1	NaN	NaN	19
227	320	1	NaN	NaN	9
232	72	1	NaN	NaN	21
238	272	1	NaN	NaN	13
245	93	1	NaN	NaN	11
247	108	1	NaN	NaN	52
247	162	1	NaN	NaN	38
248	231	1	NaN	NaN	NaN
250	134	1	NaN	NaN	NaN
250	252	1	NaN	NaN	NaN
251	153	1	NaN	NaN	NaN
251	183	1	NaN	NaN	NaN
251	218	1	NaN	NaN	NaN
253	123	1	NaN	NaN	NaN

At this point, we extracted the minutiae points and organized the information as follows:

- A CSV file containing the six columns as previously discussed: x-coordinate, y-coordinate, type of minutiae (ridge or bifurcation), distance to nearest ridge, distance to nearest bifurcation, and combined distance.
- Three histograms representing the distribution of distances for ridge points, bifurcation points, and their combined values.
- For each histogram, we have calculated the following statistical metrics: the mean, the standard deviation, the maximum count (i.e., the bin with the highest frequency), and the total number of minutiae points contributing to the histogram.

3.3 Suggested Matching approaches

This well-structured and detailed representation opens up several possibilities for fingerprint matching. The richness of the extracted information enables multiple matching strategies. For instance, matching can be based on direct comparison of the histogram shapes, evaluating similarities in statistical features such as means and standard deviations, or using more advanced methods such as vector-based distance measures (Euclidean, Intersection, Correllation or cosine similarity) between the ID vectors. Additionally, the structured format—separating ridges, bifurcations, and combined data—allows for selective matching depending on which features are more reliable in a given scan. The high level of accuracy and organization in this data ensures that it can be reliably used across different scans, even in the presence of noise, rotation, or partial captures, making it a strong foundation for robust fingerprint identification algorithms.

Moreover, this data format is well-suited for machine learning and neural network-based approaches. For example, the histogram vectors and statistical descriptors can serve as feature inputs for classifiers such as Support Vector Machines (SVMs), Random Forests, or k-Nearest Neighbours (k-NN), which can be trained to recognize and differentiate between fingerprints. Neural networks, particularly convolutional neural networks (CNNs) or fully connected deep networks, can be trained on these vectors to learn complex, non-linear relationships and improve matching accuracy. Using labelled fingerprint datasets, these models can generalize over variations in orientation, noise, and partial data, offering a powerful, scalable approach for real-world fingerprint recognition systems.

3.4 Matching attempt using ‘bhattacharyya” Histogram comparison method

Using a majority vote approach, three samples of the same finger are histogram-matched against a fourth sample. Each comparison yields a score: matches above 70% receive 1 point, and matches above 85% receive 2 points. The points are summed across all comparisons, with a total score of 6 out of 6 indicating 100% authentication.

I divided the comparison results by assigning a weight of 0.5 to the combined ridge and bifurcation histogram, and 0.5 to the ridge-only histogram. These two values were then combined into a single result. Each histogram was compared with the corresponding histograms

from the other three samples using the Bhattacharyya distance method, a statistical histogram comparison technique developed by *Anil Kumar Bhattacharyya.*

The result of the fingerprint recognition was 5/6 total votes therefore falls under the category of STRONG MATCH, the 4th fingerprint is indeed matched against the other 3 samples successfully as seen in fig 36.

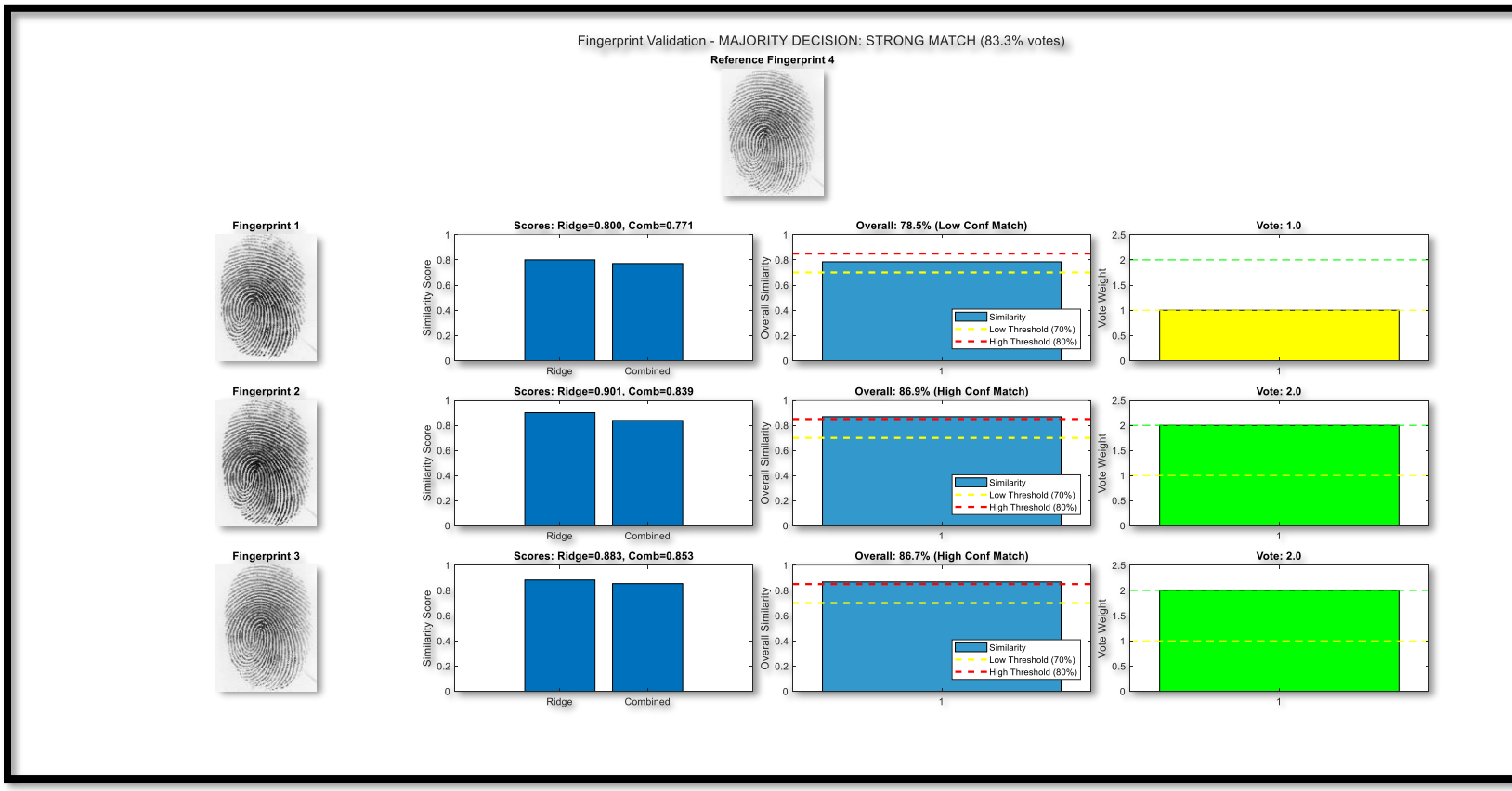

Figure 36 - Results of comparison of fingerprints using histogram comparison method

4. Summary

We successfully developed a fingerprint minutiae data extraction algorithm with high accuracy and quality, implemented entirely in MATLAB. The algorithm follows a structured and logical sequence of image processing and signal analysis steps to ensure reliable feature extraction.

The process begins with the application of morphological operations to enhance fingerprint structures and reduce background noise. This is followed by a binarization stage using the Niblack method, which effectively separates the fingerprint ridges from the background. Next, 2D convolution is applied with smoothing and cleaning masks to further refine the image.

Once the image is cleaned, a thinning process is performed to reduce the ridge lines to a single-pixel width, making them suitable for minutiae detection. A pruning step is applied afterward to eliminate false endings and small artifacts. Feature detection is then carried out using convolution with custom-designed masks to locate ridge endings and bifurcations accurately.

With the features detected, a Breadth-First Search (BFS) algorithm is used to calculate minimal distances between minutiae points, providing spatial relationship data that is robust to rotation and translation. The extracted minutiae information, including coordinates, type, and distance metrics, is exported into a structured CSV file. Finally, histograms and statistical analyses are generated to summarize the distribution of features, capturing descriptors such as mean, standard deviation, maximum frequency, and total point count. This well-structured pipeline reflects a robust and effective approach to fingerprint analysis.

5. Conclusions and Future Improvement

This project successfully achieved the development of a reliable and accurate fingerprint minutiae data extraction algorithm using MATLAB.

One key conclusion from the project is that a structured, step-by-step approach based on classical image and signal processing methods can yield high-quality feature extraction without requiring large training datasets. Additionally, the distance-based features captured in this work are resilient to noise, partial scans, and misalignments, providing a strong foundation for fingerprint identification and matching.

For future improvement, we propose incorporating the angle at which a ridge ending breaks as an additional metric. This orientation information could significantly enhance the uniqueness of the extracted minutiae features and improve the discriminatory power of the fingerprint representation.

Furthermore, the integration of neural networks and machine learning methods presents a promising avenue for advancement. Deep learning models, particularly convolutional neural networks (CNNs), could be trained to automatically detect minutiae and other structural patterns within fingerprints, reducing the need for handcrafted processing steps. Machine learning could also be used to study large datasets and uncover new statistical relationships within the fingerprint structure, optimizing and extending the set of metrics used for identification. By combining classical algorithms with learning-based techniques, future systems can achieve even greater accuracy, adaptability, and efficiency in fingerprint recognition.

6. References

[1] S. Greenberg, M. Aladjem, D. Kogan, I. Dimitrov, Fingerprint image enhancement using filtering techniques, in: International Conference on Pattern Recognition, Vol.3, pp.326-329, 2000.

[2] G. Aguilar, G. Sanchez, K. Toscano, M. Salinas, M. Nakano and H. Perez, "Fingerprint Recognition," *Second International Conference on Internet Monitoring and Protection (ICIMP 2007)*, San Jose, CA, USA, 2007, pp. 32-32, doi: 10.1109/ICIMP.2007.18.

[3] https://www.forensicsciencesimplified.org/prints/#:~:text=One%20of%20the%20most%20important,probation%2C%20parole%20and%20pardoning%20decisions

[4] https://www.sciencedirect.com/science/article/abs/pii/S2468170923000619

[5] Fingerprint matching Using Minutiae Extraction Techniques Akshay Sharma1, M.P.S. Chawla2* M.E., Department of Electrical Engineering, S.G.S.I.T.S., Indore, India Volume 2 Issue 1

[6] https://www.bayometric.com/fingerprint-image-quality/

[7] Bana, Sangram, and Dr Davinder Kaur. "Fingerprint recognition using image segmentation." *International Journal of Advanced Engineering Sciences and Technologies* 5.0 (2011): 1.

[8] Haralick, Robert M., and Linda G. Shapiro, *Computer and Robot Vision*, Vol. 1, Addison-Wesley, 1992.

[9] Kong, T. Yung and Azriel Rosenfeld, *Topological Algorithms for Digital Image Processing*, Elsevier Science, Inc., 1996.

[10] Lam, L., Seong-Whan Lee, and Ching Y. Suen, "Thinning Methodologies-A Comprehensive Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 14, No. 9, September 1992, page 879, bottom of first column through top of second column.

[11] Pratt, William K., *Digital Image Processing*, John Wiley & Sons, Inc., 1991.

[12] 2015 International Conference on Advances in Computer Engineering and Applications (ICACEA) IMS Engineering College, Ghaziabad, India 764 Fingerprint Feature Extraction Using Morphological Operations

[13] https://www.sciencedirect.com/science/article/pii/S2210832717301576

[14] https://www.geeksforgeeks.org/image-formats/

[15] C. Kang, "Research on Generation Algorithm of Bmp Format Two-dimensional Code Image," *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, Tianjin, China, 2009, pp. 433-436, doi: 10.1109/FSKD.2009.47.
keywords: {Error correction codes;IEC standards;ISO standards;Image generation;Code standards;Encoding;Image storage;Mobile handsets;DC generators;Character generation;two-dimensional},

[16] Luminance Based Conversion of Gray Scale Image to RGB Image 1G. Jyothi, 2CH. Sushma, 3D.S.S. Veeresh 1,2 ,3 Assistant Professor, Information Technology Bhoj Reddy Engineering College for Women, Hyderabad, India

[17] Xuling Liu, Duanqin Zhang, Jie Liu, Liangwen Wang, Songjing Li,

RGB Color Model Analysis for a Simple Structured Polydimethylsiloxane Pneumatic Micromixer, SLAS Technology, Volume 26, Issue 5 ,2021,

[18] Feature Extraction and Image Processing in Computer Vision 70 3 Basic Image Processing Operations, chrome extension://efaidnbmnnnibpcajpcglclefindmkaj/https://comp3204.ecs.soton.ac.uk/mark/fourth_edition_chap3_draft.pdf

[19] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://spie.org/samples/TT92.pdf

[20] Yang, Zhengxian & Zhang, Rui & Zhou, Yanxi & He, Jinlong & Shi, Jianwen & Zuo, Shikai. (2024). Summary of Document Image Binarization. 10.20944/preprints202401.1534.v1.

[21] W. Niblack, An Introduction to Image Processing, pp. 115–116, Prentice-Hall, Englewood Cliffs, NJ (1986)

[22] Basics of Image Processing — Vincent Mazet (Université de Strasbourg), 2020-2025 — CC BY-NC 4.0.

[23] https://medium.com/@timothy_terati/image-convolution-filtering-a54dce7c786b

[24] Luo, L., Wong, M. D., & Hwu, W.-m. W. (2010). An effective GPU implementation of breadth-first search. Retrieved from http://impact.crhc.illinois.edu/shared/papers/effective2010.pdf

[25] https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm

[26] https://svi.nl/ImageHistogram

[27] https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html

[28] Nasteski, Vladimir. (2017). An overview of the supervised machine learning methods. HORIZONS.B. 4. 51-62. 10.20544/HORIZONS.B.04.1.17.P05.

[29] Purwono, Ir & Ma'arif, Alfian & Rahmaniar, Wahyu & Imam, Haris & Fathurrahman, Haris Imam Karim & Frisky, Aufaclav & Haq, Qazi Mazhar Ul. (2023). Understanding of Convolutional Neural Network (CNN): A Review. International Journal of Robotics and Control Systems. 2. 739-748. 10.31763/ijrcs.v2i4.888.

[30] https://www.sciencedirect.com/science/article/pii/S0168169924008834

[31] Akbaba, Cihat & Polat, Adem & Göktürk, Dilek. (2023). Tracing 2D Growth of Pancreatic Tumoroids Using the Combination of Image Processing Techniques and Mini-Opto Tomography Imaging System. Technology in cancer research & treatment. 22. 15330338231164267. 10.1177/15330338231164267.

[32] chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://zkteco.eu/sites/default/files/content/downloads/zk9500_0.pdf

7. APPENDIX

7.1 Region of interest extraction function

```
function [roi_mask, fingerprint_roi, filtered_minutiae] =
extract_fingerprint_roi(adjustedImage, minutiae)
% Inputs:
%   adjustedImage - Grayscale fingerprint image
%   minutiae      - (Optional) Nx2 matrix of [x, y] minutiae coordinates
%
% Outputs:
%   roi_mask         - Binary mask for ROI
%   fingerprint_roi  - Image with background removed
%   filtered_minutiae - (Optional) Minutiae within ROI only

    bw = imbinarize(adjustedImage, 'adaptive', 'Sensitivity', 0.55 );
    thin_bw = bwmorph(bw, 'thin', Inf);
    se_close = strel('disk', 6);
    closed = imclose(thin_bw, se_close);
    filled = imfill(closed, 'holes');

    se_open = strel('disk', 3);
    opened = imopen(filled, se_open);

    clean = bwareaopen(opened, 500);

    roi_mask = imerode(clean, strel('disk', 12));

    fingerprint_roi = adjustedImage;
    fingerprint_roi(~roi_mask) = 255;

    if nargin == 2 && ~isempty(minutiae)
    minutiae = round(minutiae);
    [H, W] = size(roi_mask);
    in_bounds = minutiae(:,1) >= 1 & minutiae(:,1) <= W & ...
                minutiae(:,2) >= 1 & minutiae(:,2) <= H;

    valid_minutiae = minutiae(in_bounds, :);
    inds = sub2ind(size(roi_mask), valid_minutiae(:,2), valid_minutiae(:,1));
    mask_values = roi_mask(inds);
    filtered_minutiae = valid_minutiae(mask_values == 1, :);
else
    filtered_minutiae = [];
end
```

7.2 Bread first search algorithm

```
function distances = shortest_paths_to_nearest_one_diag(binaryMatrix)
    [rows, cols] = size(binaryMatrix);
    [onesY, onesX] = find(binaryMatrix);
    numOnes = length(onesX);
    distances = inf(numOnes, 1);


    directions = [ -1 -1; -1 0; -1 1;
                    0 -1;        0 1;
                    1 -1;  1 0;  1 1 ];

    for idx = 1:numOnes
        visited = false(rows, cols);
        queue = [onesY(idx), onesX(idx), 0];
        visited(onesY(idx), onesX(idx)) = true;

        found = false;

        while ~isempty(queue)
            [y, x, dist] = deal(queue(1,1), queue(1,2), queue(1,3));
            queue(1, :) = [];

            for d = 1:size(directions, 1)
                ny = y + directions(d, 1);
                nx = x + directions(d, 2);

                if ny >= 1 && ny <= rows && nx >= 1 && nx <= cols && ~visited(ny, nx)
                    visited(ny, nx) = true;
                    if binaryMatrix(ny, nx) == 1 && ~(ny == onesY(idx) && nx ==
onesX(idx))
                        distances(idx) = dist + 1;
                        found = true;
                        break;
                    else
                        queue(end+1, :) = [ny, nx, dist+1];
                    end
                end
            end

            if found
                break;
            end
        end
    end
end
```

7.3 Exporting to CSV

```
%% Exporting to Map of ones of ridges and bifurcations
map_size = size(grayImage);
map_of_ones_bif = create_map_from_coordinates(map_bif, map_size);
map_of_ones_ridges = create_map_from_coordinates(map_ridge, map_size);
ist_ridge = shortest_paths_to_nearest_one_diag(map_of_ones_ridges);
ist_bif   = shortest_paths_to_nearest_one_diag(map_of_ones_bif);
ist_combined =shortest_paths_to_nearest_one_diag(map_of_ones_ridges +
map_of_ones_bif);
if_coor = [x_bif, y_bif, 3 * ones(length(x_bif), 1)];
idge_coor = [x_rid, y_rid, ones(length(x_rid), 1)];
oordinates = [bif_coor; ridge_coor];
 Assuming dist_ridge, dist_bif, dist_combined are the input matrices
ax_len = max([length(dist_ridge), length(dist_bif),
length(dist_combined),length(coordinates)]);

 Pad each matrix with NaNs to the same length
ist_ridge_padded = pad_vector(dist_ridge, max_len);
ist_bif_padded = pad_vector(dist_bif, max_len);
ist_combined_padded = pad_vector(dist_combined, max_len);

 Now concatenate them into one matrix
istances = [dist_ridge_padded, dist_bif_padded, dist_combined_padded];

inal_data = [coordinates, distances];
olumn_names = {'X_Coordinate', 'Y_Coordinate', 'Type', 'Dist_Ridge', 'Dist_Bif',
'Dist_Combined'};

 Convert final_data to a table
inal_table = array2table(final_data, 'VariableNames', column_names);
 Export the result to CSV
writematrix(final_data, 'minutiae_data_ofir_7_2.csv');
```