



Fingerprint Recognition System

**Proposed to
Dr. Lama Afara**

**CMPS327
Image Processing
Salwa Hammoud
Mohammad Chatila
Khaled Abdul Samad**

Table of Contents

1. Abstract.....	2
1.1 Project Statement.....	2
1.2 Accomplished Objectives.....	2
2. Introduction to Fingerprints.....	3
2.1 Biometrics.....	3
2.2 Fingerprints and Their Features.....	3
2.3 Fingerprint Recognition.....	4
2.3.1 Crossing Number Based Minutiae Extraction.....	4
3. Preprocessing Fingerprints.....	5
3.1 Overview.....	5
3.2 Gray Scaling.....	5
3.3 Noise Reduction.....	6
3.3.1 Gaussian Blur.....	6
3.3.2 Median Filter.....	6
3.4 Adaptive Histogram Equalization.....	7
3.5 Binarization.....	8
3.6 Skeletonization.....	9
4. Processing Fingerprints.....	9
4.1 Minutiae Extraction.....	9
5. Postprocessing Fingerprints.....	10
5.1 Fingerprint Identification Through Minutiae Matching.....	10
6. Conclusion.....	12
7. References.....	13

1. Abstract

1.1. Project Statement

Our team decided to write a program that performs biometric authentication through fingerprint recognition. Our program could easily verify and identify fingerprints. When given an input fingerprint, our program will return whom does the input fingerprint belong to; this is based on matching score of minutiae features.

1.2. Accomplished Objectives

Our primary goal was to utilize various libraries in Python to create a fingerprint recognition system. We aimed to create this program by implementing a handful of image processing filters explained throughout the course and applied during lab sessions. After a lot of hard work on working on this project and lots of research, we have created a program that satisfies us truly.

2. Introduction to Fingerprints

2.1. Biometrics

We live in a world where technological advances are a part of our daily life. One of our greatest technological advances is biometrics. Biometrics are biological measurements — or physical characteristics — that can be used to identify individuals. The use of biometrics has allowed us to live in a more secure world. Examples of biometric security include voice recognition, fingerprint recognition, facial recognition, and iris recognition. Fingerprint recognition is the most well-known form of biometric security, where it is used in mobile devices, biometric passports, legal paperwork, and law enforcement. Therefore, we decided to create a program that verifies and identifies a person using their fingerprints.

2.2. Fingerprints and Their Features

Fingerprints are one form of biometrics; they are inexpensive to collect and analyze and they do not change as we age. Moreover, studies have shown that no two fingerprints are identical. For these reasons, fingerprints are a reliable source of biometric authentication. Fingerprints are made numerous ridges called friction ridges. All ridges form different patterns that create our unique fingerprint.

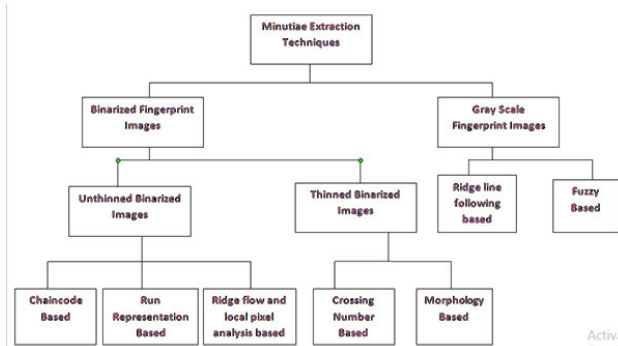


Three basic patterns often found in fingerprints are loops, whorls, and arches. Loops begin on one side of the finger, curve around or upward, and exit the other side, whorls form a circular or spiral pattern, and arches slope upward and then downward. Fingerprints are also identified through minutiae points: ridge ending, bifurcation, delta, core.



2.3.Fingerprint Recognition

Known as the most primary and accurate identification method, fingerprint recognition can be divided into two parts: identification and verification. Similar fingerprint recognition system could be used for in both cases. However, the major difference is the aim of the program. If the program asks, “who does this fingerprint belong to?” then, we are applying fingerprint identification. But, if the program asks, “is this fingerprint x’s?” then, that is fingerprint verification. There are numerous methods to achieve fingerprint recognition; these include minutiae-based technique, pattern matching technique, and image-based technique. All techniques are achieved through three steps: preprocessing, feature extraction, and feature matching. For our project, we decided to adopt minutiae-based technique. Minutiae-based extraction can be obtained in different ways depending on the preprocessing step. This is shown in the figure to the right.



2.3.1 Crossing Number Based Minutiae Extraction

Our preprocessing step includes binarization and skeletonization of the original fingerprint. Hence, we decided to match fingerprints based on crossing number. The most commonly employed method of minutiae extraction is the Crossing Number (CN) concept. This technique is often favored over other techniques for its computational efficiency and simplicity. The minutiae are extracted by scanning the local neighborhood of each ridge pixel in the image using a 3X3 kernel. The value of the CN is computed using the formula below, where $P_9 = P_1$. The value of the crossing number decides the type of minutia point as shown in the table.

P_4	P_3	P_2
P_5	P	P_1
P_6	P_7	P_8

$$CN = \frac{1}{2} \sum_{i=1}^8 |P_i - P_{i+1}|$$

CN	Property
0	Isolated Point
1	Ridge Ending Point
2	Continuing Ridge Point
3	Bifurcation Point
4	Crossing Point

The features needed for fingerprint recognition are ridge endings and bifurcations only, while the rest are considered non-minutiae points. The count and location of these two features are enough to successfully identify or verify fingerprints.

3. Preprocessing Fingerprints

3.1. Overview

To extract minutiae features in a proper manner, we need to make sure the fingerprint is of great quality. The image should be clean and free of noise. Hence, we used filters that reduce noise and enhance the image. To the enhance the image furthermore, adaptive histogram equalization, binarization, and skeletonization were applied to the fingerprints. This way we can have a perfect preprocessed image that results in an accurate verification and identification of the fingerprint.

3.2. Gray Scaling

The first step we took in preprocessing our image was to convert it from RGB to gray scale. This step is mandatory for us to be able to move on in preprocessing. To do so, we used OpenCV's built in color conversion method. Although scanned fingerprints are usually gray scale, we do this step to ensure we will not face any error with the rest of the code.

```
grayImage = cv2.cvtColor(originalImage, cv2.COLOR_BGR2GRAY)
```



3.3. Noise Reduction

Biometric identification and authentication systems need a high-quality image to achieve a reliable and accurate result. Thus, we want to make sure that the image is a noise-free fingerprint image similar to the one on the right. To do so, we need to apply two of the most prominent image processing filters: Gaussian and median blur.



3.3.1. Gaussian Blur

In some binarization-based approaches, the binarization and thinning process is preceded by a smoothing operation. We decided to use the gaussian blur. Gaussian filter is mostly used when an image is degraded by noise; it works by blurring the image by the Gaussian function. To smooth our fingerprints as part of the preprocessing step, we used OpenCV's built in gaussian blur method with kernel size 7x7 and variance equal to 0. The function and the result are shown below.

```
def gauss(img):
    smoothedImage = cv2.GaussianBlur(img, (7,7), 0)
    return smoothedImage
```

Smoothed Fingerprint:



3.3.2. Median Blur

The median blur is used to reduce noise as well, specifically pepper noise. After binarizing the image, we might find pepper noise around the fingerprint. This can result in an inaccurate result. Therefore, a median filter is the perfect filter for cleaning the image at this point. To apply it, we used OpenCV's built in median filter with kernel size 3x3.

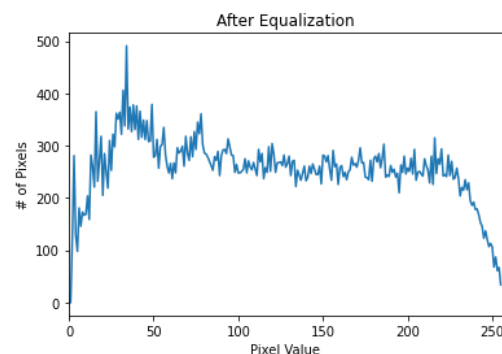
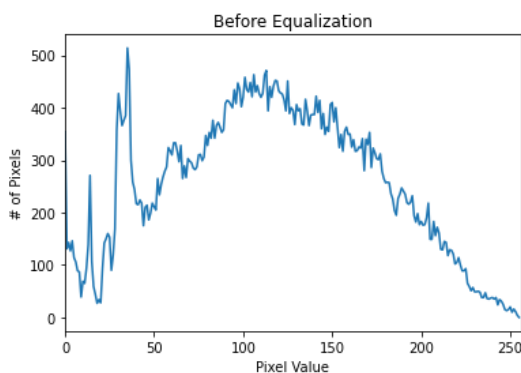
```
binaryImage = cv2.medianBlur(binaryImage,3)
```




3.4 Adaptive Histogram Equalization

Adaptive histogram equalization is an image processing technique used to improve contrast in images through stretching out intensity range. The difference between normal histogram equalization and adaptive histogram equalization is that the adaptive method computes several histograms from multiple sections of each image and uses these histograms to redistribute the lightness values of the image, whereas the normal histogram equalization uses a single histogram for an entire image. We applied this technique on our fingerprints as part of the preprocessing step to improve the image's contrast and bring out more detail in it. To do so, we used OpenCV's built in method, and we gave it two parameters: clip limit = 2 (threshold for contrast limiting) and tile size = 8x8 (size of sections in image).

```
def adaptiveHistogramEqualization(img):
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
    equalizedImage = clahe.apply(img)
    return equalizedImage
```



3.5 Binarization

One of the most important steps in preprocessing the image is binarizing it. Fingerprint binarization is the process of converting an 8-bit gray-scale fingerprint image into a 1-bit ridge image. The easiest way to binarize a fingerprint is to choose a threshold value and classify all pixels with values above this threshold as white, and all other pixels as black. However, the challenge is how to choose the threshold. We decided to use adaptive thresholding where the threshold is the mean of neighborhood. To do so, we used OpenCV's built in adaptive thresholding function. We gave the function 6 parameters: the image, the output threshold value (255), the method of adaptive thresholding (mean), the method of thresholding (binary), the pixel neighborhood size (3x3), and a constant (c=5) to fine tune our threshold value. Then, we applied the median filter mentioned before to remove any pepper noise that might affect feature extraction.

```
def binarization(img):  
    binaryImage =  
        cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,3,5)  
    binaryImage = cv2.medianBlur(binaryImage,3)  
    return binaryImage
```

before binarization



after binarization



3.5 Skeletonization

The second important step is thinning, also known as skeletonization. This is also the last preprocessing step. Without this step, feature extraction will not be accurate or even correct. To thin our fingerprint images, we used the technique of Zhang-Suen. This is a thinning algorithm that sets certain pixels in a neighborhood of pixels to 255 if it satisfies a certain set of conditions. We gave the function an inverted image where the thinning takes place on it. Then, we set the pixels back to how they were before. After a lot of trial and error, we found that this method gave us the best thinning result show to the right.




```
def thinning(img):
    thinnedImage = cv2.ximgproc.thinning(invert(img))
    data = np.array(thinnedImage)
    row, col = data.shape
    for i in range(0, row):
        for j in range(0, col):
            if(data[i, j] == 255):
                data[i, j] = 0
            else:
                data[i, j] = 255
    thinnedImage = Image.fromarray(data)
    return thinnedImage
```



4. Processing Fingerprints

4.1 Minutiae Extraction

Finally, after preprocessing the image, it's time to extract minutiae points from the fingerprints. As previously mentioned, the method of our choice was calculating crossing number where CN=1 is a ridge termination and CN=3 is a bifurcation. We created a dictionary with 2 empty lists to store the location of each minutia point (features); we also created an empty array that stores all features regardless of their type (arrFeatures). We looped on the image pixels and computed the crossing number using the kernel

	Crossing Number=2. Normal ridge pixel.
	Crossing Number=1. Termination point.
	Crossing Number=3. Bifurcation point.

shown previously. The kernel computes crossing number in an anti-clockwise manner and in a certain order. To do so, we used a 2D array with certain values to calculate the pixel value starting P4 and ending at P3 using the crossing number equation. We also created a method that draw ellipses around minutiae points to easily identify them visually as seen in the image.



```
def extraction(img):
    image2 = img_as_float(img)
    row, col = image2.shape
    features = {"bifurcation": [], "ending": []}
    arrFeatures = []
    cn1 = 0
    cn3 = 0
    for i in range(1, row - 1):
        for j in range(1, col - 1):
            minutiaeType = computation(image2, i, j)
            if minutiaeType == "ending":
                cn1 += 1
                features[minutiaeType].append((i, j))
                arrFeatures.append((i, j))
                drawing(i, j, img)
            elif minutiaeType == "bifurcation":
                cn3 += 1
                features[minutiaeType].append((i, j))
                arrFeatures.append((i, j))
                drawing(i, j, img)
    return img, cn1, cn3, features, arrFeatures

def computation(img, i, j):
    cells = [(-1, -1), (-1, 0), (-1, 1), (0, 1),
             (1, 1), (1, 0), (1, -1), (0, -1), (-1, -1)]
    if int(img[i][j]) == 0:
        p = [int(img[i + k][j + l]) for k, l in cells]
        cn = 0
        for k in range(0, 8):
            cn += abs(int(p[k]) - int(p[k + 1]))
        cn *= 0.5
        if cn == 1.0:
            return "ending"
        elif cn == 3.0:
            return "bifurcation"
        return "none"
    def drawing(y, x, img):
        draw = ImageDraw.Draw(img)
        ellipseSize = 2
        draw.ellipse([(x - ellipseSize, y - ellipseSize), (x + ellipseSize, y + ellipseSize)])
```

5. Postprocessing Fingerprints

5.1 Fingerprint Identification Through Minutiae Matching

The last step is to match fingerprints. We did this using Euclidian distance and score matching. First, we computed the spatial distance between minutiae points, then if the distance is less than a predetermined tolerance value (r_0), we consider it a matching minutia point. According to research, the best value for the tolerance is 10; therefore, we used $r_0=10$.

$$sd(m'_j, m_i) = \sqrt{(x'_j - x_i)^2 + (y'_j - y_i)^2} \leq r_0$$

Finally, after calculating the distance between almost all minutiae points, we will get the score using the formula below where n and m are the number of minutiae points in fingerprint1 and fingerprint2 respectively. The higher the score, the better the match between the fingerprints is. If we gave this method the same fingerprint twice, it would return a score of 100.

```
def calculateDistance(i1, j1, i2, j2):
    return np.sqrt(pow((i1-i2), 2) + pow((j1-j2), 2))

def matchingScore(featuresA, featuresB, tolerance=10):
    countMatching= 0
    if(len(featuresA) < len(featuresB)):
        size = len(featuresA)
    else:
        row, col = data.shape
        for i in range(0, row):
            for j in range(0, col):
                if(data[i, j] == 255):
                    data[i, j] = 0
                else:
                    data[i, j] = 255
    thinnedImage = Image.fromarray(data)
    size = len(featuresB)
    avgMinutiae = (len(featuresA) + len(featuresB))/2
    for k in range(0, size):
        spatialDistance = calculateDistance(featuresA[k][0], featuresA[k][1], featuresB[k][0], featuresB[k][1])
        if(spatialDistance <= tolerance):
            countMatching +=1
    matchingPerc = countMatching/avgMinutiae*100
    return matchingPerc
```

$$score = \frac{k}{(n+m)/2}$$

At the end, our program will identify which fingerprint does the input fingerprint match. We've given each fingerprint a name for its owner. The output displays the fingerprint before preprocessing, after preprocessing, and after feature extraction. It also says whom does the fingerprint belong to as shown:

```
The input fingerprint belongs to: Salwa
File name: 1_4.TIF
```

6. Conclusion

Fingerprint recognition is a reliable way of identifying and verifying people. It is used in many applications like biometric measurements, solving crime investigation and in security systems. There are multiple methods to achieve a fingerprint with extracted features and multiple ways of recognizing fingerprints. In our project, we used a technique that was simple yet effective. To extract features correctly, we had to preprocess our fingerprints through various filters. Preprocessing the original fingerprint involves image binarization, ridge thinning, and noise removal. And fingerprint recognition using minutia score matching method is used for matching the minutia points. All the methods and algorithms described in this report were implemented using Python.

7. References

1. https://www.researchgate.net/publication/329191381_Fingerprint_Verification_with_Crossing_Number_Extraction_and_Orientation-Based_Matching
2. <https://bura.brunel.ac.uk/bitstream/2438/7473/1/FulltextThesis.pdf>
3. https://vtechworks.lib.vt.edu/bitstream/handle/10919/34010/Hoyle_KE_T_2011.pdf?sequence=1&isAllowed=y
4. <http://www.researchinventy.com/papers/v3i6/I036056062.pdf>
5. http://ethesis.nitrkl.ac.in/2610/1/15th_may.pdf
6. <https://arxiv.org/pdf/1201.1422.pdf>
7. <https://github.com/przemekpastuszka/biometrics>
8. <https://github.com/Antoninj/biometrics-project>