HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT REPORT

FINGERPRINT AUTHENTICATION

Course: Biometric Authentication System

Lecturers:
Dr. Tran Nguyen Ngoc
Dr. Ngo Thanh Trung

Authors	Student ID
Phan Thu Ha	20225571
Nguyen Phuong Linh	20225547
Tran Thi Tuong Van	20225590
Pham Nguyen Hai Nhi	20225549

December, 2024

Contents

1 Introduction . . . 3
2 Dataset . . . 4
3 Preprocessing . . . 5
3.1 Grayscale transformation . . . 5
3.2 Noise reduction . . . 5
3.3 Segmentation . . . 6
3.4 Normalization . . . 6
3.5 Enhance Contrast . . . 7
3.6 Binarization . . . 7
3.7 Thin Images . . . 7
3.8 Library . . . 8
4 Features extraction . . . 9
4.1 Skeletonization and Harris Corner . . . 9
4.2 Local Binary Pattern (LBP) . . . 9
4.3 HOG Feature . . . 9
4.4 Gabor Filter . . . 10
5 Matching algorithm . . . 11
5.1 Cosine similarity matching . . . 11
5.1.1 Preprocessing . . . 11
5.1.2 Cosine Similarity Computation . . . 11
5.1.3 Decision making . . . 11
5.2 Support Vector Machine . . . 12
5.2.1 Overview . . . 12
5.2.2 Support Vector Machine . . . 12
5.2.3 Hyperparameters . . . 13
5.2.4 Results . . . 13
5.3 Siamese Neural Networks . . . 14
5.3.1 Overview . . . 14
5.3.2 Key features . . . 14
5.3.3 Siamese Neural Networks . . . 14
5.3.4 Result . . . 15
5.4 Gradient Boosting . . . 16
5.4.1 Overview . . . 16
5.4.2 Hyperparameter Tuning . . . 16
5.4.3 Result . . . 17
6 Conclusion . . . 18
6.1 Summary . . . 18
6.2 Insights . . . 18
6.3 Future Improvements . . . 19

1 Introduction

Fingerprint recognition has emerged as a widely utilized biometric technology, offering a secure and convenient means of personal identification. Driven by the increasing demand for secure authentication systems in various domains, such as access control, financial transactions, and mobile devices, fingerprint recognition has gained significant prominence. Its unique nature, derived from the intricate ridge and valley patterns on human fingertips, makes it a highly reliable method for authentication.

The primary aim of this project is to develop a robust and accurate fingerprint recognition web capable of determining whether two input fingerprint images originate from the same finger. This system will leverage appropriate image processing and pattern recognition techniques to analyze and compare the unique minutiae patterns present in fingerprint images.

2 Dataset

We got "Sokoto Coventry Fingerprint Dataset (SOCOFing)" by author Ruizgara from Kaggle at the URL: `"https://www.kaggle.com/datasets/ruizgara/socofing"`.

"Sokoto Coventry Fingerprint Dataset (SOCOFing)" is a biometric fingerprint database designed for academic research purposes. SOCOFing is made up of 6,000 fingerprint images from 600 African subjects and contains unique attributes such as labels for gender, hand and finger name. It also contains synthetically altered versions with three different levels of alteration for obliteration, central rotation, and z-cut.

To train machine learning models, we generated a new dataset which combines Real and Altered images from SOCOFing. This dataset consists of labeled pairs of images, indicating whether they belong to the same finger or not.

- **Positive Pair (Label = 1):** Two images are considered a positive pair if they belong to the same finger from the same subject. To create positive pairs, the code iterates through each real_image. For each real_image, it extracts the subject ID from the filename (e.g., 1__M_Left_index_finger). Then, it finds all alter_images that belong to the same subject and finger type by checking if the subject ID extracted from the alter_image filename matches the real_image's subject ID. A positive pair is formed by randomly selecting one of these alter_images and pairing it with the real_image.

- **Negative Pair (Label = 0):** A negative pair consists of two images from different fingers. To create negative pairs, the code randomly selects another real_image from the dataset, ensuring that it belongs to a different subject or finger type than the current real_image.

Our new dataset includes original images from the SOCOFing dataset, just using new labels assigned based on their pairing with other images.

3 Preprocessing

The fingerprint image is first preprocessed to remove noise and any irrelevant information. Image preprocessing normally consists of the following steps.

3.1 Grayscale transformation

Fingerprint images were converted from color to grayscale. This conversion simplifies the image representation and reduces computational complexity, as it focuses solely on the intensity values of pixels, which contain important fingerprint ridge information.

3.2 Noise reduction

Noise reduction is a crucial preprocessing step in fingerprint recognition to improve the accuracy and reliability of subsequent analysis. Two common noise reduction techniques employed in this project are medianBlur and Gaussian Blur.

- **MedianBlur:** This technique replaces each pixel with the median value of its neighboring pixels within a specified window size. MedianBlur is effective at removing salt-and-pepper noise, which often appears as isolated white or black pixels.
- **Gaussian Blur:** This technique convolves the image with a Gaussian kernel, which is a weighted average where the weights are determined by a Gaussian distribution. Gaussian Blur effectively smooths out noise while preserving edges to some extent.

The mathematical representation of Gaussian Blur involves convolving the image I(x, y) with a Gaussian kernel G(x, y):

$$I_{blur}(x, y) = I(x, y) * G(x, y) = \sum_{u,v} I(u, v)G(x - u, y - v)$$

where:

- $I(x, y)$ is the original image intensity at pixel (x, y)
- $I_{blur}(x, y)$ is the blurred image intensity at pixel (x, y)
- $G(x, y)$ is the Gaussian kernel

Both MedianBlur and Gaussian Blur can effectively reduce noise in fingerprint images, improving the clarity and visibility of minutiae points such as ridge endings and bifurcations. However, excessive blurring can also lead to the loss of important fine details, which can negatively impact the performance of subsequent fingerprint matching algorithms.

On this dataset images, we decided to use MedianBlur for our later process.

3.3 Segmentation

Segmentation is crucial for isolating the fingerprint region from the background. Simple thresholding techniques, such as those based on a fixed threshold value, often fail to accurately segment fingerprints due to their complex grayscale variations and the presence of streaks. To address these challenges, we employed an adaptive thresholding method.

Specifically, we utilized the $cv2.adaptiveThreshold()$ function from the OpenCV library. This function calculates a threshold value for each pixel based on the local pixel neighborhood.

We used the $cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C$ method, which computes a weighted sum of the neighborhood pixels using a Gaussian window.

The $cv2.THRESH\_BINARY$ thresholding type was applied, resulting in a binary image where pixels with intensities above the threshold are set to white, and those below are set to black. The blockSize parameter was set to 11, defining the size of the local neighborhood, and the C parameter was set to 2, a constant subtracted from the mean or weighted mean.

This adaptive thresholding approach effectively handles variations in illumination and contrast within the fingerprint image, resulting in accurate segmentation of the fingerprint region.

3.4 Normalization

Normalization is performed to remove the effect of sensor noise and gray-level background which are the consequence of difference in finger pressure. Normalization is used to standardize the intensity values in an image by adjusting the range of gray-level values so that it lies within a desired range of values (Maltoni et al., 2003). It is a pixel-wise operation which does not changes the range of pixel intensity values (Adler, 1998).

Let $I(i,j)$ denote the gray-level value at pixel (i,j). The normalised value $N(i,j)$ for pixel (i,j) is defined (Maltoni et al., 2003):

$$N(i,j) = \begin{cases} M_0 + \sqrt{\frac{(V_0(I(i,j))-M_I)^2}{V_i}} & \text{if } I(i,j) > M \\ M_0 - \sqrt{\frac{(V_0(I(i,j))-M_I)^2}{V_i}} & \text{otherwise} \end{cases} \tag{1}$$

Here, M_0 and V_0 are the desired mean and variance respectively and $M_0 = V_0 = 100$ (Maltoni et al., 2003). The mean $M(I)$ and variance $V(I)$ of a gray-level fingerprint image with the dimension of $L \times N$ pixels, are defined using equation (2) and (3) respectively (Maltoni et al., 2003).

$$M(I) = \frac{1}{LN} \sum_{i=0}^{L-1} \sum_{j=0}^{N-1} I(i,j) \tag{2}$$

$$V(I) = \frac{1}{LN} \sum_{i=0}^{L-1} \sum_{j=0}^{N-1} (I(i,j) - M(I))^2 \tag{3}$$

Where $I(i,j)$ represents the intensity of the pixel at row i and column j. The basic objective of normalization operation is to reduce the variations of gray-level values along the ridges and valleys (Jain et al., 1997).

3.5 Enhance Contrast

Contrast enhancement is important for improving the visibility of fingerprint features. Histogram equalization is a widely used technique to enhance image contrast by redistributing the intensity values in the image.

Histogram equalization aims to create an image with a uniform histogram, effectively stretching the intensity range of the image. The cumulative distribution function (CDF) plays a key role in this process. Let L be the number of gray levels in the image (typically 256 for 8-bit images). The CDF of an image, H(k), is defined as:

$$H(k) = \sum_{i=0}^{k} p(i)$$

where p(i) is the probability of occurrence of gray level i in the image.

Histogram equalization maps each pixel intensity i to a new intensity s using the following transformation:

$$s = (L-1) \cdot H(i)$$

This transformation effectively stretches the intensity values in regions with low contrast while compressing them in regions with high contrast, resulting in an image with a more uniform distribution of gray levels.

In this project, we employed the $cv2.equalizeHist()$ function from the OpenCV library to perform histogram equalization on the segmented fingerprint images. This function efficiently computes the histogram and applies the equalization transformation, enhancing the visibility of fingerprint ridges and valleys.

3.6 Binarization

Binarization is an essential step in fingerprint image preprocessing. It transforms the grayscale image into a binary image, where each pixel is assigned a value of either 0 (black) or 255 (white). This simplifies the image and facilitates subsequent feature extraction.

We employed the $cv2.threshold()$ function from the OpenCV library for binarization. This function applies a threshold value to each pixel in the image. Pixels with intensity values greater than the threshold are set to white, while those below the threshold are set to black. We used the $cv2.THRESH\_BINARY$ thresholding type, which results in a binary image with sharp transitions between black and white regions.

3.7 Thin Images

Thinning, also known as skeletonization, is the next step which reduces the thickness of the ridge lines to a single pixel width while preserving the overall topological structure of the fingerprint. This simplifies the image, making it easier to identify and extract minutiae points.

3.8 Library

The core of our fingerprint recognition system relies heavily on the OpenCV (Open Source Computer Vision Library) library. The OpenCV library's extensive functionality and ease of use significantly facilitated the development of our fingerprint recognition system. Its well-documented API and availability across various platforms made it an ideal choice for this project. We used this library for image reading, image processing operations and feature extraction - the next step of our project.

4 Features extraction

4.1 Skeletonization and Harris Corner

Minutiae, such as ridge endings and bifurcations, are unique characteristics of fingerprints used for identification. To extract these features, we employed a two-step approach:

- **Skeletonization:** reduces the fingerprint image to a single-pixel-wide representation while preserving its topological structure. We utilized a thinning algorithm to achieve this, effectively removing redundant pixels from the ridge lines. This step simplifies the image, making it easier to identify and locate minutiae points.
- **Harris Corner Detection:** is employed to identify points of interest in the skeletonized image. This method detects corners based on the local autocovariance matrix, which measures the change in image intensity around a pixel in different directions. Corners, which correspond to abrupt changes in image intensity, are likely to indicate minutiae points.

By combining skeletonization and Harris Corner detection, we can effectively locate and extract minutiae points from the fingerprint images. These extracted minutiae features are then used for subsequent matching and comparison.

4.2 Local Binary Pattern (LBP)

Local Binary Pattern, also known as LBP, is a simple and grayscale invariant texture descriptor measure for classification. In LBP, a binary code is generated at each pixel by thresholding it's neighbourhood pixels to either 0 or 1 based on the value of the centre pixel. The rule for finding LBP of an image is as follows:

1. Set a pixel value as center pixel.
2. Collect its neighbourhood pixels.
3. Threshold it's neighbourhood pixel value to 1 if its value is greater than or equal to centre pixel value otherwise threshold it to 0.
4. After thresholding, collect all threshold values from neighbourhood either clockwise or anti-clockwise. The collection will give you an 8-digit binary code. Convert the binary code into decimal.
5. Replace the center pixel value with resulted decimal and do the same process for all pixel values present in image.

4.3 HOG Feature

Histogram of Oriented Gradients (HOG) is a powerful feature descriptor commonly used in object detection and image recognition tasks, including fingerprint recognition.

The HOG feature descriptor captures the distribution of edge orientations within local cells of the image. The process involves the following steps:

- **Gradient Calculation:** Compute the gradient magnitude and orientation of the image using Sobel or other edge detection operators.
- **Cell Division:** Divide the image into small, overlapping cells.
- **Orientation Binning:** Within each cell, create a histogram of gradient orientations, typically using 8 or 9 bins.
- **Block Normalization:** Group cells into larger blocks. Normalize the histogram of each cell within a block to improve invariance to illumination and contrast changes. Common normalization methods include L2-norm and contrast-sensitive normalization.
- **Feature Vector Construction:** Concatenate the normalized histograms of all cells within the image to form a high-dimensional feature vector. HOG features effectively capture the shape and structure of objects within the image, making them robust to geometric and photometric transformations.

In the context of our project, HOG features can effectively represent the unique ridge and valley structures, providing discriminative information for accurate matching.

4.4 Gabor Filter

The configurations of parallel ridges and valleys with well defined frequency and orientation in a fingerprint image provide useful information which helps in removing undesired noise. The sinusoidal-shaped waves of ridges and valleys vary slowly in a local constant orientation. Therefore, a band-pass filter that is tuned to the corresponding frequency and orientation can efficiently remove the undesired noise and preserve the true ridge and valley structures.

Gabor filters have both frequency-selective and orientation-selective properties and have optimal joint resolution in both spatial and frequency domains (Daugman, 1985; Jain and Farrokhnia, 1991). Therefore, it is appropriate to use Gabor filters as band-pass filters to remove the noise and preserve true ridge/valley structures. The Gabor filter is applied to the fingerprint image by spatially convolving the image with the filter (Kovascs-Vajna et al., 1997).

5 Matching algorithm

5.1 Cosine similarity matching

5.1.1 Preprocessing

Using our shared pipeline:

- Convert the input image to grayscale.
- Resize the image to a fixed dimension (256x256 pixels) for uniformity.
- Apply median blur to reduce noise.
- Enhance contrast using histogram equalization.
- Use adaptive thresholding to binarize the image.

5.1.2 Cosine Similarity Computation

To determine the similarity between two fingerprint images, cosine similarity is computed using the following formula:

$$\text{Cosine similarity(A,B)} = \frac{A \cdot B}{\|A\|\|B\|}$$

where A and B are the flattened vectors of the fingerprint images, $\cdot$ denotes the dot product, and $\|A\|$ and $\|B\|$ are the magnitudes of vectors A and B respectively.

5.1.3 Decision making

The resulting similarity score ranges from -1 to 1, where a score close to 1 indicates a high similarity between the two fingerprint images, and a score close to -1 indicates low similarity.

We set threshold = 0.75 to determine if the fingerprints match based on the similarity score. With two input images, we preprocessed them and print out the guess after compare their cosine similarity.

This is a fast way to authenticate fingerprint, as long as the users input their fingers in the same position as when they registered that finger. Because this method will not perform well with two variants of a same object (the similarity will decrease). However, it is still more reliable to match fingerprints in compare to other distance such as Euclidean or Manhattan distance.

5.2 Support Vector Machine

5.2.1 Overview

Support Vector Machine (SVM) is a powerful machine learning algorithm commonly used for classification tasks. The main objective of the SVM algorithm is to find the optimal hyperplane in an N-dimensional space that best separates the data points into different classes within the feature space. In the context of fingerprint matching, SVM is used to classify fingerprint features by determining the optimal hyperplane that separates two classes: "match" and "non-match." The inputs to the SVM model consist of extracted features from fingerprint images, and the algorithm learns to distinguish between fingerprints that belong to the same person (match) and those that do not (non-match).

5.2.2 Support Vector Machine

- **Linear SVM** finds a hyperplane of the form:

$$\langle w, x \rangle + b = 0$$

known as the *decision boundary/surface*, which has max margin or minimal expected errors among all possible hyperplanes.

Margin, the distance between the two marginal hyperplanes, $\frac{2}{\|w\|}$, is used to guide learning.

The problem can be formulated as:
Find w and b that minimize $\frac{\langle w,w \rangle}{2}$ and satisfy the below conditions for any training instance x_i:

$$y_i \cdot (\langle w, x_i \rangle + b) \geq 1 \ \forall i = 1 \dots r$$

- **Non-linear SVM** includes 2 main steps:
  - **Step 1:** transform the input into another space, which often has higher dimensions, by using a non-linear mapping

$$\begin{aligned} \phi : X &\longrightarrow F \\ x &\mapsto \phi(x) \end{aligned}$$

so that the projection of data is linearly separable

$$\begin{aligned} \{(x_1, y_1), (x_2, y_2), \dots, (x_r, y_r)\} \\ \{(\phi(x_1), y_1), (\phi(x_2), y_2), \dots, (\phi(x_r), y_r)\} \end{aligned}$$

  - **Step 2:** use linear SVM in the new space; therefore, the new formulation can be
    Find w and b that minimize $L_p = \frac{\langle w,w \rangle}{2} + C \sum_{i=1}^{r} \varepsilon_i$ and satisfy the below conditions for any training instance xi:

$$\begin{cases} y_i \cdot (\langle w, \phi(x_i) \rangle + b) \geq 1 - \varepsilon_i, \forall i = 1 \dots r \\ \varepsilon_i \geq 1, \forall i = 1 \dots r \end{cases}$$

The corresponding *dual problem* is that:
Maximize: $L_D = \sum_{i=1}^{r} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{r} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(z) \rangle + b^*$

Such that: $\begin{cases} \sum_{i=1}^{r} \alpha_i y_y = 0 \\ 0 \leq \alpha_i \leq C, \forall i = 1 \dots r \end{cases}$

Objective function: $f(z) = \langle w^*, \phi(z) \rangle + b^* = \sum_{i \in SV} \alpha_i y_i \langle \phi(x_i), \phi(z) \rangle + b^*$

Kernel function: $K(x, z) = \langle \phi(x_i), \phi(z) \rangle$ has several choices to replace the above inner products. In our project, we choose the following kernel functions

* Polynomial ("poly")
$$K(x, z) = (\gamma x^T z + r)^d$$
where $r \in R$, $d \in N$, $\gamma \in R$
* Gaussian radial basis function ("rbf")
$$K(x, z) = exp(-\gamma \|\mathbf{x}, \mathbf{z}\|^2)$$

5.2.3 Hyperparameters

- **C (Regularization Parameter)**: controls the trade-off between the insensitive loss and sensitive loss. A larger value of 'C' means that the model will try to minimize the insensitive loss more, while a smaller value of C means that the model will be more lenient in allowing larger errors.
- **Kernel**: determines the type of transformation applied to the input data to map it into a higher-dimensional space where a linear separation is possible.
- **Gamma**: defines kernel coefficient

5.2.4 Results

We used train_size = 0.75 and test_size = 0.25 and SVC library with hyperparameters chosen for SVM model such as kernel:'linear' and C = 1.0.

- **Accuracy:** 77%
- **Precision:** 38%
- **Recall:** 50%
- **F1 score:** 43%

5.3 Siamese Neural Networks

5.3.1 Overview

A Siamese Neural Network (SNN) is a type of neural network architecture specifically designed to compare two inputs and determine their similarity. The network consists of two identical subnetworks that process the inputs independently but in parallel. The outputs of these subnetworks are then compared using a distance metric, allowing the network to learn whether the inputs are similar or dissimilar.

5.3.2 Key features

1. **Identical Sub-networks:** A defining characteristic is the use of identical subnetworks for processing each input. These subnetworks have the same architecture and parameters, ensuring that both inputs are transformed in the same way.

2. **Shared Weights:** The identical subnetworks in an SNN share the same weights. This weight sharing ensures that the network learns consistent features from both inputs, maintaining the integrity of the comparison process.

3. **Learning Similarity:** SNNs are designed to learn a similarity function that can distinguish between similar and dissimilar pairs. The network outputs a feature vector for each input, and the similarity between these vectors is calculated using a distance metric.

4. **Contrastive Loss:** is designed to minimize the distance between the outputs of similar pairs and maximize the distance between the outputs of dissimilar pairs. The contrastive loss function is defined as:

$$L = \frac{1}{2}((1-y)D^2 + y\max(0, m-D)^2)$$

   where:

   - y is the label indicating whether the inputs are similar (0) or dissimilar (1).
   - D is the distance between the feature vectors of the two inputs.
   - m is a margin parameter that defines the minimum distance for dissimilar pairs.

5.3.3 Siamese Neural Networks

1. **Input Pairs**

   - **input** Two inputs for preprocessed fingerprint images. Each image will be processed by a separate branch of the model.
   - **input_minutiate** These are the inputs for the minutiae features, which are flattened to prepare them for concatenation with the features from the images.

2. **Shared CNN Branches** Constructing two identical CNN branches

   - **Conv2D:** Convolutional layers are used to extract features from the images.
   - **MaxPooling2D:** Max pooling layers help reduce the input size, thereby decreasing the number of parameters and computations.
   - **Flatten:** The output from the convolutional and pooling layers is converted into a 1D vector to prepare for concatenation.

3. **Concatenating Features:** After extracting features from both CNN branches, concatenating these features with their corresponding minutiae inputs.

4. **Calculating Distance:** using a Lambda layer to calculate the distance between the feature vectors of the two branches.
   Calculate the absolute difference between two feature vectors and create a new vector that represents the distance between them.

$$D = |A - B| = [|a_1 - b_1|, |a_2 - b_2|, \ldots, |a_n - b_n|]$$

5. **Output Layer:** adding a Dense layer with a sigmoid activation function to predict the probability that the two fingers (or samples) are identical.
   The Dense layer can be calculated using the following formula:

$$y = f(W \cdot x + b)$$

   where:

   - y: Output of the neuron.
   - f: Activation function (e.g. ReLU, sigmoid).
   - W: Weight matrix of the neuron.
   - x: Input from the previous layer.
   - b: Bias for the neuron.

5.3.4 Result

We used train_size=0.75 and test_size=0.25 with epochs=10, batch_size=32.

- **Accuracy:** 83%
- **Precision:** 85%
- **Recall:** 83%
- **F1 score:** 83%

5.4 Gradient Boosting

5.4.1 Overview

In general, boosting is a powerful ensemble technique in machine learning. Boosting combines the predictions of multiple weak learners to create a single, more accurate strong learner. It starts by fitting an initial model to the data, here we use decision tree. Then a second model is built that focuses on accurately predicting the cases where the first model performs poorly. The combination of these two models is expected to be better than either model alone. Then we repeat this process of boosting many times. Each successive model attempts to improve the combined result of all previous models.

Gradient Boosting model based on Gradient descent, an optimization algorithm which aims to minimize a function by iteratively moving towards the minimum. It operates by calculating the gradient of the function at the current point, which indicates the direction of steepest ascent. Then, it updates the current point by taking a step in the opposite direction of the gradient, scaled by a factor known as the learning rate. This process is repeated until a stopping criterion is met, such as reaching a predefined number of iterations or when the change in the function value becomes sufficiently small. So, in each iteration:

$$W_n = W_{n-1} - \eta \frac{\partial}{\partial w} L(W_{n-1})$$

Where η is the learning rate. So:

$$c_n w_n \approx -\eta \frac{\partial}{\partial w} L(W_{n-1})$$

Where w_n is the next model to add, we need it to fit $-\eta \frac{\partial}{\partial w} L(W_{n-1})$.

5.4.2 Hyperparameter Tuning

- **Learning rate**

  It controls the contribution of each weak learner by adjusting the shrinkage factor. We choose learning rate = 0.1 to train faster, since the algorithm itself is strong.

- **Metric**

  The metric parameter specifies the evaluation metric to be used during training. In this case, we use 'binary_logloss', which is suitable for binary classification problems. Binary log loss measures the performance of a classification model where the prediction output is a probability value between 0 and 1. The loss increases as the predicted probability diverges from the actual label.

- **Boosting Type**

  The boosting_type parameter determines the boosting algorithm to be used. We use 'gbdt', which stands for Gradient Boosting Decision Tree.

- **feature_fraction**

  By default, LightGBM considers all features in a Dataset during the training process. But here, we set feature_fraction to 0.9, tells LightGBM to randomly select 90% of features at the beginning of constructing each tree. This reduces the total number of splits that have to be evaluated to add each tree node.

- **Num_leaves** The num_leaves parameter sets the maximum number of nodes per tree. This is the main parameter to control the complexity of the tree model. In our model, increasing num_leaves from 30 to 100 just affect the validation phase a bit (slightly more accurate), so I keep num_leaves=30.

5.4.3 Result

We used train_size=0.75 and test_size=0.25 for Light GBM model.

Training after preprocessing images		**Training without preprocessing images**	
Accuracy	0.906	**Accuracy**	0.94
Precision	0.915	**Precision**	1.0
Recall	0.904	**Recall**	0.884
F1 score	0.91	**F1 score**	0.938

After saving trained model to pickle file, we used those models to predict whether the two input images are matched or not.

6 Conclusion

6.1 Summary

This project aimed to develop a robust and accurate fingerprint recognition system capable of determining whether two input fingerprint images originate from the same finger. We explored various image processing techniques, including noise reduction, segmentation, normalization, contrast enhancement; and feature extraction methods such as HOG, skeletonization, Gabor and Harris Corner detection. These techniques were employed to preprocess the fingerprint images and extract meaningful features for subsequent matching.

We investigated four algorithms for fingerprint matching, including:

- **Cosine Similarity:** This simple approach measures the similarity between feature vectors based on the cosine of the angle between them. It is a fast but effective approach.
- **Support Vector Machine (SVM):** SVM is a powerful classification algorithm, but it did not yield satisfactory results in our experiments.
- **Siamese Network:** This deep learning architecture demonstrated promising performance, achieving an accuracy of over 80%.
- **LightGBM:** This gradient boosting framework exhibited the highest accuracy, surpassing 90%, indicating its effectiveness in capturing complex relationships between fingerprint features.

6.2 Insights

Effective preprocessing and Feature selection plays a vital role in the success of any fingerprint recognition system. The choice of appropriate methods significantly influenced the accuracy of the matching process.

Deep learning models, such as Siamese networks, offer significant potential for fingerprint recognition. However, they require careful training and tuning to achieve optimal performance.

Ensemble methods, such as LightGBM, can achieve high accuracy in fingerprint recognition by combining the strengths of multiple weak learners.

Handling variations in image quality remains a significant challenge. Factors such as rotation, scaling, partial fingerprints, and low image resolution can significantly degrade the performance of fingerprint recognition systems.

6.3 Future Improvements

We are currently having many direction for developing our project:

- Apply to recognision problem, telling whether a fingerprint is already in the database or not.
- Explore more advanced deep learning architectures: Investigate the use of convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention mechanisms for more robust and accurate fingerprint matching.
- Implement real-time processing: Optimize the system for real-time applications, such as mobile device authentication, by exploring hardware acceleration techniques and optimizing the code for performance.
- Enhance robustness to various image acquisition conditions: Investigate techniques to improve the system's robustness to variations in image quality, such as rotation, scaling, and partial fingerprints.

We will continue to work on enhancing the robustness of our system by addressing the challenges identified in this project, particularly focusing on improving resilience to image quality variations.

Bibliography

[1] Ruiz-Garcia, S. SOCOFing Fingerprint Dataset. https://www.kaggle.com/datasets/ruizgara/socofing

[2] Jain, A. K., Hong, L., & Pankanti, S. (2000). Fingerprint image pre-processing: A review. Pattern recognition, 33(12), 2025 2044 https://www.researchgate.net/publication/247836092_Fingerprint_image_pre-_and-post-processing

[3] Nitheshkamath https://github.com/Nitheshkamath/Fingerprint

[4] OpenCV Documentation. https://docs.opencv.org/4.x/