Faculty of Engineering and Technology

Computer Science Department

Computer Security

Assignment #1 Report

Biometric Authentication System

Prepared by:

Lana Zain 1201466

Eyas Shalhoub 1201681

Instructor:

Dr. Mohammad Khanafseh

Section: 2

Date: 05/05/2024

Abstract

Fingerprint recognition is a widely utilized biometric technology known for its reliability and accuracy in identifying individuals based on unique fingerprint patterns. This report provides a comprehensive analysis of a fingerprint recognition system developed for authentication purposes. The system employs sophisticated algorithms for feature extraction and matching, contributing to its effectiveness in verifying individual identities. Evaluation results, including metrics such as False Match Rate (FMR), False Non-Match Rate (FNMR) and Receiver Operating Characteristic (ROC) curve.

Table Of Contents

Table of figures

System Design

The system is designed as a fingerprint recognition system, utilizing the unique patterns present in an individual's fingerprints for authentication purposes. Fingerprint recognition was chosen as the modality for several reasons, including its high level of accuracy, widespread acceptance, and ease of use.

Fingerprint Recognition

Fingerprint recognition begins with capturing an individual's fingerprint image using specialized sensors. Advanced algorithms then analyze the image to identify unique features called minutiae points, including ridge endings, bifurcations, and ridge orientation. These minutiae points serve as the fingerprint's distinct characteristics. Subsequently, the system compares these extracted features with stored templates to determine a match, forming the basis of reliable biometric authentication.

Fingerprint recognition offers several advantages over other biometric modalities:

1. **Uniqueness:** Each person's fingerprint is unique, making it highly reliable for individual identification.
2. **Universality:** Almost everyone has fingerprints, making this modality applicable to a wide range of individuals.
3. **Consistency:** fingerprint patterns remain relatively stable throughout a person's life, ensuring consistency in authentication over time
4. **Ease of Capture:** Capturing fingerprints is non-invasive and straightforward, requiring only a simple touch-based sensor.

Algorithms Used in Matching and Threshold Selection

1. **Matching:**

Once features are extracted, matching algorithms compare them with stored templates to determine similarity. In our system, we utilize the ‘Euclidean distance’ algorithm, which measures the geometric distance between two sets of minutiae points.

During the matching process, the features of the user, stored in a binary file after registration, are compared with the input features to be matched. The outcome of this comparison is then normalized to conclusively determine the authentication result. This normalization process ensures that the matching result is consistent and reliable across different fingerprint samples and users.

2. **Threshold Selection:**

The main decider for a match in the system is the difference in distance between the saved and input features, this indicator is present as a numerical value, mostly between 0 and 2. But a threshold value can't be guessed, so the system's job is to choose the best threshold numerical value.

System Architecture

The system architecture includes three main components, seamlessly integrating hardware and software functionalities:

1. **Image Acquisition:**

   This component involves loading the fingerprint image from the dataset and storing the intensity values of each observation in an array.
2. **Model fitting:**

   - Assign training and testing images and labels from the loaded files.
   - Adjust the model's parameter to ensure the best results for the input images, such as image size, density and wanted features.
   - Fit the model twice: on 10 chunks (epochs) and 25.
3. **Model Accuracy and loss test:**

   - Plotted model Accuracy:

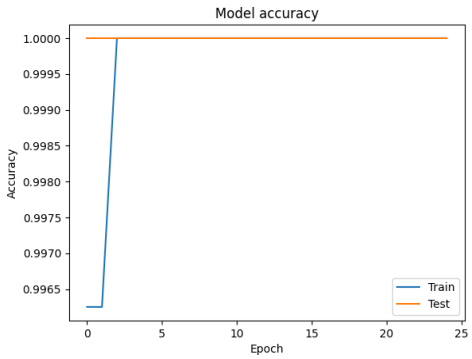

Figure 0-1:Model Accuracy

- Plotted model loss:

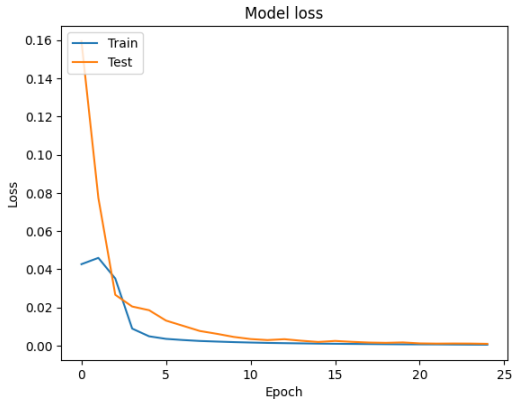

Figure 0-2: Model loss

4. **Feature Extraction:**

    - In this stage, the captured fingerprint image undergoes processing to extract unique features, such as ridge detection, minutiae extraction, and orientation field analysis.

5. **Matching and Verification:**

    - Extracted features are compared with stored templates to determine a match. Matching algorithms.
    - Thresholding techniques are then employed to determine the threshold for accepting or rejecting a match.

6. **Storage and Communication:**
    - Templates of registered fingerprints are stored in binary -.pickle- files, for comparison during authentication.

7. **Evaluation:**
    - False non-match rate (FNMR): A known fingerprint image is matched with the template of the user, knowing that it should result in a match, 8 threshold values between 0 and 2 are tested for all the users.
    - False match rate (FMR): Impostors images are matched with users templates over the same 8 thresholds (between 0 and 2), which should result in a non-match by the system.
    - Receiver Operating Characteristic curve (ROC): the curve indicates the optimal threshold value which is called the equal error rate, which occurs when the FNMR and FMR values intercept.

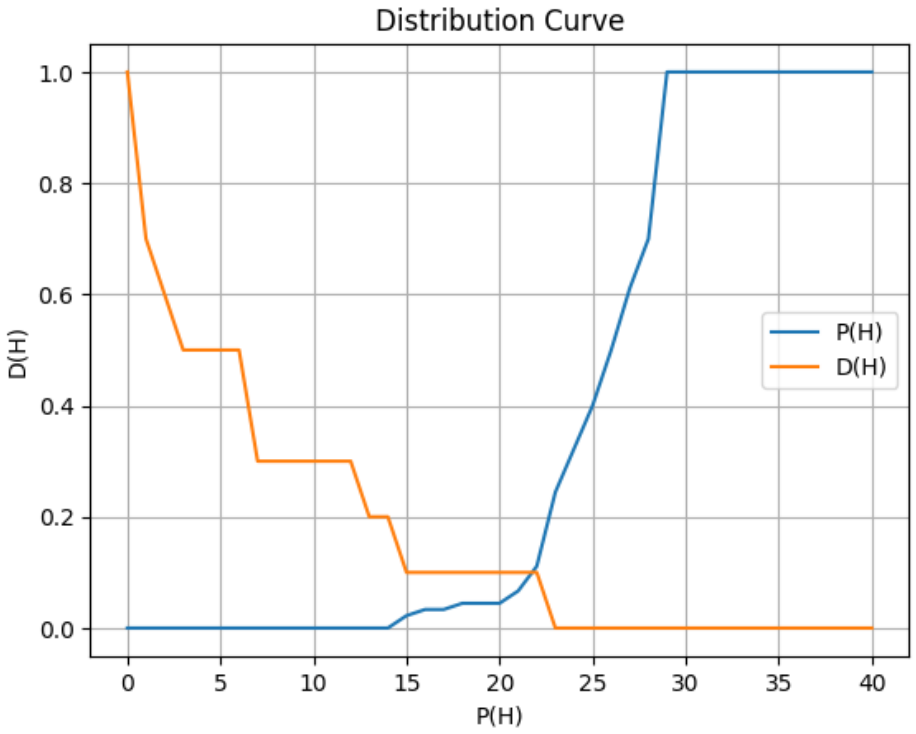

Figure 0-3: Distribution Curve

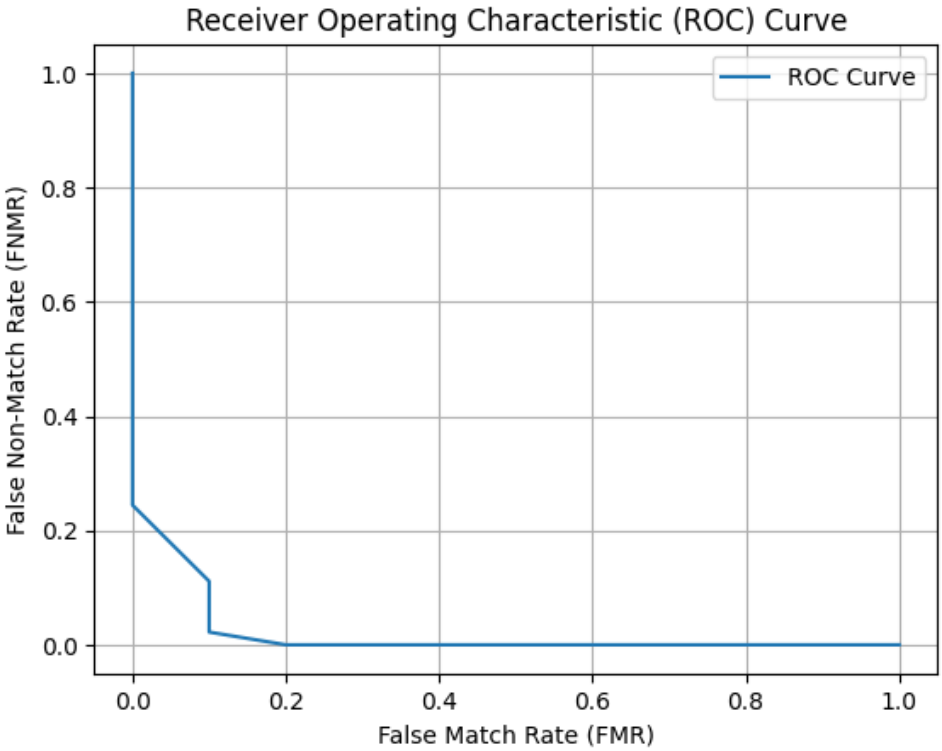

Figure 0-4: ROC curve

Source Code

Imports

- Pyplot: python's plotting library to help us understand and evaluate the model.
- Os and joblib: these libraries are needed to manipulate and access folders.
- CV2: library to deal with images, grayscale generation and reshaping.
- Tensorflow: machine learning library.

```
import matplotlib.pyplot as plt
import joblib
import os
import cv2
import numpy as np
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import models, layers
```

Image loading

- Load images from folders train and test.
- Convert all the images to grayscale.
- Resize the images.
- Assign training and testing images to their variables.

```
def load_images_from_folder(folder):

  images = []
  labels = []

  # Iterate over all of the files in the folder
  for filename in os.listdir(folder):

    img = cv2.imread(os.path.join(folder,filename), cv2.IMREAD_GRAYSCALE)

    if img is not None:

      # Add the image and label to the corresponding lists
      images.append(img)
      labels.append(int(filename.split('_')[0].split('.')[0]))

  return images, labels

x_train, y_train = load_images_from_folder('/content/dataset/train/')
x_test, y_test = load_images_from_folder('/content/dataset/test')
```

```
# Resize all of the images in the training dataset to be 160x160
x_train = [cv2.resize(img, (160, 160)) for img in x_train]
# Convert the list of training images to a NumPy array
x_train = np.array(x_train).reshape(-1, 160, 160, 1).astype('float32') / 255.0

# Convert the list of test images to a NumPy array
x_test = np.array(x_test).reshape(-1, 160, 160, 1).astype('float32') / 255.0

# Convert the list of training labels to one-hot format
y_train = to_categorical(y_train)

# Convert the list of test labels to one-hot format
y_test = to_categorical(y_test)
```

Model training

The CNN model is created (sequential model) with the needed parameters passed based on the input images and the desired outcome.

The model is fitted twice, first time with 10 iterations with testing, to be evaluated in the next step, seconds fit will have 25 iterations with evaluation passed directly.

```
model = models.Sequential()
# Add a convolutional layer with 32 filters of size 3x3
model.add(layers.Conv2D(32, (3, 3), activation="relu", input_shape=(160, 160, 1)))
# Add a max pooling layer with a pooling size of 2x2
model.add(layers.MaxPooling2D((2, 2)))
# Add another convolutional layer with 64 filters of size 3x3
model.add(layers.Conv2D(64, (3, 3), activation="relu"))
# Add another max pooling layer with a pooling size of 2x2
model.add(layers.MaxPooling2D((2, 2)))
# Flatten the output of the previous layer
model.add(layers.Flatten())
# Add a dense layer with 128 neurons
model.add(layers.Dense(128, activation="relu"))
# Add a dense layer with 10 neurons, one for each fingerprint class
model.add(layers.Dense(10, activation="softmax"))

# Compile the model using the Adam optimizer and the categorical cross-entropy loss
function
model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])

# Train the model for 10 epochs on the training dataset
model.fit(x_train, y_train, epochs=10)

# Evaluate the model on the test dataset and print the test loss and test accuracy
score = model.evaluate(x_test, y_test, verbose=0)
history = model.fit(x_train, y_train, epochs=25, validation_data=(x_test, y_test))
```

Registration and authentication

When adding a user, they are assigned an integer value saved as user_id to later differentiate their template from other users.

Registration: Features of the input image is extracted and saved in a binary (.pickle) file.

Authentication: the input image is compared with the features in the template that is loaded according to the user_id integer that is passed.

```
def register_fingerprint(image, user_id):

    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (160, 160))
    image = image.reshape(-1, 160, 160, 1).astype('float32') / 255.0

    features = model.predict(image)

    # Save the features
    features = features.flatten()
    with open(f'dataset/features/fingerprint_features{user_id}.pickle', 'wb') as f:
        joblib.dump(features, f)

def authenticate_fingerprint(image, user_id,threshold=0.25):

    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (160, 160))
    image = image.reshape(-1, 160, 160, 1).astype('float32') / 255.0

    features = model.predict(image)
    # Compare the features
    with open(f'dataset/features/fingerprint_features{user_id}.pickle', 'rb') as f:
        known_features = joblib.load(f)

    # Use the distance function to compare the features
    dist = np.linalg.norm(features - known_features, axis=1)
    print(dist)

    if dist < threshold:
        return True
    else:
        return False
```

Evaluation

Thresholds values are stored in a numPy array ranging from 0 to 2 with 0.05 difference.

FNMR: each user is compared with their template using all threshold values, each match is stored as a Boolean and the results of each user is counted and stored.

```
thresholds = np.arange(0, 2.025, 0.05)
FNMR=[]
for t in thresholds:
  i=0
  matches=[]
  for fn in sorted(os.listdir('/content/dataset/test/')):
    img = cv2.imread(os.path.join('/content/dataset/test/',fn))
    if img is not None:
      auth_result=authenticate_fingerprint(img,i,t)
      print(fn , i, auth_result)
      matches.append(auth_result)
      i=i+1
  FNMR_t= matches.count(False)/(matches.count(True)+matches.count(False))
  FNMR.append(FNMR_t)
```

FMR: each user's features are compared with all the other templates but theirs over all the threshold values, each match is counted and later saved as a FMR.

```
FMR = []
for t in thresholds:
    non_matches_all = []
    for j in range(10):
        non_matches = []
        for i, fn in enumerate(sorted(os.listdir('/content/dataset/test/'))):
            if i != j:
                img = cv2.imread(os.path.join('/content/dataset/test/', fn))
                if img is not None:
                    auth_result = authenticate_fingerprint(img, j, t)
                    print( j , i)
                    print(fn , i, auth_result)
                    non_matches.append(auth_result)
        non_matches_all.append(non_matches.count(True) / 9)
    FMR.append(np.mean(non_matches_all))
```

Equal error rate

The EER is the sweet spot that indicates the least FNMR and FMR, the index of the matching records is stored, so that the optimal threshold is known.

The EER is calculated.

```
# Determine Equal Error Rate (EER)
eer_index = np.argmin(np.abs(FMR - FNMR))
eer_threshold = thresholds[eer_index]
eer_value = (FMR[eer_index] + FNMR[eer_index]) / 2
print("Equal Error Rate (EER):", eer_value)
print("Threshold at EER:", eer_threshold)
```

After executing the code, the EER value is calculated to be approximately 0.1056 at the threshold value of 1.1:

```
Equal Error Rate (EER): 0.10555555555555557
Threshold at EER: 1.1
```

Optimal FNMR and FMR values

From the past section the EER index will indicate the value of FNMR and FMR at the optimal threshold

```
print("FNMR at EER",FNMR[eer_index])
print("FMR at EER ",FMR[eer_index])
```

```
FNMR at EER 0.1
FMR at EER  0.11111111111111112
```

Potential System Improvements

Better Model Comparison Testing

Incorporating Mean Squared Error (MSE) or Root Mean Squared Error (RMSE) into model comparison testing provides deeper insights into prediction accuracy and error magnitude, helping towards a more informed model selection.

Increasing Dataset Size

Expanding the dataset enhances model robustness and generalization by providing more diverse examples for learning, ultimately improving model performance across various scenarios.

Image Enhancement Techniques

Utilizing techniques like sharpening improves input image quality, which can result in better feature extraction and enhancing model performance for fingerprint authentication, but risk more FMR cases.

Conclusion

In summary, this report outlines our fingerprint recognition system, which is designed to authenticate users based on their unique fingerprint patterns. We've used advanced algorithms and thorough testing to ensure the system works reliably for verifying identities. The attached research paper helped as understand the connection between convolutional neural network models and fingerprint and finger vein authentication.

References

- https://www.researchgate.net/publication/337720080_Convolutional_Neural_Networks_Approach_for_Multimodal_Biometric_Identification_System_Using_the_Fusion_of_Fingerprint_Finger-vein_and_Face_images
- https://colab.research.google.com/drive/1JfsM1z2BuF_zxOX2BPatCqk0iHe0nLj-?usp=sharing