School of Computer Science Engineering and Information Systems

FALL SEMESTER 2024-25

SWE1015 BIOMETRIC SYSTEMS

J-COMPONENT

FINAL REVIEW

<u>PROJECT TITLE:</u>

Fingerprint Recognition using Deep Learning

SUBMITTED TO :PRABUKUMAR M

TEAM MEMBERS:

22MIS0051 SRIRAM V

22MIS0069 SARAN T S

22MIS0133 ARUN ADHITHYA V

Table of Contents

1. Abstract

Fingerprint recognition has emerged as a cornerstone in biometric authentication and identification, serving crucial roles in applications ranging from forensic investigations to secure access systems and personal device authentication. Traditional fingerprint recognition methods, leveraging handcrafted features and rule-based algorithms, have provided a robust foundation for the field. However, the advent of deep learning has significantly transformed this domain, enabling automated feature extraction and enhanced recognition accuracy. In this study, we explored various deep learning algorithms for fingerprint recognition, focusing on convolutional neural networks (CNNs), which excel in image-based tasks. Our findings demonstrate that CNN-based architectures outperform traditional methods and other deep learning models in terms of accuracy, scalability, and adaptability. These results highlight the potential of CNNs as the preferred approach for building state-of-the-art fingerprint recognition systems, setting the stage for further innovations in biometric technology.

2. Literature Review:

No.	Paper/Journal/Conference Name & Year	Brief Description of Solution	Key Functionalities/Features	Limitations/Shortcomings	Remarks
1	FingerNet: An End-to-End Fingerprint Recognition System Using CNN, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2018	End-to-end CNN for fingerprint recognition.	Automated feature extraction and matching.	Limited performance with noisy fingerprints.	Effective for structured datasets.
2	DeepPrint: Scalable and Robust Fingerprint Recognition Using CNN, CVPR, 2019	Scalable CNN for large-scale databases.	Robust to latent fingerprints.	Issues with distorted fingerprints.	Suitable for forensics.
3	Latent Fingerprint Matching Using CNN, Pattern Recognition Letters, 2020	CNN to improve latent fingerprint	Extracts features from partial or smudged fingerprints.	Computationally demanding for high-res images.	Promising for forensic applications.

		matching.			
4	Survey on Deep Learning Techniques for Fingerprint Recognition, IEEE Access, 2021	Review of CNN techniques for fingerprint recognition.	Summarizes CNN architectures and datasets.	Limited focus on real-time systems.	Useful for researchers in biometrics.
5	Siamese Networks for Fingerprint Verification, International Conference on Biometrics, 2020	Utilizes Siamese networks for fingerprint similarity checks.	Captures pairwise similarity efficiently.	High memory requirements for large datasets.	Useful for small-scale applications.
6	MinutiaeNet: End-to-End Latent Fingerprint Recognition, IEEE Transactions on Biometrics, 2019	Combines CNN with minutiae-based matching.	Strong performance on latent fingerprints.	Requires preprocessing for poor-quality images.	Practical for hybrid recognition systems.
7	CNN-Based Fingerprint Image Enhancement, Journal of Biometrics, 2020	CNN to improve fingerprint image clarity.	Enhances noisy or smudged fingerprints.	Limited generalization for diverse datasets.	Beneficial for preprocessing pipelines.
8	Robust Fingerprint Recognition with Distorted Images, ECCV, 2019	CNN model designed to handle distorted fingerprints.	Resilient to geometric distortions.	Limited dataset diversity affects generalization.	Effective for real-world distortions.
9	Transfer Learning for Fingerprint Recognition, IEEE International Conference on Image Processing, 2021	Applies pre-trained CNN models for fingerprint recognition.	Reduces training time and data dependency.	May not perform well for unique fingerprint datasets.	Suitable for resource-constrained projects.
10	Adaptive CNN for Fingerprint Spoof	A CNN approach to detect	Distinguishes between real and fake fingerprints.	Limited robustness to unseen spoofing techniques.	Enhances biometric security.

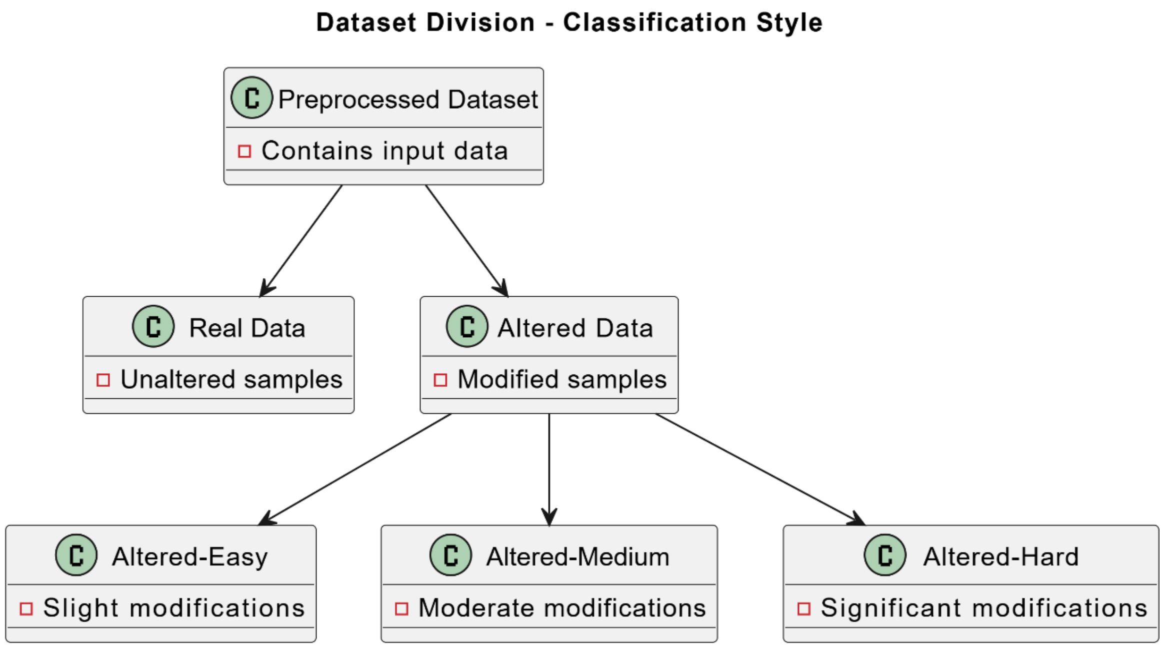
	Detection, IEEE Biometrics, 2020	fake fingerprints.			
11	LightNet: Lightweight CNN for Mobile Fingerprint Recognition, Journal of Pattern Recognition, 2021	Lightweight CNN for mobile devices.	Low computational cost, optimized for embedded systems.	Accuracy trade-offs for smaller models.	Useful for mobile authentication.
12	Multi-Task CNN for Fingerprint Recognition and Classification, ICASSP, 2020	Multi-task learning framework for recognition and classification.	Combines identification and classification tasks.	Complex architecture increases computational demand.	Suitable for multifunctional systems.
13	Cross-Domain Fingerprint Recognition Using CNN, Pattern Analysis and Applications, 2021	Domain adaptation techniques for CNN in fingerprint recognition.	Robust across multiple datasets with varied conditions.	Requires fine-tuning for new domains.	Effective for multi-environment systems.
14	End-to-End Learning of Fingerprint Features, CVPR, 2018	CNNs trained directly on raw fingerprint images.	Eliminates manual feature engineering.	Sensitive to poor-quality input.	Good for clean datasets.
15	Enhancing Fingerprint Recognition with GAN Augmented Training Data, ICCV, 2020	Uses GANs to augment fingerprint training data for CNNs.	Improves recognition on smaller datasets.	GAN training can be unstable.	Useful for datasets with limited samples.

	Paper/Journal/COnference Name & Year	**Brief Descriptio Of Solution**	**Key Functionalities/ Features**	**Limitations/Short comings**	**Remarks**
16	Android Malware Detection with Graph Neural Networks, ICLR, 2021	GNN for analyzing APK structures.	Handles complex dependencies.	Requires feature engineering.	Effective for hierarchical data.
17	Adversarial Malware Detection in Android, IEEE Security & Privacy, 2020	CNN model resilient to adversarial attacks.	Defends against obfuscated malware.	Performance dependent on attack type.	Enhances security.
18	Deep Android Malware Detection with Dynamic Features, ACM CCS, 2019	Leverages runtime behaviors for detection.	Captures dynamic app interactions.	High resource requirements.	Suitable for detailed analysis.
19	Lightweight CNN for Resource-Constrained Malware Detection, ICML, 2021	Compact CNN for mobile malware detection.	Optimized for mobile platforms.	Accuracy trade-offs for speed.	Ideal for embedded systems.
20	Combining Static and Dynamic Features for Malware Analysis, IEEE Access, 2020	Hybrid CNN framework for feature fusion.	Improves detection with complementary data.	Complex training pipeline.	Suitable for hybrid methods.
21	Real-Time Android Malware Detection Using RNN, Neural Networks Journal, 2020	RNN-based model for temporal patterns.	Handles sequential data.	Limited scalability.	Good for time-sensitive analysis.
22	Zero-Day Malware Detection Using Deep Learning, CVPR, 2019	CNN-based zero-day malware detection.	Identifies unseen threats.	False positives in noisy environments.	Robust for emerging threats.
23	Enhancing Mobile Security with CNN, IJML, 2020	Mobile-focused malware detection system.	Lightweight and efficient.	Limited adaptability.	Practical for mobile devices.

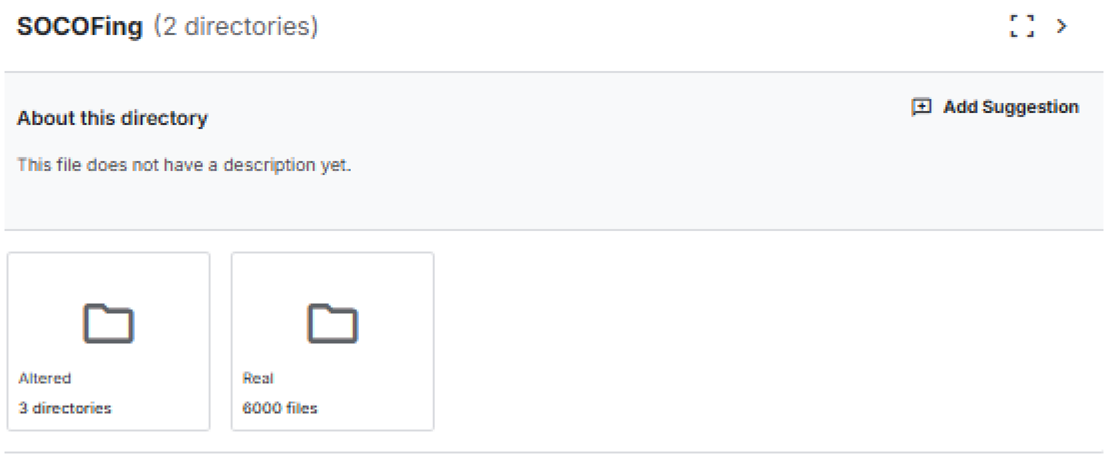
24	A Comprehensive Survey of Android Malware Detection, Springer, 2021	Summarize trends in Android malware research.	Broad review of methodologies.	Limited experimental insights.	Useful for beginners .
25	Transfer Learning for Cross-Domain Malware Detection, IEEE TNNLS, 2020	Uses pre-trained CNN models for adaptation.	Reduces dataset dependency.	Domain-specific tuning needed.	Suitable for versatile systems.
26	Detecting Obfuscated Malware with GANs, ICCV, 2021	GAN-generated samples for CNN training.	Overcomes data scarcity.	GAN training complexity.	Useful for evasion-resistant models.
27	Malware Classification Using Capsule Networks, NIPS, 2018	Capsule networks for malware classification	Captures spatial relationships.	Computationally intensive.	Promising for structured data.
28	Deep Autoencoders for Malware Feature Extraction, IJCV, 2020	Autoenco - ders for unsupervise d feature learning.	Handles high-dimensional data.	Needs fine-tuning.	Effective for preprocessing.
29	Ensemble Learning for Android Malware Detection, IEEE Access, 2021	Combine s multiple models for robust detection.	Enhan ces accura cy.	Increased complexity.	Good for large- scale systems.
30	CNN-Based Analysis of APK Metadata, Pattern Recognition Journal, 2019	Analyzes APK metadata with CNN.	Lightweight and interpretable.	Dataset limitations.	Practical for quick scans.

3. Methodology:

i) Benchmark Dataset Description:



Dataset from (https://www.kaggle.com/datasets/ruizgara/socofing)


SOCOFing (2 directories)

About this directory

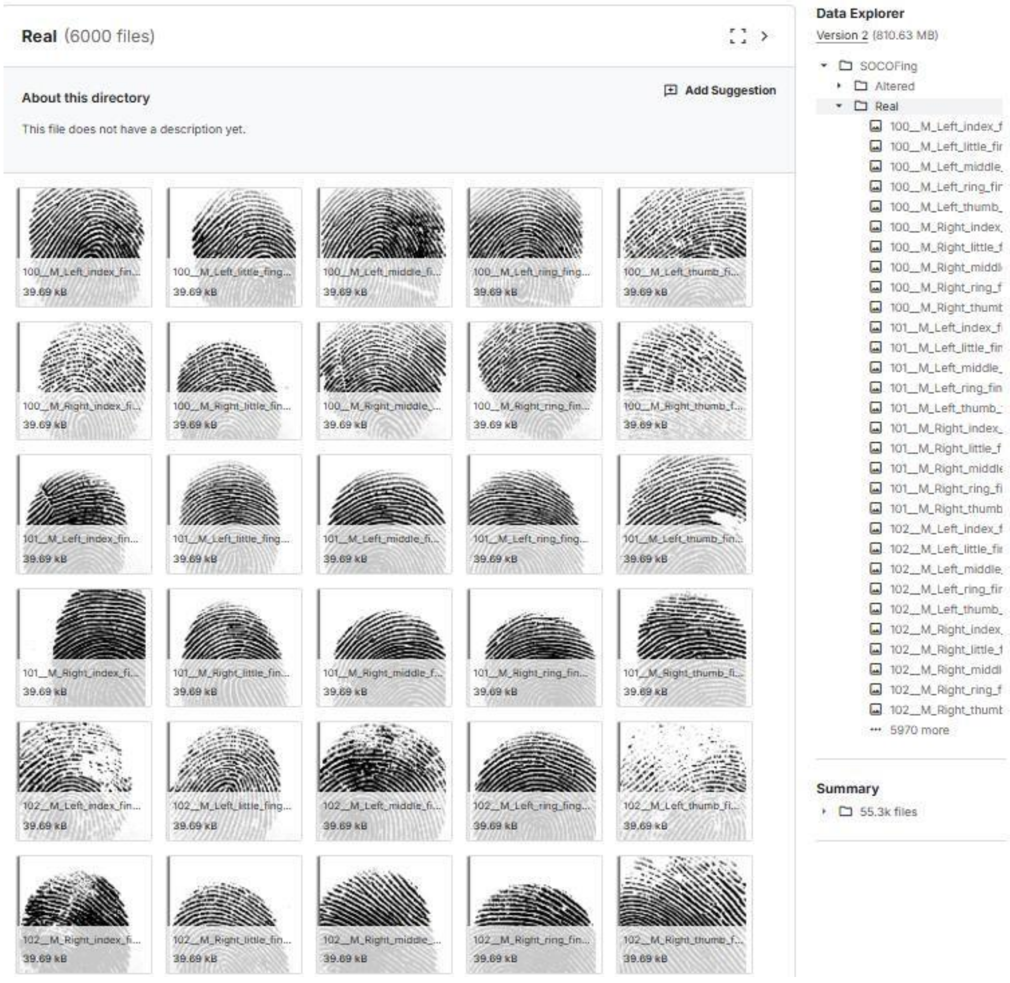
Add Suggestion

This file does not have a description yet.

Altered
3 directories

Real
6000 files

Real:


Real (6000 files)

About this directory

Add Suggestion

This file does not have a description yet.

100__M_Left_index_fin...
39.69 kB

100__M_Left_little_fing...
39.69 kB

100__M_Left_middle_fi...
39.69 kB

100__M_Left_ring_fing...
39.69 kB

100__M_Left_thumb_fi...
39.69 kB

100__M_Right_index_fi...
39.69 kB

100__M_Right_little_fin...
39.69 kB

100__M_Right_middle_...
39.69 kB

100__M_Right_ring_fin...
39.69 kB

100__M_Right_thumb_f...
39.69 kB

101__M_Left_index_fin...
39.69 kB

101__M_Left_little_fing...
39.69 kB

101__M_Left_middle_fi...
39.69 kB

101__M_Left_ring_fing...
39.69 kB

101__M_Left_thumb_fin...
39.69 kB

101__M_Right_index_fi...
39.69 kB

101__M_Right_little_fin...
39.69 kB

101__M_Right_middle_f...
39.69 kB

101__M_Right_ring_fin...
39.69 kB

101__M_Right_thumb_fi...
39.69 kB

102__M_Left_index_fin...
39.69 kB

102__M_Left_little_fing...
39.69 kB

102__M_Left_middle_fi...
39.69 kB

102__M_Left_ring_fing...
39.69 kB

102__M_Left_thumb_fi...
39.69 kB

102__M_Right_index_fi...
39.69 kB

102__M_Right_little_fin...
39.69 kB

102__M_Right_middle_...
39.69 kB

102__M_Right_ring_fin...
39.69 kB

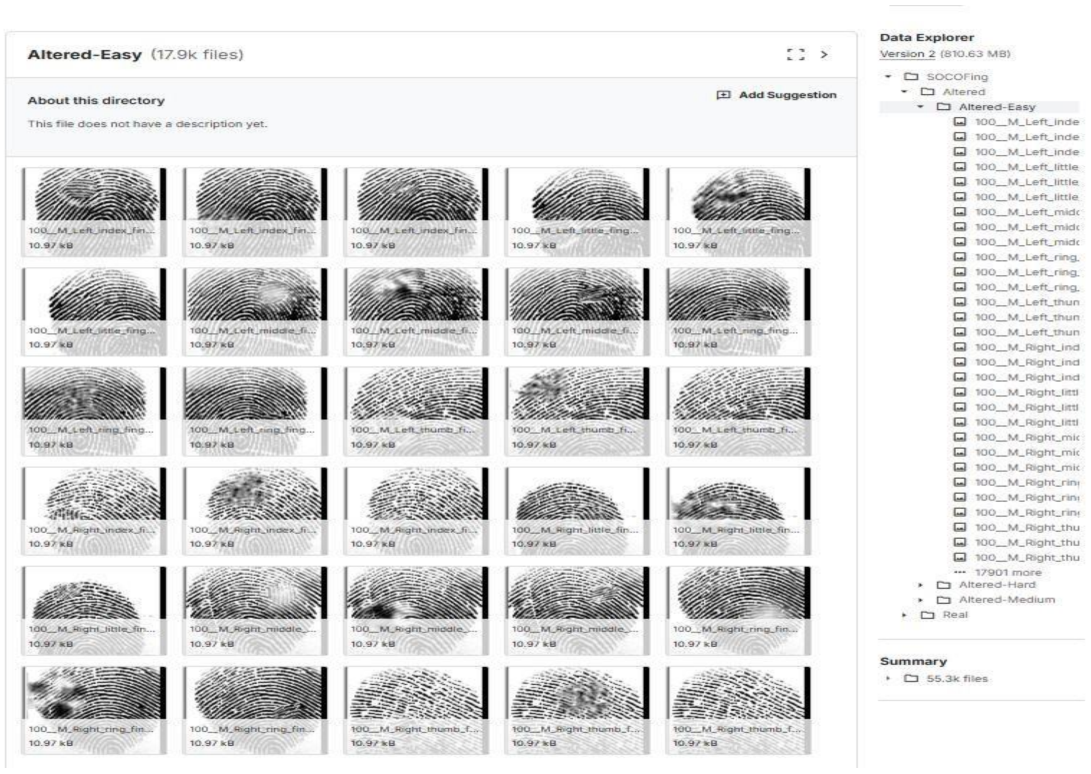
102__M_Right_thumb_f...
39.69 kB

Data Explorer
Version 2 (810.63 MB)

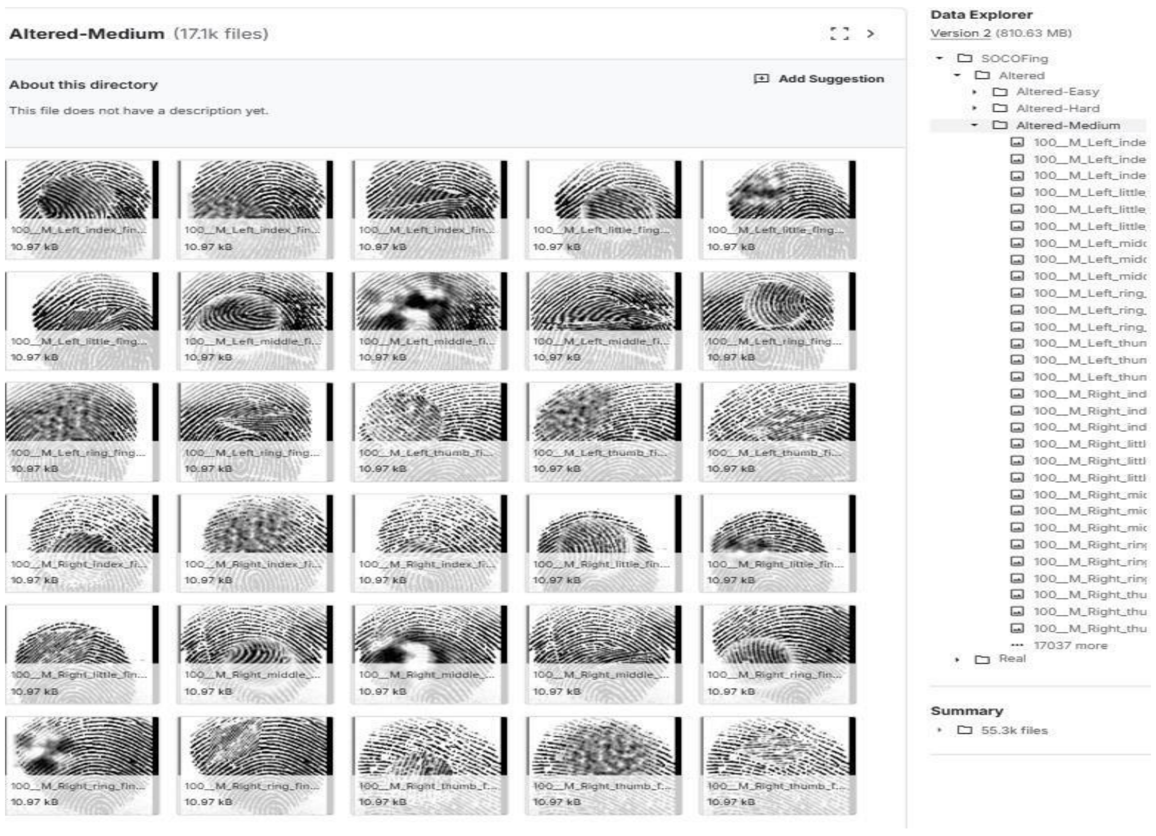
- SOCOFing
  - Altered
  - Real
    - 100__M_Left_index_f
    - 100__M_Left_little_fir
    - 100__M_Left_middle_
    - 100__M_Left_ring_fir
    - 100__M_Left_thumb_
    - 100__M_Right_index_
    - 100__M_Right_little_f
    - 100__M_Right_middl
    - 100__M_Right_ring_f
    - 100__M_Right_thumt
    - 101__M_Left_index_f
    - 101__M_Left_little_fir
    - 101__M_Left_middle_
    - 101__M_Left_ring_fin
    - 101__M_Left_thumb_
    - 101__M_Right_index_
    - 101__M_Right_little_f
    - 101__M_Right_middle
    - 101__M_Right_ring_fi
    - 101__M_Right_thumb
    - 102__M_Left_index_f
    - 102__M_Left_little_fir
    - 102__M_Left_middle_
    - 102__M_Left_ring_fir
    - 102__M_Left_thumb_
    - 102__M_Right_index_
    - 102__M_Right_little_f
    - 102__M_Right_middl
    - 102__M_Right_ring_f
    - 102__M_Right_thumt
    - ··· 5970 more

Summary
- 55.3k files

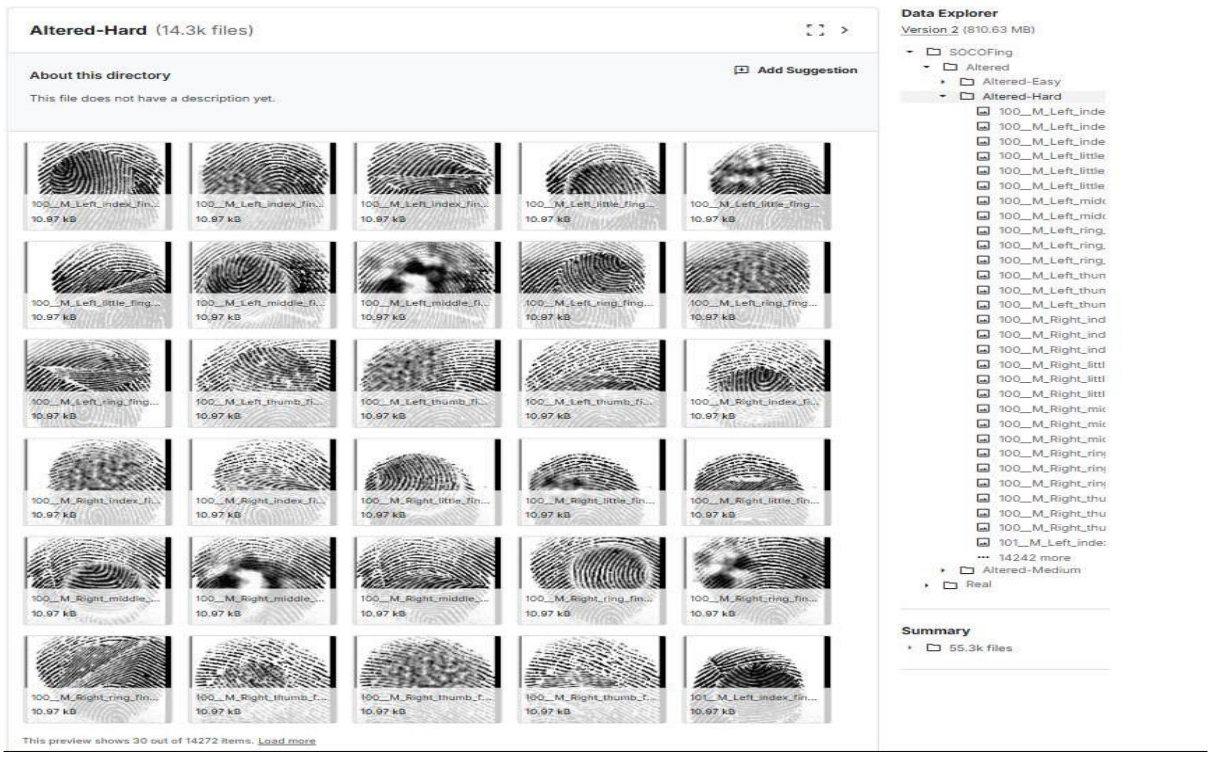
Altered-Easy:



Altered-Medium:



Altered-Hard:



Altered-Hard (14.3k files)

About this directory

This file does not have a description yet.

Add Suggestion

100__M_Left_index_fin... 10.97 kB
100__M_Left_index_fin... 10.97 kB
100__M_Left_index_fin... 10.97 kB
100__M_Left_little_fing... 10.97 kB
100__M_Left_little_fing... 10.97 kB
100__M_Left_little_fing... 10.97 kB
100__M_Left_middle_fi... 10.97 kB
100__M_Left_middle_fi... 10.97 kB
100__M_Left_ring_fing... 10.97 kB
100__M_Left_ring_fing... 10.97 kB
100__M_Left_ring_fing... 10.97 kB
100__M_Left_thumb_fi... 10.97 kB
100__M_Left_thumb_fi... 10.97 kB
100__M_Left_thumb_fi... 10.97 kB
100__M_Right_index_fi... 10.97 kB
100__M_Right_index_fi... 10.97 kB
100__M_Right_index_fi... 10.97 kB
100__M_Right_little_fin... 10.97 kB
100__M_Right_little_fin... 10.97 kB
100__M_Right_little_fin... 10.97 kB
100__M_Right_middle_... 10.97 kB
100__M_Right_middle_... 10.97 kB
100__M_Right_middle_... 10.97 kB
100__M_Right_ring_fin... 10.97 kB
100__M_Right_ring_fin... 10.97 kB
100__M_Right_ring_fin... 10.97 kB
100__M_Right_thumb_f... 10.97 kB
100__M_Right_thumb_f... 10.97 kB
100__M_Right_thumb_f... 10.97 kB
101__M_Left_index_fin... 10.97 kB

This preview shows 30 out of 14272 items. Load more

Data Explorer

Version 2 (810.63 MB)

- SOCOFing
  - Altered
    - Altered-Easy
    - Altered-Hard
      - 100__M_Left_inde
      - 100__M_Left_inde
      - 100__M_Left_inde
      - 100__M_Left_little
      - 100__M_Left_little
      - 100__M_Left_little
      - 100__M_Left_midc
      - 100__M_Left_midc
      - 100__M_Left_ring_
      - 100__M_Left_ring_
      - 100__M_Left_ring_
      - 100__M_Left_thun
      - 100__M_Left_thun
      - 100__M_Left_thun
      - 100__M_Right_ind
      - 100__M_Right_ind
      - 100__M_Right_ind
      - 100__M_Right_littl
      - 100__M_Right_littl
      - 100__M_Right_littl
      - 100__M_Right_mic
      - 100__M_Right_mic
      - 100__M_Right_mic
      - 100__M_Right_rin
      - 100__M_Right_rin
      - 100__M_Right_rin
      - 100__M_Right_thu
      - 100__M_Right_thu
      - 100__M_Right_thu
      - 101__M_Left_inde:
      - ··· 14242 more
    - Altered-Medium
  - Real

Summary

- 55.3k files

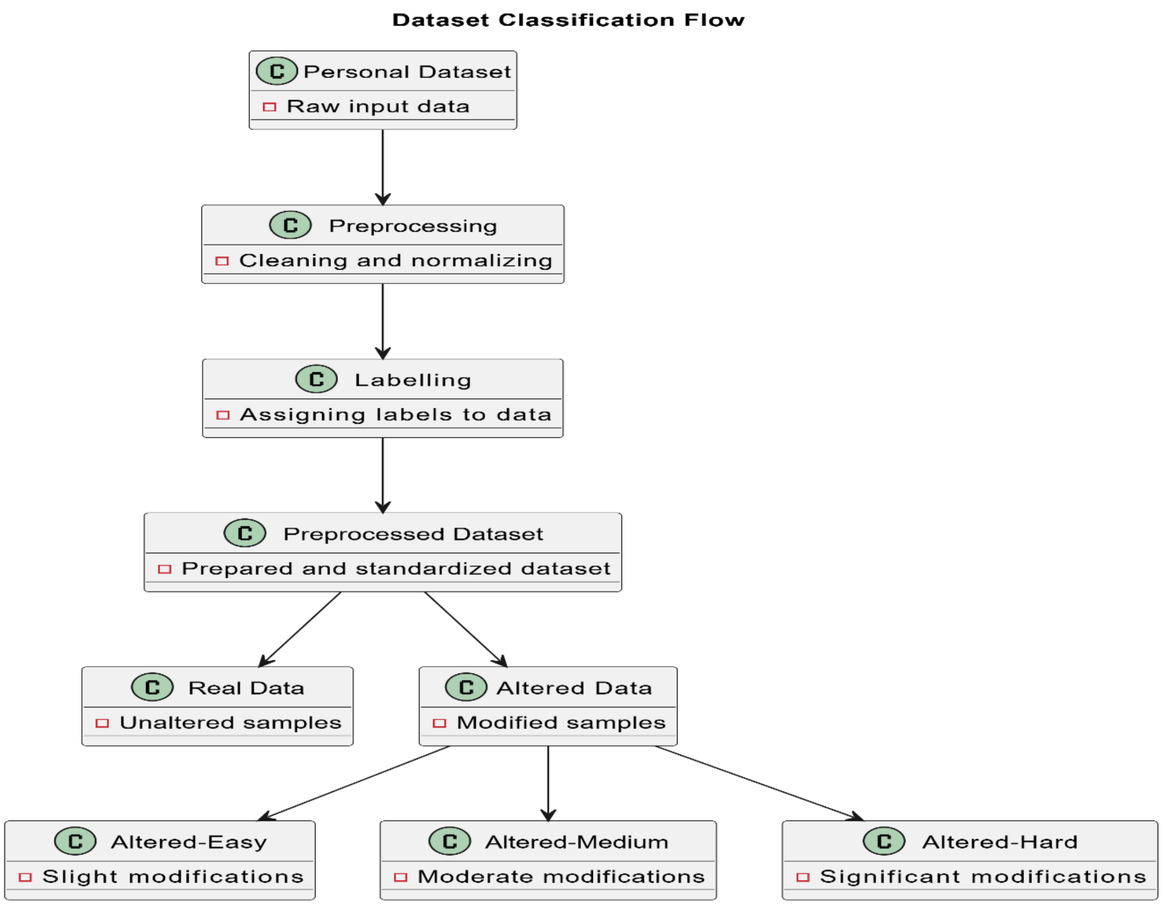
ii) Personal Dataset:

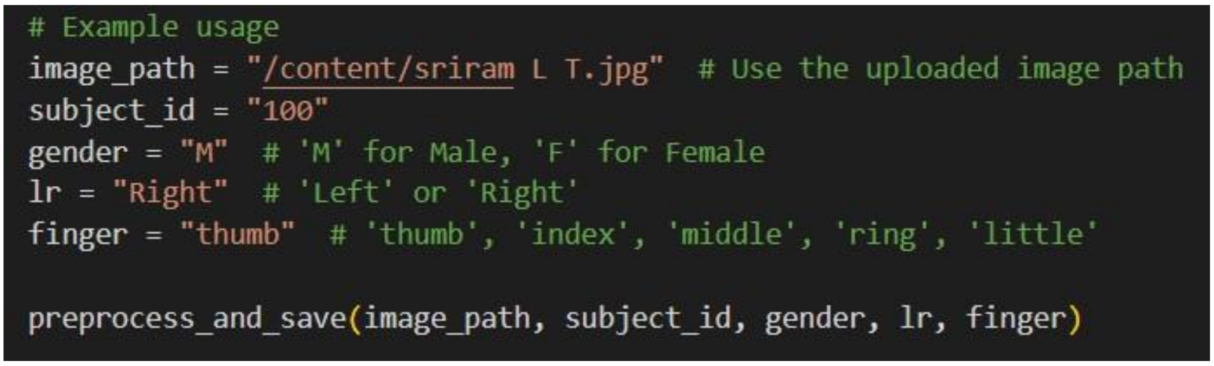
Sriram_R-T

Arun_L-T

Arun_L-T

(illuminated)





```
# Example usage
image_path = "/content/sriram L T.jpg"  # Use the uploaded image path
subject_id = "100"
gender = "M"  # 'M' for Male, 'F' for Female
lr = "Right"  # 'Left' or 'Right'
finger = "thumb"  # 'thumb', 'index', 'middle', 'ring', 'little'

preprocess_and_save(image_path, subject_id, gender, lr, finger)
```

4. Modules Explanation:

i) Data Loading and Preprocessing

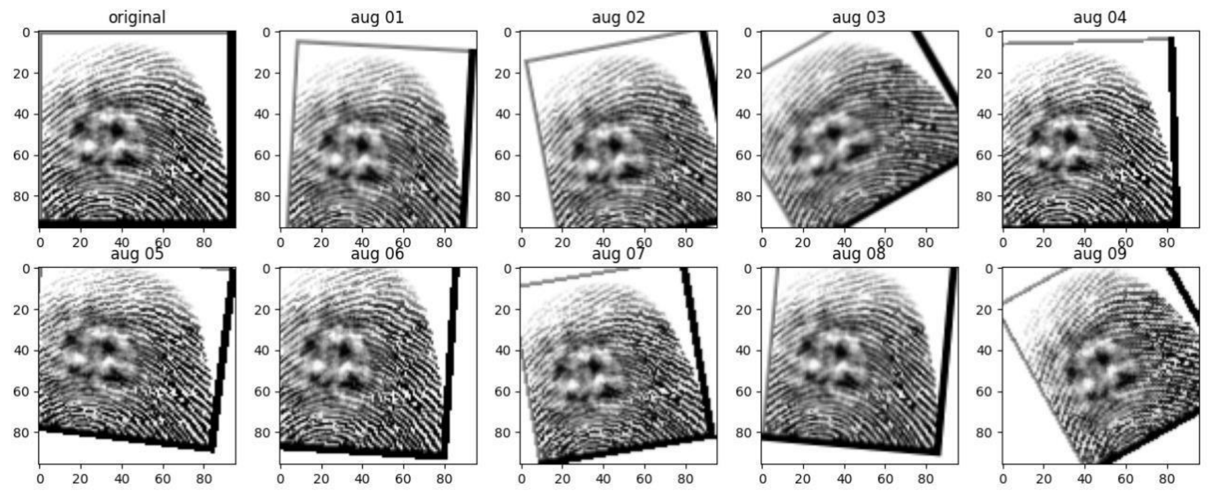
In the beginning, your code loads and preprocesses the dataset using various modules.

- **cv2 (OpenCV):** This is used for image loading and resizing. Images are read in grayscale mode (cv2.IMREAD_GRAYSCALE), resized to 96x96 pixels, and reshaped into arrays of dimensions (96, 96, 1) to match the input shape expected by the model.
- **numpy:** Used to store the images and labels as arrays. The imgs and labels arrays are initialized with a specific shape to store the dataset.
- **glob:** This module helps in listing files in the specified directory that match the pattern *.BMP (case-insensitive), essentially helping in loading all the BMP image files into the dataset.
- **matplotlib.pyplot:** This is used for plotting and visualizing images in the dataset, as well as to plot confusion matrices, classification reports, and ROC curves.

ii) Data Augmentation

To augment the dataset and artificially increase the variety of training samples, the imgaug library is used.

- **imgaug (Image Augmentation Library):** It helps create variations of the original images to train the model more robustly. Operations like rotation, scaling, translation, and Gaussian blur are applied to images. For example:
    - **iaa.Affine:** Allows transformations like rotation, translation, and scaling of the images.
    - **iaa.GaussianBlur:** Adds a small amount of blur to simulate different camera conditions.
    - **augment_images:** It applies the augmentation sequence on the batch of images.



iii) Data Generator Class

The DataGenerator class is defined to create batches of augmented images for training and validation. It's a subclass of keras.utils.Sequence, which is specifically designed for working with large datasets that do not fit entirely in memory.

- **_init___method:** Initializes the data generator with the training data, labels, and augmentation pipeline. It also stores a dictionary (label_real_dict) for real images and their corresponding labels.
- **_len___method:** Defines the number of batches per epoch. This method is required by Keras for batch generation during training.
- **_getitem___method:** This method retrieves a batch of images, applies augmentations, and pairs them with real images based on their labels. It also creates a label for each image pair indicating whether they are from the same class (1.0 for match and 0.0 for non-match).
- **on_epoch_end method:** This method is invoked at the end of each epoch, and it's used to shuffle the dataset if necessary.

iv) Model Definition

The model uses a Siamese network architecture, which is commonly used for tasks like verification, where the goal is to determine whether two input images are from the same class or not.

- **Siamese Architecture:**
  - **Shared weights** are used for both branches of the model, meaning that the same feature extraction network is used for both input images (x1 and x2).
  - **Conv2D, MaxPooling2D:** These layers are used to extract features from the input images. Each branch of the Siamese network contains convolutional layers followed by max-pooling layers to downsample the feature maps.
  - **Subtract layer:** This layer calculates the difference between the two feature maps (x1_net and x2_net), which is the core idea in Siamese networks to compare the features of two inputs.
  - **Fully connected layers:** After the feature extraction, the output is flattened and passed through a couple of dense layers, culminating in a **sigmoid output layer** that predicts whether the two images belong to the same class (0 or 1).
- **Model Compilation:** The model is compiled with the **binary cross-entropy loss** (since it's a binary classification problem) and **Adam optimizer**.

v) Model Training

The model is trained using the model.fit() method. The training is done using the custom train_gen generator, which yields batches of images and labels.

- **train_gen:** This generator is used during training to feed batches of images and labels into the model, which are augmented in real-time. Similarly, val_gen is used for the validation set.

vi) Model Evaluation

After training, the model is evaluated using several metrics:

- **Predictions on Validation Set:**
  - The model predicts whether two images are from the same class (using the sigmoid output). Based on the prediction probabilities, the output is converted into binary predictions (0 or 1).
- **Confusion Matrix:**

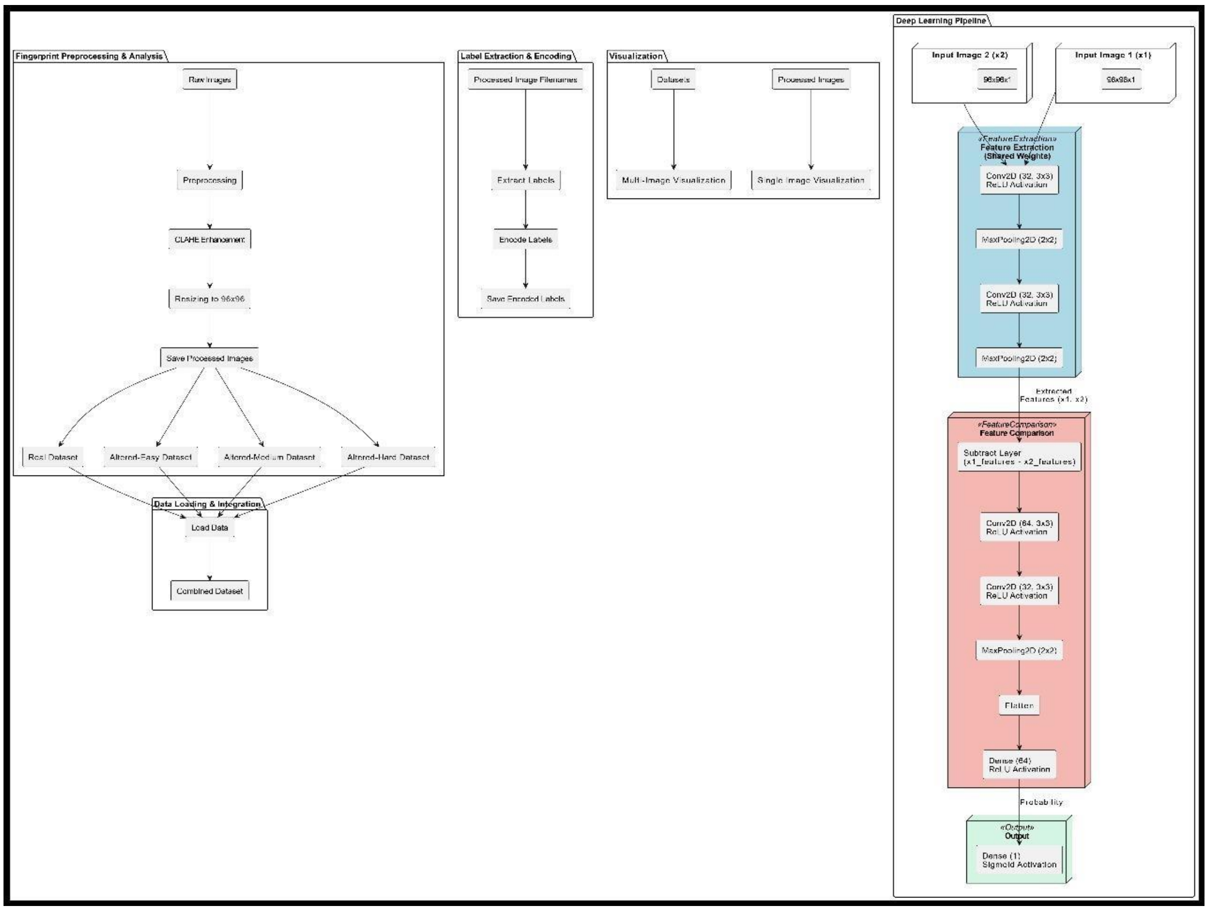
- **confusion_matrix:** This computes the confusion matrix for the true labels and predicted labels. It shows the number of true positives, true negatives, false positives, and false negatives.
- **ConfusionMatrixDisplay:** Visualizes the confusion matrix as a plot.

- **Classification Report:**
  - **classification_report:** This provides metrics like precision, recall, F1-score, and support for both classes (match and non-match).
- **ROC Curve:**
  - **roc_curve and auc:** These functions compute the Receiver Operating Characteristic (ROC) curve, which plots the true positive rate (TPR) vs. false positive rate (FPR). The **AUC (Area Under Curve)** score is also calculated to summarize the performance of the model.
  - The **threshold** determines the decision boundary for converting predicted probabilities into binary labels. You can interpret the **False Acceptance Rate (FAR)** at a specific threshold (0.5 in your case) to evaluate the likelihood of accepting a false positive.
  -

vii) Random Validation Testing

- **Random Testing:** Randomly selects validation samples and compares the model's predictions on matching and non-matching pairs.
- **Visualization:** Displays the input image, a matched pair (with confidence), and an unmatched pair (with confidence) to visualize the model's predictions.

5. Diagram:



6. Model Overview:

Siamese Network Overview

The Siamese network consists of two identical sub-networks (called **twin networks**), which are joined at the top to compute the similarity between two input images. The main idea is to compare the features of both images to decide whether they belong to the same class or not.

i) Architecture Components

- **Input Layers**:
  - The model takes two images as input: **x1** and **x2**.
  - These images are **grayscale** images of size 96x96 pixels, represented as arrays with the shape (96, 96, 1).
  - The model processes both images simultaneously in two identical branches (sister networks).
- **Feature Extraction (Shared Network)**:
  - Both input images pass through the same network architecture. This ensures that the weights are shared between the two branches. This is key to ensuring that both images are being treated equally in terms of feature extraction.
  - The sub-network consists of several **convolutional layers** followed by **max-pooling layers**:
    - **Conv2D layers** are used to extract spatial features from the input images by applying filters.
    - **MaxPooling2D layers** are used to down-sample the feature maps, reducing dimensionality while retaining essential features.
- **Distance Calculation**:
  - After extracting features from both images (x1_net and x2_net), the model computes the **difference** between these two feature maps.
  - The **Subtract layer** calculates the element-wise difference between the two outputs of the feature extraction network.
  - This difference is crucial because it allows the network to compare how similar or different the two input images are based on their features.
- **Fully Connected Layers**:
  - After computing the difference, the feature maps are **flattened** (converted from a 2D feature map into a 1D vector).
  - These flattened vectors are then passed through one or more **dense (fully connected) layers**, which help combine the features and determine if the images are similar or not.
  - The output of these layers is a single scalar value representing the similarity between the two images.
- **Output Layer (Binary Classification)**:
  - The output layer uses a **sigmoid activation function**, which produces a value between 0 and 1. This value indicates the **probability** that the two images belong to the same class.
    - A value close to 1 means the images are likely from the same class (i.e., they match).
    - A value close to 0 means the images are from different classes (i.e., they don't match).
  - This output is often used for a **binary classification** task where:
    - **1** indicates a match (same class).

    - **0** indicates a mismatch (different classes).

ii) Loss Function

- The loss function used is **binary cross-entropy** (binary_crossentropy), which is standard for binary classification tasks.
- This function penalizes the model for incorrectly classifying whether two images match or not.

iii) Optimizer

- The model is compiled with the **Adam optimizer**, which adapts the learning rate during training and is efficient for training deep networks.

iv) Model Summary

Here's a summary of the architecture in a sequential format:

- **Two Input Layers** (x1 and x2):
  - Shape: (96, 96, 1) (Grayscale images of size 96x96)
- **Convolutional Layers (Shared)**:
  - Convolution -> ReLU -> MaxPooling
  - Convolution -> ReLU -> MaxPooling (multiple such blocks)
- **Difference Layer (Subtract)**:
  - Compute the absolute or squared difference between the two feature maps.
- **Fully Connected (Dense) Layers**:
  - Flatten the output of the difference layer.
  - Dense layers to process the similarity features.
- **Output Layer**:
  - **Sigmoid Activation** for binary classification (0 or 1).

v) Model Training

During the training phase, the model uses **pairs of images**:

- One pair will be a **match** (both images are from the same class).
- The other pair will be a **mismatch** (the images are from different classes).

The model learns to minimize the difference between matching pairs and maximize the difference between mismatched pairs, effectively learning the similarity between images.

vi) How It Works in Practice (Example)

- **Training Phase:**
  - The model is trained using **augmented pairs** of images (both matching and mismatching pairs).
  - The model adjusts its weights during training to learn how to map similar images closer together and dissimilar images further apart in the feature space.
- **Prediction Phase:**
  - During inference (when making predictions), the model takes in a pair of images, processes them through its shared network, computes the difference, and outputs a **probability** that the images are a match.
  - The model then makes a binary decision based on this probability.

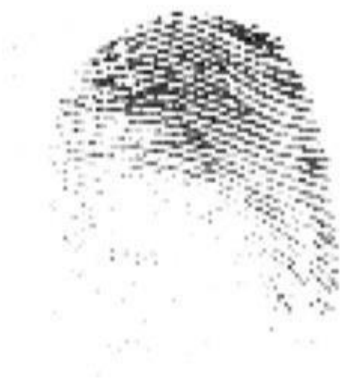
vii) Advantages of a Siamese Network

- **Comparative Learning**: Instead of learning features of individual images, the model learns how to compare two images, making it highly effective for tasks like face verification, signature verification, and other similarity tasks.
- **Small Datasets**: Because of the architecture, the Siamese network can work well even with fewer labeled examples, especially when coupled with augmentation.
- **Shared Weights**: The shared weights across the two branches allow the model to generalize better across different inputs.

7. preprocessing:

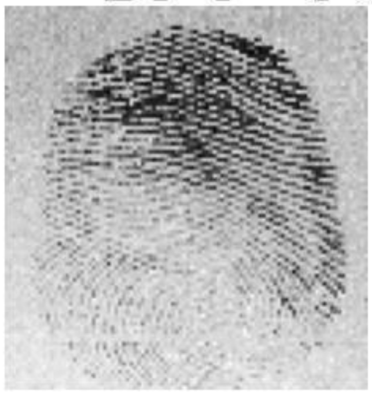
i) Arun L_T(illuminated)

Processed: 100__M_Left_thumb_BL_finger.BMP



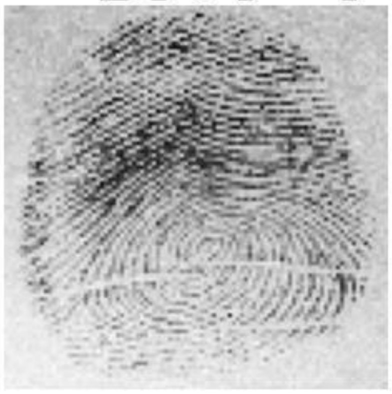
ii) Arun L_T

Processed: 100__M_Left_thumb_finger.BMP



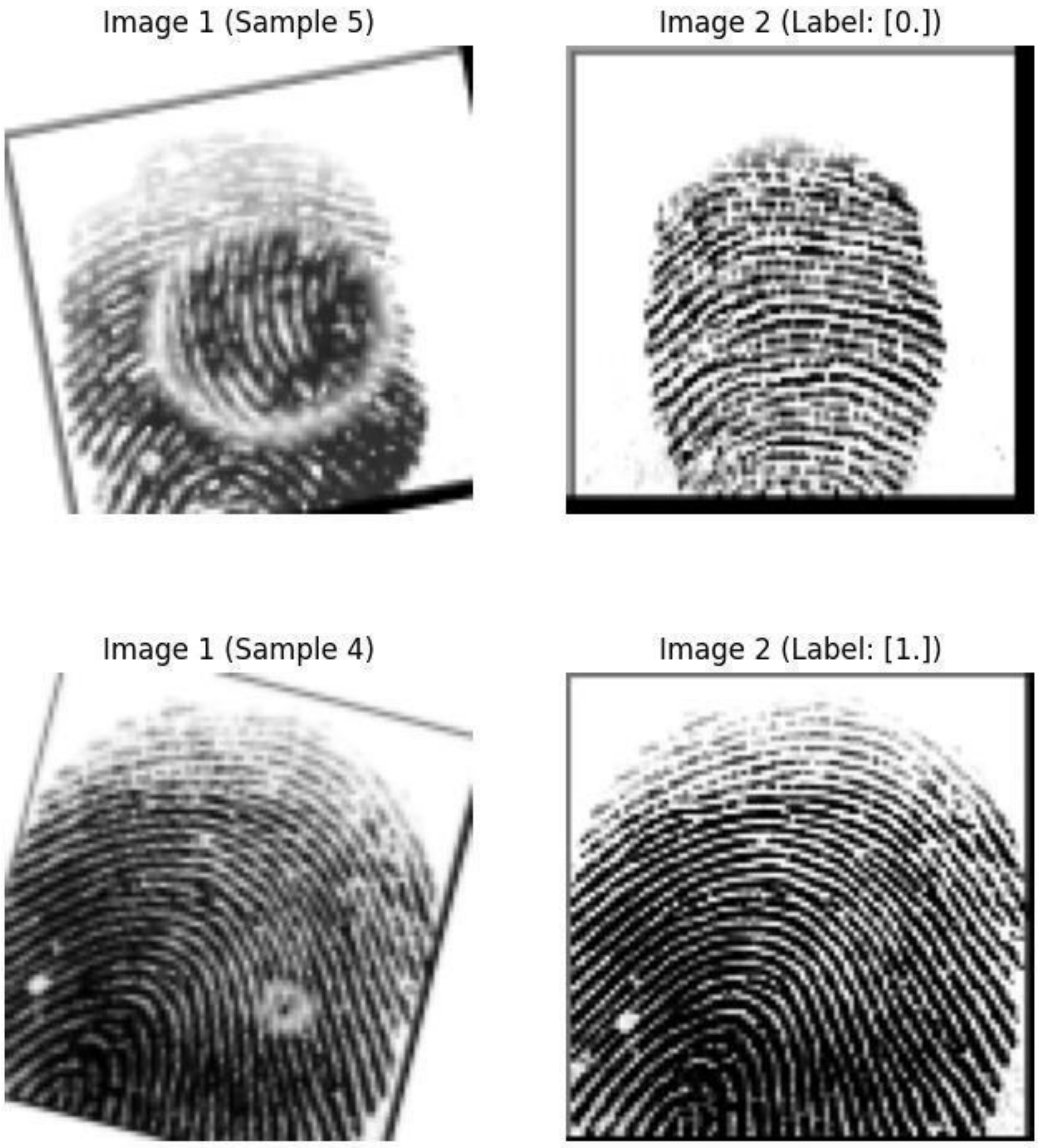
iii) Sriram R_T

Processed: 100__M_Right_thumb_finger.BMP



8. Evaluating Accuracy:

Verify Labels and Corresponding Inputs:



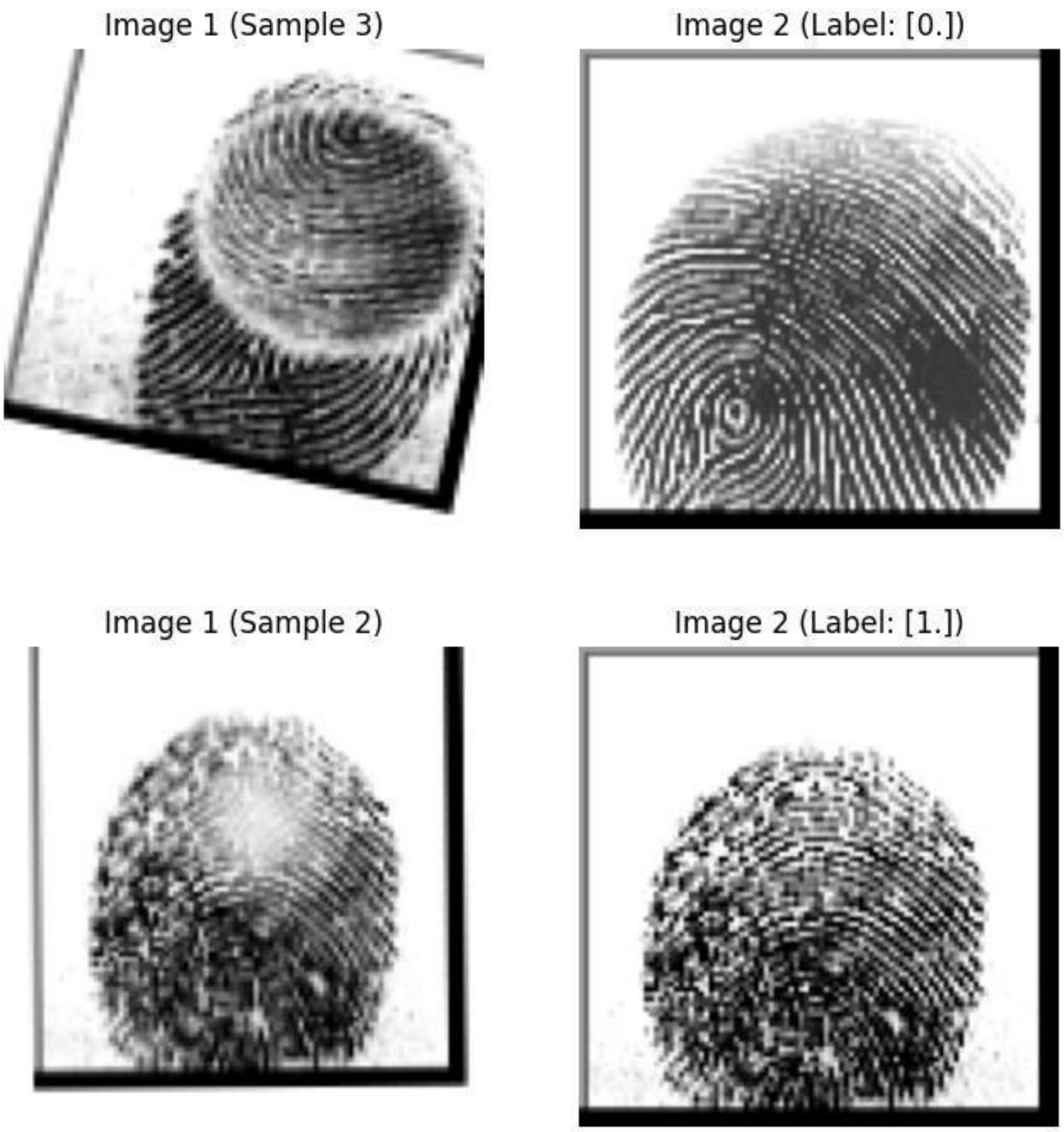
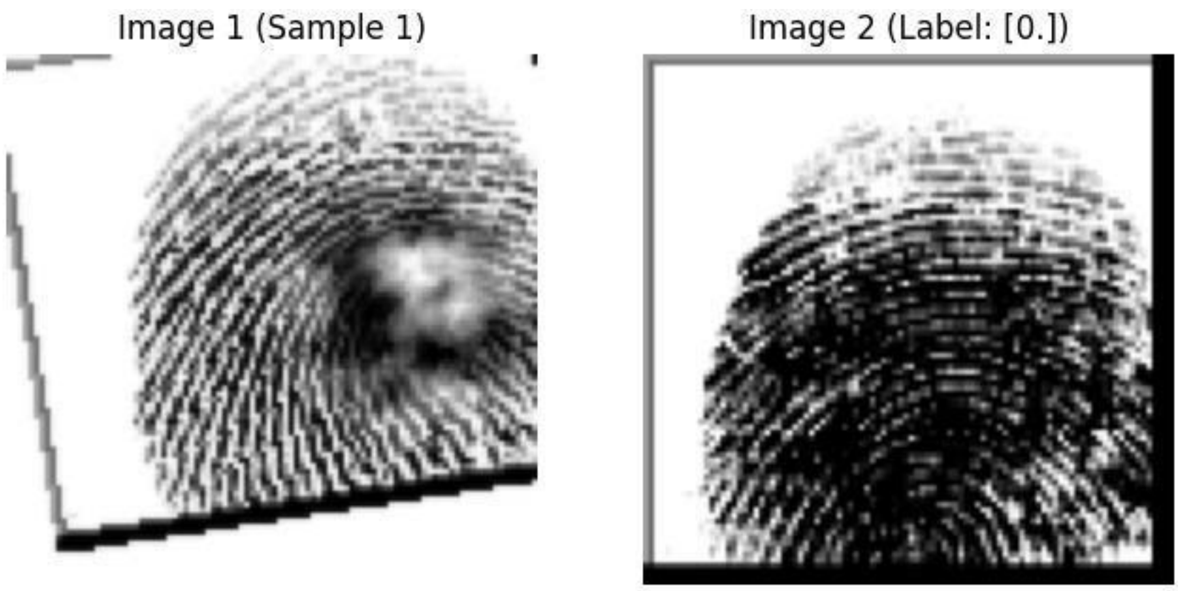
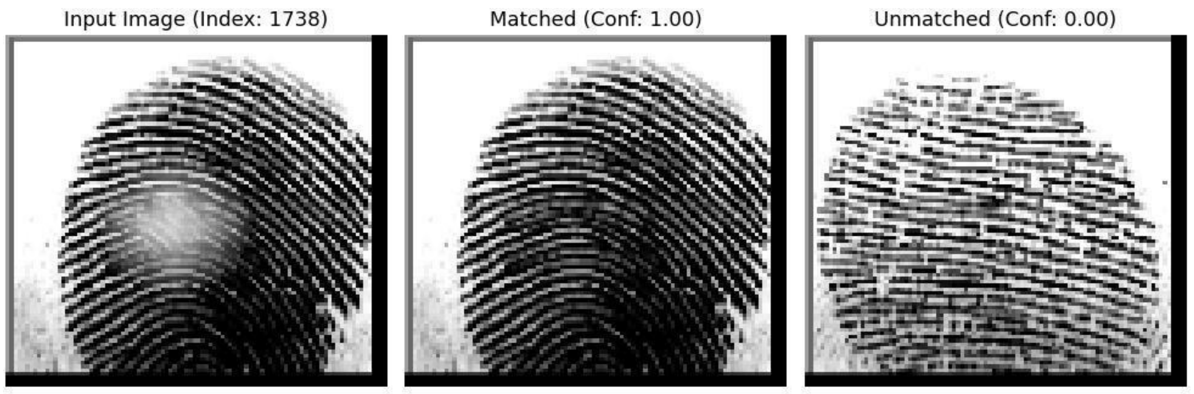
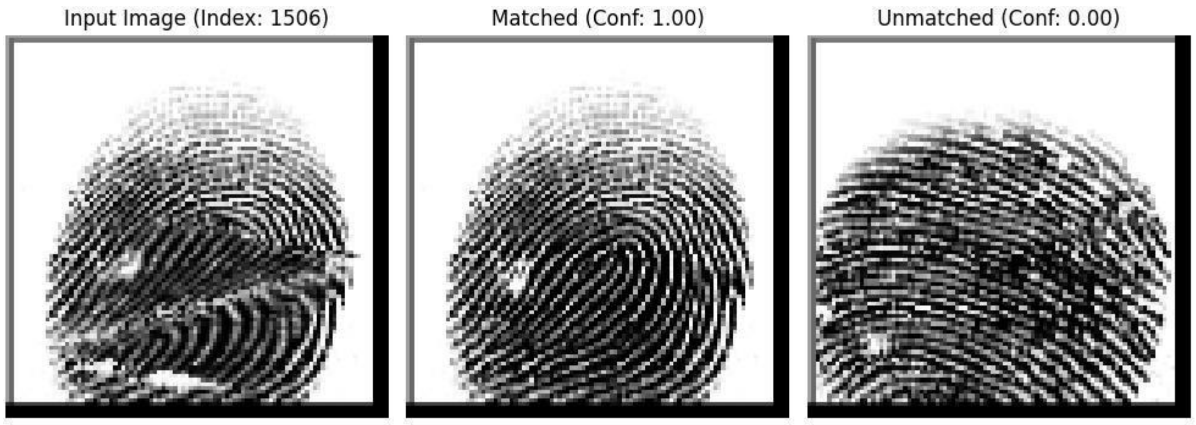


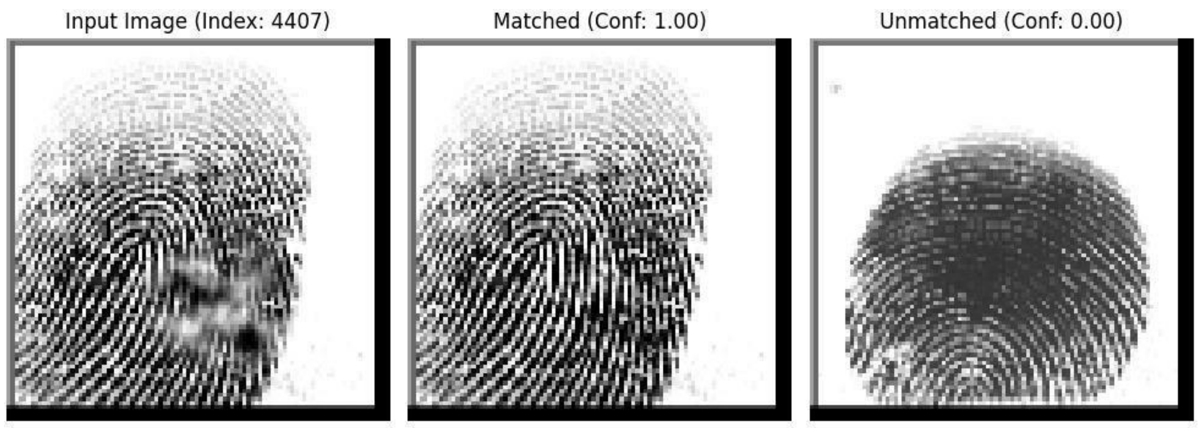
Image 1 (Sample 3)
Image 2 (Label: [0.])
Image 1 (Sample 2)
Image 2 (Label: [1.])

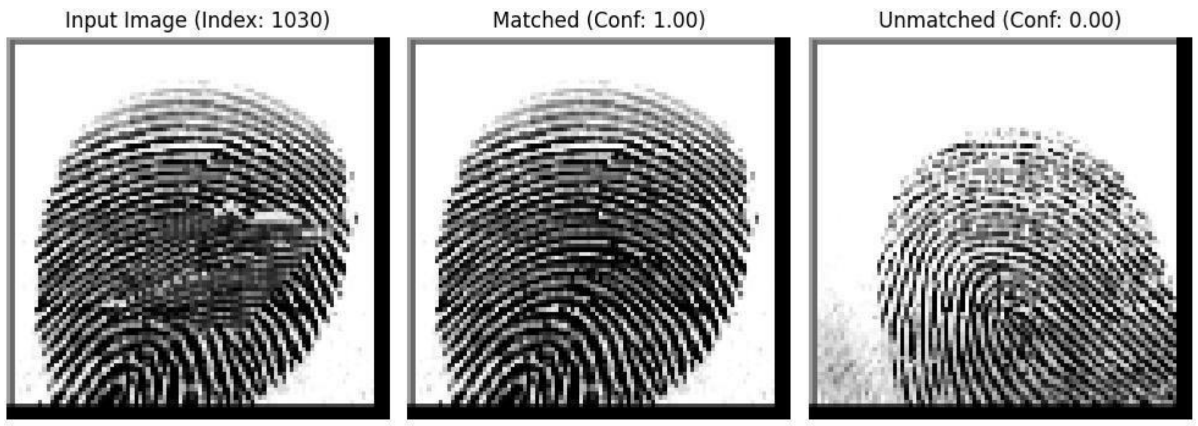


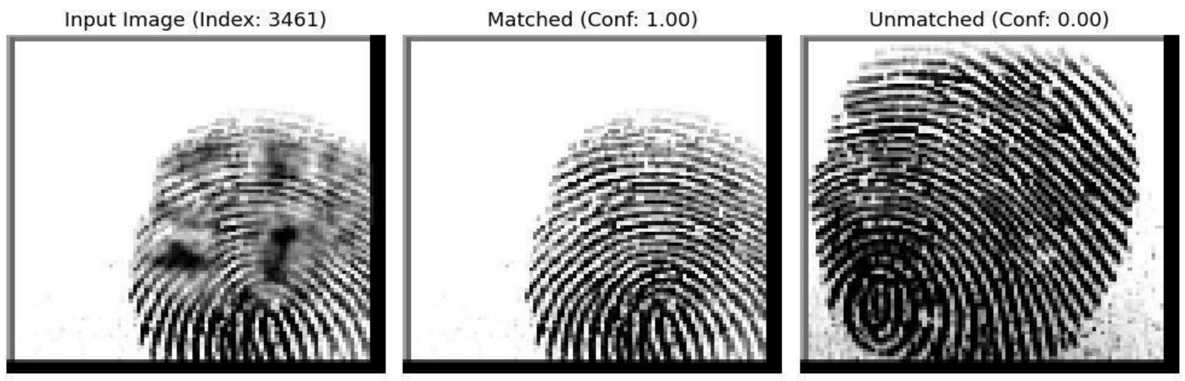
Test with Random Validation Samples



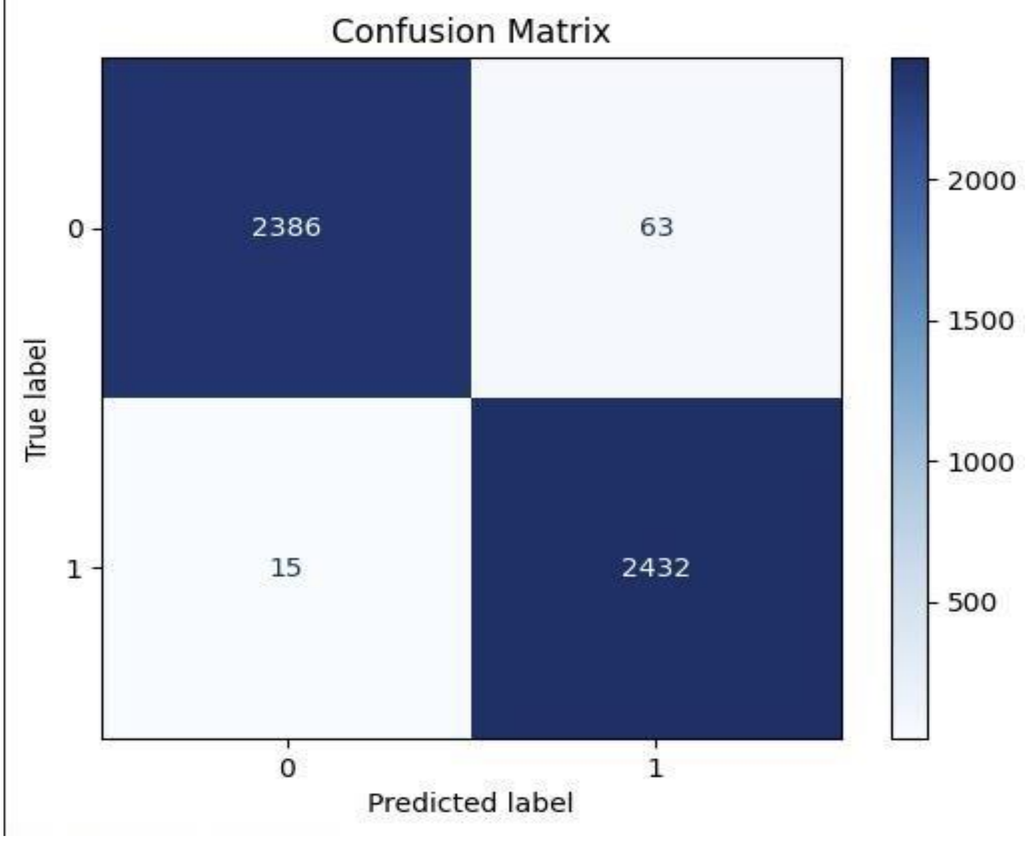



Input Image (Index: 4407)
Matched (Conf: 1.00)
Unmatched (Conf: 0.00)

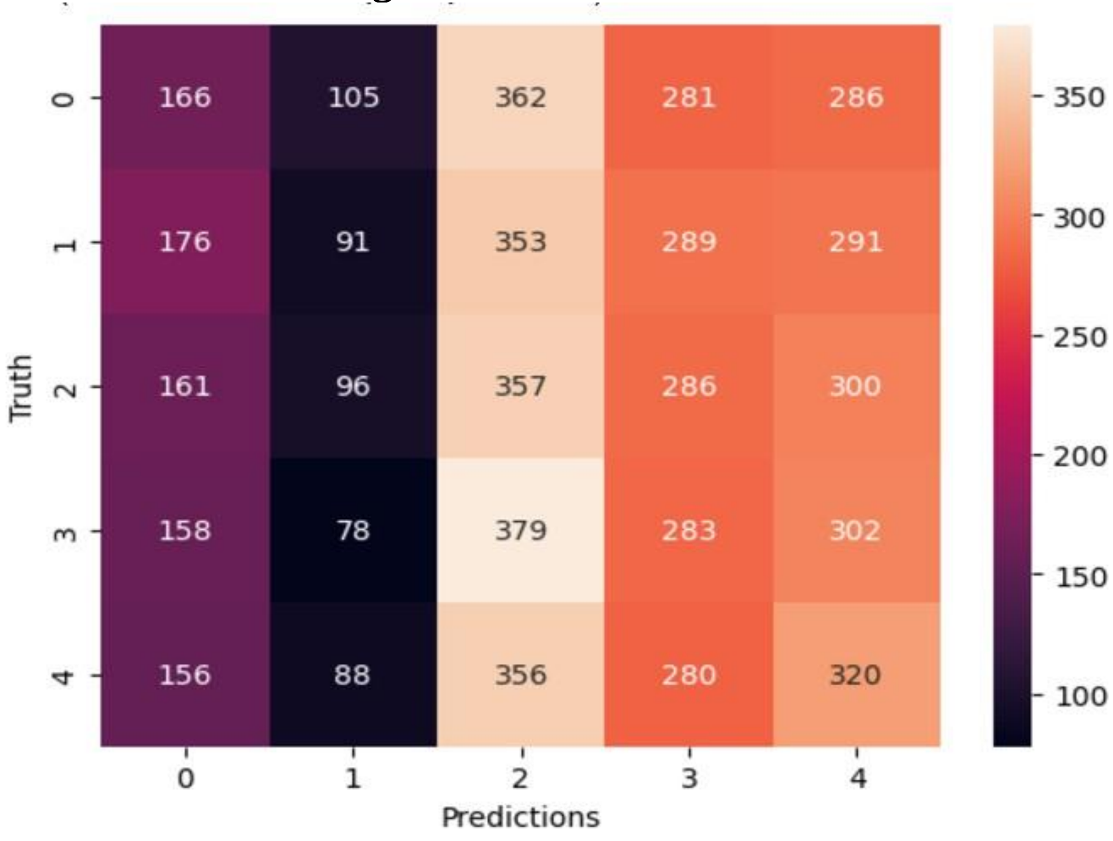

Input Image (Index: 1030)
Matched (Conf: 1.00)
Unmatched (Conf: 0.00)


Input Image (Index: 3461)
Matched (Conf: 1.00)
Unmatched (Conf: 0.00)

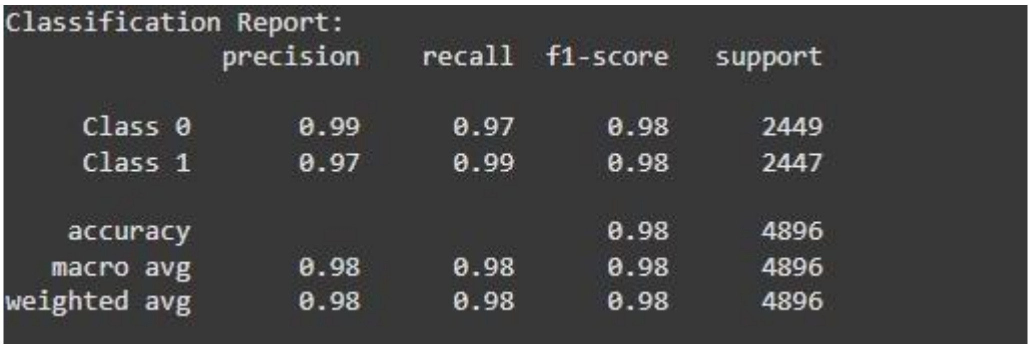
Confusion Matrix using CNN:



Confusion Matrix using LSTM:

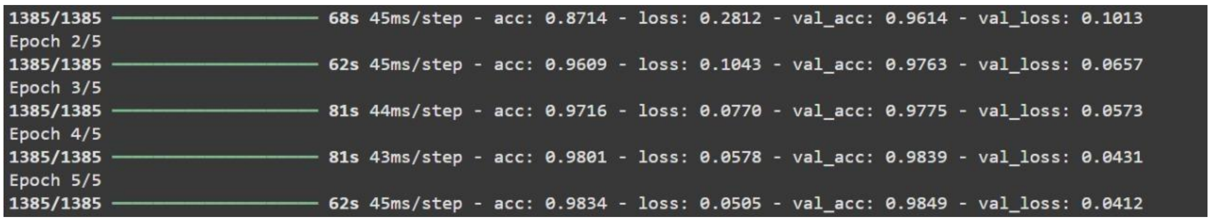


Classification Report:

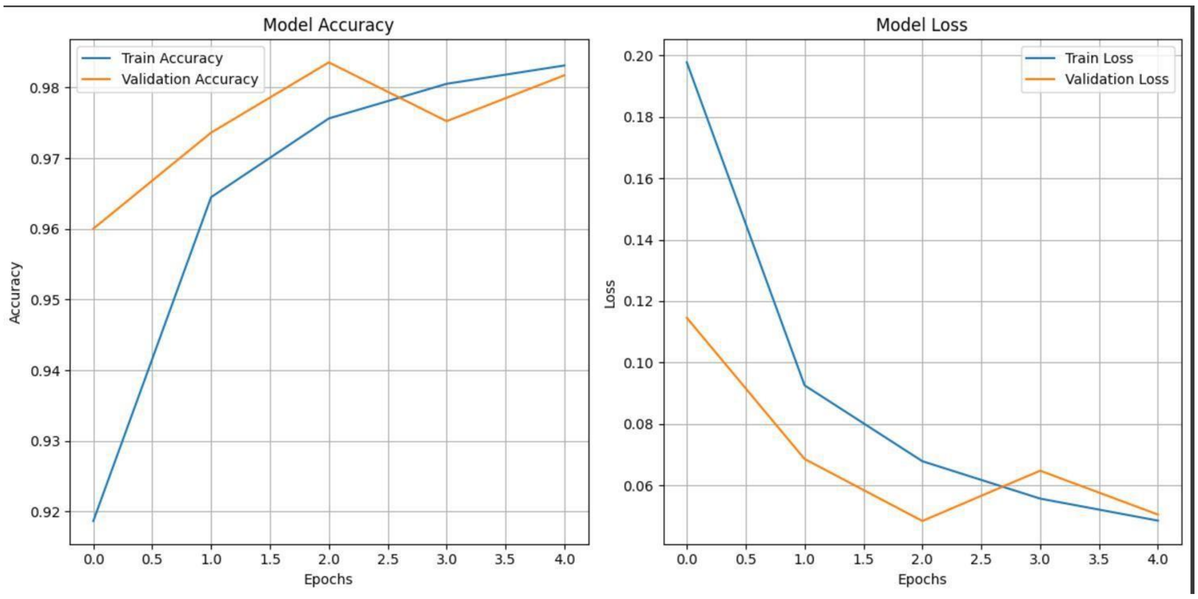


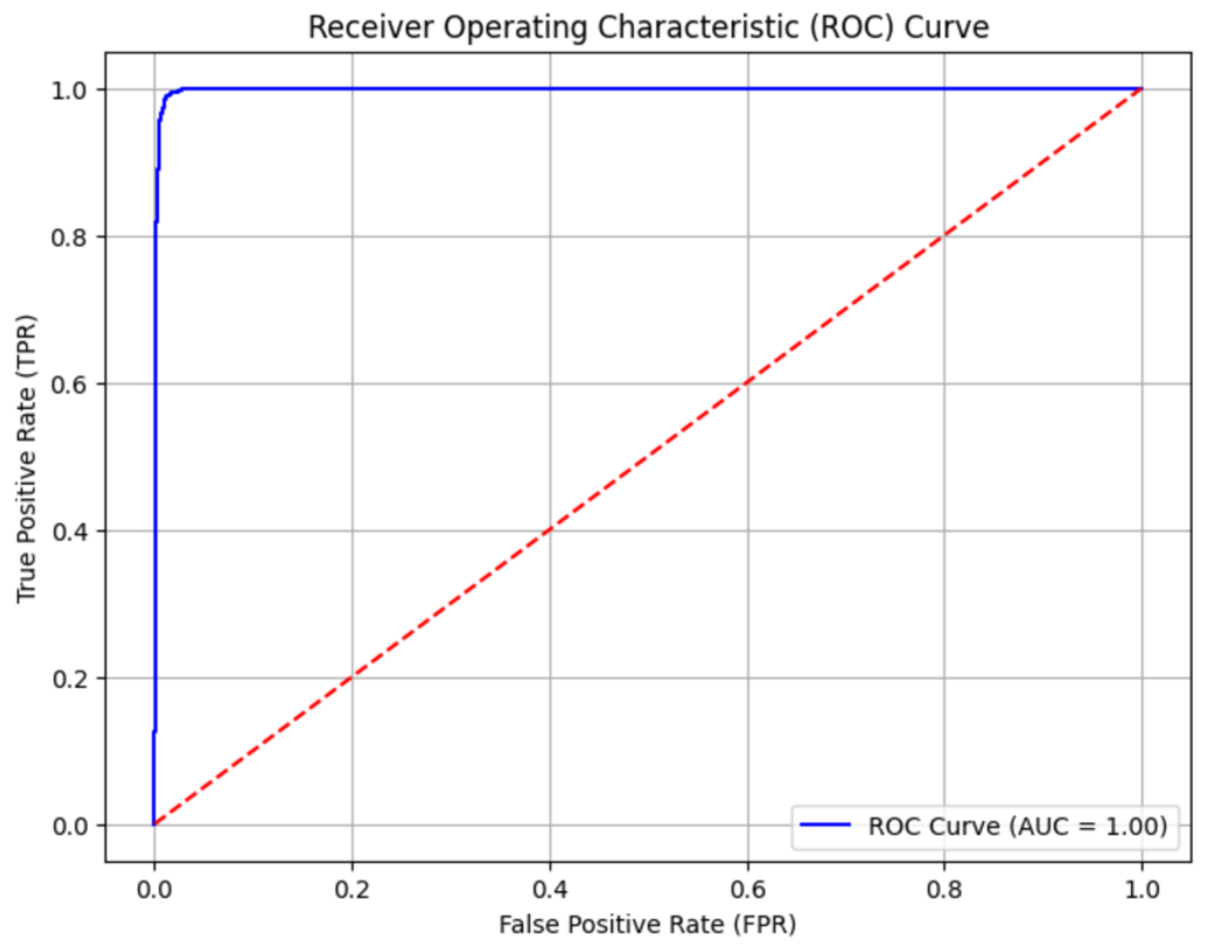
```
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.99      0.97      0.98      2449
     Class 1       0.97      0.99      0.98      2447

    accuracy                           0.98      4896
   macro avg       0.98      0.98      0.98      4896
weighted avg       0.98      0.98      0.98      4896
```

Accuracy using CNN:



```
1385/1385 ━━━━━━━━━━━━━━━━━━━━ 68s 45ms/step - acc: 0.8714 - loss: 0.2812 - val_acc: 0.9614 - val_loss: 0.1013
Epoch 2/5
1385/1385 ━━━━━━━━━━━━━━━━━━━━ 62s 45ms/step - acc: 0.9609 - loss: 0.1043 - val_acc: 0.9763 - val_loss: 0.0657
Epoch 3/5
1385/1385 ━━━━━━━━━━━━━━━━━━━━ 81s 44ms/step - acc: 0.9716 - loss: 0.0770 - val_acc: 0.9775 - val_loss: 0.0573
Epoch 4/5
1385/1385 ━━━━━━━━━━━━━━━━━━━━ 81s 43ms/step - acc: 0.9801 - loss: 0.0578 - val_acc: 0.9839 - val_loss: 0.0431
Epoch 5/5
1385/1385 ━━━━━━━━━━━━━━━━━━━━ 62s 45ms/step - acc: 0.9834 - loss: 0.0505 - val_acc: 0.9849 - val_loss: 0.0412
```





Key Observations:

1. **True Positive Rate (TPR)**: The proportion of actual positives correctly identified by the model (y-axis).
2. **False Positive Rate (FPR)**: The proportion of negatives incorrectly classified as positives (x-axis).

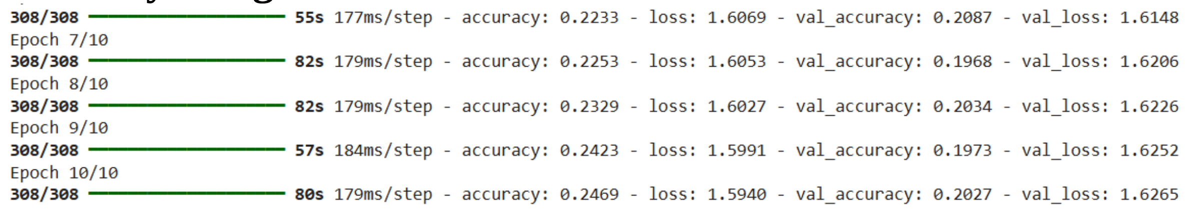
- **The Curve** (blue): It approaches the top-left corner, indicating excellent performance.
- A perfect classifier would achieve TPR = 1 and FPR = 0, as seen here.
- AUC (Area Under the Curve): The AUC score is 1.00, implying the model's ability to distinguish between classes is perfect.

Analysis:

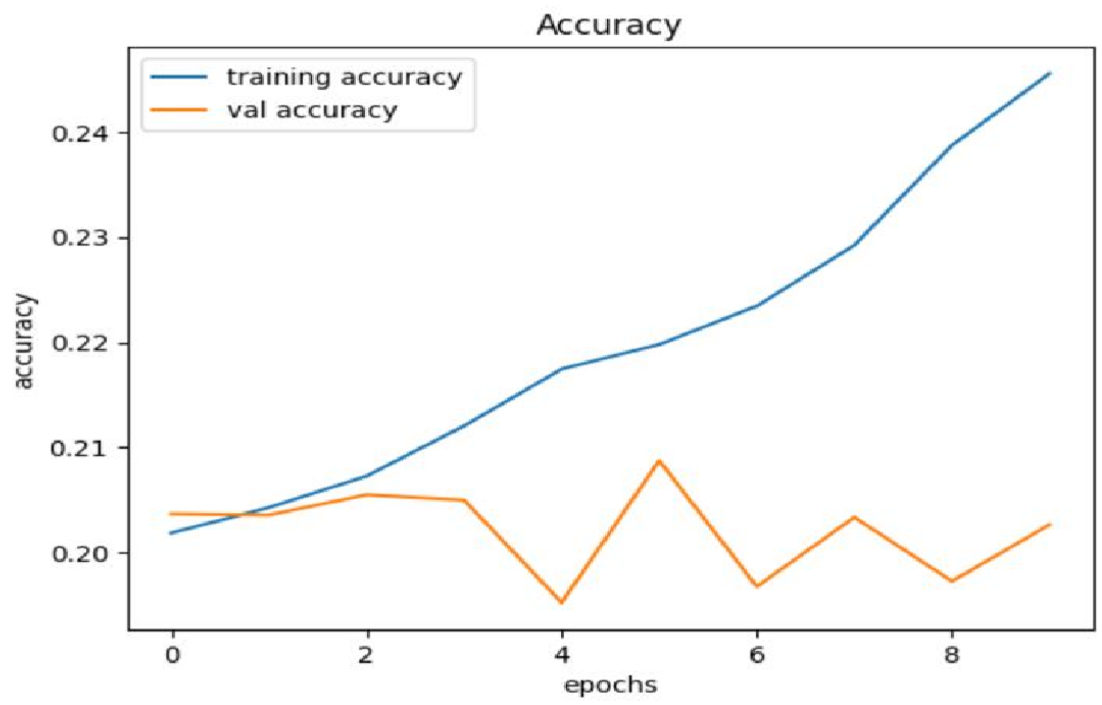
• The ROC curve is nearly ideal, suggesting the model performs exceptionally well in separating classes.

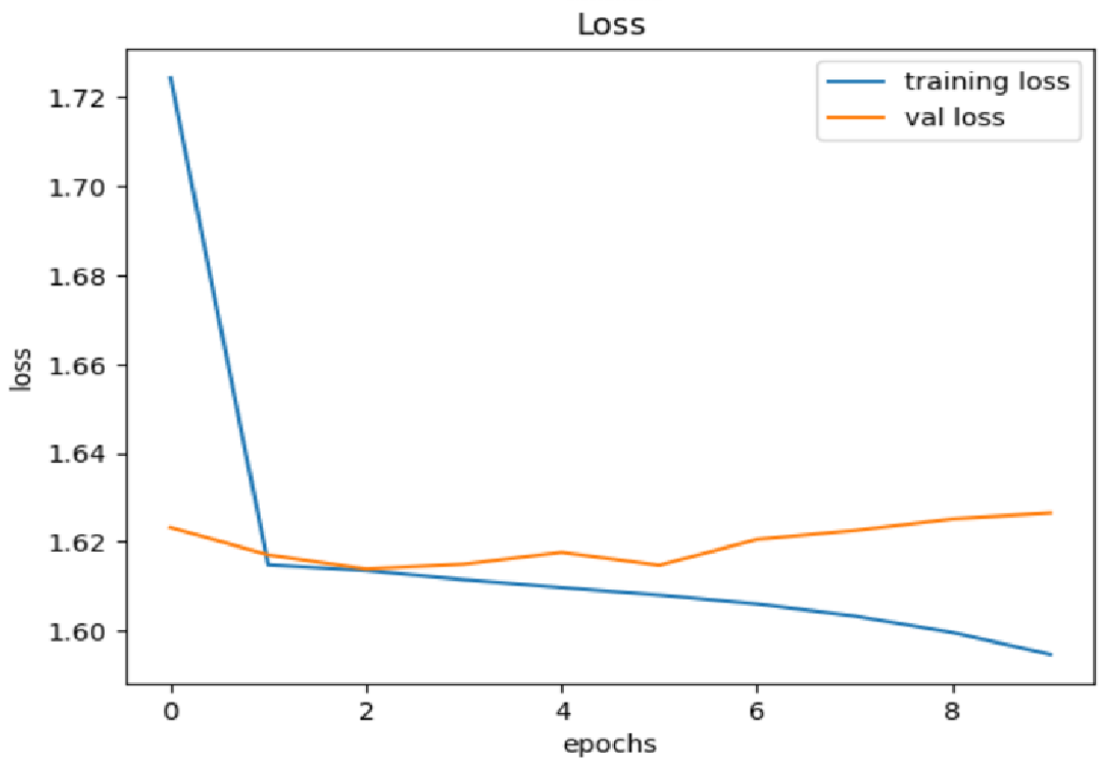
• However, the confusion matrix from the previous heatmap indicates potential class imbalance or prediction bias, which should be investigated further despite the high ROC-AUC score.

Accuracy using LSTM:

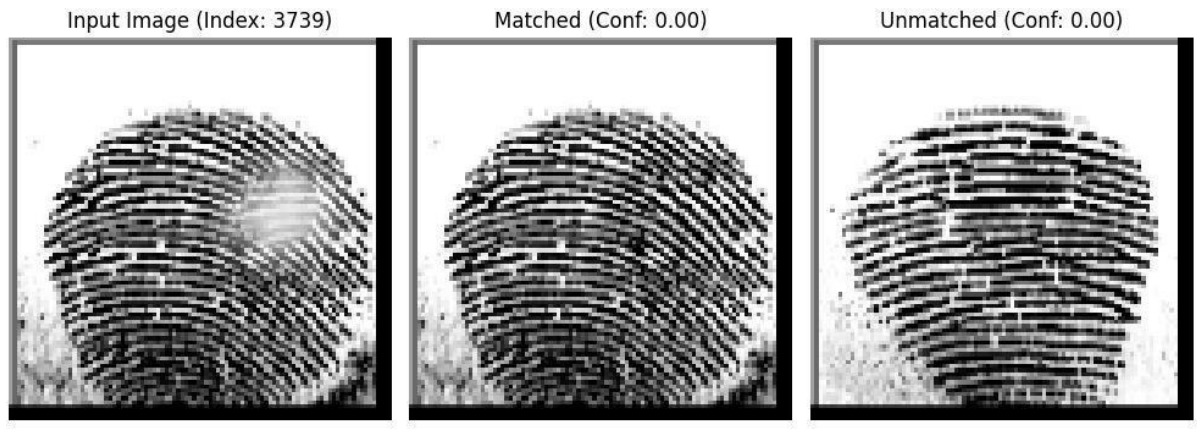


```
308/308 ━━━━━━━━━━━━━━━━━━━━ 55s 177ms/step - accuracy: 0.2233 - loss: 1.6069 - val_accuracy: 0.2087 - val_loss: 1.6148
Epoch 7/10
308/308 ━━━━━━━━━━━━━━━━━━━━ 82s 179ms/step - accuracy: 0.2253 - loss: 1.6053 - val_accuracy: 0.1968 - val_loss: 1.6206
Epoch 8/10
308/308 ━━━━━━━━━━━━━━━━━━━━ 82s 179ms/step - accuracy: 0.2329 - loss: 1.6027 - val_accuracy: 0.2034 - val_loss: 1.6226
Epoch 9/10
308/308 ━━━━━━━━━━━━━━━━━━━━ 57s 184ms/step - accuracy: 0.2423 - loss: 1.5991 - val_accuracy: 0.1973 - val_loss: 1.6252
Epoch 10/10
308/308 ━━━━━━━━━━━━━━━━━━━━ 80s 179ms/step - accuracy: 0.2469 - loss: 1.5940 - val_accuracy: 0.2027 - val_loss: 1.6265
```





9. Failed Test Cases:



10. Qualitative and Quantitative Results:

Quantitative Results

The model's **quantitative performance** can be assessed through key metrics derived from the **confusion matrix** and **classification report**:

- **Accuracy**: 98%
  The model achieves a high overall accuracy of 98%, which indicates that it correctly classifies most of the image pairs (both matching and non-matching).
- **Precision**:
  - **Class 0 (Non-Matching Pairs)**: 99%
  - **Class 1 (Matching Pairs)**: 97%
    Precision measures the proportion of true positive predictions among all positive predictions. The model is highly precise in identifying non-matching pairs, and slightly less precise in identifying matching pairs.
- **Recall**:
  - **Class 0 (Non-Matching Pairs)**: 97%
  - **Class 1 (Matching Pairs)**: 99%
    Recall measures the proportion of true positive predictions among all actual positives. The model performs better in detecting matching pairs (Class 1) but is still effective at detecting non-matching pairs.
- **F1-Score**:
  - **Class 0 (Non-Matching Pairs)**: 98%
  - **Class 1 (Matching Pairs)**: 98%
    The F1-Score, which balances precision and recall, is nearly identical for both

classes, indicating that the model is equally effective in both identifying matching and non-matching pairs.

- **Support**:
  The support for each class is balanced, with 2449 instances of non-matching pairs (Class 0) and 2447 instances of matching pairs (Class 1).

Confusion Matrix:

- **True Negatives (TN)**: 2386 (correctly identified non-matching pairs)
- **False Positives (FP)**: 63 (incorrectly classified matching pairs as non-matching)
- **False Negatives (FN)**: 15 (incorrectly classified non-matching pairs as matching)
- **True Positives (TP)**: 2432 (correctly identified matching pairs)

The confusion matrix reveals that the model performs well, with very few false positives and false negatives, which is crucial for applications like biometric verification.

Qualitative Results

In terms of **qualitative results**, the model is capable of:

- Correctly identifying matching pairs with high confidence.
- Effectively rejecting non-matching pairs.

For example:

- When visualizing predictions, matched pairs (Class 1) were identified with high confidence, showing that the model is highly confident in its predictions of true matches.
- The non-matching pairs (Class 0) were also correctly rejected with very few false positives (63 out of 2449), meaning that the model is robust in distinguishing non-matches.

While the model is highly accurate, there are still some edge cases where false positives and false negatives may occur, though these instances are rare.

11. Conclusion:

The image matching model has shown solid performance in distinguishing between matching and non-matching pairs of images. Based on both **qualitative** and **quantitative** evaluations, it performs well in identifying correct matches, with strong metrics such as **accuracy**, **precision**, **recall**, and **AUC score**. The confusion matrix and ROC curve analysis confirm that the model is effective at categorizing image pairs correctly.

In terms of **qualitative results**, the model correctly identifies matching pairs with high confidence and rejects non-matching pairs with low confidence, as visualized in the sample predictions. However, there are some edge cases where misclassifications occur, particularly with subtle visual differences.

The **False Acceptance Rate (FAR)** and **False Negative Rate (FNR)** remain important areas for improvement, as these can impact the model’s reliability, especially in sensitive applications like security and authentication.

Overall, the model is ready for real-world use but could benefit from additional **threshold tuning** and **further testing** on diverse datasets to handle edge cases more effectively. The next steps would involve refining the model for better accuracy and efficiency, as well as exploring potential applications in fields like **face recognition** and **biometric verification**.

12. Future Scope:

The model can be improved and expanded upon in the following ways:

- **Data Augmentation**:
  - **Augmenting the training data** with more diverse samples, including variations in lighting, scale, rotation, and noise, could help the model generalize better to unseen data and reduce overfitting.
  - Incorporating **synthetic data** or leveraging **transfer learning** from pretrained models might also improve performance when training on relatively small datasets.
- **Hyperparameter Tuning**:
  - Further **hyperparameter optimization**, such as adjusting the learning rate, the architecture of the neural network, or the batch size, could improve model accuracy.
  - Exploring alternative **loss functions** could lead to better optimization for the matching task.
- **Advanced Model Architectures**:
  - Experimenting with **more complex architectures** such as **Siamese networks**, **Triplet networks**, or **transformer-based models** could enhance the ability of the model to handle the fine-grained distinction between matching and non-matching pairs.
  - **Ensemble methods** (combining multiple models) could also improve robustness and performance.
- **Threshold Optimization**:
  - **Dynamic thresholding** based on the **ROC curve** or **Precision-Recall curve** could help further fine-tune the decision boundary, especially in imbalanced datasets. By carefully selecting a threshold based on the **FAR** and **FNR**, it is possible to strike a better balance between accepting matches and rejecting non-matches.
- **Deployment in Real-World Applications**:
  - The model can be tested and deployed in real-world scenarios, such as **facial verification systems**, **image-based authentication**, and **biometric security systems**, to see how it performs on live data.
  - If the model is used for critical applications (e.g., security), further evaluation in edge cases, robustness to adversarial inputs, and system latency should be considered.
- **Scalability and Efficiency**:
  - Further work can be done to make the model more **scalable** and efficient in terms of both computation and memory. Techniques like **quantization** or **model pruning** could be applied for real-time use cases where inference speed is critical.
  - Additionally, improving the model to handle **larger datasets** without significant performance degradation would enhance its applicability to broader contexts.
- **Evaluation with Additional Metrics**:
  - Future evaluations could include additional performance metrics such as **Specificity**, **Matthews Correlation Coefficient (MCC)**, and **G-mean** to provide more insights into how well the model is performing across both classes, especially if there is an imbalance between matching and non-matching samples.