

SIGNATURE VERIFICATION SYSTEM



A DESIGN PROJECT REPORT

Submitted by

SHOBICA S

THENNARASI M

UVASHINI NEKA B S

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

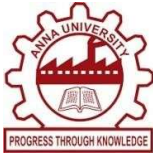
COMPUTER SCIENCE AND ENGINEERING

K RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai, Approved by AICTE, New Delhi)

Samayapuram – 621 112

DECEMBER, 2024



SIGNATURE VERIFICATION SYSTEM



A DESIGN PROJECT REPORT

Submitted by

SHOBIKA (811722104146)

THENNARASI M (811722104168)

UVASHINI NEKA B S (811722104172)

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

K RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai, Approved by AICTE, New Delhi)

Samayapuram – 621 112

DECEMBER, 2024

K RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled “**SIGNATURE VERIFICATION SYSTEM**” is bonafide work of **SHOBICA S (811722104146), THENNARASI M (811722104168), UVASHINI NEKA B S (811722104172)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. A Delphin Carolina Rani, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

Professor

Department of CSE

K Ramakrishnan College of Technology

(Autonomous)

Samayapuram – 621 112

SIGNATURE

Mrs. R Sathya, M.E.,(Ph.D).,

SUPERVISOR

Assistant Professor

Department of CSE

K Ramakrishnan College of Technology

(Autonomous)

Samayapuram – 621 112

Submitted for the viva-voice examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We jointly declare that the project report on “**SIGNATURE VERIFICATION SYSTEM**” is the result of original work done by us and best of our knowledge, similar work has not been submitted to “**ANNA UNIVERSITY CHENNAI**” for the requirement of Degree of Bachelor Of Engineering. This project report is submitted on the partial fulfilment of the requirement of the award of Degree of Bachelor of Engineering.

Signature

SHOBICA S

THENNARASI M

UVASHINI NEKA B S

Place: Samayapuram

Date:

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and indebtedness to our institution “**K RAMAKRISHNAN COLLEGE OF TECHNOLOGY**”, for providing us with the opportunity to do this project.

We are glad to credit honorable chairman **Dr. K RAMAKRISHNAN, B.E.**, for having provided for the facilities during the course of our study in college.

We would like to express our sincere thanks to our beloved Executive Director **Dr. S KUPPUSAMY, MBA, Ph.D.**, for forwarding our project and offering adequate duration to complete it.

We would like to thank **Dr. N VASUDEVAN, M.Tech., Ph.D.**, Principal, who gave opportunity to frame the project with full satisfaction.

We whole heartily thank **Dr. A DELPHIN CAROLINA RANI, M.E., Ph.D.**, Head of the Department, **COMPUTER SCIENCE AND ENGINEERING** for providing her support to pursue this project.

We express our deep and sincere gratitude and thanks to our project guide **Mrs. R SATHYA, M.E.,(Ph.D.)**, Department of **COMPUTER SCIENCE AND ENGINEERING**, for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

We render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course. We wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

Signature verification systems play a vital role in ensuring the authenticity and security of handwritten signatures in various applications, such as banking, legal documentation, and identity verification. These systems analyze and compare the unique characteristics of a person's signature, such as shape, size, and stroke dynamics, to detect forgery or unauthorized use. Traditional manual verification processes are time-consuming and prone to errors, necessitating the adoption of automated solutions to enhance accuracy and reliability. Modern signature verification systems leverage advancements in machine learning and deep learning technologies to achieve superior performance. By utilizing algorithms such as convolutional neural networks (CNNs), these systems can extract intricate features from both static and dynamic signatures. Static verification focuses on the image of the signature, while dynamic verification analyzes real-time data such as pressure, velocity, and timing. These techniques enable the system to distinguish genuine signatures from forgeries, even in cases of skilled imitation. The proposed signature verification system offers a scalable and efficient solution by integrating data preprocessing, feature extraction, and classification modules.

TABLE OF CONTENTS

| CHAPTER | TITLE | PAGE NO |
|----------|---|-----------|
| | ABSTRACT | v |
| | LIST OF FIGURES | ix |
| | LIST OF ABBREVIATIONS | x |
| 1 | INTRODUCTION | 1 |
| | 1.1 Background | 1 |
| | 1.2 Overview | 2 |
| | 1.3 Problem Statement | 3 |
| | 1.3.1 Challenges | 3 |
| | 1.4 Objective | 3 |
| | 1.5 Implication | 4 |
| 2 | LITERATURE SURVEY | 5 |
| 3 | SYSTEM ANALYSIS | 7 |
| | 3.1 Existing System | 7 |
| | 3.1.1 Traditional Methods | 7 |
| | 3.1.2 Advanced Techniques | 7 |
| | 3.2 Proposed System | 8 |
| | 3.3 Block Diagram for Proposed System | 9 |
| | 3.4 Flowchart | 10 |
| | 3.5 Process Cycle | 11 |
| | 3.6 Activity Diagram | 11 |
| 4 | MODULES | 13 |
| | 4.1 Module Description | 13 |
| | 4.1.1 Data Preprocessing Module | 13 |
| | 4.1.1.1 Core Preprocessing Techniques | 13 |
| | 4.1.1.2 Addressing Challenges | 14 |
| | 4.1.1.3 Implementation Considerations | 14 |
| | 4.1.1.4 Advanced Preprocessing Techniques | 15 |

| | | |
|----------|--|-----------|
| 4.1.1.5 | Challenges and Considerations | 15 |
| 4.1.1.6 | The Role of Preprocessing in Signature verification | 16 |
| 4.1.1.7 | Future Directions | 16 |
| 4.2 | Feature Extraction Module and Feature Selection Module | 17 |
| 4.3 | Classification and Verification Module | 18 |
| 4.3.1 | Classification Module | 18 |
| 4.3.2 | Verification Module | 19 |
| 4.4 | Post-processing Module | 20 |
| 4.5 | Template Management Module | 23 |
| 4.5.1 | Reference Signature Storage | 23 |
| 4.5.2 | Reference Signature Quality Control | 24 |
| 4.5.3 | Reference Signature Update | 24 |
| 4.5.4 | Security Considerations | 24 |
| 5 | SYSTEM SPECIFICATION | 25 |
| 5.1 | Software Requirement | 25 |
| 5.2 | Hardware Requirement | 25 |
| 5.1.1 | Visual Studio Code | 26 |
| 5.1.2 | Tensorflow | 27 |
| 5.1.3 | Matplotlib | 29 |
| 5.1.4 | Pandas | 30 |
| 5.1.5 | Keras | 31 |
| 5.1.6 | NumPy | 32 |
| 6 | METHODOLOGY | 34 |
| 6.1 | Image Processing | 34 |
| 6.1.1 | RGB to Grayscale Conversion | 34 |
| 6.1.2 | Grayscale to Binary Conversion | 35 |
| 6.1.3 | Noise Reduction and Cropping | 35 |
| 6.2 | Feature Extraction | 36 |
| 6.2.1 | Pixel Density Ratio Calculation | 36 |
| 6.2.2 | Centroid Determination | 37 |

| | |
|---|-----------|
| 6.2.3 Eccentricity and Solidity Computation | 37 |
| 6.2.4 Skewness and Kurtosis Estimation | 37 |
| 6.3 Data Preparation | 38 |
| 6.3.1 Feature Representation in CSV Format | 38 |
| 6.3.2 Training and Testing Dataset Generation | 38 |
| 6.4 Neural Network Design | 39 |
| 6.4.1 Model Architecture Definition | 39 |
| 6.4.2 Weight and Bias Initialization | 39 |
| 6.5 Model Training | 40 |
| 6.5.1 Loss Function and Optimization | 40 |
| 6.5.2 Training Epochs and Early Stopping Criteria | 40 |
| 6.6 Performance Evaluation | 41 |
| 6.6.1 Training and Testing Accuracy Calculation | 41 |
| 6.6.2 Genuine vs Forged Classification | 41 |
| 6.7 Automation and Scalability | 41 |
| 6.7.1 CSV Creation for Multiple Users | 42 |
| 6.7.2 Modular Functionality for Testing Individual Signatures | 42 |
| 7 CONCLUSION AND FUTURE ENHANCEMENT | 43 |
| 7.1 Conclusion | 43 |
| 7.2 Future Enhancement | 44 |
| APPENDIX-1 | 46 |
| APPENDIX-2 | 49 |
| REFERENCES | 51 |

LIST OF FIGURES

| FIGURE NO | FIGURE NAME | PAGE NO |
|----------------------|--|--------------------|
| 3.1 | Block Diagram | 9 |
| 3.2 | Flow of Control | 10 |
| 3.3 | Life Cycle of the Process | 11 |
| 3.4 | Activity Diagram | 11 |
| 3.5 | Action Sequence Structure of signature verification system | 12 |
| 3.6 | Module in signature verification system | 16 |
| 3.7 | Diagram representing feature extraction | 18 |
| 3.8 | Diagram representing classification module | 20 |
| 3.9 | Diagram representing phases | 23 |

LIST OF ABBREVIATIONS

| ABBREVIATION | FULL FORM |
|--------------|-------------------------------------|
| CSV | Comma-Separated Values |
| CNN | Convolutional Neural Network |
| DTW | Dynamic Time Warping |
| XAI | Explainable artificial intelligence |
| GANs | Generative Adversarial Networks |
| HMMs | Hidden Markov Models |
| k-NN | K-Nearest Neighbours |
| LSTM | Long Short-Term Memory |
| ML | Machine Learning |
| MSE | Mean Squared Error |
| NLP | Natural Language Processing |
| RNNs | Recurrent Neural Networks RNNs |
| RGBD | Red Green Blue Depth |
| RGB-IR | Red Green Blue Infrared Rays |
| SVMs | Support Vector Machines |
| VS Code | Visual Studio Code |

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Signature verification is the process of using a digital signature algorithm and a public key to verify a digital signature on data. It is a form of identity verification. Banks, intelligence services, and other prestigious institutions employ signature verification to confirm a person's identification. In bank branches and other branch capture, signature comparison is frequently employed. The signature verification software compares a direct signature or a picture of a signature to the recorded signature image. The signature serves as the authority for all legal transactions. Thus, the necessity for signature verification grows. It is distinct for the handwritten signatures to individuals and that cannot be duplicated. In addition to being a well-liked area of research in the fields of pattern recognition and image processing, signature verification also plays a significant role in numerous applications, including access control, security, and privacy. The process of certifying someone based on their handwritten signature is known as signature verification. Systems for verifying signatures come in two varieties.

Online Signature Verification System, which records details like pressure, speed, direction, etc. using an electronic device like a tablet. Offline Signature Verification System, in which the signature is written offline and verified using the image of the signature that has previously been stored. Two distinct methods can be used to verify offline signatures. One involves building models of real and fake signatures for each writer in a process known as writer dependent signature verification. Next, a writer's test signature sample is contrasted with its own training sample. This method's downside is that each new writer must have a model created in order to be confirmed. Forensic professionals utilize the second method, known as writer independent signature verification and accuracy is 84% .

1.2 OVERVIEW

A signature verification system is a technological solution designed to authenticate a person's identity by analyzing their handwritten signature. It's a crucial tool in various applications, including banking, legal documents, and secure access control. This system operates by capturing a signature, either online or offline, and then subjecting it to a rigorous analysis process.

The core of signature verification lies in feature extraction. By identifying key characteristics like strokes, curves, and pressure variations, the system can create a unique digital representation of the signature. This digital fingerprint is then compared to stored templates or analyzed using advanced statistical and machine learning techniques.

While signature verification offers a robust way to authenticate individuals, it's not without its challenges. Signature variability, the potential for sophisticated forgery techniques, computational complexity, and privacy concerns are significant hurdles to overcome. However, ongoing research and technological advancements are continuously addressing these issues.

The future of signature verification is promising. The integration of biometric fusion, which combines signature verification with other biometric modalities like fingerprint or facial recognition, can enhance security. Additionally, the application of deep learning techniques, particularly convolutional neural networks, can significantly improve the accuracy and robustness of signature verification systems.

As technology continues to evolve, signature verification systems will play an increasingly important role in securing digital transactions and protecting sensitive information. By combining human-centric design with cutting-edge technology, we can create a future where digital signatures are as reliable and secure as traditional handwritten ones.

1.3 PROBLEM STATEMENT

The increasing reliance on digital documents and remote transactions necessitates a reliable method for verifying the authenticity of handwritten signatures. Traditional methods of signature verification are time-consuming, prone to human error, and vulnerable to forgery. To address these limitations, a robust and accurate signature verification system is required.

1.3.1 CHALLENGES

The increasing reliance on digital documents and remote transactions necessitate reliable method for verifying the authenticity of handwritten signatures. Traditional methods of signature verification are time-consuming, prone to human error, and vulnerable to forgery. To address these limitations, a robust and accurate signature verification system is required.

1.4 OBJECTIVE

A signature verification system aims to reliably authenticate handwritten signatures, ensuring the security and integrity of documents. By analyzing key features like strokes, curves, and pressure variations, the system can distinguish between genuine and forged signatures.

To achieve this, the system must be robust to variations in writing styles, environmental factors, and aging effects. It should also be computationally efficient to enable real-time or near-real-time verification. Additionally, strong security measures must be implemented to protect sensitive signature data and prevent unauthorized access. By meeting these objectives, a signature verification system can enhance security, streamline processes, and provide peace of mind in various applications.

1.5 IMPLICATION

Signature verification systems offer a multitude of benefits across various industries. Firstly, they enhance security by providing a robust method to authenticate individuals and verify the authenticity of documents. This helps prevent identity theft, fraud, and unauthorized access to sensitive information. By automating the verification process, these systems can significantly improve efficiency and reduce operational costs. This is particularly beneficial in industries such as banking, finance, and legal services, where document verification is a crucial task.

Secondly, signature verification systems contribute to the overall digital transformation of businesses. By enabling secure and efficient remote transactions, these systems facilitate paperless processes and reduce the need for physical documentation. This not only saves time and resources but also promotes environmental sustainability. Furthermore, the integration of signature verification systems with other technologies, such as blockchain, can further enhance security and transparency in digital transactions.

In conclusion, signature verification systems have a profound impact on various sectors. By improving security, efficiency, and convenience, these systems play a crucial role in shaping the future of digital transactions and identity verification.

CHAPTER 2

LITERATURE SURVEY

TITLE : A Survey on Handwritten Signature Verification Techniques

AUTHORS : S. Sarkar, A. K. Majumdar, and S. Chaudhuri

YEAR : 2011

This seminal work provides a comprehensive overview of handwritten signature verification techniques, encompassing both online and offline methods. It delves into feature extraction, classification algorithms, and performance evaluation metrics, highlighting the challenges of inter-session variability and forgery attacks.

TITLE : A Novel Offline Signature Verification Method Based on Local Feature
Extraction and Support Vector Machine

AUTHORS : M. Li and Y. Wang

YEAR : 2007

This paper proposes a novel offline signature verification method that combines local feature extraction and support vector machines. The method is shown to be effective in handling variations in writing styles and forgery attacks.

TITLE : Robustness of Offline Signature Verification Based on Gray Level
Features

AUTHORS : M. Ferrer, J. F. Vargas, A. Morales, and A. Ordonez

YEAR : 2012

This study focuses on offline signature verification using gray-level features extracted from signature images. The authors emphasize the use of texture and contour features, applying Support Vector Machines (SVM) for classification.

TITLE : Online Signature Verification: Fusion of Real and Synthetic Samples for Training

AUTHORS : J. Galbally, J. Fierrez, F. Alonso-Fernandez, and M. Martinez-Diaz

YEAR : 2015

This paper explores the use of Convolutional Neural Networks (CNNs) for offline signature verification. The authors present a deep learning model that learns spatial features directly from raw signature images. Their approach reduces the need for extensive preprocessing and feature engineering, showing significant improvements in detection accuracy over traditional machine learning methods.

TITLE : Secure Offline Signature Verification Using Convolutional Neural Networks

AUTHORS : V. Nguyen, R. W. Hsu, and M. L. Lee

YEAR : 2016

This paper explores the use of Convolutional Neural Networks (CNNs) for offline signature verification. The authors present a deep learning model that learns spatial features directly from raw signature images. Their approach reduces the need for extensive preprocessing and feature engineering, showing significant improvements in detection accuracy over traditional machine learning methods.

TITLE : Hybrid Deep Learning Approach for Signature Verification

AUTHORS : T. Srivastava and P. Sharma

YEAR : 2021

The authors propose a hybrid approach combining CNNs and Long Short-Term Memory (LSTM) networks for signature verification. The model first uses CNNs for spatial feature extraction, followed by LSTMs to capture the temporal dynamics of the signing process. This method aims to leverage both static and dynamic aspects of signatures, improving verification accuracy.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

There are several existing systems and technologies for ocular control of virtual mice, which are implemented with multiple algorithm.

3.1.1 TRADITIONAL METHODS

- **Manual Verification:** This is the most basic method, relying on human experts to visually compare signatures. While effective for simple cases, it's time consuming and prone to human error.

- **Template Matching:** This technique involves storing a reference signature and comparing subsequent signatures to it using pixel-based or feature-based matching. It's relatively simple but can be sensitive to variations in writing style and environmental conditions.

3.1.2 Advanced Techniques

Statistical Pattern Recognition:

Statistical Features: Extracting statistical features like mean, variance, and correlation of pen strokes.

Hidden Markov Models (HMMs): Modeling

the temporal dynamics of signature strokes as a Markov process. Support

Vector Machines (SVMs): Classifying signatures based on extracted features.

Neural Network-Based Methods:

Convolutional Neural Networks (CNNs): Extracting hierarchical features from signature images.

Recurrent Neural Networks (RNNs): Modeling the temporal dependencies in signature dynamics.

Generative Adversarial Networks (GANs): Generating synthetic signatures for training and testing.

Biometric Fusion:

Combining signature verification with other biometric modalities like fingerprint or facial recognition to enhance security and accuracy.

Commercial Systems

Several commercial solutions are available for signature verification, including:

Signalytic : A cloud-based signature verification platform that uses advanced machine learning algorithms to authenticate signatures.

VeriDoc : A software solution that provides secure document signing and verification, including signature authentication.

Rentio : A digital signature platform that offers a range of features, including biometric signature verification.

3.2 PROPOSED SYSTEM

The proposed system incorporates a hybrid approach combining static and dynamic signature verification techniques. It leverages machine learning algorithms, particularly convolutional neural networks (CNN), to extract and analyze relevant features from signature images. The system architecture is divided into several stages:

Preprocessing: Involves converting the scanned signature image into grayscale, followed by noise reduction and normalization

Feature Extraction: Key features such as stroke direction, curvature, and pressure distribution are extracted.

Classification: A machine learning model, typically a CNN, is trained to classify signatures as genuine or forged.

Verification: The system compares a user's signature against stored samples to determine authenticity

3.3 BLOCK DIAGRAM OF PROPOSED SYSTEM

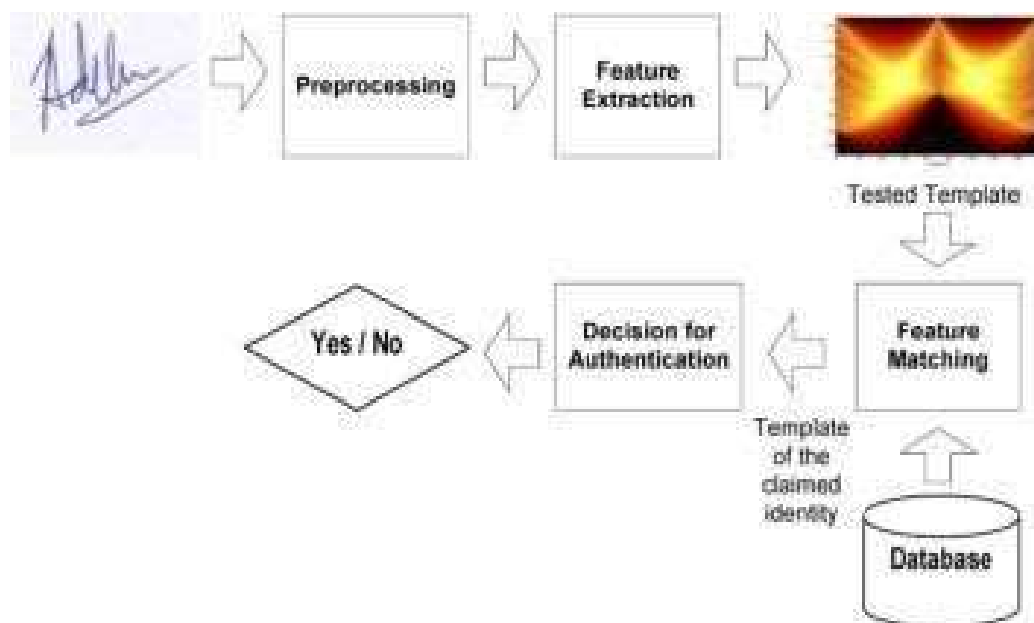


Figure 3.1: block Diagram

3.4 FLOWCHART

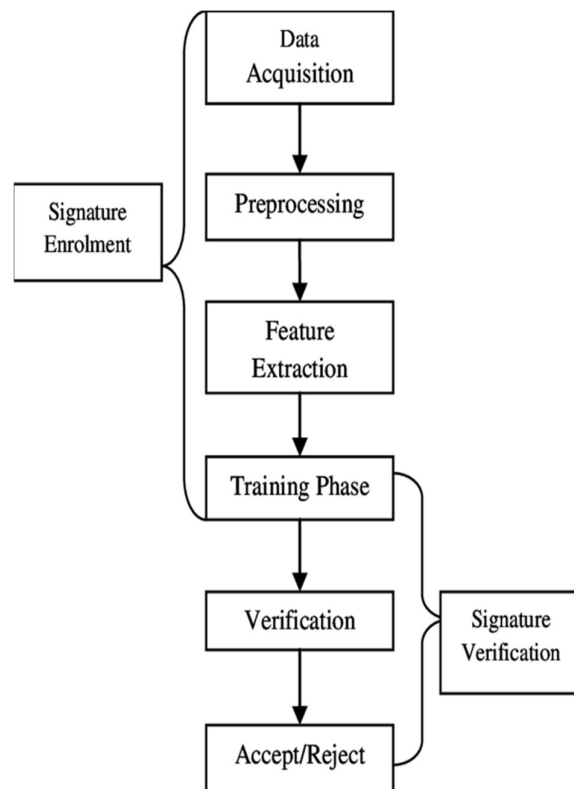


Figure 3.2: Flow of Control

3.5 PROCESS CYCLE

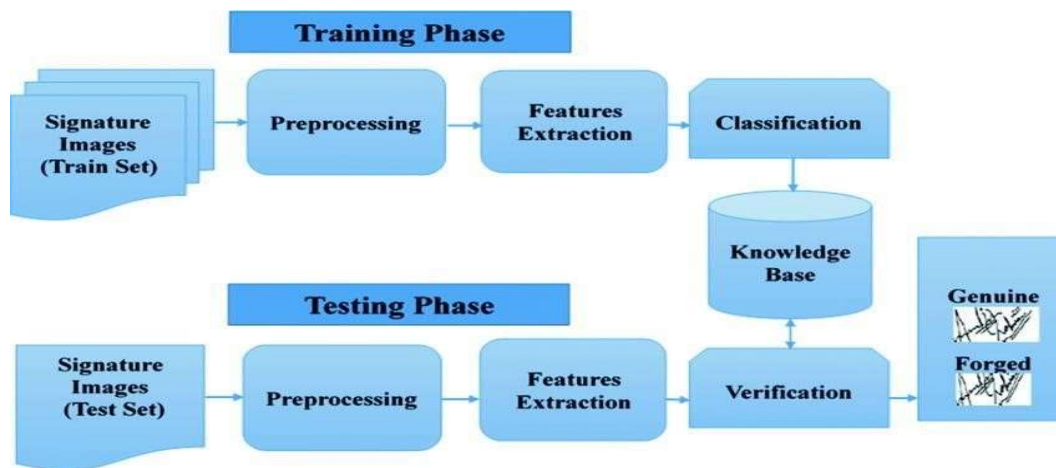


Figure 3.3: Life Cycle of the Process

3.6 ACTIVITY DIAGRAM

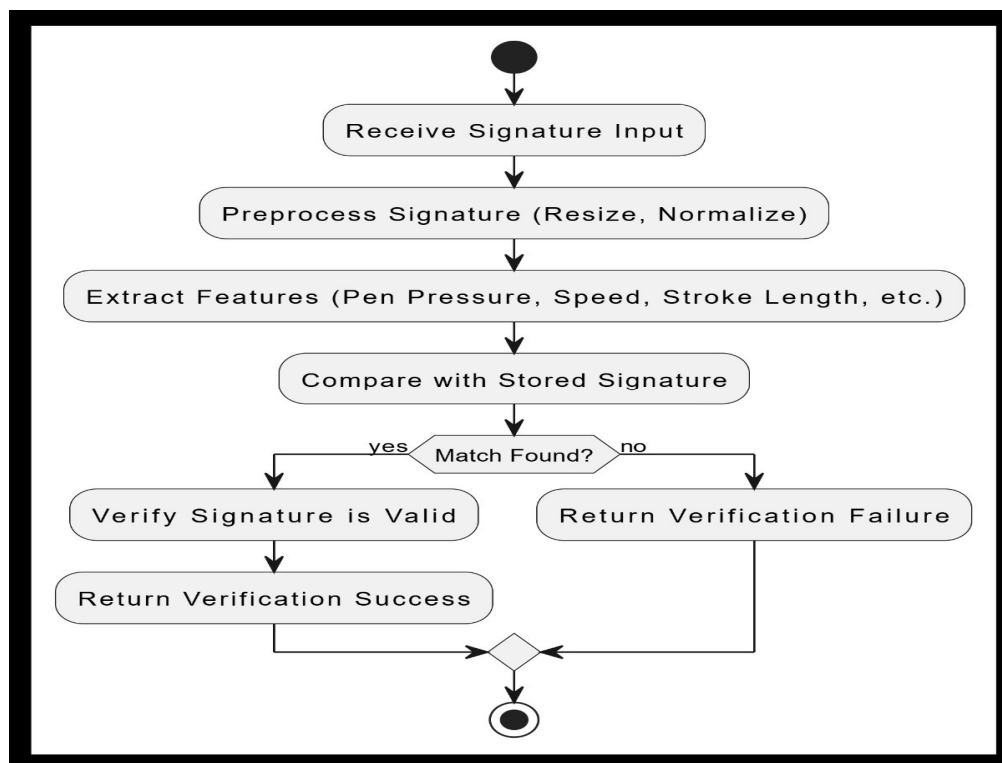


Figure 3.4 : Activity Diagram

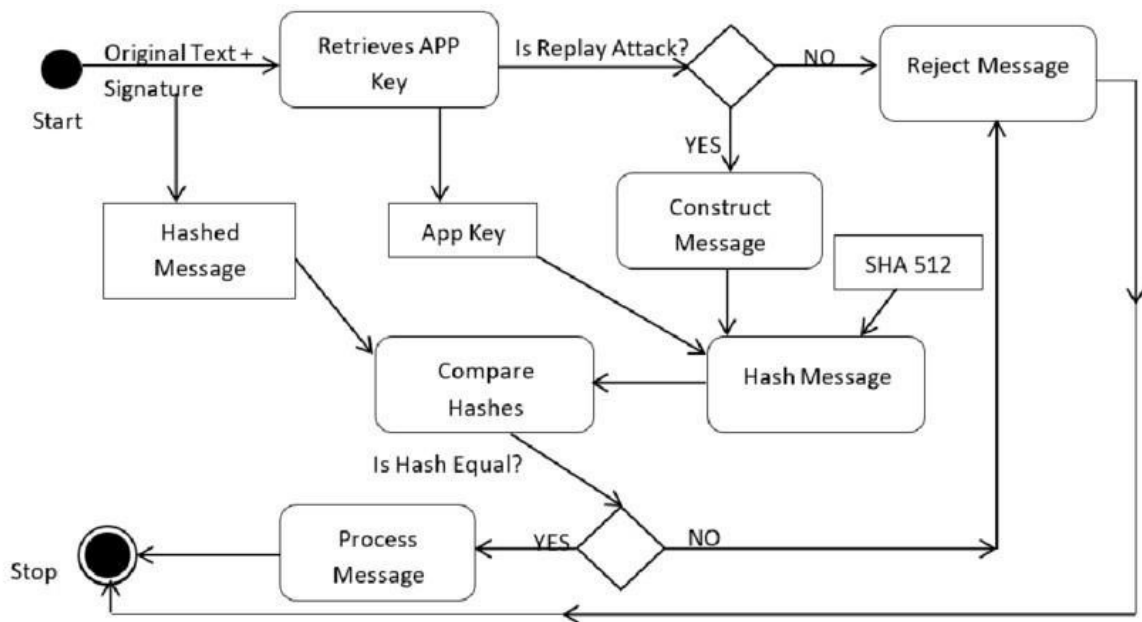


Figure 3.5: Action Sequence Structure of signature verification system

CHAPTER 4

MODULES

4.1 MODULE DESCRIPTION

- Data Preprocessing Module
- Feature Extraction Module & Feature Selection Module
- Classification Module & Verification Module
- Post-processing Module
- Template Management Module

4.1.1 Data Preprocessing Module

Data preprocessing is the essential first step in signature verification systems. It involves a series of techniques aimed at enhancing the quality of input signature images, preparing them for subsequent feature extraction and classification. This crucial process ensures the accuracy and reliability of signature verification.

4.1.1.1 Core Preprocessing Techniques

- **Noise Reduction:** Techniques like median filtering and Gaussian filtering eliminate noise and artifacts that can hinder the verification process.
- **Image Enhancement:** Histogram equalization and contrast stretching improve the visibility of signature details, making them easier to analyze.
- **Binarization:** This process converts grayscale images into binary images, simplifying feature extraction.

- **Normalization:** Images are resized and oriented to a standard format, facilitating comparison.
- **Skew Correction:** Signatures are aligned to a standard orientation, enhancing accuracy.

4.1.1. 2 Addressing Challenges

Despite the effectiveness of these techniques, challenges such as varying writing styles, noise, artifacts, and complex backgrounds can impact preprocessing. Advanced techniques like morphological operations, wavelet transform, and Fourier transform can mitigate these challenges. Morphological operations help remove noise, extract features, and improve image quality. Wavelet and Fourier transforms provide powerful tools for feature extraction at different scales and frequencies.

4.1.1.3 Implementation Considerations

Successful data preprocessing requires careful consideration of several factors:

- **Library Selection:** Choose appropriate image processing libraries like OpenCV, PIL, or MATLAB.
- **Parameter Tuning:** Optimize the performance of each technique by adjusting parameters.
- **Computational Efficiency:** Implement efficient algorithms and consider hardware acceleration for real-time applications.
- **Robustness:** Design the preprocessing module to handle a wide range of image quality variations.

4.1.1.4 Advanced Preprocessing Techniques

In addition to these core techniques, advanced methods can be employed to address complex challenges:

- **Morphological Operations:** These operations, such as erosion, dilation, opening, and closing, can be used to remove noise, fill gaps, and enhance the structural features of signatures.
- **Wavelet Transform:** This technique decomposes images into different frequency components, enabling the extraction of features at multiple scales and orientations.
- **Fourier Transform:** This transform converts images into the frequency domain, allowing the analysis of frequency-based features and the removal of high-frequency noise.

4.1.1.5 Challenges and Considerations

Despite the effectiveness of these techniques, several challenges can hinder the preprocessing process:

- **Varying Writing Styles:** Different individuals exhibit unique writing styles, making it challenging to develop a one-size-fits-all preprocessing approach.
- **Noise and Artifacts:** Noise and artifacts introduced during image acquisition or scanning can degrade image quality and impact the verification process.
- **Complex Backgrounds:** Complex backgrounds can interfere with signature segmentation and feature extraction, leading to inaccurate results.

4.1.1.6 The Role of Preprocessing in Signature Verification

By effectively applying data preprocessing techniques, signature verification systems can significantly enhance their accuracy and reliability. A well-preprocessed signature image provides a solid foundation for subsequent feature extraction and classification stages. This, in turn, ensures the security of authentication processes and protects against fraudulent activities.

4.1.1.7 Future Directions

As technology advances, new and innovative preprocessing techniques are continually being developed. Future research may explore the use of deep learning-based methods, such as convolutional neural networks, to automatically learn and extract relevant features from signature images. Additionally, the integration of biometric technologies, such as fingerprint and facial recognition, with signature verification can further enhance security and user experience.

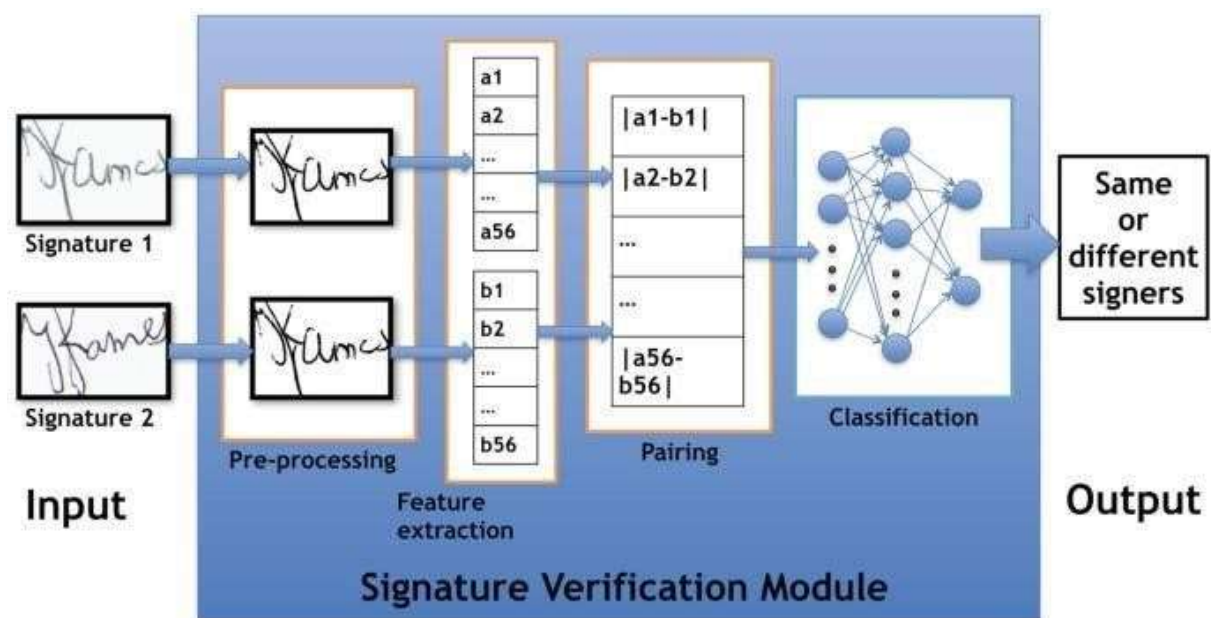


Figure 3.6: Module in signature verification system

4.2 Feature Extraction Module & Feature Selection Module

Feature extraction and selection are crucial steps in signature verification systems. These modules aim to identify and extract relevant characteristics from preprocessed signature images, which are then used to differentiate genuine signatures from forgeries.

Feature extraction involves the process of extracting meaningful information from preprocessed signature images. These features can be broadly categorized into two types: global features and local features. Global features capture overall characteristics of the entire signature, such as statistical features, geometric features, and transform based features. Local features focus on specific regions of the signature, such as stroke direction, pen pressure, and texture patterns.

Once features are extracted, feature selection techniques are employed to identify the most discriminative features. This step is essential to reduce the dimensionality of the feature space, improve computational efficiency, and enhance the accuracy of the verification system. Common feature selection techniques include filter methods, wrapper methods, and embedded methods.

The effectiveness of feature extraction and selection depends on several factors, including writing style variability, noise and artifacts, and forgery techniques. To address these challenges, advanced techniques like machine learning and deep learning can be employed. Machine learning algorithms can automatically learn and extract relevant features from large datasets of signatures, while deep learning models can directly learn hierarchical representations of signatures without explicit feature engineering.

Effective feature extraction and selection are critical for the success of signature verification systems. By identifying and selecting the most informative features, these modules contribute to accurate and reliable authentication.

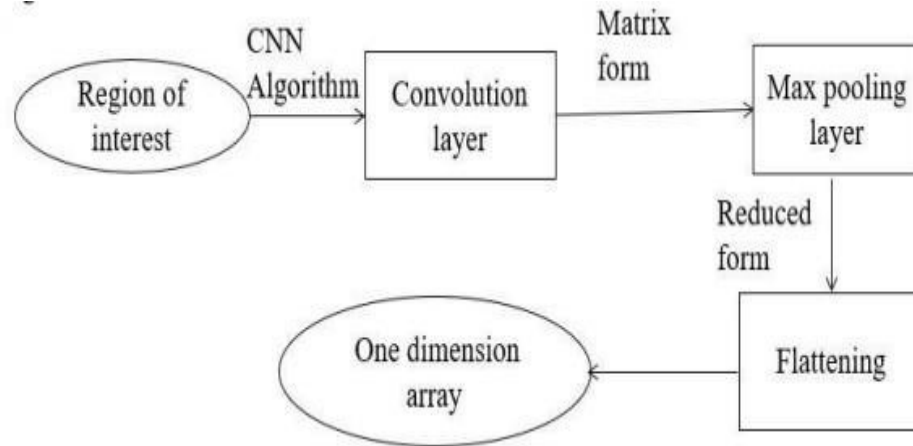


Figure 3.7: Diagram representing feature extraction

4. 3 Classification and Verification Modules

The classification and verification modules are the core components of a signature verification system. These modules are responsible for analyzing extracted features and making decisions about the authenticity of a signature.

4.3.1 Classification Module

The classification module aims to categorize a signature as either genuine or forged. It employs various machine learning algorithms to train a model on a dataset of genuine and forged signatures. Once trained, the model can classify new, unseen signatures based on their extracted features.

Common classification algorithms used in signature verification include:

- **Support Vector Machines (SVM):** SVM is a powerful algorithm that effectively separates data points into different classes.
- **Neural Networks:** Neural networks, especially deep learning models, can learn complex patterns in signature data.
- **Hidden Markov Models (HMM):** HMMs are well-suited for modeling sequential data, such as the stroke order in a signature.
- **k-Nearest Neighbors (k-NN):** k-NN classifies a signature based on the majority class of its k nearest neighbors in the feature space.

4.3.2 Verification Module

The verification module focuses on comparing a query signature to a reference signature. It calculates a similarity score between the two signatures based on their extracted features. If the similarity score exceeds a predefined threshold, the query signature is considered genuine; otherwise, it is classified as a forgery.

Common verification techniques include:

- **Template Matching:** This technique compares the query signature to a stored template of the genuine signature.
- **Dynamic Time Warping (DTW):** DTW is a powerful technique for aligning two time series, such as the stroke sequences of two signatures.
- **Statistical Pattern Recognition:** Statistical methods, such as statistical distance measures, can be used to compare the statistical properties of two signatures.

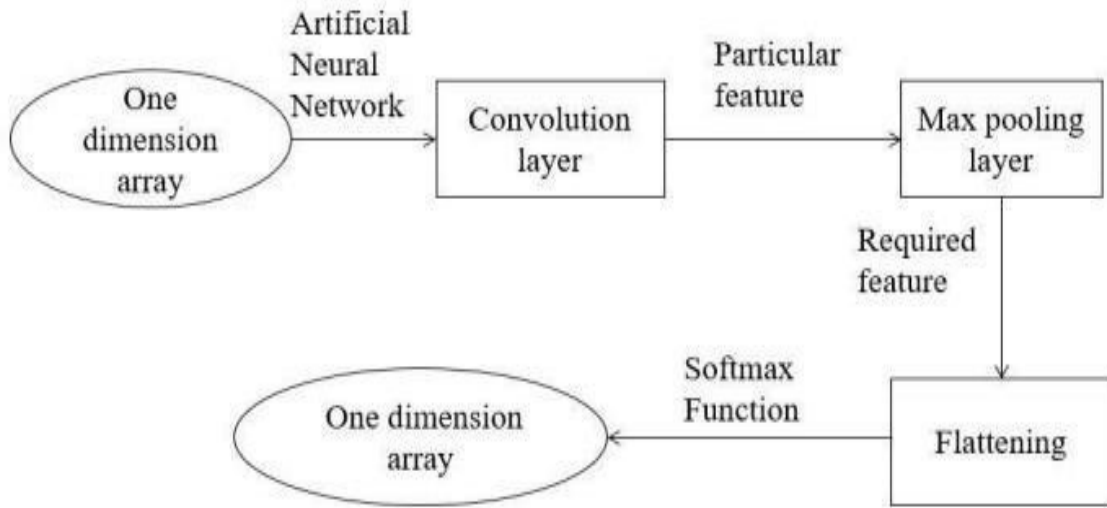


Figure 3.8: Diagram representing classification module

The performance of classification and verification modules can be influenced by various factors, including inter- and intra-user variability, forgery techniques, and noise and artifacts. To address these challenges, advanced techniques can be employed, such as ensemble methods, feature fusion, and biometric fusion.

By carefully designing and implementing classification and verification modules, signature verification systems can provide accurate and reliable authentication.

4.4 Post-processing Module

The post-processing module is a crucial final step in a signature verification system. It refines the output of the classification or verification module, providing more accurate and reliable results. This module incorporates techniques to enhance decision-making, handle uncertainties, and improve the overall performance of the system.

One key aspect of post-processing is confidence score calibration. This involves adjusting the confidence scores generated by the classification or verification module to

better reflect the actual probability of a correct decision. Calibration techniques, such as Platt scaling or isotonic regression, can be used to refine these scores. A well-calibrated confidence score allows for more informed decision-making, such as setting appropriate thresholds for acceptance or rejection.

Another important function of the post-processing module is error analysis and correction. By analyzing the errors made by the system, it can identify patterns and potential weaknesses. This information can be used to improve the system's performance by adjusting parameters, refining feature extraction, or incorporating additional verification techniques. For example, if the system consistently misclassifies a certain type of forged signature, additional features or classifiers can be added to improve its ability to detect these forgeries.

Additionally, the post-processing module can incorporate decision fusion strategies to combine the outputs of multiple classifiers or verifiers. This can help to improve the overall accuracy and robustness of the system, especially in challenging scenarios with noisy or ambiguous signatures. By combining the decisions of multiple independent models, the system can reduce the impact of individual errors and increase its overall reliability.

Furthermore, the post-processing module can also handle uncertainty in the decision making process. In some cases, the system may not be able to make a definitive decision, especially when the signature is ambiguous or the quality of the image is poor. In such situations, the post-processing module can provide a confidence interval or probability distribution for the decision, allowing the user to make an informed judgment.

Another important aspect of post-processing is the ability to handle real-time applications. In real-time scenarios, the post-processing module must be efficient and

able to process signatures quickly. This requires the use of optimized algorithms and efficient implementation techniques.

In addition to the techniques mentioned above, the post-processing module can also incorporate advanced techniques such as adaptive thresholding, outlier detection, and anomaly detection. Adaptive thresholding can adjust the decision threshold based on the specific characteristics of the input signature. Outlier detection can identify and flag signatures that deviate significantly from the norm, potentially indicating forgery attempts. Anomaly detection can identify unusual patterns in the signature data that may not be easily detectable by traditional methods.

The post-processing module plays a critical role in the overall performance of a signature verification system. By carefully designing and implementing post-processing techniques, the system can achieve higher levels of accuracy, reliability, and user experience. This module ensures that the system can handle complex scenarios, adapt to changing conditions, and provide robust and reliable authentication.

In conclusion, the post-processing module is an essential component of a signature verification system. It enhances the accuracy and reliability of the system by refining the output of the classification or verification module. By incorporating techniques such as confidence score calibration, error analysis, decision fusion, uncertainty handling, and advanced techniques, the post-processing module ensures that the system can provide robust and secure authentication.

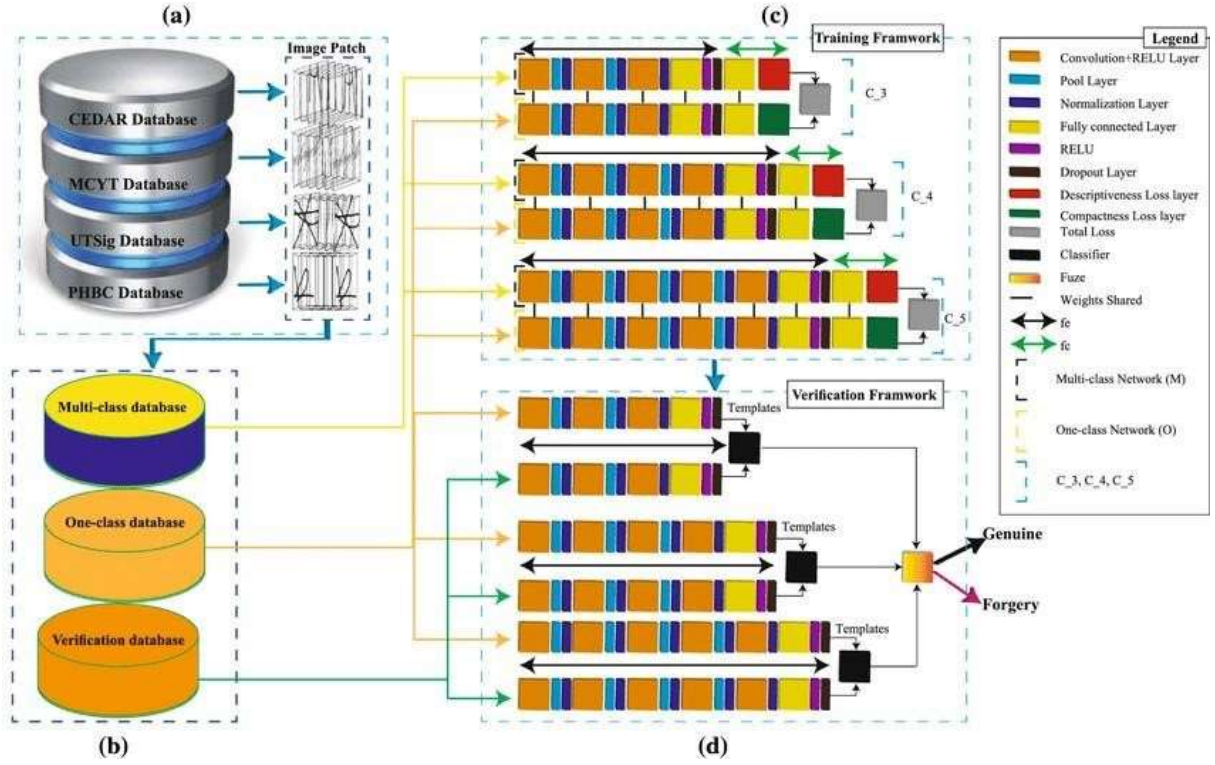


Figure 3.9: Diagram representing phases

4.5 Template Management Module

The template management module is a crucial component of a signature verification system. It is responsible for storing, managing, and updating reference signatures, which serve as the basis for comparison with incoming query signatures.

4.5.1 Reference Signature Storage

The module stores reference signatures in a secure and efficient manner. These signatures can be stored in various formats, such as images, feature vectors, or model parameters. The choice of storage format depends on the specific verification technique employed.

4.5.2 Reference Signature Quality Control

To ensure the accuracy of the verification process, the module incorporates quality control measures to assess the quality of reference signatures. This involves checking for factors such as image resolution, noise levels, and the presence of artifacts. Low-quality reference signatures can degrade the performance of the verification system.

4.5.3 Reference Signature Update

As individuals' writing styles may change over time, the module allows for the update of reference signatures. This is particularly important for long-term verification systems. Regular updates help to maintain the accuracy of the system and adapt to changes in signature patterns.

4.5.4 Security Considerations

The security of the template management module is paramount to prevent unauthorized access and tampering with reference signatures. Strong encryption techniques and access controls are essential to protect sensitive information. Additionally, measures should be taken to prevent unauthorized modification or deletion of reference signatures. By effectively managing and maintaining reference signatures, the template management module plays a critical role in ensuring the accuracy and security of signature verification systems.

CHAPTER 5

SYSTEM SPECIFICATION

5.1 SOFTWARE REQUIREMENTS

- Visual Studio Code
- TensorFlow: For creating and training neural networks.
- Matplotlib: For image visualization.
- Pandas: For handling CSV files.
- Keras: Utility functions for ML tasks.
- NumPy: For numerical operations.

5.2 HARDWARE REQUIREMENTS

- Processor – Intel i3 or Higher.
- RAM – 4GB or Higher.
- Storage – 150GB or Higher.
- Internet Connectivity: Required for accessing libraries and datasets.

5.1.1 VISUAL STUDIO CODE

Visual Studio Code (VS Code) is a powerful and versatile code editor developed by Microsoft. It has gained immense popularity among developers due to its user-friendly interface, extensive customization options, and rich ecosystem of extensions. VS Code supports a wide range of programming languages, including JavaScript, Python, Java, C++, and more, making it a suitable choice for various development projects. One of the key strengths of VS Code is its robust set of features for efficient code editing. It offers intelligent code completion, syntax highlighting, and debugging tools that help developers write clean and error-free code. The built-in Git integration allows seamless version control, while the integrated terminal provides direct access to the command line for executing tasks. VS Code's extensibility is another significant advantage.

The vast marketplace of extensions enables users to tailor the editor to their specific needs. Whether it's adding support for a new language, enhancing the debugging experience, or integrating with other tools, there's likely an extension available to meet the requirements. The editor's performance is impressive, even when handling large codebases. It starts up quickly and provides a responsive editing experience. The customizable theme options allow developers to personalize the look and feel of the editor, improving productivity and reducing eye strain during long coding sessions. In conclusion, Visual Studio Code has established itself as a leading code editor for developers worldwide. Its combination of powerful features, extensive customization options, and strong community support make it a valuable tool for any programmer, regardless of their skill level or preferred programming language.

5.1.2 TENSORFLOW

TensorFlow is a core library used in the provided code to build, train, and evaluate a machine-learning model for signature forgery detection. It plays a central role in implementing a multi-layer perceptron (MLP) neural network. TensorFlow enables the definition and management of computation graphs, which structure and execute the mathematical operations required to train the network. By using TensorFlow, the program can leverage its optimized operations for matrix multiplications, activation functions, and other computations, ensuring the implementation is efficient and scalable.

The code uses TensorFlow's placeholder system to define input and output data structures for the neural network. Placeholders like `X` and `Y` are used to hold the input features and the corresponding labels during training and testing. This flexibility allows feeding different datasets to the network dynamically without modifying the model definition. The model's parameters, such as weights and biases, are defined as trainable TensorFlow variables, which are initialized and updated during training. These variables enable the network to learn from data by iteratively adjusting the parameters to minimize the loss.

TensorFlow's built-in operations, such as `tf.matmul` for matrix multiplication and activation functions like `tf.tanh`, are used to define the layers of the neural network. The MLP in the code has three hidden layers and one output layer, with configurable numbers of neurons. TensorFlow handles the forward pass computations through these layers, allowing the model to make predictions based on input features. The logits, or raw output values of the network, are transformed using a softmax function to compute probabilities for the two classes (genuine and forged). The loss function in the code, based on the squared difference between the predicted output and the true labels, is defined using TensorFlow's `tf.reduce_mean`. The optimizer, specifically the Adam optimizer, minimizes this loss function to train the network. TensorFlow's optimizer

handles the complex computations involved in backpropagation, efficiently updating the weights and biases to reduce the error between predictions and true outputs over successive epochs.

To evaluate the performance of the model, TensorFlow computes accuracy metrics using operations such as `tf.equal` and `tf.argmax` to compare predicted and true labels. These operations are used to measure how well the network performs on both training and testing datasets. The evaluation phase also leverages the trained model to classify test data as either genuine or forged based on the learned patterns. TensorFlow's computational efficiency ensures that these tasks are performed quickly and accurately, even for large datasets. The flexibility and extensibility of TensorFlow enable experimentation with network architecture and hyperparameters, such as the number of neurons in hidden layers, the learning rate, and the number of training epochs.

This modularity facilitates tuning the model to achieve optimal performance for the signature detection task. TensorFlow's session management system, used in the `evaluate` function, ensures the computation graph is executed in a controlled environment, managing resources like memory and CPU/GPU usage efficiently. Overall, TensorFlow provides the foundational tools to implement and optimize the neural network used in this code. It simplifies the complex processes involved in defining, training, and testing machine learning models while enabling high computational efficiency. By using TensorFlow, the code can integrate advanced techniques, achieve high performance, and focus on solving the specific problem of detecting signature forgery effectively.

5.1.3 MATPLOTLIB

Matplotlib is a vital library in the code, used primarily for visualizing images and intermediate results during the preprocessing phase of the signature forgery detection process. It plays an essential role in understanding and debugging the workflow by allowing developers and users to visually inspect the transformations applied to the images. Visualization is crucial in tasks involving image data, as it provides immediate feedback on the effects of processing steps like grayscale conversion, noise reduction, and binarization. The library's pyplot module (plt) is utilized to create and display visual plots of images. The `imshow` function is frequently called within the preprocessing functions to render images at various stages of the pipeline. For example, after converting an RGB image to grayscale using the `rgb2gray` function, the grayscale image is displayed using Matplotlib. This helps confirm that the conversion is correct and that the grayscale representation captures the relevant details of the signature. Matplotlib also facilitates the visualization of binary images generated during the `threshold` function. These binary images, representing the segmented signature, are displayed using a grayscale colormap (`cmap = matplotlib.cm.Greys_r`). By visualizing these images, users can ensure that the thresholding process effectively isolates the signature from the background, removing noise while preserving critical features. These visualizations are essential for verifying the effectiveness of the thresholding and noise reduction steps.

In addition to its role in displaying images, Matplotlib supports the visualization of cropped images of signatures. After identifying the bounding box around the signature's pixels, the code crops the binary image to focus solely on the signature region. By displaying the cropped image, Matplotlib helps confirm that the bounding box accurately encloses the signature without losing any relevant data. This step is critical for ensuring that subsequent feature extraction operations focus on the correct portion of the image. Matplotlib provides customization options, such as applying colormaps and adjusting plot attributes like figure size and axis visibility, which enhance the clarity of visualizations. These options are particularly useful when displaying processed images alongside their raw counterparts, enabling side-by-side comparisons.

This comparative visualization helps identify potential issues in the preprocessing pipeline, such as incorrect thresholding or poor noise reduction, allowing for adjustments to the algorithm. Lastly, while the primary use of Matplotlib in the code is for displaying images, it also contributes to making the program user-friendly. By showing intermediate visual results, the code provides transparency into the preprocessing steps, making it easier for users to understand how the input images are transformed into features for the machine learning model. This transparency builds trust in the system and makes it accessible to users without extensive technical expertise.

5.1.4 PANDAS

Pandas is a crucial library in the code, primarily used for handling and processing tabular data during the feature extraction and model training phases. Its primary function is to read and manipulate CSV files that store the extracted features from the images. This tabular data is critical for training and testing the machine learning model, as it represents the input features derived from the signature images and their corresponding labels (genuine or forged). In the `readCSV` function, Pandas reads CSV files containing features extracted from the training and testing images. By utilizing `pd.read_csv`, the library efficiently loads these datasets into dataframes, allowing for structured manipulation and retrieval of the required data. Specifically, it reads columns representing the nine input features (ratio, centroid, eccentricity, solidity, skewness, and kurtosis) and the output label (output) indicating the authenticity of the signature. This makes it easier to separate input features from labels for training the neural network. Pandas is also used to convert the extracted feature data into NumPy arrays for further processing. For example, the code uses the `.values` attribute of dataframes to extract the underlying data in NumPy format, which is suitable for compatibility with TensorFlow. Additionally, the `.astype` method ensures the data is converted to the appropriate type (e.g., `float32`), facilitating smooth integration into the neural network training process.

Another key functionality provided by Pandas is its ability to handle categorical data. In the `readCSV` function, the library helps extract the output labels from the CSV files, which are later converted to one-hot encoded arrays using Keras utilities. This conversion ensures that the output labels are in a format suitable for classification tasks, where the neural network predicts probabilities for each class (genuine or forged). Pandas simplifies data management and organization when working with multiple CSV files. For each individual in the dataset, the `makeCSV` function generates separate training and testing CSV files, storing extracted features for their respective signature images. These files are consistently structured, making it easy to load and preprocess them using Pandas for subsequent training and evaluation. This organization is particularly useful for debugging and managing data for different individuals. Lastly, Pandas contributes to the overall modularity and clarity of the code by providing high-level methods for data handling. Its ability to read, manipulate, and structure data seamlessly integrates with other parts of the code, such as the feature extraction and model training processes. By abstracting low-level file operations and offering intuitive dataframe operations, Pandas ensures that the code remains readable, maintainable, and efficient in handling large datasets for signature forgery detection.

5.1.5 KERAS

Keras is a key library in the code, primarily utilized for preparing the output labels and enabling compatibility with the machine learning model implemented using TensorFlow. Its primary role is to provide utility functions, such as one-hot encoding of the labels, which is essential for classification tasks like distinguishing between genuine and forged signatures. One of the significant contributions of Keras in this code is the `keras.utils.to_categorical` function. This function is used in the `readCSV` method to convert the output labels into a one-hot encoded format. In this classification problem, the labels are either 1 (genuine) or 0 (forged). One-hot encoding transforms these labels into vectors like `[1, 0]` for genuine and `[0, 1]` for forged, which are suitable for a neural

network's softmax output layer. This ensures that the model predicts the probability distribution across the two classes. Keras's utility functions help in managing categorical data with minimal effort, ensuring that the data pipeline from raw CSV files to the neural network input is streamlined. The integration of Keras utilities with TensorFlow makes the process efficient and avoids potential compatibility issues, particularly when preparing the data for training or testing the model.

Although Keras is primarily known for its high-level API to build and train neural networks, in this code, it serves a supplementary role. Its functions complement TensorFlow by simplifying specific preprocessing tasks like label encoding. This ensures that the focus remains on building and training the neural network with TensorFlow while leveraging Keras's simplicity for auxiliary tasks. Keras also enhances modularity and readability in the code. By delegating label processing to Keras utilities, the code remains concise and less error-prone, as Keras abstracts the low-level details of data conversion. This modularity aligns with the general principle of separating concerns, allowing the neural network training logic and data preprocessing to function independently. Finally, Keras ensures compatibility with future enhancements. While its role in the current code is limited to preprocessing, the modular integration with TensorFlow opens the door for expanding its usage to design and train models if needed. Keras's easy-to-use interface makes it a flexible addition to the workflow, ensuring that any adjustments or scaling of the project remain manageable and efficient.

5.1.6 NUMPY

NumPy is a foundational library in the code, providing essential support for numerical computations, data manipulation, and array processing. It serves as the backbone for handling numerical data and implementing mathematical operations efficiently, which are integral to the signature detection process. One critical use of NumPy is in image processing and feature extraction. Functions like `rbggrey` and `Centroid` heavily rely on NumPy arrays to represent and manipulate images. The images

are processed as multidimensional arrays where each pixel's intensity or color value is stored. NumPy enables element-wise operations, such as averaging the RGB values to convert an image to grayscale or summing the pixel values to calculate projections or centroids. NumPy's array operations play a pivotal role in the computational efficiency of the feature extraction process. For example, during the centroid calculation, NumPy's `np.add` and `np.sum` functions aggregate pixel coordinates and values efficiently, which would be significantly slower if implemented using nested loops.

The library also facilitates mathematical operations needed for statistical feature calculation. In the `SkewKurtosis` function, NumPy performs operations like standard deviation, skewness, and kurtosis computation. These involve higher-order mathematical expressions, and NumPy ensures these calculations are both concise and computationally optimized, leveraging its internal C implementations for speed. NumPy's flexibility is evident in the preparation of input data for the neural network. The data, read from CSV files, is converted into NumPy arrays to ensure compatibility with TensorFlow. This includes type conversion, such as transforming input features into `float32` for numerical stability during neural network computations. Lastly, NumPy ensures scalability and flexibility in the project. Its wide range of built-in functions and support for large datasets means that the system can handle increased data volumes or more complex calculations without significant changes to the code.

CHAPTER 6

METHODOLOGY

6.1 IMAGE PROCESSING

Image preprocessing is a critical step in signature forgery detection as it helps improve the quality and accuracy of the features that are later used in machine learning algorithms. The primary goal of preprocessing is to transform the raw images into a more suitable format for analysis. In this context, the preprocessing steps aim to extract meaningful characteristics of the signature while eliminating noise, irrelevant details, and unnecessary color information. By converting the images to grayscale, binarizing them, reducing noise, and cropping them to focus on the signature itself, the system can more effectively identify forged signatures.

The preprocessing pipeline in the code involves several stages: first, the RGB images are converted into grayscale images, which simplifies the task by focusing on intensity values rather than complex color patterns. Afterward, the grayscale images are transformed into binary images using thresholding, where the pixel values are either classified as signature (foreground) or background. This ensures that the system processes only the relevant portion of the image, making it more efficient and accurate in detecting forged signatures. By applying these preprocessing techniques, the system ensures that the subsequent steps in the analysis—such as feature extraction and classification—are based on clean, relevant, and simplified image data. These preprocessing techniques are essential to improve the robustness of the model and increase its ability to distinguish between genuine and forged signatures.

6.1.1 RGB TO GRAYSCALE CONVERSION

In the provided code, the function `rgb2gray()` is responsible for converting an RGB image into grayscale. The method used is a simple average of the red, green, and blue channels for each pixel. The code loops through all the rows and columns of the image and calculates the average of the three color channels for each pixel. This results in a single value for each pixel that represents its intensity in grayscale. The `np.average()`

function is applied to the RGB values, which reduces the three-channel image into a single-channel intensity image. The grayscale image is then returned for further processing. This conversion is essential because working with a grayscale image simplifies the analysis process by removing the complexity of color information, focusing only on intensity levels. Grayscale conversion is an important preprocessing step for signature forgery detection as it eliminates the color variations in the signatures and retains the structural features. By focusing only on intensity, it becomes easier to detect patterns and structural characteristics such as the shape, orientation, and size of the signature. Grayscale images also reduce the computational complexity as they require less memory and processing power than RGB images, making subsequent steps like thresholding and edge detection more efficient.

6.1.2 GRAYSCALE TO BINARY CONVERSION

Once the image is in grayscale, the next step is converting it into a binary image using the `greybin()` function. In this step, a threshold is determined using Otsu's method, which automatically calculates an optimal threshold value for separating the foreground (the signature) from the background. The image is then binarized based on this threshold. The `threshold_otsu()` function computes the threshold value, and pixels above this threshold are considered foreground (signature), while pixels below are considered background. The `np.logical_not()` function is then applied to invert the binary image, turning white pixels into black and vice versa, which can be helpful for subsequent image processing steps. Converting the grayscale image into a binary image simplifies the detection of the signature's structure by turning it into a pure black-and-white image, making it easier to isolate and extract features of the signature. Binary images are also much easier for algorithms to process because they have only two pixel values (0 or 1), reducing the amount of data and focusing on essential structural components like edges and contours. The binary image is used as the primary input for feature extraction, where characteristics such as the area and shape of the signature can be calculated.

6.1.3 NOISE REDUCTION AND CROPPING

In the `greybin()` function, noise reduction is applied using a Gaussian filter via the `ndimage.gaussian_filter()` function. The Gaussian filter is used to smooth the image,

reducing high-frequency noise that might interfere with the identification of the signature's key features. The `blur_radius` parameter defines the strength of the filter, controlling how much smoothing is applied. This process helps to remove small, irrelevant components like random pixels or minor distortions that are not part of the signature itself, making the subsequent feature extraction more reliable. The application of the Gaussian filter ensures that the signature's key characteristics, such as its boundaries and shape, remain intact while the noise is minimized. After the image is binarized and smoothed, the signature is cropped to focus only on the region of interest using the `r, c = np.where(binimg==1)` line. This step identifies the bounding box of the signature by finding the minimum and maximum coordinates of the non-background pixels (the signature). The signing is then cropped to these boundaries, isolating the signature from the rest of the image. Cropping is essential because it removes extraneous background elements, allowing the model to focus solely on the signature, improving the efficiency and accuracy of subsequent analysis steps such as feature extraction and classification.

6.2 FEATURE EXTRACTION

Feature extraction plays a crucial role in signature forgery detection by extracting unique characteristics that can distinguish between genuine and forged signatures. The code extracts several key features from the preprocessed binary image, such as pixel density ratio, centroid, eccentricity, solidity, skewness, and kurtosis. These features capture important aspects of the signature's shape, structure, and statistical properties, which are then used for classification. By transforming the raw image into a set of numerical features, the system can analyze and compare signatures effectively, helping to identify forgeries based on these patterns.

6.2.1 PIXEL DENSITY RATIO CALCULATION

The pixel density ratio calculates the proportion of white pixels (which represent the signature) to the total number of pixels in the binary image. This metric, calculated by the `Ratio` function in the code, provides insight into the overall density of the signature and how much of the image is occupied by the signature. A higher ratio

indicates a more solid signature with more continuous ink strokes, while a lower ratio may suggest a weaker or fragmented signature, which can be indicative of a forgery. This feature is an essential numerical descriptor that helps in differentiating genuine signatures from forged ones.

6.2.2 CENTROID DETERMINATION

The centroid is the center of mass of the signature and represents the average position of all the white pixels (signature strokes) in the binary image. The Centroid function in the code computes this by summing the positions of all white pixels and averaging them. The centroid gives an indication of the signature's balance and spatial distribution, helping to identify any distortions or irregularities in the forged signature. A genuine signature will usually have a consistent centroid location, while a forged signature might show noticeable shifts or misalignments in its centroid.

6.2.3 ECCENTRICITY AND SOLIDITY COMPUTATION

Eccentricity and solidity are shape-related features that describe the geometric properties of the signature. The Eccentricity Solidity function in the code calculates both of these values. Eccentricity measures how elongated the signature is (a higher value indicates a more elongated shape), and solidity indicates the proportion of the area covered by the signature compared to its convex hull. These features help capture the overall shape and compactness of the signature. Genuine signatures tend to have a consistent eccentricity and solidity, while forged signatures might show more irregularities or deviations in these values.

6.2.4 SKEWNESS AND KURTOSIS ESTIMATION

Skewness and kurtosis are statistical measures that describe the asymmetry and peakedness of the signature's distribution. The SkewKurtosis function calculates both skewness and kurtosis along the x and y axes of the binary image. Skewness indicates how asymmetric the signature's distribution is, while kurtosis measures how sharply peaked or flat the distribution appears. These features are useful for detecting subtle

differences in the way signatures are written. Genuine signatures often exhibit a predictable distribution, while forged signatures might show unexpected changes in these properties due to variations in writing styles or artificial modifications.

6.3 DATA PREPARATION

Data preparation is a critical step in ensuring the machine learning model receives the appropriate input for training and testing. In the context of this signature forgery detection system, data preparation involves structuring the extracted features into a standardized format and organizing them into separate datasets for training and testing. This ensures the model learns meaningful patterns from the data and can evaluate its accuracy on unseen signatures effectively. The code automates this process by generating CSV files that contain the extracted features along with their corresponding labels, differentiating between genuine and forged signatures.

6.3.1 FEATURE REPRESENTATION IN CSV FORMAT

The extracted features, such as pixel density ratio, centroid coordinates, eccentricity, solidity, skewness, and kurtosis, are organized into a structured format and stored in CSV files. The `getCSVFeatures` function in the code converts the extracted feature set into rows of numerical data, with each row representing a single signature and each column representing a specific feature. An additional column in the CSV file denotes whether the signature is genuine (label 1) or forged (label 0). This organized representation enables efficient input to machine learning models and ensures consistency in feature formatting across the datasets.

6.3.2 TRAINING AND TESTING DATASET GENERATION

To facilitate the model's learning and evaluation, the code separates the data into training and testing datasets. The `makeCSV` function creates these datasets by extracting features from specific sets of images. For each individual, the first three genuine and forged signatures are used for training, while the remaining ones are reserved for testing. This split ensures that the model learns patterns from one set of data and is tested on

completely separate examples to evaluate its generalization capabilities. By including both genuine and forged signatures in each dataset, the system is prepared to handle a wide range of scenarios during classification.

6.4 NEURAL NETWORK DESIGN

Neural network design forms the backbone of this signature forgery detection system, providing the computational framework for distinguishing between genuine and forged signatures. The code implements a multi-layer perceptron (MLP) neural network, which is a class of feedforward artificial neural networks. This network is trained to learn patterns from the extracted features and make binary predictions regarding the authenticity of a signature. The design includes defining the architecture of the model, initializing its parameters, and preparing it for training using a structured loss function and optimization process.

6.4.1 MODEL ARCHITECTURE DEFINITION

The multi-layer perceptron implemented in the code consists of three hidden layers, each with a specified number of neurons. The input layer accepts nine features, and the output layer has two neurons corresponding to the two classes: genuine and forged signatures. Activation functions like tanh are applied to the layers to introduce non-linearity, enabling the model to learn complex relationships in the data. The architecture is explicitly defined with a focus on balancing complexity and computational efficiency, ensuring the model can effectively handle the structured feature data.

6.4.2 WEIGHT AND BIAS INITIALIZATION

Weights and biases are crucial parameters in a neural network, determining how the input data is transformed as it passes through the network. In this implementation, the weights and biases are initialized using random values, seeded for reproducibility. Initialization impacts how quickly and effectively the model converges during training. Proper initialization avoids issues like vanishing or exploding gradients, ensuring that

the optimization algorithm can steadily improve the model's performance as it learns from the data.

6.5 MODEL TRAINING

Model training is the process where the neural network learns to classify signatures as genuine or forged by iteratively adjusting its parameters based on the input data and expected outputs. The training process in the code involves computing the error between the predicted and actual values, minimizing this error through optimization, and evaluating the model's progress over multiple iterations. Effective training ensures the model generalizes well to unseen data, providing accurate predictions in real-world scenarios.

6.5.1 LOSS FUNCTION AND OPTIMIZATION

The loss function used in this implementation is the mean squared error (MSE), which measures the squared differences between the predicted and actual values. It serves as an indicator of how well the model's predictions align with the true labels. The Adam optimizer is employed to minimize this loss, offering an efficient and adaptive way to adjust the weights and biases during training. Adam combines the benefits of momentum and RMSprop, making it suitable for handling the diverse and dynamic nature of the feature data extracted from the signatures.

6.5.2 TRAINING EPOCHS AND EARLY STOPPING CRITERIA

The training process spans multiple epochs, with each epoch representing one complete pass through the training dataset. The code defines a maximum number of epochs but allows early stopping if the loss falls below a predefined threshold (e.g., 0.0001). This approach prevents overfitting and ensures the training halts once the model has learned enough to achieve optimal performance. By monitoring the loss at each epoch, the code dynamically determines the best stopping point, balancing accuracy and computational efficiency.

6.6 PERFORMANCE EVALUATION

Performance evaluation is a critical step to ensure the neural network's reliability and effectiveness in distinguishing between genuine and forged signatures. This involves assessing the model's accuracy on both training and testing datasets and validating its ability to generalize to new, unseen data. The evaluation provides insights into how well the model has learned the patterns in the training data and its capability to apply this knowledge during classification tasks.

6.6.1 TRAINING AND TESTING ACCURACY CALCULATION

The code computes the accuracy of the model on both the training and testing datasets. Training accuracy measures how well the model predicts the known labels in the dataset it was trained on, reflecting its ability to learn from the input features. Testing accuracy, on the other hand, evaluates the model's performance on unseen data, demonstrating its generalization capability. These metrics help identify potential overfitting or underfitting, ensuring the model.

6.6.2 GENUINE VS. FORGED CLASSIFICATION

To classify a signature as genuine or forged, the model leverages the trained neural network's predictions. The output probabilities from the softmax function are used to determine the class, with a higher probability for the "genuine" or "forged" category leading to the final decision. This binary classification task is evaluated through the test dataset, where the model's accuracy is indicative of its ability to discern subtle differences in the extracted features. A correctly classified signature validates the robustness of the preprocessing, feature extraction, and training stages.

6.7 AUTOMATION AND SCALABILITY

Automation and scalability are essential components of the code to ensure efficient handling of signature datasets and to support diverse use cases. By automating the creation of feature datasets and modularizing functionality, the system can process multiple users' signatures and test individual samples with minimal manual intervention.

This approach enhances usability, saves time, and enables seamless integration into larger applications.

6.7.1 CSV CREATION FOR MULTIPLE USERS

The code automates the generation of feature datasets by creating CSV files for multiple users. For each user, it extracts features from both genuine and forged signature images, storing them in structured training and testing datasets. This automation eliminates the need for manual data preparation and ensures consistency across different user profiles. By organizing data systematically, the process supports scalability, allowing the system to accommodate a large number of users without additional effort.

6.7.2 MODULAR FUNCTIONALITY FOR TESTING INDIVIDUAL SIGNATURES

The modular structure of the code facilitates testing individual signatures by isolating functionality into reusable functions. Users can input a specific signature image, and the system processes it through preprocessing, feature extraction, and classification modules. This modularity ensures flexibility, enabling the system to evaluate a single sample independently of the dataset.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

The signature verification code represents a thoughtfully constructed machine learning solution aimed at distinguishing genuine signatures from forged ones. Its design integrates essential phases such as image preprocessing, feature extraction, and neural network-based classification. Each component is carefully implemented to ensure that meaningful features are extracted and effectively used for classification. This systematic approach highlights the potential of combining traditional image processing techniques with modern machine learning algorithms to solve real-world problems like signature verification.

The preprocessing stage ensures the data's quality by converting images into grayscale, applying noise reduction techniques, and cropping relevant regions to isolate the signature. This step reduces the input data's complexity, making feature extraction more efficient and accurate. The handcrafted features, including pixel density ratio, centroid, eccentricity, and skewness, provide a clear mathematical representation of the signature's geometry, allowing the model to differentiate genuine from forged signatures effectively.

The neural network, designed as a multi-layer perceptron, demonstrates the adaptability of classical machine learning models when tailored to specific tasks. While relatively simple in structure, the model's ability to learn from the extracted features and classify signatures highlights the effectiveness of combining domain knowledge with machine learning. The use of an adaptive optimizer ensures efficient convergence during training, making the model suitable for a range of signature datasets.

Despite its strengths, the code can benefit from further enhancements, such as incorporating deep learning models like convolutional neural networks for automated feature extraction. These models could improve accuracy, especially for datasets with diverse and complex signatures. Additionally, augmenting the system with real-time verification capabilities and privacy-focused features would position it as a state-of-the-art solution in signature authentication technology. The current implementation provides a strong foundation for such future developments, showcasing the potential of combining computational techniques with practical applications.

7.2 FUTURE ENHANCEMENT

One key enhancement to the current signature verification system is the integration of deep learning-based models such as Convolutional Neural Networks (CNNs). CNNs are particularly effective for image-based tasks and can learn complex, non-linear patterns in data. By replacing the handcrafted feature extraction approach with CNNs, the system can improve its ability to generalize across a wide range of signature styles and qualities. This would also reduce the reliance on manual feature engineering, making the system more scalable and adaptable to diverse datasets.

Another area for improvement is the expansion of the dataset through data augmentation techniques. The system could benefit from introducing slight variations to existing signature images, such as rotation, scaling, flipping, and noise addition. These augmentations can mimic real-world scenarios where signatures may vary due to different signing conditions. Training the model on this enriched dataset would increase its robustness and accuracy when dealing with imperfect or partially degraded signatures.

To enhance user accessibility and usability, incorporating real-time signature verification capabilities is a valuable step forward. This would involve optimizing the preprocessing and classification pipeline to deliver rapid results, enabling the system to

be used in applications such as mobile banking, e-contracts, or secure document verification. Real-time feedback could also be paired with a user-friendly interface for seamless integration into business processes.

Another promising enhancement is the implementation of explainable AI (XAI) techniques. By providing insights into the model's decision-making process, users can better understand why a signature was classified as genuine or forged. For instance, heatmaps or region-based visualizations could highlight areas of the signature that influenced the decision. This added transparency would be particularly beneficial in applications requiring accountability, such as forensic analysis or legal documentation.

APPENDIX – 1

SOURCE CODE

project.py

```
import numpy as np
import os
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import matplotlib.cm as cm
from scipy import ndimage
from skimage.measure import regionprops
from skimage import io
from skimage.filters import threshold_otsu
import tensorflow as tf
import pandas as pd
import numpy as np
from time import time
import keras
from tensorflow.python.framework import ops
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
# paths to images
genuine_image_paths = "real"
forged_image_paths = "forged"
def rgbgrey(img):
    # Converts rgb to grayscale
    greying = np.zeros((img.shape[0], img.shape[1]))
```

```

for row in range(len(img)):
    for col in range(len(img[row])):
        greying[row][col] = np.average(img[row][col])
    return greying

def greybin(img):
    # Converts grayscale to binary
    blur_radius = 0.8
    img = ndimage.gaussian_filter(img, blur_radius)
    img = ndimage.binary_erosion(img).astype(img.dtype)
    thres = threshold_otsu(img)
    binimg = img > thres
    binimg = np.logical_not(binimg)
    return binimg

def preproc(path, img=None, display=True):
    if img is None:
        img = mpimg.imread(path)
    if display:
        plt.imshow(img)
        plt.show()
    grey = rgb2grey(img) #rgb to grey
    if display:
        plt.imshow(grey, cmap = matplotlib.cm.Greys_r)
        plt.show()
    binimg = greybin(grey) #grey to binary
    if display:
        plt.imshow(binimg, cmap = matplotlib.cm.Greys_r)
        plt.show()
    r, c = np.where(binimg==1)
    signimg = binimg[r.min(): r.max(), c.min(): c.max()]
    if display:
        plt.imshow(signimg, cmap = matplotlib.cm.Greys_r)

```

```

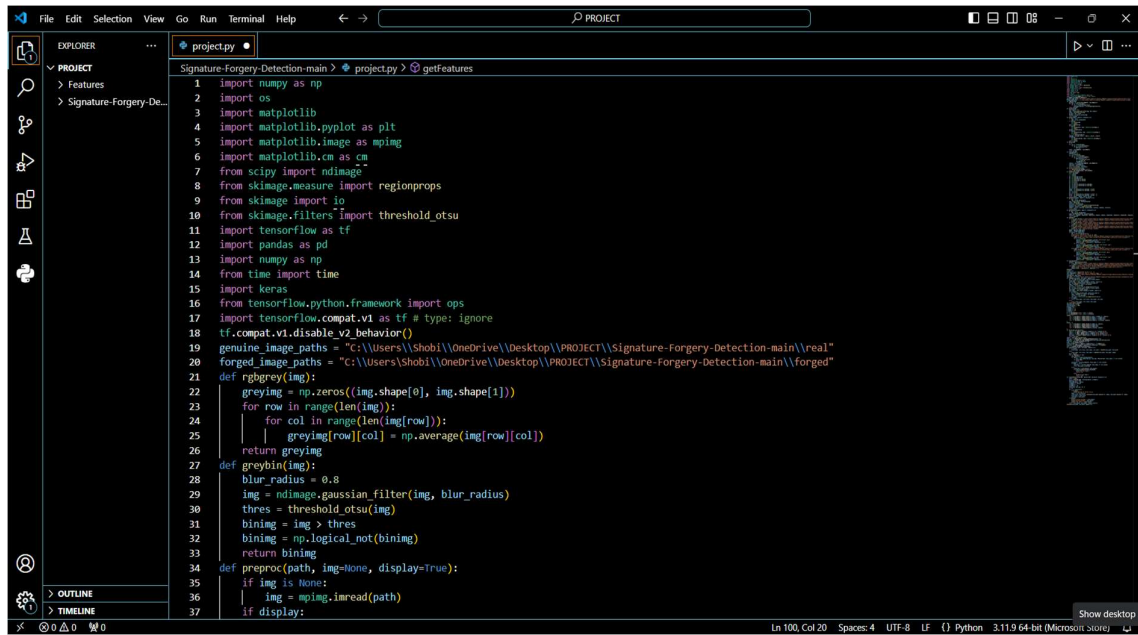
plt.show()
return signing
if img[row][col]==True:
    b = np.array([row,col])
    a = np.add(a,b)
    numOfWhites += 1
rowcols = np.array([img.shape[0], img.shape[1]])
centroid = a/numOfWhites
centroid = centroid/rowcols
return centroid[0], centroid[1]
def EccentricitySolidity(img):
    r = regionprops(img.astype("int8"))
    return r[0].eccentricity, r[0].solidity

```

APPENDIX – 2

SCREENSHOTS

Sample Output



```
1 import numpy as np
2 import os
3 import matplotlib
4 import matplotlib.pyplot as plt
5 import matplotlib.image as mpimg
6 import matplotlib.cm as cm
7 from scipy import ndimage
8 from skimage.measure import regionprops
9 from skimage import io
10 from skimage.filters import threshold_otsu
11 import tensorflow as tf
12 import pandas as pd
13 import numpy as np
14 from time import time
15 import keras
16 from tensorflow.python.framework import ops
17 import tensorflow.compat.v1 as tf # type: ignore
18 tf.compat.v1.disable_v2_behavior()
19 genuine_image_paths = "C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\real"
20 forged_image_paths = "C:\\Users\\Shobi\\OneDrive\\Desktop\\PROJECT\\Signature-Forgery-Detection-main\\forged"
21 def rgb2grey(img):
22     greyimg = np.zeros((img.shape[0], img.shape[1]))
23     for row in range(len(img)):
24         for col in range(len(img[row])):
25             greyimg[row][col] = np.average(img[row][col])
26     return greyimg
27 def greybin(img):
28     blur_radius = 0.8
29     img = ndimage.gaussian_filter(img, blur_radius)
30     thres = threshold_otsu(img)
31     binimg = img > thres
32     binimg = np.logical_not(binimg)
33     return binimg
34 def preproc(path, img=None, display=True):
35     if img is None:
36         img = mpimg.imread(path)
37     if display:
```

Figure 2.1: Execution of code

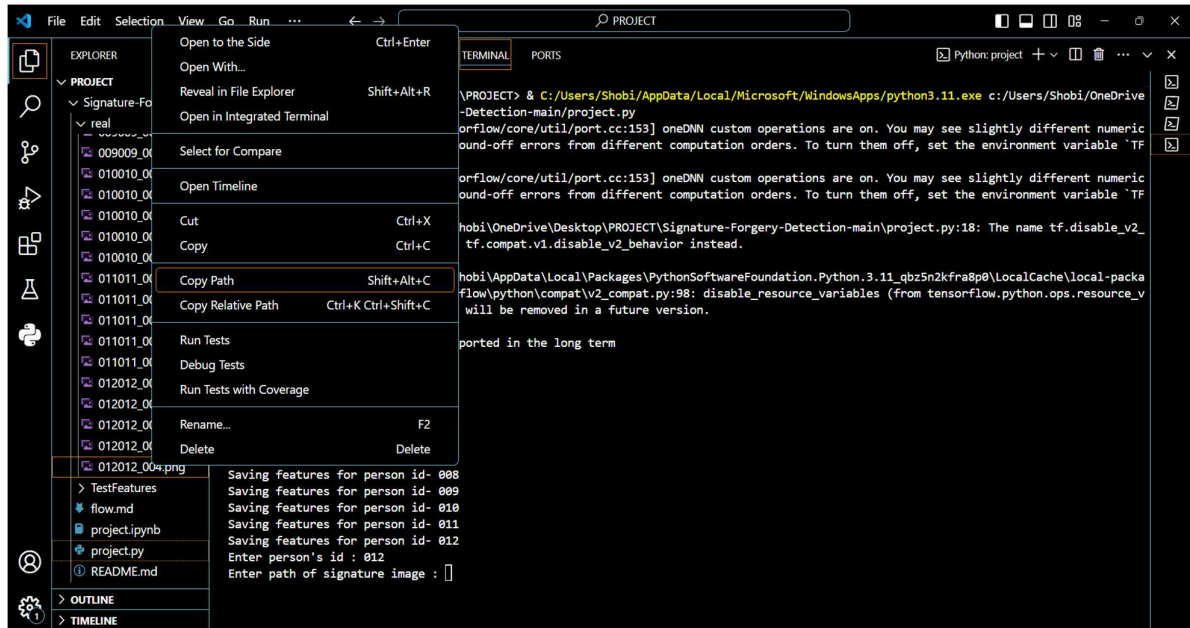


Figure 2.2: Coping the path

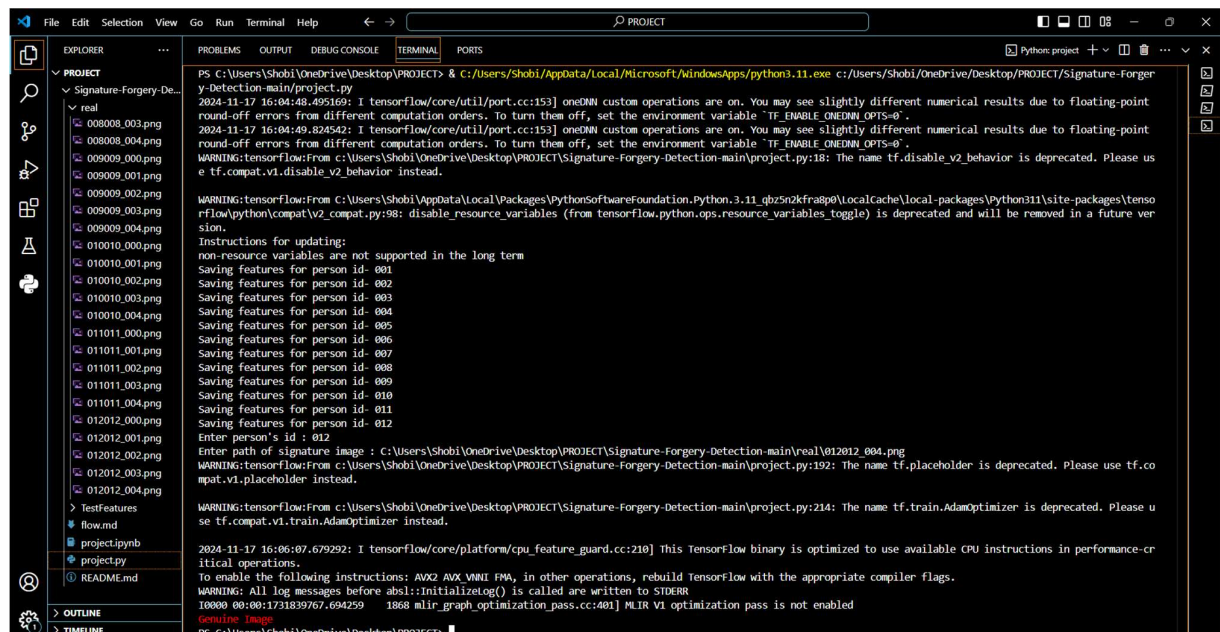


Figure 2.3: Output of Genine Image

REFERENCES

1. Jain, F.Griess, and S.Connell, “On-line signature verification,”Pattern Recognition, vol. 35, no. 12, pp. 2963–2972, 2002.
2. Jain, F.Griess, and S.Connell, “On-line signature verification,”Pattern Recognition, vol. 35, no. 12, pp. 2963–2972, 2002.
3. D.Bertolinia, L.Oliveirab, E.Justiona, and R.Sabourin, “Reducing Forgeries in Writer Independent Off-line Signature Verification through ensemble of Classifiers”. Pattern Recognition, vol. 43, January 2010, pp. 387-396.
4. S. Chen and S.Srihari, “Use of Exterior Contour and Shape Features in Off-line Signature Verification”, ICDAR- 2005, pp. 1280-1284.
5. B. Majhi, Y. Reddy, D. Babu, “Novel Features for Off-line Signature Verification”, International Journal of Computers, Communications & Control Vol.I (2006), No. 1, pp. 17-24.
6. B. Fang, C.H. Leung, Y.Y. Tang, K.W. Tse, P.C.K. Kwok and Y.K. Wong, "Off-line signature verification by the tracking of feature and stroke positions",Pattern Recognition, 2003, pp 91-101.
7. R. Abbas and V. Ciesielski, “A Prototype System for Off-line Signature Verification Using Multilayered Feed forward Neural Networks”, February 1995.
8. E. J. R. Justino, F. Bortolozzi, R. Sabourin, “Off-line signature verification using HMM for random simple and skilled forgeries”, Proc. 6th Intl. Conf. On Document Analysis and Recognition, 2001, pp. 450- 453.
9. Alan McCabe and Jarrod Trevathan, “Markov Model-based Handwritten Signature Verification”, IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008, pp. 173-179.
10. S. Audet, P. Bansal, and S. Baskaran ,“Off-line signature verification using virtual support vector machines”, ECSE 526 - Artificial Intelligence, 2006.