International Journal of Trend in Scientific Research and Development (IJTSRD)
Special Issue on Advancements and Emerging Trends in Computer Applications - Innovations, Challenges, and Future Prospects
Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Real - Time Signature Authentication

Mayur Onkar

Department of Computer Application, G. H. Raisoni University, Amravati, Maharashtra, India

ABSTRACT

The increasing need for secure identity verification, biometric authentication methods have gained prominence. Among them, handwritten signature authentication remains a widely accepted and legally recognized method. This paper presents a real-time signature authentication system utilizing deep learning techniques for accurate and efficient verification. The system captures dynamic features of a signature, such as pressure, speed, and stroke order, and verifies it against a stored template using a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) based model. The proposed system achieves a high accuracy on benchmark datasets and is deployable on lightweight edge devices, ensuring scalability and real-time performance.

1. INTRODUCTION

In Signature verification has long been used in banking, legal documents, and personal identification. Traditional methods are prone to forgery and inconsistency, particularly in offline scenarios. This paper proposes a real-time online signature verification system that captures dynamic information during the signing process and employs deep learning to authenticate the user.

1. Motivation

- Need for secure, fast, and user-friendly identity verification.
- Limitation of static (offline) signature verification systems.
- Increasing use of touch-enabled devices and IoT in secure transactions.

2. Objective

To develop a robust real-time signature authentication system using a hybrid CNN-LSTM architecture capable of verifying signatures with high accuracy and minimal latency.

2. Related Work

Previous research includes statistical methods, Dynamic Time Warping (DTW), Hidden Markov Models (HMM), and machine learning techniques. However, these often fail to generalize well across varied signing behaviors. Recent advancements in deep learning, especially CNN and RNN-based models, have shown promise in capturing both spatial and temporal features in biometric signals.

- System Architecture

1. Data Acquisition

- Collected using stylus/touch input devices.
- Data includes x-y coordinates, pen pressure, timestamp, and pen-up/pen-down events.
- Dataset: SigWiComp2013, MCYT-100, or custom-collected signatures.

2. Preprocessing

- Noise filtering and normalization.
- Feature extraction: velocity, acceleration, pressure, and stroke order.
- Data augmentation using jittering, scaling, and rotation.

3. Model Design

- CNN Layer: Extracts spatial features from signature images or strokes.
- LSTM Layer: Captures temporal dependencies in the signing process.
- Fully Connected Layers: For final classification as genuine or forged.

Architecture Summary:

Input → CNN → LSTM → Dense → Sigmoid Output

4. Training & Validation

- Loss function: Binary Cross Entropy.
- Optimizer: Adam.
- Evaluation Metrics: Accuracy, Precision, Recall, F1-score.

5. Implementation

- Frontend: Built using JavaScript/React for real-time signature capture.
- Backend: Python + Flask/Django with TensorFlow or PyTorch for model inference.
- Deployment: Dockerized backend hosted on cloud or Raspberry Pi for edge applications.

3. Required specification

1. Software Requirements:

- Java Development Kit (JDK) 11 or later – Essential for the backend development and execution of the application.
- Spring Boot Framework – For creating a scalable and robust backend system.
- Machine Learning Libraries (e.g., Weka, TensorFlow, or Deeplearning4j) – To implement signature recognition algorithms and data training models.
- Database (MySQL/PostgreSQL) – For storing signature data, user profiles, and authentication logs.
- Frontend Framework (React, Angular, or Vue.js) – For building the user interface for digital signature capture and authentication.
- Page No:2 WebSocket/Socket.IO – For real-time signature data transmission and validation.
- Cloud Services (AWS, Google Cloud, or Azure) – For scalable deployment and secure data storage.
- Mobile Application Development (React Native or Flutter) – For mobile support for signature-based authentication.

2. Hardware Requirements:

- Processor: Intel Core i5 or equivalent for smooth execution.
- RAM: Minimum of 8GB to support machine learning model execution and real-time data processing.

- Storage: At least 50GB of free space for data storage, signature models, and logs.
- Signature Capture Device: A digitizing tablet or touchscreen device to accurately capture signature data (pressure, speed, stroke dynamics).
- Internet Connection: Required for cloud storage, API calls, and real-time data validation.

4. Methodology

1. Data Collection & Preprocessing
- Gather real-time signature data from users through a digital signature pad, touchscreen device, or stylus-enabled interface.
- Capture key biometric features such as stroke order, pressure, speed, acceleration, and direction of pen movement.
- Store multiple samples per user to create a unique signature profile for training and validation.

2. Feature Extraction
- Extract relevant features from the captured signature, including pressure variation, pen velocity, stroke timing, and curvature.
- Convert the extracted features into numerical values to facilitate machine learning processing.

3. Machine Learning Model Training
- Train a classification model using supervised learning algorithms such as Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), or Support Vector Machines (SVM).
- Use labeled datasets of genuine and forged signatures to improve model accuracy.
- Implement feature normalization techniques to handle variations in signature styles over time.

4. Real-Time Authentication Process
- Capture a new signature in real time during a transaction or document signing process.
- Compare the real-time signature with the stored reference signature using machine learning-based classification techniques.
- If the signature matches within an acceptable threshold, authentication is granted; otherwise, it is flagged as suspicious.

5. Performance Evaluation & Accuracy Optimization
- Evaluate model performance using accuracy, precision, recall, and F1-score.
- Optimize the model by fine-tuning hyperparameters and using larger datasets for better generalization.
- Implement anomaly detection techniques to identify possible signature forgery attempts.

6. Integration with Digital Systems
- Deploy the authentication system as a standalone application or integrate it with existing digital platforms, such as banking systems, legal documentation services, and secure enterprise applications.
- Provide an API for seamless integration with third-party applications that require signature-based authentication.

7. Security & Scalability Enhancements
- Implement encryption mechanisms to protect stored signature data from unauthorized access.
- Ensure scalability to handle a large number of users and transactions in real time.
- Continuously update the system to adapt to evolving security threats and improvements in AI-based authentication methods.
- Page No:45. Implementation Plan with Modules

5. Result and Evaluation

Model. Accuracy. Precision. Recall. F1-score
CNN Only. 85.4%. 83.1%. 84.9%. 84.0%
LSTM Only. 88.2%. 86.4%. 87.8%. 87.1%
CNN + LSTM. 94.6%. 92.5%. 93.7%. 93.1%

5.1. Real-Time Performance

- Average processing time per signature: ~150 ms.
- Compatible with touchscreen tablets and smartphones.

6. Conclusion and Future Work

This paper presents a real-time signature authentication system using a deep learning model that effectively combines CNN and LSTM architectures. The system shows promising results and is suitable for real-world deployment. Future work includes:

- Expanding the dataset with diverse users.
- Improving robustness against high-skilled forgeries.
- Integrating with blockchain for tamper-proof verification.

Reference

[1] Java Documentation: https://docs.oracle.com/en/java/

[2] Spring Boot Framework: https://spring.io/projects/spring-boot

[3] Weka Machine Learning Library: https://www.cs.waikato.ac.nz/ml/weka/

[4] TensorFlow: https://www.tensorflow.org/

[5] Deeplearning4j: https://deeplearning4j.org/

[6] AWS Cloud Services: https://aws.amazon.com/

[7] Google Cloud Platform: https://cloud.google.com/