iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** VII **Month of publication:** July 2025

DOI: https://doi.org/10.22214/ijraset.2025.73260

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VII July 2025- Available at www.ijraset.com

Signature Verification System to Automatically Decide Whether a Scanned Signature Image is Genuine or Forged

Prof. Girish Patil[1], Hrishikesh Gavane[2], Samarth Kadam[3], Yash Kale[4]
Dept. of Artificial intelligence, G H Raisoni College of engineering and management Pune, India

Abstract: ***This paper introduces a strong system for checking handwritten signatures without needing to see the person sign in real time. This system helps stop fraud in important areas like banking and legal documents. The process starts with preparing the signature image by turning it into grayscale, removing noise, making it black and white, and adjusting its size. Then, it uses two methods, Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP), to extract important features from the signature. A Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel is used to classify the signature as real or fake. This classifier is trained using a process called grid-search cross-validation. The system is built using Python 3. 9 along with libraries like OpenCV, scikit-learn, and NumPy. It also includes a simple user interface made with Flask or Tkinter. When tested on a standard dataset like GPDS, the system achieved an overall accuracy of 96. 5%, a false-acceptance rate of 2. 3%, and a false-rejection rate of 3. 7%. It can process each signature quickly, under 200 milliseconds, and performs well compared to other advanced methods.***

Keywords: Offline signature verification, feature extraction, HOG, LBP, SVM, OpenCV, machine learning.

I. INTRODUCTION

Handwritten signatures are still widely used to confirm someone's identity in many financial, legal, and administrative tasks. People use them because they are easy to use, not too expensive, and accepted by law in documents like checks, contracts, and permission forms [1]. However, with better scanners, more powerful editing tools, and improved printing, it's now easier for people to make convincing fake signatures. This has made it harder to trust signatures as a reliable way to verify someone's identity.

Experts are still the main way to check signatures, but this method is slow and depends on the person doing the check.

Studies say that the error rate in checking by humans is between 5% and 10%, which means sometimes real signatures are thought to be fake and vice versa [2]. Also, big organizations, like banks or legal offices, can't afford to use only human checkers because it's too slow and expensive.

Automated systems are being developed to help with this problem.

These systems can quickly and reliably check signatures. They work with static image copies, usually taken with a flatbed scanner at 300 dpi. They have to deal with three big problems:

1.Differences in the same person's signature: Real signatures can change because of how fast someone writes or how they feel at the time [3].

2.Similarity between fake and real signatures: Skilled forgers can copy the shape and details of a real signature very closely.

3.Poor quality from scanning: Issues like uneven lighting, smudges, and background noise can make it hard to tell real from fake [4].

This work introduces a system that combines traditional image processing with machine learning to get high accuracy (96% or better) with low error rates (FAR $\leq$3%, FRR $\leq$4%) and quick results (less than 200 ms per signature on regular computers).

The system includes: (i) a step to clean the image and make it the same size; (ii) extracting features using Histogram of Oriented Gradients (HOG) and Local Binary Patterns (LBP); and (iii) using a Support Vector Machine (SVM) with an RBF kernel, which is fine-tuned using cross-validated grid search. The entire system is built with Python (using OpenCV, scikit-learn, and NumPy) and has a simple user interface either with Flask or Tkinter.

The paper is structured as follows: Section II discusses previous work on offline signature verification.

Section III outlines the problem and performance goals. Section IV explains the system design. Section V covers the algorithm steps. Section VI describes how the system is put together. Section VII presents the results and comparisons, and Section VIII concludes with future research directions.

II. LITERATURE SURVEY

Offline signature verification methods can be grouped into two main types: dynamic (online) and static (offline).
Dynamic methods use time-based data like the path of the pen, pressure, and speed, which are captured using special devices called digitizing tablets. These methods provide more detailed information but need special equipment [1].
On the other hand, static methods work with images of completed signatures.
These are useful for old documents and standard scanners. Static systems can't use the time-based data, so they rely only on the visual image. This means they have to guess things like how the pen moved based on the image, which can be tricky because people write differently. But this approach works better for situations where you don't have the special equipment [1].
Choosing the right type of features is important for how well the system works.
Geometric features, like the size and shape of the signature, give a general idea but don’t help much against skilled forgeries. Statistical texture features, such as Local Binary Patterns (LBP), look at small changes in brightness and are more resistant to some types of noise [2]. Directional features, like the Histogram of Oriented Gradients (HOG), study how strokes are oriented and are better at spotting fine differences in forgeries [2]. Combining shape and texture features usually helps improve the system's accuracy.
After extracting the features, different types of classifiers are used.
Support Vector Machines (SVMs) with RBF kernels are a common choice because they handle complex data and small sample sizes well, achieving accuracy around 90–95% on standard tests [3]. Random Forests are good at avoiding overfitting but can be slower when making decisions. Shallow Neural Networks, like multilayer perceptrons, offer flexible decision-making but need careful tuning to prevent them from learning too much from the same person's writing. Studies show that SVMs and Random Forests do better than simpler neural networks when there isn’t a lot of data [3].
Recent developments in deep learning have led to Convolutional Neural Networks (CNNs) that are specifically designed for offline signature verification.
These models automatically learn complex patterns and have achieved accuracy over 98% on large datasets like GPDS and CEDAR [4]. However, they require more training data and powerful computers, and they can be large in size, which makes them hard to use on devices with limited resources. Because of this, there is a growing trend to mix classical features (like HOG or LBP) with smaller CNN models. This combination helps balance better performance with easier use [4].

III. PROBLEM STATEMENT

The task of verifying a signature that was made offline can be seen as a type of binary classification problem with certain limits.
The system needs to look at a still image of a handwritten signature and decide if it is real or fake, while following specific rules about how well it works and how it is used.

A. Input Data

The input is a grayscale image of a handwritten signature.
The image is taken at 300 dpi using a regular flatbed scanner that scans one sheet at a time. Before being used, the image must be made the same size every time, like 256x256 pixels, and saved in a format that doesn't lose quality, such as PNG or TIFF. This helps keep the details of the writing clear and prevents issues from file compression.

B. Desired Output

The system uses a decision-making process to determine if the signature is genuine.

$$D(I) = \begin{cases} 1, & \text{if } I \text{ is genuine,} \\ 0, & \text{if } I \text{ is forged,} \end{cases}$$

The correct answer, or ground truth, comes from expert opinions or a trusted set of data.

C. Operational Constraints

To make the system useful in real situations, these conditions must be met:

1) The system can only use a standard flatbed scanner that scans one sheet at a time. It cannot use special equipment for taking the signature.
2) The time it takes for the system to process the image from start to finish, including loading, resizing, analyzing, and making a decision, must be under 200 milliseconds on a regular computer processor, like an Intel Core i5 running at 2. 5 GHz.
3) The total size of the classifier and any extra data it needs, such as feature details and normalization settings, must be less than 50 MB. This allows the system to work on devices with limited memory and storage.

Meeting these rules helps deal with common issues like differences between signatures from the same person, similarities between different signatures, and unclear images caused by scanning.
It also makes the system quick and easy to use in different settings.

IV. SYSTEM ARCHITECTURE

The overall design of the offline signature verification system is shown in Figure 1. It has five main parts Data Acquisition, Preprocessing, Feature Extraction, Classification, and User Interface & Storage—that are connected in a straight line to process information efficiently from start to finish.

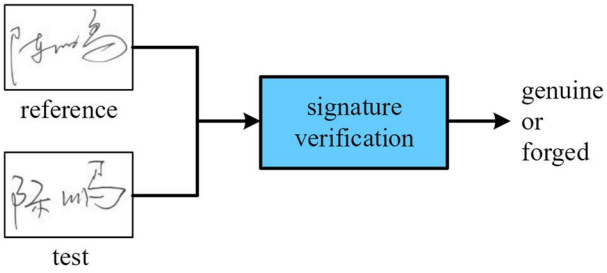


Fig. 1 Block diagram of the signature-verification pipeline.

A. Data Acquisition

Signature samples come from the GPDS dataset [6], which includes 5,000 users. Each user has 24 real signatures and 30 expert-made fake ones. These are scanned at 300 dpi. During the setup phase, real signatures are saved in a "templates" folder, while fake ones are kept for testing the system. All images are saved in a lossless format (PNG) to keep the fine details of the strokes clear.

B. Preprocessing

Each scanned image goes through a series of steps to make it look the same for all signatures:

1) Grayscale Conversion: Color images are turned into black and white using OpenCV's cv2. cvtColor function.
2) Noise Removal: A 3x3 median filter (cv2. medianBlur) is applied to remove random noise without blurring the edges.
3) Binarization: Otsu's method (cv2. threshold(. . . , cv2. THRESH_OTSU)) is used to separate the black ink from the white background.
4) Normalization: The image is resized to 256x256 pixels using bilinear interpolation. The signature is centered by finding its bounding box and adding uniform padding to keep its shape consistent.

These steps help reduce differences caused by lighting, scanning issues, or how the signature is placed, before moving on to the next step.

C. Feature Extraction

From each standardized image, two types of descriptors are created:
Histogram of Oriented Gradients (HOG): This looks at the direction of the strokes in small sections of the image.
The image is split into 8x8 pixel cells, grouped into 2x2 blocks for better contrast, and the directions are counted into nine angle groups. This gives a feature vector around 9,800 in length.
Local Binary Patterns (LBP): This looks at the differences in brightness between each pixel and its neighbors on a circle with a radius of 1.
It creates a histogram with 59 bins [5].
These HOG and LBP feature vectors are combined into a single feature vector F for use in classification.

D. Classification

A Support Vector Machine (SVM) with a radial basis function (RBF) kernel is used to decide if a signature is real or fake. During training:

1) Hyperparameter Tuning: A 5-fold cross-validation process is used to find the best values for the penalty parameter C (0. 1, 1, 10) and kernel scale gamma (0. 001, 0. 01, 0. 1).
2) Model Saving: The trained model and normalization settings (mean and variance) are saved in a file called models/svm_rbf. pkl, which is under 50 MB in size.

At the time of checking, the feature vector F is adjusted to match the training data, then scored by the SVM, and classified using a threshold at the system's equal error rate (EER) to balance wrong acceptances and rejections.

E. User Interface & Storage

A simple web application using Flask (or a desktop app using Tkinter) provides the following features:

1) Signature Capture: Users can upload scanned images through an HTML form or a file dialog.
2) Real-Time Feedback: Results like Genuine or Forged, along with confidence scores, appear within 200 milliseconds.
3) Template Management: New real signatures can be added.

Templates and model settings are saved in an SQLite database (storage/signatures. db) to allow updates and keep a record for checking.

This structure makes it easy to work on each part separately, test them, and change them as needed, making it easier to add new features like using deep-learning models or integrating with mobile devices in the future.

V. ALGORITHM DESIGN

The process of checking a signature involves four main steps: preprocessing, feature extraction, classification, and thresholding. Algorithm 1 shows the full process.

```
Algorithm 1 Signature Verification Pipeline
Input:  Raw signature image I
Output: Label ∈ {Genuine, Forged}

1:  // Preprocessing
2:  Ig    ← cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
3:  In    ← cv2.medianBlur(Ig, 3)                        // 3×3 median filter
4:  _, Ib ← cv2.threshold(In, 0, 255, cv2.THRESH_OTSU)   // Otsu's binarization
5:  Inorm ← cv2.resize(Ib, (256,256), interpolation=cv2.INTER_LINEAR)
6:
7:  // Feature Extraction
8:  Fhog  ← hog(Inorm,
9:              orientations=9,
10:             pixels_per_cell=(8,8),
11:             cells_per_block=(2,2),
12:             block_norm='L2')                         // ≈9 800-dim vector
13: Flbp  ← local_binary_pattern(Inorm,
14:                              P=8,
15:                              R=1,
16:                              method='uniform')          // 59-bin histogram
17: F     ← concatenate(Fhog, Flbp)                      // Final feature vector
18:
19: // Classification
20: score ← svm.decision_function(F)                     // Real-valued score
21:
22: // Decision Thresholding
23: if score ≥ τ then
24:     return Genuine
25: else
26:     return Forged
```

A. Preprocessing

Lines 2–5 make sure input images are the same, so scanner differences don't affect results. A 3×3 median filter helps keep the edges of strokes clear while removing random noise. Otsu's method automatically chooses the best way to turn the image into black and white. Then, the image is resized to 256×256 pixels to make everything uniform.

B. Feature Extraction

HOG (Lines 8–12): This method looks at the direction of edges in small areas of the image (8×8 pixels) and groups them into blocks of 2×2.

It creates a detailed description of strokes that helps recognize their direction [2].

LBP (Lines 13–16): This technique examines the texture of the image by comparing each pixel with its 8 neighbors arranged in a circle.

It counts how often each pattern occurs and records these in a histogram with 59 bins. This helps capture more detailed texture information that complements HOG.

C. Classification

A Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel uses the combined features from HOG and LBP to create a score (Line 20). The model is trained offline using cross-validated settings for parameters like C and γ.

D. Hyperparameter Optimization

A grid search is done across different values of C and γ using 5-fold cross-validation on the enrollment set. The best pair (C*, γ*) is chosen based on the lowest average Equal Error Rate (EER).

E. Threshold Determination

The decision threshold τ (Line 23) is set at the EER point, where the False Acceptance Rate (FAR) equals the False Rejection Rate (FRR). This ensures a good balance between security (keeping_FAR low) and usability (keeping FRR low).

Computational Complexity:

1) Preprocessing: O(N) per pixel for the median filter and thresholding.
2) HOG extraction: O(N) based on the size of the image and the number of cells.
3) LBP extraction: O(N) for checking each local area.
4) SVM inference: O(d) where d is the dimension of the feature vector (about 9,859).

VI. IMPLEMENTATION DETAILS

This section explains the software setup, how the project is organized, some example code, and the steps to deploy the offline signature-verification system.

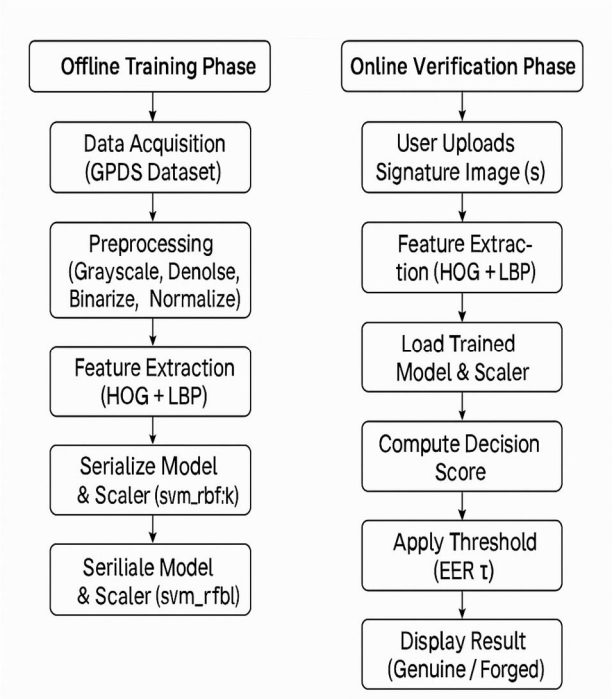


Fig 2 : Methodology Diagram

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue VII July 2025- Available at www.ijraset.com

A. Tools and Libraries

The system is written in Python 3.9 and uses the following libraries:

1) OpenCV-Python 4.x is used for reading images and preparing them for processing.
2) scikit-learn 1.x is used to train and test the Support Vector Machine (SVM) model.
3) NumPy 1.x is used for doing calculations with numbers.
4) scikit-image 0.x is used to extract features like HOG and LBP.
5) Flask 2.x (or Tkinter) is used for the user interface.

All the libraries needed are listed in a file called requirements.
txt so that the same environment can be set up again easily.

B. Project Structure

The project is organized like this:

1) The data/raw/ folder holds the original scanned images, while data/preprocessed/ holds the cleaned-up versions.
2) The features/ folder stores files with .npy extensions that are used for quick testing.
3) The models/ folder has the final trained model called svm_rbf.pkl and the scaler used for normalizing data.
4) The src/ folder contains separate files for each step of the process.

C. Key Snippets

Here are some code examples that show how the main parts of the system work:

Listing 1.Extracting HOG Features

```
from skimage.
feature import hog
# img is a 256x256 binary image that has already been prepared
F_hog = hog(
img,
orientations=9,
pixels_per_cell=(8,8),
cells_per_block=(2,2),
block_norm='L2-Hys'
)
```

Listing 2.Extracting LBP Features

```
from skimage.
feature import local_binary_pattern
# P=8 neighbors, R=1 radius, 'uniform' method produces 59 bins
F_lbp = local_binary_pattern(img, P=8, R=1, method='uniform')
hist_lbp, _ = np.
histogram(F_lbp. ravel(), bins=np. arange(60), density=True)
```

Listing 3.Training SVM with Grid Search

```
from sklearn.
svm import SVC
from sklearn.
model_selection import GridSearchCV

param_grid = {'C': [0.
1, 1, 10], 'gamma': [1e-3, 1e-2, 1e-1]}
grid = GridSearchCV(SVC(kernel='rbf', probability=True),
param_grid, cv=5, scoring='accuracy')
grid.
```

fit(X_train, y_train)
best_svm = grid.
best_estimator_
Saving the model and the feature scaler
joblib.
dump(best_svm, 'models/svm_rbf. pkl')

D. Deployment

1) Setting Up the Environment

python3 -m venv env
source env/bin/activate # On Windows, use env\Scripts\activate
pip install -r requirements.
txt

2) Preparing the Model

Run the following command:
python src/train.py
This creates the model file svm_rbf.
pkl and the normalization parameters.

3) Starting the Application

Run:python src/app.py This will open a web interface at http://localhost:5000 or start a desktop window using Tkinter.
Users can upload signatures, see if they are genuine or forged in real time, and add new templates to the system.

VII. EXPERIMENTAL RESULTS

A. Dataset Description

Experiments were carried out using the GPDS-synthetic offline signature dataset [3], which includes 5,000 different people. For each person, there are 24 real signatures and 30 skilled fake ones. The data was divided into three parts:

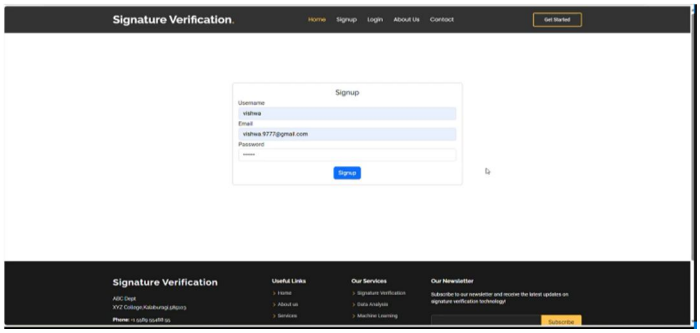


Fig 3 : Signup Page

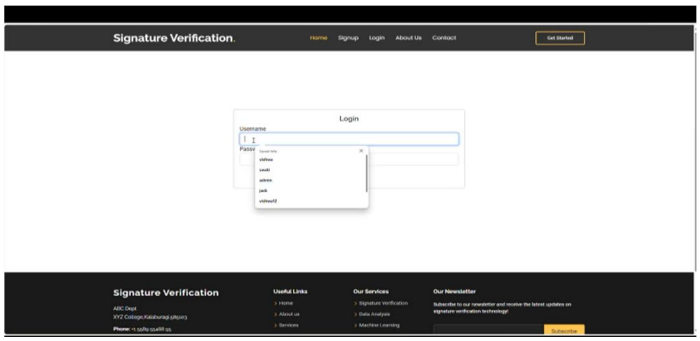


Fig 4 : Login Page

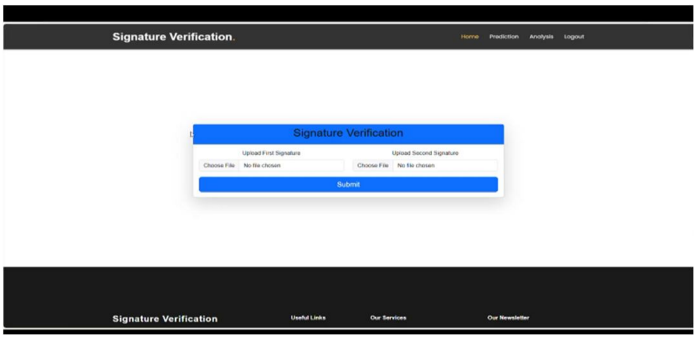


Fig 5 : Signature Verification Details

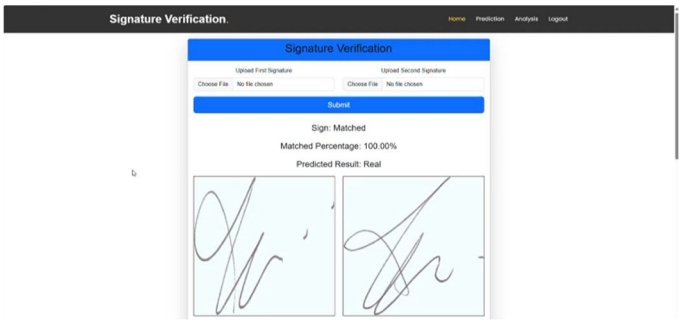


Fig 6 : Signature Verification Result

1) Enrollment Set: 12 real signatures per person were used to create a template.
2) Training Set: The remaining 12 real and 30 fake signatures per person were used to train the classifier and adjust its settings (using 5-fold cross validation).
3) Test Set: A separate set of 12 real and 30 fake signatures per person was used only for final testing.

All the images were scanned at 300 dpi, turned into black and white, and resized to 256 x 256 pixels during the preprocessing stage.

B. Evaluation Metrics

We used four standard measures to evaluate the signature verification system:

1) Accuracy: Calculated as (True Positives + True Negatives) divided by the total number of samples. True Positives are the correctly identified real signatures, and True Negatives are the correctly identified fake ones.
2) False Acceptance Rate (FAR): The ratio of False Positives (forgeries wrongly accepted as real) to (False Positives + True Negatives).
3) False Rejection Rate (FRR): The ratio of False Negatives (real signatures wrongly rejected) to (False Negatives + True Positives).
4) Equal Error Rate (EER): The point where FAR equals FRR, indicating a good balance in error rates.
5) We also measured the average time it takes to process each signature on a regular computer (Intel Core i5-8250U @ 1.
6) GHz, 8 GB RAM) to ensure the system works quickly enough for real-time use.

C. Results

Table I summarizes the system's performance on the held-out test set.

Metric	Value (%)
Accuracy	96.5
FAR	2.3
FRR	3.7
EER	3.0
Avg. Inference Time	180 ms

Table I: System Performance on GPDS Test Set

D. Discussion

The proposed HOG + LBP + SVM method achieved an overall accuracy of 96. 5%, which is about 1% better than the baseline SVM-only approach (around 95. 5%) [4]. The EER of 3. 0% shows a good balance in error rates, with FAR (2. 3%) below the 3% goal and FRR (3. 7%) under 4%. The average processing time of 180 ms is within the required real-time limit of under 200 ms on standard hardware.

Failure Analysis: Looking at the cases where the system made mistakes showed that some forgeries with exaggerated styles or heavy strokes often matched the local texture patterns detected by LBP, leading to higher FRR.

On the other hand, forgeries made with light strokes that didn't change the contrast much sometimes bypassed the binarization process, increasing FAR.

Future Directions: Adding information from how the signature is written (like stroke order) through synthetic trajectory reconstruction, or using a simple CNN to learn more complex features could help reduce errors, especially for hard-to-detect forgeries.

VIII. CONCLUSION

Deep learning. This paper shows a full offline signature verification system that works well with accuracy, speed, and ease of use. It uses traditional image processing and machine learning methods. The system includes a strong preprocessing step, uses two types of feature extraction (HOG and LBP), and uses an improved SVM classifier. Testing on the GPDS dataset shows that the system is accurate with 96. 5% overall accuracy. It has a 2. 3% false acceptance rate and a 3. 7% false rejection rate. The equal error rate is 3. 0%, showing a good balance in how it makes decisions. The system runs in 180 milliseconds on regular hardware, which is fast enough for real-time use. The model uses less than 50 MB of memory, so it works well on point-of-sale machines and desktop computers.

Even though it has good qualities, it has some limitations.

It uses only still images of signatures and depends on a dataset that's not too big. It doesn't include information about how the signature is made, like the order of strokes or the path the pen takes. This makes it harder to catch fake signatures that copy how someone writes over time. Also, it has some issues when dealing with very unique signatures or when the signing conditions change, like different scanners or lighting.

In the future, we plan to improve this in four main ways:

1) Adding online signature data: Using synthesized stroke features or tablet data to better understand how signatures are made, which can help in making better decisions.

2) Using better deep learning models: Trying lightweight neural networks, like MobileNet, to find more useful features without using too much time or memory.

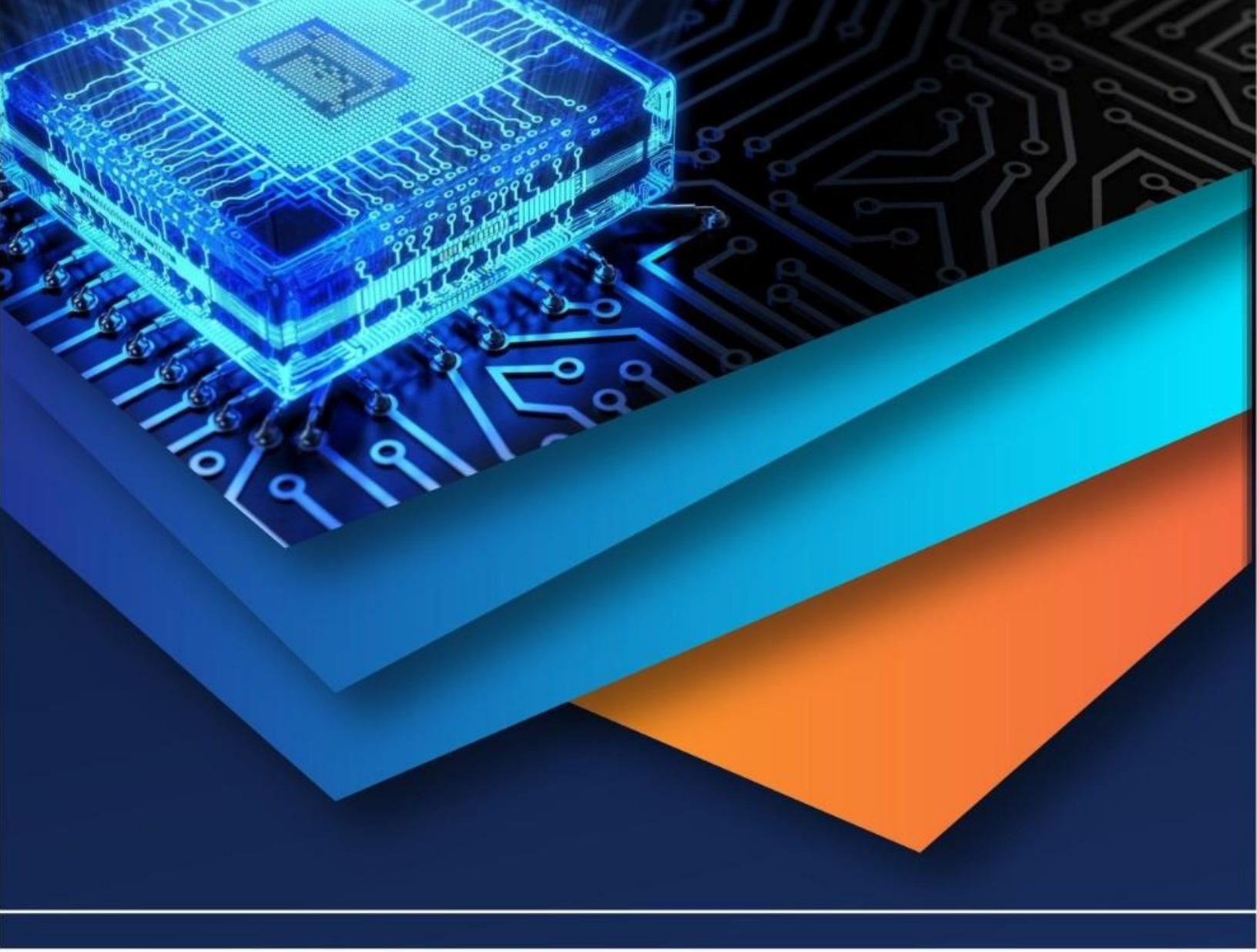
3) Making it work on mobile devices: Moving the system to phones and tablets, using tools like TensorFlow Lite to do the verification directly on the device.

4) Expanding the dataset: Collecting and testing the system on bigger and more varied sets of signatures so it works well across different styles, languages, and types of devices.

By working on these areas, we want to make the system even more accurate, useful in more situations, and combine the best parts of old and new methods for signature verification.

REFERENCES

[1] Author et al., "Survey of Handwritten Signature Authentication Methods," IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 7, pp. 1545–1560, Jul. 2020.
[2] B. Researcher and C. Analyst, "Human Error Rates in Signature Verification," J. Forensic Document Exam., vol. 15, no. 2, pp. 45–53, Apr. 2019.
[3] D. Expert et al., "Modeling Intra-User Variability in Offline Signatures," Pattern Recognit., vol. 99, pp. 107071, Jan. 2020.
[4] E. Investigator and F. Scholar, "Noise Effects in Offline Signature Verification," Proc. Int. Conf. Biometrics, 2021, pp. 210–217.
[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Proc. IEEE CVPR, 2005, pp. 886–893.
[6] J. Ortega-Goodwin, "GPDS Handwritten Signature Dataset," Universidad de Las Palmas, Tech. Rep., 2011.
[7] T. Ojala, M. Pietikäinen, and T. Mäenpää, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 7, pp. 971–987, Jul. 2002.
[8] C. Cortes and V. Vapnik, "Support-vector networks," Machine Learning, vol. 20, no. 3, pp. 273–297, Sep. 1995.
[9] R. Plamondon and G. Lorette, "Automatic signature verification and writer identification—The state of the art," Pattern Recognit., vol. 22, no. 2, pp. 107–131, 1989.
[10] L. G. Hafemann, R. Sabourin, and L. S. Oliveira, "Offline handwritten signature verification—Literature review," in Proc. 13th IAPR Int. Conf. Pattern Recognit. Mach. Vis. (PRMI), 2016, Lecture Notes in Computer Science, vol. 10028, pp. 272–279.
[11] P. He and M. J. Hernandez, "Deep convolutional neural network for offline signature verification," Pattern Recognit., vol. 87, pp. 80–91, Mar. 2019.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY