Midterm Project README

Document Forgery Detection

Laasya Chenchala(U12533113)

Prem Kumar (U49571671)

Nishitha Kaluvala(U83727321)

Signature Forgery Detection – README

Purpose, Aim, and Objective

Signature forgery detection is an advanced machine learning approach used to differentiate between genuine and forged signatures. This project utilizes **image processing and deep learning** techniques to analyze signatures and classify them as **authentic or forged**.

Every biometric authentication system consists of three key processes:

1. **Enrollment** – Capturing and storing genuine signatures.
2. **Identification** – Analyzing the signature features for comparison.
3. **Verification** – Comparing the input signature with the stored data for authenticity.

This project employs **image processing, feature extraction, and neural networks** to detect forged signatures. The system enhances, preprocesses, and classifies handwritten signatures using a **trained neural network model**.

Background of Project

Forgery detection is a critical aspect of document authentication in banks, legal institutions, and financial systems. Handwritten signature verification remains a commonly used biometric method. However, traditional manual verification is **error-prone** and **time-consuming**.

Challenges in Signature Forgery Detection:

- High similarity between genuine and forged signatures.
- Variations in a person's handwriting due to aging, stress, or writing conditions.
- Difficulty in manually distinguishing fine variations between real and fake signatures.

This project **automates** the verification process using deep learning models, significantly **reducing human errors** and improving authentication accuracy.

Scope of Project

This system is designed to:

- Process **handwritten signatures** and **enhance** them for better analysis.
- Extract **unique features** from the signature using image processing techniques.
- Train a **Neural Network model** to distinguish genuine signatures from forgeries.
- Allow real-time signature classification by predicting if a signature is **authentic (1) or forged (0).**
- Store processed signature images and extracted features for **further analysis**.

Modules Description

This project is composed of **five main modules**:

1. Signature Preprocessing (signprocessing.py)

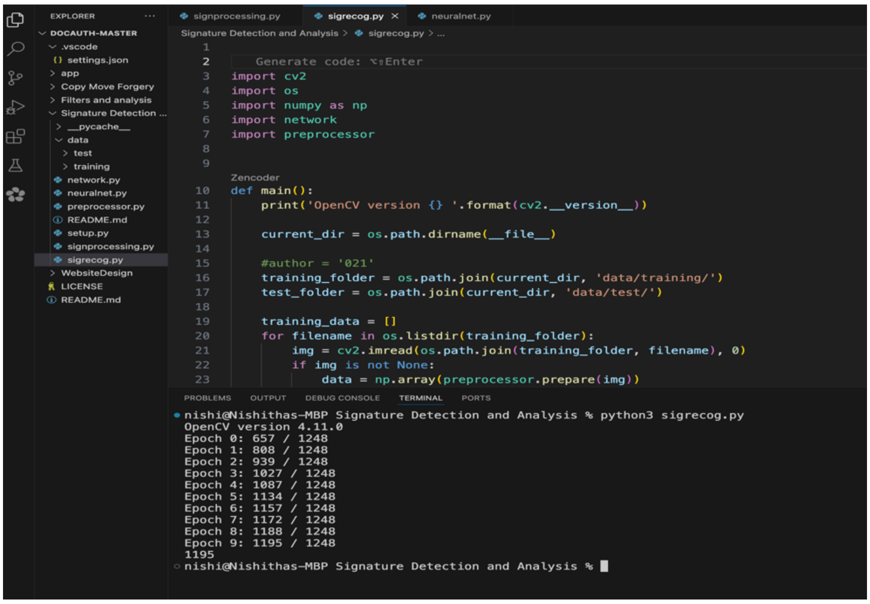
- Enhances contrast and brightness of the signature.
- Removes unnecessary background noise.
- Crops the signature to **focus on the relevant area**.
- Saves the processed signature for feature extraction.

2. Feature Extraction (preprocessor.py)

- Converts the **processed signature** into a numerical format.
- Extracts key **features** such as pixel distribution, aspect ratio, and intensity.
- Saves the extracted features as **NumPy arrays** for training.

3. Model Training (sigrecog.py / neuralnet.py)

- Uses a **Neural Network model** to classify signatures.
- Trains the model using both **genuine and forged signatures**.
- Evaluates the model's accuracy in signature verification.



4. Real-Time Prediction (predict.py)

- Selects an image from the training dataset for testing.
- Enhances, preprocesses, and extracts features from the input signature.
- Feeds the features into the trained neural network.
- Predicts whether the signature is **Genuine (1) or Fake (0)**.

5. Model and Data Storage

- Stores **processed signatures** and **extracted features** in organized folders.
- Saves the **trained model** (trained_model.pkl) for future use.
- Maintains logs for **prediction results**.

Software Requirements

The software setup requires the following dependencies:

Component	Details
Programming Language	Python
Operating Systems	Windows, Linux, Mac
IDE	Visual Studio Code, Jupyter Notebook, Sublime Text
Libraries Used	OpenCV, NumPy, PIL, TensorFlow/Keras, Matplotlib

Existing System and Drawbacks

Existing Methods

Traditional signature verification methods involve **manual inspection** or **biometric-based authentication systems** like:

- Visual comparison by experts.
- Pen pressure-based authentication.
- Optical character recognition (OCR).

Drawbacks of Existing Systems

- **Prone to human error** – Experts can misclassify signatures.
- **Time-consuming** – Manual verification takes time in large-scale systems.
- **Forgery detection limitations** – Simple OCR-based methods cannot detect sophisticated forgeries.

Proposed System

This project implements **an AI-based solution** that:

- **Automates signature verification** using **Neural Networks**.
- **Preprocesses** and **enhances** images for better accuracy.
- **Extracts features** and stores data efficiently.
- **Predicts** whether a signature is real or fake with **high accuracy**.

Advantages of the Proposed System

- **Fast & Efficient** – Automates the verification process, reducing human effort.
- **Secure & Reliable** – Uses deep learning to differentiate genuine and forged signatures.
- **Improved Accuracy** – Detects fine details in handwriting using feature extraction.
- **Scalable** – Can be implemented in **banks, legal firms, and government offices**.

How to Execute the Project

1. Install Required Libraries

Before running the project, install dependencies:

pip install numpy opencv-python Pillow tensorflow

2. Run Signature Preprocessing

To enhance and crop signature images:

python signprocessing.py

- This will process all raw signatures in **signatures/** and store them in **processed_signatures/**.

3. Extract Features

Convert processed images into feature vectors:

python preprocessor.py

- Extracted features will be stored in **features/**.

4. Train the Model

Train the neural network using the extracted data:

python sigrecog.py

- The model will be saved in **model/trained_model.pkl**.

5. Test a Signature

To classify a **random training signature** as genuine or fake:

python predict.py

6. View the Prediction

The output will show:

Prediction: Genuine (1)

or

Prediction: Fake (0)

Conclusion

This project provides a **robust, AI-based** method for **signature forgery detection**. It significantly **reduces human effort**, improves **authentication accuracy**, and ensures **secure document verification**.

Copy-Move Forgery Detection – README

Purpose, Aim, and Objective

Copy-move forgery is a common technique used in image manipulation, where a section of an image is copied and pasted elsewhere in the same image to **conceal or misrepresent** information. This project implements **copy-move forgery detection** using **image processing and machine learning techniques**.

The system detects **tampered regions** in an image by:

1. **Dividing the image into blocks** for detailed analysis.
2. **Extracting color, texture, and feature descriptors** from each block.
3. **Applying PCA for feature reduction** to efficiently compare blocks.
4. **Detecting duplicated regions** through similarity analysis.
5. **Reconstructing the forged areas** to highlight the tampered portions.

Background of Project

Image forgery detection is essential in **digital forensics, journalism, and security applications**. The copy-move forgery technique is particularly challenging to detect because **the tampered region originates from the same image**, making detection difficult using traditional approaches.

Challenges in Copy-Move Forgery Detection

- **Minor modifications** such as scaling, rotation, and compression can mask duplications.
- **High similarity between original and forged regions** requires robust feature extraction.
- **Computational efficiency** is critical for processing high-resolution images.

This project leverages **block-based feature extraction, Principal Component Analysis (PCA), and similarity checks** to efficiently detect forged areas.

Scope of Project

This system is designed to:

- **Analyze digital images** for signs of copy-move forgery.
- **Extract robust features** that remain effective even after minor modifications.
- **Detect duplicated regions** by comparing block features.
- **Generate tampering maps** to highlight forged areas.

- **Provide an automated pipeline** for large-scale image forgery detection.

Modules Description

This project is composed of **five main modules**:

1. Block-Based Feature Extraction (blocks.py)

- **Divides an image into fixed-size blocks** (e.g., 32x32 pixels).
- Extracts **color and intensity features** from each block.
- Applies **Principal Component Analysis (PCA)** to reduce dimensionality.

2. Feature Storage and Sorting (containers.py)

- Stores **extracted image blocks** and their **computed features**.
- Sorts **blocks based on similarity** for efficient comparison.

3. Image Processing & Forgery Detection (image_object.py)

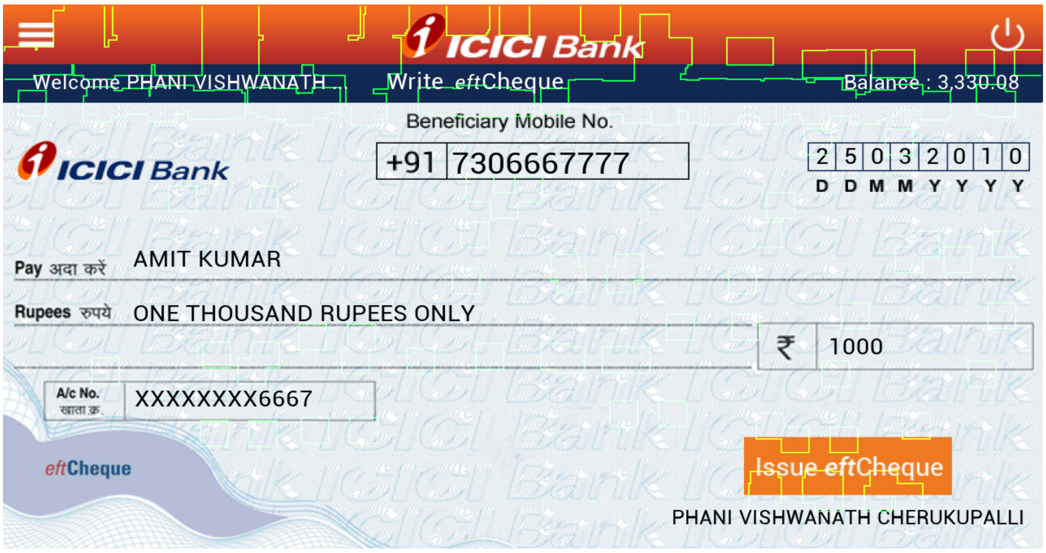
- **Loads and converts images** to grayscale if needed.
- **Extracts overlapping image blocks** and computes feature vectors.
- **Compares blocks** to identify **similar regions**.
- **Flags duplicated sections** as potential forgeries.

4. Main Detection Logic (copy_move_detection.py)

- Runs detection on **either a single image** (detect()) or an **entire folder** (detect_dir()).
- Calls ImageObject.ImageObject() to **process and analyze** each image.
- Stores **detection results** in the **output directory**.

5. CLI Execution & Image Preprocessing (main_cli_file.py)

- **Loads input images** and resizes them for processing.
- Calls CopyMoveDetection.detect() to **run forgery detection**.
- Outputs **forgery detection results**.



<u>Software Requirements</u>

The software setup requires the following dependencies:

Component	**Details**
Programming Language	Python
Operating Systems	Windows, Linux, Mac
IDE	Visual Studio Code, Jupyter Notebook, Sublime Text
Libraries Used	OpenCV, NumPy, PIL, tqdm, SciPy, scikit-learn

<u>Existing System and Drawbacks</u>

<u>Existing Methods</u>

Traditional forgery detection methods rely on **visual inspection** or **low-level image statistics**, which may not always detect sophisticated copy-move attacks.

Drawbacks of Existing Systems

- **Limited robustness** against scaling, rotation, or compression.
- **Slow processing times** for high-resolution images.
- **Difficulty in precisely locating forgeries** in complex backgrounds.

Proposed System

This project introduces an **automated pipeline** that:

- **Uses PCA-based feature extraction** to reduce data dimensionality while preserving essential patterns.
- **Efficiently compares image blocks** to identify similarities.
- **Visualizes tampered regions** by reconstructing the manipulated sections.

Advantages of the Proposed System

- **Automated & Efficient** – Eliminates the need for manual inspection.
- **Robust Feature Extraction** – Handles minor modifications like resizing or compression
- **Scalable for Large Datasets** – Can process multiple images automatically.
- **Tampering Map Visualization** – Clearly highlights forged areas for better understanding.

How to Execute the Project

https://datasetninja.com/cheque-detection#images – Dataset Used

1. Install Required Libraries

Before running the project, install dependencies:

pip install numpy opencv-python Pillow tqdm scipy scikit-learn

2. Run Forgery Detection on a Single Image

To detect copy-move forgery in one image:

python main_cli_file.py

- This analyzes test_images/Test-2.png and saves results in results/.

3. Run Detection on a Folder

To detect forgery in all images within a directory:

python -c "import CopyMoveDetection; CopyMoveDetection.detect_dir('test_images/', 'results/', 32)"

- This processes **all images in test_images/** and stores results in results/.

4. View the Detection Results

Forgery detection results will be saved in the **output directory**, displaying highlighted tampered regions.

Data Storage Structure

The following folders are used:

- project_folder/
- test_images/ # Input images
- processed_images/ # Preprocessed images
- features/ # Extracted feature vectors
- results/ # Output results with tampering maps
- model/ # Trained model (if applicable)
- blocks.py # Block-based feature extraction
- containers.py # Stores and sorts extracted features
- image_object.py # Main image processing and forgery detection logic
- copy_move_detection.py # Detects forgery on single or multiple images
- main_cli_file.py # Runs the forgery detection pipeline

Conclusion

This project provides a **robust, automated method** for **copy-move forgery detection**. By leveraging **PCA-based feature extraction and efficient block comparison**, it enhances **digital image authentication and forensic analysis**.