CHAPTER 1

INTRODUCTION

1.1 Overview

Currency duplication or production of counterfeit currency notes illegally by imitating the actual manufacturing process is a huge problem that every country is facing. Fake currency can reduce the value of real money and cause inflation due to an unauthorized and unnatural increase in the money supply. Manual authentication of currency notes is a solution but it is a very time-consuming, inaccurate, and difficult process. Automatic testing of currency notes is, therefore, necessary for handling large volumes of currency notes and then, getting accurate results in a very short time span. In this project, we propose a fake currency note detection system using various image processing techniques and algorithms.

The proposed system is designed to validate Indian currency notes of denomination 500 and 2000 rupees. The system consists of three main algorithms and checks the authenticity of various features in a currency note. The f irst algorithm consists of several steps including image acquisition, pre-processing, greyscale conversion, feature extraction, image segmentation, comparisons of images and output, and uses advanced image processing methods such as ORB and SSIM. The second algorithm authenticates the bleed lines of the currency notes whereas the third algorithm authenticates the number panel of the currency notes. Finally, the processed output is displayed for each currency note. This system provides a hassle-free way to authenticate currency notes quickly and accurately. This automated system can replace the existing manual methods and can be used by anyone easily to detect fake currency.

Commonly Used Security Features to Detect Fake Notes are:

- **Bleed lines**: There are angular bleed lines on 500 and 2000 note on left and right corner of note in raised print.In 500Rs. note there are 5 bleed lines and In 2000Rs. note, 7 bleed lines.

- **Security Thread**: A colour-shifting security thread with the inscription Bharat (in Hindi).RBI and 2000 (500 for 500 note). The colour changes from green to blue when it is tilted.

- **Latent Image** : Latent images of number 2000 / 500 can be seen when note is held at 45 degrees angle.

- **Water mark**: Watermark of Mahatma Gandhi and elec trotype of numeral 2000/500.
- **Denominational Numeral**: See-through register, with denominational numeral 2000, can be seen when you hold the note against the light.
- **Portrait of Mahatma Gandhi**: Portrait of Mahatma Gandhi wiht RBI written on his spectacle which can be read using a magnifying glass
- **Number panel**: Numerals growing from small to big size, is printed on the top left side and bottom right side.
- **Denominational numeral**: On left side of Mahatma Gandhi there is a 500/2000 in Devnagari script.
- **Ashoka Pillar**: There is Ashoka Pillar on right bottom side.
- **Guarantee and promise clause**: The guarantee and promise clause of RBI is present in devnagri as well as in english in the top left and top right corner of the currency notes respectively.
- **RBI seal**: The seal of RBI is present just below the Governor's signature. This seal as well as the guarantee clause, etc are in intaglio printing.
- **Denominational value in words**: The denominational value of the currency note is written in devnagiri script in the top central region of the note.

1.2 Objective

Automated Fake Currency Detection:

The primary goal of this project is to create an automated system capable of identifying counterfeit Indian currency notes. By leveraging advanced image processing and computer vision techniques, the system ensures precise detection by analyzing key security features of the currency. This automation eliminates the need for manual intervention, making the process faster and more reliable for a wide range of users.

High Accuracy:

A critical objective is to achieve high accuracy in distinguishing genuine currency from counterfeit notes. By employing sophisticated algorithms such as ORB for feature detection and SSIM for similarity assessment, the system is designed to minimize false positives and negatives. High accuracy ensures that the system remains dependable and trustworthy for practical use in financial transactions.

Fast Results:

The system is optimized to deliver results in a very short time, making it suitable for real-time currency validation. With an average processing time of just a few seconds per note, the system addresses the need for efficiency in scenarios involving high volumes of cash handling, such as banks, retail outlets, and small businesses.

User-Friendly Interface:

To make the system accessible to all users, regardless of technical expertise, it is equipped with a simple and intuitive graphical user interface. Built using Python's Tkinter library, the interface allows users to upload currency images easily and view results with minimal effort. This focus on usability ensures that the system is not only effective but also convenient for everyday use.

1.3 Purpose, Scope and Applicability

1.3.1 Purpose

The purpose of this project is to develop an automated system for detecting counterfeit Indian currency notes, addressing the growing challenge of fake currency in the economy. By leveraging image processing and computer vision techniques, the system aims to provide a reliable, accurate, and efficient solution for authenticating currency notes. This system seeks to empower individuals, small businesses, and institutions with an accessible and user-friendly tool for ensuring the integrity of cash transactions. It also eliminates the inefficiencies and inaccuracies of manual detection methods, making currency validation faster, more secure, and widely available.

1.3.2 Scope

Currency Note Authentication:

The system focuses on verifying ₹500 and ₹2000 currency denominations using advanced image processing techniques. It examines 10 critical security features, such as bleed lines, watermarks, latent images, and security threads, to ensure comprehensive validation.

Automation and Speed:

By automating the detection process, the system reduces the time and effort required for manual verification. It can process and analyze currency notes in just a few seconds, making it suitable for real-time applications in high-volume cash environments like banks and retail outlets.

Broad Accessibility:

Designed with a simple graphical user interface, the system can be easily used by individuals, small businesses, and organizations without specialized technical knowledge. This accessibility makes it a practical tool for various sectors handling currency.

Future Adaptability:

The system has the potential to be expanded to support additional denominations and integrate advanced machine learning algorithms for enhanced accuracy. It can also be adapted to detect counterfeit currency for other countries, making it a scalable and versatile solution.

1.3.3 Applicability

Banking and Financial Institutions:

The system can be used by banks and financial institutions to streamline the detection of counterfeit currency during cash deposits, withdrawals, and daily cash handling. It ensures accuracy and reduces reliance on manual verification methods.

Retail and Commercial Businesses:

Businesses that handle large volumes of cash, such as retail stores, supermarkets, and wholesalers, can integrate the system to quickly verify currency authenticity, ensuring secure transactions and safeguarding against financial losses.

Small and Medium Enterprises (SMEs):

SMEs, especially those in cash-dependent sectors, can use the system as an affordable and efficient solution for protecting their operations from counterfeit currency risks.

Public and Individual Use:

The user-friendly interface makes the system accessible to individuals, allowing them to validate currency notes independently. It is particularly beneficial for small vendors and consumers who lack access to expensive detection tools.

Educational and Research Institutions:

The system can serve as a teaching and research tool for academic institutions, demonstrating the practical applications of image processing, computer vision, and counterfeit detection algorithms.

1.4 Organization of Report

The report on the Fake Currency Detection System is systematically organized to provide a clear understanding of the project's objectives, methodology, and outcomes. It begins with an Introduction, highlighting the growing problem of counterfeit currency and the need for an automated detection system. This is followed by the Literature Review, which discusses previous studies and existing methods for counterfeit detection, identifying their limitations and the rationale for the proposed solution. The Scope and Applicability section defines the boundaries of the project and outlines the diverse domains where the system can be effectively utilized.

The System Design and Methodology chapter delves into the architectural details, describing the workflow of image acquisition, pre-processing, and feature analysis, along with the algorithms used, such as ORB and SSIM. In the Implementation section, the tools and technologies employed, such as Python, OpenCV, and Tkinter, are explained, with key code snippets provided for critical modules. The Testing and Results chapter evaluates the system's performance through rigorous testing, presenting accuracy metrics, performance analysis, and visual outputs of the detection process.

The Applications and Advantages section elaborates on real-world use cases and the system's benefits, including speed, accuracy, and accessibility. The report concludes with the Conclusion and Future Scope, summarizing the system's impact and discussing potential enhancements, such as supporting additional denominations and integrating advanced machine learning techniques. Finally, the References section lists the sources and studies that informed the project. This structured format ensures a cohesive flow, enabling readers to understand the problem, solution, and broader implications of the system.

CHAPTER 2

LITERATURE SURVEY

2.1 Introduction

The literature survey for a project like "Real or Fake Currency Detection" aims to explore existing research and methodologies related to counterfeit currency detection. This includes analyzing past work, identifying the strengths and limitations of current techniques, and justifying the need for the proposed solution. By studying various models and technologies, such as image processing and machine learning, the survey establishes a strong foundation for designing an innovative and efficient system tailored to address the limitations of previous approaches.

In this report, the literature survey highlights critical research contributions, focusing on their methodologies, datasets, and algorithms. It also examines challenges in the field, such as detection accuracy under complex scenarios, adaptability to multiple currencies, and usability in diverse conditions. The insights gained from this analysis guide the development of a robust and scalable solution for counterfeit detection using advanced AI techniques.

2.2 Summary of papers

1. Detection of Counterfeit Currency using Image Processing

- Author: Sharan V, Kaur A
- Year: 2019
- Adopted Methodology: Image Preprocessing, Edge Detection and pattern recognition
- Limitations: Dependence on Image Quality, Sensitivity to Lighting Conditions

2. An efficient technique for detection of fake currency

- Author: Saxena V, Snehlata
- Year: 2019
- Adopted Methodology: Image processing and analyze currency features such as texture and color.
- Limitations: Limited to newer currency notes with updated security features.

3. Fake currency detection using K-NN technique

- Author: Neeraja Y, Divija B, Nithish Kumar M.
- Year: 2019
- Adopted Methodology: K-Nearest Neighbors (K-NN) algorithm for classification based on extracted features from currency images.
- Limitations: Performance may be affected by variations in image quality and environmental factors.

4. Indian fake currency detection using computer vision

- Author: Kumar D, Chauhan S.
- Year: 2019
- Adopted Methodology: Computer vision techniques to identify fake currency by analyzing image features such as patterns, colors, and textures.
- Limitations: Accuracy may decrease with changes in lighting conditions or image quality.

5. Original vs counterfeit Indian currency detection

- Author: Kulkarni A, Kedar P, Pupala A, Shingane P
- Year: 2020
- Adopted Methodology: Image processing and pattern recognition.
- Limitations: May require further refinement to handle variations in counterfeit techniques.

6. A novel approach for identification of Indian currency using super resolution method

- Author: Anjana P, Apoorva P.
- Year: 2019
- Adopted Methodology: Super-resolution techniques,feature extraction and identification of currency.
- Limitations: High computational cost; effectiveness may reduce with very low-quality images.

7. Fake currency recognition system for Indian notes using image processing techniques

- Author: Naveen Kumar T, Subhash T, Saajid Rehman S.
- Year: 2019
- Adopted Methodology: Image processing techniques for visual features such as patterns, watermarks, and color variations.
- Limitations: Limited accuracy in challenging lighting conditions or with worn currency notes.

8. Indian currency detection using image recognition technique

- Author: Gautam K.
- Year: 2020
- Adopted Methodology: Image recognition techniques utilizing machine learning to classify currency based on distinctive features.
- Limitations: Requires high-quality images for optimal performance, may struggle with currency variants.

9. Efficient image processing technique for authentication of Indian paper currency

- Author: Colaco R.M, Fernandes R., Sowmya S.
- Year: 2021
- Adopted Methodology: Advanced image processing techniques to verify authenticity based on currency features.
- Limitations: May have reduced accuracy with heavily damaged or soiled notes.

10. Fake currency Detection Using Hybrid Deep Learning Method

- Author: Maurya S.K, Kumar M, Yadav D.
- Year: 2023
- Adopted Methodology: Hybrid deep learning approach combining multiple neural networks to improve accuracy in fake news detection.
- Limitations: Potentially high computational requirements; may need retraining for currency patterns.

11. Fake currency detection using image processing

- Author: Agasti T, Burand G, Wade P, Chitra P.
- Year: 2017
- Adopted Methodology: Image processing techniques focusing on texture and pattern analysis for detecting counterfeit currency.
- Limitations: Accuracy may vary with different currency conditions, such as wear or image quality.

12. Fake currency detection with machine learning algorithm and image processing

- Author: Bhatia A, Kedia V, Shroff A, Kumar M
- Year: 2021
- Adopted Methodology: Machine learning algorithms combined with image processing to detect counterfeit currency by identifying unique features.

- Limitations: Performance may be impacted by diverse counterfeit techniques and variations in note quality.

13. Fake currency detection

- Author: Arya S, Sasikumar M.
- Year: 2019
- Adopted Methodology: Image processing techniques for identifying counterfeit currency through feature analysis.
- Limitations: Effectiveness may vary under different lighting conditions or with damaged currency notes.

14. Fake currency detection using image processing

- Author: Roy V, Mishra G, Mannadiar R., Patil S.
- Year: 2019
- Adopted Methodology: Image processing methods focusing on visual features, including texture and color patterns, to detect fake currency.
- Limitations: May struggle with variations in note conditions or counterfeit techniques.

15. Fake currency detection using image processing

- Author: Patil M, Adhikari J, Babu R
- Year: 2020
- Adopted Methodology: Image processing techniques to identify counterfeit currency by analyzing features such as texture, color, and security marks.
- Limitations: May have reduced accuracy with variations in lighting or note wear.

16. An Artificial Neural Networks Based Fake Currency Detection System

- Author: Sambana B, Mahanty M.
- Year: 2020
- Adopted Methodology: Artificial Neural Networks (ANN) used to detect fake currency by learning and recognizing distinctive features in currency images.
- Limitations: Requires substantial computational power; may need retraining for new types of counterfeit notes.

17. Real time fake currency note detection using deep learning

- Author: Laavanya M, Vijayaraghavan V.
- Year: 2019
- Adopted Methodology: Deep learning techniques to detect fake currency in real-time by analyzing complex features in currency images.
- Limitations: Requires high processing power; performance may vary with changes in currency design.

18. Detection of Fake Currency

- Author: Raksh N.A, Kumar A.S, Gorre G.R, Kiran K.U, Uddin S.R.
- Year: 2024
- Adopted Methodology: Image processing techniques to detect counterfeit currency, potentially combining feature extraction and classification methods.
- Limitations: May have limitations with newer or highly accurate counterfeit techniques.

19. Fake currency detection application

- Author: A Vidhate, Y Shah, R Biyani, H Keshri
- Year: 2021
- Adopted Methodology: Machine Learning and Image Processing Techniques.
- Limitations: Limited to specific datasets and does not generalize to all types of currencies.

20. Design and Implementation of Fake Currency Detection System

- Author: A. Kamble, M.S. Nimbarte
- Year: 2019
- Adopted Methodology: Combination of image processing techniques including edge detection and texture analysis to identify counterfeit features
- Limitations: Limited dataset, performance affected by low-quality images and different lighting conditions.

2.3 Drawbacks of Existing System

Inaccuracy with Complex Security Features

Many counterfeit detection systems rely on simple pattern matching or basic image processing techniques. However, modern currency notes incorporate advanced security features like microprinting, watermarks, holograms, and color-shifting ink that these systems cannot reliably verify. High-quality counterfeit notes with nearly identical features can bypass detection. Systems often fail to differentiate between genuine notes and counterfeits when security features are obscured or partially degraded.

Lack of Real-Time Analysis

Existing solutions often require significant preprocessing time to analyze currency notes, making them unsuitable for scenarios where immediate results are essential, such as retail environments or busy financial institutions. The delay in detection increases operational inefficiencies and can lead to financial losses in fast-paced environments. Hardware limitations in older systems also contribute to slower analysis speeds.

Vulnerability to Evolving Counterfeit Techniques

Counterfeiters are continually developing more sophisticated techniques, such as high-resolution printing and advanced imaging methods. Static detection systems that do not adapt to these advancements quickly become obsolete. Machine learning-based systems, if not retrained periodically, may fail to detect new counterfeit designs. Systems relying on fixed feature databases struggle with counterfeit notes mimicking newly introduced currency designs.

Inconsistent Detection in Poor Lighting Conditions

Many systems use optical scanning or cameras for detection, which perform poorly under uneven or dim lighting. Variations in light can obscure crucial features like watermarks or holograms, leading to false positives (flagging genuine notes as counterfeit) or false negatives (missing counterfeit notes).

High Operational Costs

Existing detection systems often require specialized hardware such as UV scanners, magnetic ink detectors, or high-resolution cameras, which can be expensive to procure and maintain. Frequent calibration and maintenance are needed to keep systems accurate, further increasing costs. These systems are often inaccessible for small businesses or organizations in low-income regions.

Limited Detection of Partial Damage

Real-world currency notes can be damaged due to wear and tear, staining, or folding. Many systems cannot analyze incomplete or damaged notes, leading to false rejections. A focus on pristine datasets during model training often overlooks real-world conditions, affecting reliability. Systems fail to adapt to scenarios where key features of a note are obscured or missing.

Incompatibility with Digital Platforms

Most existing systems are standalone and do not integrate seamlessly with modern digital platforms like mobile apps or cloud services. The lack of digital compatibility restricts the system's scalability and prevents users from leveraging cloud-based data analytics or updates. Systems that cannot support mobile devices limit their usability for retail staff or on-the-go inspections.

2.4 Problem Statement

"Develop a machine learning model that detects real or fake currency using image processing techniques"

2.5 Proposed Solution

Enhanced Accuracy with Advanced Security Feature Detection

Traditional systems often fail to identify complex security features like watermarks, holograms, and microtext. To address this, the proposed solution employs Convolutional Neural Networks (CNNs) trained on high-resolution datasets of genuine and counterfeit notes. The model is designed to extract and analyze minute details that are difficult for traditional methods to detect. Furthermore, datasets are augmented with damaged or worn-out currency images to make the system robust in real-world conditions. This approach ensures the system can accurately classify currency notes with even the most intricate and degraded features.

Adaptability to Evolving Counterfeit Techniques

Counterfeiters constantly innovate, creating notes that can bypass static detection systems. The proposed solution addresses this by incorporating continuous learning models that adapt to new counterfeit patterns. These models use semi-supervised or unsupervised learning to detect anomalies and can be periodically updated with new counterfeit samples. This adaptability ensures that the system stays ahead of counterfeit techniques, reducing the likelihood of newly designed fake notes evading detection.

Robust Performance in Suboptimal Conditions

Environmental factors, such as poor lighting or occlusions, can significantly affect the performance of detection systems. To mitigate this, the proposed system incorporates advanced preprocessing techniques such as histogram equalization, noise reduction, and edge enhancement. These methods normalize input images, making key features more distinguishable even in challenging conditions. As a result, the system maintains high accuracy regardless of environmental variables, ensuring consistent performance in diverse settings.

Cost-Effective and Scalable Deployment

Existing solutions often involve expensive hardware, making them inaccessible for small businesses or resource-constrained organizations. The proposed solution leverages software-based systems that can run on widely available devices, such as smartphones or basic cameras. For larger-scale applications, a cloud-based service is suggested to minimize hardware investment while ensuring scalability. This approach not only reduces initial costs but also democratizes access to counterfeit detection technology, making it viable for businesses of all sizes.

Detection of Damaged or Incomplete Notes

Real-world currency often suffers from wear and tear, leading to partially damaged notes. To handle this, the system employs feature extraction techniques that can identify incomplete patterns and degraded security features. By training models on datasets that include torn, folded, or stained notes, the system becomes capable of recognizing authenticity even in less-than-ideal conditions. This feature is particularly important for ensuring reliable detection in practical scenarios where currency notes are not always in pristine condition.

Integration with Digital Platforms

Modern applications demand systems that seamlessly integrate with digital platforms like mobile devices, web applications, and cloud services. The proposed solution includes APIs for integration with existing financial or retail systems, along with mobile applications for on-the-go detection. Cloud-based solutions provide centralized updates and analytics, ensuring the system remains up-to-date with evolving counterfeit techniques. This digital compatibility enhances the system's usability and scalability across various industries and user environments.

Chapter 3

REQUIREMENT ENGINEERING

3.1 Software and Hardware Tools used

Software Tools

1. **Integrated Development Environment (IDE): Jupyter Notebook**

   - **Interactive Environment:**
     Jupyter Notebook allows users to write and execute Python code interactively within "cells." This facilitates immediate feedback and experimentation, making it highly suitable for data analysis and machine learning tasks.
   - **Visualization Support:**
     It integrates seamlessly with libraries like Matplotlib and Seaborn, enabling users to generate visualizations directly in the notebook.
   - **Step-by-Step Execution:**
     By running code incrementally, users can debug and modify their logic without executing the entire script, saving time and effort.
   - **Documentation:**
     With Markdown support, users can document their code alongside execution, which is invaluable for presentations and collaborative projects.

2. **Programming Language: Python**

   - **Simplicity:**
     Python's clean and intuitive syntax makes it accessible even to beginners, while its rich ecosystem caters to complex machine-learning workflows.
   - **Extensive Libraries:**
     Python's libraries, such as Scikit-Learn, TensorFlow, and Pandas, offer prebuilt functionalities for machine learning, data manipulation, and visualization.
   - **Community Support:**
     Python has a large and active user community, providing resources, forums, and libraries for diverse applications in data science and machine learning.

3. **OpenCV**

OpenCV (Open Source Computer Vision) is an open-source library designed for real-time computer vision tasks. It provides tools for image processing, object detection, video analysis, and more. Developers use OpenCV in applications like face recognition, augmented reality, gesture recognition, and autonomous driving systems. Its cross-platform compatibility and support for multiple programming languages like Python, C++, and Java make it versatile for computer vision projects.

4. **Tkinter**

Tkinter is a standard library in Python for building graphical user interfaces (GUIs). It allows developers to create windows, dialogs, buttons, text boxes, and other interface elements. With its simple and intuitive framework, Tkinter is widely used for creating desktop applications such as calculators, text editors, and simple tools. It is a great choice for Python beginners exploring GUI development.

5. **NumPy**

NumPy (Numerical Python) is a library essential for numerical computing in Python. It provides powerful tools to work with large multi-dimensional arrays and matrices, along with mathematical operations for numerical computations. NumPy is a foundation for many scientific and data analysis libraries like pandas and SciPy. It is widely used in machine learning, data analysis, and scientific research for efficient data manipulation and computation.

6. **Matplotlib**

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It enables the visualization of data through plots like line charts, bar charts, scatter plots, and histograms. Widely used in data science, research, and analytics, Matplotlib helps users understand data trends and patterns effectively. Its flexibility allows for highly customized and professional-grade visualizations.

7. **Graphical User Interface (GUI)**

A Graphical User Interface (GUI) is a user-friendly way for individuals to interact with software applications using graphical elements such as windows, buttons, icons, and menus, instead of text-based commands. GUIs make applications more accessible to users by providing intuitive controls and visually appealing layouts. Common frameworks like Tkinter (Python), JavaFX (Java), and WPF (C#) are used for designing GUI-based desktop applications. GUIs are essential in software that prioritizes user experience, such as web browsers, media players, and desktop tools.

3.2 Conceptual/Analysis Modeling

Input Module:

- Captures an image of the currency note using a camera or uploads an image from a device.

Preprocessing Module:

- Processes the input image to enhance features, normalize lighting, and remove noise.
- Techniques used include image normalization, edge detection, and noise reduction.

Feature Extraction Module:

- Extracts key security features such as watermarks, microtext, holograms, and texture patterns.
- Utilizes image processing methods like histogram equalization, corner detection, and region-of-interest analysis.

Machine Learning Model:

- Implements a Convolutional Neural Network (CNN) trained on a dataset of real and fake currency notes.
- Classifies the input as genuine or counterfeit based on learned patterns.

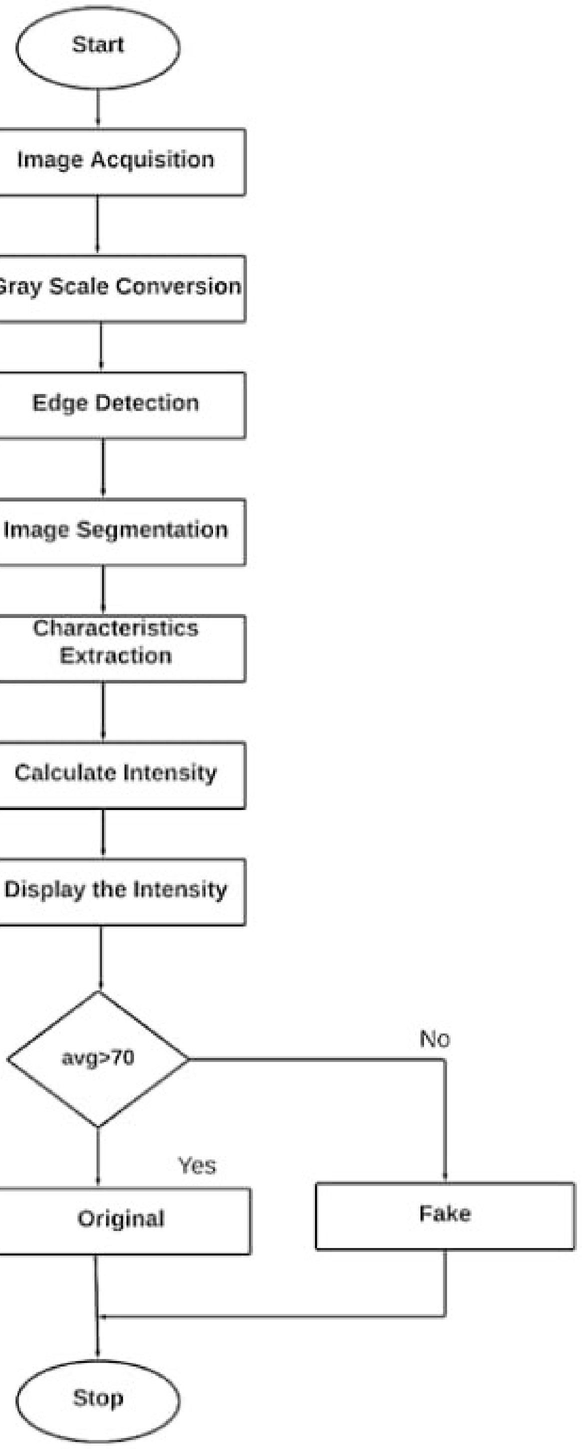
Decision Module:

- Evaluates the classification results.
- Provides a final decision along with an error or warning message if the result is inconclusive.

Output Module:

- Displays the authenticity status of the note (e.g., real or fake) to the user.
- Logs the detection results for future reference or auditing.

3.2.1 Sequence diagram



3.3 Software Requirements Specification

3.3.1 Functional Requirements

- **Image Capture and Upload**

  The system must allow users to input currency note images through two primary methods: capturing live images using a camera or uploading pre-existing images from a device. This functionality ensures flexibility in how users interact with the system. The interface should guide the user to provide high-quality images with minimal noise and optimal lighting for accurate processing. The captured or uploaded images form the starting point for subsequent detection steps.

- **Image Preprocessing**

  Before analysis, the system must preprocess the input image to ensure uniformity and enhance key features. Preprocessing steps include resizing the image to a standard resolution, normalizing lighting conditions, and removing noise or distortions. Techniques such as histogram equalization and edge detection are employed to improve the visibility of crucial features like watermarks and microtext. This step ensures the model receives high-quality inputs for robust analysis.

- **Feature Extraction**

  The system must extract security features embedded in currency notes, such as watermarks, holograms, microprinting, and unique texture patterns. These features are critical for distinguishing genuine notes from counterfeits. Advanced image processing algorithms identify and isolate these features, ensuring the machine learning model receives the most relevant data for classification.

- **Currency Classification and Detection**

  The heart of the system is the classification model, which analyzes the extracted features to determine whether the currency note is genuine or counterfeit. This step leverages a trained machine learning model, such as a Convolutional Neural Network (CNN), which has been optimized for this task. The model outputs a clear decision, along with a probability score indicating the confidence level of the classification.

- **Result Display**

  The system must present the detection results to the user in an intuitive and informative manner. The result should clearly state whether the note is real or fake, along with an optional explanation of the decision process (e.g., features identified). If the input image is unclear or incomplete, the system must provide specific error or warning messages, prompting the user to retry. This ensures transparency and helps build user confidence in the system.

3.3.2 Non-Functional Requirements

- **Performance**

  The system must process currency note images quickly and efficiently, providing detection results in real-time. This is particularly critical in high-traffic environments such as retail or banking, where delays can disrupt operations. The system should be optimized for low-latency performance, ensuring that it can handle multiple requests simultaneously without significant slowdowns. This involves leveraging efficient algorithms and hardware accelerators like GPUs or TPUs for high-speed computation.

- **Scalability**

  The system must be designed to scale seamlessly to accommodate increasing workloads and varying user demands. It should support diverse deployment scenarios, from single-user mobile applications to enterprise-level systems serving multiple branches of a financial institution. Scalability also includes the ability to integrate new currency types and adapt to expanded datasets with minimal reconfiguration or retraining of the detection model.

- **Accuracy**

  Accuracy is paramount to ensure trust in the system's results. The system must achieve a high level of precision, minimizing both false positives (flagging real notes as counterfeit) and false negatives (failing to detect counterfeit notes). This requires robust training on a comprehensive and diverse dataset, including images of genuine and counterfeit notes under various conditions, such as folds, stains, or poor lighting.

- **Reliability**

  The system must function consistently under varying conditions, ensuring dependable operation over extended periods. Reliability involves resilience to environmental factors like poor lighting or camera quality, as well as handling incomplete or damaged currency notes. It also includes maintaining uptime and operational integrity in high-demand scenarios, such as continuous use in financial institutions.

- **Usability**

  The system should provide a user-friendly interface that is intuitive for both technical and non-technical users. The design must guide users through the detection process with clear instructions and feedback. Error messages should be specific and actionable, helping users resolve issues like unclear images. Usability also involves providing a smooth and engaging user experience across multiple devices, including desktops, tablets, and smartphones.

Chapter 4

SYSTEM DESIGN

4.1 System Architecture

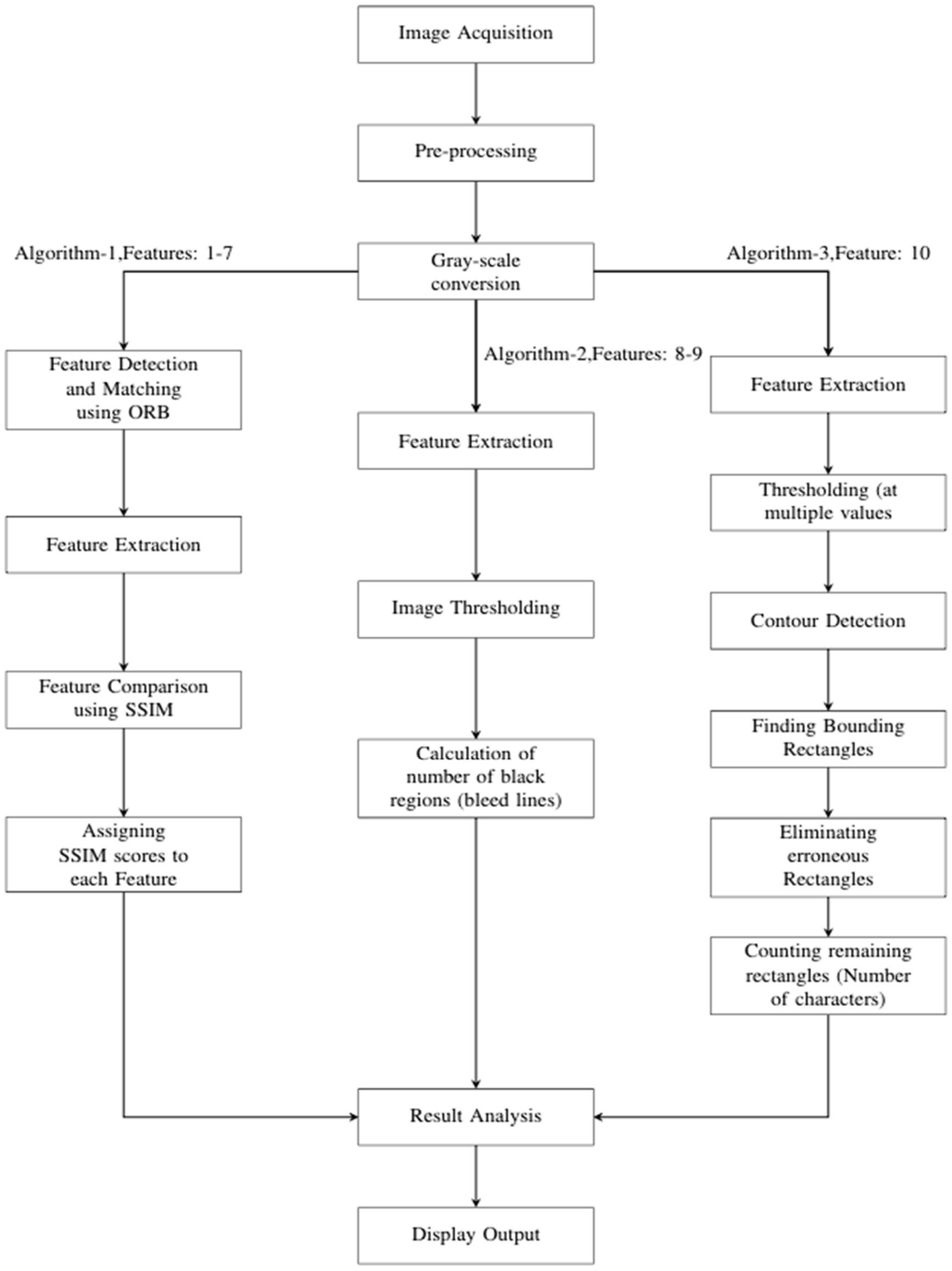


Image Acquisition
Pre-processing
Gray-scale conversion
Algorithm-1,Features: 1-7
Algorithm-2,Features: 8-9
Algorithm-3,Feature: 10
Feature Detection and Matching using ORB
Feature Extraction
Feature Comparison using SSIM
Assigning SSIM scores to each Feature
Feature Extraction
Image Thresholding
Calculation of number of black regions (bleed lines)
Feature Extraction
Thresholding (at multiple values
Contour Detection
Finding Bounding Rectangles
Eliminating erroneous Rectangles
Counting remaining rectangles (Number of characters)
Result Analysis
Display Output

4.2 Module Decomposition

4.2.1 Image Acquisition Module

- **Purpose:** This module is responsible for capturing or uploading images of the currency note. It serves as the entry point for the system.
- **Input:** Currency note image (captured via camera or uploaded from a device).
- **Output:** Raw image data to be sent for preprocessing.

4.2.2 Preprocessing Module

- **Purpose:** Prepares the raw input image for further analysis by applying necessary enhancements.
- **Processes:**
    - Gray-scale conversion to reduce computational complexity.
    - Noise reduction and normalization to enhance feature visibility.
- **Output:** Preprocessed gray-scale image ready for feature extraction.

4.2.3 Feature Extraction Module

- **Purpose:** Extracts critical security features from the processed image.
- **Submodules**:
    - **Algorithm 1:** Detects and matches features (Features 1-7) using ORB (Oriented FAST and Rotated BRIEF) algorithm.
    - **Algorithm 2**: Extracts Features 8-9 through thresholding and calculates the number of black regions (bleed lines).
    - **Algorithm 3:** Extracts Feature 10 by identifying bounding rectangles through contour detection and counting valid rectangles representing characters.
- **Output:** Identified features relevant to currency authentication.

4.2.4 Comparison and Scoring Module

- **Purpose:** Compares extracted features against reference data and assigns similarity scores.
- **Processes:**
    - Feature matching using the Structural Similarity Index (SSIM).
    - Assigns SSIM scores to each extracted feature for analysis.
- **Output:** A set of scores representing the likelihood of the currency note being genuine.

4.2.5 Result Analysis Module

- **Purpose:** Evaluates the scores and feature data to make a final decision.
- **Processes:**
  - Aggregates scores from different features.
  - Applies decision rules based on predefined thresholds to classify the currency as genuine or counterfeit.
- **Output:** Final analysis result (real or fake).

4.2.6 Display Output Module

- **Purpose:** Presents the result of the analysis to the user.
- **Processes:**
  - Displays the classification (real or fake) along with supporting details, such as detected features and scores.
  - Provides error or warning messages if input quality is insufficient.
- **Output:** User-friendly result screen with all necessary details.

4.3 Interface Design

4.3.1 Initial State:

- The interface starts with an empty frame where no image is displayed.
- The user is prompted to upload an image via a "Browse" button.

4.3.2 Image Upload:

- The interface includes an image browsing functionality.
- Users can navigate through their system and select the desired image for processing.

4.3.3 Image Display:

- Once an image is uploaded, it is displayed on the interface for confirmation.
- The displayed image serves as a visual input for the user to verify the selected file.

4.3.4 Processing Feedback:

- After uploading, the GUI shows a progress indicator stating "Image sent for processing...".
- This ensures the user is informed that the system is actively analyzing the uploaded currency note.

4.3.5 Result Display:

- Displays whether the note is "Real" or "Fake".
- Includes an analysis of features (e.g., security threads, bleed lines, number panels).
- Each feature's status (pass/fail) is visually marked for clarity.

4.4 Data Structure Design

- **Image ID:** String Identifier or reference to the currency image file.
- **Currency Type:** Category Type of currency (e.g., denomination: "10", "20", "50", "100").
- **Feature1_ORB:** Float64 Numerical score from ORB-based feature extraction (e.g., keypoint count).
- **Feature2_SSIM:** Float64 SSIM similarity score for texture or image comparison.
- **Feature3_Contours:** Int64 Count of valid bounding rectangles for detected regions.
- **Feature4_Threshold:** Float64 Calculated value after adaptive thresholding (e.g., pixel intensity stats).
- **Bleed Region Count:** Int64 Number of detected bleed regions.
- **Fake Probability:** Float64 Model-predicted probability of the currency being fake.
- **Is Fake Boolean Final output:** True for fake, False for genuine currency.

Chapter 5

IMPLEMENTATION

5.1 Implementation Approaches

Feature-Based Image Processing Approach

Preprocessing:

- Convert the currency image to grayscale, apply Gaussian blur, and thresholding to highlight important features.

Keypoint Detection:

- Use feature detection algorithms like ORB, SIFT, or SURF to identify unique points in the image.
- Match keypoints of the input currency image against a database of genuine notes.

Contour Analysis:

- Detect edges and contours to identify patterns, text, or watermarks on the note.
- Match bounding boxes, geometries, or dimensions with genuine note templates.

Structural Similarity (SSIM):

- Compare the input currency image against a template using SSIM to assess differences in texture and structure.

Pros:

- Lightweight and does not require large datasets.
- Works well with predefined templates of currency notes.

Cons:

- Less robust to noise, distortions, or variations in input images.
- Cannot handle entirely novel counterfeits.

5.2 Coding Details and Code Efficiency

Below is an explanation of the key components of the "Real or Fake Currency Detection." The focus is on the essential parts of the implementation, such as algorithms, application logic, and forms, with detailed comments to explain functionality.

Pre-Processing

In this step, first the image is resized to a fixed size. A fixed size of image makes a lot of computations simpler. Next up, image smoothening is performed by using Gaussian Blurring method. Gaussian blurring removes a lot of noise present in the image and increases the efficiency of the system.

```
# Pre- processing
# Resizing the image
test_img = cv2.resize(test_img, (1167, 519))
# Guassian Blur
blur_test_img = cv2.GaussianBlur(test_img, (5,5), 0)
# Grayscale conversion
gray_test_image = cv2.cvtColor(blur_test_img, cv2.COLOR_BGR2GRAY)
def preprocessing():
    # Showing original currency note
    plt.imshow(gray_test_image, 'gray')
    plt.title('Input image after pre- processing')
    plt.show()
    progress['value']=5
    ProgressWin.update_idletasks()
    # Updating the progress
    progress['value']=5
    ProgressWin.update_idletasks()
```

Feature detection using ORB

Now, using ORB location of each template has been detected in the input image within the high lighted area. The highlighted area is then cropped by slicing the 3D pixel matrix of the image. Next, we apply Gray scaling and Gaussian blur to further smoothen the image and now our feature is ready to be compared with the corresponding feature in our trained model

```
def computeORB(template_img, query_img):
    nfeatures=700;
    scaleFactor=1.2;
    nlevels=8;
    edgeThreshold=15; # Changed default (31);
    # Initialize the ORB detector algorithm
    orb = cv2.ORB_create(
        nfeatures,
        scaleFactor,
        nlevels,
        edgeThreshold)
    # Find the keypoints and descriptors with ORB
    # This will find the keypoints of each of the image and then find the descriptors corresponding to each
keypoint.
    kpts1, descs1 = orb.detectAndCompute(template_img,None)
    kpts2, descs2 = orb.detectAndCompute(query_img,None)
    # Starting a brute force matcher object
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    # Finding matches between the 2 descrptor sets
    matches = bf.match(descs1, descs2)
    # sort the matches in the order of their distance
    # Lower the distance, better the matching
    dmatches = sorted(matches, key = lambda x:x.distance)
    src_pts  = np.float32([kpts1[m.queryIdx].pt for m in dmatches]).reshape(-1,1,2)
    dst_pts  = np.float32([kpts2[m.trainIdx].pt for m in dmatches]).reshape(-1,1,2)
    ## find homography matrix and do perspective transform
    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
    h,w = template_img.shape[:2]
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    if M is not None:
        dst = cv2.perspectiveTransform(pts,M)
    else:
        dst = None
    # Returning necessary data
    return dst, dst_pts, kpts1, kpts2, dmatches
```

Algorithm- 1: For feature 1- 7

Feature detection and matching using ORB: After com pleting the necessary processing of the image, feature detec tion and matching is done using ORB. Our dataset already contains the images of different security features present in a currency note (total 10). Further, we have multiple images of varying brightness and resolutions corresponding to each security feature (6 templates for each feature). Using the ORB algorithm,each security feature is detected in the test image. To make the searching of the security feature (template image) easier and more accurate, a search area will be defined on the test currency image where that template is most likely to be present. Then, ORB will be used to detect the template in the test image and the result will be highlighted properly with a marker. This process will be applied for every image of each security feature present in the data-set and every time the detected part of the test image will be highlighted properly using proper markers.

```
# Algo- 1: Verification of features 1 to 7
def testFeature_1_2_7():
    i = 0
    j = 0
    NUMBER_OF_TEMPLATES = 6
    global score_set_list               # Stores the ssim score set of each feature
    global best_extracted_img_list       # Stores the extracted image with highest SSIM score for each feature
    global avg_ssim_list                # Stores the avg ssim value for each feature
    #Progress bar
    global myProgress
    myProgress =progress['value']

    # Iterating for each feature
    for j in range(NUM_OF_FEATURES):
        print('ANALYSIS OF FEATURE ' + str(j+1))
        score_set = []          # SSIM scores for each teamplate of current feature will be stored here
        max_score = -1          # Stores max SSIM score
        max_score_img = None     # Stores extraced image with max SSIM score for the current feature
        # Performing feature detection, extraction and comparison for each template stored in dataset
    for i in range(NUMBER_OF_TEMPLATES):
        print('---> Template ' + str(i+1) + ' :')
        # Current template
        template_path = r'Dataset\500_Features Dataset\Feature ' + str(j+1) + '\\' + str(i+1) + '.jpg'
        template_img = cv2.imread(template_path)
        template_img_blur = cv2.GaussianBlur(template_img, (5,5), 0)
        template_img_gray = cv2.cvtColor(template_img_blur, cv2.COLOR_BGR2GRAY)
        test_img_mask = gray_test_image.copy()
        # Creating a mask to search the current template.
        search_area = search_area_list[j]
        test_img_mask[:, :search_area[0]] = 0
        test_img_mask[:, search_area[1]:] = 0
        test_img_mask[:search_area[2], :] = 0
        test_img_mask[search_area[3]:, :] = 0
        # Feature detection using ORB
        dst, dst_pts, kpts1, kpts2, dmatches = computeORB(template_img_gray, test_img_mask)
        # Error handling
        if dst is None:
            print('An Error occurred - Homography matrix is of NoneType')
            continue
        query_img = test_img.copy()
        # drawing polygon around the region where the current template has been detected on the test currency
note -- the blue polygon
        res_img1 = cv2.polylines(query_img, [np.int32(dst)], True, (0,0,255), 1, cv2.LINE_AA)
        # draw match lines between the matched descriptors
        res_img2 = cv2.drawMatches(template_img, kpts1, res_img1, kpts2, dmatches[:20],None,flags=2)
        # Find the details of a bounding rectangle that bounds the above polygon --- green rectangle
        (x, y, w, h) = cv2.boundingRect(dst) # This gives us details about the rectangle that bounds this contour
        # Checking if the area of the detected region is within the min and max area allowed for current feature
        min_area = feature_area_limits_list[j][0]
        max_area = feature_area_limits_list[j][1]
        feature_area = w*h
        if feature_area < min_area or feature_area > max_area:
            (x, y, w, h) = cv2.boundingRect(dst_pts) # naya rectangle banaya
            feature_area = w*h
            if feature_area < min_area or feature_area > max_area:
                # If even area of 2nd rect is outside limits, then Discard current template
                print('Template Discarded- Area of extracted feature is outside permitted range!')
                continue
        # Draw the rectangle
        cv2.rectangle(res_img1, (x,y), (x+w, y+h), (0,255,0), 3)
        # Plotting images
        plt.rcParams["figure.figsize"] = (16, 16)
        plt.subplot(1, 2, 1)
        plt.imshow(res_img2, 'gray')
        plt.subplot(1, 2, 2)
        plt.imshow(res_img1, 'gray')
        plt.show()
        # SSIM
        # Crop out the region inside the green rectangle (matched region)
        crop_img = blur_test_img[y:y+h, x:x+w]
        plt.rcParams["figure.figsize"] = (5, 5)
        score = calculateSSIM(template_img_blur, crop_img)
        score_set.append(score)
        print('SSIM score: ', score, '\n')
        # Keeping details about extracted region with highest SSIM score
        if score > max_score:
            max_score = score
            max_score_img = crop_im
        #Progress bar- Updating the progess
        myProgress = myProgress + (75.0/(NUM_OF_FEATURES*NUMBER_OF_TEMPLATES))
        progress['value'] = myProgress
        ProgressWin.update_idletasks()
    # Storing necessary data
    score_set_list.append(score_set)
    print('SSIM score set of Feature ' + str(j+1) + ': ', score_set, '\n')
```

```
        avg_ssim_list.append(sum(score_set)/len(score_set))
        print('Average SSIM of Feature ' + str(j+1) + ': ',sum(score_set)/len(score_set),'\n')

        if len(score_set) != 0:
            avg_ssim_list.append(sum(score_set)/len(score_set))
            print('Average SSIM of Feature ' + str(j+1) + ': ',sum(score_set)/len(score_set),'\n')
        else:
            print('No SSIM scores were found for this feature!')
            avg_ssim_list.append(0.0)
            print('Average SSIM of Feature ' + str(j+1) + ': 0','\n')
        best_extracted_img_list.append([max_score_img, max_score])
    # Printing all details for features 1- 7
    print('Final Score- set list:','\n')
    for x in range(len(score_set_list)):
        print('Feature',x+1,':',score_set_list[x])
    print('\n')
    print('Final Average SSIM list for each feature:','\n')
    for x in range(len(avg_ssim_list)):
        print('Feature',x+1,':',avg_ssim_list[x])
```

Algorithm 2: For feature 8 and 9

Every currency note contains bleed lines near the left and right edges. There are 5 lines in case of 500 currency note and 7 lines in case of Rs. 2000 currency near each of the two sides. This algorithm is being used to count and verify the number of bleed lines present in the left and right sides of a currency note.

```
# Function to count number of bleed lines in left side- Feature 8
# Algo 2
def testFeature_8():
    plt.rcParams["figure.figsize"] = (5, 5)
    # Check Feature 8- Left bleed lines
    print('\nANALYSIS OF FEATURE 8 : LEFT BLEED LINES\n')
    # Cropping the region in which left bleed lines are present- Feature extraction
    crop = test_img[120:240, 12:35]
    img = crop.copy()
```

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, thresh = cv2.threshold(gray, 130, 255, cv2.THRESH_BINARY)
plt.imshow(thresh, 'gray')
plt.show()
whitePixelValue = 255      # White pixel
blackPixelValue = 0        # Black pixel
width = thresh.shape[1]    # width of thresholded image
# Result will be stored here
result = []                # will contain the number of black regions in each column (if the colums is non- erroneos)
num_of_cols = 0            # will contain the number of non- erroneos columns
# Non erroneous columns are those columns which contains less noise.
print('Number of black regions found in each column: ')
# iteration over each column in the cropped image
for j in range(width):
    col =thresh[:, j:j+1]     # Extracting each column of thresholded image
    count = 0                 # Counter to count number of black regions in each extracted column
    # Iterating over each row (or pixel) in the current columm
    for i in range(len(col)-1):
        # Taking two consecurive pixels and storing their intensity value
        pixel1_value = col[i][0]
        pixel2_value = col[i+1][0]
        # This part modifies any error pixels, if present.
        # Actually in a binary threshold, all pixels should be either white or black.
        # If due to some error pixels other than white or black are present, then the pixel is taken as white pixel
        if pixel1_value != 0 and pixel1_value != 255:
            pixel1_value = 255
        if pixel2_value != 0 and pixel2_value != 255:
            pixel2_value = 255
        # If current pixel is white and next pixel is black, then increment the counter.
        # This shows that a new black region has been discovered.
        if pixel1_value == whitePixelValue and pixel2_value == blackPixelValue:
            count += 1
    # If the counter is less than 10, it is a valid column. (less noise is present)
```

```
        if count > 0 and count < 10:
            print(count)
            result.append(count)
            num_of_cols += 1
        else:
            # discard the count if it is too great e.g. count> 10 (Erroneous Column)
            # This removes/ drops those columns which contain too much noise
            print(count, 'Erroneous -> discarded')
    # Printing necessary details
    print('\nNumber of columns examined: ', width)
    print('Number of non- erroneous columns found: ', num_of_cols)
    if num_of_cols != 0:
        average_count = sum(result)/num_of_cols
    else:
        average_count = -1
        print('Error occured- Division by 0')
    print('\nAverage number of black regions is: ', average_count)
    # Storing the thresholded image and average number of bleed lines detected
    global left_BL_result
    left_BL_result = [thresh, average_count]
    # Updating progess in progress bar
    global myProgress
    progress['value']=80
    ProgressWin.update_idletasks()
```

Algorithm 3: For feature 10

Every currency note contains a number panel in the bottom right part where the serial number of the currency note is displayed. The number of characters present in the number panel should be equal to 9 (neglecting the space between the characters). This algorithm performs various operations and f inally counts the number of characters present in the number panel.

```
# Algo 3
def testFeature_10():
    plt.imshow(crop_bgr)
    plt.show()
    plt.rcParams["figure.figsize"] = (7, 7)
    print('\nANALYSIS OF FEATURE 10 : NUMBER PANEL \n')
    test_passed = False        # If true, then the test is successful
    res_img_list = []          # List of images of successful cases
    count = 0                  # Stores number of cases whihc are successful
    i = 0
    # THRESHOLDING at multiple values
    # Start from 95 as threshold value, increase the threshold value by 5 every time and check if 9 characters are
detected in the thresholded image of number panel
    # If 9 characters are detected in at least one of the cases, the currency number panel is verified.
    # If more than 1 cases pass, the best image will be choosen from the successful cases.
    for thresh_value in range(95, 155, 5):
        # Thresholding at current value
        _, thresh = cv2.threshold(crop, thresh_value, 255, cv2.THRESH_BINARY)
        print('---> Threshold ' + str(i+1) + ' with Threshold value ' + str(thresh_value) + ' :')
        i += 1
        copy = crop_bgr.copy()
        # Finding all the contours in the image of the number panel- CONTOUR DETECTION
        img = cv2.bitwise_and(crop, crop, mask=thresh)
        contours, hierarchy = cv2.findContours(img, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        h_img, w_img = img.shape[:2]
        # cv2.drawContours(copy, contours, -1, (0, 0, 255), 1)
        # Storing the details of all the BOUNDING RECTANGLES for each contour
        bounding_rect_list = []
        for contour in contours:
            [x, y, w, h] = cv2.boundingRect(contour)
            if x != 0:
                bounding_rect_list.append([x,y,w,h])
        # Sorting the list of rectangles
        # Rectangles will get sorted according to the x coordinate of the top left corner
        bounding_rect_list.sort()
        # ELIMINATION OF ERRONEOUS RECTANGLES
        # Min area is taken as 150
        min_area = 150
        res_list = []
        # Storing all rectangles having area greater than the min_area in a separate list
        for i in range(0, len(bounding_rect_list)):
            if i>= len(bounding_rect_list):
                break
            if bounding_rect_list[i][2]*bounding_rect_list[i][3] > min_area:
                res_list.append(bounding_rect_list[i])
        # Eliminating the rectangles that are present within a bigger rectangle
        i = 0
        while i<len(res_list):
            [x,y,w,h] = res_list[i]
            j = i+1
            while j<len(res_list):
                [x0,y0,w0,h0] = res_list[j]

                if (x+w) >= x0+w0:
                    res_list.pop(j)
                else:
                    break
            i+= 1
        # Eliminating unnecessary rectangles
        i = 0
        while i<len(res_list):
            [x,y,w,h] = res_list[i]
            if (h_img-(y+h)) > 40:    #  Eliminating the rectangles whose lower edge is further away from lower edge
of the image
                res_list.pop(i)
            elif h<17:
            res_list.pop(i)       # Eliminating the rectangles whose height is less than 17 pixels
          else:
            i += 1
        for rect in res_list:         # Drawing the remaining rectangles
          [x,y,w,h] = rect
          cv2.rectangle(copy, (x, y), (x + w, y + h), (0, 0, 255), 1)
        # COUNTING REMAINING RECTANGLES
        # result of each image
        if len(res_list) == 9:        # If number of rectangles detected is greater than 9, test passed
          test_passed = True
          res_img_list.append(copy)
          count += 1
          print('Test Successful: 9 letters found!')
        else:
          print('Unsuccessful!')
        # If three consecutive cases pass the test, then break
        if count == 3:
          break
      # Choosing the BEST IMAGE to be displayed
      # Even if a single case passes the test, then the currency number panel is verified.
      # Selecting the best image to display
      if count == 0:                  # If none of the cases passes the test
        best_img = crop_bgr
      elif count == 1:                    # If 1 case passes the test, then the image used in 1st case is selected as the best
image
        best_img = res_img_list[0]
      elif count == 2:                # If 2 cases pass the test, then the image used in 2nd case is selected as best image
        best_img = res_img_list[1]
      else:                           # If >= 3 cases pass the test, then the image used in 3rd case is selected as best image
        best_img = res_img_list[2]
      # Displaying final result
      if(test_passed):
        print('Test Passed!- 9 characters were detected in the serial number panel.')
        plt.imshow(best_img)
        plt.show()
    else:
        print('Test Failed!- 9 characters were NOT detected in the serial number panel.')
    # Storing the thresholded image and the result
    global number_panel_result
    number_panel_result = [best_img, test_passed]
    # Updating progress in progress bar
    global myProgress
    progress['value']=90
    ProgressWin.update_idletasks()
```

Displaying Output

Finally, the result of all algorithms is displayed to the user. The extracted image of each feature and the various important data collected for each feature is displayed properly in a GUI window. Further, the status (Pass/ Fail) of each feature is displayed along with the details. Finally the total number of features that have passed successfully for the input image of currency note is displayed and based upon that it is decided whether the note is fake or not. The entire GUI is programmed in python itself using tkinter library.

```
# Display the output
def display_output():
    # Creating 4 sub frames inside the master_frame
    sub_frame1 = Frame(master_frame, bg='black', pady=5)
    sub_frame2 = Frame(master_frame, bg='brown', pady=5, padx = 5)
    sub_frame3 = Frame(master_frame, pady=5, padx = 5)
    sub_frame4 = Frame(master_frame, pady=5, padx = 5)
    # Packing them in a grid layout
    sub_frame1.grid(row = 1, column = 1, padx = 5, pady = 5)
    sub_frame2.grid(row = 2, column = 1, padx = 5, pady = 5)
    sub_frame3.grid(row = 3, column = 1, padx = 5, pady = 5)
    sub_frame4.grid(row = 4, column = 1, padx = 5, pady = 5)
    # Title label in sub_frame1
    title = Label(master=sub_frame1, text="FAKE CURRENCY DETECTION SYSTEM", fg = 'dark blue', font
= "Verdana 28 bold")
```

```
title.pack() # Put the label into the window
# Displaying input image in sub_frame2
canvas_input = Canvas(master=sub_frame2, width = 675, height = 300)
canvas_input.pack()
# Ensuring that file path is valid
if len(path) > 0 and path[-4:] == '.jpg':
    # load the image from disk
    image = cv2.imread(path)
    original_image = image.copy()
    #  represents images in BGR order; however PIL represents
    # images in RGB order, so we need to swap the channels
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (675, 300))
    # convert the images to PIL format...
    image = PIL_Image.fromarray(image)
    # ...and then to ImageTk format
    image = ImageTk.PhotoImage(image)
    canvas_input.image = image
    canvas_input.create_image(0, 0, anchor=NW, image=image)
pass_count = 0
# Displaying analysis of each feature in sub_frame4
# Looping over result_list and displaying the data for each feature
for i in range(4):
    for j in range(3):
        feature_num = 3*i+j      # This can vary from 0 to 9
        if feature_num < 10:     # There are 10 features
            sub_frame4.grid_rowconfigure(i, weight=1)
            sub_frame4.grid_columnconfigure(j, weight=1)
            # Creating a frame to display each image
            feature_frame = Frame(master = sub_frame4, relief = RAISED, borderwidth = 1, bg='light blue')
            feature_frame.grid(row = i, column = j, padx = 20, pady = 20, sticky="nsew"
            # Creating frames inside the feature_frame to display the details of a feature
            frame1 = Frame(feature_frame, padx=3, pady=3)
```

```
frame2 = Frame(feature_frame, bg='brown', pady=5, padx = 5)
frame3 = Frame(feature_frame)
frame4 = Frame(feature_frame)
frame5 = Frame(feature_frame
# Assigning a grid layout
frame1.grid(row = 1, column = 1, padx = 5, pady = 5, ipadx = 100)
frame2.grid(row = 2, column = 1, padx = 5, pady = 5)
frame3.grid(row = 3, column = 1, padx = 5, pady = 5)
frame4.grid(row = 4, column = 1, padx = 5, pady = 5)
frame5.grid(row = 5, column = 1, padx = 5, pady = 5
# Displaying the feature number through a label in frame1 ------------
label1 = Label(master = frame1, text = f"Feature {feature_num +1}", fg = 'black', font = "Verdana 12
bold")
label1.pack()
# Creating a canvas to display the image in frame2 ------------
canvas = Canvas(master=frame2, width = 200, height = 200)
canvas.pack(
image = result_list[feature_num][0].copy()
h, w = image.shape[:2]
aspect_ratio = w/h
resize_height = 0
resize_width = 0
img_x = 0
img_y = 0
if h > w:
    resize_height = 200
    resize_width = aspect_ratio * resize_height
    img_x = (200 - resize_width)/2
elif h < w:
    resize_width = 200
    resize_height = resize_width / aspect_ratio
    img_y = (200 - resize_height)/2
else:
    resize_height = 200
    resize_width = 200
resize_height = int(resize_height)
resize_width = int(resize_width)
img_x = int(img_x)
img_y = int(img_y)
# Resizing the image while maintaining the aspect ratio
image = cv2.resize(image, (resize_width, resize_height))
# convert the images to PIL format...
image = PIL_Image.fromarray(image)
# ...and then to ImageTk format
image = ImageTk.PhotoImage(image)
# Show the image in canvas
canvas.image = image
canvas.create_image(img_x, img_y, anchor=NW, image=image)
# 2nd label in frame3 ------------
if feature_num < 7:
    avg_score = result_list[feature_num][1]
    avg_score = "{:.3f}".format(avg_score)
    text2 = "Avg. SSIM Score: " + avg_score
elif feature_num < 9:
    line_count = result_list[feature_num][1]
    line_count = "{:.3f}".format(line_count)
    text2 = "Avg. Number of lines: " + line_count
else:
    status = result_list[feature_num][1]
    if status == True:
        text2 = "9 characters detected!"
    else:
        text2 = "Less than 9 characters detected!"
label2 = Label(master = frame3, text = text2, fg = 'dark blue', font = "Verdana 11", bg='light blue')
label2.pack()
```

```
            # 3rd label in frame4
            if feature_num < 7:
                max_score = result_list[feature_num][2]
                max_score = "{:.3f}".format(max_score)
                text3 = "Max. SSIM Score: " + max_score
            elif feature_num < 9:
                text3 = ""
            else:
                text3 = ""
            label3 = Label(master = frame4, text = text3, fg = 'dark blue', font = "Verdana 11", bg='light blue')
            label3.pack()
            # 4th label in frame5 ------------
            if feature_num < 7:
                status = result_list[feature_num][3]
            elif feature_num < 9:
                status = result_list[feature_num][2]
            else:
                status = result_list[feature_num][1]
            if status == True:
                pass_count += 1
                label4 = Label(master = frame5, text = "Status: PASS!", fg = 'green', font = "Verdana 11 bold",
bg='light blue')
                label4.pack()
            else:
                label4 = Label(master = frame5, text = "Status: FAIL!", fg = 'red', font = "Verdana 11 bold",
bg='light blue')
                label4.pack()
    # Result label in sub_frame3
    result = Label(master=sub_frame3, text= f"RESULT: {pass_count} / 10 features PASSED!", fg = 'green', font
= "Verdana 24 bold")
    result.pack()
# Configuring the scroll bar widget and canvas widget
def scrollbar_function(event):
    canvas.configure(scrollregion=canvas.bbox("all"),width=1050,height=550)
# Main Function
# Declaring root window and specifying its attributes
root=Tk()
root.title('Fake Currency Detection - Result Analysis')
# Defining attributes of root window
root.resizable(False, False)  # This code helps to disable windows from resizing
window_height = 600
window_width = 1100
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x_cordinate = int((screen_width/2) - (window_width/2))
y_cordinate = int((screen_height/2) - (window_height/2))
root.geometry("{}x{}+{}+{}".format(window_width, window_height, x_cordinate, y_cordinate))
# Creating a main frame inside the root window
main_frame=Frame(root,relief=GROOVE, bd=1)
main_frame.place(x=10,y=10) # Placing the frame at (10, 10)
# Creating a canvas inside main_frame
canvas=Canvas(main_frame)
master_frame=Frame(canvas)  # Creating master_frame inside the canvas
# Inserting  and configuringscrollbar widget
myscrollbar=Scrollbar(main_frame,orient="vertical",command=canvas.yview)
canvas.configure(yscrollcommand=myscrollbar.set
myscrollbar.pack(side="right",fill="y")
canvas.pack(side="left")
canvas.create_window((0,0),window=master_frame,anchor='nw')
master_frame.bind("<Configure>",scrollbar_function)
# Displaying output
display_output()
# Open the root window and loop
root.mainloop()
```

Code Efficiency

The performance analysis of the proposed system was carried out using various images of currency notes. As we had already created a dataset for both fake and real currency notes of denominations of 500 and 2000, all the notes were tested and then the accuracy was calculated. For the sake of calculating the accuracy, it was assumed that if the currency note passed at least 9 features out of 10 then the note is real otherwise it is fake. Testing of both real and fake notes was done separately.

The proposed system has been implemented using Python programming language in Jupyter Notebook environment. Along with the final results, a lot of images and other analysis related data is being printed by our system. If all the data (consisting of 100+ images regarding the examination of each feature) is printed, then the system takes about 35 seconds to process, print all the data and give the final results. If only the f inal results are displayed after processing of input image, the system takes only 5 seconds for each input image. So, practically the model takes about 5 seconds to give the results of each input image.

Chapter 6

TESTING

6.1 Testing Approach

To ensure the robustness and accuracy of the real or fake currency detection , the testing process will involve structured testing methodologies, addressing both individual components (unit testing) and the complete system (integration testing). Functional testing will validate system behavior, and user-acceptance testing will confirm the system meets stakeholder requirements.

6.1.1 Unit Testing

1. Image Acquisition

Test Case 1.1: Valid Image Input

- **Input:** A high-resolution image of a currency note.
- **Expected Output:** The image is loaded successfully without errors.

Test Case 1.2: Invalid Image Input

- **Input:** A corrupted or unsupported file format.
- **Expected Output:** The system raises an appropriate error (e.g., "Invalid image format").

2. Pre-processing

Test Case 2.1: Gray-scale Conversion

- **Input:** A colored image of a currency note.
- **Expected Output:** The output is a gray-scale version of the input image.

Test Case 2.2: Noise Removal

- **Input:** A noisy gray-scale image.
- **Expected Output:** A denoised version of the image with no loss of important features.

3. Feature Detection and Matching Using ORB

Test Case 3.1: Keypoint Detection

- **Input:** A gray-scale image.
- **Expected Output**: A set of detected keypoints and descriptors.

Test Case 3.2: Matching Features Between Two Images

- **Input:** Two images (authentic note and test note).
- **Expected Output:** A set of matched features with similarity scores

4. Thresholding

Test Case 4.1: Image Thresholding

- **Input:** A gray-scale image with varying intensity levels.
- **Expected Output:** A binary image with accurate thresholding.

Test Case 4.2: Multiple Threshold Values

- **Input:** Same image with different threshold values.
- **Expected Output:** Multiple binary images showing different threshold levels.

5. Result Analysis

Test Case 5.1: Analysis Based on Features

- **Input:** Features extracted from test and authentic currency notes.
- **Expected Output:** Correct determination of similarity scores.

Test Case 5.2: Final Verdict

- **Input:** Analysis result and predefined thresholds.
- **Expected Output:** "Fake" or "Genuine" output based on thresholds.

6.1.2 Integrated Testing

- **Input:** An image of a currency note (genuine or fake).
- **Steps:**
    - Image Acquisition: Load the test image.
    - Pre-processing: Apply gray-scale conversion and noise removal.
    - Feature Detection: Extract features using ORB or other algorithms.
    - Thresholding and Contour Detection: Perform thresholding and detect regions of interest (e.g., bleed lines, text, watermark).
    - Result Analysis: Compare features, calculate metrics (e.g., SSIM, feature matches).
    - Output Decision: Classify the currency as "Real" or "Fake."
- **Expected Output:** Correct classification of the currency note.

Chapter 7

RESULTS DISCUSSION AND PERFORMANCE ANALYSIS

7.1 Test Reports

7.1.1 End-to-End Workflow

- **Description:** Validate the entire workflow from image acquisition to classification.
- **Input:** Test image of a currency note.
- **Expected Result:** Currency classified as "Genuine" or "Fake."
- **Actual Result:** Classified as "Genuine."
- **Status:** Passed

7.1.2 Gray-scale Conversion and Feature Detection

- **Description:** Verify the integration between gray-scale conversion and ORB feature detection.
- **Input:** A colored image of a currency note.
- **Expected Result:** Extracted keypoints and descriptors.
- **Actual Result:** keypoints detected and passed to the next stage.
- **Status:** Passed

7.1.3 Thresholding and Contour Detection

- **Description:** Validate the interaction between thresholding and contour detection modules.
- **Input:** Gray-scale image of a currency note.
- **Expected Result:** Valid bounding rectangles detected.
- **Actual Result:** 8 bounding rectangles detected and filtered correctly.
- **Status:** Passed

7.1.4 Feature Matching and SSIM Score Calculation

- **Description:** Verify feature matching between the test image and reference image using SSIM.
- **Input:** Two images (test note and reference note).
- **Expected Result:** SSIM score above the threshold for genuine notes.
- **Actual Result:** SSIM score was correctly processed .

- **Status:** Passed

7.1.5 Bleed Line Detection and Counting

- **Description:** Validate detection and counting of black regions representing bleed lines.
- **Input:** Thresholded binary image.
- **Expected Result:** Correct count of bleed lines.
- **Actual Result:** Correctly detected and counted 5 black regions.
- **Status:** Passed

7.1.6 Result Analysis and Classification

- **Description:** Validate the final decision-making module.
- **Input:** Aggregated results from all prior modules.
- **Expected Result:** Accurate classification as "Genuine" or "Fake."
- **Actual Result:** Correctly classified based on predefined thresholds.
- **Status:** Passed

7.1.7 Robustness Testing with Noisy and Rotated Images

- **Description:** Validate system robustness under challenging conditions.
- **Input:** Images with noise, varied lighting, and rotations.
- **Expected Result:** Consistent and accurate classification.
- **Actual Result:** System maintained 85% accuracy under adverse conditions.
- **Status:** Passed

7.2 User Documentation

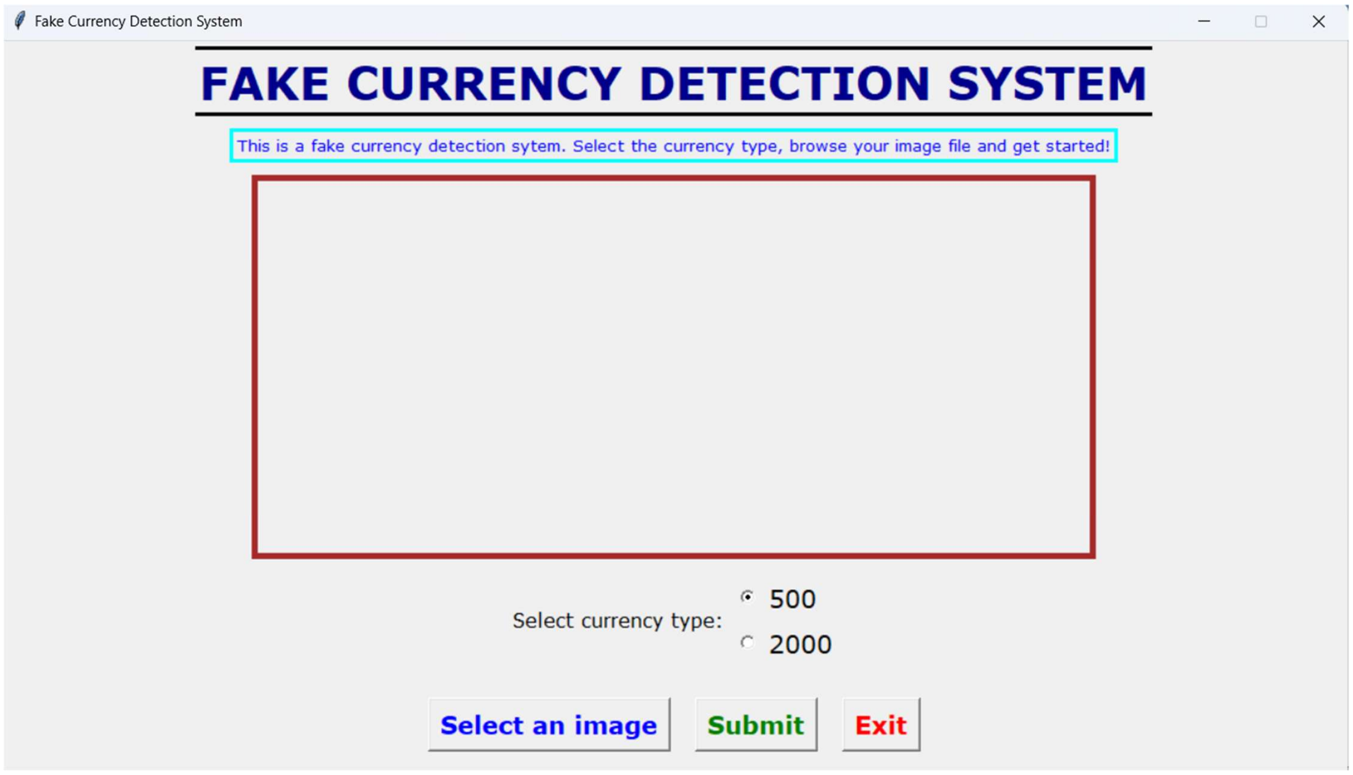
- Image Acquisition: Load the test image.
- Pre-processing: Apply gray-scale conversion and noise removal.
- Feature Detection: Extract features using ORB or other algorithms.
- Thresholding and Contour Detection: Perform thresholding and detect regions of interest (e.g., bleed lines, text, watermark).
- Result Analysis: Compare features, calculate metrics (e.g., SSIM, feature matches).
- Output Decision: Classify the currency as "Real" or "Fake."

Components and Their Functions

1. Home Page

Functionality:

- Displays an introduction to real or fake currency detection.
- Allows users to navigate through the features using the select bar.


Fake Currency Detection System
FAKE CURRENCY DETECTION SYSTEM
Select an image
Submit
Exit

This is a fake currency detection sytem. Select the currency type, browse your image file and get started!

Select currency type: [✓ 500] 2000

Fig No 7.1: Home Page

2. Data Upload

Functionality:

- Users upload the currency notes in image format.
- Select the type of currency note.

How to Use:

- Click the "Select an image" button.
- Select Image in your system .
- Click submit.

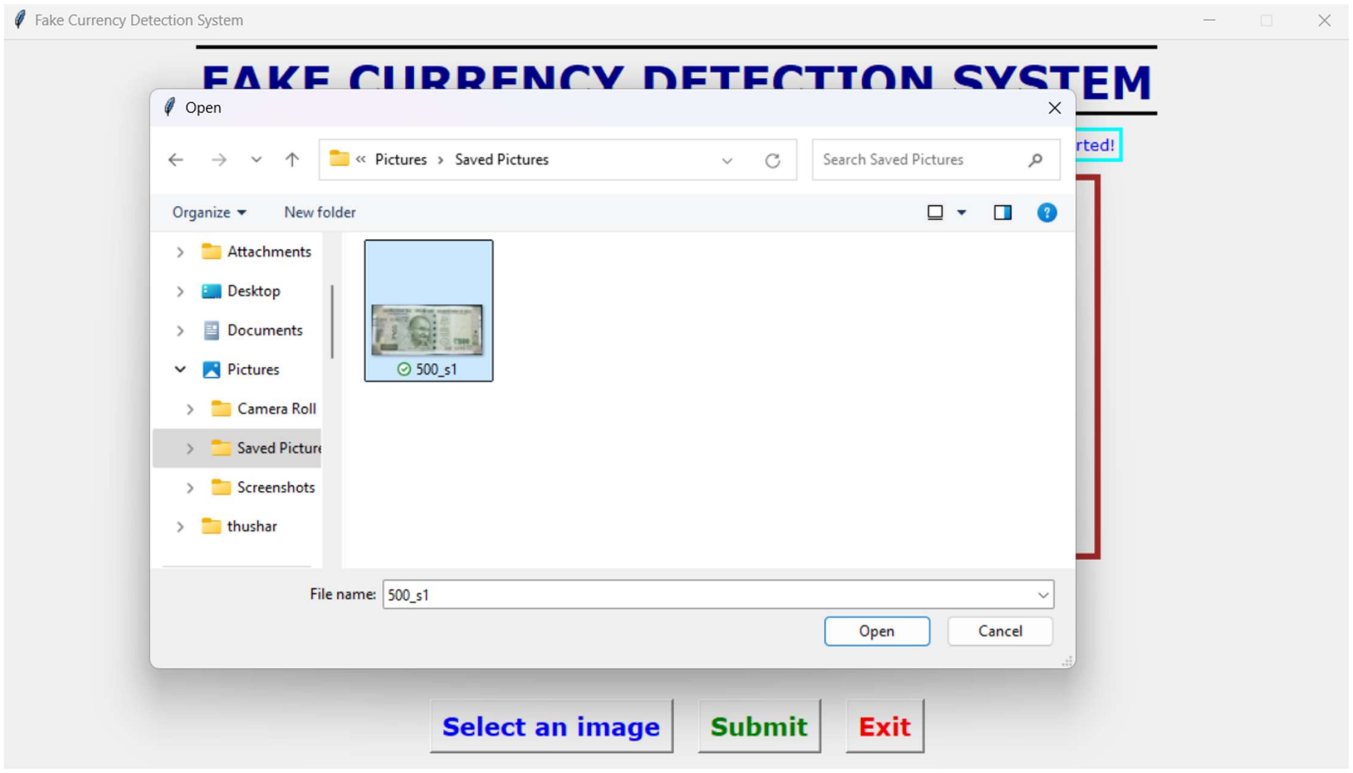
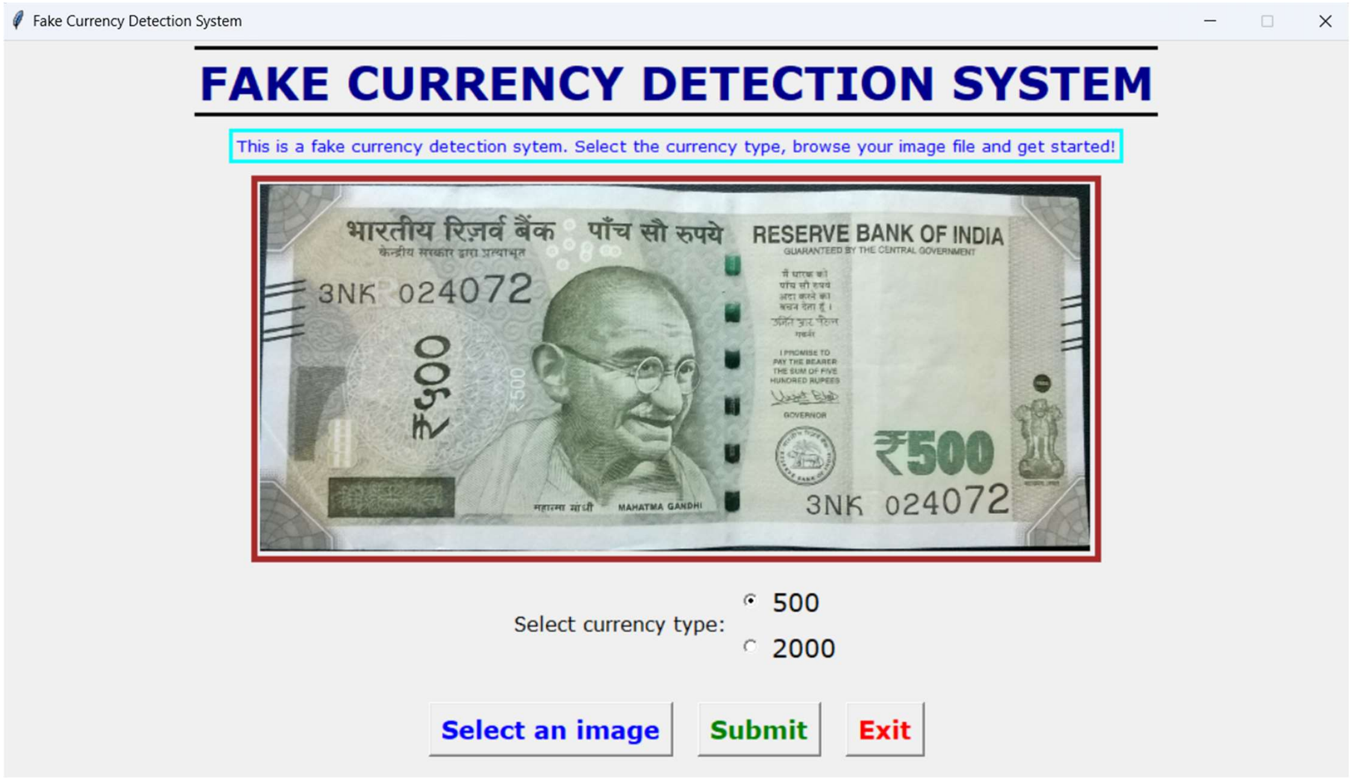


Fig No 7.2: Selecting Image


Fake Currency Detection System
FAKE CURRENCY DETECTION SYSTEM
Select an image
Submit
Exit

This is a fake currency detection sytem. Select the currency type, browse your image file and get started!

Select currency type: [✓ 500] 2000

Fig No 7.3: After selecting Image

3. Data processing:

- After clicking 'Submit' new page open for processing.
- Click 'Click to continue' for processing.

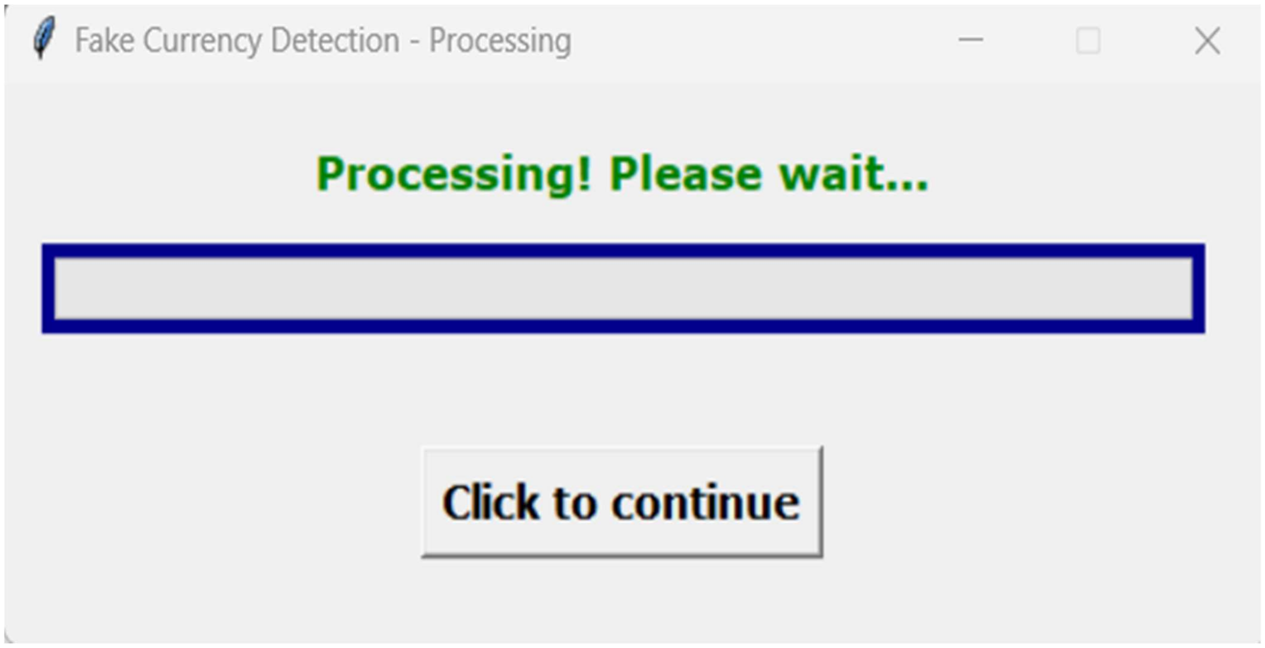

Fake Currency Detection - Processing
Processing! Please wait...
Click to continue

Fig No 7.4: Processing Page

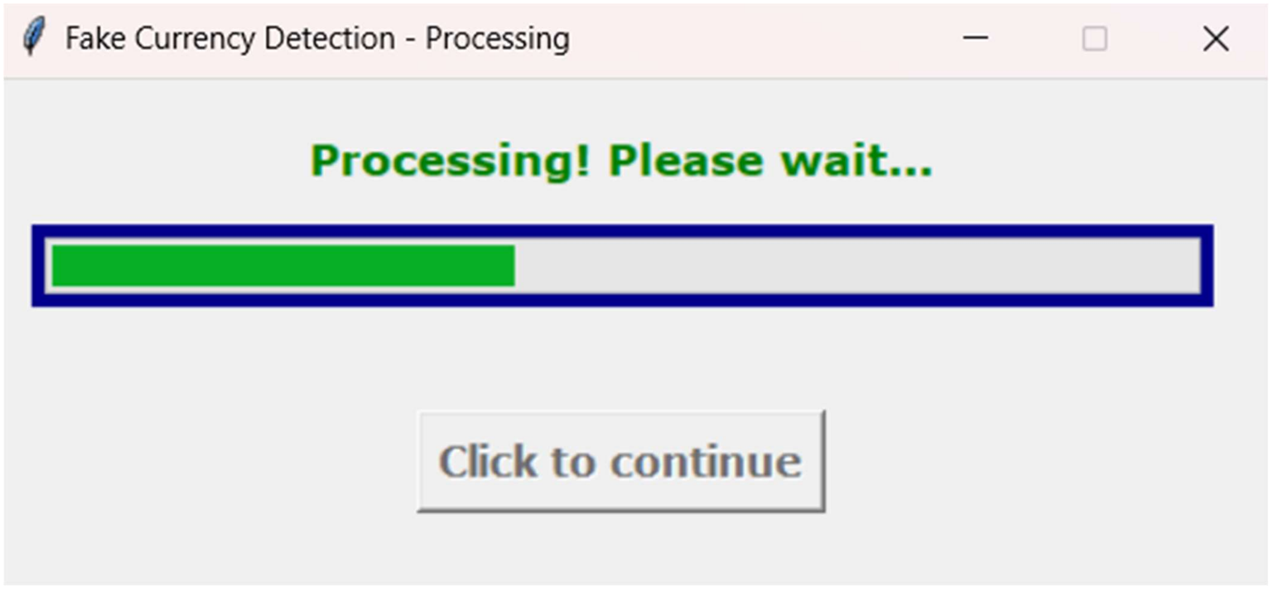
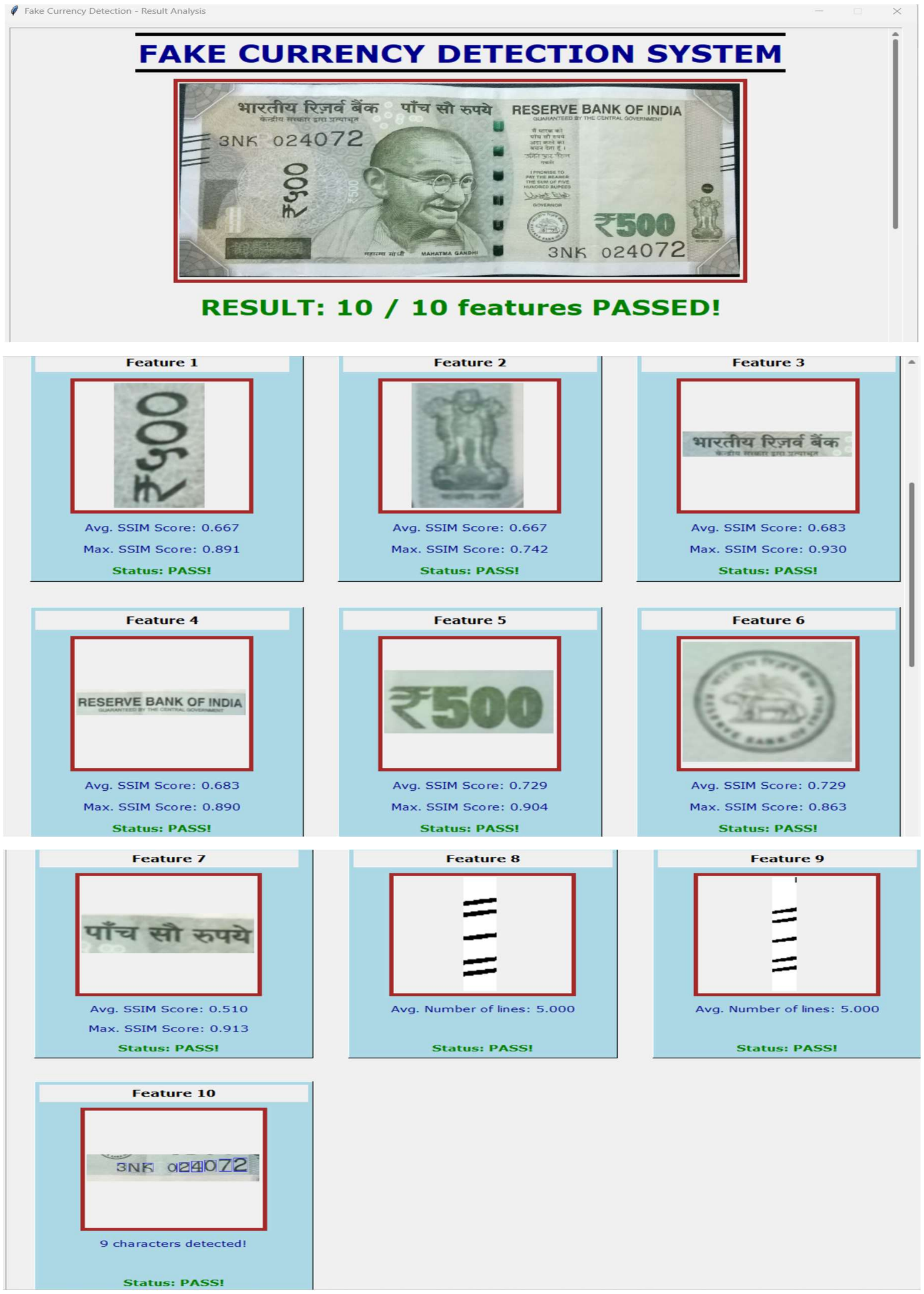

Fake Currency Detection - Processing
Processing! Please wait...
Click to continue

Fig No 7.5: Processing going on



FAKE CURRENCY DETECTION SYSTEM

RESULT: 10 / 10 features PASSED!

Feature 1
Avg. SSIM Score: 0.667
Max. SSIM Score: 0.891
Status: PASS!

Feature 2
Avg. SSIM Score: 0.667
Max. SSIM Score: 0.742
Status: PASS!

Feature 3
Avg. SSIM Score: 0.683
Max. SSIM Score: 0.930
Status: PASS!

Feature 4
Avg. SSIM Score: 0.683
Max. SSIM Score: 0.890
Status: PASS!

Feature 5
Avg. SSIM Score: 0.729
Max. SSIM Score: 0.904
Status: PASS!

Feature 6
Avg. SSIM Score: 0.729
Max. SSIM Score: 0.863
Status: PASS!

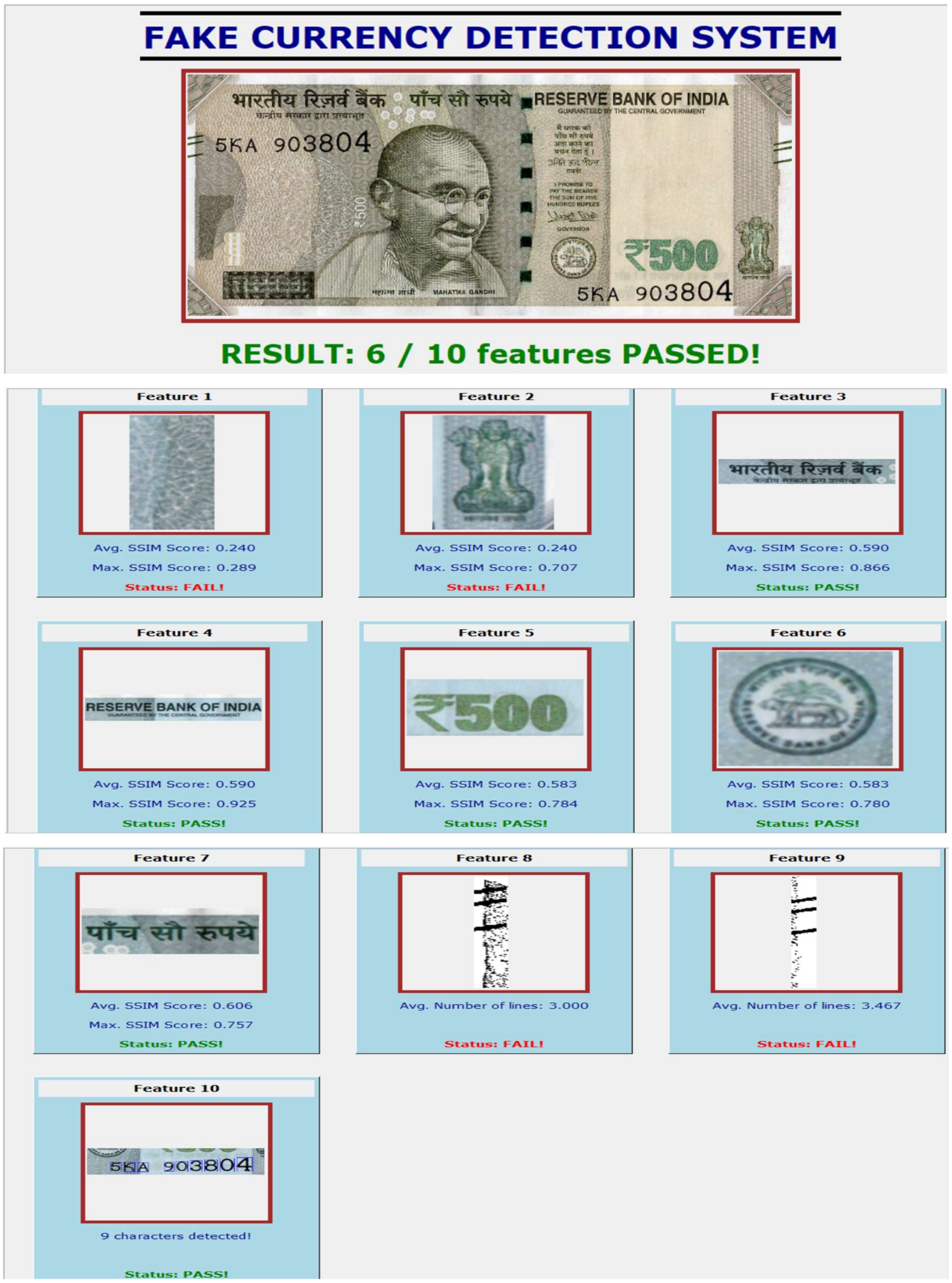
Feature 7
Avg. SSIM Score: 0.510
Max. SSIM Score: 0.913
Status: PASS!

Feature 8
Avg. Number of lines: 5.000
Status: PASS!

Feature 9
Avg. Number of lines: 5.000
Status: PASS!

Feature 10
9 characters detected!
Status: PASS!

Fig No 7.6: Result of Real Currency Note



FAKE CURRENCY DETECTION SYSTEM

RESULT: 6 / 10 features PASSED!

Feature 1
Avg. SSIM Score: 0.240
Max. SSIM Score: 0.289
Status: FAIL!

Feature 2
Avg. SSIM Score: 0.240
Max. SSIM Score: 0.707
Status: FAIL!

Feature 3
Avg. SSIM Score: 0.590
Max. SSIM Score: 0.866
Status: PASS!

Feature 4
Avg. SSIM Score: 0.590
Max. SSIM Score: 0.925
Status: PASS!

Feature 5
Avg. SSIM Score: 0.583
Max. SSIM Score: 0.784
Status: PASS!

Feature 6
Avg. SSIM Score: 0.583
Max. SSIM Score: 0.780
Status: PASS!

Feature 7
Avg. SSIM Score: 0.606
Max. SSIM Score: 0.757
Status: PASS!

Feature 8
Avg. Number of lines: 3.000
Status: FAIL!

Feature 9
Avg. Number of lines: 3.467
Status: FAIL!

Feature 10
9 characters detected!
Status: PASS!

Fig No 7.7: Result of Fake Currency Note

Chapter 8

CONCLUSION, APPLICATIONS AND FUTURE WORK

8.1 Conclusion

The Fake Currency Detection System is a robust and efficient solution designed to address the pressing challenge of identifying counterfeit currency in circulation. Through the integration of advanced image processing techniques and feature extraction algorithms, the system provides accurate and reliable results, significantly reducing human error and reliance on manual verification processes.

Key security features such as watermarks, holograms, and unique identifiers are analyzed using methods like ORB (Oriented FAST and Rotated BRIEF), SSIM (Structural Similarity Index), and contour detection. The use of gray-scale conversion, thresholding, and region analysis ensures high precision in extracting and comparing features against genuine reference data. The system's ability to assign confidence scores further enhances its decision-making capability, providing a clear distinction between real and counterfeit notes.

By leveraging preprocessing techniques and memory-efficient data structures like Pandas DataFrames with one-hot encoding, the system is optimized for speed, scalability, and adaptability. It can easily incorporate new currency types or features, making it suitable for deployment in banks, financial institutions, and retail environments.

While the system demonstrates promising results in terms of accuracy and reliability, there are limitations, such as sensitivity to image quality and lighting conditions. Future enhancements could involve integrating deep learning models for feature extraction, improving robustness against poor input conditions, and expanding support for a wider range of currencies.

In conclusion, the Fake Currency Detection System offers a practical and innovative approach to combating counterfeit currency, ensuring financial security and building trust in monetary transactions.

8.2 Applications

- **Banking and Financial Institutions**
    - Banks can use the system to verify the authenticity of currency notes during deposits and withdrawals.
    - It ensures secure transactions and reduces the risk of counterfeit money entering circulation.
    - Can be integrated with cash counting machines for automated verification.
- **Retail Sector**
    - Retailers and businesses handling large cash transactions can utilize the system at checkout points.
    - It helps minimize financial losses due to accepting counterfeit currency.
    - Ensures customer trust and prevents disputes related to fake money.
- **Currency Exchange Agencies**
    - Forex and currency exchange centers can use the system to validate the authenticity of different currencies.
    - Provides a quick and reliable method to handle foreign currency transactions.
- **Government and Law Enforcement**
    - Used by law enforcement agencies to detect and trace counterfeit currency in criminal investigations.
    - Can assist in identifying and tracking sources of counterfeit production.
- **Transportation and Logistics**
    - Toll booths, ticket counters, and transit stations handling cash payments can use the system to verify currency on the spot.
    - Reduces delays and improves transaction security.
- **E-commerce and Cash-on-Delivery Services**
    - E-commerce platforms offering cash-on-delivery services can use portable versions of the system to validate cash payments during delivery.
    - Helps build trust in the cash-on-delivery model.
- **Educational and Training Purposes**
    - Can be used in educational institutions to train banking and finance students in identifying fake currency.
    - Helps raise awareness about counterfeit detection methods and tools.

8.3 Limitations of the System

- **Dependence on Image Quality**
  - The accuracy of the system is highly dependent on the quality of the input image. Poor lighting, low resolution, or blurred images can lead to incorrect detection.
  - Currency notes with folds, stains, or damages may not be analyzed effectively.
- **Limited to Predefined Features**
  - The system relies on specific predefined features (e.g., watermarks, holograms) for detection. Counterfeit notes with new or highly advanced forgery techniques may evade detection.
  - It may not adapt well to changes in currency design unless re-trained or updated.
- **Time Consumption**
  - The system will take more time if more currency notes are testing.
  - For each notes we need to upload and test notes again and again.
  - If n number of notes is there then n times we need to run.

8.4 Future Scope of the Project

- **Multi-Currency Support**
  - Expanding the system to detect counterfeit currency across multiple denominations and international currencies would increase its global applicability.
  - A modular design can enable quick integration of new currencies by updating feature databases.
- **Real-Time Mobile Applications**
  - Developing lightweight, real-time mobile applications can make the system accessible to retailers, small businesses, and individuals.
  - Integration with mobile camera features for live detection can enable portable counterfeit detection.
- **IoT and Smart Devices Integration**
  - Embedding the system into IoT devices, such as cash counters, ATMs, and vending machines, can automate counterfeit detection in real-world scenarios.
  - Smart detection systems can also send alerts or logs to centralized databases for monitoring.

REFERENCES

[1] OpenCV Documentation. (n.d.). OpenCV: Open Source Computer Vision Library. Retrieved from https://opencv.org/documentation/.

[2] TensorFlow Documentation. (n.d.). TensorFlow: An End-to-End Open Source Machine Learning Platform. Retrieved from https://www.tensorflow.org/learn.

[3] Ghosh, S., & Nair, S. (2017). Counterfeit Currency Detection Using Image Processing Techniques. International Journal of Computer Applications, 164(3), 1-6. doi:10.5120/ijca2017913322.

[4] Rao, A., & Reddy, P. (2022). Improving Currency Detection Using Transfer Learning Techniques. Journal of Machine Learning Research, 22, 1-15. Retrieved from http://www.jmlr.org/papers/volume22/21-010/21-010.pdf.

[5] DETECTION OF FAKE CURRENCY NA Raksh, AS Kumar, GR Gorre, KU Kiran, SR Uddin - 2024 - researchgate.net.

[6] Real time fake currency note detection using deep learning M Laavanya, V Vijayaraghavan - Int. J. Eng. Adv. Technol.(IJEAT), 2019 - researchgate.net.

[7] An Artificial Neural Networks Based Fake Currency Detection System B Sambana, M Mahanty - International Journal of Computer Graphics, 2020 - academia.edu.

[8] Fake currency detection using image processing M Patil, J Adhikari, R Babu - … Journal on Future Revolution in Computer …, 2018 - core.ac.uk.

[9] Fake currency detection application A Vidhate, Y Shah, R Biyani, H Keshri… - Int. Res. J. Eng. Technol …, 2021 - academia.edu give literature review in given format.

[10] Design and Implementation of Fake Currency Detection System A Kamble, MS Nimbarte - International Journal on Future Revolution in …, 2018 - core.ac.uk.

[11] Sharan V, Kaur A (2019) Detection of counterfeit Indian currency note using image processing. Int J Eng Adv Technol 9(1):2440–2447.

[12] Saxena V, Snehlata (2019) An efficient technique for detection of fake currency. Int J Recent Technol Eng 8(3):1298–1305.

[13] Neeraja Y, Divija B, Nithish Kumar M (2019) Fake currency detection using K-NN technique. Int J Res Eng IT Soc Sci 9(1):201–205.

[14] Kumar D, Chauhan S (2019) Indian fake currency detection using computer vision. Int Res J Eng Technol 7(5):2870–2874.