FAKE CURRENCY DETECTION WITH FEATURE EXTRACTION USING ORB ALGORITHM

JONNA VENKATESH (20JJ1A0420)

BASHAVENI VENKATESH (20JJ1A0403)

GUGULOTH JAYANTH KUMAR (20JJ1A0418)

Department of Electronics and Communication Engineering

Jawaharlal Nehru Technological University Hyderabad

University College of Engineering Jagtial

UGC Autonomous Institution, Govt. of India

(Accredited by NAAC with 'A+' Grade)

Nachupally (Kondagattu), Jagtial Dist. - 505501, Telangana

2024

FAKE CURRENCY DETECTION WITH FEATURE EXTRACTION USING ORB ALGORITHM

A MAJOR PROJECT REPORT
SUBMITTED IN PARTIAL FULFILMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING
BY

JONNA VENKATESH (20JJ1A0420)
BASHAVENI VENKATESH (20JJ1A0403)
GUGULOTH JAYANTH KUMAR (20JJ1A0418)

Department of Electronics and Communication Engineering
Jawaharlal Nehru Technological University Hyderabad
University College of Engineering Jagtial
UGC Autonomous Institution, Govt.of India
(Accredited by NAAC with 'A+' Grade)
Nachupally (Kondagattu), Jagtial Dist. - 505501, Telangana

2024

Jawaharlal Nehru Technological University Hyderabad
University College of Engineering Jagtial

UGC Autonomous Institution, Govt.of India

(Accredited by NAAC with 'A+' Grade)

Nachupally (Kondagattu), Jagtial Dist. - 505 501, Telangana

Department of Electronics and Communication Engineering

CERTIFICATE

Date: / /

This is to certify that the project work entitled ***Fake currency detection with feature extraction using ORB algorithm*** *is a bonafide work carried out by* ***J. VENKATESH, B. VENKATESH*** *and* ***G. JAYANTH KUMAR*** *bearing Roll Nos.* ***20JJ1A0420, 20JJ1A0403,*** *and* ***20JJ1A0418*** *in partial fulfilment of the requirements for the degree of* ***BACHELOR OF TECHNOLOGY*** *in* ***ELECTRONICS & COMMUNICATION ENGINEERING*** *by the Jawaharlal Nehru Technological University Hyderabad during the academic year2023-24.*

The results embodied in this report have not been submitted to any otherUniversity or Institution for the award of any degree or diploma.

Dr. D. NAGA SUDHA	**Dr. B. PRABHAKAR**
Associate Professor, ECE	**Professor of ECE**
Project Guide	**Head of the Department ECE**

ACKNOWLEDGMENTS

The success of this project would not have been possible without the timely help and guidance of many people. We wish to express our sincere gratitude to all those who have helped and guided us through the completion of the project.

It is our pleasure to thank our Project Guide, **Dr. D. Naga Sudha, Associate Professor, Department of ECE**, for the guidance and suggestions throughout the project, without which we would not have been able to complete this project successfully.

We express our sincere gratitude to **Prof. Dr. B. Prabhakar, Project Coordinator, Head of Department, ECE and Principal of JNTUH-UCEJ**, for their encouragement and for providing facilities to accomplish our project successfully.

We would like to thank our classmates, all faculty members, and non-teaching staff of the ECE Department, JNTUH-UCEJ for their direct and indirect help during the project work.

Finally, we wish to thank our family members and our friends for their interest and assistance which has enabled us to complete the project work successfully.

J. Venkatesh (20JJ1A0420)
B. Venkatesh (20JJ1A0403)
G. Jayanth Kumar (20JJ1A0418)

ABSTRACT

Counterfeiting of currency has come a real trouble to the livelihood of people as well as the frugality of our country. Though fake currency sensors are available, they're confined to banks and commercial services leaving common people and small businesses vulnerable. So, in this design, we will probe the colourful security features of Indian currency and also, prepare a software- grounded system to descry and abate fake Indian currency by using advanced image processing and computer vision ways. This currency authentication system is designed fully using Python language in Jupyter Notebook terrain.

Due to the growth of technology, the fake currency product has been increased which degraded the frugality of our country. Then the suggested system uses OpenCV to fete whether the given note is original or fake. It consists of machine literacy ways that are carried out using suitable mechanisms. A fake currency discovery system is introduced which uses the edge discovery to descry the lines directly and also directly descry angles of respectable notes. Then we use a sensor that's trained with the help of stored information which is analogous to the bone that's to be tested or compared latterly; within those modules, anchor lines are defined that are farther depicted in posterior test patterns.

In order to give training for the sensor in offline a microprocessor is programmed. This is done with a sample currency attained by surveying given note into our proposed system, then a frame like design is attained by training image format. also, notes analogous to attained frame are to be linked. Inside the template, the microprocessor determines anchor lines that are farther depicted in that test format; it spins and moves the design before it matches to the training format, so that anchor lines which corresponds to the line can be linked in that trained dataset i.e., the pattern designed; and compares them with the test format to know if those anchor lines lie inside that test format. The system is proposed in a way that it shows if the currency is fake or it's original.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Currency duplication or product of fake currency notes immorally by imitating the factual manufacturing process is a huge problem that every country is facing. Fake currency can reduce the value of real plutocrat and beget affectation due to an unauthorized and unnatural increase in the plutocrat force. Homemade authentication of currency notes is a result but it's a veritably time- consuming, inaccurate, and delicate process. Automatic testing of currency notes is, thus, necessary for handling large volumes of currency notes and also, getting accurate results in a veritably short time span. In this design, we propose a fake currency note discovery system using colorful image processing ways and algorithms.

The proposed system is designed to validate Indian currency notes of denotation 500 and 2000 rupees. The system consists of three main algorithms and checks the authenticity of colorful features in a currency note. The first algorithm consists of several way including image accession, pre-processing, greyscale conversion, point birth, image segmentation, comparisons of images and affair, and uses advanced image processing styles similar as sphere and SSIM. The alternate algorithm authenticates the bleed lines of the currency notes whereas the third algorithm authenticates the number panel of the currency notes. Eventually, the reused affair is displayed for each currency note. This system provides a hassle-free way to authenticate currency notes snappily and directly. This automated system can replace the being primer styles and can be used by anyone fluently to descry fake currency.

1.2 Aim of the Project

To test the authenticity of Indian currency notes by preparing a system which takes the image of currency bill as input and gives the final result by applying various image processing and computer vision techniques and algorithms.

1.3 Objectives of the Project

- The main objective of the project is to identify the fake Indian currency notes through an automated system by using Image processing and computer vision techniques,
- The system should have high accuracy.
- The system should be able to give the final results in a short time.
- The system should have a User-friendly interface, to make it convenient to use and understand.

1.4 Methodology

➢ Preparation of Dataset

- The first step is the medication of a dataset containing images of different currency notes (both fake and real) and images of different features of each of the currency notes
- The dataset will contain the following depositories

  -Sub-dataset for Rs. 500 currency notes

  1) Images of real note
  2) Images of fake notes
  3) Multiple images of each security point(template)

  -Sub-dataset for Rs. 2000 currency notes

  1) Images of real notes
  2) Images of fake notes
  3) Multiple images of each security point(template)
- The colorful security features that we're considering are for Rs. 500 currency notes – Total 10 features)

  -Rs. 500 in Devanagari and English script (2 features)

  - Ashoka pillar hallmark (1 point)
  - RBI symbols in Hindi and English (2 features)
  - 500 rupees written in Hindi (1 point)
  - RBI totem (1 point)
  - Bleed lines on left and right side (2 features)
  - Number Panel (1 point)

- **Image Acquisition**

  Next, the image of the test-currency note is taken as input and fed it into the system. The image should be taken from a digital camera or preferably, using a scanner. The image should have a proper resolution, proper brightness and should not be hazy or unclear. Blurred images and images with less detail may adversely affect the performance of the system.

- **Pre-processing**

  Next, the pre-processing of the input image is done. 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.

- **Gray-scale conversion**

  Gray scale conversion is mainly used because an RGB image has 3 channels whereas a gray image has only one channel. This makes the computation and processing on images much easier in the case of gray scaled images.

- **Train Your Network**

  Given our training set of images, we can now train our network. The goal here is for our network to learn how to recognize each of the categories in our labeled data. When the model makes a mistake, it learns from this mistake and improves itself.

- **Evaluate**

  Last, we need to evaluate our trained network. For each of the images in our testing set, we present them to the network and ask it to predict what it thinks the label of the image is. We then tabulate the predictions of the model for an image in the testing set. Finally, these model predictions are compared to the ground-truth labels from our testing set. The ground-truth labels represent what the image category actually is. From there, we can compute the number of predictions our classifier got correct and compute aggregate reports such as precision, recall, and f-measure, which are used to quantify the performance of our network as a whole.

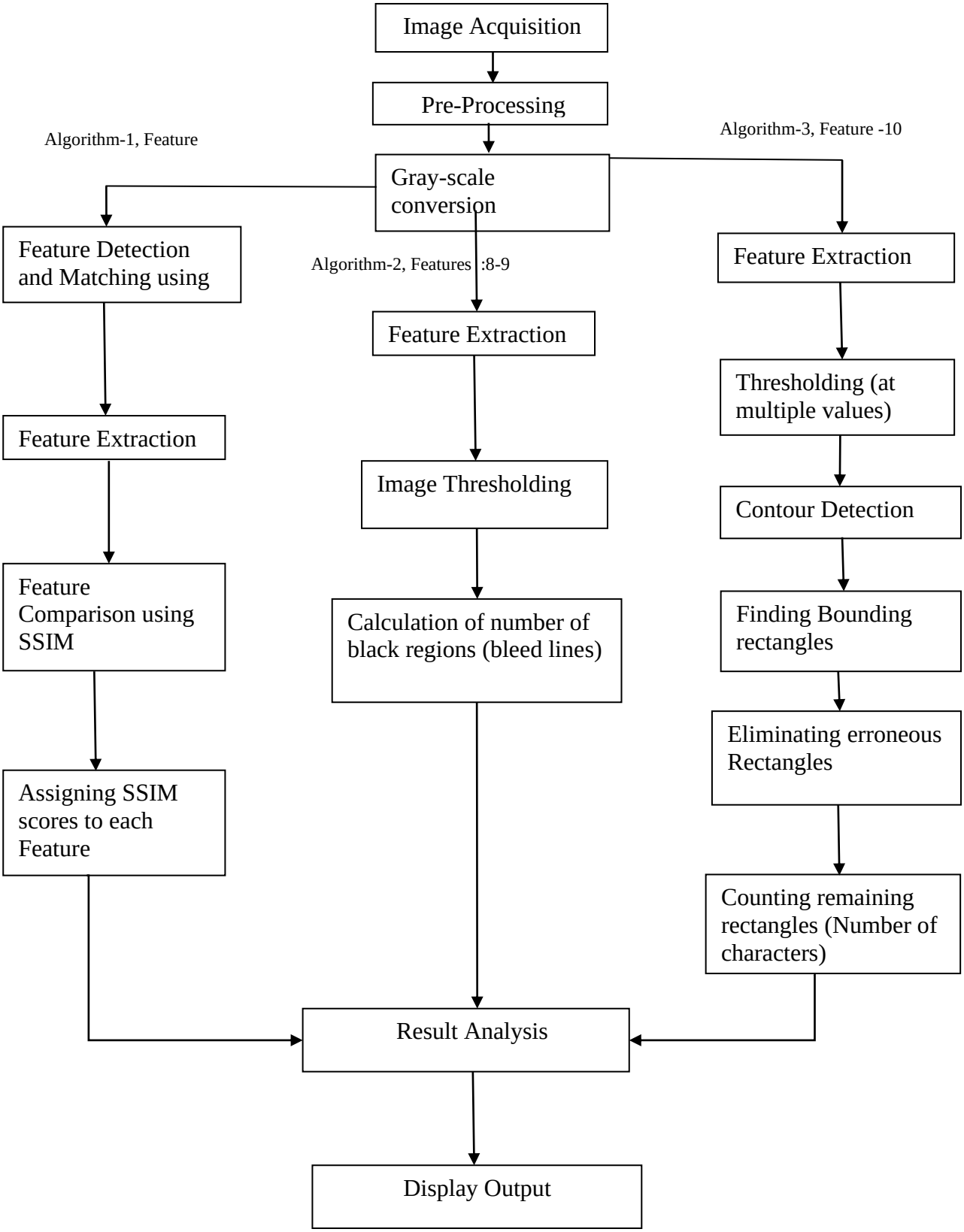

Figure 1.1 Flow Diagram

1.5 Organization of the thesis

This report is organized into majorly into 5 different sections and each section provides detailed description about the project. The 5 sections mentioned are:

1) **Introduction** – This section provides you overview of the project, what's the major problem that is being addressed, objectives, which methodology we are following to implement this project and information about remaining part of the report.
2) **Literature Survey** – This section provides previous work of this problem and their limitations.
3) **System requirements and Specifications** – This section provides information about functional and non-functional requirements of this project.
4) **Expected outcome** – This section provides about various modules used in this project design.
5) **Advantages and Applications** – What are the advantages of the approach or framework being developed comparatively to previous existing one and information regarding application of the project in various fields.

CHAPTER 2
DESCRIPTION OF ORB ALGORITHM

2.1 Introduction

ORB (Oriented FAST and rotated BRIEF) was brought up by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in their paper ORB: An efficient alternative to SIFT or SURF in 2011. As the title says, it is a good alternative to SIFT and SURF in computation cost, matching performance and mainly the patents. Yes, SIFT and SURF are patented and you are supposed to pay them for its use. But ORB is not.

2.2 ORB Algorithm

ORB is a feature detection and description algorithm used in computer vision. ORB is designed to efficiently detect and describe keypoints (unique and identifiable location in an image) in images, making it useful for various tasks such as object recognition, tracking, and image stitching.

ORB mixes techniques used in the FAST keypoint detection and the BRIEF descriptor: FAST (Features from Accelerated Segment Test) and BRIEF (Binary Robust Independent Elementary Features). Before explaining FAST and BRIEF, let's briefly discuss keypoint detection and feature descriptors. Keypoint detection is all about finding special, easily recognizable spots in an image, while a feature descriptor gives us a numeric summary of what's around each of those spots.Now let's talk about the FAST and BRIEF algorithms. FAST is an intensity-based corner detection algorithm used to identify keypoints, and BRIEF is a feature descriptor that generates binary feature descriptors for these keypoints.

ORB is basically a fusion of FAST key point detector and BRIEF descriptor with many modifications to enhance the performance. First it uses FAST to find key points, then apply Harris corner measure to find top N points among them. It also uses pyramid to produce multiscale-features. But one problem is that, FAST doesn't compute the orientation.It computes the intensity weighted centroid of the patch with located corner at center. The direction of the vector from this corner point to centroid gives the orientation.
To improve the rotation invariance, moments are computed with x and y which should be in a circular region of radius r, where r is the size of the patch.

Now for descriptors, ORB use BRIEF descriptors. But we have already seen that BRIEF performs poorly with rotation. So, what ORB does is to "steer" BRIEF according to the orientation of key points. For any feature set of n binary tests at location (xi, yi), define a 2×n matrix, S which contains the coordinates of these pixels. Then using the orientation of patch, θ, its rotation matrix is found and rotates the S to get steered(rotated) version $S\theta$.

ORB discretize the angle to increments of $2\pi/30$ (12 degrees), and construct a lookup table of precomputed BRIEF patterns. As long as the key point orientation θ is consistent across views, the correct set of points $S\theta$ will be used to compute its descriptor.

BRIEF has an important property that each bit feature has a large variance and a mean near 0.5. But once it is oriented along key point direction, it loses this property and become more distributed. High variance makes a feature more discriminative, since it responds differentially to inputs. Another desirable property is to have the tests uncorrelated, since then each test will contribute to the result. To resolve all these, ORB runs a greedy search among all possible binary tests to find the ones that have both high variance and means close to 0.5, as well as being uncorrelated. The result is called **rBRIEF**.

For descriptor matching, multi-probe LSH which improves on the traditional LSH, is used. The paper says ORB is much faster than SURF and SIFT and ORB descriptor works better than SURF. ORB is a good choice in low-power devices for panorama stitching etc.

ORB in OpenCV

As usual, we have to create an ORB object with the function, **cv.ORB()** or using feature2d common interface. It has a number of optional parameters. Most useful ones are nFeatures which denotes maximum number of features to be retained (by default 500), scoreType which denotes whether Harris score or FAST score to rank the features (by default, Harris score) etc. Another parameter, WTA_K decides number of points that produce each element of the oriented BRIEF descriptor. By default, it is two, i.e. selects two points at a time. In that case, for matching, NORM_HAMMING distance is used. If WTA_K is 3 or 4, which takes 3 or 4 points to produce BRIEF descriptor, then matching distance is defined by NORM_HAMMING2.

2.3 Feature Detection and matching Using ORB

After completing the necessary processing of the image, feature detection 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.

2.4 FAST (Feature from Accelerated and Segment Test)

Features from accelerated segment test (FAST) is a corner detection method, which could be used to extract feature points and later used to track and map objects in many computer vision tasks. The FAST corner detector was originally developed by Edward Rosten and Tom Drummond and was published in 2006. The most promising advantage of the FAST corner detector is its computational efficiency. Moreover, when machine learning techniques are applied, superior performance in terms of computation time and resources can be realized. The FAST corner detector is very suitable for real-time video processing application because of this high-speed performance.

Feature Detection using FAST

The algorithm is explained below:

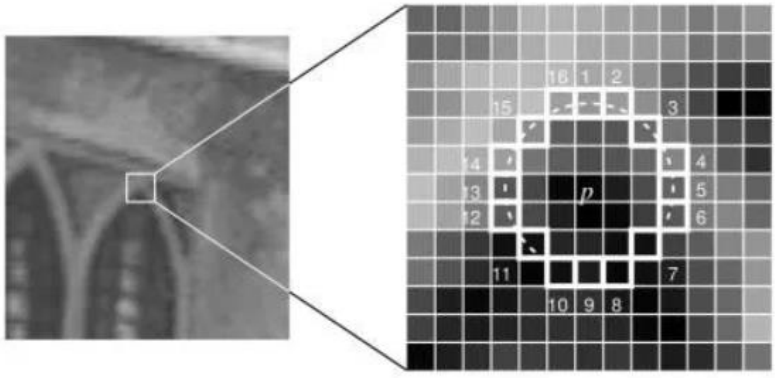

Figure 2.1 Feature Detection using FAST

- Select a pixel ***p*** in the image which is to be identified as an interest point or not. Let its intensity be ***Ip***.

- Select appropriate threshold value ***t***.

- Consider a circle of 16 pixels around the pixel under test. (This is a Bresenham circle of radius 3.)

- Now the pixel ***p*** is a corner if there exists a set of ***n*** contiguous pixels in the circle (of 16 pixels) which are all brighter than ***Ip + t***, or all darker than ***Ip - t***. (The authors have used ***n***= 12 in the first version of the algorithm)

- To make the algorithm fast, first compare the intensity of pixels 1, 5, 9 and 13 of the circles with ***Ip***. As evident from the figure above, at least three of these four pixels should satisfy the threshold criterion so that the interest point will exist.

- If at least three of the four-pixel values — ***I1, I5, I9, I13*** are not above or below ***Ip + t***, then ***p*** is not an interest point (corner). In this case reject the pixel ***p*** as a possible interest point. Else if at least three of the pixels are above or below ***Ip + t***, then check for all 16 pixels and check if 12 contiguous pixels fall in the criterion.

- Repeat the procedure for all the pixels in the image.

There are a few limitations to the algorithm. First, for ***n***<12, the algorithm does not work very well in all cases because when ***n***<12 the number of interest points detected are very high. Second, the order in which the 16 pixels are queried determines the speed of the algorithm. A machine learning approach has been added to the algorithm to deal with these issues.

However, FAST features do not have an orientation component and multiscale features. So orb algorithm uses a multiscale image pyramid. An image pyramid is a multiscale representation of a single image, that consist of sequences of images all of which are versions of the image at different resolutions. Each level in the pyramid contains the downsampled version of the image than the previous level. Once orb has created a pyramid it uses the fast algorithm to detect keypoints in the image. By detecting keypoints at each level orb is effectively locating key points at a different scale. In this way, ORB is partial scale invariant.

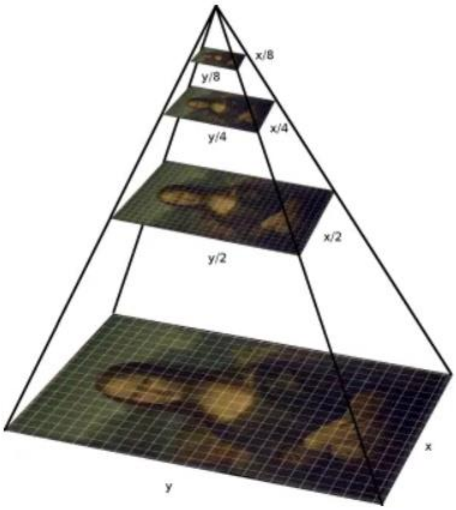

Figure 2.2 Multiscale image Pyramid

After locating keypoints orb now assign an orientation to each keypoint like left or right facing depending on how the levels of intensity change around that keypoint. For detecting intensity change orb uses intensity centroid. The intensity centroid assumes that a corner's intensity is offset from its center, and this vector may be used to impute an orientation.

First, the moments of a patch are defined as:

$$m_{pq} = \sum_{xy} x^p y^q I(x, y)$$

ORB descriptor- Patch's moment's definition

With these moments we can find the centroid, the "Center of mass" of the patch as:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right)$$

ORB descriptor – center of the mass of the patch

We can construct a vector from the corner's center O to the centroid – OC.
The orientation of the patch is given by:

$$\theta = atan\,2(m_{01}, m_{10})$$

ORB descriptor – Oriented of the patch

2.5 Brief (Binary robust independent elementary feature)

Brief takes all keypoints found by the fast algorithm and convert it into a binary feature vector so that together they can represent an object. Binary features vector also know as binary feature descriptor is a feature vector that only contains 1 and 0. In brief, each keypoint is described by a feature vector which is 128–512 bits string.

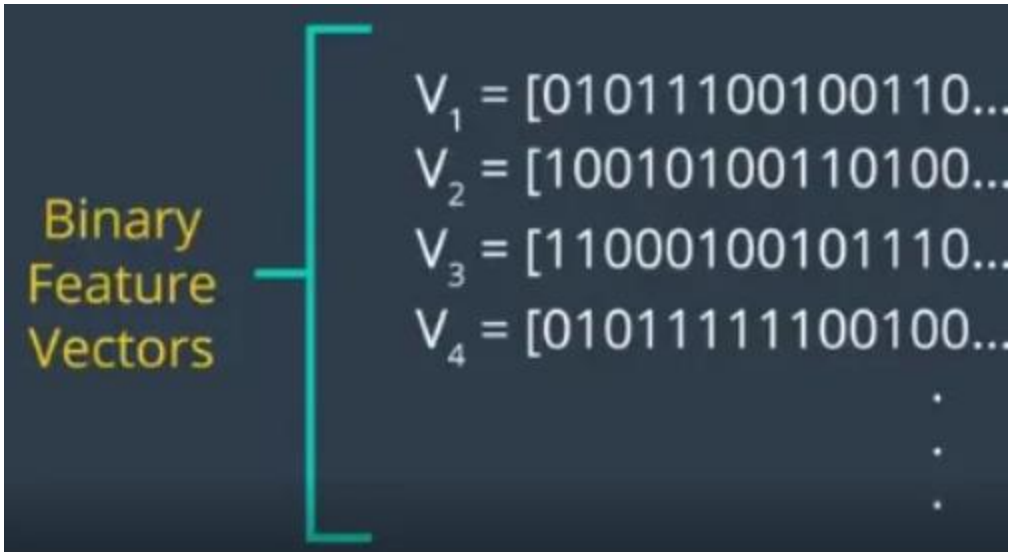

Figure 2.3 Binary Feature Vectors

BRIEF start by smoothing image using a Gaussian kernel in order to prevent the descriptor from being sensitive to high- frequency noise. Then brief select a random pair of pixels in a defined neighborhood around that keypoint. The defined neighborhood around pixel is known as a patch, which is a square of some pixel width and height. The first pixel in the random pair is drawn from a Gaussian distribution centered around the keypoint with a stranded deviation or spread of sigma. The second pixel in the random pair is drawn from a Gaussian distribution centered around the first pixel with a standard deviation or spread of sigma by two. Now if the first pixel is brighter than the second, it assigns the value of 1 to corresponding bit else 0.

Again, brief select a random pair and assign the value to them. For a 128-bit vector, brief repeat this process for 128 times for a keypoint. Brief create a vector like this for each keypoint in an image. However, BRIEF also isn't invariant to rotation so orb uses rBRIEF(Rotation-aware BRIEF). ORB tries to add this functionality, without losing out on the speed aspect of BRIEF.

Figure 2.4 Key point Detection Using BRIEF

Consider a smoothed image patch, p. A binary test τ is defined by:

Where $\tau(p;x,y)$ is defined as:

$$\tau(p;x,y) = \begin{cases} 1: p(x) < p(y) \\ 0: p(y) \geq p(y) \end{cases}$$

where p(x) is the intensity of p at a point x. The feature is defined as a vector of n binary tests:

$$f(n) = \sum_{1<i<n} 2^{i-1}\tau(p;x_i,y_i)$$

The matching performance of BRIEF falls off sharply for in-plane rotation of more than a few degrees. ORB proposes a method to steer BRIEF according to the orientation of the keypoints. For any feature set of n binary tests at location (xi, yi), we need the 2x n matrix:

$$s = \begin{Bmatrix} x_1 \dots x_n \\ y_1 \dots y_n \end{Bmatrix}$$

It uses the patch orientation θ and the corresponding rotation matrix Rθ, and construct a steered version Sθ of S:

$$s_\theta = R_\theta S$$

Now, the steered BRIEF operator becomes:

$$g_n(p,\theta) = f_n(p) | (\mathbf{x}_i, \mathbf{y}_i) \in S_\theta$$

Then it discretizes the angle to increments of $2\pi/30$ (12 degrees), and construct a lookup table of precomputed BRIEF patterns. As long as the keypoint orientation θ is consistent across views, the correct set of points $S\theta$ will be used to compute its descriptor.

ORB specifies the rBRIEF algorithm as follows:

1) Run each test against all training patches.

2) Order the tests by their distance from a mean of 0.5, forming the vector T.

3) Greedy search:

- Put the first test into the result vector R and remove it from T.
- Take the next test from T, and compare it against all tests in R. If its absolute correlation is greater than a threshold, discard it; else add it to R.
- Repeat the previous step until there are 256 tests in R. If there are fewer than 256, raise the threshold and try again

rBRIEF shows significant improvement in the variance and correlation over steered BRIEF.

2.6 Advantages and Disadvantages of ORB algorithm

The ORB (Oriented FAST and Rotated BRIEF) algorithm is a feature detection and description method used in computer vision. Here are some advantages and disadvantages:

Advantages:

1. Fast computation: ORB algorithm is designed to be computationally efficient, making it suitable for real-time applications such as object tracking and augmented reality.
2. Rotation invariance: ORB features are invariant to image rotation, meaning they can detect and describe keypoints regardless of the orientation of the object in the image.

3.Scale invariance: ORB features are also invariant to changes in scale, allowing them to detect objects at different sizes within the same image.
4. Robustness: ORB is robust to noise and illumination changes, making it suitable for use in various environments.
5.Binary descriptors: ORB uses binary descriptors, which are compact and require less memory compared to traditional descriptors like SIFT or SURF.

Disadvantages:

1. Limited invariance: While ORB features are rotation and scale invariant to some extent, they may not perform as well as other algorithms in highly complex or distorted images.
2. Limited robustness to viewpoint changes: ORB may struggle with significant changes in viewpoint, where the object's appearance drastically alters.
3. Limited feature matching quality: The binary nature of ORB descriptors may lead to lower matching quality compared to some other algorithms, especially in scenarios with significant occlusion or viewpoint changes.

Overall, while ORB offers fast computation and robustness to certain image transformations, its performance can vary depending on the specific application and the characteristics of the input images. It's often used in scenarios where real-time processing and efficiency are prioritized over robustness in complex scenes.

2.7 Conclusion

In this Chapter description of ORB algorithm explained

CHAPTER 3

SYSTEM REQUIREMENTS AND SPECIFICATION

3.1 Introduction

System Requirement Specification is a fundamental document, which forms the foundation of the software development process. It not only lists the requirements of a system but also has a description of its major feature. A System requirement specification is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two- way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it. It is important to note that a System requirement specification contains functional and non-functional requirements only.

It doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements.

3.2 Hardware Specification

- Processor: intel core i3 or above
- Processor speed: 500Mhz or above
- RAM: 4GB or Above

3.3 Software Requirement

- Python
- Python IDE (Jupiter Note book)

Software Libraries Required

- OpenCV
- NumPy
- Tkinter

Software Specification:

Python

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.
Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object oriented and functional programming. Python is often described as a “batteries included” language due to its comparative standard library.

Figure 3.1 Python logo

History Of Python

- Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Smalltalk, and Unix shell and other scripting languages.

- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).
- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Applications Of Python

Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read − Python code is more clearly defined and visible to the eyes.

Easy-to-maintain − Python's source code is fairly easy-to-maintain.

A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases − Python provides interfaces to all major commercial databases.

GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable − Python provides a better structure and support for large programs than shell scripting.

Features Of Python

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking. It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java

Jupyter Notebook

Jupyter Notebook is a notebook authoring application, under the Project Jupyter umbrella. Built on the power of the computational notebook format, Jupyter Notebook offers fast, interactive new ways to prototype and explain your code, explore and visualize your data, and share your ideas with others.

Notebooks extend the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. The Jupyter notebook combines two components:

1) **A web application:** A browser-based editing program for interactive authoring of computational notebooks which provides a fast interactive environment for prototyping and explaining code, exploring and visualizing data, and sharing ideas with others
2) **Computational Notebook documents**: A shareable document that combines computer code, plain language descriptions, data, rich visualizations like 3D models, charts, mathematics, graphs and figures, and interactive controls

Notebook user Interface

When you create a new notebook document, you will be presented with the **notebook name**, a **menu bar**, a **toolbar** and an empty **code cell**.

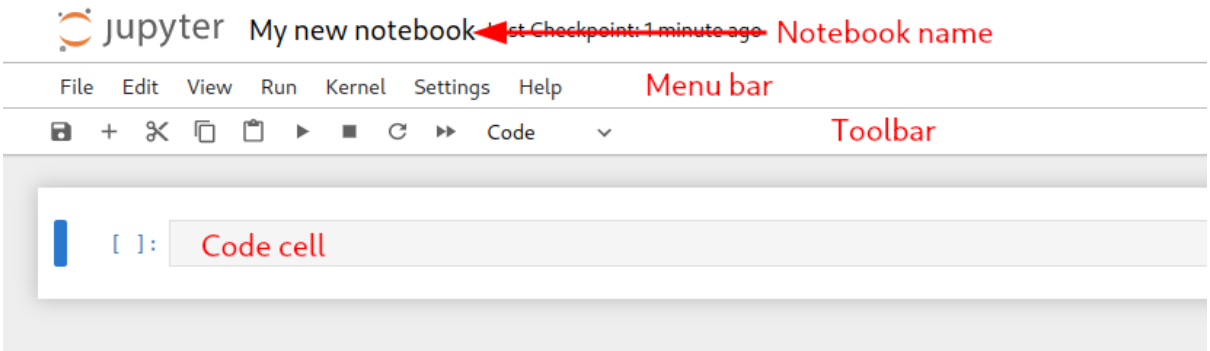

Figure 3.2 Jupyter notebook user interface

3.3Libraries Specification

OpenCV

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in NumPy can be combined with OpenCV

OpenCV is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as NumPy which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e. whatever operations one can do in NumPy can be combined with OpenCV
It is used for:

- Reading an image
- Extraction the RGB values of a pixel
- Extracting the Region of Interest (ROI)
- Resizing the image
- Rotating the image
- Drawing a Rectangle
- Displaying text

NumPy

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, Fourier transform, and matrices. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy. Arrays are very frequently used in data science, where speed and resources are very important.

Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib

Structural Similarity Index

When comparing images, the mean squared error (MSE)–while simple to implement–is not highly indicative of perceived similarity. Structural similarity aims to address this shortcoming by taking texture into account.

Tkinter

Python provides various options for developing graphical user interfaces (GUIs). The most important features are listed below.

- **Tkinter** − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look this option in this chapter.
- **wxPython** − This is an open-source Python interface for wxWidgets GUI toolkit. You can find a complete tutorial on WxPython
- **PyQt** −This is also a Python interface for a popular cross-platform QtGUI library. TutorialsPoint has a very good tutorial on PyQt
- **JPython** − JPython is a Python port for Java, which gives Python scripts seamless access to the Java class libraries on the local machine

3.5 FUNCTIONAL REQUIREMENTS

- Device must be enabled or disabled by user.
- Device should be able to extract the features of the image captured by camera.
- Device should be able to tell the real image or fake image notes.
- GUI performance should be stable without severely lagging

3.6 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements are the requirements which are not directly concerned with the specific function delivered by the system. They specify the criteria that can be used to judge the operation of a system rather than specific behaviours. They may relate to emergent system properties such as reliability, response time and store occupancy. Non-functional requirements arise through the user needs, because of budget constraints, organizational policies, and the need for interoperability with other software and hardware systems or because of external factors such as:

Usability

The system must be easy to learn for both users of the device and helpers who are the users of the GUI interface.

Reliability

The reliability of the device essentially depends on the software tools (OpenCV, NumPy, TensorFlow etc.) and hardware tools (camera, computer etc.) used for the system development.

3.7 Conclusion

In this chapter system requirements, hardware and specifications, Libraries specifications are discussed and explained.

CHAPTER 4

IMPLEMENTATION

4.1 Introduction

Implementation is the process of converting a new system design into an operational one. It is the key stage in achieving a successful new system. It must therefore be carefully planned and controlled. The implementation of a system is done after the development effort is completed.

Steps for Implementation

- Write up Installation of Hardware and Software utilities.
- Write up about sample data used.
- Write up about debugging phase.
- Implementation steps

The implementation phase of software development is concerned with translating design specifications into source code. The primary goal of implementation is to write source code and internal documentation so that conformance of the code to its specifications can be easily verified and so that debugging testing and modification are eased. This goal can be achieved by making the source code as clear and straightforward as possible. Simplicity clarity and elegance are the hallmarks of good programs and these characteristics have been implemented in each program module.

The goals of implementation are as follows

- Minimize the memory required.
- Maximize output readability.
- Maximize source text readability.
- Minimize the number of source statements
- Minimize development time

Figure 4.1 ORB* Feature *detection and Matching

4.2 Algorithm - 1

Training Dataset

It is initial set of data used to help a program understand how to apply neural networks and produce results. It is then complimented by validation and testing dataset. The initial approach to the training set images that are to be processed in order to reduce the data, by thresholding them into a binary image.

Feature Extraction

It is a process of dimension reduction of initial raw dataset to more manageable groups for processing. In the feature extraction stage redundancy from the data is removed. Now, using ORB location of each template has been detected in the input image within the highlighted 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.

Figure 4.2 Features in 500 | currency bill

Figure 4.3 Features in 2000 | currency bill

Feature comparison using SSIM

From the previous step, the part of the test currency image which matches with each of the templates will be generated. In this method, the original template will be compared with the extracted feature and then a score will be given for the similarity between the two images using SSIM.

$$\text{SSIM}\,(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

The Structural Similarity Index (SSIM) is a scoring system that quantifies the image quality degradation that is caused by processing such as data compression or by losses in data transmission. Basically, it looks for similarity between two images. It is a part of skimage library and uses the above-mentioned formula to calculate similarity. It returns a value between -1 and 1. Closer the SSIM to 1, higher is the similarity. So, for every security feature, the SSIM value between each image of that security feature and the corresponding extracted feature from the test image will be calculated. Then, the mean SSIM for each security feature is calculated and stored.

4.3 Algorithm 2: For features 8 and 9

1) Feature Extraction: In the first step, the region in which the bleed lines are present are extracted by cropping the image. So, a part near the left and right edges of the input currency note image is carefully extracted.

2) Image Thresholding: In the 2nd step, the image is thresholded using a suitable value. This ensures that only the black bleed lines remain on a white background and makes further processing quite easy.

3) Calculation of number of bleed lines: The 3rd step involves calculation of number of bleed lines. In this step, first we iterate over each column of the thresholded image.

Then we iterate over each pixel of each column. Then, we calculate the number of black regions in each column by increasing a counter whenever current pixel of the column is white and the immediate next pixel is black. Similarly, we, count number of black regions for each column, but, if the number of black regions is too large (>= 10), then that column is erroneous and it is discarded. Finally, the average count of black regions is calculated by considering the non- erroneous columns only and the result is displayed as the number of bleed lines. This count should be approximately 5 for Rs 500 currency notes and 7 for Rs 2000 currency notes.

4.4 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 finally counts the number of characters present in the number panel.

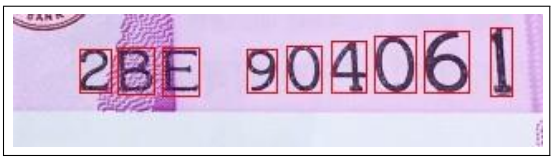

Figure 4.4 Number Panel Detection

Image Thresholding (with multiple values): The 1st step in this algorithm is again thresholding with suitable value so that only the black characters remain in the number

panel on a white background and become easy to detect.

But in this algorithm, thresholding is done using multiple values, i.e. first the image is thresholded at the initial value (90), then all other steps mentioned below are done and number of characters are calculated. After that, the threshold value is increased by 5 every time and the process of calculation of number of characters is repeated till either we reach the final value (150, in our case) or we find enough proof that 9 characters are present in the number panel.

Contour Detection: In the 2nd step, contour detection of the thresholded image of number panel is done.

Finding Bounding Rectangles: In the 3rd step, the bounding rectangle for each contour is found. The details of each rectangle is put inside a list.

Eliminating erroneous rectangles: The list of rectangles computed in the previous step may contain a number of erroneous and unnecessary rectangles due to noise present in the image. These erroneous rectangles need to be eliminated. So, in this step, all rectangles whose area is either too big or too small are eliminated. Then, the rectangles which are bound by a bigger rectangle are also eliminated. Finally, those rectangles which are positioned completely too high in the number panel are also eliminated.

Calculation of number of characters: The rectangles re- maining after the previous elimination step are those rectangles which bound only each character of the number panel. The number of rectangles still remaining is calculated and this gives us the number of characters detected in that particular thresholded image.

The above process is repeated for multiple threshold values (starting form 90 or 95 and increasing by 5 in each iteration). The algorithm stops if either it detects 9 characters in three consecutive iterations or if the threshold value reaches the maximum value (150 in our case).

4.5 Pseudo code for Controller:

```
# Controller

# Importing important libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim
```

```
#Resizing the Plots
plt.rcParams["figure.figsize"] = (12, 12)
%store -z  # Deleting all pre- stored variables
# Running gui_1.ipynb

%run ./gui_1.ipynb
# The GUI file produces and then stores the variables path and option
# Reading the stored variables

%store -r selectedImage
%store -r path
%store -r option

print ('Image selected: ', selectedImage)
print('Path: ', path)
print('Currency type: ', option)
if selectedImage == True:
    if option == 1:                    # For 500 currency note
        %run ./500_Testing.ipynb
    elif option == 2:                   # For 2000 currency note
        %run ./2000_Testing.ipynb
if selectedImage == True:
    # The above file produces and stores result_list variable
    # Reading the variable
    %store -r result_list

    # Showing the results
    # The result list variable is a list of lists and each list conatins          details about each
feature
    for x in result_list:
        if x[0] is not None:
            plt.imshow(x[0])   # Showing images
            plt.show()

if selectedImage == True:
# Show output in GUI

    %run ./gui_2.ipynb

%store -z  # Deleting all pre- stored variables
```

4.6 Conclusion

In this chapter various algorithms used in the project and implementation of those are discussed and explained.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Introduction

In this chapter we discuss the project outputs and the sequential process of the project, i.e., flowchart of the project, testing the controller using various cases, experimental set up and results of the project.

5.2 Flow Chart

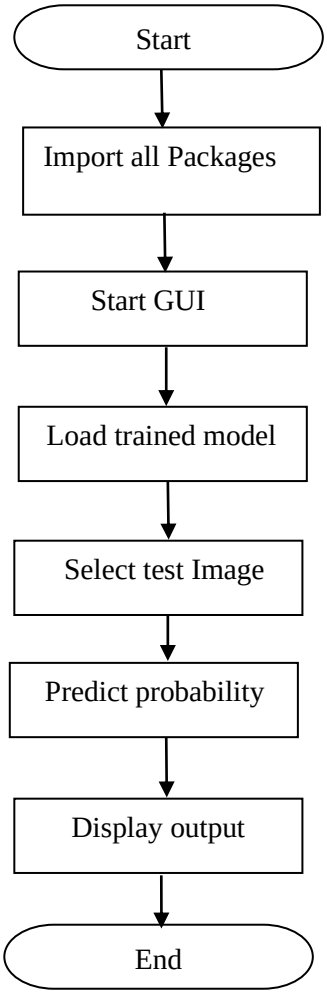

Figure 5.1 Flow Chart

Description of Flowchart:

When we start the controller, it imports all necessary packages and starts GUI for user interface. Here we load trained model and will give test image to the trained model. The trained model will predict probability and compares with the threshold values and display output.

5.3 Experimental Setup

The Following (figure 5.2) shows the experimental setup of the project

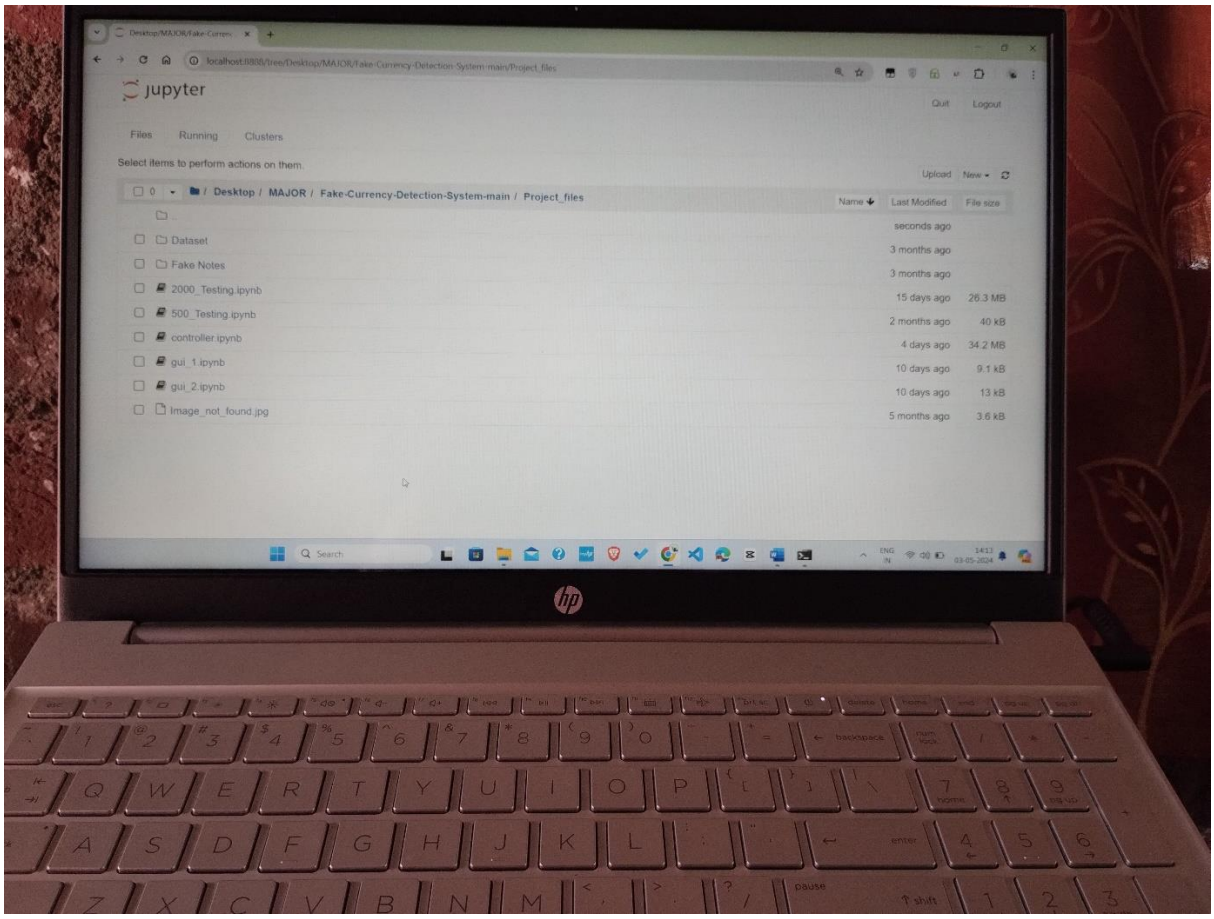

Figure 5.2 Experimental setup

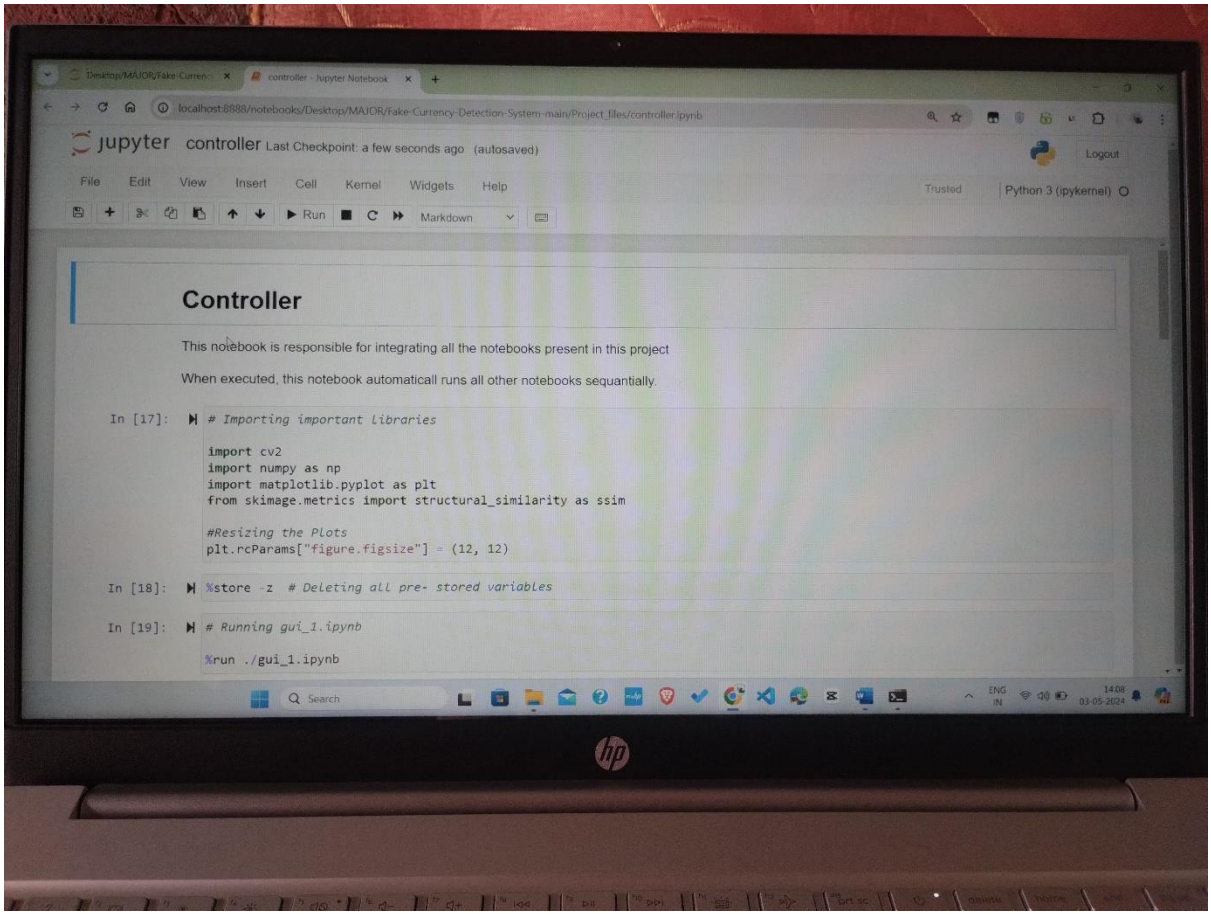

Figure 5.3 Jupyter Notebook Interface

The experimental setup consists of all the required components like laptop with required specifications and keyboard for input and proper internet

5.4 Testing

5.4.1 Introduction

Software testing is an examination that is carried out to offer information to stakeholders regarding the quality of the product or service being tested. Software testing can also give a corporation with an objective, unbiased picture of the software, allowing them to grasp and comprehend the risks associated with software implementation. The process of executing a programme or application with the goal of detecting software bugs is known as testing (errors or other defects). Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- responds correctly to all kinds of inputs.
- performs its functions within an acceptable time.
- Is sufficiently usable.
- can be installed and run in its intended environments
- Achieves the general result its stakeholder's desire.

The testing steps are:

- Unit Testing
- Validation Testing
- Integration Testing
- User Acceptance Testing
- Output Testing

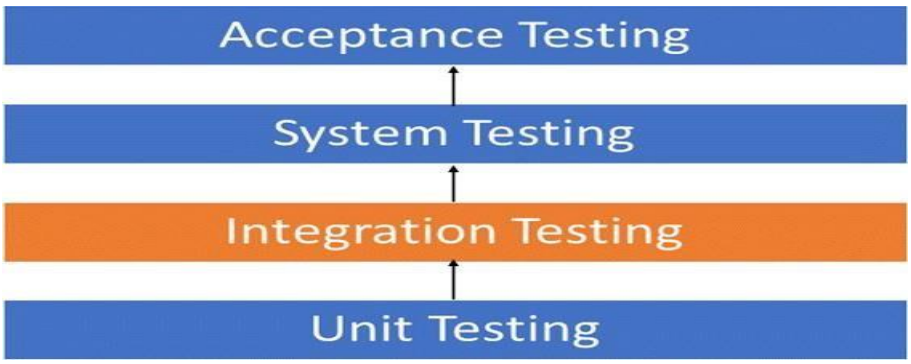

Figure 5.4 Testing

5.4.2 Methods of Testing

1) Unit Testing

Individual units or components of software are tested in unit testing, which is a sort of software testing. The goal is to ensure that each unit of software code works as intended. Unit Testing is done during the development (coding phase) of an application by the developers. Unit tests are used to isolate a part of code and ensure that it is correct. A singular function, method, procedure, module, or object might be considered a unit. Unit testing is the first step in the testing process before moving on to integration testing.

Because software developers sometimes want to save time by doing limited unit testing, this is a misconception because insufficient unit testing leads to large costs in defect correction during System Testing, Integration Testing, and even Beta Testing after the program has been constructed. It saves time if proper unit testing is done early in the development process.

Table 1: Unit Testing

Function Name	Test Results
Load the model file	Insert the trained model file to the code
Prediction	Predicted values of currency
Display result	Output is displayed successfully

2) Validation Testing

The process of determining if software meets stated business requirements during the development process or at the end of the development process.Validation testing guarantees that the product fits the needs of the customer. It can also be defined as demonstrating that the product performs as expected when used in the right setting.

3) Integration Testing

Integration testing is a sort of testing in which software modules are conceptually linked and tested as a unit. A typical software project is made up of several software modules written by various programmers. The goal of this level of testing is to find flaws in the way various software modules interact when they're combined.

Although each software module is unit tested, defects still exist for various reasons like

- In general, a Module is created by a single software developer whose programming logic and understanding may differ from that of other programmers. Integration testing is required to ensure that the software parts work together.
- There's a good risk that the clients' needs will alter throughout module development. These new needs may not be unit tested necessitating system integration testing.
- There's a chance that the software modules database interfaces are incorrect.
- External hardware interfaces, if present, may be incorrect.
- Inadequate exception handling may result in problems.

4) User Acceptance testing

User Acceptance Testing (UAT) is a sort of testing in which the end user or customer verifies and accepts the software system before it is moved to the production environment.

After functional, integration, and system testing, UAT is performed in the final step of testing An acceptance test's performance is essentially the user's show. User motivation and knowledge are essential for the system's proper operation. The aforesaid tests were carried out on the newly designed system, which met all of the requirements. The following test case designs were used for all of the above testing methodologies.

5.4.3 Testing Cases

Table 2: Input Frame Test Case

Test Case	**1**
Name of Test	Input frame/image
Input	Load the input images/frame
Expected output	Image is been fetched for further process
Actual output	Image is been fetched for further process
Result	Successful

Table 3: Convert to grayscale test Case

Test Case	**1**
Name of Test	Convert to gray scale
Input	Input images
Expected output	Convert RGB to gray scale image
Actual output	Convert RGB to gray scale image
Result	Successful

Table 4: Detect edge Test Case

Test Case	**1**
Name of Test	Detect Edge
Input	Gray scale image
Expected output	Detecting edge
Actual output	Detecting edge
Result	Successful

Table 5: Output frame test case

Test Case	**1**
Name of Test	Output Frame
Input	Image frame and converted image
Expected output	Output frame
Actual output	Output frame
Result	Successful

5.5 Results

In this section, the results 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.

The proposed system authenticates the input image of currency note through image processing.

The input image passes through various algorithms in which the image is processed and each extracted feature is thoroughly examined. The results are calculated in the following manner:

Algorithm 1 (Feature 1-7): This algorithm finally col- lects the average SSIM score and the max. SSIM score for each feature. A feature passes the test and is real if the average SSIM score is greater than a minimum value (this value has to be decided after proper testing). A feature also passes the test if the max. SSIM score is too high (probably greater than 0.8).

Algorithm 2 (Feature 8-9): This algorithm finally returns the average number of bleed lines present in the left and right sides of a currency note. Each feature passes the test if the average number of bleed lines is closer to 5, in case of Rs 500 currency note, and 7, in case of 2000 currency note.

Algorithm 3 (Feature 10): This algorithm finally returns the number of characters present in the number panel of the currency note. This feature passes the test if the number of characters detected is equal to 9 (for at least one threshold value).

Figure 5.5 to Figure 5.16 are snapshots of the GUI used in our implementation.

Figure 5.5 Initially no image is displayed and user is asked to insert image

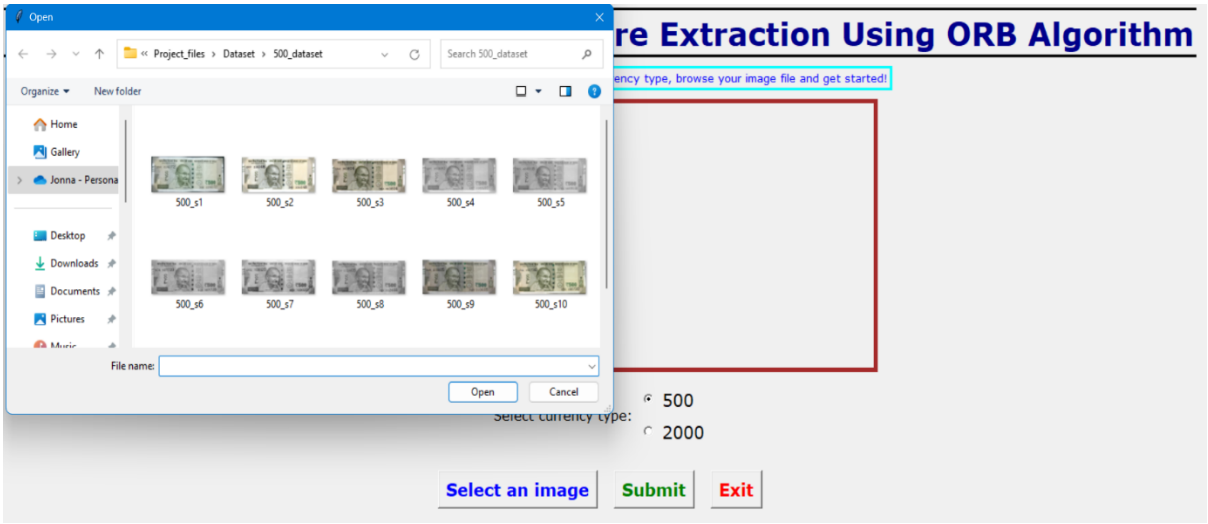

Figure 5.6 Browsing Image

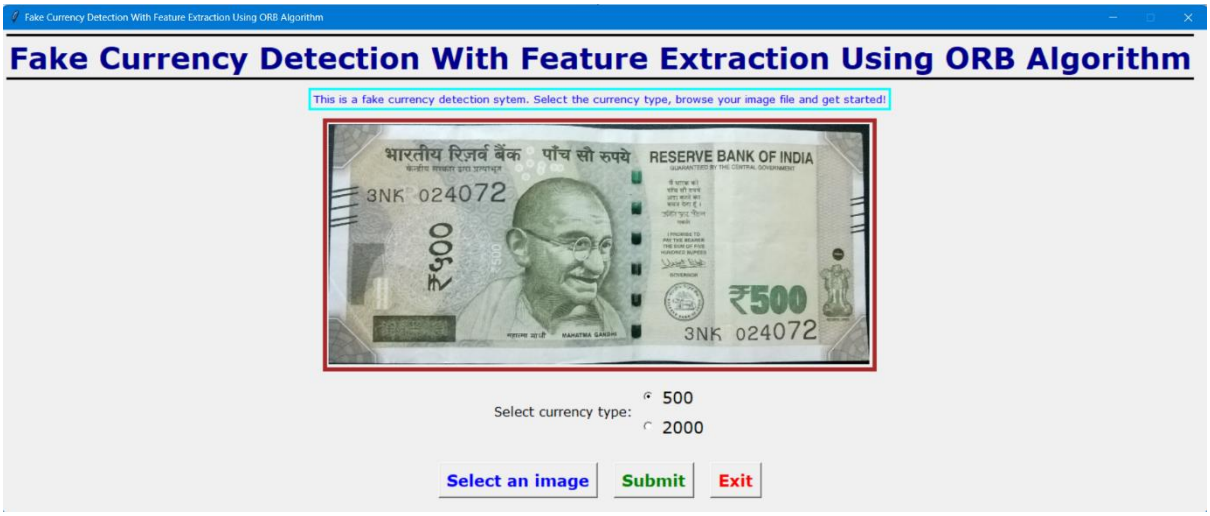

Figure 5.7 Input image of currency note

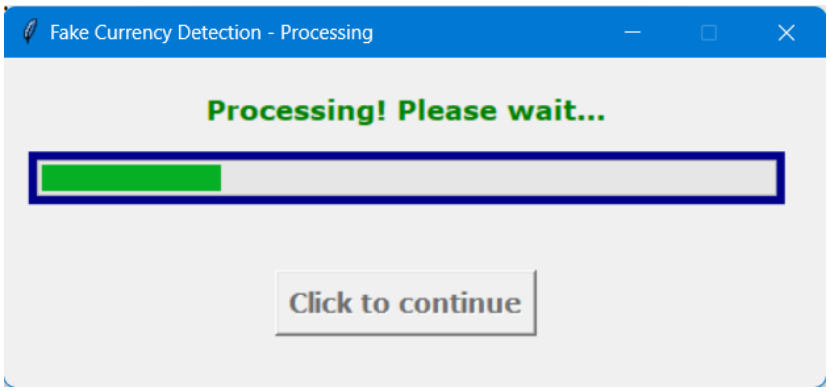

Figure 5.8 Sent for processing

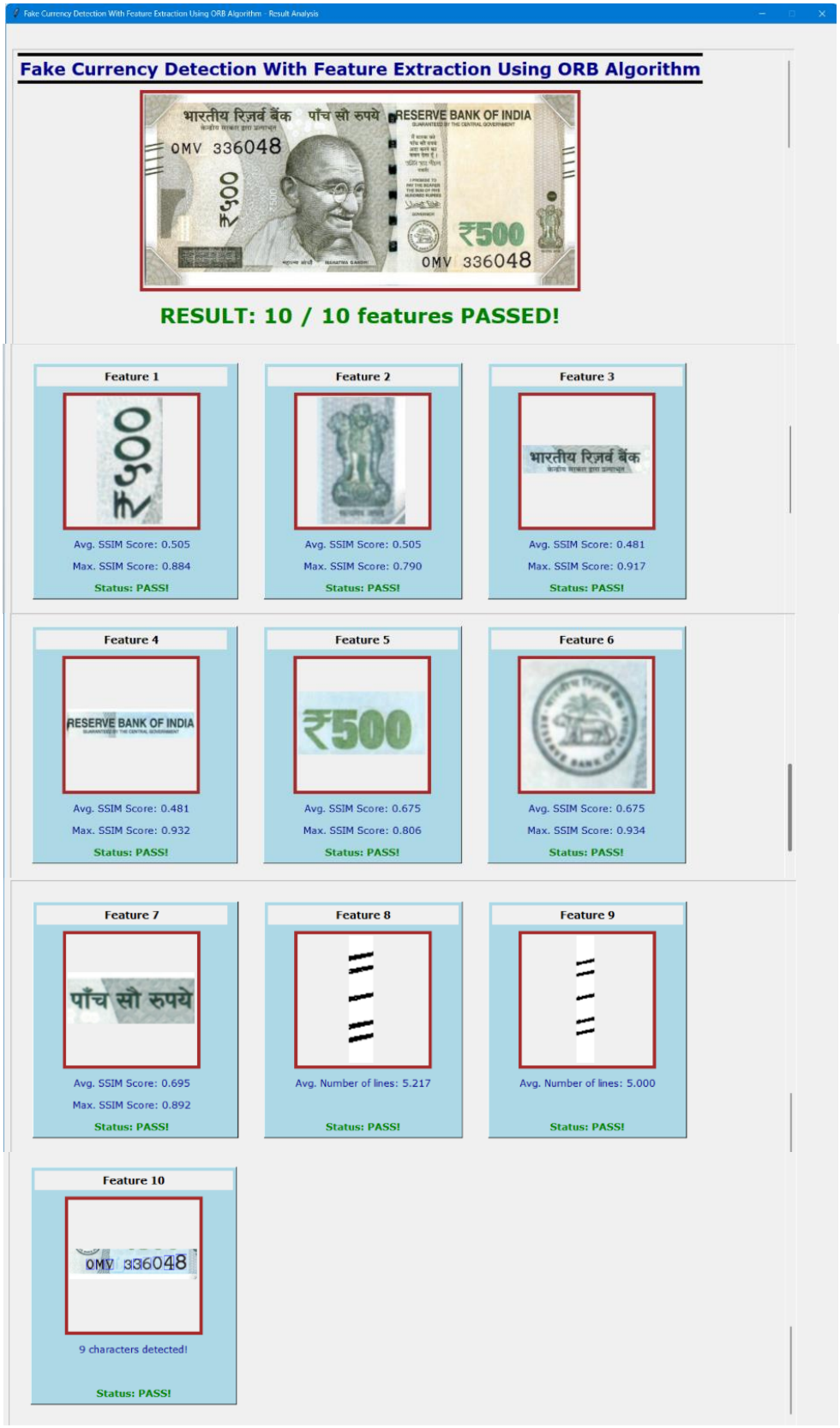

Figure 5.9 GUI showing final result (Real note)

```
Final Score- set list:

Feature 1 : [0.25329650133225556, 0.8840734055096348, 0.5407302962273, 0.28037888640473024, 0.5319809804974847, 0.5396722
459190744]
Feature 2 : [0.5270556717541895, 0.7898821587433548, 0.39033380826552266, 0.21847479699305197]
Feature 3 : [0.7408370790148724, 0.9166765860566546, 0.7161135958259065, 0.5374118189058186, 0.5778185764243041, 0.561223
3917957087]
Feature 4 : [0.4949473605864941, 0.9321880084041907, 0.7801497280896231, 0.6676218480861739, 0.6122363764039539, 0.683319
4979061994]
Feature 5 : [0.6259712370097882, 0.8063367331368295, 0.7877045483757704, 0.7380970284295493]
Feature 6 : [0.43789124347355896, 0.9343732518606919, 0.6575671584672279, 0.4484620950411458, 0.27577529865020156]
Feature 7 : [0.5518559055143, 0.8920252696065883, 0.6861747028804408, 0.6122361611411877, 0.6570448122453043, 0.586326269
837529]
```

```
Number of columns examined:  23
Number of non- erroneous columns found:  23

Average number of black regions is:  5.217391304347826
```

```
Number of columns examined:  20
Number of non- erroneous columns found:  20

Average number of black regions is:  5.0

ANALYSIS OF FEATURE 10 : NUMBER PANEL

---> Threshold 1 with Threshold value 95 :
Unsuccessful!
---> Threshold 9 with Threshold value 100 :
Unsuccessful!
---> Threshold 9 with Threshold value 105 :
Test Successful: 9 letters found!
---> Threshold 10 with Threshold value 110 :
Test Successful: 9 letters found!
---> Threshold 10 with Threshold value 115 :
Test Successful: 9 letters found!
Test Passed!- 9 characters were detected in the serial number panel.
RESULT ANALYSIS

Feature 1: Successful
Feature 2: Successful
Feature 3: Successful
Feature 4: Successful
Feature 5: Successful
Feature 6: Successful
Feature 7: Successful
Feature 8: Successful- 5 bleed lines found in left part of currency note
Feature 9: Successful- 5 bleed lines found in right part of currency note
Feature 10: Successful- 9 characters found in number panel of currency note

Result Summary:
10 out of 10 features are VERIFIED!
Stored 'result_list' (list)
```

Figure 5.10 Result Analysis of real 500 currency note

Figure 5.11 GUI showing final result (Fake note)

```
Feature 1 : [0.29168697136248756, 0.056853812848283364, 0.23827019781626813, 0.33766875919375394, 0.308570731504678]
Feature 2 : [0.37147435650766814, 0.2888311857599636, 0.2725983085809651, 0.3890728558076636, 0.3503883233100522, 0.40374
689771922023]
Feature 3 : [0.7323244907506803, 0.9143517808446638, 0.7246059029807199, 0.5131867263257835, 0.5689787380867782, 0.550307
6553806933]
Feature 4 : [0.464267421599463, 0.8768302328784651, 0.7747898334945383, 0.6050145984354287, 0.5942113444759779, 0.6943556
005788478]
Feature 5 : [0.6441844414495458, 0.8262172147758545, 0.808841089237395, 0.6604895587375484, 0.6445461447870275]
Feature 6 : [0.6763931779504856, 0.6001354710430151, 0.7487246222142622, 0.5666141622243465, 0.3419501181311942]
Feature 7 : [0.462262336184423, 0.9281649846233918, 0.6257811834993164, 0.5881611794194166, 0.6409173772175031, 0.5664761
202387192]
```

```
Number of columns examined:  20
Number of non- erroneous columns found:  20

Average number of black regions is:  3.05
Number of columns examined:  23
Number of non- erroneous columns found:  23

Average number of black regions is:  3.0869565217391304


ANALYSIS OF FEATURE 10 : NUMBER PANEL

---> Threshold 1 with Threshold value 95 :
Unsuccessful!
---> Threshold 9 with Threshold value 100 :
Test Successful: 9 letters found!
---> Threshold 10 with Threshold value 105 :
Test Successful: 9 letters found!
---> Threshold 10 with Threshold value 110 :
Test Successful: 9 letters found!
Test Passed!- 9 characters were detected in the serial number panel.


RESULT ANALYSIS

Feature 1: Unsuccessful
Feature 2: Unsuccessful
Feature 3: Successful
Feature 4: Successful
Feature 5: Successful
Feature 6: Successful
Feature 7: Successful
Feature 8: Unsuccessful!
Feature 9: Unsuccessful!
Feature 10: Successful- 9 characters found in number panel of currency note

Result Summary:
6 out of 10 features are VERIFIED!
```

Figure 5.12 Result analysis of 500 fake currency note

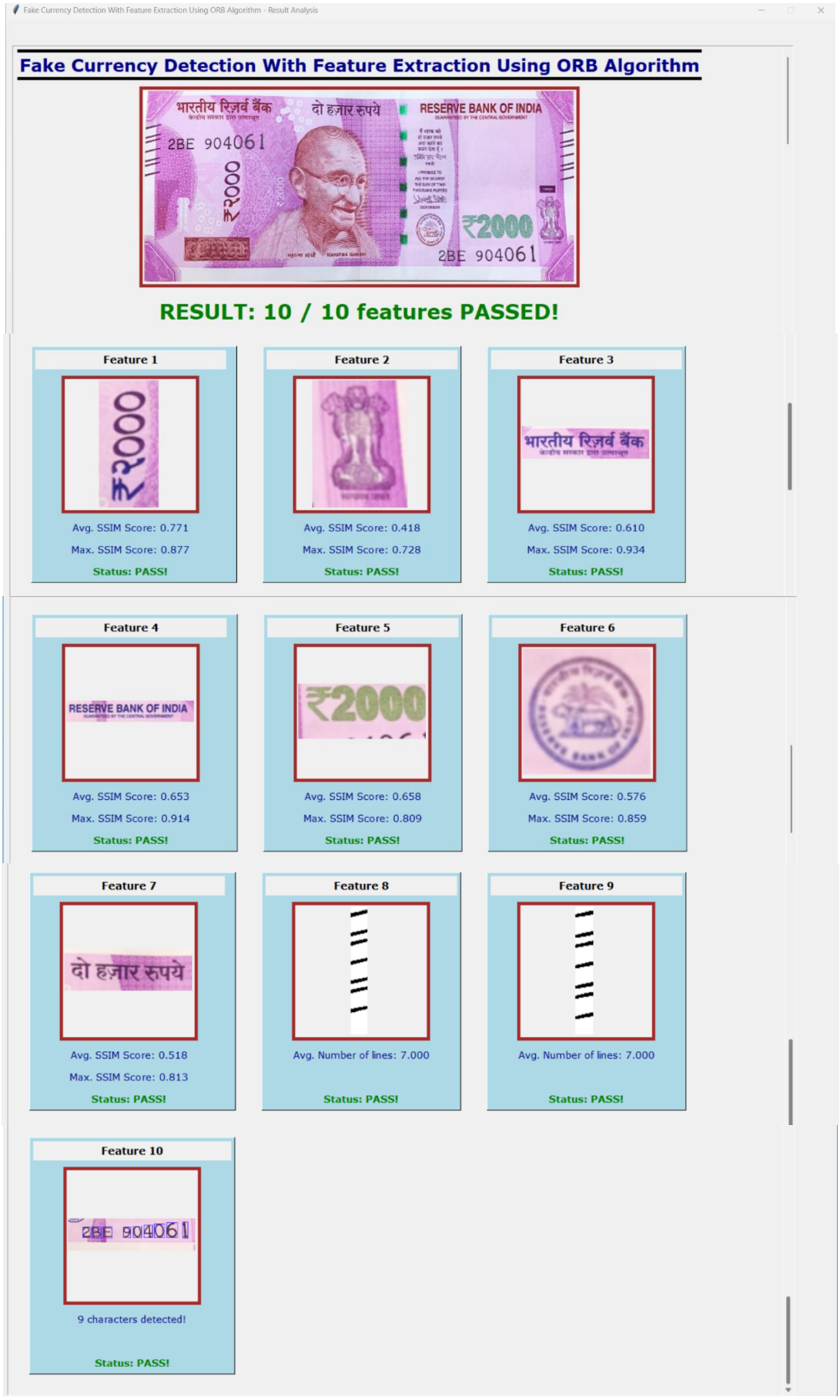

Figure 5.13 GUI showing final result for 2000 currency note

```
Final Score- set list:

Feature 1 : [0.8768639144981325, 0.7292631586285556, 0.7508656570701558, 0.7322303337864629, 0.8334965111329322, 0.7052532095
098544]
Feature 2 : [0.7279643584766854, 0.24857131667920093, 0.5424998826541236, 0.2024159064261172, 0.4766506124310216, 0.311367914
711798]
Feature 3 : [0.9339527104073755, 0.5149441631132722, 0.4528280871132562, 0.5581367295383562, 0.5579608097831591, 0.6437104155
105735]
Feature 4 : [0.9138426978521372, 0.8184054440154294, 0.6815077598517413, 0.45465507658774906, 0.39771191950459617]
Feature 5 : [0.8094922559201554, 0.5342628991072657, 0.7972661867106916, 0.5879375678629434, 0.6718042505188937, 0.5466612608
24316]
Feature 6 : [0.8586384414625262, 0.45189214955995843, 0.329863687167565, 0.5524421518036705, 0.5361709404608963, 0.7288088460
385562]
Feature 7 : [0.8134974042374281, 0.2726587468800321, 0.5452537567066095, 0.3768855549151073, 0.5398981198205617, 0.5609925801
399116]
Number of columns examined:  20
Number of non- erroneous columns found:  20

Average number of black regions is:  7.0
Number of columns examined:  20
Number of non- erroneous columns found:  20

Average number of black regions is:  7.0

ANALYSIS OF FEATURE 10 : NUMBER PANEL

---> Threshold 1 with Threshold value 90 :
Unsuccessful!
---> Threshold 2 with Threshold value 95 :
Unsuccessful!
---> Threshold 3 with Threshold value 100 :
Unsuccessful!
---> Threshold 4 with Threshold value 105 :
Test Successful: 9 letters found!
---> Threshold 5 with Threshold value 110 :
Test Successful: 9 letters found!
---> Threshold 6 with Threshold value 115 :
Test Successful: 9 letters found!
Test Passed!- 9 characters were detected in the serial number panel.

RESULT ANALYSIS

Feature 1: Successful
Feature 2: Successful
Feature 3: Successful
Feature 4: Successful
Feature 5: Successful
Feature 6: Successful
Feature 7: Successful
Feature 8: Successful- 7 bleed lines found in left part of currency note
Feature 9: Successful- 7 bleed lines found in right part of currency note
Feature 10: Successful- 9 characters found in number panel of currency note

Result Summary:
10 out of 10 features are VERIFIED!
```

Figure 5.14 Result analysis of real 2000 currency note

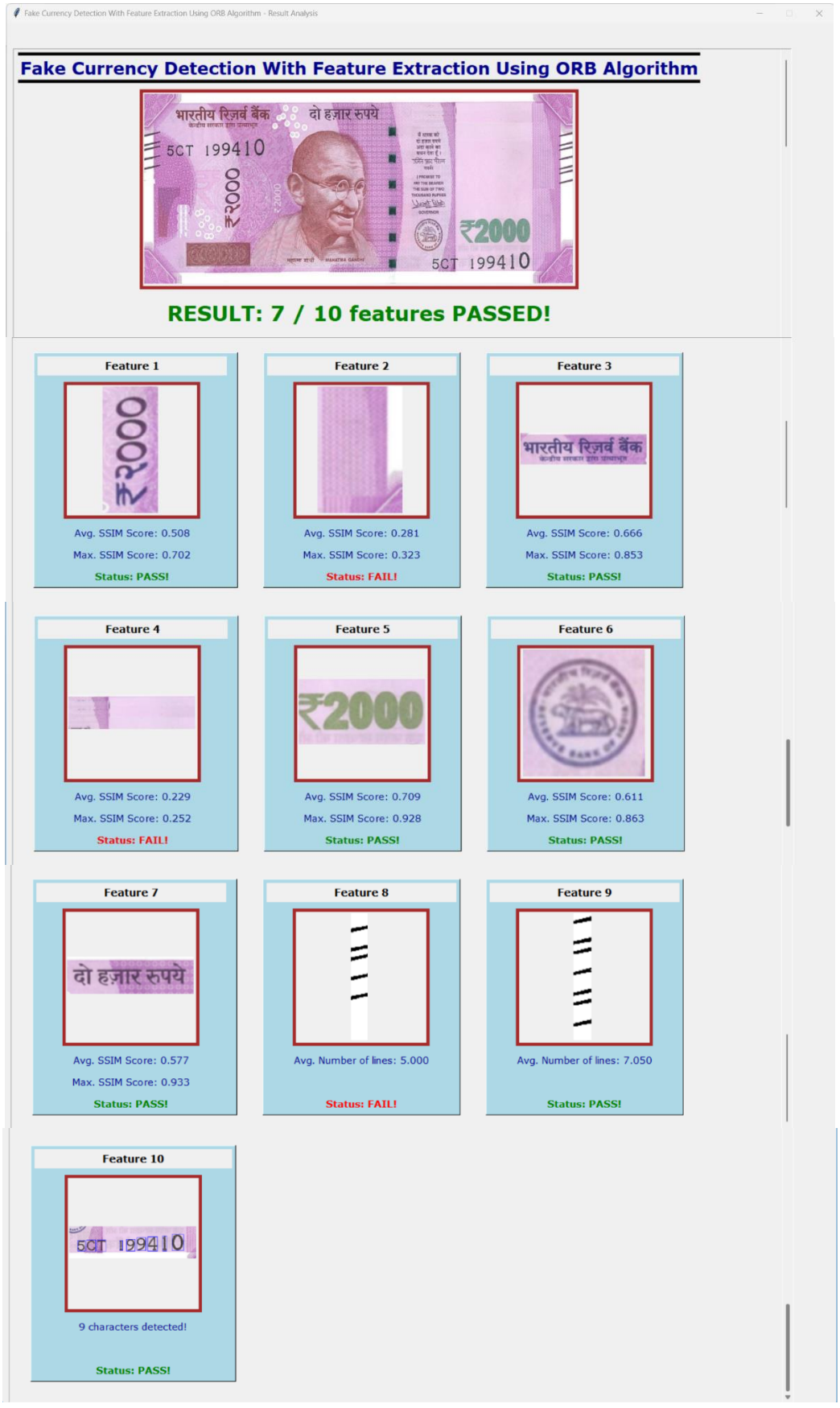

Figure 5.15 GUI showing final result for 2000 fake note

```
Final Score- set list:

Feature 1 : [0.5787145081258045, 0.537036213360286, 0.37963611666595487, 0.5355502374090146, 0.7018671786286825, 0.3177723152
4301097]
Feature 2 : [0.2934782076277512, 0.31168516502620924, 0.3227461898516194, 0.2625611421143617, 0.21695305154480304]
Feature 3 : [0.39839748635097255, 0.5679722942936346, 0.7527178417240362, 0.8530087036809987, 0.706326427725009, 0.7146347505
414843]
Feature 4 : [0.25172588558061554, 0.20719671356139235]
Feature 5 : [0.5193281749009976, 0.7268199891139163, 0.6944184142714459, 0.9283627819421696, 0.7849336627224742, 0.5988690467
238876]
Feature 6 : [0.13970731392845365, 0.6598323463131567, 0.693076388530157, 0.8602707373746893, 0.8627647858766737, 0.4503561305
955691]
Feature 7 : [0.24437831613617425, 0.32338731142865884, 0.6209178796838989, 0.9327344079160317, 0.7316515978153265, 0.61141490
33073787]

Number of columns examined:  20
Number of non- erroneous columns found:  20

Average number of black regions is:  5.0

ANALYSIS OF FEATURE 9 : RIGHT BLEED LINES
Number of columns examined:  20
Number of non- erroneous columns found:  20

Average number of black regions is:  7.05

ANALYSIS OF FEATURE 10 : NUMBER PANEL

---> Threshold 1 with Threshold value 90 :
Unsuccessful!
---> Threshold 2 with Threshold value 95 :
Unsuccessful!
---> Threshold 3 with Threshold value 100 :
Unsuccessful!
---> Threshold 4 with Threshold value 105 :
Unsuccessful!
---> Threshold 5 with Threshold value 110 :
Unsuccessful!
---> Threshold 6 with Threshold value 115 :
Test Successful: 9 letters found!
---> Threshold 7 with Threshold value 120 :
Test Successful: 9 letters found!
---> Threshold 8 with Threshold value 125 :
Test Successful: 9 letters found!
Test Passed!- 9 characters were detected in the serial number panel.

RESULT ANALYSIS

Feature 1: Successful
Feature 2: Unsuccessful
Feature 3: Successful
Feature 4: Unsuccessful
Feature 5: Successful
Feature 6: Successful
Feature 7: Successful
Feature 8: Unsuccessful!
Feature 9: Successful- 7 bleed lines found in right part of currency note
Feature 10: Successful- 9 characters found in number panel of currency note

Result Summary:
7 out of 10 features are VERIFIED!
```

Figure 5.16 Result analysis of fake 2000 currency note

5.6 Software Code

```
#  Testing of Rs. 500 currency notes
#similar code is used for 2000 currenct notes with different dataset
This notebook carries out the complete evaluation of an input Rs. 500 currency note
# Importing all necessary libraries

import cv2                          # Importing Opencv library for image processing and computer vision
import numpy as np                  # Importing numpy library
import matplotlib.pyplot as plt       # Importing matplotlib library to plot the images directly in
notebook
from skimage.metrics import structural_similarity as ssim   # Importing ssim calculating modules
from skimage library

# Importing tkinter library to build GUI
from tkinter import *
from tkinter.ttk import Progressbar

import time

#Resizing the Plots
plt.rcParams["figure.figsize"] = (12, 12)
# Declaring variable for progress bar to store the total progress
myProgress =0.0
# This Ipython magic will retrieve the path of the input image which is stored when the gui_1.ipynb is
executed.

%store -r path

# A sample path:
# path = r'Dataset\500_dataset\500_s1.jpg'

print('Path of input image: ', path)
# Reading the image
test_img = cv2.imread(path)
# Pre- processing
# 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()
#  Calculating SSIM of the two images sent as parameters

def calculateSSIM(template_img, query_img):
    min_w = min(template_img.shape[1], query_img.shape[1])
    min_h = min(template_img.shape[0], query_img.shape[0])

    # Resizing the two images so that both have same dimensions
    img1 = cv2.resize(template_img, (min_w, min_h))
    img2 = cv2.resize(query_img, (min_w, min_h))

    # Conversion to gray- scale
    img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Plotting the images
    plt.subplot(1, 2, 1)
    plt.imshow(img1, 'gray')

    plt.subplot(1, 2, 2)
    plt.imshow(img2, 'gray')

    plt.show()

    # Find the SSIM score and return
    score = ssim(img1, img2)
    return score
# Feature detection using ORB
def computeORB(template_img, query_img):
    # ==================== Creating orb object ============================
    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)
```

```
    #  Brute Force Matching
    # 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)

    # Image homography
    ## extract the matched keypoints
    dst_pts = None
    if len(dmatches) >= 4:
        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)

        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
    else:
        print("Not enough matched keypoints to find homography.")
        dst = None

    # This finds the template region in the test currency note
    # Returning necessary data
    return dst, dst_pts, kpts1, kpts2, dmatches
# Values for specifying search area of features 1 to 7
search_area_list = [[200,300,200,370],
                    [1050,1500,300,450],
                    [100,450,20,120],
                    [690,1050,20,120],
                    [820,1050,350,430],
                    [700,810,330,430],
                    [400,650,0,100]]

# Values of max_area and min_area for each feature for features 1 to 7
feature_area_limits_list = [[12000,17000],
                            [10000,18000],
                            [20000,30000],
                            [24000,36000],
                            [15000,25000],
                            [7000,13000],
                            [11000,18000]]

score_set_list = []             # Stores the ssim score set of each feature
best_extracted_img_list = []    # Stores the extracted image with highest SSIM score for each feature
```

```
avg_ssim_list = []              # Stores the avg ssim value for each feature
NUM_OF_FEATURES = 7             # Number of features
# 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
                # 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_img
```

```
            #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])

left_BL_result = []
right_BL_result = []
result_list = []
number_panel_result = []
# Feature 8: Left Bleed Lines
# 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()
# Feature 9: Right Bleed Lines
# Function to count number of bleed lines in right side- Feature 9

def testFeature_9():
    plt.rcParams["figure.figsize"] = (5, 5)

    # Check Feature 9- Right bleed lines
    print('\nANALYSIS OF FEATURE 9 : RIGHT BLEED LINES\n')

    # Cropping the region in which left bleed lines are present- Feature extraction
    crop = test_img[120:260, 1135:1155]

    img = crop.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Thresholding the image
    _, 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 right_BL_result
    right_BL_result = [thresh, average_count]
    global myProgress

    # Updating progess in progress bar
    progress['value']=85
    ProgressWin.update_idletasks()
# Feature 10: Currency Number Panel
# Cropping out the number panel.

# Cropping out the number panel. - FEATURE EXTRACTION
```

```
crop = gray_test_image[410:500, 700:1080]          # This is the cropped gray image
crop_bgr = test_img[410:500, 700:1080]             # This is the cropped BGR image
# 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()
# Results
# Result analysis

def testResult():
    plt.rcParams["figure.figsize"] = (3, 3)

    print('\n\nRESULT ANALYSIS\n')

    # Stores the min allowed SSIM score for each feature
    min_ssim_score_list = [0.4, 0.4, 0.5, 0.4, 0.5, 0.45, 0.5]

    global result_list
    result_list = []              # To store details of each feature
    successful_features_count = 0  # To store number of features which passed the test

    # Feature 1 to 7: Results
    for i in range(NUM_OF_FEATURES):
        avg_score = avg_ssim_list[i]
        img, max_score = best_extracted_img_list[i]
        status = False
```

```
    min_allowed_score = min_ssim_score_list[i]

    # A feature passes the test if its avg SSIM score is greater than a min. decided value
    # or if its max SSIM score is greater than 0.8
    if avg_score >= min_allowed_score or max_score >= 0.79:
        status = True
        successful_features_count += 1
        print('Feature ' + str(i+1) + ': Successful')
    else:
        status = False
        print('Feature ' + str(i+1) + ': Unsuccessful')

    if img is None:
        img = cv2.imread('Image_not_found.jpg')

    result_list.append([img, avg_score, max_score, status])


# Feature 8: Left Bleed lines

img, line_count = left_BL_result[:]

# The feature passes the test if number of bleed lines is close to 7 (6.7 - 7.6)
if line_count >= 4.7 and line_count <= 5.6:
    status = True
    successful_features_count += 1
    print('Feature 8: Successful- 5 bleed lines found in left part of currency note')
else:
    status = False
    print('Feature 8: Unsuccessful!')

if img is None:
    img = cv2.imread('Image_not_found.jpg')
result_list.append([img, line_count, status])


# Feature 9: Right Bleed lines
img, line_count = right_BL_result[:]

# The feature passes the test if number of bleed lines is close to 7 (6.7 - 7.6)
if line_count >= 4.7 and line_count <= 5.6:
    status = True
    successful_features_count += 1
    print('Feature 9: Successful- 5 bleed lines found in right part of currency note')
else:
    status = False
    print('Feature 9: Unsuccessful!')

if img is None:
    img = cv2.imread('Image_not_found.jpg')
result_list.append([img, line_count, status])


# Feature 10: Currency Number Panel
```

```
    img, status = number_panel_result[:]
    # The feature passes the test if 9 characters are detected in the number panel
    if status:
        successful_features_count += 1
        print('Feature 10: Successful- 9 characters found in number panel of currency note')
    else:
        print('Feature 10: Unsuccessful!')

    if img is None:
        img = cv2.imread('Image_not_found.jpg')
    result_list.append([img, status])

    # printing FINAL RESULT

    print('\nResult Summary:')
    print(str(successful_features_count) + ' out of 10 features are VERIFIED!')

    # Updating progress bar
    global myProgress
    progress['value']=97
    ProgressWin.update_idletasks()
# Driver cell
# creating tkinter window
# Call all testing functions
def Testing():
    button.config(state = DISABLED)
    result_list.clear()
    preprocessing()
    testFeature_1_2_7()
    testFeature_8()
    testFeature_9()
    testFeature_10()
    testResult()
    progress['value']=100
    ProgressWin.update_idletasks()
    time.sleep(0.8)
    ProgressWin.destroy()

def exitGUI():
    ProgressWin.destroy()

# creating tkinter window
ProgressWin = Tk()
ProgressWin.title("Processing Image")
ProgressWin.title('Fake Currency Detection - Processing')


# Defining attributes of root window
ProgressWin.resizable(False, False)  # This code helps to disable windows from resizing

window_height = 200
window_width = 500

screen_width = ProgressWin.winfo_screenwidth()
```

```
screen_height = ProgressWin.winfo_screenheight()
x_cordinate = int((screen_width/2) - (window_width/2))
y_cordinate = int((screen_height/2) - (window_height/2))

ProgressWin.geometry("{}x{}+{}+{}".format(window_width, window_height, x_cordinate,
y_cordinate))

# Creating a main frame inside the root window
main_frame=Frame(ProgressWin,relief=GROOVE)
main_frame.place(x=10,y=10) # Placing the frame at (10, 10)

# Creating sub- frames
frame1 = Frame(main_frame, padx=3, pady=3)
frame2 = Frame(main_frame, bg='dark blue', pady=5, padx = 5)
frame3 = Frame(main_frame, pady=5, padx = 5)

frame1.grid(row = 1, column = 1, padx = 5, pady = 5)
frame2.grid(row = 2, column = 1, padx = 5, pady = 5)
frame3.grid(row = 3, column = 1, padx = 5, pady = 30)


# Title label in sub_frame1
label = Label(master=frame1, text="Processing! Please wait...", fg = 'green', font = "Verdana 13
bold")
label.pack() # Put the label into the window
# Progress bar widget
progress = Progressbar(frame2, orient = HORIZONTAL, length = 450, mode = 'determinate')
progress.pack()
# Button widget
button = Button(frame3, text = 'Click to continue', command = Testing, font = "Verdana 12 bold",
pady = 5)
button.pack()

# Run the GUI
ProgressWin.mainloop()
# Store the list containing the final result of each feature

%store result_list
```

5.7 Conclusion

In this chapter we have seen the schematic diagram, flowchart, experimental setup of the project and their results.

CHAPTER 6
CONCLUSION AND FUTURE SCOPE

This project has successfully developed a user-friendly graphical user interface for Fake currency detection with feature extraction using ORB algorithm. The application allows user to load a pre-trained testing python model, enabling quick and efficient currency authentication without need for model training again and again.

The final result is displayed, which helps in prediction of currency whether its real or fake. In addition, this includes a feature to visualize the model's training performance through accuracy and graphs.

We plan to integrate more sophisticated algorithms and expand our system to support a wider range of currency types and denominations worldwide. In addition to visual feedback, we envision enhancing accessibility features by refining and customizing the sound outcomes, ensuring a seamless and inclusive experience for all user, including those with disabilities. By continuously refining our design and implementation, we strive to create a robust platform that empowers users to interact with currency information with even greater efficiency and inclusivity in the future.

CHAPTER 7

REFERENCES

[1] S. R. Darade and G.Gidveer, “Automaticrecognition of fake indian currency note,” in *2016 international conference on Electrical Power and Energy Systems (ICEPES)*. IEEE, 2016, pp. 290–294.

[2] B. P. Yadav, C. Patil, R. Karhe, and P. Patil, “An automatic recognition of fake indian paper currency note using matlab,” *Int. J. Eng. Sci. Innov. Technol*, vol. 3, pp. 560–566, 2014.

[3] A. Zarin and J. Uddin, “A hybrid fake banknote detection model using ocr, face recognition and hough features,” in *2019 Cybersecurity and Cyberforensics Conference (CCC)*. IEEE, 2019, pp. 91–95.

[4] M. S. Veling, M. J. P. Sawal, M. S. A. Bandekar, M. T. C. Patil, and M. A. L. Sawant, “Fake indian currency recognition system by using matlab.”

[5] F. A. B, P. Mangayarkarasi, Akhilendu, A. A. S, and M. K, “Fake Indian currency note recognition,” vol. 7, pp. 4766–4770, 2020. [Online]. Available: https://www.irjet.net/archives/V7/i5/IRJET-V7I5915.pdf

[6] “Introduction to ORB (FAST and BRIEF)”, https://medium.com/@deepanshut041/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[7] B. P. Yadav, C. Patil, R. Karhe, and P. Patil, “An automatic recognition of fake indian paper currency note using matlab,” *Int. J. Eng. Sci. Innov. Technol*, vol. 3, pp. 560–566, 2014.