A

Project Report on

IDENTIFICATION OF FAKE INDIAN CURRENCY USING CONVOLUTIONAL NEURAL NETWORK

Submitted to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR, ANANTAPURAMU

in partial fulfillment of the requirements for the award of the Degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

KALANJERI BHUVANESWARI	(219E1A0589)
JOGIREDDYGARI SAI BHAVANA	(219E1A0584)
JERRI RAJESH	(219E1A0583)
KALUVA ABDULLA	(219E1A0595)

Under the Guidance of

Dr. A. SARITHA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SRI VENKATESWARA ENGINEERING COLLEGE

(Approved by AICTE, New Delhi NBA &NAAC Accredited Institution with UGC section 2(f) & 12(b) & Affiliated to JNTUA, Ananthapuramu)

Karakambadi Road, TIRUPATI – 517507 (2021-2025)

SRI VENKATESWARA ENGINEERING COLLEGE

(Approved by AICTE, New Delhi NBA &NAAC Accredited Institution with UGC section 2(f) & 12(b) & Affiliated to JNTUA, Ananthapuramu)

Karakambadi Road, TIRUPATI – 517507

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project report entitled "**IDENTIFICATION OF FAKE INDIAN CURRENCY USING CONVOLUTIONAL NEURAL NETWORK**" is a bonafide record of the project work done and submitted by

KALANJERI BHUVANESWARI	**(219E1A0589)**
JOGIREDDYGARI SAI BHAVANA	**(219E1A0584)**
JERRI RAJESH	**(219E1A0583)**
KALUVA ABDULLA	**(219E1A0595)**

for the partial fulfillment of the requirements for the award of B.Tech Degree in COMPUTER SCIENCE AND ENGINEERING, JNT University Anantapur, Ananthapuramu.

GUIDE:
Dr. A. SARITHA, B. Tech, M. Tech, Ph.D
Professor
Dept. of. CSE

HEAD OF THE DEPARTMENT:
Dr. A. GANESH, B. Tech, M. Tech, Ph.D (PDF)
Professor & HOD
Dept. of. CSE

External Viva-Voce Exam Held on _____________

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We hereby declare that the project report entitled "**IDENTIFICATION OF FAKE INDIAN CURRENCY USING CONVOLUTIONAL NEURAL NETWORK**" done by us under the guidance of **Dr.A.SARITHA**, and is submitted in partial fulfillment of the requirements for the award of the Bachelor's degree in **Computer Science and Engineering**. This project is the result of our own effort and it has not been submitted to any other University or Institution for the award of any degree or diploma other than specified above.

KALANJERI BHUVANESWARI	**(219E1A0589)**
JOGIREDDYGARI SAI BHAVANA	**(219E1A0584)**
JERRI RAJESH	**(219E1A0583)**
KALUVA ABDULLA	**(219E1A0595)**

ACKNOWLEDGEMENT

We are thankful to our guide **Dr.A.SARITHA** for her valuable guidance and encouragement.Her helping attitude and suggestions helped us in the successful completion of the project .

We would like to express our gratefulness and sincere thanks to **Dr.A.GANESH**, Head of the Department of **COMPUTER SCIENCE AND ENGINEERING**, for his kind help and encouragement during the course of our study and in the successful completion of the project work.

We have great pleasure in expressing our hearty thanks to our beloved Principal **Dr. C. CHANDRASEKHAR**, for spending his valuable time with us to complete this project. Successful completion of any project cannot be done without proper support and encouragement.

We sincerely thank to the **MANAGEMENT** for providing all the necessary facilities during the course of study.

We would like to thank our parents and friends, who have the greatest contributions in all our achievements, for the great care and blessings in making us successful in all our endeavors.

KALANJERI BHUVANESWARI	**(219E1A0589)**
JOGIREDDYGARI SAI BHAVANA	**(219E1A0584)**
JERRI RAJESH	**(219E1A0583)**
KALUVA ABDULLA	**(219E1A0595)**

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	– Artificial Intelligence
ML	– Machine Learning
NLP	– Natural Language Processing
OCR	– Optical Character Recognition
CNN	– Convolutional Neural Network
RNN	– Recurrent Neural Network
LSTM	– Long Short-Term Memory
API	– Application Programming Interface
GUI	– Graphical User Interface
TF-IDF	– Term Frequency-Inverse Document Frequency
SVM	– Support Vector Machine

ABSTRACT

The spread of counterfeit currency is a major threat to economic stability, requiring advanced methods for accurate detection. This project, titled "Identification of Fake Indian Currency Using Convolutional Neural Networks," introduces a new approach to detect fake currency using deep learning techniques. It focuses on three main models: MobileNet, ResNet, and a hybrid model that combines MobileNet with Support Vector Machines (SVM). Another version of the hybrid model combines MobileNet with both SVM and Random Forest. MobileNet and ResNet, known for being efficient and accurate in recognizing images, are tested to see how well they can distinguish real Indian currency from counterfeit notes. The hybrid model improves detection by combining the strengths of MobileNet and SVM, making it better at handling complex fake currency patterns. Adding SVM and Random Forest to MobileNet further boosts the model's ability to classify currency accurately by using ensemble learning techniques. These models are evaluated based on how accurate, precise, reliable, and robust they are in real-world situations. The results show how convolutional neural networks can significantly improve counterfeit currency detection, offering valuable insights to strengthen financial security.

Keywords: Counterfeit Detection, Convolutional Neural Networks, MobileNet, Support Vector Machines (SVM), Random Forest, Hybrid Model, Indian Currency, Image Classification, Machine Learning.

KALANJERI BHUVANESWARI	**(219E1A0589)**
JOGIREDDYGARI SAI BHAVANA	**(219E1A0584)**
JERRI RAJESH	**(219E1A0583)**
KALUVA ABDULLA	**(219E1A0595)**

1. INTRODUCTION

1.1 OBJECTIVE OF PROJECT:

The objective of this project is to develop a sophisticated counterfeit detection system for Indian currency using advanced Convolutional Neural Networks (CNNs). By employing models such as MobileNet, and integrating hybrid approaches combining CNNs with Support Vector Machines (SVM) and Random Forest, the project aims to enhance the accuracy and efficiency of counterfeit detection. The system will be designed to automatically and reliably distinguish between genuine and fake notes, minimizing human error and improving overall security. The ultimate goal is to provide a scalable, high-performance solution that strengthens the financial system's resilience against currency fraud.

1.2 PROBLEM STATEMENT:

Counterfeit currency poses a significant challenge to financial systems, leading to economic losses and undermining trust in the currency. Current detection methods often rely on manual inspection or basic machine learning techniques, which may lack precision and efficiency. This project addresses the problem of detecting fake Indian currency by leveraging advanced Convolutional Neural Networks (CNNs) such as MobileNet. It also explores hybrid approaches combining CNNs with Support Vector Machines (SVM) and Random Forest to improve accuracy and reliability. The goal is to develop a robust, automated system capable of distinguishing between genuine and counterfeit notes with high precision.

1.3 MOTIVATION:

The motivation behind this project stems from the increasing threat of counterfeit currency, which undermines economic stability and financial security. Traditional methods of detecting fake notes are often labor-intensive and prone to errors. By employing advanced Convolutional Neural Networks (CNNs) and combining them with machine learning algorithms like Support Vector Machines (SVM) and Random Forest, this project aims to develop a highly accurate and efficient system for identifying counterfeit Indian currency.

The integration of these cutting-edge technologies promises to enhance detection capabilities, reduce human error, and provide a scalable solution to safeguard against currency fraud.

1.4 SCOPE:

The scope of this project encompasses the development and evaluation of advanced machine learning models for counterfeit detection of Indian currency. It includes the implementation of Convolutional Neural Networks (CNNs) such as MobileNet and , as well as hybrid models integrating CNNs with Support Vector Machines (SVM) and Random Forest. The project focuses on creating a robust system capable of accurately classifying currency images as genuine or counterfeit. It also involves assessing the performance of these models in various real-world scenarios, ensuring scalability, and exploring their integration into existing financial security systems. The approach aims to enhance detection capabilities and reduce counterfeit-related risks.

1.5 PROJECT INTRODUCTION:

Counterfeit currency is a pervasive issue that disrupts financial systems and undermines economic stability. As technology evolves, so too do the methods used by counterfeiters, necessitating increasingly sophisticated detection techniques. Traditional methods of counterfeit detection, often reliant on manual inspection and basic verification tools, are becoming inadequate in the face of advanced counterfeiting techniques. This project seeks to address this challenge by leveraging the power of Convolutional Neural Networks (CNNs), a class of deep learning algorithms known for their efficacy in image analysis.

The introduction of CNN models such as MobileNet and offers a promising approach to enhancing counterfeit detection systems. MobileNet's lightweight architecture is designed for efficient performance on resource-constrained devices, while 's deep residual learning facilitates high accuracy in complex classification tasks. Additionally, hybrid models combining CNNs with Support Vector Machines (SVM) and Random Forest aim to capitalize on the strengths of ensemble learning techniques.

This project will develop a robust, automated system capable of accurately distinguishing between genuine and counterfeit Indian currency. By improving detection accuracy and efficiency, the proposed system has the potential to significantly strengthen financial security measures, providing a crucial tool for safeguarding against currency fraud.

2 LITERATURE SURVEY

2.1 Related work:

1. **A. S. Deshmukh, M. R. Kshirsagar, and A. S. Patel, "Detection of Fake Currency Using Convolutional Neural Networks," International Journal of Computer Applications, vol. 179, no. 12, pp. 6-10, 2018.**

   This study investigates the use of Convolutional Neural Networks (CNNs) for detecting counterfeit currency. It focuses on analyzing images of Indian banknotes to identify counterfeit features. The paper demonstrates how CNNs can be trained on a dataset of genuine and fake currency notes to improve detection accuracy, showing promising results in distinguishing between real and counterfeit Indian notes.

2. **S. D. Patil and M. N. Jadhav, "A Survey on Fake Currency Detection Techniques Using Image Processing," International Journal of Engineering Research and Applications (IJERA), vol. 9, no. 2, pp. 26-30, 2019.**

   This survey provides a comprehensive review of various image processing techniques employed for counterfeit currency detection. It covers traditional methods and modern approaches, including those specifically for Indian currency. The paper discusses the challenges and advancements in the field, providing a foundation for understanding the effectiveness of different detection techniques.

3. **R. Tiwari, A. K. Sinha, and R. R. Sinha, "An Intelligent System for Fake Currency Detection using Machine Learning Techniques," Proceedings of the International Conference on Recent Innovations in Computing, 2020.**

   This research paper presents an intelligent system combining machine learning algorithms for detecting counterfeit currency. It focuses on Indian notes and evaluates the performance of various algorithms, including Support Vector Machines (SVM) and Convolutional Neural Networks (CNNs). The study highlights how these techniques can be integrated to improve detection accuracy and reliability.

4. **P. R. Gupta and A. D. Sharma, "Counterfeit Currency Detection System using Deep Learning Techniques," International Journal of Computer Applications, vol. 179, no. 6, pp. 10- 15, 2018.**

   This paper explores the application of deep learning techniques, particularly CNNs, for counterfeit currency detection. It details how deep learning models are applied to images of Indian banknotes to enhance detection capabilities. The study demonstrates the advantages of deep learning over traditional methods in terms of accuracy and efficiency.

5. **N. R. Desai, S. S. Patil, and R. D. Rao, "Real-Time Fake Currency Detection System using Machine Learning Algorithms," International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), vol. 10, no. 5, pp. 40-45, 2020.**

   This paper discusses the development of a real-time system for counterfeit currency detection using machine learning algorithms. It focuses on Indian currency and includes the implementation of various models, including CNNs and ensemble methods. The research emphasizes the system's ability to process currency images quickly and accurately, making it suitable for practical applications.

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Existing methods for counterfeit currency detection often rely on Convolutional Neural Networks (CNNs) due to their effectiveness in image classification tasks. CNN-based systems analyze currency images to identify features such as textures, patterns, and security marks. Common approaches use models like LeNet, AlexNet, or VGG for feature extraction and classification. These models are trained on large datasets of genuine and counterfeit notes to learn distinguishing characteristics. Despite their effectiveness, these methods may face limitations in handling variations in counterfeit quality and environmental conditions. Enhancing CNN models with advanced architectures and hybrid techniques can improve detection accuracy and reliability.

3.2 DISADVANTAGES OF EXISTING SYSTEM

Existing CNN-based methods for counterfeit currency detection have several disadvantages:

1. **Computational Complexity:** Deep CNN models can be resource-intensive, requiring substantial computational power and memory, which may be challenging for real-time applications.

2. **Sensitivity to Variations:** CNNs might struggle with variations in counterfeit quality, image resolution, and environmental conditions, potentially leading to false positives or negatives.

3. **Data Requirements:** Training effective CNN models requires large datasets of high-quality images, including diverse examples of both genuine and counterfeit notes, which may not always be available.

4. **Overfitting:** Deep models can overfit to training data if not properly regularized, affecting their performance on unseen counterfeit types.

5. **Interpretability:** CNNs often operate as "black boxes," making it difficult to understand and interpret the features and decisions made by the model, which can be a drawback for security applications where transparency is important.

3.3 PROPOSED SYSTEM

The proposed system enhances counterfeit currency detection by integrating advanced Convolutional Neural Networks (CNNs) with hybrid machine learning models. It utilizes MobileNet for their efficient and accurate image classification capabilities, combined with Support Vector Machines (SVM) and Random Forest to leverage ensemble learning techniques. This hybrid approach aims to improve detection precision and robustness by combining the strengths of each model. The system will process currency images to identify subtle counterfeit features, offering a scalable and real-time solution. By addressing the limitations of existing methods, this system seeks to provide a more reliable and efficient tool for safeguarding against counterfeit currency.

3.4 ADVANTAGES OF THE PROPOSED SYSTEM

The proposed system offers several advantages:

1. **Enhanced Accuracy:** By combining CNNs like MobileNet with SVM and Random Forest, the system leverages multiple models' strengths to improve detection accuracy and reduce false positives and negatives.

2. **Efficient Processing:** MobileNet is lightweight architecture ensures efficient performance on resource-constrained devices, making the system suitable for real-time applications.

3. **Robustness:** The hybrid approach enhances robustness against variations in counterfeit quality, image resolution, and environmental conditions, leading to more reliable detection.

4. **Scalability:** The system can be scaled to handle large volumes of currency images, making it suitable for various applications, from banknote validation to large-scale financial institutions.

5. **Flexibility:** The integration of multiple algorithms allows for adaptability to new types of counterfeits and evolving threats, improving the system’s long-term effectiveness.

3.5 PROJECT FLOW

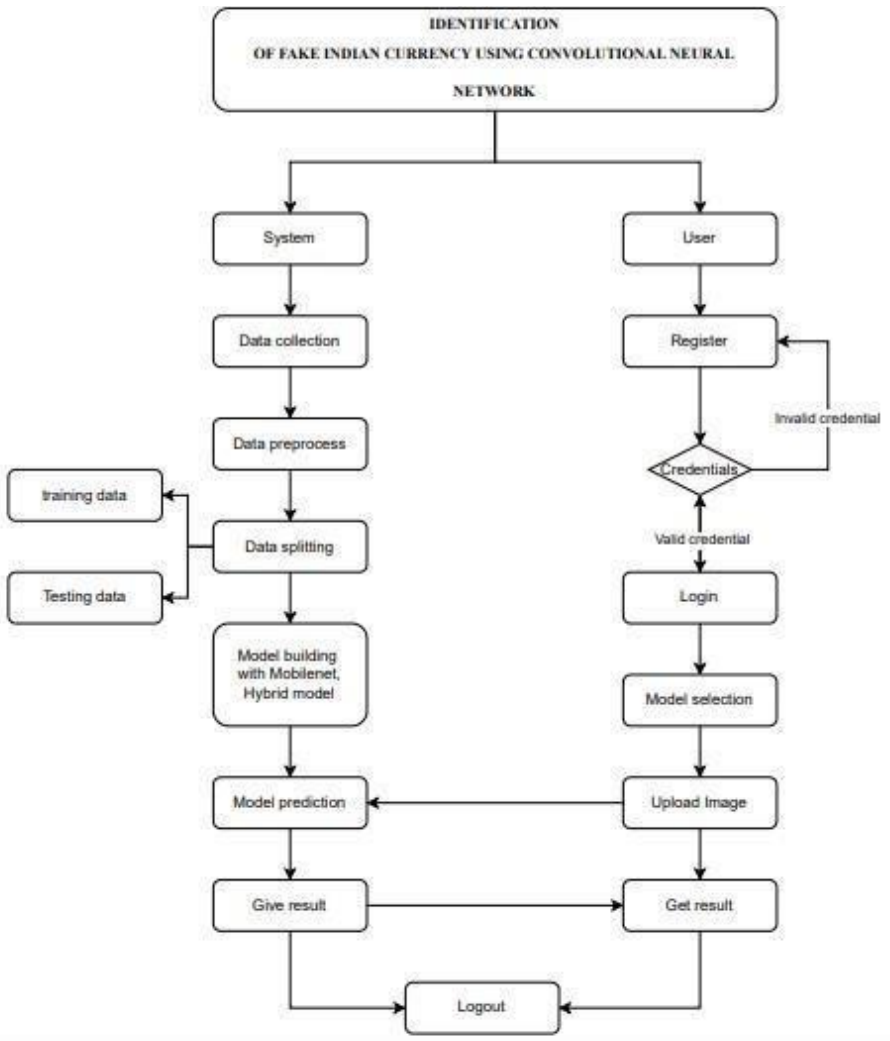


Fig 3.5: Project Flow

4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Requirement's analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and nonfunctional requirements.

Functional Requirements:

Functional requirements are the basic features that a user expects from a system and must be included as per the agreement in the contract. These requirements define what input the system will accept, what processing it will carry out, and what output it will deliver. They are concerned with the specific behavior and functionalities of the system and are visible in the final product. Unlike non-functional requirements, which focus on performance, quality, and user experience, functional requirements describe the core operations of the system. For example, the system must authenticate a user whenever they log in, shut down in case of a cyberattack, and send a verification email to the user when they register for the first time. These requirements directly influence how the system behaves and interacts with users.

Non functional Requirements:

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called nonbehavioral requirements.

4.2 HARDWARE REQUIREMENTS

Processor	-I3/Intel
RAM	-8GB (min)
Hard Disk	-128 GB (min)

4.3 SOFTWARE REQUIREMENS

Operating System	: Windows 7/8/10
Server side Script	: HTML, CSS, Bootstrap & JS
Programming Language	: Python
Libraries	:Flask, Torch, Tensorflow, Pandas, Mysql.connector
IDE/Workbench	: VSCode
Server Deployment	: Xampp Server
Database	: MySQL

4.4 ARCHITECTURE:

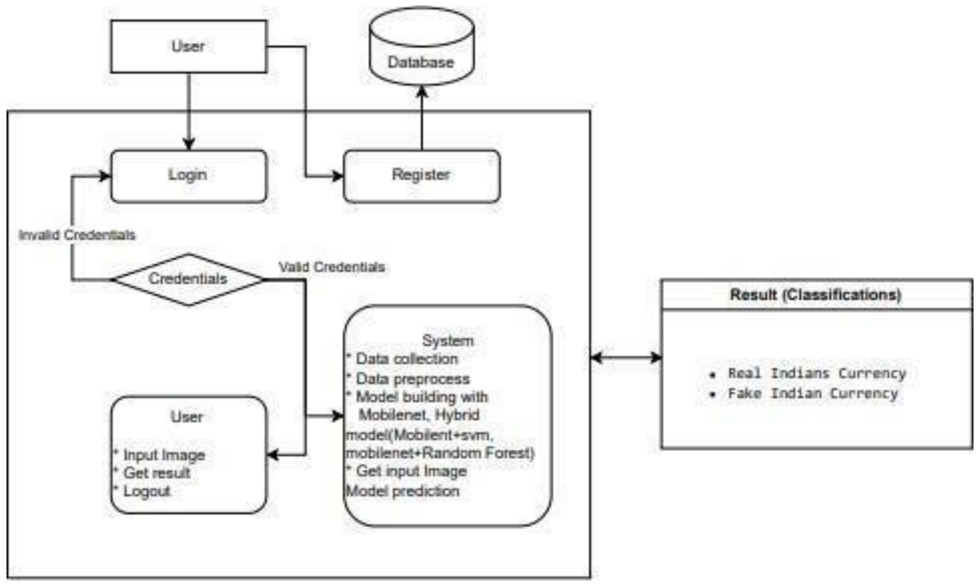


Fig 4.4: Architecture

5.METHODOLOGIES

5.1 MobileNet

- The methodology for employing MobileNet in counterfeit currency detection begins with data collection and preparation. A diverse dataset of Indian currency images, encompassing genuine and counterfeit notes across various denominations and conditions, is gathered. The images are then preprocessed by resizing them to MobileNet's input dimensions (224x224 pixels), normalizing pixel values, and applying data augmentation techniques like rotation and flipping to enhance model generalization.

- Next, MobileNet, known for its lightweight and efficient architecture using depthwise separable convolutions, is selected. A pre-trained MobileNet model, originally trained on the ImageNet dataset, is used to leverage learned features. This model is fine-tuned for the counterfeit detection task by replacing the final classification layer with a new dense layer suitable for binary classification (genuine vs. counterfeit). The weights of earlier layers are frozen to retain pre-learned features, while the new layers are trained with a low learning rate to adapt the model to the specific dataset.

- During training, the dataset is split into training, validation, and test sets. The MobileNet model is trained using the training set, and its performance is validated on the validation set. Metrics such as accuracy, precision, recall, and F1-score are monitored to evaluate the model's performance, with adjustments made to hyperparameters and regularization techniques to avoid overfitting.

- Finally, the trained MobileNet model is integrated into a real-time detection system. It is optimized for inference speed and memory usage to ensure efficient performance on deployment devices. The system is continuously tested with new data to ensure robustness and accuracy in detecting counterfeit currency.

5.2 MobileNet + SVM Hybrid Model

The methodology for using a MobileNet and Support Vector Machine (SVM) hybrid model for counterfeit currency detection begins with data collection and preparation. A diverse dataset of Indian currency images, including both genuine and counterfeit notes, is collected and preprocessed by resizing them to MobileNet's input dimensions (224x224 pixels), normalizing pixel values, and applying data augmentation techniques like rotation and flipping to enhance model robustness.

MobileNet is employed to extract high-level features from these images. Instead of using the final classification layer, the model is adapted to output feature vectors from an intermediate layer. These feature vectors are then used as input to an SVM classifier, which is trained to differentiate between genuine and counterfeit currency. The SVM leverages its ability to define optimal decision boundaries in high-dimensional spaces, improving classification accuracy.

Training involves splitting the dataset into training, validation, and test sets. MobileNet is first trained or fine-tuned to extract meaningful features, while the SVM is trained separately on these features. Performance metrics such as accuracy, precision, recall, and F1-score are monitored during validation to optimize hyperparameters for both models.

The hybrid model, combining MobileNet's feature extraction with SVM's classification, is then integrated into a real-time detection system. This system is optimized for efficient processing and deployed on relevant devices. Continuous testing with new data ensures the model's robustness and accuracy in detecting counterfeit currency, providing a scalable and effective solution.

MobileNet + Random Forest Hybrid Model

The methodology for employing a MobileNet and Random Forest hybrid model for counterfeit currency detection begins with comprehensive data collection and preparation. A diverse dataset of Indian currency images, encompassing both genuine and counterfeit notes, is gathered. The images are preprocessed by resizing them to MobileNet's input dimensions (typically 224x224 pixels), normalizing pixel values, and applying data augmentation techniques such as rotation and flipping to enhance the model's robustness.

MobileNet is used to extract high-level features from these images. Instead of utilizing the final classification layer, MobileNet is configured to output feature vectors from one of its intermediate layers. These feature vectors capture the essential characteristics of the currency images.

These extracted features are then fed into a Random Forest classifier. The Random Forest, an ensemble learning technique based on decision trees, is trained to classify the features into genuine or counterfeit categories. It benefits from aggregating the decisions of multiple decision trees, thereby improving classification accuracy and handling complex, high-dimensional data effectively.

The training process involves splitting the dataset into training, validation, and test sets. MobileNet is first trained or fine-tuned to extract robust features, while the Random Forest classifier is trained on these features. During validation, metrics such as accuracy, precision, recall, and F1-score are monitored to fine-tune hyperparameters for both MobileNet and Random Forest.

Finally, the hybrid model, integrating MobileNet's feature extraction with Random Forest's classification capabilities, is deployed into a real-time detection system. The system is optimized for efficient processing and integrated into relevant devices. Continuous testing with new data ensures the system's reliability and accuracy in identifying counterfeit currency, offering a scalable and effective solution for financial security.

6. SYSTEM DESIGN

6.1 INTRODUCTION OF INPUT DESIGN:

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.
Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties −

- It should serve specific purpose effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding
  - What are the inputs needed for the system?
  - How end users respond to different elements of forms and screens.

Objectives for Input Design:

The objectives of input design are :

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.

OUTPUT DESIGN:

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

Objectives of Output Design:

The objectives of input design are:

- To develop output design that serves the intended purpose and eliminates the production of unwanted output.
- To develop the output design that meets the end user's requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decisions.

6.2 UML Diagrams

6.2.1 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

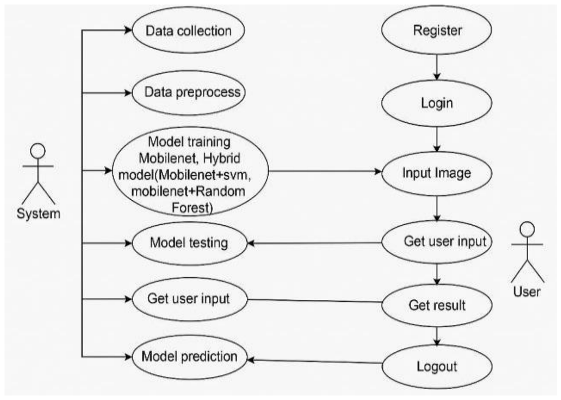


Fig 6.2.1: Use Case Diagram

6.2.2 CLASS DIAGRAM

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

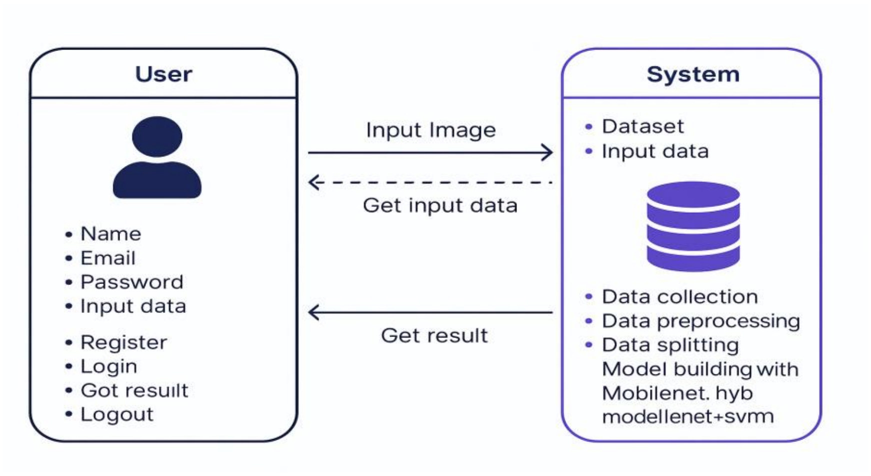


Fig 6.2.2: Class Diagram

6.2.3 SEQUENCE DIAGRAM

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

A **Sequence Diagram** is a type of **UML (Unified Modeling Language)** diagram that shows how objects interact with each other in a particular sequence over time. It's mainly used to visualize the flow of messages between different parts of a system during a specific use case or process.

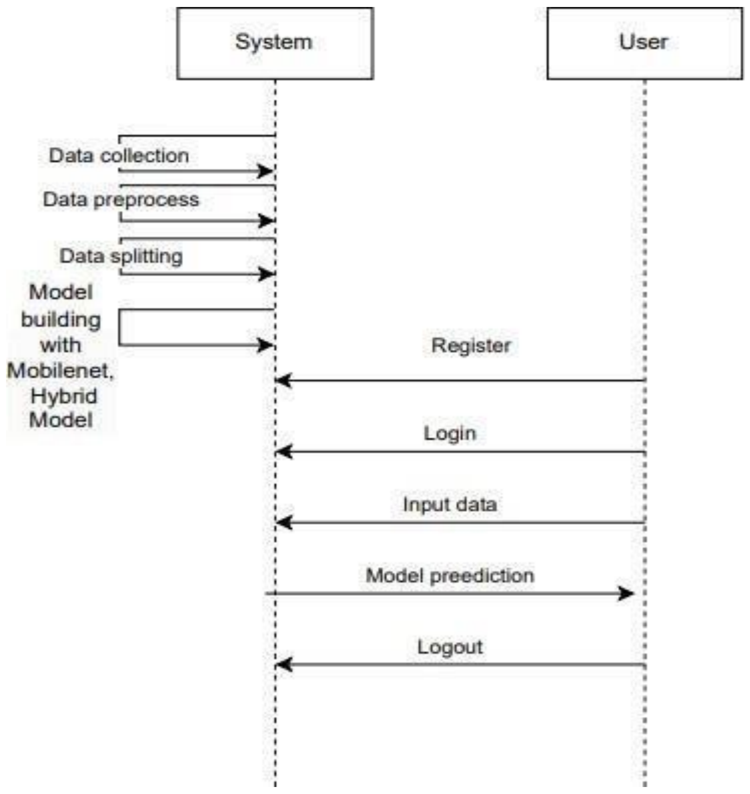


Fig 6.2.3: Sequence Diagram

6.2.4 COLLABORATION DIAGRAM:

In collaboration diagram the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram.

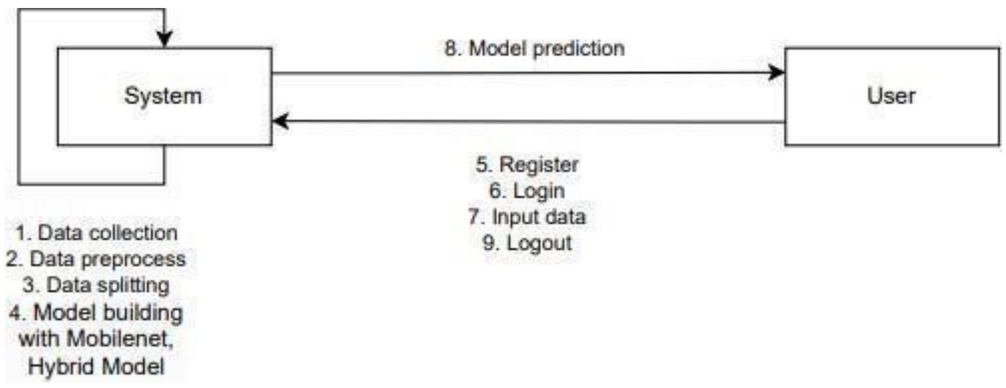


Fig 6.2.4: Collaboration Diagram

6.2.5 DEPLOYMENT DIAGRAM:

Deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware's used to deploy the application.

System
User

Fig 6.2.5: Deployment Diagram

6.2.6 COMPONENT DIAGRAM:

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical **c**omponents in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required functions is covered by

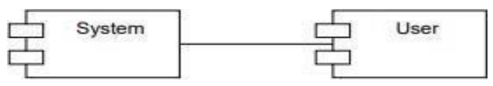


Fig 6.2.6: Component Diagram

6.2.7 ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
An **Activity Diagram** is a **behavioral UML diagram** that visually represents the **flow of control or data** from one activity to another. It models the **dynamic aspects of a system**, making it useful for both software engineers and business analysts.

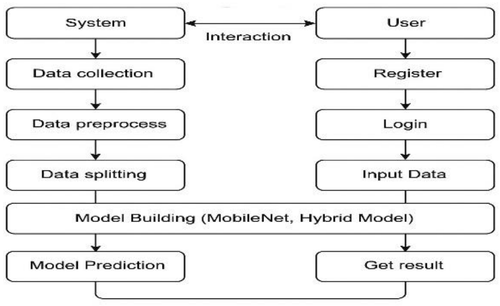


Fig 6.2.7: Activity Diagram

7. IMPLEMENTATION AND RESULTS

7.1 MODULES

1. System:

1.1 Data Collection: This module involves gathering a comprehensive dataset of currency images, including both genuine and counterfeit notes. The dataset is divided into training and testing subsets, typically with an 80% to 20% split.

1.2 Data Preprocessing: This step includes image resizing, normalization, and augmentation to enhance the quality and variability of the dataset. The preprocessed images are then ready for model training and testing.

1.3 Model Training: The CNN models (MobileNet) are trained using 80% of the dataset. This involves fine-tuning model parameters and optimizing performance to accurately classify currency images.

1.4 Hybrid Model Integration: Combining CNNs with Support Vector Machines (SVM) and Random Forest to create a hybrid model. This module focuses on integrating these models and tuning their combined performance.

1.5 Model Testing: The remaining 20% of the dataset is used to evaluate the performance of the trained models. Metrics such as accuracy, precision, and recall are computed to assess model effectiveness.

1.6 Model Saving: Once trained, the models (including the hybrid model) are saved in a format such as .h5 or .pkl to preserve their learned parameters and weights.

1.7 Model Prediction: New currency images are input into the trained models to predict whether they are genuine or counterfeit. This module handles the prediction process and outputs results.

2.User:

1.8 Register: Users register an account in the system with their credentials to gain access.

1.9 Login: Users log in with their registered credentials to access the currency detection features.

1.10 Upload Data: Users can upload currency images for detection. These images are processed by the model for analysis.

1.11 View Results: Users receive and view predictions from the model, which indicates whether the uploaded currency is genuine or counterfeit.

1.12 Logout: Users can log out of the system to ensure their session and personal data are secure.

7.2 OUTPUT SCREENS:

INDEX PAGE: This is the index page of the project website.

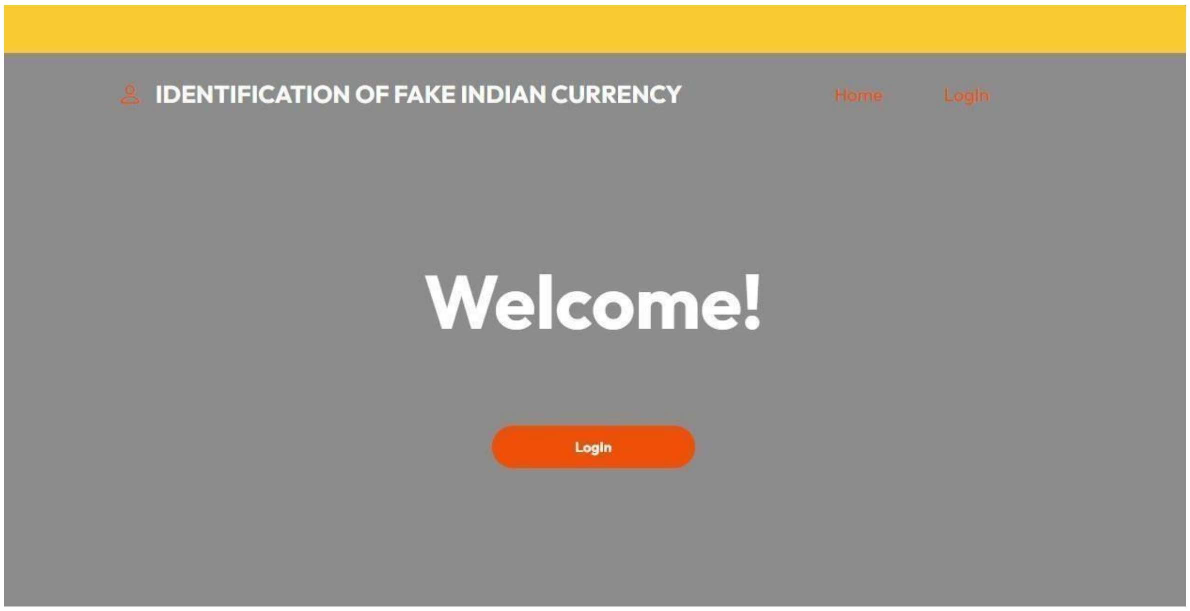


Fig: Output Screens

REGISTRATION PAGE: In this page user can register with their credentials such as name, email,password.

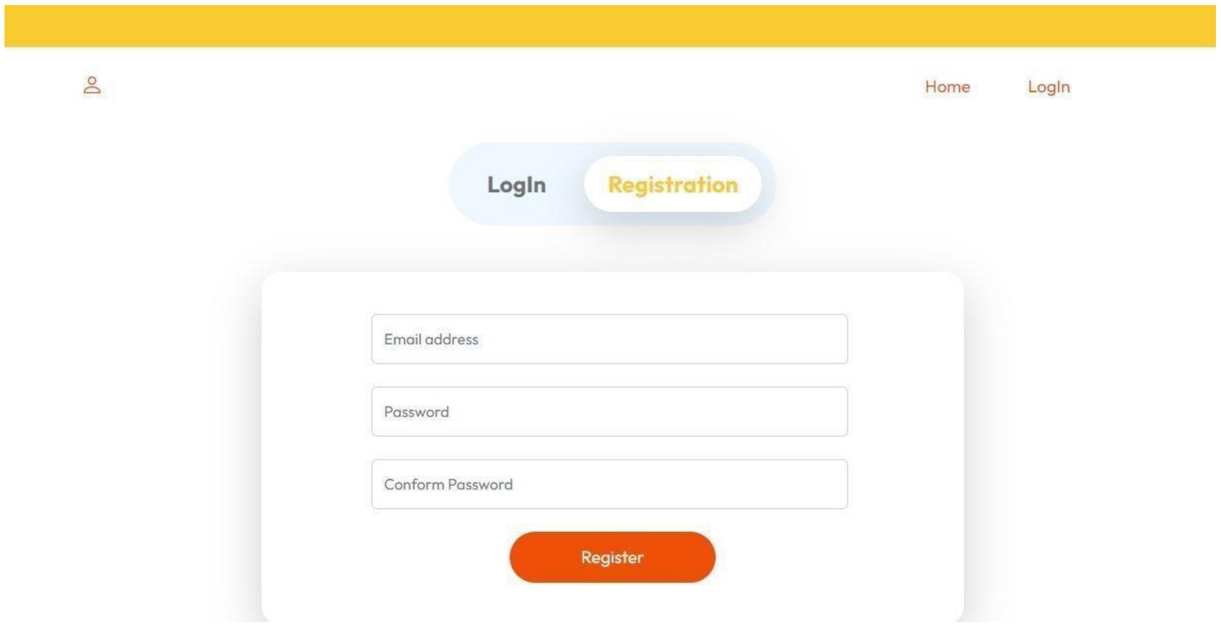


Fig: Login Page

LOGIN PAGE: In this page, user can login with their registered credentials.

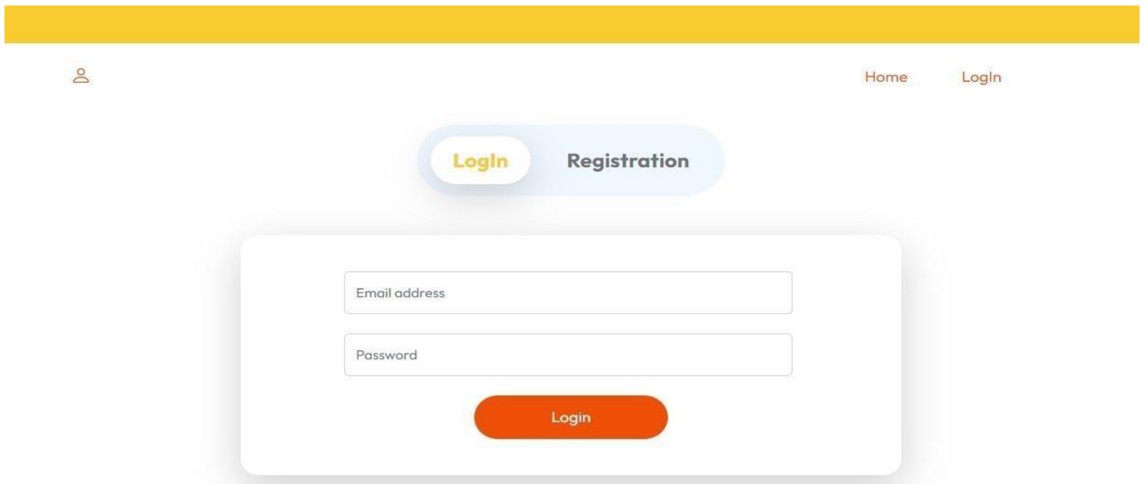
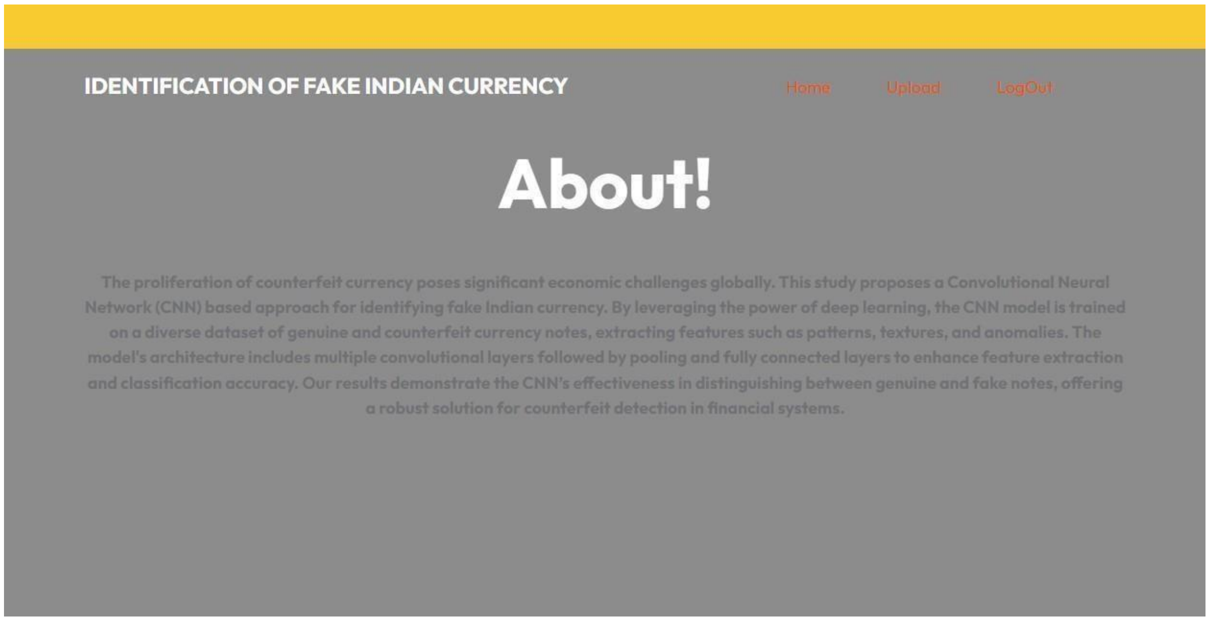


Fig: Home Page

HOME PAGE: After successfully login, this page will be shown.



About!

The proliferation of counterfeit currency poses significant economic challenges globally. This study proposes a Convolutional Neural Network (CNN) based approach for identifying fake Indian currency. By leveraging the power of deep learning, the CNN model is trained on a diverse dataset of genuine and counterfeit currency notes, extracting features such as patterns, textures, and anomalies. The model's architecture includes multiple convolutional layers followed by pooling and fully connected layers to enhance feature extraction and classification accuracy. Our results demonstrate the CNN's effectiveness in distinguishing between genuine and fake notes, offering a robust solution for counterfeit detection in financial systems.

Fig: Prediction Page

PREDICTION PAGE: In here user can upload image and get prediction.

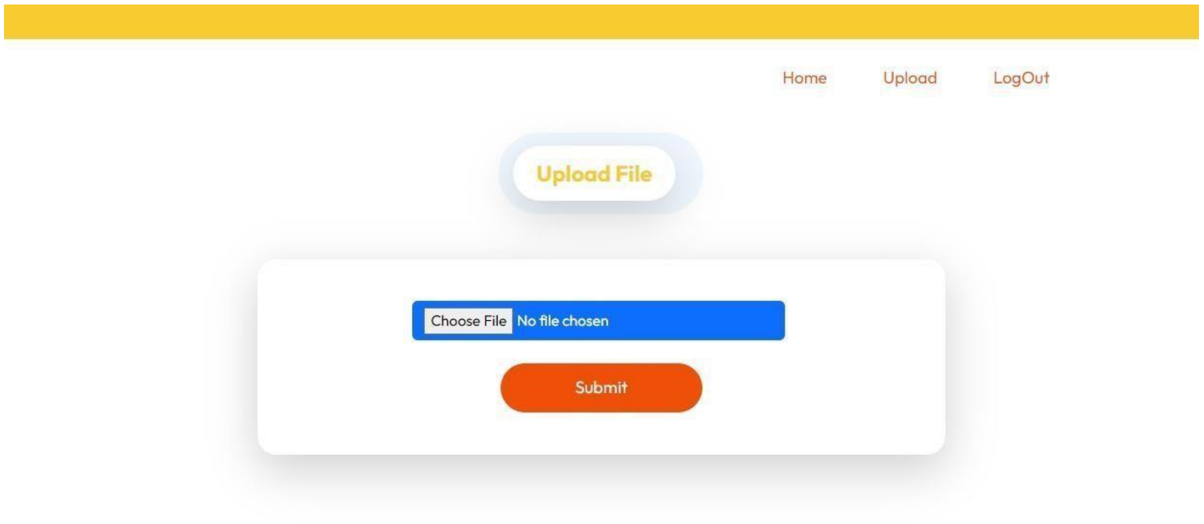


Fig: Result Page

RESULT PAGE: In here, user result will be display.

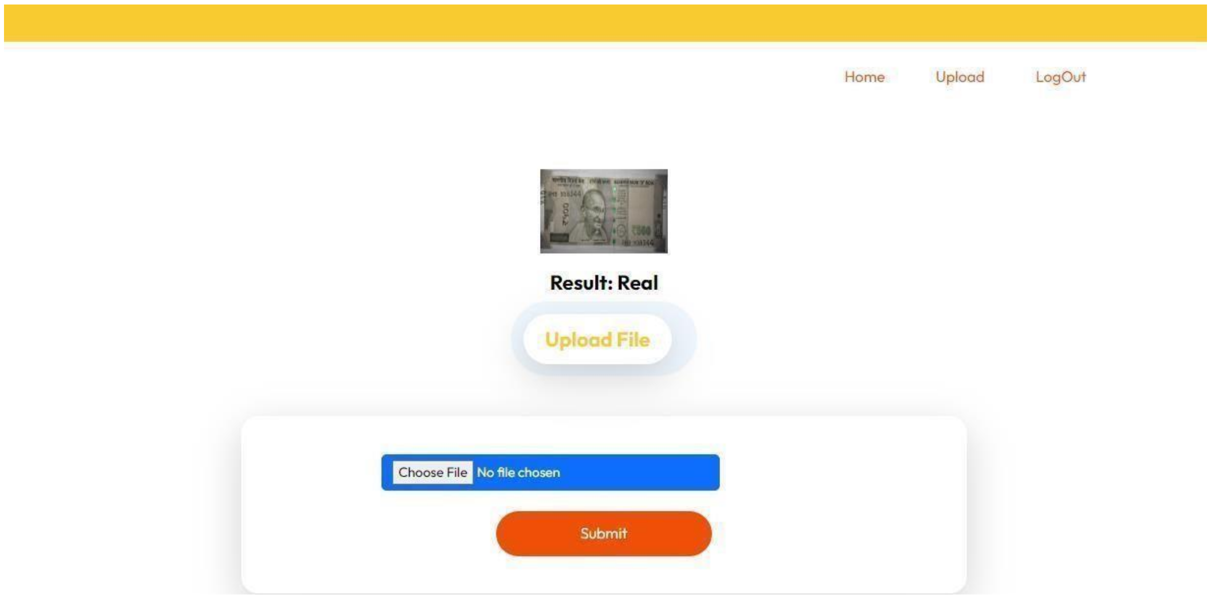


Fig: Prediction – Real

Here, if we upload real Indian currency image, we will get prediction as **"Real".**

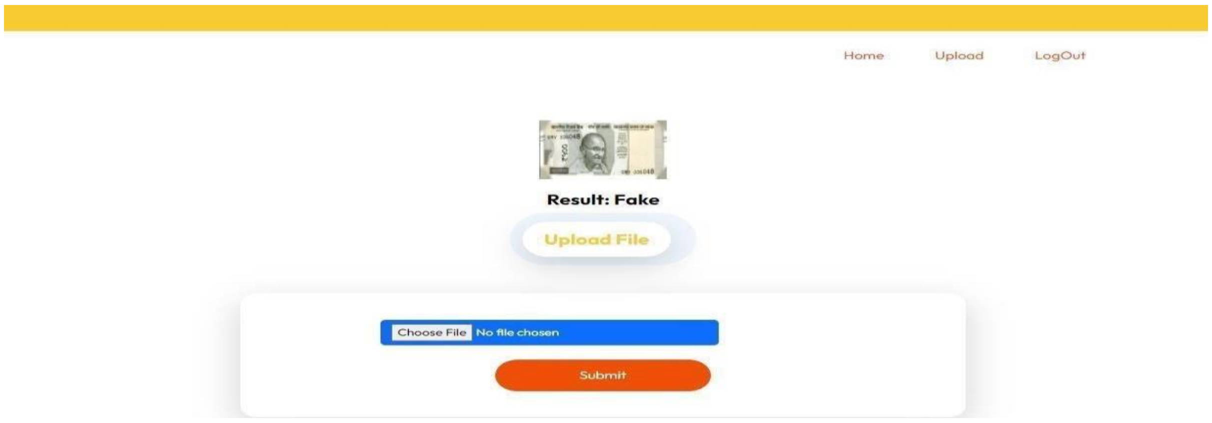
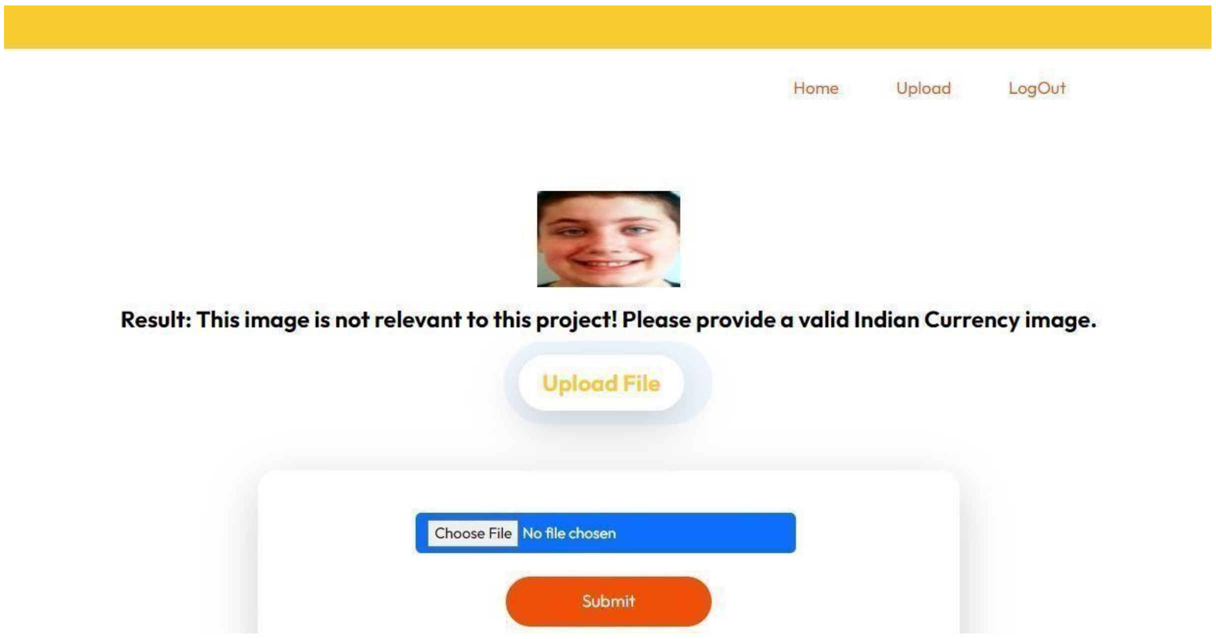


Fig: Prediction – Fake

Here, if we upload fake Indian currency image, we will get prediction as **"Fake".**



Result: This image is not relevant to this project! Please provide a valid Indian Currency image.

Fig: Notification if irrelevant image

Here, if we upload any image other than the Indian currency we will message as **“This image is not relevant to this project! Please provide a valid Indian currency image.”**

8. SYSTEM STUDY AND TESTING

8.1 FEASIBILITY STUDY

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- Economical feasibility
- Technical feasibility
- Social feasibility

Economical Feasibility

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

System Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

8.2 TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration.

Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Functional testing

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manual

Functional testing is centered on the following items:

Valid Input	: identified classes of valid input must be accepted.
Invalid Input	: identified classes of invalid input must be rejected.
Functions	: identified functions must be exercised.
Output	: identified classes of application outputs must be exercised.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, a a black box. you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

TEST CASES

Input	Output	Result
Input features	Tested for different features given by user on the different model.	Success
classification	Tested for different input features given by the user on different features from the models are created using the different algorithms and data.	Success

Test cases Model building

S.NO	Test cases	I/O	Expected O/T	Actual O/T	P/F (It produced P. If this not F will come)
1	Read the datasets.	Dataset's path.	Datasets need to read successfully.	Datasets fetched successfully.	P
2	Verify Image Preprocessing	Sample currency images	Images are preprocessed correctly (resized, normalized)	Preprocessed images match expected format.	P
3	Verify CNN Model Building	Model architecture parameters	Model should be built with the correct architecture	Model built as expected (print architecture summary) .	P

4	Verify Training Process	Training dataset, epochs, batch size	Model should train without errors, and loss should decrease over epochs	Model trains successfully and shows decreasing loss .	P
5	Verify Validation Process	Validation dataset	Model should validate without errors and metrics should be within expected range	Model validates successfully with expected metrics (e.g.,accuracy)	P

9. SAMPLE SOURCE CODE

Importing necessary Libraries :

BACKEND:

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Setup the ImageDataGenerator for data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2)
train_generator = train_datagen.flow_from_directory(
    r'Indian Currency Dataset\train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset="training")
validation_generator = train_datagen.flow_from_directory(
    r'Indian Currency Dataset\train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    subset="validation")
test_datagen = ImageDataGenerator(rescale=1./255)
test_generator = test_datagen.flow_from_directory(
    r'Indian Currency Dataset\test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False)
import tensorflow as tf
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten,
BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Load the pre-trained MobileNet model without the top layer
base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False
# Adding custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(2, activation='softmax')(x)
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
```

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Fit the model
history=model.fit(
    train_generator,
    epochs=10, validation_data=validation_generator)
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
hist_=pd.DataFrame(history.history)
hist_
plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
plt.plot(hist_['loss'],label='Train_Loss')
plt.plot(hist_['val_loss'],label='Validation_Loss')
plt.title('Train_Loss & Validation_Loss',fontsize=20)
plt.legend()
plt.subplot(1,2,2)
plt.plot(hist_['accuracy'],label='Train_Accuracy')
plt.plot(hist_['val_accuracy'],label='Validation_Accuracy')
plt.title('Train_Accuracy & Validation_Accuracy',fontsize=20)
plt.legend()
plt.show()
acc= model.evaluate(test_generator)
print('Test Accuracy =', acc)
predictions = model.predict(test_generator)
y_pred = np.argmax(predictions,axis=1)
y_test_ = test_generator.classes
df = pd.DataFrame({'Actual': y_test_, 'Prediction': y_pred})
df CM = confusion_matrix(y_test_,y_pred)
CM_percent = CM.astype('float') / CM.sum(axis=1)[:, np.newaxis]

sns.heatmap(CM_percent,fmt='g',center = True,cbar=False,annot=True,cmap='Blues')
CM ClassificationReport = classification_report(y_test_,y_pred)
print('Classification Report is : ', ClassificationReport )
model.save("mobilenet.keras")
import cv2
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import img_to_array, load_img
# Load the pre-trained model
model = keras.models.load_model("mobilenet.keras")
# Define image path
image_path = r"Indian Currency Dataset\test\real\test (45).jpg"
# Load and prepare the image
def prepare_image(image_path, target_size):
    img = load_img(image_path, target_size=target_size)  # Load the image
    img_array = img_to_array(img)  # Convert the image to numpy array
    img_array = img_array / 255.0  # Scale the image (if your model requires normalization)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
# Prepare the image
prepared_image = prepare_image(image_path, target_size=(224, 224))
# Predict using the model
predictions = model.predict(prepared_image)
predicted_class = np.argmax(predictions, axis=1)  # Get the class with highest probability
# Output the result
class_labels = ['Fake', 'Real']
predicted_class_label = class_labels[predicted_class[0]]
```

```
print(f"Predicted Class: {predicted_class_label}")
import tensorflow as tf
from tensorflow.keras.applications import ResNet101
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, Flatten,
BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Load the pre-trained MobileNet model without the top layer
base_model = ResNet101(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False
# Adding custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
predictions = Dense(2, activation='softmax')(x)
# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Fit the model
history=model.fit(
    train_generator,
    epochs=10, validation_data=validation_generator)
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
hist_=pd.DataFrame(history.history)
hist_plt.figure(figsize=(15,10))
plt.subplot(1,2,1)
plt.plot(hist_['loss'],label='Train_Loss')
plt.plot(hist_['val_loss'],label='Validation_Loss')
plt.title('Train_Loss & Validation_Loss',fontsize=20)
plt.legend()
plt.subplot(1,2,2)
plt.plot(hist_['accuracy'],label='Train_Accuracy')
plt.plot(hist_['val_accuracy'],label='Validation_Accuracy')
plt.title('Train_Accuracy & Validation_Accuracy',fontsize=20)
plt.legend()
plt.show()
acc= model.evaluate(test_generator)
print('Test Accuracy =', acc)
predictions = model.predict(test_generator)
y_pred = np.argmax(predictions,axis=1)
y_test_ = test_generator.classes
df = pd.DataFrame({'Actual': y_test_, 'Prediction': y_pred})
df CM = confusion_matrix(y_test_,y_pred)
CM_percent = CM.astype('float') / CM.sum(axis=1)[:, np.newaxis]
sns.heatmap(CM_percent,fmt='g',center = True,cbar=False,annot=True,cmap='Blues')
CM ClassificationReport = classification_report(y_test_,y_pred)
print('Classification Report is : ', ClassificationReport )
model.save("resnet.keras")
import cv2
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import img_to_array, load_img
# Load the pre-trained model
model = keras.models.load_model("resnet.keras")
# Define image path
image_path = r"Indian Currency Dataset\test\real\test (39).jpg"
# Load and prepare the image
def prepare_image(image_path, target_size):
    img = load_img(image_path, target_size=target_size)  # Load the image
    img_array = img_to_array(img)  # Convert the image to numpy array
    img_array = img_array / 255.0  # Scale the image (if your model requires normalization)
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
# Prepare the image
prepared_image = prepare_image(image_path, target_size=(224, 224))
# Predict using the model
predictions = model.predict(prepared_image)
predicted_class = np.argmax(predictions, axis=1)  # Get the class with highest probability
# Output the result
class_labels = ['Fake', 'Real']
predicted_class_label = class_labels[predicted_class[0]]
print(f"Predicted Class: {predicted_class_label}")
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D
import numpy as np
# Load the base model
base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False
# Add a global spatial average pooling layer
x = base_model.output
x = GlobalAveragePooling2D()(x)
```

```
model = Model(inputs=base_model.input, outputs=x)
def extract_features(generator, sample_count):
    # Assuming the model's output is 1024-dimensional features
    # Initialize arrays to hold the extracted features and labels.
    # Note: We use None for flexible sizing in the features' first dimension.
    features = np.zeros((0, 1024))  # Adjust 1024 if a different model output size is expected
    labels = np.zeros((0,))
    # Keep track of the number of samples processed
    processed_samples = 0
    for inputs_batch, labels_batch in generator:
        # Ensure not to process more than the sample_count
        if processed_samples < sample_count:
            # Predict features for the current batch of images
            features_batch = model.predict(inputs_batch)
             actual_batch_size = features_batch.shape[0]
            # Update the total number of processed samples
            processed_samples += actual_batch_size
            # Append the predicted features and the true labels to their respective arrays
            features = np.append(features, features_batch, axis=0)
            labels = np.append(labels, np.argmax(labels_batch, axis=1), axis=0)
            # If we've processed enough samples, break from the loop
            if processed_samples >= sample_count:
                break
        else:
            break
    # If we've processed more samples than needed, truncate the arrays
    if processed_samples > sample_count:
        features = features[:sample_count]
        labels = labels[:sample_count]
    return features, labels
```

```
train_features, train_labels = extract_features(train_generator, 200) # Adjust 200 to actual size
validation_features, validation_labels = extract_features(validation_generator, 50) # Adjust 50 to
actual size
model.save("feature_extractor.h5")
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Create and train the Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(train_features, train_labels)
# Predict on the validation set
validation_predictions = rf.predict(validation_features)
# Evaluate the model
print("Validation Accuracy: ", accuracy_score(validation_labels, validation_predictions))
print("Classification Report:\n", classification_report(validation_labels, validation_predictions))
confusion_matrix(validation_labels, validation_predictions)
import seaborn as sns
CM = confusion_matrix(validation_labels, validation_predictions)
CM_percent = CM.astype('float') / CM.sum(axis=1)[:, np.newaxis]
sns.heatmap(CM_percent,fmt='g',center = True,cbar=False,annot=True,cmap='Blues')
CM from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
import joblib
# It's often a good idea to scale features before using SVM
scaler = StandardScaler()
train_features_scaled = scaler.fit_transform(train_features)
validation_features_scaled = scaler.transform(validation_features)
# Train the SVM model
svm_model = SVC(kernel='linear', C=1.0, decision_function_shape='ovo')
svm_model.fit(train_features_scaled, train_labels)
# Predict on the validation set
```

```
validation_predictions = svm_model.predict(validation_features_scaled)
# Evaluate the model
print("Validation Accuracy: ", accuracy_score(validation_labels, validation_predictions))
print("Classification Report:\n", classification_report(validation_labels, validation_predictions))
joblib.dump(scaler, 'scaler.save')
confusion_matrix(validation_labels, validation_predictions)
import seaborn as sns
CM = confusion_matrix(validation_labels, validation_predictions)
CM_percent = CM.astype('float') / CM.sum(axis=1)[:, np.newaxis]
sns.heatmap(CM_percent,fmt='g',center = True,cbar=False,annot=True,cmap='Blues')
CM import joblib
joblib.dump(svm_model, 'svm_model.pkl')
from tensorflow.keras.models import load_model
import joblib
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.applications.mobilenet import preprocess_input
import numpy as np
def load_and_preprocess_image(image_path):
    # Load the image file, resizing it to 224x224 pixels (as expected by MobileNet)
    img = load_img(image_path, target_size=(224, 224))
    # Convert the image to a numpy array
    img_array = img_to_array(img)
    # Expand dimensions to match the shape expected by the pre-trained model: (1, 224, 224, 3)
    img_array_expanded = np.expand_dims(img_array, axis=0)
    # Preprocess the image for the pre-trained model
    return preprocess_input(img_array_expanded)
# Load the feature extraction model
feature_extraction_model_path = 'feature_extractor.h5'
feature_extraction_model = load_model(feature_extraction_model_path)
# Load the scaler
```

```
scaler_path = 'scaler.save'
scaler = joblib.load(scaler_path)
# Load the SVM classifier
svm_classifier_path = 'svm_model.pkl'
svm_classifier = joblib.load(svm_classifier_path)
# Assuming you have a function to load and preprocess images named
load_and_preprocess_image
preprocessed_image = load_and_preprocess_image(r'Indian Currency Dataset\test\real\test
(46).jpg')
# Extract features
features = feature_extraction_model.predict(preprocessed_image)
# Scale features
scaled_features = scaler.transform(features.reshape(1, -1))
# Predict with the SVM model
predicted_class = svm_classifier.predict(scaled_features)
print("Predicted class:", predicted_class)
if predicted_class==0:
   result="Fake"
   print(result)
else:
   result="Real"
   print(result)
```

FRONTEND :

```
from flask import
Flask,render_template,flash,redirect,request,send_from_
directory,url_for, send_file
import mysql.connector, os
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# from tensorflow_docs.vis import embed
from tensorflow import keras
#from imutils import paths
# import cv2
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.preprocessing.image import img_to_array,
load_img
from flask import
Flask,render_template,session,flash,redirect,request,send
_from_directory,url_for
import mysql.connector, os
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from datetime import datetime
import time
# Load the pre-trained model
model = keras.models.load_model("mobilenet.keras")
# Load and prepare the image
def prepare_image(image_path, target_size):
    img = load_img(image_path, target_size=target_size)
# Load the image
    img_array = img_to_array(img)  # Convert the image
to numpy array
```

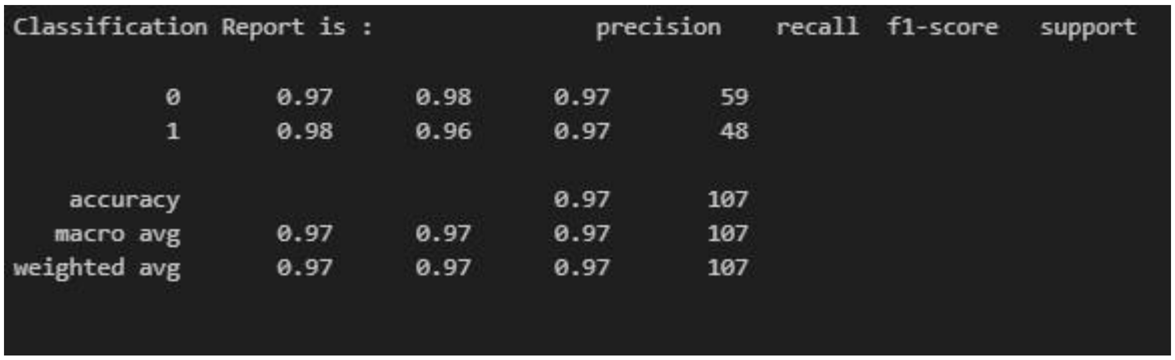
```
    img_array = img_array / 255.0  # Scale the image (if
your model requires normalization)
    img_array = np.expand_dims(img_array, axis=0)  #
Add batch dimension
    return img_array
app = Flask(_name_)
app.secret_key = 'your_secret_key_here'
mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="",
    port="3306",
    database='fake_indian_currency'
)
mycursor = mydb.cursor()
def executionquery(query,values):
    mycursor.execute(query,values)
    mydb.commit()
    return
def retrivequery1(query,values):
    mycursor.execute(query,values)
    data = mycursor.fetchall()
    return data
def retrivequery2(query):
    mycursor.execute(query)
    data = mycursor.fetchall()
    return data
@app.route('/')
def index():
    return render_template('index.html')
```

```
@app.route('/register', methods=["GET", "POST"])
def register():
    if request.method == "POST":
        email = request.form['email']
        password = request.form['password']
        c_password = request.form['c_password']
        if password == c_password:
            query = "SELECT UPPER(email) FROM users"
            email_data = retrivequery2(query)
            email_data_list = []
            for i in email_data:
                email_data_list.append(i[0])
            if email.upper() not in email_data_list:
                query = "INSERT INTO users (email,
password) VALUES (%s, %s)"
                values = (email, password)
                executionquery(query, values)
                return render_template('login.html',
message="Successfully Registered! Please go to login
section")
            return render_template('login.html',
message="This email ID is already exists!")
        return render_template('login.html',
message="Conform password is not match!")

    return render_template('login.html')
@app.route('/login', methods=["GET", "POST"])
def login():
    if request.method == "POST":
        email = request.form['email']
        password = request.form['password']
        query = "SELECT UPPER(email) FROM users"
        email_data = retrivequery2(query)
        email_data_list = []
        for i in email_data:
            email_data_list.append(i[0])
        if email.upper() in email_data_list:
            query = "SELECT UPPER(password) FROM
users WHERE email = %s"
            values = (email,)
            password__data = retrivequery1(query, values)
            if password.upper() == password__data[0][0]:
                global user_email
                user_email = email
                return redirect("/home")
            return render_template('home.html', message=
"Invalid Password!!")
        return render_template('login.html', message= "This
email ID does not exist!")
    return render_template('login.html')
@app.route('/home')
def home():
    return render_template('home.html')
@app.route("/upload", methods=["POST", "GET"])
def upload():
    if request.method == 'POST':
        myfile = request.files['file']
        fn = myfile.filename
        file_extension = fn.split('.')[-1].lower()
        accepted_formats = ['jpg', 'png', 'jpeg', 'jfif']
        if file_extension not in accepted_formats:
            flash("Image formats only Accepted", "danger")
            return render_template("upload.html")
        mypath = os.path.join('static/uploaded_images', fn)
        myfile.save(mypath)
        rev = relevent_finder(mypath)
        if rev != "relevent":
            prediction = "This image is not relevant to this
project! Please provide a valid Indian Currency image."
            return render_template("upload.html", mypath =
mypath, prediction=prediction)
        new_model = load_model(r"mobilenet.keras")
        test_image = image.load_img(mypath,
target_size=(224, 224))
        test_image = image.img_to_array(test_image) / 255
        test_image = np.expand_dims(test_image, axis=0)
        result = new_model.predict(test_image)
        classes = ['Fake', 'Real']
        prediction = classes[np.argmax(result)]
        print(prediction)
        return render_template("upload.html", mypath =
mypath, prediction=prediction)
    return render_template('upload.html')
def relevent_finder(img):
    import torch
    from torchvision import transforms
    from PIL import Image
    import torch.nn as nn
    from torchvision import models
    import os
    # Device configuration
    device = torch.device("cuda" if
torch.cuda.is_available() else "cpu")
    # Image transformations
    image_transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
    ])
    # Define the model class (same as the one used during
training)
    class MobileNetModel(nn.Module):
        def _init_(self, num_classes):
            super(MobileNetModel, self)._init_()
            self.mobilenet =
models.mobilenet_v2(pretrained=True)
            num_features =
self.mobilenet.classifier[1].in_features
            self.mobilenet.classifier[1] =
nn.Linear(num_features, num_classes)
        def forward(self, x):
            return self.mobilenet(x)
    # Load the trained model
    model = MobileNetModel(num_classes=2)
    model.load_state_dict(torch.load(r"mobilenet.pt"))
    model = model.to(device)
    model.eval()
    def predict_image(image_path):
        # Load and preprocess the image
        image = Image.open(image_path).convert('RGB')
        image = image_transform(image).unsqueeze(0)  #
Add batch dimension
        image = image.to(device)


        # Perform the prediction
        with torch.no_grad():
            output = model(image)
            _, predicted = torch.max(output, 1)
        return predicted.item()
    # Helper function to map the prediction to label
    def map_prediction_to_label(prediction):
        label_mapping = {0: "relevent", 1: "irrelevent"}
        return label_mapping.get(prediction, "Unknown")
    # Example usage
    image_path = img
    prediction = predict_image(image_path)
    predicted_label = map_prediction_to_label(prediction)
    print(5555555555555555, predicted_label)
    return predicted_label
if _name_ == '_main_':
    app.run(debug = True)
```

10.RESULT



```
Classification Report is :               precision    recall  f1-score   support

           0       0.97      0.98      0.97        59
           1       0.98      0.96      0.97        48

    accuracy                           0.97       107
   macro avg       0.97      0.97      0.97       107
weighted avg       0.97      0.97      0.97       107
```

Fig 10.1: Classification Report for Mobile Net

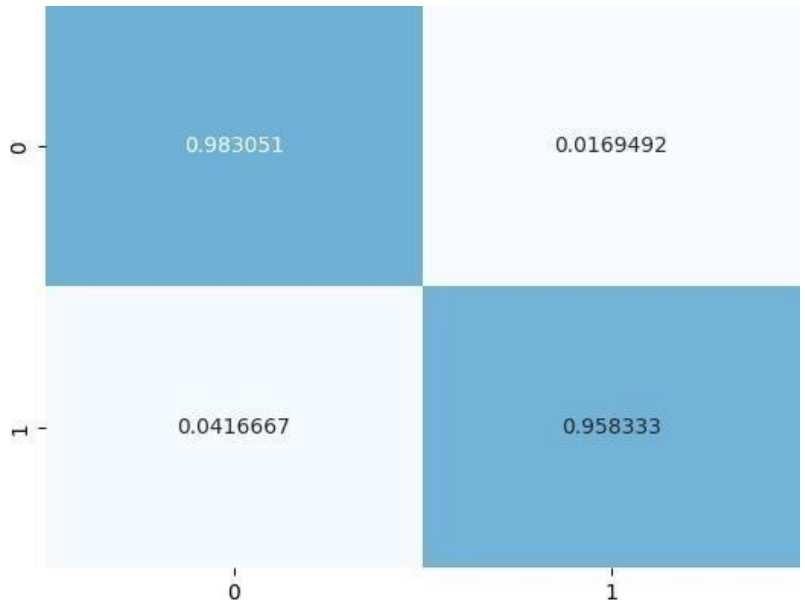
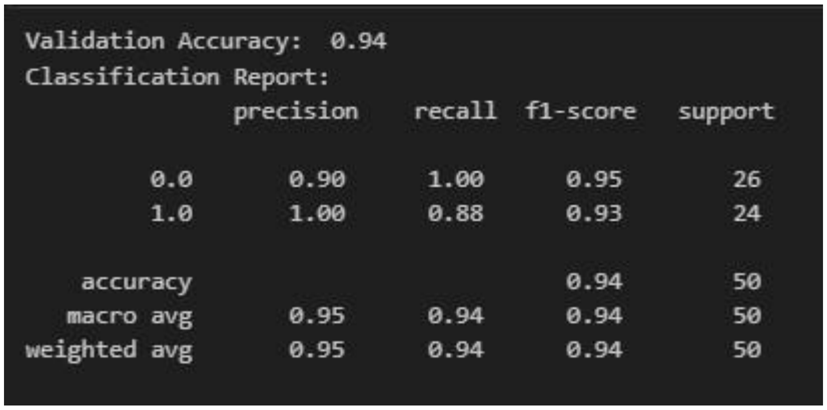


Fig 10.2: Confusion Matrix for Mobile Net



```
Validation Accuracy:  0.94
Classification Report:
              precision    recall  f1-score   support

         0.0       0.90      1.00      0.95        26
         1.0       1.00      0.88      0.93        24

    accuracy                           0.94        50
   macro avg       0.95      0.94      0.94        50
weighted avg       0.95      0.94      0.94        50
```

Fig 10.3: Classification Report for Hybrid Model (Mobilenet+Random Forest)

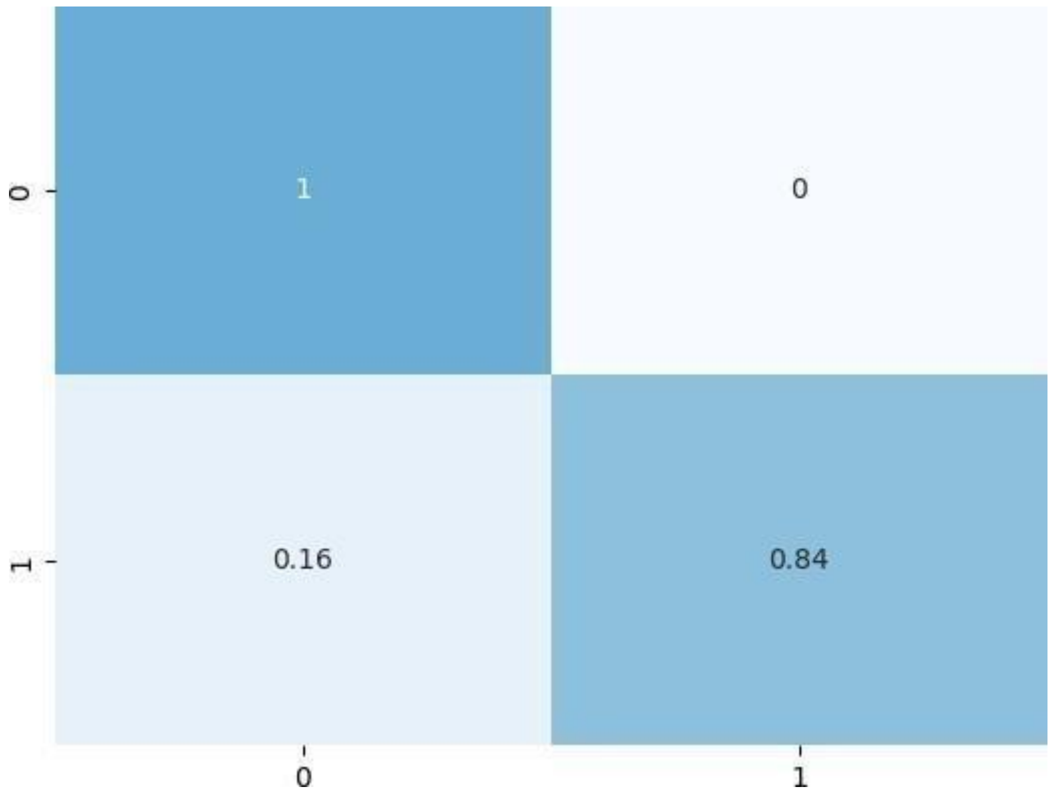
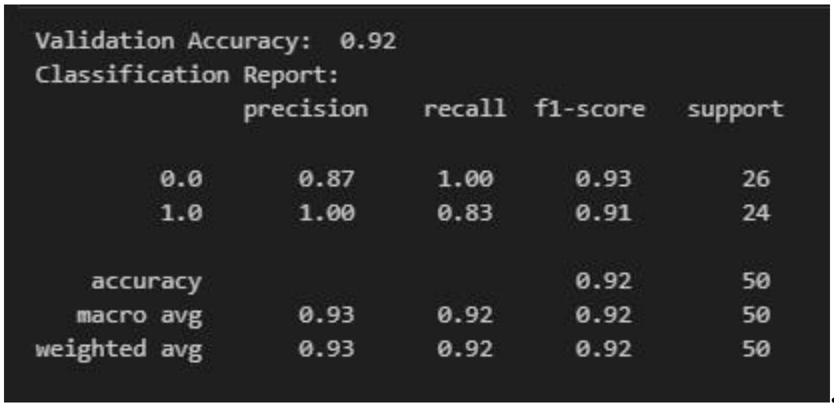


Fig 10.4: Confusion Matrix for Hybrid Model (Mobilenet+Random Forest)

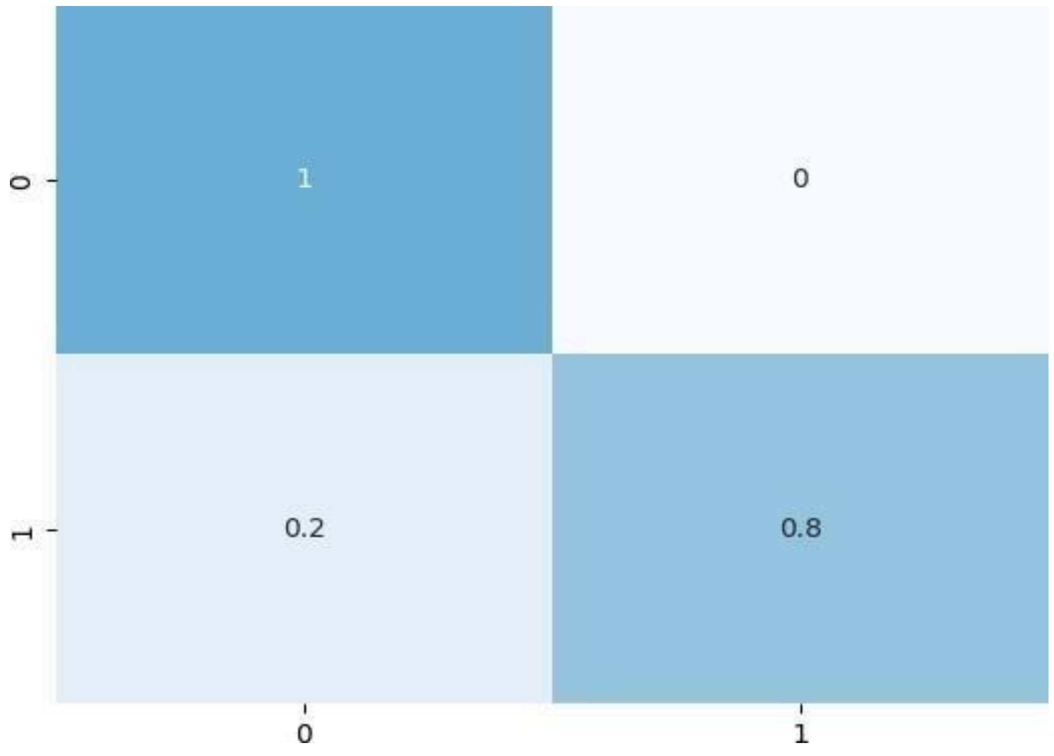


```
Validation Accuracy:  0.92
Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      1.00      0.93        26
         1.0       1.00      0.83      0.91        24

    accuracy                           0.92        50
   macro avg       0.93      0.92      0.92        50
weighted avg       0.93      0.92      0.92        50
```

Fig 10.5: Classification Report for Hybrid Model (Mobilenet+ SVM)



$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 1 & 0 \\ 1 & 0.2 & 0.8 \end{array}$$

Fig 10.6: Confusion Matrix for Hybrid Model (Mobilenet+SVM)

11. CONCLUSION

This project shows that the **MobileNet model** is highly effective in detecting fake Indian currency, achieving a strong **accuracy of 97%**. It performed very well for both real and fake notes, with **precision and recall scores close to 1**, which means it made very few mistakes. The **confusion matrix** also proved this, showing only a few wrong predictions.

On the other hand, a **hybrid model** that combined MobileNet with a **Random Forest classifier** had a slightly lower accuracy of **94%**. Although it was very good at identifying fake notes (perfect precision), its **recall was lower (0.80)**, meaning it missed more fake notes compared to the MobileNet model alone.

In conclusion, the **MobileNet model is more reliable and accurate** for detecting fake currency and is a better choice for real-time applications. In the future, more research can be done to improve these models or try new combinations to get even better results.

12. FUTURE ENHANCEMENT

To make fake Indian currency detection using Convolutional Neural Networks (CNNs) even better, we can plan a few improvements. First, we can use more advanced models like EfficientNet, which can understand images more deeply and accurately.

Using **transfer learning**—where we take models trained on large image datasets and adjust them for currency detection—can help, especially when we don't have many training images.

Another good idea is to use **ensemble learning**, where we combine different models to make better predictions. For example, we can mix CNNs with other methods like **support vector machines (SVM)** or **gradient boosting** to get stronger results.

Improving the dataset is also very important. We can add more examples of fake notes with different lighting, angles, and background conditions. Using **data augmentation**—like rotating, zooming, or shifting images—can help the model learn better and become more flexible.

13. BIBLIOGRAPHY:

REFERENCES:

1. **Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). MobileNets:** Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861.

2. **Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet:** An Extremely Efficient Convolutional Neural Network for Mobile Devices. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 6848-6856).

3. **He, K., Zhang, X., Ren, S., & Sun, J. (2016):** Deep Residual Learning for Image Recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

4. **Tan, M., & Le, Q. V. (2019). EfficientNet:** Rethinking Model Scaling for Convolutional Neural Networks. In International Conference on Machine Learning (pp. 6105-6114). PMLR.

5. **Khoda, B. S., Islam, M. T., & Islam, S. (2020):** A Hybrid Approach of CNN and Random Forest for Detecting Image-based Fake Currency. Journal of Electrical Engineering, Electronics, and Computer Science, 29(1), 45-52.

6. **Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017):** Inception-v4, Inception- and the Impact of Residual Connections on Learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 31, No. 1).

7. **Chollet, F. (2017). Xception:** Deep Learning with Depthwise Separable Convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1251-1258).

8. **Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012):** Imagenet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems, 25.

9. **Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD:** Single Shot MultiBox Detector. In European Conference on Computer Vision (pp. 21-37). Springer, Cham.

10. **Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014):** Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 580-587).

11. **Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN:** Towards Real-Time Object Detection with Region Proposal Networks. In Advances in Neural Information Processing Systems (pp. 91-99).

12. **Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2**: Inverted Residuals and Linear Bottlenecks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4510-4520).

13. **LeCun, Y., Bengio, Y., & Hinton, G. (2015):** Deep Learning. Nature, 521(7553), 436-89.