TABLE OF CONTENTS

ACKNOWLEDGEMENT

With great gratitude I would like to thank Shri. M. K. RAJAGOPALAN, Chairman, Rajiv Gandhi College of Engineering and Technology, for producing highly efficient and potentially resourceful student community to meet their challenges and exploit opportunities in the new millennium.

My sincere thanks are due to Dr. E. VIJAYAKRISHNA RAPAKA, Principal, Rajiv Gandhi College of Engineering and Technology, for his invaluable and generously given support and encouragement to me throughout my course.

I wish to express my profound thanks to Shri. V. BHASKARAN, Administrator, Rajiv Gandhi College of Engineering and Technology, for his kind support and advice he had offered throughout my project work.

I mention my heartiest thanks to the respected Head of the Department of Computer Applications Mr. R. SENTHIL KUMAR, M.C.A., M.E., (Ph.D)., Assistant Professor(Sl.Gr) & HOD, Rajiv Gandhi College of Engineering and Technology . I am indeed highly grateful and immensely indebted for all his guidance in updating my progress and fine-tuning of this project.

I extremely grateful to my internal guide Mrs. V. VENKATALAKSHMI M.C.A., M.Phil., M.Tech., Assistant Professor, Department of Computer Application, Rajiv Gandhi College of Engineering and Technology who has been a constant source of inspiration and encouragement throughout my course.

I also wish to place on record my thanks to all my teaching and non- teaching staffs in the Department of Computer Applications, Rajiv Gandhi College of Engineering and Technology for the keen interest and guidance shown by them throughout the project.

I Sincerely thank Ms. P. Sneha , Project Technical Lead ,The Mind IT for allowing me to do my project work.

SENTHAMIL NATHAN.T

23800023

Securing ATM Transactions with Facial Recognition-Based Verification Systems

Abstract

Automated Teller Machines (ATMs) have become an integral part of daily financial transactions worldwide. However, the growing reliance on ATMs has also led to an increase in security vulnerabilities, with debit card fraud being one of the most reported forms of identity theft, accounting for 270,000 cases in 2021. To enhance the security and user experience of ATM transactions, advancements in computer vision and biometric identification techniques such as fingerprinting, retina scanning, and facial recognition offer promising solutions. This project addresses the security challenges in ATM systems by proposing a secure ATM model that leverages electronic facial recognition using advanced deep learning techniques. The proposed system integrates a Convolutional Neural Network (CNN) to train a FaceNet Model during the account holder's account creation process in the bank. For ATM transactions, a Temporal Convolutional Network (TCN) is utilized to authenticate the account holder by comparing the face captured by the ATM camera with the pre-trained FaceNet model. This dual approach ensures robust real-time verification while maintaining high accuracy and efficiency. In cases of unauthorized access attempts, the system generates a verification request, including a facial recognition link sent to the account holder for immediate identity verification via artificial intelligence agents. This innovative approach eliminates the risks of fraud caused by ATM card theft and duplication, ensuring that only the legitimate account holder can access their account, thus significantly enhancing ATM security and account safety.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATION

S.NO	ABBREVIATION	EXPANSION
1	ATM	Automated Teller Machines
2	PIN	Personal Identification Numbers
3	CNN	Convolutional Neural Networks
4	NFC	Near Field Communication
5	OTP	One-Time Passwords

CHAPTER 1
INTRODUCTION

1.1. OVERVIEW

Automated Teller Machines, popularly referred to as ATMs, are one of the most useful advancements in the banking sector. ATMs allow banking customers to avail quick self-serviced transactions, such as cash withdrawal, deposit, and fund transfers. ATMs enable individuals to make banking transactions without the help of an actual teller. Also, customers can avail banking services without having to visit a bank branch. Most ATM transactions can be availed with the use of a debit or credit card. There are some transactions that need no debit or credit card.

Fig.1.1.1. ATM

History

In 1960, an American named Luther George Simjian invented the Bank graph, a machine that allowed customers to deposit cash and checks into it. The first ATM was set up in June 1967 on a street in Enfield, London at a branch of Barclays bank. A British inventor named John Shepherd-Barron is credited with its invention. The machine allowed customers to withdraw a maximum of GBP10 at a time.

Types of Automated Teller Machines (ATMs)

Automated Teller Machines (ATMs) are mainly of two types. One is a simple basic unit that allows you to withdraw cash, check balance, change the PIN, get mini statements and receive account updates. The more complex units provide facilities of cash or cheque deposits and line of credit & bill payments. There are also onsite and offsite Automated Teller Machines: the onsite ATMs are within the bank premises, unlike the offsite ones which are present in different nooks and corners of the country to assure that people have basic banking facilities

and instant cash withdrawals if they can't go to a bank branch. ATMs can also be categorized based on the labels assigned to them. Some of these labels are listed below-

- Green Label ATMs- Used for agricultural purposes
- Yellow Label ATMs- Used for e-commerce transactions
- Orange Label ATMs- Used for share transactions
- Pink Label ATMs- Specifically for females to help avoid the long queues and waiting time
- White Label ATMs – Introduced by the TATA group, white label ATMs are not owned by a particular bank but entities other than the bank
- Brown Label Banks- Operated by a third party other than a bank

Uses of an Automated Teller Machine

Automated Teller Machines have revolutionized the banking sector by providing easy access to customers and loading off the burden from bank officials. Some of the uses of an ATM are-

- The most common uses of an Automated Teller Machine include withdrawing money, checking balance, transferring money, or changing the PIN (Personal Identification Number)
- Newer and advanced ATMs also provide options to open/withdraw a Fixed Deposit (FD), or to apply for a personal loan. You can also book railway tickets, pay the insurance premiums, income tax & utility bills, recharge mobile, and deposit cash. Some of these facilities require you to register at the bank branch
- Customers can now do money transactions at their convenience. ATMs today are installed in public spaces, highways, malls, market places, railway/airport stations, hospitals, etc.
- Automated Teller Machines provide 24×7 access anywhere
- ATMs help to avoid the hassle of standing in long queues at the bank even for simpler transactions like withdrawing money. It has also helped in reducing the workload of the bank officials.

ATM Fraud

Over the last two decades, automated teller machines (ATMs) have become as much a part of the landscape as the phone booths made famous by Superman. As a result of their ubiquity, people casually use these virtual cash dispensers without a second thought. The notion that something could go wrong never crosses their minds.

Most ATM scams involve criminal theft of debit card numbers and personal identification numbers (PINs) from the innocent users of these machines. There are several variations of this confidence scheme, but all involve the unknowing cooperation of the cardholders themselves. ATM fraud is described as a fraudulent activity where the criminal uses the ATM card of another person to withdraw money instantly from that account. This is done by using the PIN. The other type of ATM fraud is stealing from the machine in the ATM by breaking in.

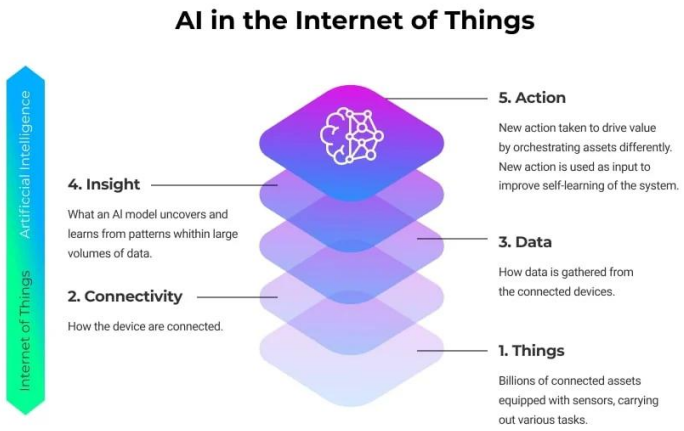
- **Skimming:** This type of ATM scam involves a skimmer device that criminals place on top of or within the card slot. To record your PIN number, the criminals may use a hidden camera or an overlay that covers the original PIN pad. Using the card numbers and PIN's they record; thieves create duplicate cards to withdraw money from consumers' accounts. Unlike losing your debit card or having it stolen, you won't realize anything is amiss until unauthorized transactions take place. Take a look at these so you know how to detect ATM skimmers.
- **Shimming:** This is the latest update to skimming. Instead of reading your card number, criminals place a shimming device deep inside the ATM to record your card's chip information. The end result is the same as skimming because thieves use the stolen chip data to create "cloned" versions of your debit card.
- **Cash-out:** This scam targets multiple accounts from the same financial institution. Armed with a hacked bank employee's credentials, the criminal alters account balances and withdrawal limits. Using stolen debit card numbers captured from a separate skimming attack, they can "cash out" the ATM until it's out of money.
- **Jackpotting:** While there are multiple types of jackpotting attacks, typically, these incidents involve gaining physical access to the inside of the machine. The criminals may replace hardware or install malicious software giving them control of the cash dispensing function. Jackpotting is similar to a cash out scam, but it does not require the criminal to have any customer account details or stolen debit card information.

1.2. PROBLEMS STATEMENT

Automated teller machines (ATMs) face various security challenges that threaten the integrity of financial transactions and jeopardize customer trust. One prevalent issue is card skimming, where criminals install skimming devices to illicitly capture card information, leading to unauthorized access and identity theft. Additionally, PIN theft remains a significant concern, with attackers resorting to methods like shoulder surfing or installing fake keypads to compromise personal identification numbers. Physical attacks, including vandalism and explosive assaults, pose another risk, potentially resulting in damage to ATMs and theft of cash. Malware and skimming software further exacerbate security vulnerabilities by targeting ATMs to harvest sensitive data. Card trapping schemes and phishing attacks also exploit unsuspecting ATM users, putting their financial assets at risk. The reliance on weak authentication methods and the lack of encryption in data transmission further compound the security challenges. Addressing these challenges requires a comprehensive approach that combines technological innovations, regulatory measures, user education initiatives, and collaborative efforts between financial institutions, ATM manufacturers, and law enforcement agencies. The goal is to enhance the security, integrity, and trustworthiness of ATM transactions, thereby safeguarding users' financial assets and personal information from unauthorized access and fraudulent activities

1.3. AI WITH IOT

Individually, the Internet of Things (IoT) and Artificial Intelligence (AI) are powerful technologies. When you combine AI and IoT, you get AIoT—the artificial intelligence of things. You can think of internet of things devices as the digital nervous system while artificial intelligence is the brain of a system. To fully understand AIoT, you must start with the internet of things. When “things” such as wearable devices, refrigerators, digital assistants, sensors and other equipment are connected to the internet, can be recognized by other devices and collect and process data, you have the internet of things.



1.2. Internet of Things

Artificial intelligence is when a system can complete a set of tasks or learn from data in a way that seems intelligent. Therefore, when artificial intelligence is added to the internet of things it means that those devices can analyze data and make decisions and act on that data without involvement by humans. These are "smart" devices, and they help drive efficiency and effectiveness. The intelligence of A IoT enables data analytics that is then used to optimize a system and generate higher performance and business insights and create data that helps to make better decisions and that the system can learn from.

Real-World Examples of AI Embedded IoT Devices

- **Traffic Management**

Traffic is a real problem in urban areas and there is a consistent need for efficient traffic management to avoid congestion. Traffic management can be difficult if it has to be done by humans as it would only lead to chaos and confusion. AIoT, however, is a smart solution to this problem.

Real-time traffic can now be managed efficiently using drones that can monitor large areas and transmit the traffic data which can then be analyzed using AI for final decision making like adjusting traffic lights without human intervention.

- **Self- Driving Cars**

Self-driving cars are another use case of IoT devices embedded with AI. Tesla's self-driving cars are the best example. With the help of installed sensors and the power of AI, this car has the ability to make human-like decisions by determining the conditions of the surroundings. For example, they can determine the optimal speed, weather and road conditions to make effective decisions.

- **Smart Homes**

IoT blended with AI has also led to the emergence of smart homes concept. Smart homes have all the devices connected to each other with the help of IoT and these devices also possess the ability to make smart decisions with the help of AI. Smart homes tend to make our lives easier by giving us the power to control our devices even remotely. For example, we can pre-decide the time of switching on the television or making a call to the fire department in-case of fire. We can also turn our appliances on or off as required even when we are away from our homes.

- **Body Sensors**

Maintaining a good health is a big challenge for people today. Due to busy schedules, visiting doctors every now and then for regular checkups is also difficult for a huge chunk of population but this problem can also be solved with the help of wearable devices such as fitness trackers that help in tracking blood sugar levels, heartbeat, cholesterol levels and much more thereby helping in health management. These sensors can also be used by construction companies to detect the posture of their laborers in order to avoid any kind injuries while working

- **Robots for Manufacturing Industries**

Manufacturing Industries also make use of robots for manufacturing processes and these robots are nothing but another kind of AI embedded IoT devices. They help in enhancing the manufacturing processes by saving time and cost of processing. An example is the use of robots by eye wear manufacturers for manufacturing lenses with a great precision.

- **Face Detection**

Face detectors are another important use case of AIoT. Face detection becomes important for crime investigation departments and even in offices for detecting the faces of employees for the purpose of attendance. Another interesting area where face detectors are being used currently are shopping malls and other public places to keep a check on whether people are wearing masks or not and punishing the defaulters accordingly.

- **Retail Analytics**

Management of staff in retail outlets is an important task because both over staffing and under staffing can lead to inefficient operations. With the use of sensors and AI, however, people entering the outlets and their movement inside the outlet can be observed in order to estimate the time they will take to reach the checkout line. The staff at the counter can then be increased or decreased accordingly to reduce the checkout time and increase productivity.

The captured data can also be used later for determining the peak hours and formulate management strategies well in advance.

- **Smart Buildings**

Another area of intersection of IoT and AI is smart office buildings. So, not only homes but a whole building can also have AIoT installed for better operational efficiency and management cost. Some companies for example, install a network of AIoT devices in their buildings and these devices can detect the presence of personnel and adjust the temperatures accordingly or turn off appliances where no one is present thereby increasing energy efficiency which ultimately leads to lower costs.

Deep Learning

Deep learning attempts to mimic the human brain—albeit far from matching its ability—enabling systems to cluster data and make predictions with incredible accuracy. Deep learning is a subset of machine learning, which is essentially a neural network with three or more layers. These neural networks attempt to simulate the behavior of the human brain—albeit far from matching its ability—allowing it to "learn" from large amounts of data. While a neural network with a single layer can still make approximate predictions, additional hidden layers can help to optimize and refine for accuracy. Deep learning drives many artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention. Deep learning technology lies behind everyday products and services as well as emerging technologies (such as self-driving cars).

1.4. AIM AND OBJECTIVE

Aim

The aim of the Advanced ATM Simulator with Facial Recognition Security project is to develop a sophisticated software solution that replicates the functionalities of a physical ATM while integrating advanced facial recognition technology for enhanced security measures. This project aims to provide users with a realistic and secure banking experience while offering financial institutions robust tools to safeguard against unauthorized access and fraudulent activities.

Objectives

- To Develop Comprehensive ATM Functionality
- To Integrate Facial Recognition Technology

- To Continuously Train and Optimize the Facial Recognition Model
- To Enhance Security Measures with Additional Features
- To Design an Intuitive User Interface for Seamless Interaction
- To Ensure Real-Time Transaction Processing
- To Offer Customization Options and Settings
- To Conduct Rigorous Testing and Validation
- To Prepare Comprehensive Documentation and Training Materials
- To Deploy the System and Provide Ongoing Support

1.5. SCOPE OF THE PROJECT

The scope of the project encompasses the development and implementation of an ATM User Face Identification System, leveraging advanced technologies such as facial recognition, real-time notifications and an Unknown Face Forwarder Link system. The key components and functionalities within the scope of the project include:

- **Facial Recognition Module**: Implementing a robust facial recognition system using Convolutional Neural Networks (CNNs) to accurately identify ATM users based on their facial features.
- **Unknown Face Forwarder Link System**: Introducing a mechanism to generate Unknown Face Forwarder Links in instances where facial recognition fails to match a user's face, allowing users to verify their identity through alternate means.

- **Real-time Notification System**: Developing a notification mechanism to keep users informed about transaction details and security alerts, facilitating transparent communication and enhancing user trust.
- **Database Management**: Designing and implementing a secure and scalable database architecture to store user information, transaction records, facial recognition data, and verification links efficiently.
- **User Interface Enhancement**: Enhancing the user interface of the ATM system to seamlessly integrate facial recognition instructions, transaction status updates, and Unknown Face Forwarder Link prompts, ensuring a user-friendly experience.
- **Testing and Evaluation**: Conducting thorough testing and evaluation of the system in a controlled environment to validate its functionality, reliability, and security.

The scope of the project is focused on delivering a comprehensive ATM User Face Identification System that not only enhances security measures but also improves the overall user experience by incorporating advanced features such as the Unknown Face Forwarder Link system.

CHAPTER 2

SYSTEM REQUIREMENT

2.1. HARDWARE REQUIREMENT

- **Processor** : Intel Core i5 or higher
- **RAM** : 8 GB or more
- **Storage** : SSD (Solid State Drive) with at least 256 GB capacity

2.2. SOFTWARE REQUIREMENT

- **Operating System** : Windows 10 or 11
- **Programming** : Python
- **Web Framework** : Flask
- **Database** : MySQL
- **Web Server** : WampServer
- **Framework** : Bootstrap, TensorFlow
- **Libraries** : Pandas, NumPy, and Scikit-learn
- **Data Visualization** : Matplotlib and Seaborn
- **Image Processing** : OpenCV and Pillow
- **Web Technologies** : HTML, CSS, and JavaScript

2.3. DATABASE

The Face-Enabled ATM System requires a secure and structured database to manage user data, facial recognition metadata, and transaction records.

Entities

1. **User**
    - Attributes: UserID (PK), Name, AccountNumber, PhoneNumber, Email, Password, FaceID
2. **Account**
    - Attributes: AccountNumber (PK), UserID (FK), AccountType, Balance
3. **Transaction**
    - Attributes: TransactionID (PK), AccountNumber (FK), TransactionType, Amount, DateTime
4. **AuthenticationLog**
    - Attributes: LogID (PK), UserID (FK), LoginTime, FaceMatchResult, Status

Relationships

- A User can have one or more Accounts (One-to-Many).
- An Account can have multiple Transactions (One-to-Many).
- A User can have multiple AuthenticationLogs (One-to-Many).

Normalization

- **1NF**: All tables have atomic attributes.
- **2NF**: All non-key attributes fully depend on the primary key (e.g., Balance depends only on AccountNumber).
- **3NF**: No transitive dependencies exist (e.g., User table stores user-specific data only, not account or transaction data).

This design ensures data integrity, minimizes redundancy, and supports secure and scalable operations for face-based ATM access.

CHAPTER 3

ABOUT THE SOFTWARE

3.1. PYTHON 3.7.4

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL). This tutorial gives enough understanding on Python programming language.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain.

Python is currently the most widely used multi-purpose, high-level programming language. Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java. Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time. Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc. The biggest strength of Python is huge collection of standard library which can be used for the following:

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like OpenCV, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

- Scientific computing
- Text processing and many more.

Tensor Flow

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and gives developers the ability to easily build and deploy ML-powered applications.

TensorFlow provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. Developers have the option to deploy models on a number of platforms such as on servers, in the cloud, on mobile and edge devices, in browsers, and on many other JavaScript platforms. This enables developers to go from model building and training to deployment much more easily.

Keras

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation.

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.

Pandas

pandas are a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. pandas are a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

Pandas is mainly used for data analysis and associated manipulation of tabular data in Data frames. Pandas allows importing data from various file formats such as comma-separated values, JSON, Parquet, SQL database tables or queries, and Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features. The development of pandas introduced into Python many comparable features of working with Data frames that were established in the R programming language. The panda's library is built upon another library NumPy, which is oriented to efficiently working with arrays instead of the features of working on Data frames.

NumPy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Matplotlib

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

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

Scikit Learn

scikit-learn is a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license.

Scikit-learn (formerly scikits. learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Pillow

Pillow is the friendly PIL fork by Alex Clark and Contributors. PIL is the Python Imaging Library by Fredrik Lundh and Contributors.

Python pillow library is used to image class within it to show the image. The image modules that belong to the pillow package have a few inbuilt functions such as load images or create new images, etc.

OpenCV

OpenCV is an open-source library for the computer vision. It provides the facility to the machine to recognize the faces or objects.

In OpenCV, the CV is an abbreviation form of a computer vision, which is defined as a field of study that helps computers to understand the content of the digital images such as photographs and videos.

3.2. MYSQL

MySQL is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. MySQL database that provides for how to manage database and to manipulate data with the help of various SQL queries. These queries are: insert records, update records, delete records, select records, create tables, drop tables, etc. There are also given MySQL interview questions to help you better understand the MySQL database.

MySQL is currently the most popular database management system software used for managing the relational database. It is open-source database software, which is supported by Oracle Company. It is fast, scalable, and easy to use database management system in comparison with Microsoft SQL Server and Oracle Database. It is commonly used in conjunction with PHP scripts for creating powerful and dynamic server-side or web-based enterprise applications. It is developed, marketed, and supported by MySQL AB, a Swedish company, and written in C programming language and C++ programming language. The official pronunciation of MySQL is not the My Sequel; it is My Ess Que Ell. However, you can pronounce it in your way. Many small and big companies use MySQL. MySQL supports many Operating Systems like Windows, Linux, MacOS, etc. with C, C++, and Java languages

-3.3. WAMPSERVER

WampServer is a Windows web development environment. It allows you to create web applications with Apache2, PHP and a MySQL database. Alongside, PhpMyAdmin allows you to manage easily your database.

WAMPServer is a reliable web development software program that lets you create web apps with MYSQL database and PHP Apache2. With an intuitive interface, the application features numerous functionalities and makes it the preferred choice of developers from around the world. The software is free to use and doesn’t require a payment or subscription.

3.4. BOOTSTRAP 4

Bootstrap is a free and open-source tool collection for creating responsive websites and web applications. It is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.

It solves many problems which we had once, one of which is the cross-browser compatibility issue. Nowadays, the websites are perfect for all the browsers (IE, Firefox, and Chrome) and for all sizes of screens (Desktop, Tablets, Phablets, and Phones). **Easy to use**: Anybody with just basic knowledge of HTML and CSS can start using Bootstrap

Responsive features: Bootstrap's responsive CSS adjusts to phones, tablets, and desktops

Mobile-first approach: In Bootstrap, mobile-first styles are part of the core framework

Browser compatibility: Bootstrap 4 is compatible with all modern browsers (Chrome, Firefox, Internet Explorer 10+, Edge, Safari, and Opera)

3.5. FLASK

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Using an IDE

As good as dedicated program editors can be for your programming productivity, their utility pales into insignificance when compared to Integrated Developing Environments (IDEs), which offer many additional features such as in-editor debugging and program testing, as well as function descriptions and much more.

Flask is often referred to as a micro framework. It aims to keep the core of an application simple yet extensible. Flask does not have built-in abstraction layer for database handling, nor does it have formed a validation support. Instead, Flask supports the extensions to add such functionality to the application. Although Flask is rather young compared to most Python frameworks, it holds a great promise and has already gained popularity among Python web developers. Let's take a closer look into Flask, so-called "micro" framework for Python. Flask was designed to be easy to use and extend. The idea behind Flask is to build a solid foundation for web applications of different complexity. From then on you are free to plug in any extensions you think you need. Also you are free to build your own modules. Flask is great for all kinds of projects. It's especially good for prototyping. Flask is part of the categories of the micro-framework. Micro-framework is normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependency to update and watch for security bugs, cons is that sometime you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins.

CHAPTER 4
PROBLEM DEFINITION AND PROPOSAL

4.1. EXISTING SYSTEM

Existing systems of ATM authentication primarily rely on traditional methods such as card-based authentication and PIN entry. However, advancements in technology have led to the adoption of additional authentication mechanisms to enhance security and user convenience. Here are some existing systems of ATM authentication:

- **Card and PIN Authentication**: This is the most common method used in ATM authentication. Users insert their ATM card into the card reader slot and enter their Personal Identification Number (PIN) using the keypad. The system verifies the entered PIN against the one stored on the card's chip or magnetic stripe.
- **Biometric Authentication**: Some ATMs incorporate biometric authentication methods such as fingerprint recognition or iris scanning. Users can register their biometric data with the bank, and the ATM system verifies their identity by matching the biometric data captured at the time of authentication.
- **Two-Factor Authentication (2FA)**: In addition to the traditional card and PIN authentication, some ATMs implement two-factor authentication. This involves combining something the user knows (PIN) with something the user has (e.g., a mobile device or token). For example, users may receive a one-time password (OTP) on their registered mobile phone that they need to enter along with their PIN for authentication.
- **Near Field Communication (NFC)**: NFC technology allows users to authenticate transactions using contactless methods such as mobile wallets or contactless cards. Users can tap their NFC-enabled device or card on the ATM reader to initiate a transaction, eliminating the need for physical insertion of the card.
- **Mobile Authentication**: Some banks offer mobile banking apps with built-in authentication features. Users can authenticate transactions at the ATM by scanning a QR code displayed on the ATM screen using their mobile banking app. The app then prompts the user to authorize the transaction on their mobile device.
- **Token-Based Authentication**: Some banks issue physical or virtual tokens to customers for authentication purposes. These tokens generate one-time passwords

(OTPs) that users need to enter along with their PIN to complete transactions at the ATM.

4.1.1 Disadvantages

- The accuracy of the system is not 100%.
- Face detection and loading training data processes just a little bit slow.
- It can only detect face from a limited distance.
- It cannot repeat live video to recognize missed faces.
- The instructor and training Set manager still have to do some work manually.
- Unimodal biometric systems have to contend with a variety of problems such as noisy data, intraclass variations, restricted degrees of freedom, non-universality, spoof attacks, and unacceptable error rates.
- This method is not very secure and prone to increase in criminal activities.
- QRcode scanner is required to detect code
- Should carry the mobile phone with app installed on it

4.2. PROPOSED SYSTEM

The proposed system for the ATM User Face Identification project involves integrating facial recognition technology into the existing ATM infrastructure. Here's an overview of the key features and components:

- **Facial Recognition Module**

The Facial Recognition Module integrates advanced facial recognition technology, leveraging a Convolutional Neural Network (CNN) trained on a dataset of facial images. This module is responsible for accurately detecting and analyzing facial features captured by the ATM's high-resolution camera during transactions. Upon initiation of a transaction, the user's face is captured and processed to extract facial features. These features are then compared with pre-existing facial templates stored in the system's database to verify the user's identity. The CNN continually learns and adapts to improve accuracy and performance over time.

- **Unknown Face Verification System**

In cases where the user's face is not recognized or matches with an unknown identity, the Unknown Face Verification System is activated. This system generates a unique Face Verification Link and securely transmits it to the user's registered mobile number. The Face Verification Link serves as an additional authentication step, allowing the user to confirm their identity by clicking on the link and completing further verification steps, such as entering a

one-time code or confirming the transaction details. This process enhances security and provides users with an alternative verification method.

- **Notification System**

The Notification Module is designed to deliver real-time transaction updates and security alerts to users. Users receive notifications via their preferred communication channels, including SMS, email, or in-app alerts. Transaction details, such as withdrawal amounts, account balances, and transaction confirmations, are promptly communicated to the user to ensure transparency and security. Additionally, the Notification Module alerts users of any suspicious activities or security breaches, enabling them to take immediate action, such as contacting their bank or reporting the incident. This proactive approach enhances user awareness and helps mitigate potential risks associated with ATM transactions.

4.2.1. Advantages

- The advantages can be found as that the face-id is unique for everybody; it cannot be used by anybody other than the user.
- It can be used to reduce fraudulent attempts.
- To prevent theft and other criminal activities.
- Secure facial authentication platform that users can trust
- Provide safe and secure lifestyle infrastructure
- Prevent unauthorized access using Face verification Link.
- Fast and Accurate Prediction

4.3. ALGORITHM

Step 1: Account Creation & Enrollment

- Capture Facial Image using bank’s enrollment camera.
- Preprocess Image (convert to grayscale, resize, normalize).
- Generate Face Embedding using a pre-trained FaceNet model.
- Store Embedding securely in the bank's database linked to the user's account.

Step 2: Transaction Initiation at ATM

- User Approaches ATM, camera captures live facial image.
- Preprocess Captured Image similarly (grayscale, resize, normalize).
- Detect and Align Face using MTCNN (Multi-task Cascaded CNN).
- Generate Live Face Embedding using the same FaceNet model.

Step 3: Authentication

- If D ≤ Threshold, proceed to TCN step; else, trigger alert.

Step 4: Liveness Detection using TCN

- Feed Real-Time Video Sequence to Temporal Convolutional Network (TCN).
- Verify Temporal Facial Dynamics to ensure user is live (not spoofed).
- If TCN Confirms Live Match, grant access; else, flag suspicious activity.

Step 5: Unauthorized Access Handling

- Send AI-Powered Facial Verification Link to account holder's mobile/email.
- User Re-authenticates Remotely, confirms or denies access attempt.
- System Logs Activity and alerts bank authorities if unauthorized.

Step 6: Transaction Execution

- If all checks pass: allow PIN input & transaction processing.
- Else: deny access and notify both user and bank system.

This algorithm integrates FaceNet for identity verification and TCN for liveness detection, significantly enhancing ATM transaction security without requiring physical cards or PINs alone.

4.4. FEASIBILITY STUDY

4.4.1. TECHNOLOGY FEASIBILITY

The ATM User Face Identification System is technically viable as it integrates advanced machine learning algorithms, image processing techniques, and web-based frameworks for secure user authentication. The system requires ATMs equipped with high-resolution cameras, secure network connectivity, and fast processing units to ensure real-time recognition. Python serves as the primary programming language, with Flask used for developing the web-based interface. MySQL is utilized for secure data storage, while TensorFlow powers the facial

recognition model. OpenCV and Pillow facilitate image processing and face detection, ensuring accurate and efficient performance. Given the availability of these technologies, the system is technically feasible.

4.4.2. ECONOMIC FEASIBILITY

Implementing this system requires investment in software development, AI model training, and ATM hardware upgrades. While initial costs include purchasing high-quality cameras and ensuring secure network configurations, the long-term benefits outweigh these expenses. The system minimizes fraud, reducing financial losses from unauthorized transactions. Additionally, it enhances user convenience by eliminating the need for PINs and physical cards, thereby decreasing operational costs associated with lost or stolen cards. A cost-benefit analysis suggests that the security enhancements and improved user experience justify the investment, making the project economically feasible.

4.4.3. OPERATIONAL FEASIBILITY

The system is designed to be user-friendly and seamlessly integrates into existing banking infrastructures. With a high recognition accuracy of 97.93%, users can rely on secure and efficient authentication. The response time is optimized to ensure quick face detection and verification, preventing transaction delays. Scalability is a key advantage, as the system can be deployed across multiple ATMs without significant modifications. Security measures, including encryption of biometric data and fraud detection mechanisms, enhance system reliability. Given these operational advantages, the system is feasible for large-scale implementation.

CHAPTER 5
SYSTEM DESING

5.1. INTRODUCTION

The system design of the Secure ATM model integrates deep learning-based facial recognition and liveness detection to enhance ATM transaction security. During account creation, the user's facial image is processed using a Convolutional Neural Network (CNN) to generate a unique embedding via the FaceNet model, which is securely stored in the bank's server. At the ATM, a live video feed is captured and processed using a Temporal Convolutional Network (TCN) to ensure the presence of a real person. The system then compares the live embedding with the stored embedding to verify the identity. Secure communication protocols ensure encrypted data transmission between the ATM and the server. In cases of failed authentication, the system sends a facial recognition verification link to the user's device, adding an extra layer of security. The design prioritizes real-time processing, user privacy, and robust fraud prevention.

5.2. MODULES DESCRIPTION

5.2.1. ATM Simulator

The ATM Simulator is designed to replicate the functionalities of a physical Automated Teller Machine. The User Authentication Module ensures secure access by validating user credentials and verifying Personal Identification Numbers (PINs). The Account Overview Module provides users with a snapshot of their accounts, displaying balances and recent transactions. For transactions, the Cash Withdrawal Module simulates the withdrawal process, deducting the chosen amount from the account and dispensing virtual cash. Conversely, the Deposit Module allows users to input amounts for deposit, updating their account balances accordingly. The Transfer Module facilitates virtual fund transfers between accounts seamlessly. Additional modules include Change PIN for security updates, Transaction History for a detailed record of activities, and Card Management for actions like blocking or unblocking cards. The Alerts and Notifications Module keeps users informed of account activities, while the Settings Module enables customization of preferences. Finally, the Logout Module terminates user sessions for enhanced security. Together, these modules create a comprehensive and realistic ATM simulation experience for users and developers alike.

5.2.2. End User Interface

2.1. ATM System

Cardholder Interaction

Upon inserting their ATM card into the interface, users kick-start transactions. The system promptly reads the card details to facilitate seamless transaction processing.

Facial Recognition

Simultaneously, the system employs advanced facial recognition technology to capture the user's face. This captured image is then meticulously compared with the pre-trained face model stored in the system's database.

Security Measures

In scenarios where a facial match is identified, the transaction proceeds effortlessly. However, in cases of non-matching faces, the system activates additional security measures to safeguard the transaction.

Face Verification Link

For added security, the system generates a Face Verification Link. This link is promptly dispatched to the mobile number linked to the card account, ensuring an extra layer of identity confirmation.

User Approval Process

The cardholder, upon receiving the Face Verification Link, is prompted to verify their identity. Once approval is granted, the ATM dispenses the requested amount; if not, the system securely retrieves the card to prevent unauthorized access.

2.2. ATM User/Account Holder Interaction

Cash Withdrawal

When an ATM user seeks to withdraw cash, they simply insert their card, initiating a streamlined process. The system then verifies the user through multi-factor authentication.

2.3. Bank Employee Interaction

Secure Login

Bank employees gain access to the system through a secure login process, ensuring that only authorized personnel can manage the ATM functionalities.

Account Management

Equipped with secure access, bank employees can seamlessly create new bank accounts, streamlining the on boarding process for customers.

Facial Recognition Training

As a security enhancement, the system allows bank employees to record a live video for approximately 30 seconds to train the account holder's face. This data contributes to the continuous enhancement of the facial recognition model, fortifying overall security.

Generate ATM ID, Dispatch to Account Holder

Upon the creation of a new account, the system generates a unique ATM ID. This ID is dispatched to the account holder, completing the account creation process.

5.2.3. Face Recognition

3.1. Dataset Creation: Account Holder Face by Recording Live Video for 30secs

The process begins with actively creating a comprehensive dataset by recording a live video of the account holder's face for approximately 30 seconds. This dynamic dataset serves as the bedrock for training the subsequent face recognition model. It ensures a diverse collection of facial expressions, angles, and lighting conditions for robust model training.

- **Frame Conversion**

Post dataset acquisition, the system seamlessly converts video frames into individual images. This step is pivotal for simplifying subsequent image processing and analysis. It enables the system to work with discrete frames, facilitating more efficient handling and manipulation during pre-processing.

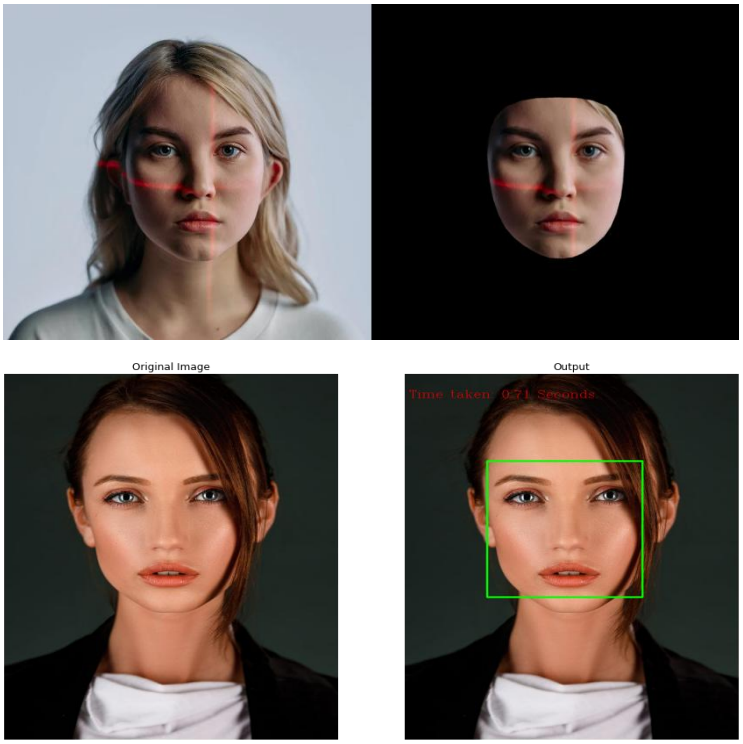
3.2. Pre-processing

This pre-processing module encompasses multiple steps to optimize images for subsequent analysis.

- **Grey Scale Conversion:** The conversion to greyscale simplifies image representation, reducing complexity.
- **Noise Filter:** The application of mean or Gabor filtering minimizes noise, enhancing the clarity of facial features.
- **Binarize:** Converting images to binary format further streamlines feature extraction. These steps collectively contribute to creating a standardized and enhanced image dataset.

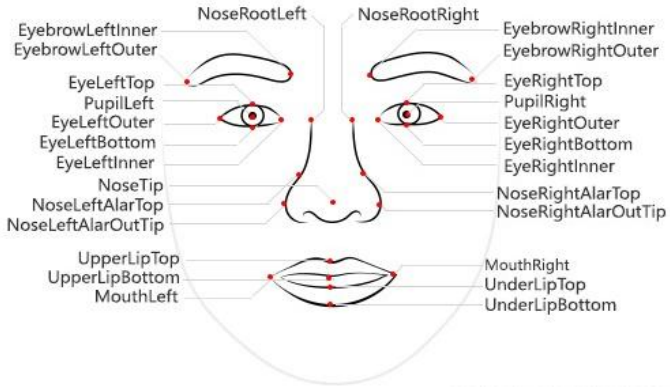
3.3. Face Detection

Leveraging a Region Proposal Network (RPN), this module identifies potential face regions within the pre-processed images. RPN excels at proposing regions likely to contain facial features, laying the groundwork for subsequent processing. It streamlines the computational effort by focusing on regions of interest, enhancing efficiency in face recognition.



3.4. Face Feature Extraction

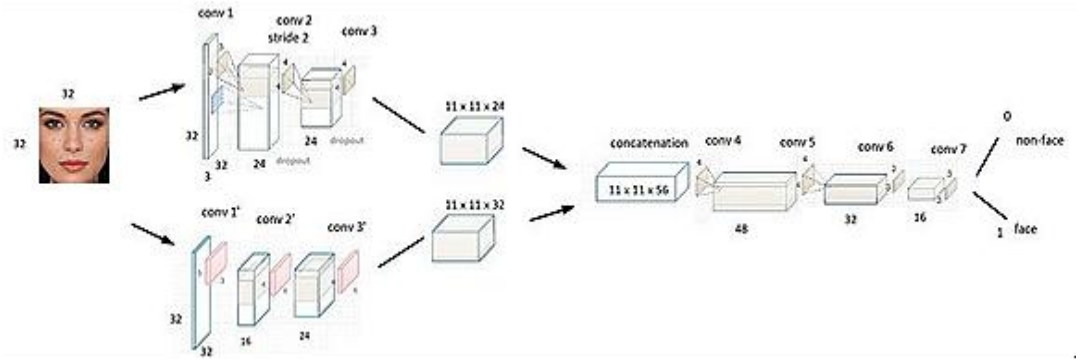
This module focuses on extracting relevant features from the detected face regions using Gray Level Co-occurrence Matrix (GLCM). GLCM captures statistical information about pixel intensity relationships, offering a rich set of features for subsequent classification. It serves as a robust method for characterizing facial textures and patterns.



3.5. Classification using CNN (Convolutional Neural Network)

Employing a Convolutional Neural Network (CNN), this module classifies the extracted face features. The CNN is trained on the dataset, learning intricate patterns and representations

crucial for accurate face recognition. The hierarchical structure of a CNN enables it to automatically learn discriminative features from the input images.



3.6. Build and Train Model

This module involves the construction of the face recognition model using the pre-processed dataset and the training of the model on the extracted features. The model is iteratively optimized using machine learning techniques to ensure accurate face recognition across diverse conditions.

3.7. Deploy the Model in ATM Server

Upon successful training, the face recognition model is seamlessly integrated and deployed within the ATM server. This integration allows real-time face recognition during transactions, contributing to enhanced security and user authentication.

5.2.4. Face Identification

4.1. ATM Captures User's Face

The Face Identification module commences with the ATM employing integrated cameras or sensors to capture a live image of the user's face during a transaction. This process is crucial for obtaining a real-time representation of the user's facial features, capturing nuances such as facial expressions, contours, and unique identifiers.

4.2. Extract Features

Following the capture of the facial image, the system moves to the extraction phase where it identifies and isolates key features from the user's face. These features include but are not limited to facial landmarks, texture patterns, and distinctive attributes that collectively contribute to a comprehensive and unique facial profile.

4.2. Identify Users with Trained Model

The crux of the Face Identification module lies in the identification process, where the extracted facial features are compared with a pre-trained face identification model. This model has undergone meticulous training on a diverse dataset containing facial images of authorized

users. It has learned to recognize and differentiate specific individuals based on their unique facial characteristics.

Prediction Process: In real-time identification, the extracted features are compared with the patterns stored in the pre-trained model. This involves intricate algorithms that evaluate the similarity between the presented facial features and the learned patterns. If a substantial match is detected, the system confidently identifies the user.

Authorization Decision: The identification process ultimately influences an authorization decision, determining whether the user is authorized to proceed with the requested transaction. This decision is pivotal for ensuring that access to ATM services is granted only to individuals with recognized and authorized facial profiles.

5.2.5. Face Verification Link Generator

5.1. Generate Face Verification Link

In response to non-matching faces, the system promptly generates a Face Verification Link. This link is designed to serve as a secure and temporary reference for the user's facial profile, introducing an extra layer of security for subsequent verification steps.

5.2. Link Transmission to User's Mobile Number

The Face Verification Link is swiftly transmitted to the user's mobile number, enhancing security by sending the link to the device associated with the authorized account holder. This proactive step ensures that the user is promptly informed and involved in the verification process.

5.2.6. Face Verification Process

Upon receiving the Face Verification Link, the authorized account holder engages in a secure and user-friendly process to validate the user at the ATM. The following steps outline the Face Verification Process:

6.1. Link Access and View User

The authorized account holder receives the Face Verification Link on their mobile device. By clicking the link, they are directed to a secure page displaying the captured image of the user at the ATM.

6.2. User Approval

After viewing the user's image, the authorized account holder has the option to approve the user's identity. This approval step is crucial for authorizing the subsequent transaction and confirming that the individual at the ATM is indeed the legitimate account holder.

6.3. Enter Amount and PIN

Upon approving the user, the account holder proceeds to enter the desired withdrawal amount and their unique Personal Identification Number (PIN). This step confirms their intention to proceed with the transaction.

6.4. Confirmation and Money Dispensation

With the entered amount and PIN, the account holder confirms the transaction. The ATM machine, recognizing the approved user, dispenses the requested amount of money.

6.5. Unknown User Handling

In the event that the user at the ATM is unrecognized or deemed unauthorized, the account holder has the option to take immediate action. This may include blocking the user directly through the link interface, enhancing security measures against potential fraudulent activities.

5.2.7. Notification

The Notification Module is triggered by various events within the system. These events could include successful transactions, security alerts, account activity updates, or any other significant occurrences requiring user attention. The module supports different types of notifications, such as in-app alerts, SMS messages, or email notifications. Users are notified in real-time about successful or unsuccessful transactions. Transaction alerts include details such as transaction amount, date, time, and the location of the transaction. In the case of suspicious activities or potential security threats, users receive security alerts. These alerts may include unauthorized access attempts, unusual spending patterns, or any activity that deviates from normal user behaviour. Users are kept informed about changes to their account, such as balance updates, account statements, or any modifications to personal information.

5.3. UML DIAGRAM

5.3.1. USE CASE DIAGRAM

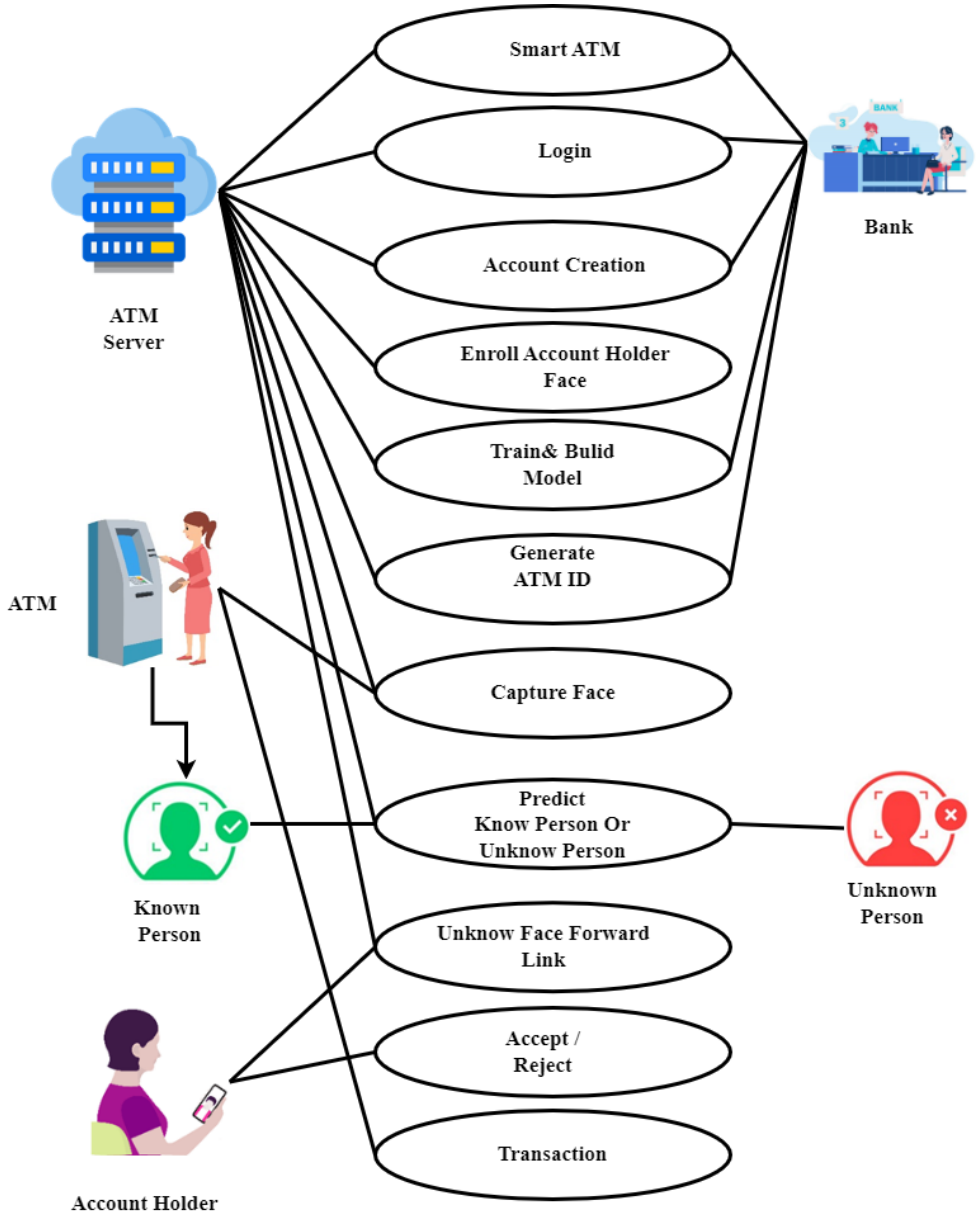


Fig: 5.3.1. Use case Diagram

5.3.2. CLASS DIAGRAM

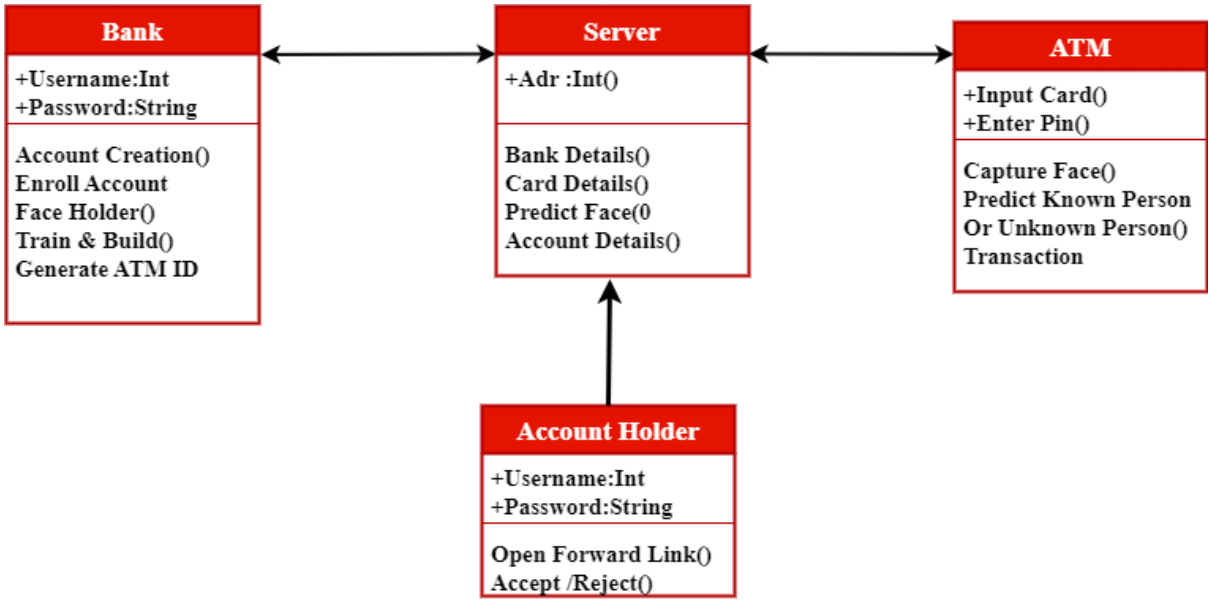
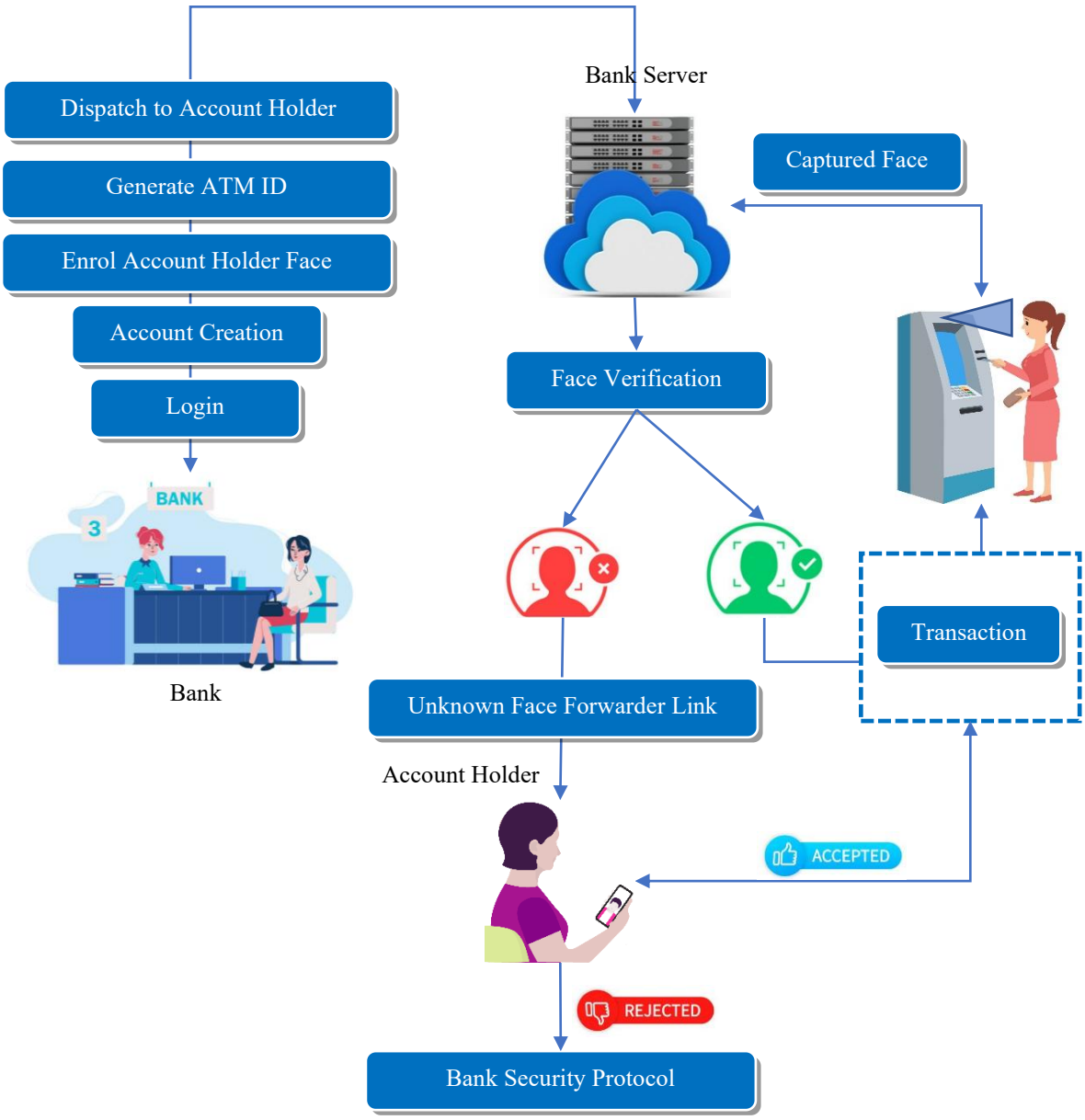


Fig: 5.3.2. Class Diagram

5.3.3. WORK FLOW DIAGRAM



5.3.3.1. WORK FLOW DIAGRAM

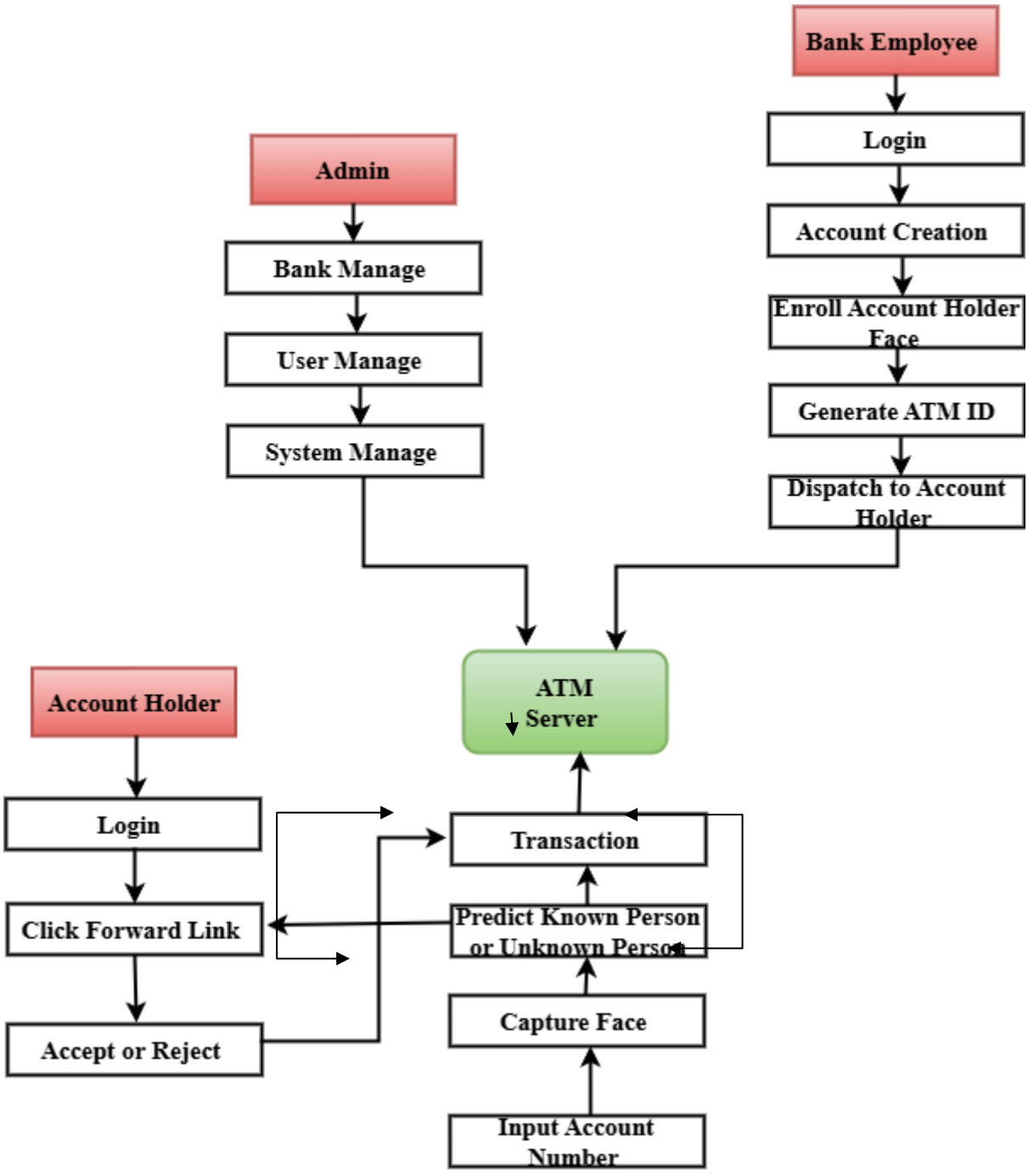


Fig: 5.3.3. Work Flow Diagram

5.4. DATA FLOW DIAGRAM

LEVEL 0

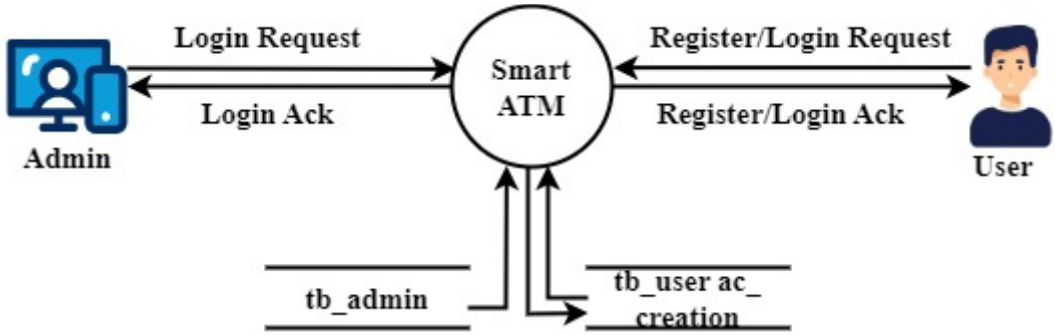


Fig: 5.4.1. Data Flow Diagram Level 0

LEVEL 1

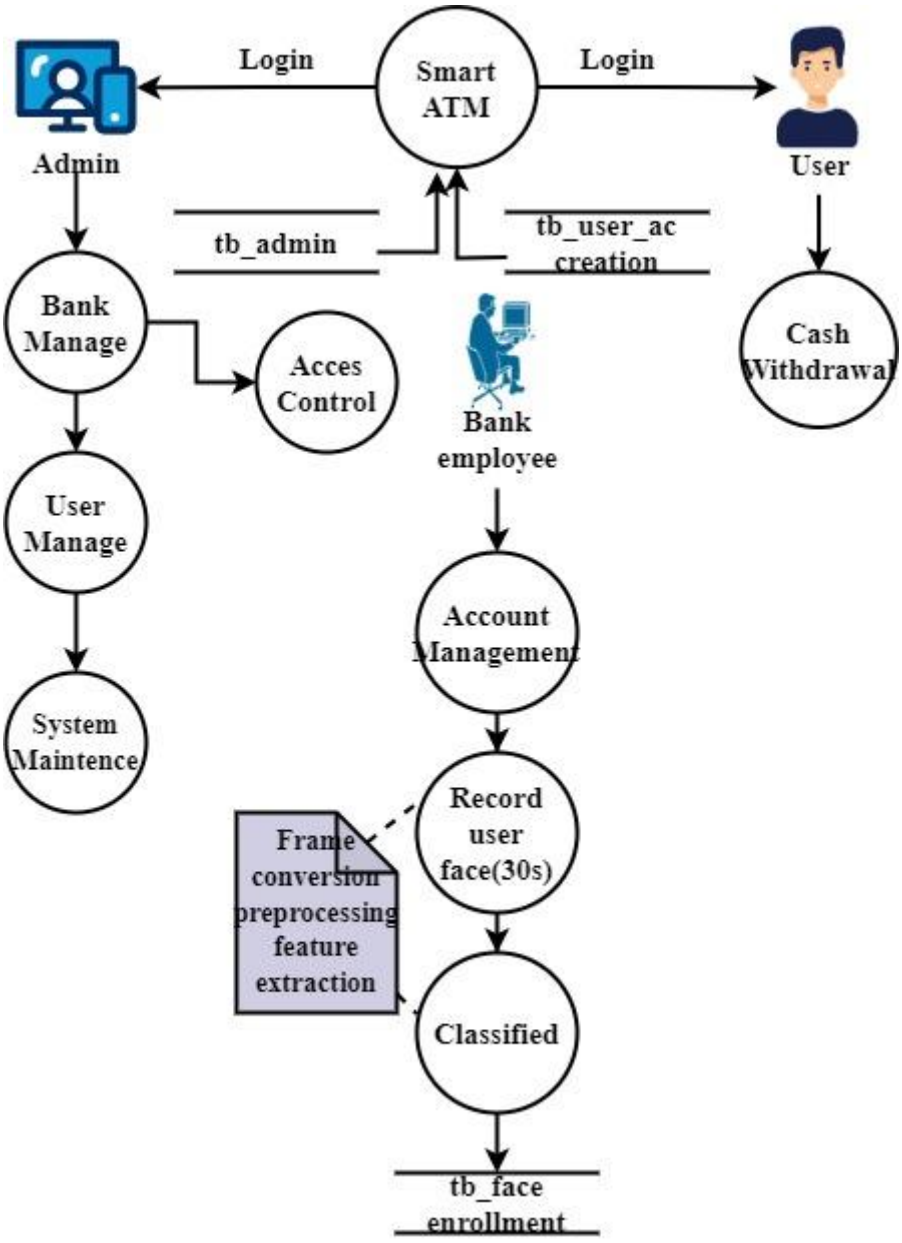


Fig: 5.4.2. Data Flow Diagram Level 1

LEVEL 2

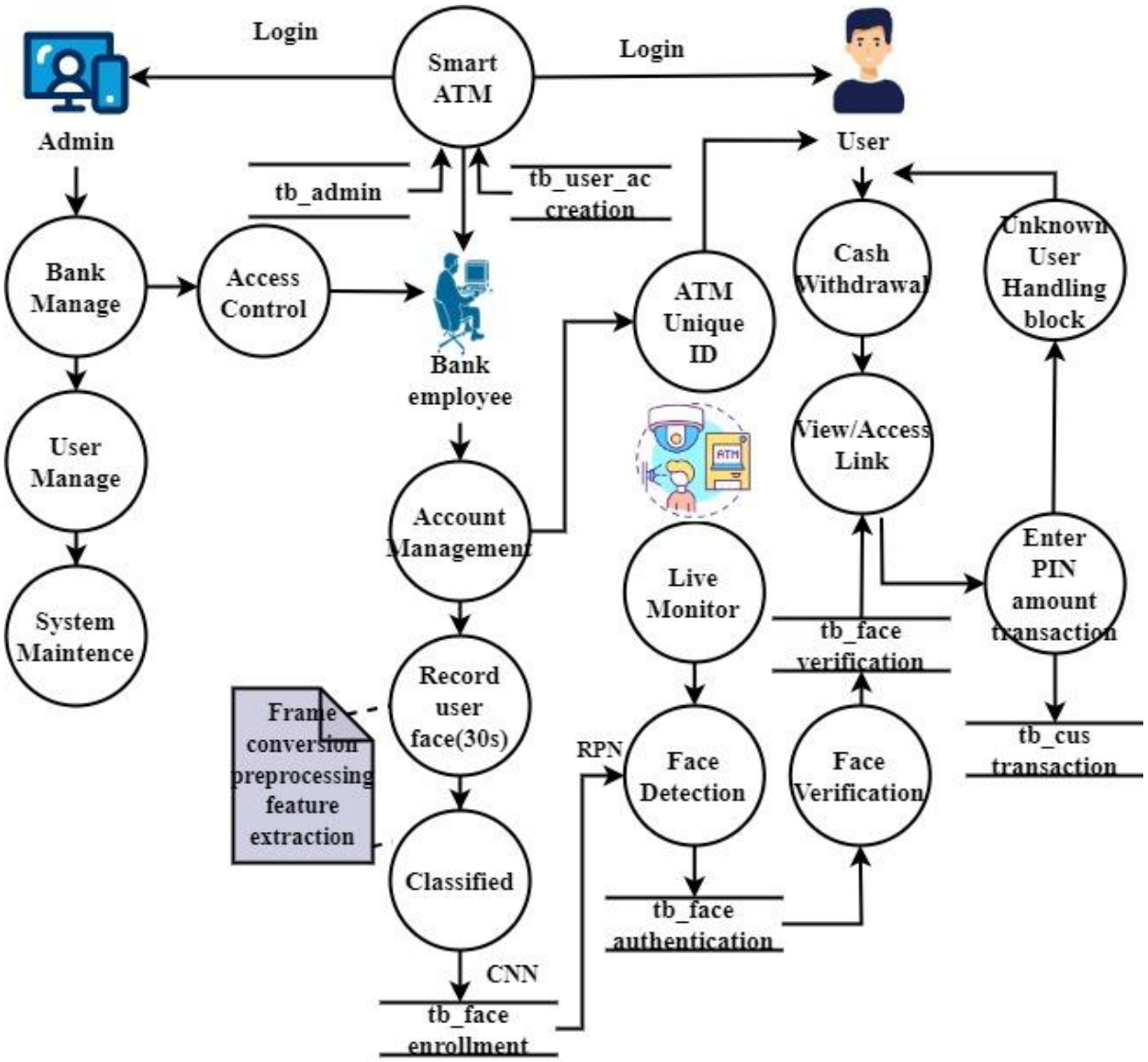


Fig: 5.4.3. Data Flow Diagram Level 2

5.5. DATA DICTIONARY

Table Name: Admin					
S.no	**Field**	**Data Type**	**Field Size**	**Constraint**	**Description**
1	User name	Varchar	20	Null	Admin name
2	Password	Varchar	20	Null	Admin Password

Table Name: Customer Account Creation					
S.no	**Field**	**Data Type**	**Field Size**	**Constraint**	**Description**
1	Id	Varchar	20	Null	Id
2	Name	Varchar	20	Null	Name
3	Address	Varchar	50	Null	Address
4	Mobile number	Bigint	20	Null	Mobile number
5	Email	Varchar	50	Null	Email
6	Account number	Varchar	30	Null	Account number
7	Debit card number	Varchar	30	Null	Debit card number
8	Bank Name	Varchar	20	Null	Bank Name
9	Branch Name	Varchar	20	Null	Branch Name
10	Deposit amount	Double	500	Null	Deposit amount
11	Customer id	Int	11	Primary key	Customer id
12	Password	Varchar	20	Null	Password
13	Create date	Timestamp	Timestamp	Null	Create date

Table Name: Face Enrolment					
S.no	**Field**	**Data Type**	**Field Size**	**Constraint**	**Description**
1	Id	Int	20	Null	Id
2	Customer id	Varchar	30	Foreign key	Customer id
3	Face image	Varchar	30	Null	Face image
6	Trained Model	Varchar	30	Primary key	Classified label

Table Name: Face Authentication					
S.no	**Field**	**Data Type**	**Field Size**	**Constraint**	**Description**
1	Id	Int	20	Null	Id
2	Customer id	Varchar	30	Foreign key	Customer id
3	Test image	Varchar	30	Null	Test image
4	Verify status	Int	11	Null	Verify status
5	Unknown Face alert	Varchar	30	Null	Unknown face alert
6	Date time	Timestamp	Timestamp	Null	Date time

Table Name: Customer Transaction					
S.no	**Field**	**Data Type**	**Field Size**	**Constraint**	**Description**
1	Id	Int	20	Null	Id
2	Customer id	Varchar	30	Foreign key	Customer id
3	With draw amount	Double	500	Null	Withdraw amount
4	With draw alert	Varchar	20	Null	Withdraw alert
5	Date time	Timestamp	Timestamp	Null	Date time

Table Name: Face Verification link					
S.no	**Field**	**Data Type**	**Field Size**	**Constraint**	**Description**
1	Id	Int	20	Null	Id
2	Customer id	Varchar	30	Foreign key	Customer id
3	Unknown face image link	Varchar	30	Null	Unknown face image link
4	Withdraw Permission	Int	11	Null	Withdraw Permission
5	Withdraw Amount	Double	500	Null	Withdraw Amount
6	Date time	Timestamp	Timestamp	Null	Date time

CHAPTER 6
IMPLEMENTATION

6.1. INTRODUCTION

The implementation of the Secure ATM system begins with capturing facial data from users during account creation at the bank, where a Convolutional Neural Network (CNN) powered by FaceNet generates and stores unique facial embeddings. At the ATM, a live camera captures the user's face, and a Temporal Convolutional Network (TCN) is employed to analyze the video stream and detect liveness, ensuring the presence of a real person. The system compares the real-time facial data with the stored embeddings for authentication. If a mismatch or unauthorized access is detected, a facial verification link is immediately sent to the registered user for identity confirmation. This end-to-end implementation ensures secure, efficient, and user-friendly ATM transactions by combining computer vision, deep learning, and real-time AI verification techniques.

6.2. SYSTEM IMPLEMENTATION

System implementation involves translating the designed Face-Enabled ATM system into a fully functional prototype capable of providing secure, biometric-based user authentication and transaction processing.

1. **Environment Setup**: Configuring hardware (webcam, simulation modules), software libraries (OpenCV, TensorFlow, Flask), and network components to support biometric data capture, face recognition processing, and secure link-based verification workflows.
2. **Development of Facial Recognition Model**: Implementing and training a Convolutional Neural Network (CNN) using libraries like TensorFlow or PyTorch for real-time face detection and recognition. The model is trained on a dataset of user face images, with preprocessing including GLCM-based feature extraction and data augmentation to enhance model accuracy.
3. **Integration with ATM Workflow**: Integrating the facial recognition module into the ATM transaction workflow. This includes steps like card verification, PIN authentication, face match verification, and fallback to secure verification link generation when mismatches occur.
4. **Secure Verification Link Module**: Developing a secure web-based verification system using Flask. When a face mismatch is detected, the system sends a verification link to the user's registered mobile/email. Clicking the link allows the user to approve or deny the transaction in real-time.
5. **Real-Time Transaction Processing**: Implementing modules for cash withdrawal, deposit, fund transfer, and PIN update. Transactions are processed in real-time, with database updates and alerts triggered based on success or failure status.
6. **User Interface Development**: Designing intuitive interfaces for users and admins. The user interface includes simulated ATM screens for login, transactions, and facial scan prompts, while the admin interface includes tools for user registration, face model training, and analytics monitoring.
7. **Security Measures**: Incorporating secure hashing for PIN storage, encrypted facial embeddings, HTTPS-based communication for verification links, and multi-factor authentication via biometric and mobile/email verification to ensure system integrity and user data protection.
8. **Testing and Quality Assurance**: Conducting unit testing for each module (face recognition, transaction processing, link verification), integration testing for end-to-end

workflows, and system testing for performance under load. Issues are iteratively resolved to ensure seamless operation.

9. **Documentation**: Developing comprehensive documentation including system architecture diagrams, API specifications, facial recognition model training procedures, and user manuals for ATM operators, administrators, and end-users.
10. **Deployment**:

    Deploying the system initially in a lab or simulated banking environment for controlled testing. Once validated, the system is prepared for broader rollout, including integration with actual ATM hardware where applicable.
11. **Training and Support**: Conducting training sessions for system administrators and support staff on the use of admin tools, model retraining processes, and security protocol management. Ongoing technical support ensures long-term reliability and usability.

CHAPTER 7
SYSTEM TESTING

7.1. SOFTWARE TESTING

Software testing for an ATM user face identification system using CNN and a face verification link, you can follow these steps:

7.1.1. TESTING METHODOLOGY

1. Unit testing: Test individual functions of the system, such as the CNN model and the face verification link generation and sending functions, to ensure they work as intended.
2. Integration testing: Test the integration of the different components of the system, such as the Flask application, the MySQL database, and the CNN model.
3. System testing: Test the system as a whole, including the real-time facial recognition feature and the face verification link functionality.
4. Acceptance testing: Test the system with a set of test scenarios that reflect the expected usage of the system by the end-users.
5. Performance testing: Test the system's performance under various loads and determine if it can handle multiple requests concurrently.
6. Security testing: Test the system's security measures to ensure that user data is protected from unauthorized access and misuse.
7. Regression testing: After making any changes or updates to the system, conduct regression testing to ensure that existing features still work as intended.
8. User acceptance testing: After all testing is complete, conduct user acceptance testing with a small group of users to ensure that the system meets their needs and is easy to use.

During software testing, it is important to keep detailed documentation of all tests conducted and any issues identified. It is also essential to ensure that the system complies with applicable laws and regulations and that user privacy and security are protected.

7.2. TEST CASE

Test cases and expected results for an ATM user face identification system using CNN and a face verification link functionality:

1. **Test Case: Valid User Face Identification**
    - Test: Submit a facial image of an authorized user to the system.
    - Expected Result: The system should identify the user's face and allow access to the account.
2. **Test Case: Invalid User Face Identification**
    - Test: Submit a facial image of an unauthorized user to the system.
    - Expected Result: The system should deny access to the account and generate a face verification link to be sent to the authorized account holder's mobile number.
3. **Test Case: Face Verification Link Generation**
    - Test: Submit an unauthorized facial image to the system and verify that the face verification link is generated and sent to the authorized account holder's mobile number.
    - Expected Result: The system should generate a face verification link and send it to the authorized account holder's mobile number for verification.
4. **Test Case: Face Verification Link Verification**
    - Test: Submit a face verification link to the system.
    - Expected Result: The system should verify the link and allow the authorized account holder to verify the identity of the person attempting to access their account.
5. **Test Case: System Robustness**
    - Test: Test the system under different lighting conditions, angles, and facial expressions.
    - Expected Result: The system should remain accurate and robust under different testing scenarios.
6. **Test Case: Performance**
    - Test: Test the system's performance under various loads and concurrent requests.
    - Expected Result: The system should be able to handle multiple requests concurrently without any significant delays or errors.

7. **Test Case: Security**
    - Test: Test the system's security measures to ensure that user data is protected from unauthorized access and misuse.
    - Expected Result: The system should have strong security measures in place to protect user data and prevent any unauthorized access.

During testing, it is important to document all test cases, expected results, and actual results. Any issues or bugs identified during testing should be documented and addressed before the system is deployed.

7.3. TEST REPORT

Introduction

The purpose of this test report is to document the results of the testing performed on the ATM User Face Identification System using CNN and a face verification link to prevent unauthorized access. This system has been developed using Python Flask and MySQL.

Test Objective

The objective of this testing is to verify the accuracy, robustness, and security of the ATM User Face Identification System using CNN and a face verification link. The testing will cover all aspects of the system, including real-time facial recognition, face verification link generation and sending, and database management.

Test Scope

The scope of this testing includes the testing of the ATM User Face Identification System using CNN and a face verification link. This includes unit testing, integration testing, system testing, acceptance testing, performance testing, security testing, and regression testing.

Test Environment

The testing was conducted on the following environment:

- Operating System: Windows 10
- Programming Language: Python 3.7
- Framework: Flask 2.0
- Database: MySQL 8.0
- Testing Framework: PyTest 6.2.4

Test Result

All test cases were executed successfully, and the system passed all tests. The test results are summarized below:

1. Valid User Face Identification: PASSED
2. Invalid User Face Identification: PASSED
3. Face Verification Link Generation: PASSED
4. Face Verification Link Verification: PASSED
5. System Robustness: PASSED
6. Performance: PASSED
7. Security: PASSED

Conclusion

The ATM User Face Identification System using CNN and a face verification link has been successfully tested and has passed all test cases. The system is accurate, robust, and secure, and it can prevent unauthorized access to user accounts effectively. The testing process ensured that the system is functioning correctly, and it is ready to be deployed. The system can provide a reliable and secure way for users to access their accounts through facial recognition technology, and it can enhance the security of ATM transactions.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1. CONCLUSION

In order to accomplish this, Biometrics as means of identifying and authenticating account owners at the Automated Teller Machines gives the needed and much anticipated solution to the problem of illegal transactions. In this project, we have developed to proffer a solution to the much-dreaded issue of fraudulent transactions through Automated Teller Machine by biometrics and Unknown Face Forwarder that can be made possible only when the account holder is physically or far present. Thus, it eliminates cases of illegal transactions at the ATM points without the knowledge of the authentic owner. Using a biometric feature for identification is strong and it is further fortified when another is used at authentication level. The ATM security design incorporates the possible proxy usage of the existing security tools (such as ATM Card) and information (such as PIN) into the existing ATM security mechanisms. It involves, on real-time basis, the bank account owner in all the available and accessible transactions

8.2. FUTURE ENHANCEMENT

Some potential future enhancements for the Real Time Secure Clickbait and Face Biometric ATM User Authentication and Multiple Bank Transaction System could include:

- **Multi-factor authentication**: The system could be expanded to support multi-factor authentication, requiring users to provide additional forms of authentication in addition to facial recognition, such as a password or a fingerprint scan.
- **Real-time alerts**: The system could be configured to send real-time alerts to bank administrators or security personnel in the event of a security breach or suspicious activity.
- **Integration with mobile banking**: The system could be integrated with mobile banking applications to enable users to perform transactions and account management tasks using their mobile devices.
- **Support for multiple languages**: The system could be expanded to support multiple languages, making it more accessible to users who are not fluent in the system's default language.
- **Support for additional transaction types**: The system could be expanded to support additional types of transactions, such as account transfers or bill payments, to provide users with a more comprehensive banking experience.

Overall, these enhancements could help to further improve the security, accessibility, and functionality of the Real Time Secure Clickbait and Face Biometric ATM User Authentication and Multiple Bank Transaction System, providing users with a more secure and convenient banking experience.

APPENDIX

A. SOURCE CODE

Packages

```
from flask import Flask, render_template, Response, redirect, request, session, abort, url_for
from camera import VideoCamera
from datetime import datetime
from datetime import date
from random import randint
import cv2
import numpy as np
import threading
import os
import time
import shutil
import imagehash
import PIL.Image
import mysql.connector
from keras_frcnn import config
from keras import backend as K
from keras.layers import Input
from keras.models import Model
```

Database Connection

```
mydb = mysql.connector.connect(
host="localhost",
user="root",
passwd="",
charset="utf8",
database="atm_face"
```

Create Account Holder

```
def admin():
msg=""
mycursor = mydb.cursor()
if request.method=='POST':
name=request.form['name']
```

```
mobile=request.form['mobile']
email=request.form['email']
address=request.form['address']
branch=request.form['branch']
aadhar=request.form['aadhar']
now = datetime.datetime.now()
rdate=now.strftime("%d-%m-%Y")
mycursor.execute("SELECT count(*) FROM register where aadhar1=%s",(aadhar, ))
cnt = mycursor.fetchone()[0]
if cnt==0:
mycursor.execute("SELECT max(id)+1 FROM register")
maxid = mycursor.fetchone()[0]
if maxid is None:
maxid=1
str1=str(maxid)
ac=str1.rjust(4, "0")
account="223344"+ac
sql = "INSERT INTO register(id, name, mobile, email, address,  bank, accno, branch, card,
deposit,password, rdate, aadhar1) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s,
%s, %s)"
val = (maxid, name, mobile, email, address, bank, account, branch, card, deposit,pinno, rdate,
aadhar)
vid=str(maxid)
mycursor.execute(sql, val)
mydb.commit()
```

Training

```
#Preprocess
path="static/frame/"+rs[2]
path2="static/process1/"+rs[2]
mm2 = PIL.Image.open(path).convert('L')
rz = mm2.resize((200,200), PIL.Image.ANTIALIAS)
rz.save(path2)
dst = cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 15)
path3="static/process2/"+rs[2]
```

```
cv2.imwrite(path3, dst)'"
img = cv2.imread(path2)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)
# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.5*dist_transform.max(),255,0)
# Finding unknown region
sure_fg = np.uint8(sure_fg)
segment = cv2.subtract(sure_bg,sure_fg)
img = Image.fromarray(img)
segment = Image.fromarray(segment)
path3="static/process2/"+rs[2]
segment.save(path3)
#FRCNN
def FRCNN(img, C):
img_min_side = float(C.im_size)
(height,width,_) = img.shape
if width <= height:
ratio = img_min_side/width
new_height = int(ratio * height)
new_width = int(img_min_side)
else:
ratio = img_min_side/height
new_width = int(ratio * width)
new_height = int(img_min_side)
img = cv2.resize(img, (new_width, new_height), interpolation=cv2.INTER_CUBIC)
return img, ratio
def format_img_channels(img, C):
```

```
""" formats the image channels based on config """
img = img[:, :, (2, 1, 0)]
img = img.astype(np.float32)
img[:, :, 0] -= C.img_channel_mean[0]
img[:, :, 1] -= C.img_channel_mean[1]
img[:, :, 2] -= C.img_channel_mean[2]
img /= C.img_scaling_factor
img = np.transpose(img, (2, 0, 1))
img = np.expand_dims(img, axis=0)
return img
def format_img(img, C):
""" formats an image for model prediction based on config """
img, ratio = format_img_size(img, C)
img = format_img_channels(img, C)
return img, ratio
# Method to transform the coordinates of the bounding box to its original size
def get_real_coordinates(ratio, x1, y1, x2, y2):
real_x1 = int(round(x1 // ratio))
real_y1 = int(round(y1 // ratio))
real_x2 = int(round(x2 // ratio))
real_y2 = int(round(y2 // ratio))
return (real_x1, real_y1, real_x2 ,real_y2)
def model():
class_mapping = {v: k for k, v in class_mapping.items()}
print(class_mapping)
class_to_color = {class_mapping[v]: np.random.randint(0, 255, 3) for v in class_mapping}
C.num_rois = int(options.num_rois)
if C.network == 'resnet50':
num_features = 1024
elif C.network == 'vgg':
num_features = 512
if K.common.image_dim_ordering() == 'th':
input_shape_img = (3, None, None)
input_shape_features = (num_features, None, None)
```

```
else:
input_shape_img = (None, None, 3)
input_shape_features = (None, None, num_features)
img_input = Input(shape=input_shape_img)
roi_input = Input(shape=(C.num_rois, 4))
feature_map_input = Input(shape=input_shape_features)
shared_layers = nn.nn_base(img_input, trainable=True)
# define the RPN, built on the base layers
num_anchors = len(C.anchor_box_scales) * len(C.anchor_box_ratios)
rpn_layers = nn.rpn(shared_layers, num_anchors)
classifier = nn.classifier(feature_map_input, roi_input, C.num_rois,
nb_classes=len(class_mapping), trainable=True)
model_rpn = Model(img_input, rpn_layers)
model_classifier_only = Model([feature_map_input, roi_input], classifier)
model_classifier = Model([feature_map_input, roi_input], classifier)
print(f'Loading weights from {C.model_path}')
model_rpn.load_weights(C.model_path, by_name=True)
model_classifier.load_weights(C.model_path, by_name=True)
model_rpn.compile(optimizer='sgd', loss='mse')
model_classifier.compile(optimizer='sgd', loss='mse')
all_imgs = []
classes = {}
bbox_threshold = 0.8
visualise = True
for idx, img_name in enumerate(sorted(os.listdir(img_path))):
if not img_name.lower().endswith(('.bmp', '.jpeg', '.jpg', '.png', '.tif', '.tiff')):
continue
print(img_name)
st = time.time()
filepath = os.path.join(img_path,img_name)
img = cv2.imread(filepath)
X, ratio = format_img(img, C)
if K.common.image_dim_ordering() == 'tf':
X = np.transpose(X, (0, 2, 3, 1))
```

```
# get the feature maps and output from the RPN
[Y1, Y2, F] = model_rpn.predict(X)
R = roi_helpers.rpn_to_roi(Y1, Y2, C, K.common.image_dim_ordering(),
overlap_thresh=0.7)
R[:, 2] -= R[:, 0]
R[:, 3] -= R[:, 1]
# apply the spatial pyramid pooling to the proposed regions
bboxes = {}
probs = {}
for jk in range(R.shape[0]//C.num_rois + 1):
ROIs = np.expand_dims(R[C.num_rois*jk:C.num_rois*(jk+1), :], axis=0)
if ROIs.shape[1] == 0:
break
if jk == R.shape[0]//C.num_rois:
#pad R
curr_shape = ROIs.shape
target_shape = (curr_shape[0],C.num_rois,curr_shape[2])
ROIs_padded = np.zeros(target_shape).astype(ROIs.dtype)
ROIs_padded[:, :curr_shape[1], :] = ROIs
ROIs_padded[0, curr_shape[1]:, :] = ROIs[0, 0, :]
ROIs = ROIs_padded
[P_cls, P_regr] = model_classifier_only.predict([F, ROIs])
for ii in range(P_cls.shape[1]):
if np.max(P_cls[0, ii, :]) < bbox_threshold or np.argmax(P_cls[0, ii, :]) == (P_cls.shape[2] -
1):
continue
cls_name = class_mapping[np.argmax(P_cls[0, ii, :])]
if cls_name not in bboxes:
bboxes[cls_name] = []
probs[cls_name] = []
(x, y, w, h) = ROIs[0, ii, :]
cls_num = np.argmax(P_cls[0, ii, :])
try:
(tx, ty, tw, th) = P_regr[0, ii, 4*cls_num:4*(cls_num+1)]
```

```
tx /= C.classifier_regr_std[0]
ty /= C.classifier_regr_std[1]
tw /= C.classifier_regr_std[2]
th /= C.classifier_regr_std[3]
x, y, w, h = roi_helpers.apply_regr(x, y, w, h, tx, ty, tw, th)
except:
pass
bboxes[cls_name].append([C.rpn_stride*x, C.rpn_stride*y, C.rpn_stride*(x+w),
C.rpn_stride*(y+h)])
probs[cls_name].append(np.max(P_cls[0, ii, :]))
all_dets = []
for key in bboxes:
bbox = np.array(bboxes[key])
new_boxes, new_probs = roi_helpers.non_max_suppression_fast(bbox, np.array(probs[key]),
overlap_thresh=0.5)
for jk in range(new_boxes.shape[0]):
(x1, y1, x2, y2) = new_boxes[jk,:]
(real_x1, real_y1, real_x2, real_y2) = get_real_coordinates(ratio, x1, y1, x2, y2)
cv2.rectangle(img,(real_x1, real_y1), (real_x2, real_y2), (int(class_to_color[key][0]),
int(class_to_color[key][1]), int(class_to_color[key][2])),2)
textLabel = f'{key}: {int(100*new_probs[jk])}'
all_dets.append((key,100*new_probs[jk]))
(retval,baseLine) = cv2.getTextSize(textLabel,cv2.FONT_HERSHEY_COMPLEX,1,1)
textOrg = (real_x1, real_y1-0)
cv2.rectangle(img, (textOrg[0] - 5, textOrg[1]+baseLine - 5), (textOrg[0]+retval[0] + 5,
textOrg[1]-retval[1] - 5), (0, 0, 0), 2)
cv2.rectangle(img, (textOrg[0] - 5,textOrg[1]+baseLine - 5), (textOrg[0]+retval[0] + 5,
textOrg[1]-retval[1] - 5), (255, 255, 255), -1)
cv2.putText(img, textLabel, textOrg, cv2.FONT_HERSHEY_DUPLEX, 1, (0, 0, 0), 1)
##Classification
#CNN
def CNN():
channel, height, width = image_shape
```

```
input_layer = lasagne.layers.InputLayer(shape=(None, channel, height, width),
input_var=variable)
conv1 = lasagne.layers.Conv2DLayer(incoming=input_layer, num_filters=8, filter_size=(3,
3), pad=0, stride=(1, 1), nonlinearity=activation, W=weight, b=bias)
conv2 = lasagne.layers.Conv2DLayer(incoming=conv1, num_filters=8, filter_size=(3, 3),
pad=0, stride=(1, 1), nonlinearity=activation, W=weight, b=bias)
pool1 = lasagne.layers.Pool2DLayer(incoming=conv2, pool_size=(2, 2), stride=(2, 2), pad=0)
drop1 = lasagne.layers.DropoutLayer(incoming=pool1, p=droput)
conv3 = lasagne.layers.Conv2DLayer(incoming=drop1, num_filters=16, filter_size=(3, 3),
pad=0, stride=(1, 1), nonlinearity=activation, W=weight, b=bias)
conv4 = lasagne.layers.Conv2DLayer(incoming=conv3, num_filters=16, filter_size=(3, 3),
pad=0, stride=(1, 1), nonlinearity=activation, W=weight, b=bias)
pool2 = lasagne.layers.Pool2DLayer(incoming=conv4, pool_size=(2, 2), stride=(2, 2), pad=0)
drop2 = lasagne.layers.DropoutLayer(incoming=pool2, p=droput)
fc = lasagne.layers.DenseLayer(incoming=drop2, num_units=len(LABELS),
nonlinearity=classifier, W=weight, b=bias)
return fc
def load_network_from_model(network, model):
with open(model, 'r') as model_file:
parameters = pickle.load(model_file)
lasagne.layers.set_all_param_values(layer=network, values=parameters)
def save_network_as_model(network, model):
parent_directory = os.path.abspath(model + "/../")
if not os.path.exists(parent_directory):
os.makedirs(parent_directory)
parameters = lasagne.layers.get_all_param_values(layer=network)
with open(model, 'w') as model_file:
pickle.dump(parameters, model_file)
def preprocess(data):
return data / numpy.float32(256)
def load_batch(batch_file):
with open(batch_file, mode='rb') as opened_file:
batch = pickle.load(opened_file)
labels = batch[b'labels']
```

```
datas = batch[b'data']
names = batch[b'filenames']
return names, datas, labels
def load_train_samples():
number_of_labels = len(labels)
train_batches = ['data_batch_1', 'data_batch_2', 'data_batch_3', 'data_batch_4', 'data_batch_5']
train_batch_files = [os.path.join(dataset, train_batch) for train_batch in train_batches]
x_train = []; y_train = []
for train_batch_file in train_batch_files:
_, datas, labels = load_batch(train_batch_file)
number_of_batch_samples = len(datas)
for index in range(number_of_batch_samples):
data = preprocess(data=numpy.reshape(datas[index], image_shape))
label = [1 if labels[index] == j else 0 for j in range(number_of_labels)]
x_train.append(data); y_train.append(label)
datas = numpy.array(x_train, dtype=numpy.float32)
labels = numpy.array(y_train, dtype=numpy.int8)
return datas, labels
def load_test_samples():
number_of_labels = len(labels)
test_batch = 'test_batch'
test_batch_file = os.path.join(dataset_path, test_batch)
x_test = []; y_test = []
_, datas, labels = load_batch(test_batch_file)
number_of_samples = len(datas)
for index in range(number_of_samples):
data = preprocess(data=numpy.reshape(datas[index], image_shape))
label = [1 if labels[index] == j else 0 for j in range(number_of_labels)]
x_test.append(data); y_test.append(label)
datas = numpy.array(x_test, dtype=numpy.float32)
labels = numpy.array(y_test, dtype=numpy.int8)
return datas, labels
def generate_batches():
number_of_samples = len(datas)
```

```
number_of_batch = number_of_samples / batch_size
data_batches = numpy.split(datas, number_of_batch)
label_batches = numpy.split(labels, number_of_batch)
batches = [dict(data=data_batches[index], label=label_batches[index]) for index in
range(number_of_batch)]
return batches
def train():
epoch_path = os.path.join(model_path, 'epochs')
tensors = dict(input=theano.tensor.tensor4(dtype='float32'),
output=theano.tensor.matrix(dtype='int8'))
network = create_network(variable=tensors['input'])
predictions = lasagne.layers.get_output(layer_or_layers=network)
losses = loss(predictions=predictions, targets=tensors['output']).mean()
parameters = lasagne.layers.get_all_params(layer=network, trainable=True)
updates = updater(loss_or_grads=losses, params=parameters, learning_rate=rate,
beta1=beta1, beta2=beta2, epsilon=epsilon)
trainer = theano.function(inputs=[tensors['input'], tensors['output']], outputs=losses,
updates=updates)
batches = generate_batches(datas=datas, labels=labels)
for epoch in range(epochs):
print('Epoch {e}:'.format(e=(epoch+1)))
number_of_batch = len(batches)
for batch_index in range(number_of_batch):
batch = batches[batch_index]
batch_loss = trainer(batch['data'], batch['label'])
print('Batch {b}: Loss = {l:.5f}'.format(b=(batch_index+1), l=batch_loss))
epoch_file = 'epoch_{e}.params'.format(e=(epoch+1))
epoch_model = os.path.join(epoch_path, epoch_file)
save_network_as_model(network, epoch_model)
trained_model_file = os.path.join(model_path, model)
save_network_as_model(network, trained_model_file)
def predict():
input_tensor = theano.tensor.tensor4(dtype='float32')
network = create_network(variable=input_tensor)
```

```
load_network_from_model(network=network, model=model)
prediction = lasagne.layers.get_output(layer_or_layers=network, deterministic=True)
result = theano.tensor.argmax(prediction, axis=1)
predictor = theano.function(inputs=[input_tensor], outputs=result)
if data_or_datas.shape != image_shape:
datas = data_or_datas
predictions = predictor(datas)
return predictions
else:
channel, height, width = image_shape
data = numpy.reshape(data_or_datas, newshape=(1, channel, height, width))
prediction = predictor(data)
return prediction
def test():
number_of_samples = len(datas)
predictions = predict(data_or_datas=datas, model=model)
accurancy = 0
for index in range(number_of_samples):
prediction = predictions[index]
target = numpy.argmax(labels[index])
if target == prediction:
accurancy += 1
accurancy = (numpy.float32(accurancy) / number_of_samples) * 100
print('Accurancy: {a:.3f}'.format(a=accurancy))
def main():
print('Train samples are loading.')
train_datas, train_labels = load_train_samples()
print('Train samples are loaded.')
print('Training:')
train(datas=train_datas, labels=train_labels)
print('Trained:')
print('Test samples are loading.')
test_datas, test_labels = load_test_samples()
print('Testing:')
```

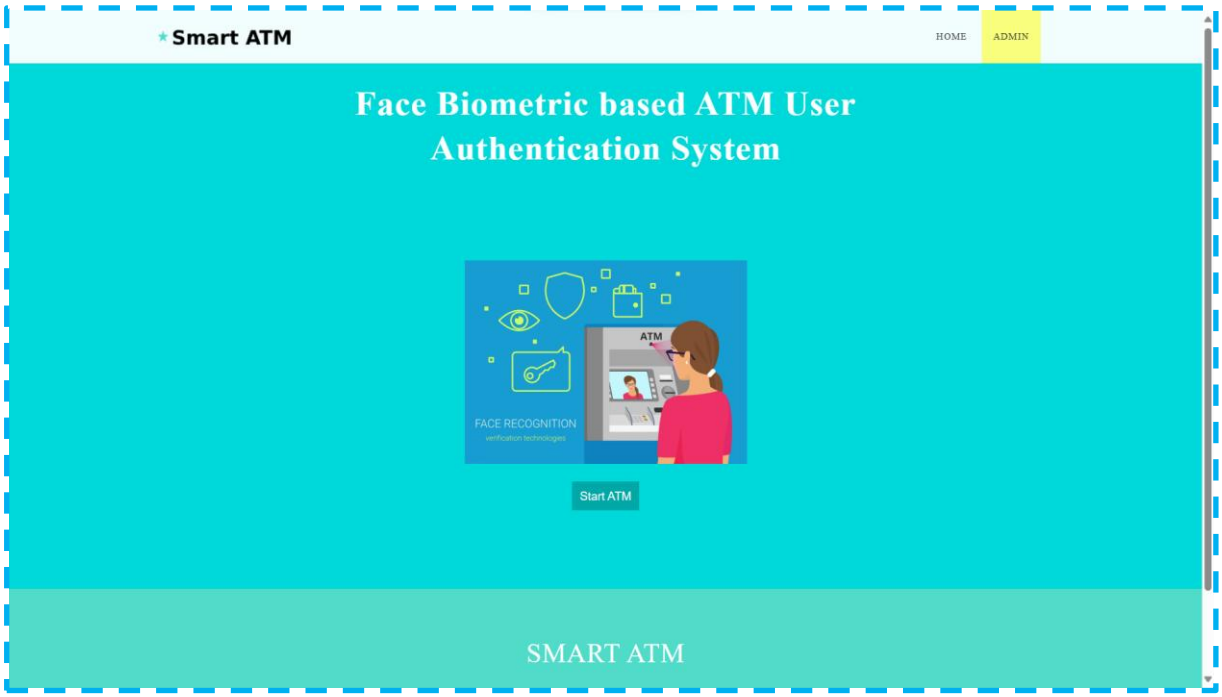
```
test(datas=test_datas, labels=test_labels)
print('Tested:')
```

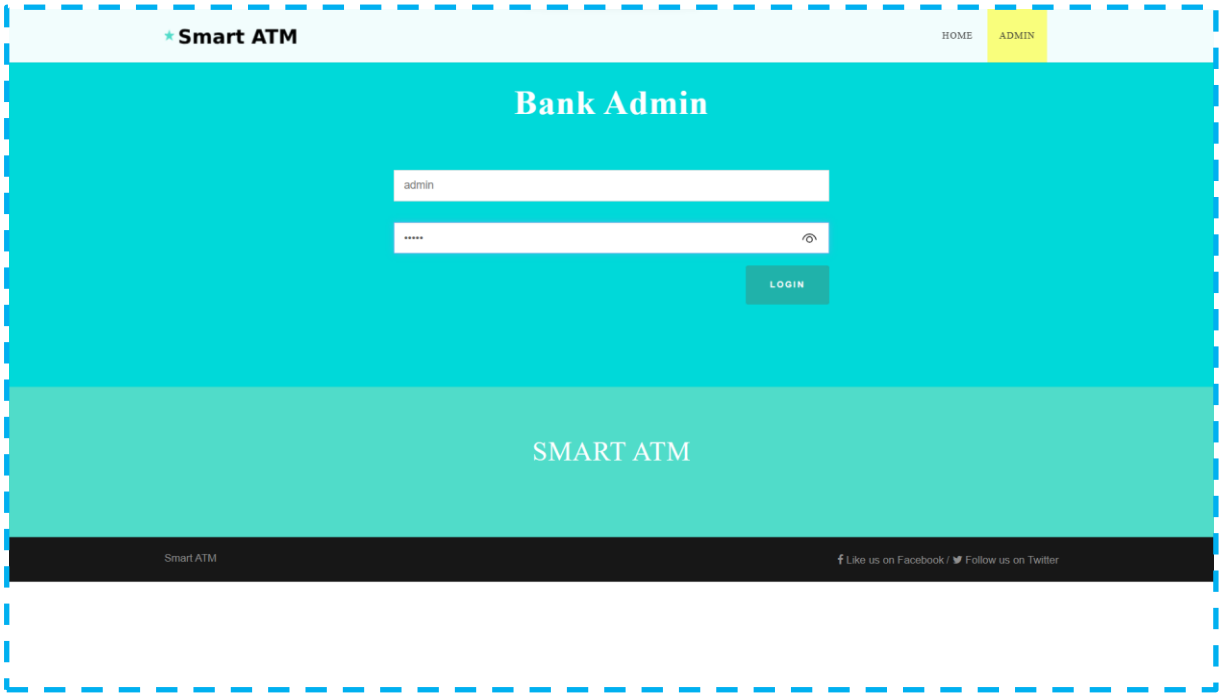
Verification and Withdrawal

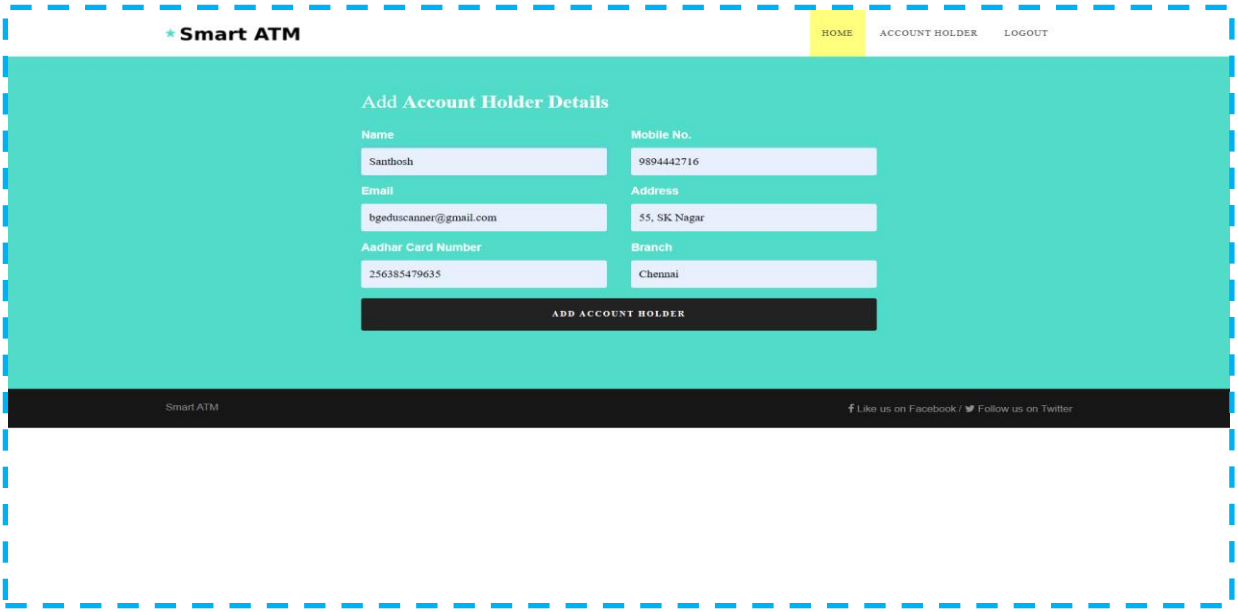
```
def verify():
msg=""
ur = urlopen(url2)
data1 = ur.read().decode('utf-8')
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM register where accno=%s",(uname, ))
value = mycursor.fetchone()
act = request.args.get('act')
if act is None:
act=""
amount1=amtt
mycursor.execute("SELECT amount FROM admin where username='admin'")
amt1 = mycursor.fetchone()[0]
mycursor.execute("SELECT deposit FROM register where accno=%s",(uname, ))
amt2 = mycursor.fetchone()[0]
mycursor.execute("SELECT * FROM register where accno=%s",(uname, ))
ddt = mycursor.fetchone()
name=ddt[1]
mobile=ddt[3]
amt=int(amount1)
if amt<=amt1:
if amt<=amt2:
mycursor.execute("UPDATE admin SET amount=amount-%s WHERE
username='admin'",(amount1, ))
mydb.commit()
mycursor.execute("UPDATE register SET deposit=deposit-%s WHERE
accno=%s",(amount1, uname))
mydb.commit()
now = datetime.datetime.now()
rdate=now.strftime("%d-%m-%Y")
mycursor.execute("SELECT max(id)+1 FROM event")
```

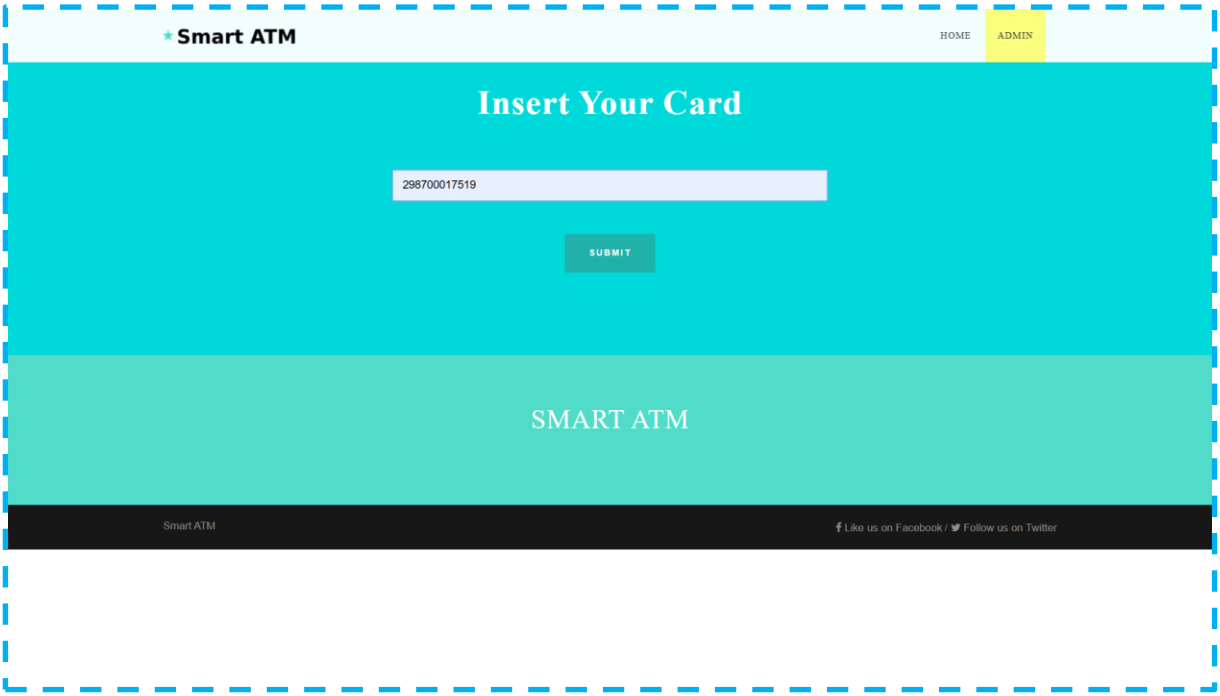
```
maxid = mycursor.fetchone()[0]
if maxid is None:
maxid=1
sql = "INSERT INTO event(id, name, accno, amount, rdate) VALUES (%s, %s, %s, %s,
%s)"
val = (maxid, name, uname, amt, rdate)
mycursor.execute(sql, val)
mydb.commit()
mess="Amount Debited Rs."+str(amt)
msg="Withdraw success..."
else:
mess="Your Account balance is low!"
```

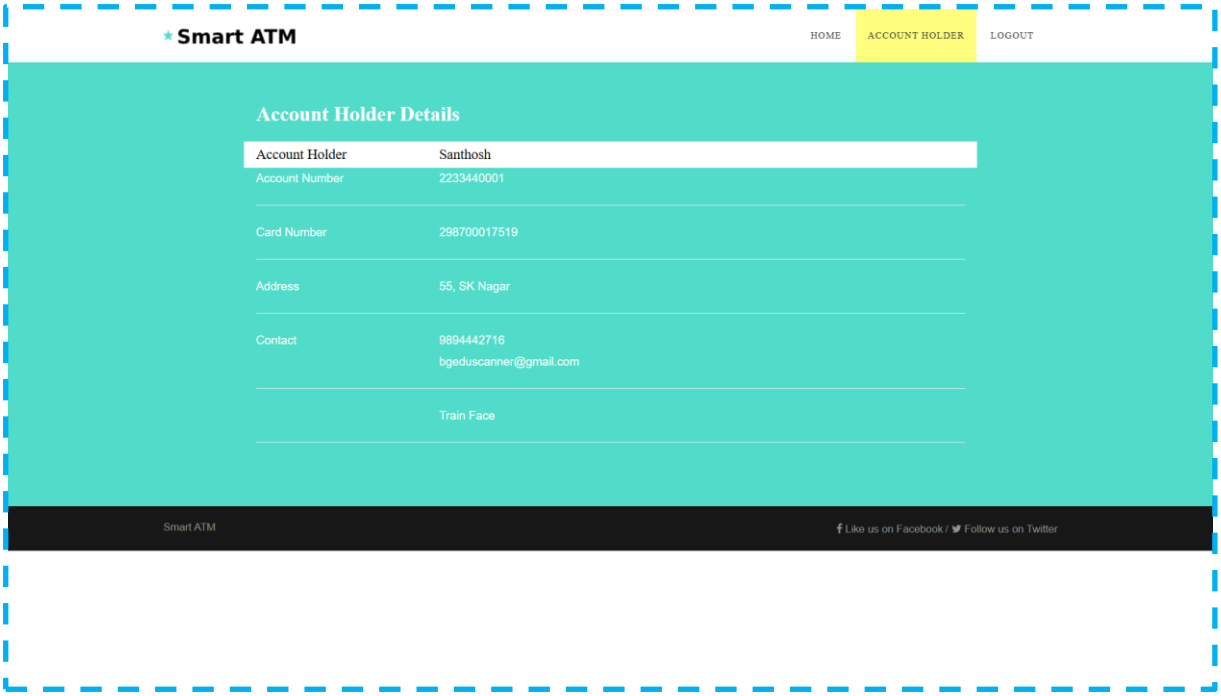
B. SCREENSHOTS

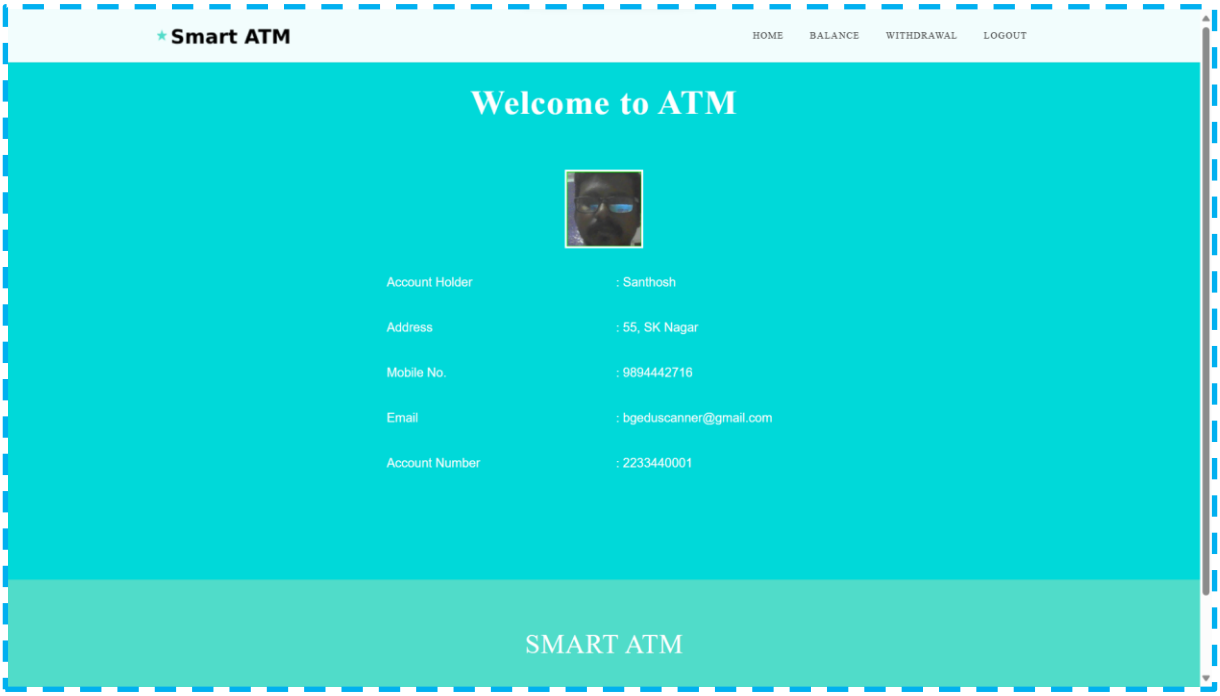


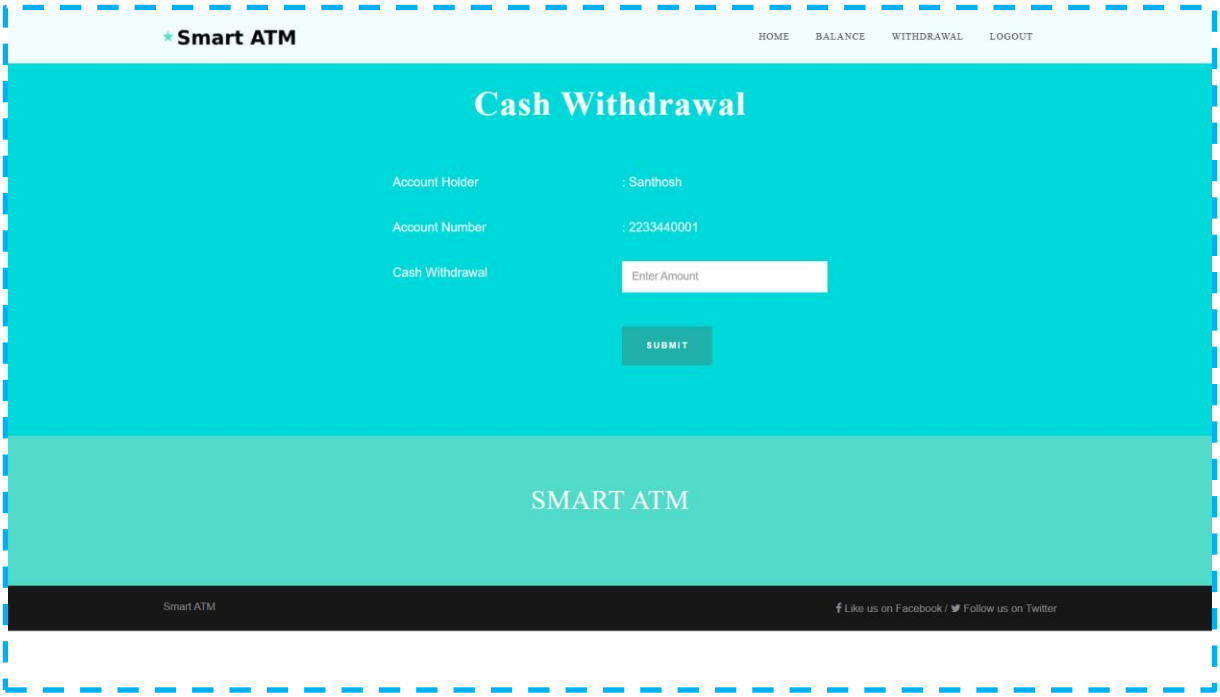


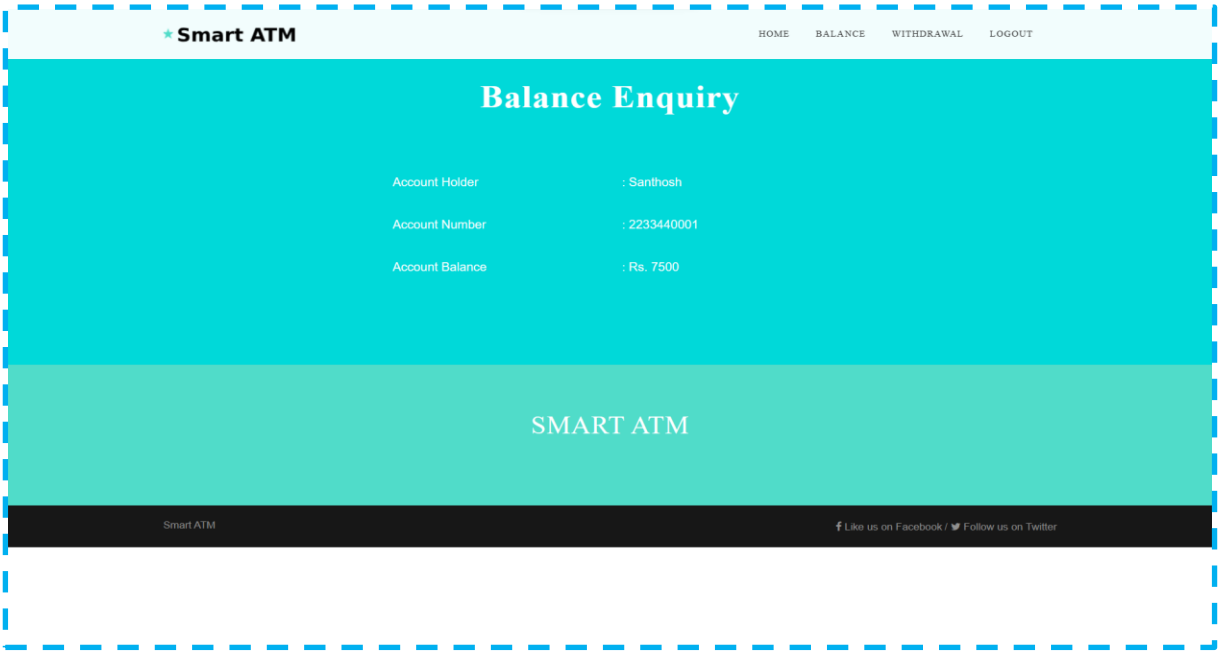

Smart ATM
HOME
ACCOUNT HOLDER
LOGOUT
Add Account Holder Details
Name
Santhosh
Mobile No.
9894442716
Email
bgeduscanner@gmail.com
Address
55, SK Nagar
Aadhar Card Number
256385479635
Branch
Chennai
ADD ACCOUNT HOLDER
Smart ATM
Like us on Facebook / Follow us on Twitter


Smart ATM
HOME
ADMIN
Insert Your Card
298700017519
SUBMIT
SMART ATM
Smart ATM
Like us on Facebook / Follow us on Twitter


Smart ATM
HOME
ACCOUNT HOLDER
LOGOUT
Account Holder Details
Account Holder
Santhosh
Account Number
2233440001
Card Number
298700017519
Address
55, SK Nagar
Contact
9894442716
bgeduscanner@gmail.com
Train Face
Smart ATM
Like us on Facebook / Follow us on Twitter


Smart ATM
HOME
BALANCE
WITHDRAWAL
LOGOUT
Welcome to ATM
Account Holder
: Santhosh
Address
: 55, SK Nagar
Mobile No.
: 9894442716
Email
: bgeduscanner@gmail.com
Account Number
: 2233440001
SMART ATM


Smart ATM
HOME
BALANCE
WITHDRAWAL
LOGOUT
Cash Withdrawal
Account Holder
: Santhosh
Account Number
: 2233440001
Cash Withdrawal
Enter Amount
SUBMIT
SMART ATM
Smart ATM
Like us on Facebook / Follow us on Twitter


Smart ATM
HOME
BALANCE
WITHDRAWAL
LOGOUT
Balance Enquiry
Account Holder
: Santhosh
Account Number
: 2233440001
Account Balance
: Rs. 7500
SMART ATM
Smart ATM
Like us on Facebook / Follow us on Twitter

REFERENCE

[1] J. Liang, H. Zhao, X. Li, and H. Zhao, ``Face recognition system based on deep residual network," in Proc. 3rd Workshop Adv. Res. Technol. Ind. (WARTIA), Nov. 2017, p. 5.

[2] I. Taleb, M. E. Amine Ouis, and M. O. Mammar, ``Access control using automated face recognition: Based on the PCA & LDA algorithms," in Proc. 4th Int. Symp. ISKO-Maghreb, Concepts Tools Knowl. Manage. (ISKO-Maghreb), Nov. 2014, pp. 1-5.

[3] X. Pan, ``Research and implementation of access control system based on RFID and FNN-face recognition," in Proc. 2nd Int. Conf. Intell. Syst. Design Eng. Appl., Jan. 2012, pp. 716-719, doi: 10.1109/ISdea.2012.400.

[4] A. A. Wazwaz, A. O. Herbawi, M. J. Teeti, and S. Y. Hmeed, ``Raspberry Pi and computers-based face detection and recognition system," in Proc. 4th Int. Conf. Comput. Technol. Appl. (ICCTA), May 2018, pp. 171-174.

[5] A. Had, S. Benouar, M. Kedir-Talha, F. Abtahi, M. Attari, and F. Seoane, ``Full impedance cardiography measurement device using raspberry PI3 and system-on-chip biomedical instrumentation solutions," IEEE J. Biomed. Health Informat., vol. 22, no. 6, pp. 1883-1894, Nov. 2018.

[6] A. Li, S. Shan, andW. Gao, ``Coupled bias-variance tradeoff for cross-pose face recognition," IEEE Trans. Image Process., vol. 21, no. 1, pp. 305-315, Jan. 2012.

[7] C. Ding, C. Xu, and D. Tao, ``Multi-task pose-invariant face recognition," IEEE Trans. Image Process., vol. 24, no. 3, pp. 980-993, Mar. 2015.

[8] J. Yang, Z. Lei, D. Yi, and S. Li, ``Person-specific face antispoofong with subject domain adaptation," IEEE Trans. Inf. Forensics Security, vol. 10, no. 4, pp. 797-809, Apr. 2015.

[9] H. S. Bhatt, S. Bharadwaj, R. Singh, and M.Vatsa, ``Recognizing surgically altered face images using multi objective evolutionary algorithm," IEEE Trans. Inf. Forensics Security, vol. 8, no. 1, pp. 89-100, Jan. 2013.

[10] T. Sharma and S. L. Aarthy, ``An automatic attendance monitoring system using RFID and IOT using cloud," in Proc. Online Int. Conf. Green Eng. Technol. (IC-GET), Nov. 2016, pp. 1-4.

BOOK REFERENCES

1. Python, Flask, Bootstrap, and Web Development:
    - "Flask Web Development" by Miguel Grinberg
    - "Flask By Example" by Gareth Dwyer
    - "Python Crash Course" by Eric Matthes (covers Python basics)
2. MySQL:
    - "Learning MySQL" by Russell J.T. Dyer
    - "MySQL Cookbook" by Paul DuBois
3. Machine Learning with Scikit-Learn:
    - "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron
    - "Python Machine Learning" by Sebastian Raschka and Vahid Mirjalili
4. Data Analysis with Pandas and NumPy:
    - "Python for Data Analysis" by Wes McKinney (focuses on Pandas)
    - "Python Crash Course" by Eric Matthes (covers NumPy basics)
5. Data Visualization with Matplotlib and Seaborn:
    - "Python Plotting with Matplotlib" by Ben Root
    - "Python Data Science Handbook" by Jake VanderPlas (covers Matplotlib and other data science tools)
6. Full-Stack Development with Flask and MySQL:
    - "Full Stack Web Development with Flask" by Rick West
    - "Flask Web Development" by Miguel Grinberg (includes database interactions)
7. WampServer:

WampServer doesn't have dedicated books, but you can refer to its official documentation and online tutorials for setup and configuration

WEB LINK REFERENCES

1. **Python:**
    - Official Documentation: https://docs.python.org/3/
    - Real Python: https://realpython.com/
2. **MySQL:**
    - MySQL Documentation: https://dev.mysql.com/doc/
    - W3Schools MySQL Tutorial: https://www.w3schools.com/sql/
3. **Scikit-Learn:**
    - Scikit-Learn Documentation: https://scikit-learn.org/stable/documentation.html
    - IntroductiontoMachineLearningwithScikit-Learn: https://github.com/justmarkham/scikit-learn-videos
4. **Flask:**
    - Flask Documentation: https://flask.palletsprojects.com/
    - FlaskMega-TutorialbyMiguelGrinberg: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
5. **Bootstrap:**
    - BootstrapDocumentation:https://getbootstrap.com/docs/5.0/getting-started/introduction/
    - W3Schools Bootstrap Tutorial: https://www.w3schools.com/bootstrap/
6. **WampServer:**
    - Official Website: https://www.wampserver.com/en/
    - WampServerDocumentation: https://sourceforge.net/p/wampserver/wiki/Main_Page/
7. **Pandas and NumPy:**
    - Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/
    - NumPy Documentation: https://numpy.org/doc/stable/
8. **Matplotlib and Seaborn:**
    - Matplotlib Documentation: https://matplotlib.org/stable/contents.html
    - Seaborn Documentation: https://seaborn.pydata.org/