A project report on

ADVANCED ATTENDANCE TRACKING THROUGH FACIAL RECOGNITION TECHNOLOGY

Submitted in partial fulfillment for the award of the degree of

Master of Computer Applications

By

PRANAV SANJAY MANAPURE (23MCA1041)

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November, 2024

ADVANCED
ATTENDANCE TRACKING THROUGH FACIAL RECOGNITION TECHNOLOGY

Submitted in partial fulfillment for the award of the degree of

Master of Computer Applications

by

PRANAV SANJAY MANAPURE (23MCA1041)

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November, 2024

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled "ADVANCED ATTENDANCE TRACKING THROUGH FACIAL RECOGNITION TECHNOLOGY" submitted by me, for the award of the degree of Master of Computer Applications, Vellore Institute of Technology ,Chennai is a record of bonafide work carried out by me under the supervision of Dr. N Prem Snakar.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date:

Signature of the Candidate

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled **“ADVANCED ATTENDANCE TRACKING THROUGH FACIAL RECOGNITION TECHNOLOGY”** is prepared and submitted by **PRANAV SANJAY MANAPURE (23MCA1041)** to VIT Chennai, in partial fulfillment of the requirement for the award of the degree of **Master of Computer Applications,** programme and as a part of PMCA698J - Dissertation I is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr. N. Prem Sankar.

Date:

Signature of the Internal Examiner	Signature of the External Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department,

Master of Computer Applications

Name: **Dr.B.Saleena Professor**

Date:

(Seal of SCOPE)

ABSTRACT

Facial recognition technology is revolutionizing attendance tracking by providing an automated, contactless, and efficient solution for recording presence in real time. This project explores the implementation of an advanced attendance monitoring system that leverages facial recognition to automatically identify and verify individuals based on unique facial features, addressing key challenges of traditional attendance methods. Students or employees need only stand in front of a camera for the system to record their attendance, eliminating the need for identification cards or manual sign-ins.

This system is designed with three core components: an optimized dataset, a robust recognition algorithm, and an efficient architectural framework. The project utilizes either the CelebA or IMDB-WIKI dataset for training, while employing advanced models such as FaceNet with an Inception module for feature extraction and K-Nearest Neighbors (KNN) for classification. FaceNet's innovative triplet loss function refines the model's accuracy by minimizing intra-class variation, while KNN offers a contactless solution that balances simplicity and efficiency. In addition, Haar Cascade classifiers enhance initial face detection, ensuring the system's effectiveness even in low-resolution, varied lighting conditions, and with moving faces.

The proposed model also addresses common research gaps, such as difficulty in crowded environments, implementation challenges with low-resolution images, and handling various angles and lighting. With these considerations, our model aims to achieve accurate, real-time recognition, using an integrated frontend application with Firebase as a real-time database for robust tracking and storage.

This thesis concludes with the development of a real-time application to demonstrate the practical viability of this attendance monitoring system. By combining deep learning and cloud-based data management, this system is positioned as a secure, scalable, and user-friendly solution for attendance tracking in educational and professional environments, enhancing productivity and security while mitigating attendance fraud.

ACKNOWLEDGEMENT

It is my pleasure to express with a deep sense of gratitude to Dr. N. Prem Sankar, Assistant Professor (Sr-I), SCOPE, Vellore Institute of Technology, Chennai, for his constant guidance, continual encouragement, and understanding; more than all, he taught me patience in my endeavor. My association with him is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Machine Learning.

It is with gratitude that I would like to extend thanks to our honorable Chancellor, Dr. G. Viswanathan, Vice Presidents, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan and Dr. G V Selvam, Executive Director Dr. Sandhya Pentareddy, Assistant Vice-President, Ms. Kadhambari S. Viswanathan, Vice-Chancellor, Dr. V. S. Kanchana Bhaaskaran, Pro-Vice Chancellor Dr.T. Thyagarajan and Additional Registrar, Dr. P.K.Manoharan for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dean, Dr.R.Ganesan, Associate Dean Academics, Dr.R. Parvathi and Associate Dean Research, Dr.S.Geetha, SCOPE, Vellore Institute of Technology, Chennai, for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In a jubilant mood, I express ingeniously my whole-hearted thanks to Dr.B.Saleena, Head of the Department and Dr.B.Prakash, Project Coordinator , SCOPE, Vellore Institute of Technology, Chennai, for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculty member and staffs at Vellore Institute of Technology, Chennai, who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: Name of the student

CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 INTRODUCTION

Facial recognition technology has swiftly become one of the most groundbreaking advancements in artificial intelligence (AI) and computer vision. Its precision in identifying individuals through distinct facial characteristics has revolutionized various industries, including security, law enforcement, customer service, and targeted marketing. This technology has proven especially beneficial for attendance tracking systems, providing an automated and efficient solution that overcomes the limitations of traditional attendance methods, such as manual registers or RFID card systems. These older methods are often vulnerable to human error, fraudulent practices, and administrative inefficiencies, making the need for a more reliable and seamless solution essential.

Facial recognition works by identifying a face in an image or video, extracting unique features, and comparing them with a stored database to find a match. This innovative approach, powered by advanced algorithms and machine learning, has enabled various applications like unlocking mobile devices, tracking attendance, and managing security access. The versatility of this technology has resulted in its widespread adoption in daily-use devices, ensuring convenience and ease of use. For example, facial recognition is now a common feature for unlocking smartphones, as shown in Figure (1), where a person uses facial recognition technology on their mobile device.

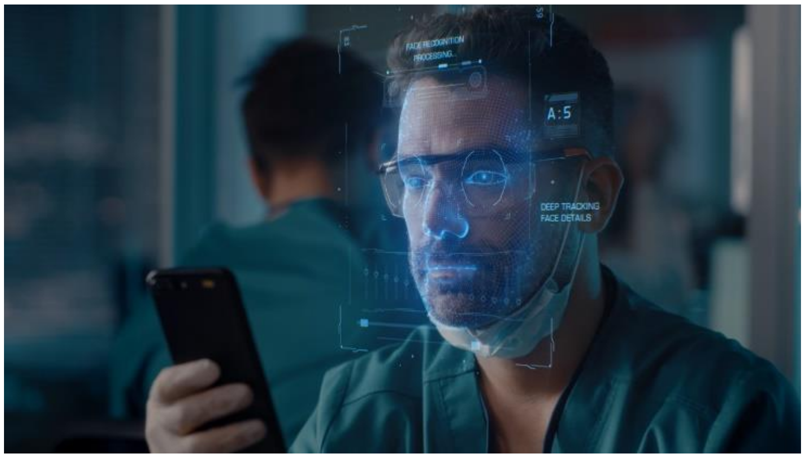


Figure 1.1: User Attempting Face Recognition Unlock on Mobile Device

A key advancement of facial recognition technology is its application in smart attendance monitoring systems. These systems utilize facial recognition algorithms to detect and identify faces automatically, capturing essential data such as date, time, and location. This approach allows institutions and organizations to streamline attendance tracking processes by eliminating manual entry and minimizing errors. Figure (2) illustrates a person scanning their face to mark their attendance in a classroom setting, showcasing how facial recognition technology is transforming attendance systems into a seamless and touchless process.

Figure 1.1.2: Face Recognition System for Automated Attendance

The integration of facial recognition into an attendance system has several advantages, including enhanced security, reduced administrative tasks, and increased accuracy. By requiring physical presence for attendance, the technology significantly reduces instances of proxy attendance or attendance fraud, creating a more secure and efficient process. Such systems are particularly beneficial in educational institutions, where monitoring the attendance of large student groups can be challenging. Figure (3) depicts a situation where a single camera is utilized to capture and recognize all the students in a classroom, efficiently recording their attendance through facial recognition technology.

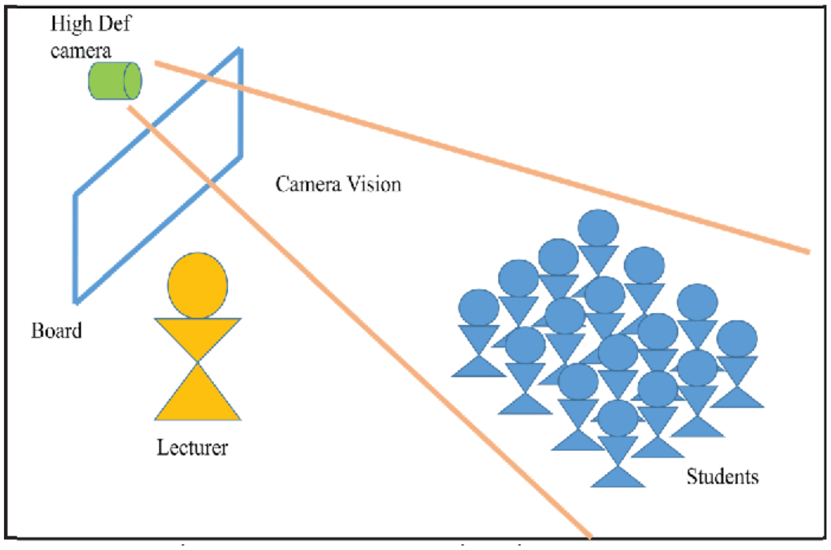


Figure 1.1.3: Single Camera System Detecting Multiple Faces for Attendance

This project explores the potential of facial recognition as a robust solution for attendance monitoring, utilizing cutting-edge algorithms like FaceNet and K-Nearest Neighbors (KNN). Recent developments in deep learning, such as FaceNet's triplet loss function for precise feature extraction, have improved the accuracy and reliability of facial recognition systems, making them suitable for real-time applications. The system is further enhanced by integrating it with Firebase, a real-time database, to provide instant updates and seamless record-keeping. Through this approach, the project aims to deliver a scalable, efficient, and secure attendance tracking solution suitable for various organizational needs.

1.2 BACKGROUND OF FACIAL RECOGNITION TECHNOLOGY

Facial recognition technology is a subset of biometric identification that relies on analyzing and recognizing a person's unique facial features. This technology identifies a face within an image or video, analyzes its distinct features, and compares them with stored data to confirm a person's identity. While it was initially designed for security and surveillance purposes, facial recognition has since evolved to serve various consumer applications, such as unlocking smartphones, targeted marketing, and automated attendance tracking systems. The technology combines computer vision, image processing, and machine learning algorithms to achieve high accuracy and adaptability, even in real-time applications. With advancements like FaceNet, facial recognition systems can now distinguish subtle facial features with precision, paving the way for accurate, touchless attendance tracking solutions.

1.3 PROBLEM STATEMENT

Traditional attendance tracking methods, such as manual registers and RFID cards, are often susceptible to issues like human error, proxy attendance, and administrative burden. These methods not only compromise the accuracy and integrity of attendance records but also consume significant time and resources, especially in large organizations. In educational institutions and businesses, accurately monitoring attendance is critical for evaluating participation, productivity, and accountability. The problem lies in developing an attendance system that can overcome these limitations while ensuring security, efficiency, and real-time processing.

1.4 OBJECTIVES OF THE PROJECT

The primary objective of this project is to create an efficient, reliable, and secure attendance tracking system using facial recognition technology. Specific goals include:

- Building a facial recognition model that can identify individuals in varied lighting conditions, angles, and low-resolution images.
- Implementing real-time recognition to accommodate dynamic environments, such as classrooms or workplaces.

- Integrating Firebase as a real-time database to store and update attendance records instantly, making it accessible and scalable for larger applications.

1.5 SCOPE OF THE PROJECT

This project focuses on developing a facial recognition-based attendance system suitable for educational and organizational environments. The system is designed to be versatile, working with both single and multiple faces in real-time. By incorporating a Firebase real-time database, the project scope extends beyond simple identification to provide instantaneous updates on attendance records. The system's modularity and integration with front-end applications, such as web or mobile interfaces, further enhance its usability and scalability across diverse sectors.

1.6 IMPORTANCE OF THE STUDY

This study addresses the growing need for efficient, automated attendance systems that maintain accuracy while minimizing manual intervention. By eliminating the potential for errors and fraud associated with traditional methods, facial recognition systems provide a secure alternative. The study also contributes to the broader field of AI and biometric technology by demonstrating how facial recognition can be implemented in a real-world, practical application. Additionally, the use of Firebase for real-time data storage highlights the importance of cloud-based solutions in modern applications.

1.7 CHALLENGES IN TRADITIONAL ATTENDANCE SYSTEMS

Traditional attendance methods, such as paper-based registers or RFID cards, pose several challenges. These methods are often time-consuming, requiring manual entry and verification, and they leave room for errors, intentional or accidental proxy attendance, and lost records. In larger settings, like universities or corporations, the logistical challenges of managing accurate attendance records for hundreds of individuals further amplify these issues. The lack of automation not only affects productivity but also decreases the accuracy and reliability of attendance records.

1.8 ADVANCEMENTS IN ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING FOR ATTENDANCE SYSTEMS

Machine Learning (ML) and Artificial Intelligence (AI) have transformed attendance management systems by offering automated, precise, and efficient solutions. Advanced deep learning frameworks like FaceNet significantly enhance the capability of systems to recognize distinct facial characteristics, even under challenging conditions. Similarly, machine learning methods, including K-Nearest Neighbors (KNN), improve classification accuracy, ensuring real-time recognition is both effective and dependable. Moreover, cloud computing and real-time databases, such as Firebase, allow attendance systems to provide instantaneous updates, making them accessible across multiple platforms. These advancements not only streamline attendance tracking but also ensure security and data integrity, setting a new standard for attendance management systems.

1.9 OVERVIEW OF TECHNOLOGIES AND TOOLS USED

The development of a reliable facial recognition-based attendance system involves several advanced technologies and tools. By leveraging these technologies, this project can deliver high accuracy, speed, and real-time updates, which are essential for a robust attendance management system. At the core of this system are machine learning and deep learning frameworks that handle the processing of facial images, the training of algorithms for recognition, and real-time data storage and retrieval. These components work in synergy to provide an end-to-end solution that is both scalable and adaptable to different environments, such as educational institutions and workplaces.

1.9.1 DEEP LEARNING

Deep learning is a subset of artificial intelligence (AI) that uses neural networks with multiple layers to extract and analyze complex patterns in data. In facial recognition, deep learning techniques are fundamental as they allow for the extraction of unique facial features from images, which are then used to distinguish individuals accurately. Deep learning techniques, including convolutional neural networks (CNNs), have demonstrated exceptional performance in analyzing images and videos. When trained on extensive datasets, these models are capable of accurately recognizing faces, even under difficult conditions like different lighting, angles, or facial expressions.

1.9.2 FACIAL RECOGNITION ALGORITHMS

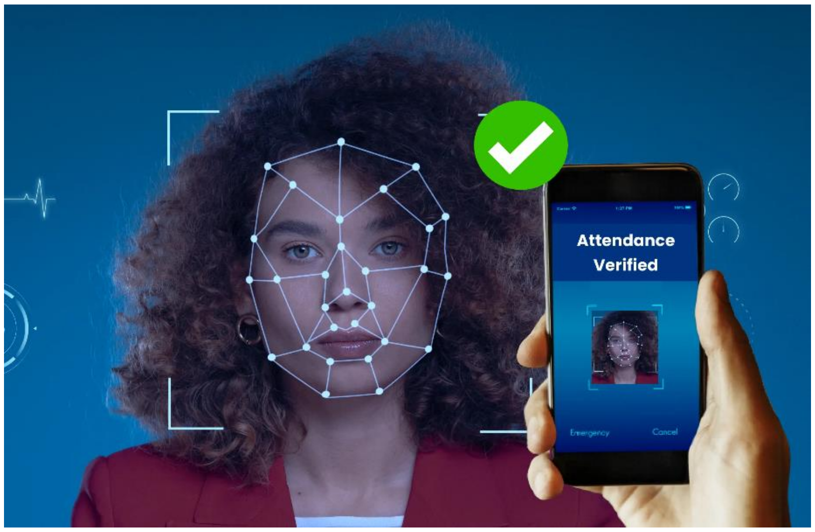


Figure 1.9.2: Model Processing Facial Features for Recognition

Facial recognition algorithms play a critical role in analyzing and matching facial features for identification. Various algorithms, including deep learning-based models and traditional machine learning techniques, contribute to different aspects of the process. For instance, models like FaceNet (described in 1.9.3) utilize a triplet loss function to enhance feature extraction, making facial embeddings unique for each individual. Similarly, Haar Cascade Classifier is a popular tool in initial face detection steps, effectively identifying the location of faces within images or video frames. These algorithms, together, support real-time and high-accuracy identification necessary for attendance systems.

1.9.3 FACENET

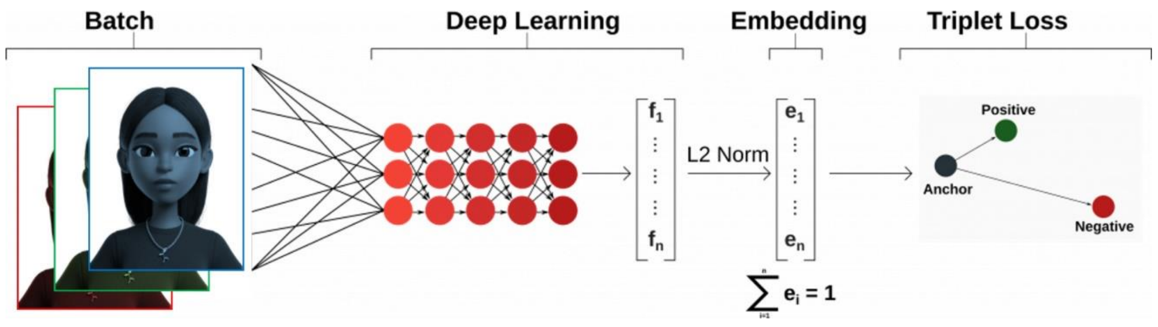


Figure 1.9.3: FaceNet Model Processing Facial Image and Extracting Features

FaceNet, a cutting-edge deep learning model created by Google, is specifically tailored for facial recognition tasks. It converts facial images into a compact embedding space where the distances represent facial similarity. This model is highly effective for building reliable face recognition systems, as it can accurately differentiate between similar faces. By employing the triplet loss function, FaceNet reduces the distance between images of the same individual while increasing the distance for different people, making it an excellent foundation for dependable attendance tracking in settings with large populations.

FaceNet's architecture is based on the Inception network, which enables it to perform complex feature extraction with high efficiency and minimal computational resources. Unlike traditional facial recognition models that rely on predefined templates or multiple feature comparisons, FaceNet creates a unique embedding for each face, reducing the need for storage and increasing processing speed. This streamlined approach is especially beneficial for real-time applications, where quick and accurate identification is essential. By embedding faces into a high-dimensional space where distances represent similarity, FaceNet can be seamlessly integrated into attendance systems, handling large datasets and varied environments with ease.

1.9.4 HAAR CASCADE CLASSIFIER

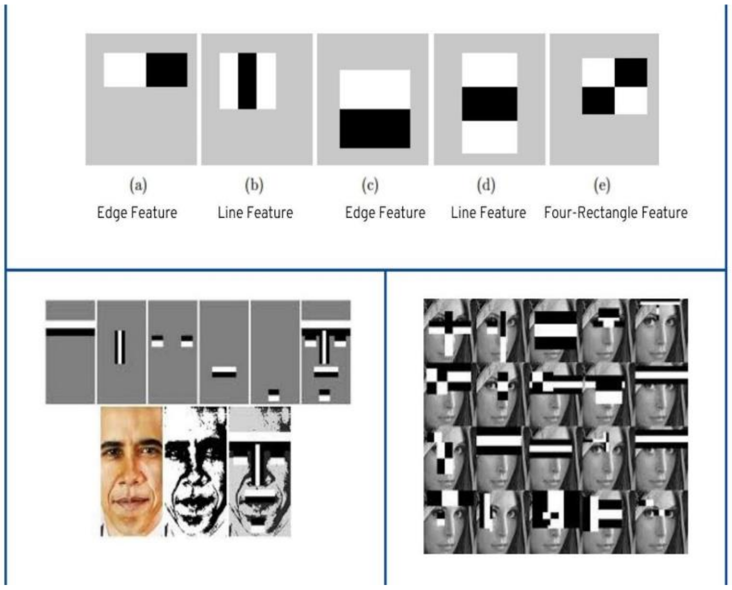


Figure 1.9.4: Haar Cascade Detecting Edges and Features in an Image

The Haar Cascade Classifier is a machine learning technique widely utilized for detecting objects in both images and videos. In facial recognition systems, it is typically employed as a preliminary step to detect faces before applying more complex algorithms. Haar-like features, such as edge, line, and four-rectangle features, help identify the contours of a face, making it a highly efficient and fast approach for face detection. While not as precise as deep learning models for recognition, Haar Cascade plays a critical role in identifying facial regions within an image.

The Haar Cascade Classifier operates by analyzing Haar-like features through a series of stages, each designed to filter out non-facial regions in an image with increasing precision. It uses a cascade of weak classifiers that rapidly eliminates non-face areas, allowing the algorithm to focus on likely facial regions, significantly speeding up the detection process. Due to its lightweight computational requirements, Haar Cascade is ideal for embedded systems and real-time applications where processing power may be limited. This makes it particularly valuable in attendance systems, where quick face detection is necessary to initiate the recognition process accurately, creating a streamlined pipeline that efficiently balances speed and accuracy.

1.9.5 K-NEAREST NEIGHBORS (KNN)

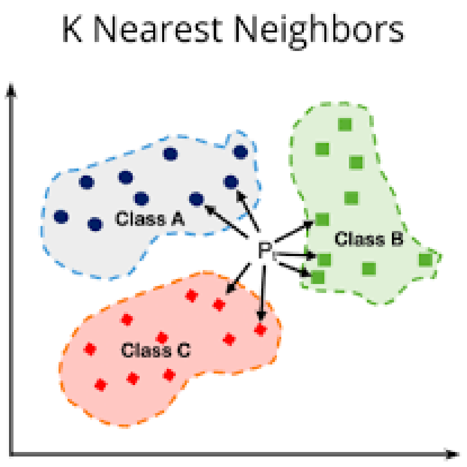


Figure 1.9.5: KNN Decision Boundary Based on Nearest Neighbors

The K-Nearest Neighbors (KNN) algorithm is a supervised learning technique that can be applied in facial recognition for classification and verification. Once a face has been detected and relevant features extracted, KNN classifies the face based on the "distance" to known faces in the dataset. KNN is valuable for attendance systems as it's computationally efficient, particularly for smaller datasets, and can complement deep learning models by providing quick, instance-based recognition.

1.9.6 REAL-TIME DATABASES (FIREBASE)

For real-time attendance tracking, a reliable database is essential, and Firebase is a popular choice for its real-time data synchronization capabilities. Firebase enables instant updates to attendance records across devices, making the system more accessible and efficient. The integration of Firebase also supports scalability and flexibility, allowing multiple users to access data simultaneously. This ensures that attendance records remain up-to-date and can be viewed by authorized personnel in real-time.

1.10 PROJECT SIGNIFICANCE IN REAL-WORLD APPLICATIONS

This project has significant implications for real-world applications across multiple sectors, including education, corporate, and security environments. Automated attendance systems based on facial recognition offer an efficient, secure, and user-friendly alternative to traditional attendance tracking methods. For educational institutions, this system can help streamline attendance management, reduce administrative workload, and enhance accountability among students. In corporate settings, automated attendance tracking can improve productivity by reducing time spent on attendance verification and minimizing instances of proxy attendance. Additionally, by ensuring that only authorized individuals are marked as present, facial recognition enhances security and prevents unauthorized access.

In sectors requiring secure access control, such as government buildings and healthcare facilities, the integration of facial recognition further promotes security and operational efficiency. Real-time databases like Firebase support scalability and flexibility, enabling organizations of any size to maintain accurate, up-to-date records with ease. Overall, this project contributes to the advancement of attendance tracking by integrating AI-driven facial recognition and cloud technology for a modern, practical, and scalable solution.

Chapter 2

Literature Survey

2.1 INTRODUCTION

The advancement of facial recognition technology has revolutionized attendance systems, shifting from manual sign-in processes to automated and contactless solutions. Progress in artificial intelligence, computer vision, and deep learning has significantly enhanced the accuracy and efficiency of these systems in practical use. This chapter explores key studies from 2017 to the present, focusing on research in three primary domains: Image Processing (IP), Machine Learning (ML), and Deep Learning (DL). Each study is categorized by its primary focus on specific algorithms within these domains. For example, if a study primarily employs machine learning techniques, it is categorized under ML. This survey offers insights into the various approaches, technologies, and methodologies employed across recent literature, examining both successes and challenges in developing robust facial recognition systems for attendance management.

2.2 LITERATURE SURVEY

The literature survey is organized into three sections according to the primary domains employed in facial recognition for attendance systems.

2.2.1 IMAGE PROCESSING

The image processing domain provides fundamental techniques for identifying and verifying facial regions within images. This includes systematic reviews and specific applications in attendance systems.

In the study titled "Student attendance with face recognition (LBPH or CNN): Systematic literature review" by Budimana et al. [1], a thorough analysis of 30 research articles was carried out to evaluate the effectiveness of the Local Binary Pattern Histogram (LBPH) and Convolutional Neural Network (CNN) algorithms in student attendance systems. This study, based on PRISMA methodology, highlights essential factors influencing facial recognition accuracy, such as camera quality and varied facial orientations, and demonstrates that CNN generally provides higher accuracy than LBPH in surveillance scenarios.

Face Recognition in Attendance Systems: A Systematic Review: Anshari et al. [2], in their study, conducted a systematic review using the Kitchenham method, often used in research concerning P2P lending and attendance systems. They evaluated the role of facial recognition in diverse organizational contexts, identifying common security methods and success factors for effective implementation. This research highlights the importance of finding solutions that strike a balance between ensuring security and maintaining ease of use.

Face Recognition-Based Attendance Management System: Smitha, Hegde, and Afshin [5] proposed an automated attendance model that simplifies attendance tracking by storing students' facial images in a database. The system utilizes a Haar Cascade Classifier for detecting faces and LBPH for recognizing them, allowing it to efficiently record attendance, generate reports, and notify faculty of absentees through email.

Facial Recognition Attendance Monitoring Utilizing Deep Learning Techniques: In their study, Manjula et al. [9] utilized Eigenfaces, Fisherfaces, and LBPH approaches to enhance the precision of facial feature extraction through detailed histogram analysis. Their method employs Euclidean distance metrics to compare facial images, offering an efficient and dependable recognition system designed for real-time attendance tracking.

Case Study at MEDIU Using Viola-Jones for Face Detection: In the case study by Huda and Hilles [4], an automated attendance system was developed for Al-Madinah International University, utilizing the Viola-Jones algorithm. This system relies on a tailored dataset (MEDUE-S-V-DB) and various classifiers, enhancing attendance tracking accuracy in an educational setting by utilizing video data with diverse facial attributes.

2.2.2 MACHINE LEARNING

Machine learning algorithms have demonstrated their effectiveness in attendance systems by utilizing classifiers and decision trees to improve the accuracy of recognition.

Feature-Based Algorithms for Face Recognition in Attendance Systems: The research conducted by Gowda et al. [6] explores a machine learning model that utilizes decision tree and support vector machine classifiers for attendance tracking. Through the development of a refined metric space using deep metric learning, the study enhances the classification of facial features, resulting in improved efficiency for both the enrollment and recognition processes.

A Smart Attendance System based on Facial Recognition using Deep Transfer Learning: Alhanaeea et al. [7] explored transfer learning by utilizing pretrained networks, including SqueezeNet and GoogleNet. This approach improved model adaptability to new datasets, highlighting the benefits of data augmentation and fine-tuning, ultimately producing a flexible model for smart attendance systems.

2.2.3 DEEP LEARNING

Deep learning has revolutionized face recognition in attendance systems, with neural networks such as CNN and ResNet providing high accuracy in complex real-world environments.

Single Sample Face Recognition Using CNN: Filippidou and Papakostas [8] applied CNN-based object classifiers (MobileNetV2, ResNet50V2, DenseNet121, and InceptionV3) for single-sample face recognition. This approach supports both in-person and online attendance tracking, demonstrating accuracy rates as high as 100% in ideal scenarios.

Face Recognition for Real-Time Classroom Attendance: Singh et al. [15] developed a real-time attendance system that captures students’ facial images through webcam feeds. Using the Haar Cascade Classifier for face detection and LBPH for recognition, the system automatically marks attendance in an Excel sheet, facilitating seamless management of classroom attendance.

Development of CNN-Based Automatic Class Attendance System: In their study, Chowdhury et al. [13] utilized CNN-based face recognition combined with Histogram of Oriented Gradients (HOG) for accurate face detection and recognition. The system's architecture incorporates data entry, recognition, training, and attendance entry, automating attendance tracking and reducing manual intervention.

Python-Based Automated Attendance System: Trivedi et al. [11] introduced an attendance management system built with Python, Django, and Flask, supported by a MySQL backend. The system enables administrators to upload annotated data, while end users can access attendance information through web-based queries, streamlining attendance tracking and improving accessibility.

Comprehensive Model Using CNN, VGG-19, and ResNet-50: Sah et al. [12] developed a model using multiple architectures—Decision Tree, SVM, CNN, VGG-19, and ResNet-50—to test their effectiveness in face recognition. This approach resulted in CNN achieving the highest accuracy at 96.82%, demonstrating the strength of deep learning methods in attendance systems.

Chapter 3

Existing and Proposed System

3.1 EXISTING SYSTEM

Current attendance tracking systems utilizing facial recognition technology represent a significant advancement over traditional methods, such as manual sign-in sheets or RFID card-based systems. These systems leverage facial recognition algorithms to streamline the process, enhancing both security and convenience by automatically capturing attendance data through live camera feeds. Existing solutions employ a variety of algorithms, including Local Binary Patterns Histograms (LBPH), Eigenfaces, Fisherfaces, and Convolutional Neural Networks (CNNs), each providing different levels of accuracy and performance based on system requirements.

Most existing systems are implemented through software applications on desktops or mobile devices that are linked to a database. These systems typically capture live images or video of individuals as they enter specific areas (e.g., classrooms or workspaces), analyze these images to detect and identify faces, and mark attendance accordingly. Although effective in controlled settings, these systems often face challenges with large groups or real-time processing. In such scenarios, factors like lighting conditions, camera quality, and facial angles play a critical role in determining accuracy, resulting in possible misidentification or missed attendance records.

3.2 DISADVANTAGES OF EXISTING SYSTEM

While existing face recognition-based attendance systems provide convenience and increased automation, they have certain limitations:

1. Sensitivity to Environmental Conditions: Many facial recognition systems struggle

in conditions where lighting is poor, or facial angles are unconventional. Bright light, shadows, or the presence of multiple people in a frame can lead to reduced accuracy in identifying individuals.

2. High Processing and Hardware Requirements: Some systems require substantial computational resources to handle real-time processing, particularly in high-traffic areas. To achieve acceptable performance levels, such systems may need dedicated graphics processing units (GPUs) or high-definition cameras, which can increase operational costs.

3. Difficulty Handling Dynamic or Crowded Environments: Identifying faces accurately in crowded or constantly moving environments remains a significant challenge. Many systems are designed to work best when individuals are relatively stationary, which may not be realistic in busy workplaces or school hallways.

4. Privacy Concerns and Data Security: Given that facial recognition technology inherently involves the collection and storage of personal biometric data, existing systems often face scrutiny over privacy concerns. Without robust data protection mechanisms, there is a risk of unauthorized access or misuse of sensitive information, which is particularly concerning in environments like educational institutions.

5. Limited Flexibility in User Training: In many existing systems, users must enroll by registering their face, often with specific requirements on facial orientation or expressions. This can restrict the system’s ability to recognize faces in various angles or expressions, impacting the user experience and potentially requiring additional training data.

3.3 PROPOSED SYSTEM

The proposed system seeks to overcome the limitations of current attendance systems by creating a highly precise and efficient facial recognition solution that performs effectively under various challenging conditions. This system integrates advanced algorithms with a user-friendly interface designed for real-time performance, making it adaptable to both educational institutions and corporate environments.

1. High-Performance Algorithms: The system utilizes powerful algorithms such as FaceNet and K-Nearest Neighbors (KNN) to enhance the precision of identification. FaceNet uses a deep convolutional network to map facial features into a high-dimensional space, where Euclidean distances between faces are minimized for matched identities. KNN provides efficient classification, allowing the system to quickly recognize enrolled faces even with minimal training data.

2. Multi-Environment Adaptability: Designed to function reliably under varied conditions, the system can handle differences in lighting, resolution, and facial angles. By integrating techniques such as histogram equalization and facial feature normalization, the system maintains accuracy even when the image quality or conditions are less than ideal.

3. Real-Time Processing: The proposed system focuses on optimizing computational requirements to enable real-time attendance marking. By employing techniques like feature extraction and dimensionality reduction, the system minimizes latency, which allows it to process video feeds or live images from multiple cameras without noticeable delays.

4. Enhanced Privacy and Security Measures: Recognizing the sensitivity of biometric data, the proposed system incorporates advanced encryption techniques to protect data both in transit and at rest. The system also provides customizable privacy options, allowing administrators to configure access controls and data retention policies based on their organizational requirements.

3.4 ADVANTAGES OF PROPOSED SYSTEM

The suggested system provides numerous benefits compared to both conventional and existing face recognition-based attendance methods:

1. Improved Accuracy Across Conditions: The integration of advanced algorithms allows for high accuracy in varied lighting, angles, and background conditions. Unlike many existing systems, which may struggle with environmental factors, this system is optimized to operate reliably across a wide range of settings.

2. Scalability for Large Organizations: The proposed system's streamlined architecture and optimized processing make it suitable for large-scale deployment. It can handle attendance tracking for hundreds or thousands of individuals with minimal computational strain, making it ideal for use in large organizations or academic institutions.

3. Cost-Effective and Low Power Requirements: By reducing the system's dependency on high-end hardware, the proposed solution minimizes operational costs. Its efficient processing also reduces power consumption, making it a sustainable choice for institutions looking to implement facial recognition systems on a budget.

4. Enhanced Security and Data Protection: With advanced encryption and secure data storage, the system ensures that biometric data remains protected from unauthorized access. Additionally, its privacy settings allow administrators to control data access and retention, addressing privacy concerns and regulatory requirements.

5. User-Friendly and Flexible Design: The system's interface is intuitive, designed to require minimal training. It supports flexible user enrollment options, allowing individuals to register in various lighting and facial orientation conditions, which

broadens its usability and ensures seamless adoption by users.

3.5 SYSTEM ARCHITECTURE

The architecture of the proposed facial recognition-based attendance tracking system is structured to deliver optimal accuracy, efficiency, and seamless integration with existing infrastructure. The architecture can be divided into multiple stages, each with distinct functionalities that contribute to the overall operation of the system. Figure 8 provides a visual representation of the architecture, outlining each component and its role within the system.

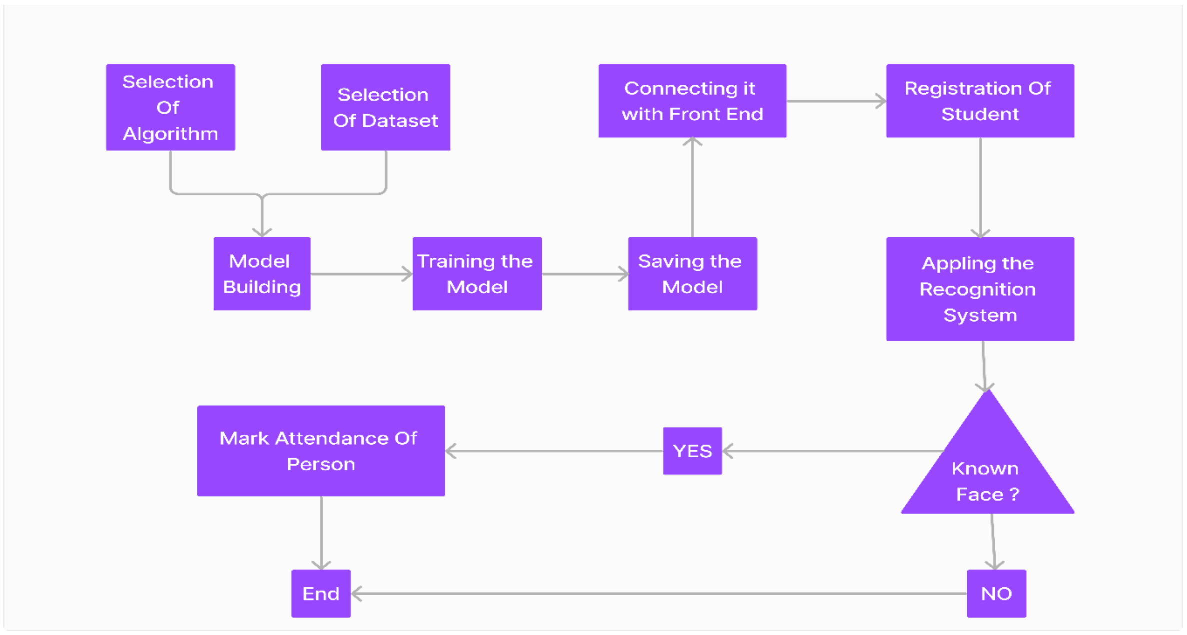


Figure 8: Diagram of visual representation of the architecture.

Components of the Architecture:

1. Image Capture Module:
   - The process starts with an image capture module, where a camera records live video or takes still photos of individuals as they enter a specified area. High-quality cameras with adjustable resolution are recommended to maximize the clarity of images captured in different lighting conditions.
   - This module can be integrated with CCTV cameras already present on site, reducing setup costs and leveraging existing resources.

2. Pre-processing Module:
    - Captured images undergo a series of pre-processing steps to enhance recognition accuracy. These steps include:
        - Face Detection: Identifying and isolating faces in the image using techniques like Haar Cascade Classifier.
        - Image Enhancement: Adjusting brightness, contrast, and resolution to ensure that facial features are clear.
        - Normalization: Scaling and aligning detected faces for consistent size and orientation, improving recognition accuracy.
3. Feature Extraction Module:
    - In this module, facial features are extracted using advanced algorithms like FaceNet, which transforms facial images into a compact feature vector in a high-dimensional space. These feature vectors identify distinct traits of each person's face, enabling the system to accurately differentiate between various faces.
4. Classification and Matching Module:
    - The extracted feature vectors are then processed through a K-Nearest Neighbors (KNN) model. The KNN algorithm compares the feature vector of the detected face against vectors stored in the database to identify the individual. This module classifies the face as either a registered individual (known face) or an unknown face, depending on the similarity score.
    - This process also allows for real-time recognition, enabling the system to quickly and accurately determine identities and mark attendance.
5. Attendance Logging and Storage Module:
    - Once an individual is identified, their attendance is marked and timestamped in a centralized database. This module securely logs attendance records, which can be accessed by administrators or exported for reporting and analysis.
    - The database also supports retrieval and analysis functions, allowing administrators to generate attendance summaries and monitor attendance trends over time.

6. Front-End Interface:
    - The front-end interface allows administrators to view attendance records, configure system settings, and generate reports. It can be a web or mobile application, offering a user-friendly and intuitive interface for interaction. The interface also includes options for enrolling new individuals, managing user profiles, and configuring system settings.

3.6 WORKING

The workflow of the proposed system involves a sequential process of capturing, identifying, and recording attendance data, which can be summarized in the following steps:

1. Initialization:
    - The system initializes by establishing a connection with the camera, configuring settings based on the environment, and ensuring that data storage and processing modules are ready for operation.
2. Face Detection and Capture:
    - When an individual approaches the designated area, the camera captures their image in real-time. The system immediately applies face detection algorithms to identify faces within the captured image, isolating them from the background.
3. Image Pre-processing:
    - Detected faces are pre-processed to standardize quality and prepare them for feature extraction. This includes aligning the faces and enhancing features to improve recognition accuracy.
4. Feature Extraction and Comparison:
    - The system uses FaceNet to convert each detected face into a feature vector, which represents unique facial characteristics. The vector is subsequently compared with the stored feature vectors in the database using the KNN algorithm to check for a match.
    - If the similarity score is above a predefined threshold, the individual is identified as a registered user, and their attendance is marked.

5. Attendance Marking:
    - The system automatically logs the attendance of each recognized individual, including the date and time, into a central database. If an unrecognized face is detected, the system will not mark attendance but can provide an alert or notification if configured.
6. Data Storage and Reporting:
    - Attendance data is securely stored in the database, allowing for future retrieval and analysis. Administrators can generate reports, access logs, and monitor attendance patterns directly from the interface.
7. End of Session:
    - The system continues this process for each person that presents themselves within the camera's range. Once all attendance sessions for the day or event are complete, the system can be closed or left running for continuous operation.

Chapter 4

System Analysis

4.1 MODULE DESCRIPTION

This chapter presents a detailed analysis of the core modules in the facial recognition-based attendance system. The modules include face detection, data pre-processing, feature extraction, classification, and attendance logging. Each module is designed to ensure high accuracy and efficient operation of the attendance tracking system.

4.1.1 IMAGE CAPTURE AND FACE DETECTION

This module captures images using a camera and detects faces within those images. MTCNN (Multi-Task Cascaded Convolutional Neural Network) is utilized for robust face detection, including extracting bounding boxes for detected faces.

Here's how MTCNN is used for face detection:

```
import cv2 as cv
from mtcnn.mtcnn import MTCNN
import matplotlib.pyplot as plt

# Load an image
img = cv.imread("/content/PranavProfile.png")

# Initialize the MTCNN face detector
detector = MTCNN()

# Detect faces in the image
results = detector.detect_faces(img)

# Extract bounding box coordinates
x, y, w, h = results[0]['box']
img = cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 3)  #
Draw a red rectangle around the detected face

# Display the image
```

```
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x78a2809ae200>
```

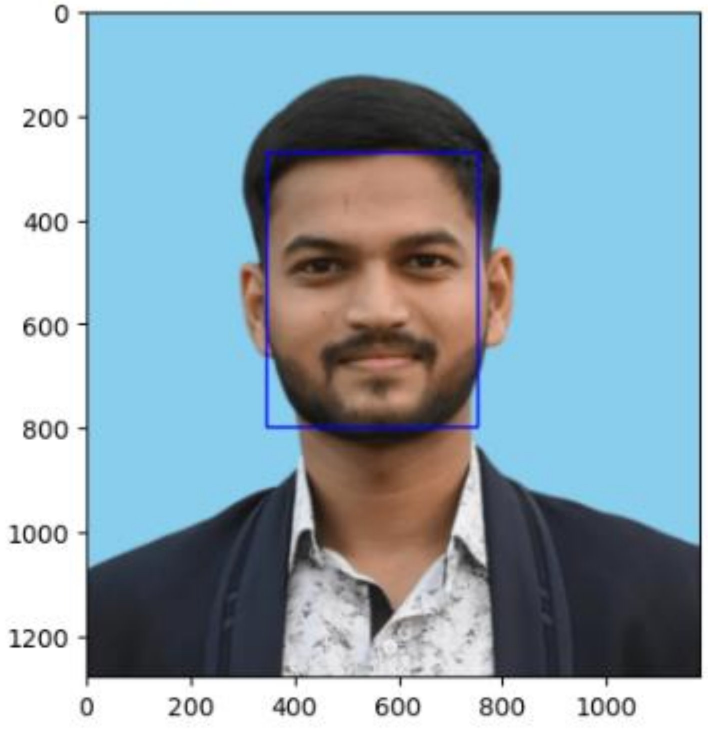


Figure 4.1.1: Sample picture of the dataset and creating a rectangle over the face.

The MTCNN detector identifies faces by using bounding boxes, which allows for effective face extraction. The bounding boxes provide top-left corner coordinates along with width and height for each detected face, which are critical in isolating the face region for subsequent modules.

4.1.2 DATA PRE-PROCESSING

In this module, detected faces undergo data pre-processing, which includes resizing, smoothing, and noise reduction. This ensures that faces are consistently prepared for feature extraction, enhancing the model's accuracy. Gaussian blur is applied for noise reduction, and each face is resized to 160x160 pixels, the input size required by FaceNet.

```
import cv2 as cv
import numpy as np
```

```python
# Crop and resize the detected face
my_face = img[y:y + h, x:x + w]
my_face = cv.resize(my_face, (160, 160))

# Apply Gaussian blur to reduce noise
blurred_face = cv.GaussianBlur(my_face, (5, 5), 0)
face_arr = cv.resize(blurred_face, (160, 160))

# Display the processed face
plt.imshow(face_arr)
```

```
<matplotlib.image.AxesImage at 0x78a26fb5a980>
```

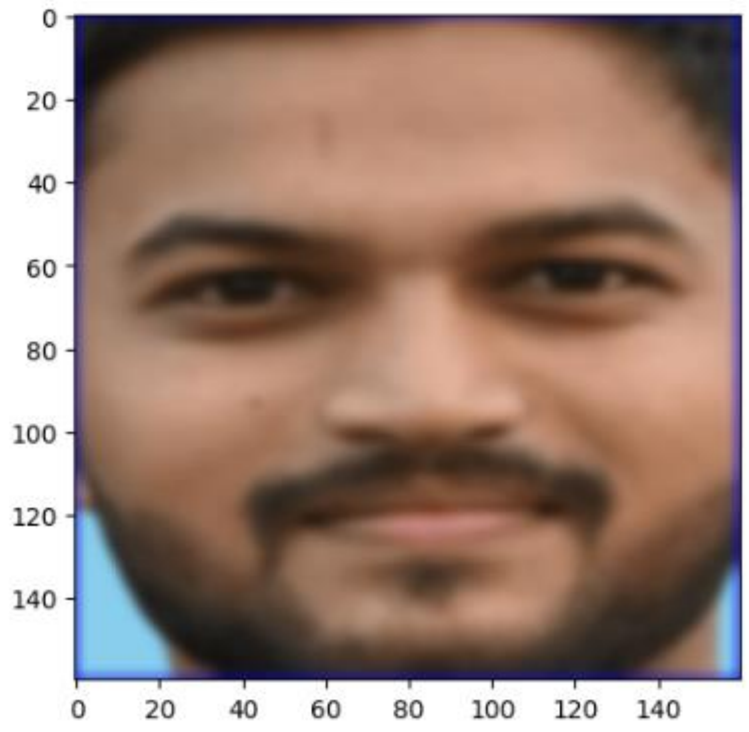


Figure 4.1.2: Apply GaussianBlur and resizing the image.

The following code provides a class for loading and pre-processing face images from a specified directory:

```python
import os
import cv2 as cv
from mtcnn.mtcnn import MTCNN
import numpy as np

class FACELOADING:
    def __init__(self, directory):
        self.directory = directory
        self.target_size = (160, 160)
        self.X = []
        self.Y = []
        self.detector = MTCNN()

    def extract_face(self, filename):
        img = cv.imread(filename)
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        x, y, w, h = self.detector.detect_faces(img)[0]['box']
        face = img[y:y + h, x:x + w]
        face_arr = cv.resize(face, self.target_size)
        return face_arr

    def load_classes(self):
        for sub_dir in os.listdir(self.directory):
            path = os.path.join(self.directory, sub_dir)
            FACES = [self.extract_face(os.path.join(path, im_name))
for im_name in os.listdir(path)]
            self.X.extend(FACES)
            self.Y.extend([sub_dir] * len(FACES))
        return np.asarray(self.X), np.asarray(self.Y)
```

This class manages the loading, detection, and resizing of images to ensure they are ready for feature extraction.

4.1.3 FEATURE EXTRACTION USING FACENET

The FaceNet model extracts a 512-dimensional embedding vector representing unique facial features. This vector serves as an identity representation for each face.

```
from keras_facenet import FaceNet

embedder = FaceNet()

def get_embedding(face_img):
    face_img = face_img.astype('float32')
    face_img = np.expand_dims(face_img, axis=0)
    yhat = embedder.embeddings(face_img)
    return yhat[0]
```

4.1.4 CLASSIFICATION WITH K-NEAREST NEIGHBORS (KNN)

The KNN algorithm classifies the face embeddings obtained from FaceNet. It identifies the closest known embeddings in the dataset to recognize and classify individuals.

4.1.5 ATTENDANCE LOGGING AND DATA STORAGE

Upon successful classification, attendance is logged with a timestamp and saved in a database. This module provides a secure and organized record of attendance data.

4.1.6 FRONT-END INTERFACE AND USER INTERACTION

The user interface allows administrators and students to interact with the system, facilitating tasks such as checking attendance and registering new students.

Chapter 5

Methodology

5.1 FACENET MODEL

FaceNet model is used to transforms faces into a unique 128-dimensional embedding space, where the Euclidean distance between these embeddings reflects how similar the faces are. This feature makes FaceNet highly effective for tasks such as face verification, recognition, and clustering.

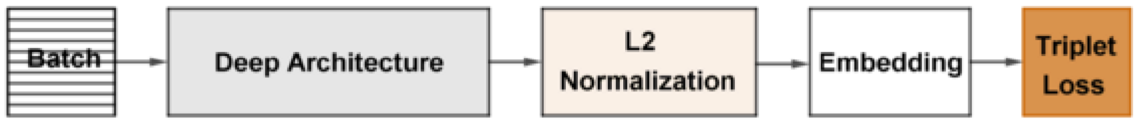


Figure 5.1: FaceNet Model Architecture

5.2 K-NEAREST NEIGHBORS (KNN) CLASSIFIER

K-Nearest Neighbors (k-NN) is a type of algorithm that does not make any assumptions about data distribution, and it operates in a "lazy learning" manner, primarily applied for tasks like classification and regression. In the context of this project, KNN is used as a classifier to identify the closest matches to new face data by evaluating distances between known face embeddings in the dataset. Normalization of training data significantly enhances accuracy, as KNN’s effectiveness relies on the distances between points.

5.3 PROPOSED SYSTEM ARCHITECTURE

The architecture of the proposed system integrates data collection, preprocessing, feature extraction, model building, and deployment.

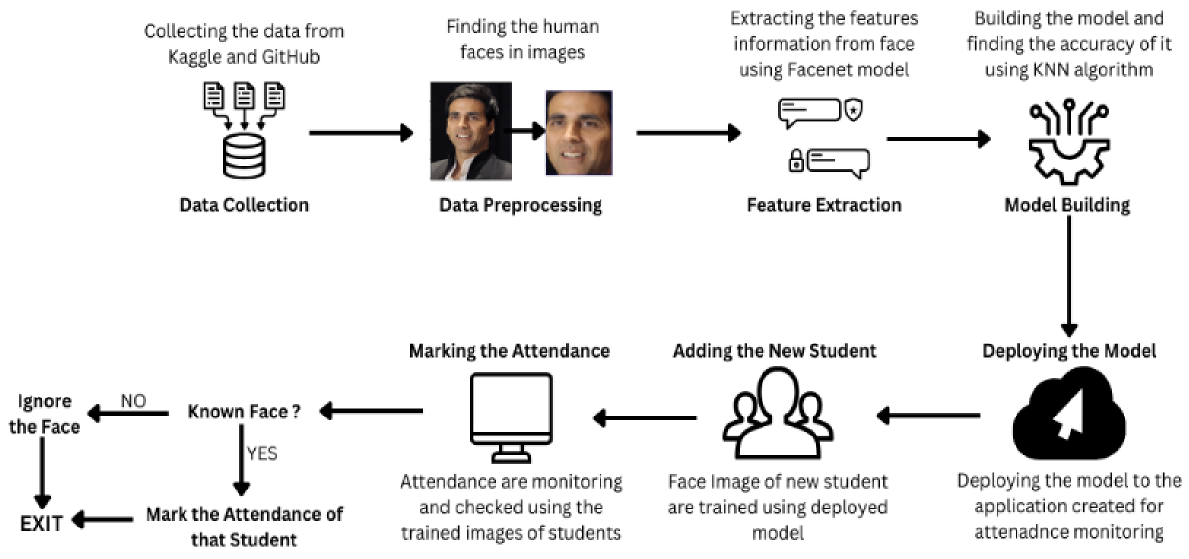

Collecting the data from Kaggle and GitHub
Data Collection
Finding the human faces in images
Data Preprocessing
Extracting the features information from face using Facenet model
Feature Extraction
Building the model and finding the accuracy of it using KNN algorithm
Model Building
Deploying the Model
Deploying the model to the application created for attenadnce monitoring
Adding the New Student
Face Image of new student are trained using deployed model
Marking the Attendance
Attendance are monitoring and checked using the trained images of students
Known Face ?
YES
NO
Ignore the Face
Mark the Attendance of that Student
EXIT

Figure 5.3: Architecture Diagram for the Model.

5.3.1 DATA COLLECTION

The system begins by collecting facial image data from reliable sources, such as Kaggle and GitHub. This dataset includes a diverse set of facial images needed for training and testing the facial recognition model.

5.3.2 DATA PREPROCESSING

After gathering the data, it is subjected to preprocessing steps, which involve:

• Resizing Images: Standardizing image dimensions for consistency.

• Converting Color Space: Transforming images into the RGB format.

• Noise Reduction: Eliminating unnecessary noise to enhance the model's accuracy.

Data preprocessing prepares the images for effective feature extraction and reduces computational load.

5.3.3 FEATURE EXTRACTION

Feature extraction involves identifying unique facial characteristics for each individual. Using FaceNet, this step produces a 128-dimensional embedding for each face, capturing distinct features critical for distinguishing between individuals.

Code For Feature Extraction and Preprocessing

```python
# Importing necessary libraries
import cv2 as cv
import os
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from mtcnn.mtcnn import MTCNN
from keras_facenet import FaceNet

# Function to extract features using FaceNet
def get_embedding(face_img):
    embedder = FaceNet()
    return embedder.embeddings([face_img])[0]

# Sample image preprocessing and feature extraction
img = cv.imread("/content/sample_face.png")
img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
detector = MTCNN()
face_coordinates = detector.detect_faces(img_rgb)[0]['box']
x, y, w, h = face_coordinates
face_img = cv.resize(img_rgb[y:y+h, x:x+w], (160, 160))
face_embedding = get_embedding(face_img)
plt.imshow(face_img)
plt.show()
```

This code snippet demonstrates loading an image, detecting a face using MTCNN, resizing the face region, and generating a FaceNet embedding. The embedding acts as a feature vector that will be used in the classification stage.

Output for Feature Extraction and Preprocessing:

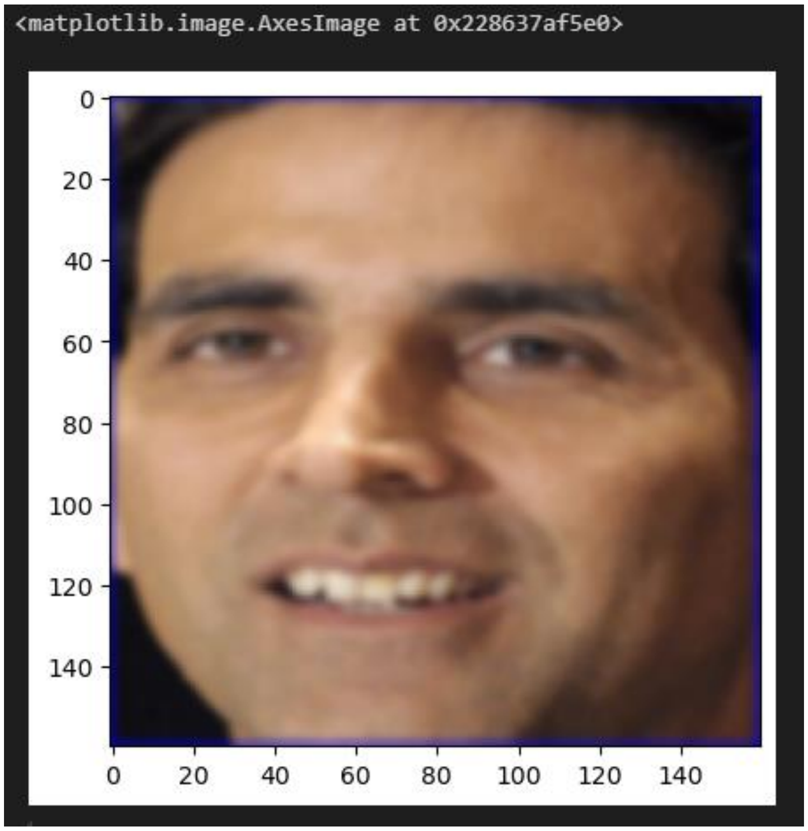

```
<matplotlib.image.AxesImage at 0x228637af5e0>
```

Figure 5.3.3: Output of Feature Extraction and Preprocessing.

5.3.4 MODEL BUILDING

Once feature extraction is complete, a KNN model is trained to classify face embeddings based on known individuals in the dataset. This model allows the system to recognize and label faces by matching new embeddings to previously stored embeddings.

5.3.5 MARKING ATTENDANCE

The recognition system is now capable of marking attendance by processing real-time images. Upon presenting a face to the camera, the system checks for:

1. Face Detection: Determines if a face is present in the image.
2. Face Matching: Compares the embedding of the detected face with known embeddings in the dataset.

If the face matches an existing entry, the system records attendance for the individual.

5.3.6 KNOWN FACE CHECK

The system authenticates identified faces by matching them with the stored embeddings. When a match is confirmed, the attendance of the corresponding student is recorded. If no match is detected, the individual is marked as unidentified.

5.3.7 NEW STUDENT ADDITION

If an unknown face is detected, the system prompts for user input to register the new student. The student’s facial data is then saved and included in the model for future recognition.

5.4 CODE IMPLEMENTATION FOR FACE DATA LOADING AND PROCESSING

The following code defines a class for loading and preprocessing facial images, preparing the data for model training.

```
# Define a class for loading and processing face images
class FACELOADING:
    def __init__(self, directory):
        self.directory = directory
        self.target_size = (160, 160)
        self.X = []
        self.Y = []
        self.detector = MTCNN()

    def extract_face(self, filename):
        img = cv.imread(filename)
        img_rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        x, y, w, h = self.detector.detect_faces(img_rgb)[0]['box']
        face_img = img_rgb[y:y+h, x:x+w]
        return cv.resize(face_img, self.target_size)

    def load_classes(self):
        for label in os.listdir(self.directory):
            class_dir = os.path.join(self.directory, label)
            faces = [self.extract_face(os.path.join(class_dir, img))
for img in os.listdir(class_dir)]
```

```
            self.X.extend(faces)
            self.Y.extend([label] * len(faces))
        return np.array(self.X), np.array(self.Y)

# Load face data from specified directory
face_loader = FACELOADING("/path/to/data")
X, Y = face_loader.load_classes()
```

The FACELOADING class loads images from a directory, detects faces using MTCNN, and resizes them for use with FaceNet. Each face image is categorized according to its respective directory, creating an organized dataset for training the recognition model.

Chapter 6

Results

6.1 AVERAGE ACCURACY ACROSS ITERATIONS

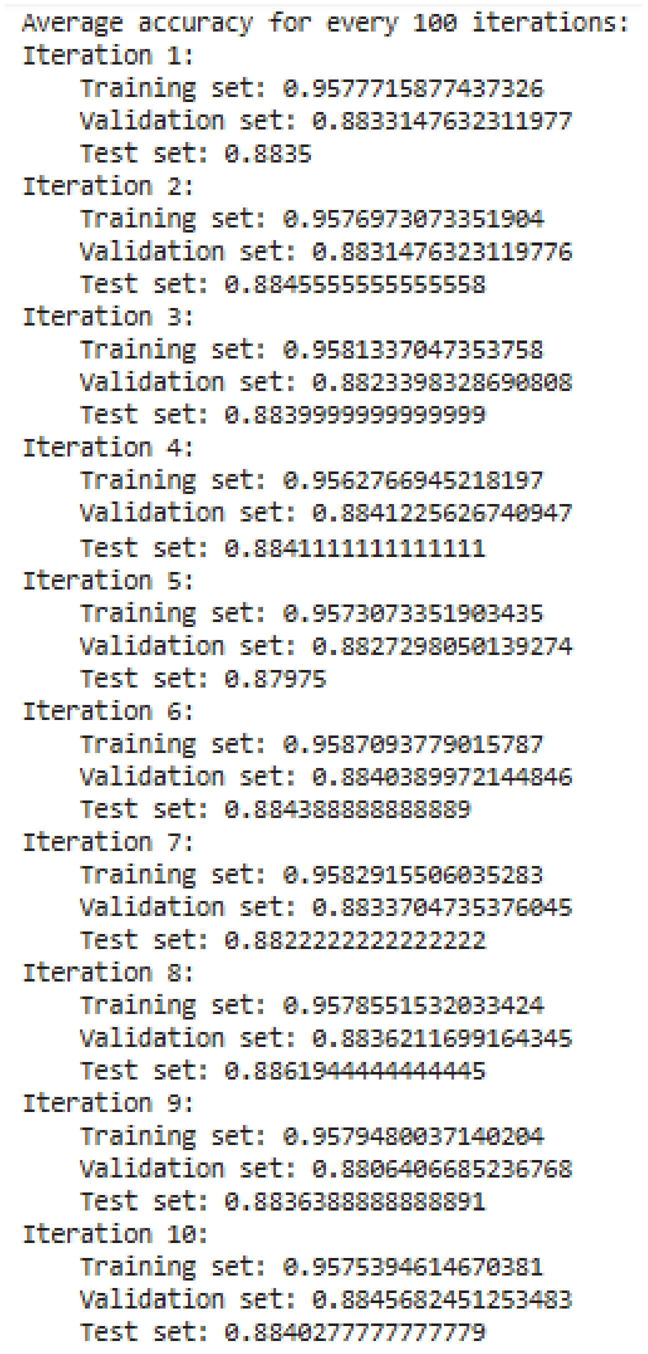
This chapter presents the results of the proposed model's performance. Different metrics, including training accuracy, validation accuracy, and test accuracy, are assessed to evaluate the model's performance. Additionally, graphical representations provide insights into the distribution and accuracy trends across multiple iterations.

In this section, we compute the average accuracy for the training, validation, and test datasets after every 100 iterations. This provides a clearer understanding of the model's performance over an extended period and helps to identify any variations or patterns.

The code below calculates the average accuracy for every 100 iterations. Using the mean accuracy for each set of 100 iterations allows us to observe the model's stability over time. The results are printed for each of the 10 iterations and are also visualized in a line plot.

```
# Find average accuracy for every 100 values
train_avg = [np.mean(train_scores[i:i+100]) for i in range(0,
len(train_scores), 100)]
validation_avg = [np.mean(validation_scores[i:i+100]) for i in
range(0, len(validation_scores), 100)]
test_avg = [np.mean(test_scores[i:i+100]) for i in range(0,
len(test_scores), 100)]

# Print average accuracy for every 100 iterations
print("Average accuracy for every 100 iterations:")
for i in range(10):
    print(f"Iteration {i+1}:")
    print(f"    Training set: {train_avg[i]}")
    print(f"    Validation set: {validation_avg[i]}")
    print(f"    Test set: {test_avg[i]}")
```

Results of Average Accuracy for Every 100 Iterations:



```
Average accuracy for every 100 iterations:
Iteration 1:
    Training set: 0.9577715877437326
    Validation set: 0.8833147632311977
    Test set: 0.8835
Iteration 2:
    Training set: 0.9576973073351904
    Validation set: 0.8831476323119776
    Test set: 0.8845555555555558
Iteration 3:
    Training set: 0.9581337047353758
    Validation set: 0.8823398328690808
    Test set: 0.8839999999999999
Iteration 4:
    Training set: 0.9562766945218197
    Validation set: 0.8841225626740947
    Test set: 0.8841111111111111
Iteration 5:
    Training set: 0.9573073351903435
    Validation set: 0.8827298050139274
    Test set: 0.87975
Iteration 6:
    Training set: 0.9587093779015787
    Validation set: 0.8840389972144846
    Test set: 0.884388888888889
Iteration 7:
    Training set: 0.9582915506035283
    Validation set: 0.8833704735376045
    Test set: 0.8822222222222222
Iteration 8:
    Training set: 0.9578551532033424
    Validation set: 0.8836211699164345
    Test set: 0.8861944444444445
Iteration 9:
    Training set: 0.9579480037140204
    Validation set: 0.8806406685236768
    Test set: 0.8836388888888891
Iteration 10:
    Training set: 0.9575394614670381
    Validation set: 0.8845682451253483
    Test set: 0.8840277777777779
```

Figure 6.1: Results of Average Accuracy for Every 100 Iterations

```
# Plot the graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), train_avg, label='Training Set', marker='o')
plt.plot(range(1, 11), validation_avg, label='Validation Set',
marker='o')
plt.plot(range(1, 11), test_avg, label='Test Set', marker='o')
plt.title('Average Accuracy across 100 iterations')
plt.xlabel('Iteration')
plt.ylabel('Average Accuracy')
plt.xticks(range(1, 11))
plt.legend()
plt.grid(True)
plt.show()
```

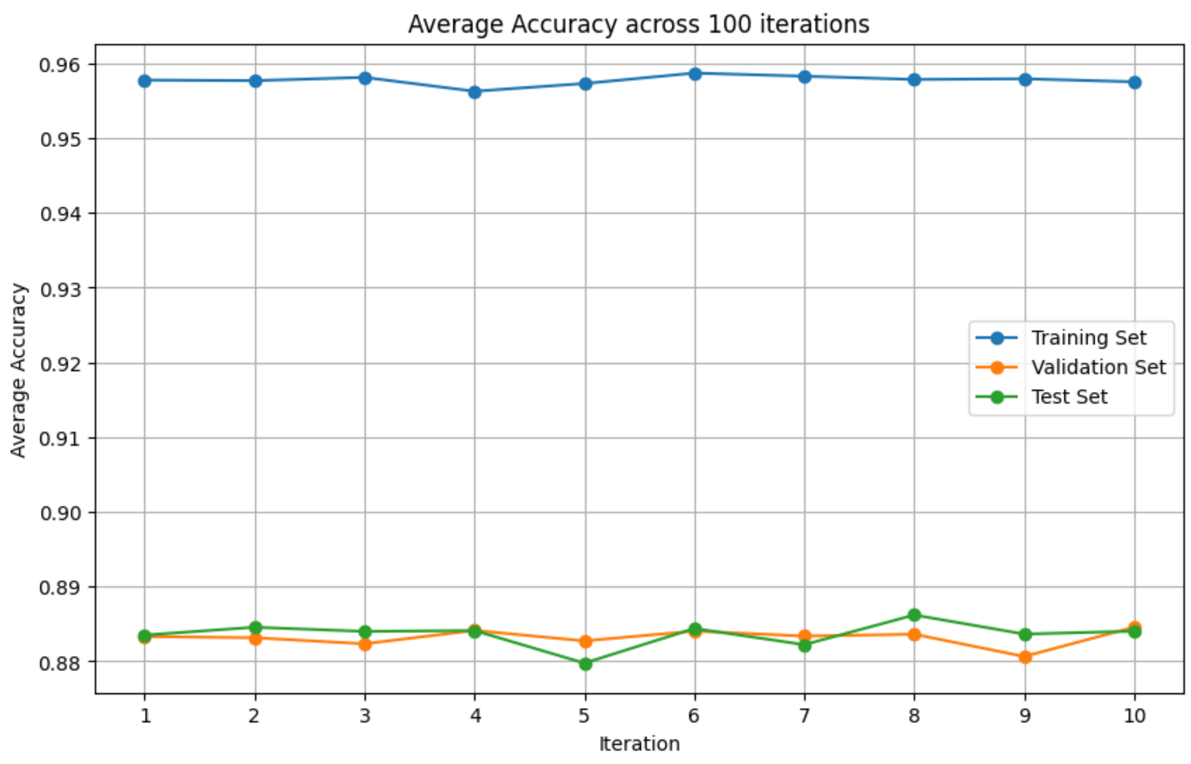


Figure 6.1.1: Line plot of average accuracy across 100 iterations

6.2 FACE EMBEDDING DISTRIBUTION VISUALIZATION

To understand the distribution of face embeddings generated by the model, a t-SNE (t-Distributed Stochastic Neighbor Embedding) visualization is employed. This 2D scatter plot helps illustrate how the model clusters different facial embeddings, showing the separation between distinct classes.

The code uses t-SNE for dimensionality reduction, visualizing high-dimensional face embeddings in a 2D space. The scatter plot represents each embedding point, with colors representing different classes.

```
# Using t-SNE for dimensionality reduction (visualize in 2D)
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0)
X_2d = tsne.fit_transform(EMBEDDED_X)

# Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_2d[:,0], y=X_2d[:,1], hue=Y, palette='viridis')
plt.title('Distribution of Face Embeddings')
plt.show()
```

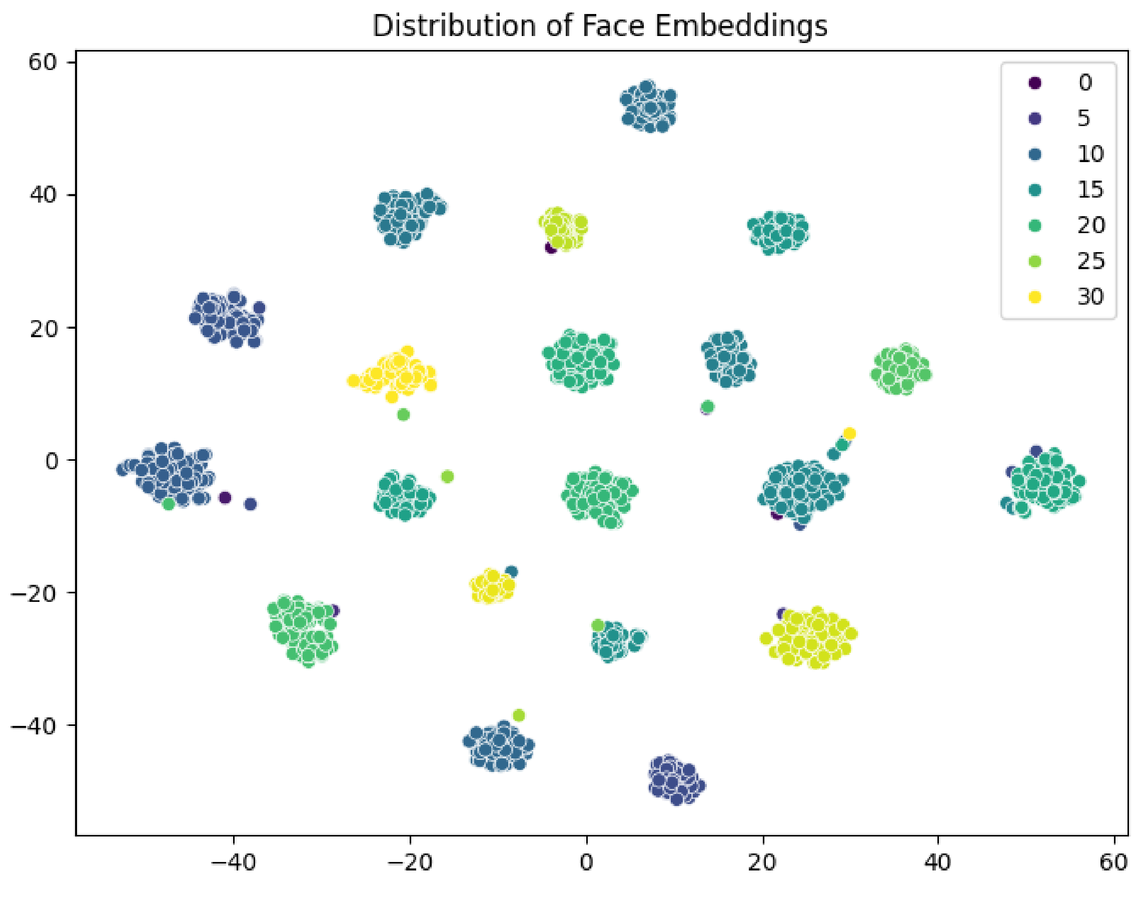


Figure 6.2: Scatter plot of face embeddings using t-SNE

6.3 3D REPRESENTATION OF ACCURACY RELATIONS

To visualize the relationship between the training, validation, and test accuracies during the first iteration, a 3D scatter plot is used. This 3D representation allows for an intuitive understanding of the differences and potential correlations among the three metrics in a single view.

In this code, the 3D plot shows a single point corresponding to the average accuracies from the first iteration for each set. This visualization can be helpful for analyzing the model’s accuracy metrics cohesively.

```
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Select the first element from each list for Iteration 1
avg_train_accuracy = train_avg[0]  # Average training accuracy for
Iteration 1
avg_validation_accuracy = validation_avg[0]  # Average validation
accuracy for Iteration 1
avg_accuracy = test_avg[0]
 Iteration 1

# Plot the single point
ax.scatter(avg_train_accuracy, avg_validation_accuracy,
avg_test_accuracy, c='b', marker='o')

# Define the axes labels
ax.set_xlabel('Training Accuracy')
ax.set_ylabel('Validation Accuracy')
ax.set_zlabel('Test Accuracy')

plt.title('Confusion Matrix - Accuracy Relation (Iteration 1)')
plt.show()
```

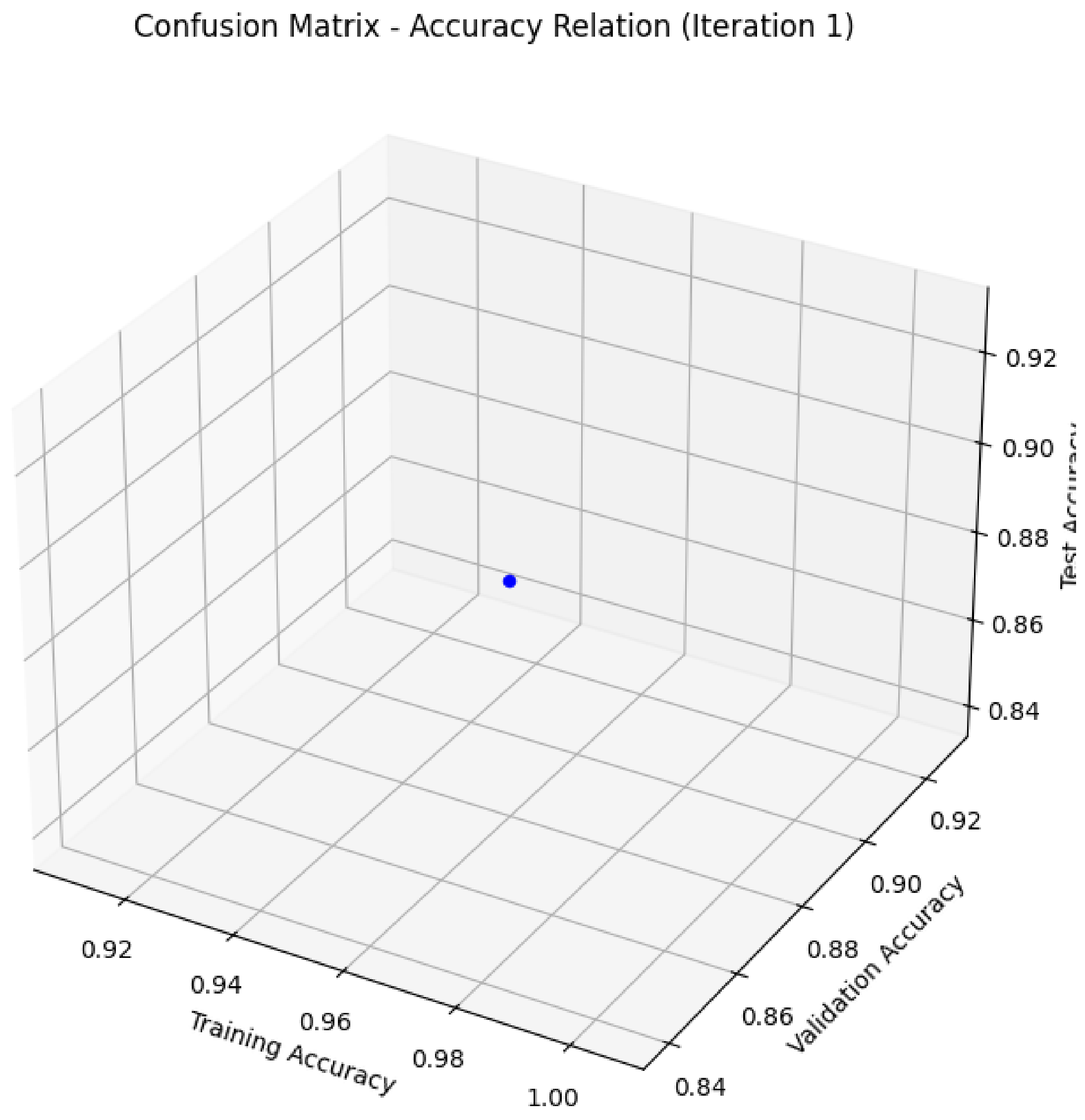
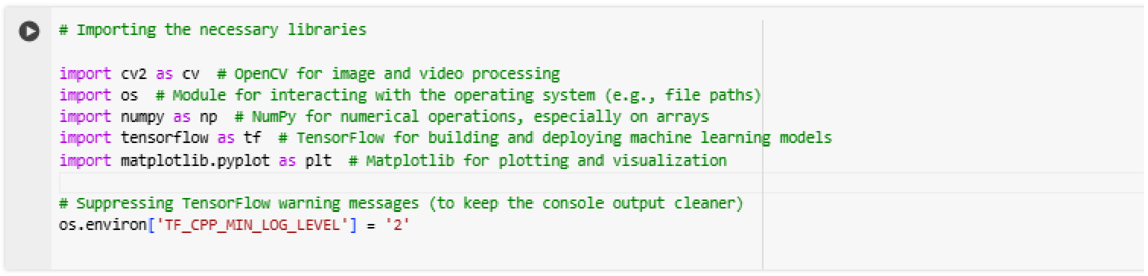


Figure 6.3: 3D scatter plot showing accuracy relations for the first iteration

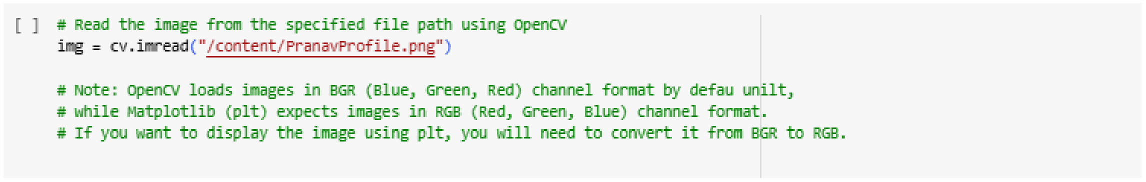
Chapter 7

Code and Output Screenshots

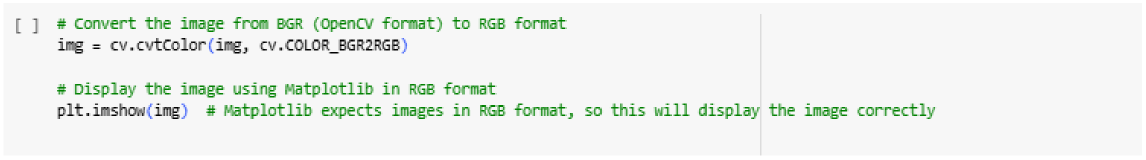
8.1 Code and Output Screenshots



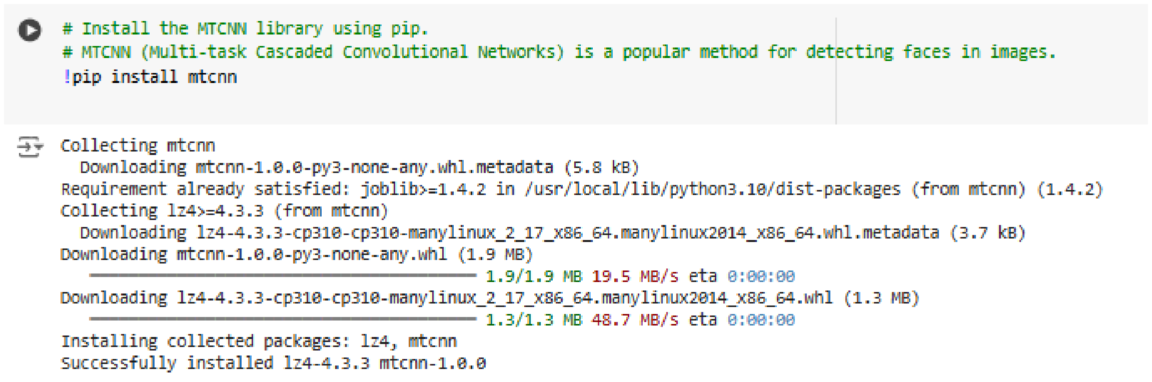
```
# Importing the necessary libraries

import cv2 as cv  # OpenCV for image and video processing
import os  # Module for interacting with the operating system (e.g., file paths)
import numpy as np  # NumPy for numerical operations, especially on arrays
import tensorflow as tf  # TensorFlow for building and deploying machine learning models
import matplotlib.pyplot as plt  # Matplotlib for plotting and visualization

# Suppressing TensorFlow warning messages (to keep the console output cleaner)
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```



```
[ ] # Read the image from the specified file path using OpenCV
    img = cv.imread("/content/PranavProfile.png")

    # Note: OpenCV loads images in BGR (Blue, Green, Red) channel format by defau unilt,
    # while Matplotlib (plt) expects images in RGB (Red, Green, Blue) channel format.
    # If you want to display the image using plt, you will need to convert it from BGR to RGB.
```

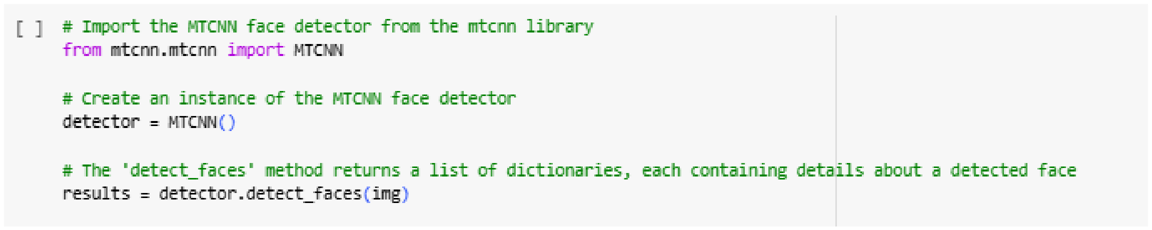


```
[ ] # Convert the image from BGR (OpenCV format) to RGB format
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

    # Display the image using Matplotlib in RGB format
    plt.imshow(img)  # Matplotlib expects images in RGB format, so this will display the image correctly
```

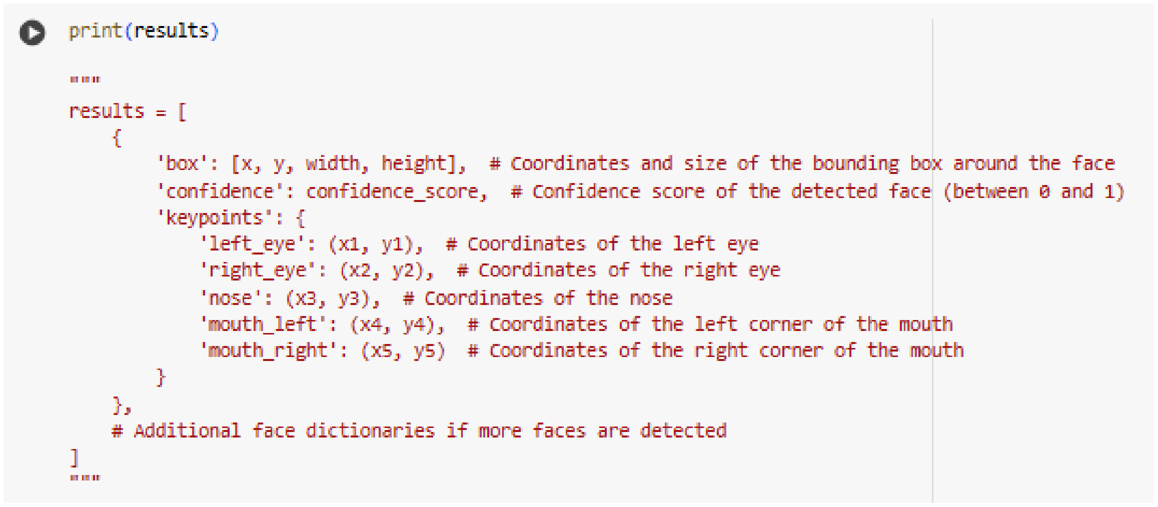


```
# Install the MTCNN library using pip.
# MTCNN (Multi-task Cascaded Convolutional Networks) is a popular method for detecting faces in images.
!pip install mtcnn
```

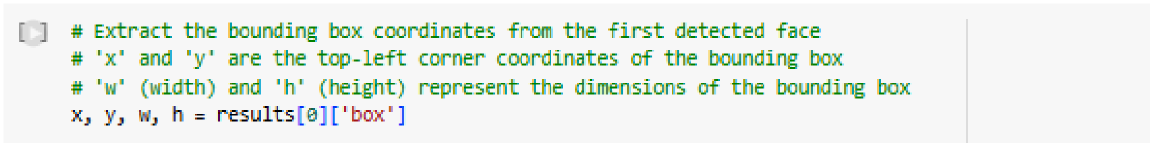
```
Collecting mtcnn
  Downloading mtcnn-1.0.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: joblib>=1.4.2 in /usr/local/lib/python3.10/dist-packages (from mtcnn) (1.4.2)
Collecting lz4>=4.3.3 (from mtcnn)
  Downloading lz4-4.3.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.7 kB)
Downloading mtcnn-1.0.0-py3-none-any.whl (1.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 19.5 MB/s eta 0:00:00
Downloading lz4-4.3.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 48.7 MB/s eta 0:00:00
Installing collected packages: lz4, mtcnn
Successfully installed lz4-4.3.3 mtcnn-1.0.0
```



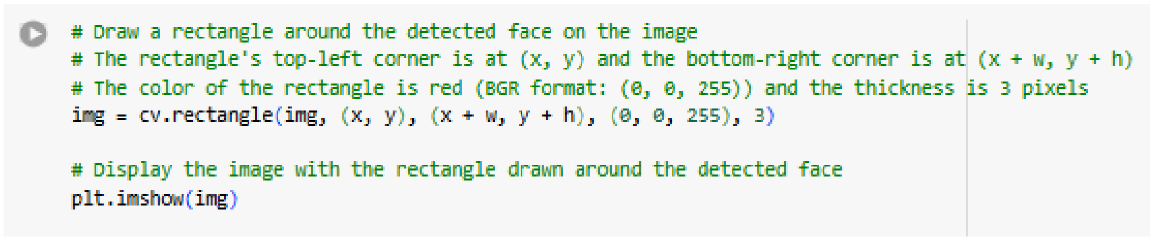
```
[ ] # Import the MTCNN face detector from the mtcnn library
    from mtcnn.mtcnn import MTCNN

    # Create an instance of the MTCNN face detector
    detector = MTCNN()

    # The 'detect_faces' method returns a list of dictionaries, each containing details about a detected face
    results = detector.detect_faces(img)
```



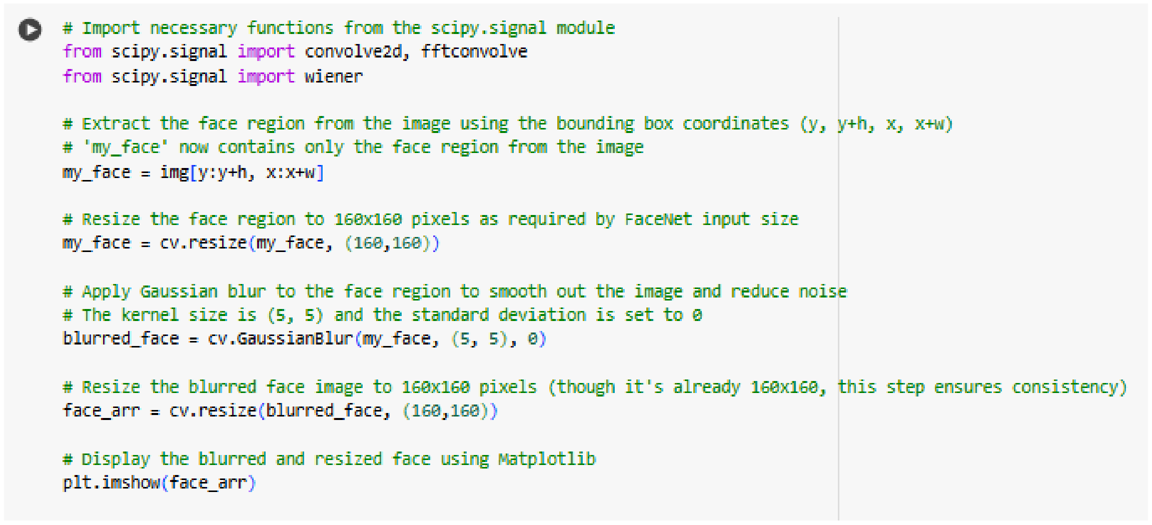
```
print(results)

"""
results = [
    {
        'box': [x, y, width, height],  # Coordinates and size of the bounding box around the face
        'confidence': confidence_score,  # Confidence score of the detected face (between 0 and 1)
        'keypoints': {
            'left_eye': (x1, y1),  # Coordinates of the left eye
            'right_eye': (x2, y2),  # Coordinates of the right eye
            'nose': (x3, y3),  # Coordinates of the nose
            'mouth_left': (x4, y4),  # Coordinates of the left corner of the mouth
            'mouth_right': (x5, y5)  # Coordinates of the right corner of the mouth
        }
    },
    # Additional face dictionaries if more faces are detected
]
"""
```



```
# Extract the bounding box coordinates from the first detected face
# 'x' and 'y' are the top-left corner coordinates of the bounding box
# 'w' (width) and 'h' (height) represent the dimensions of the bounding box
x, y, w, h = results[0]['box']
```



```
# Draw a rectangle around the detected face on the image
# The rectangle's top-left corner is at (x, y) and the bottom-right corner is at (x + w, y + h)
# The color of the rectangle is red (BGR format: (0, 0, 255)) and the thickness is 3 pixels
img = cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 3)

# Display the image with the rectangle drawn around the detected face
plt.imshow(img)
```

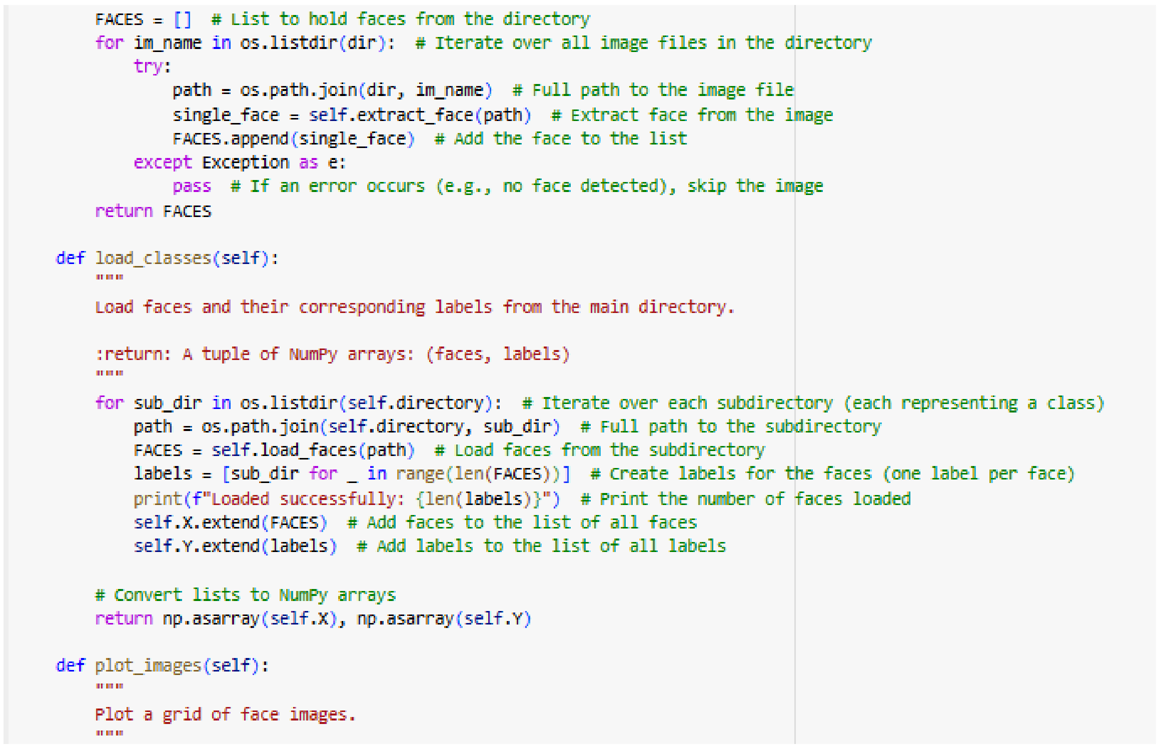


```
# Import necessary functions from the scipy.signal module
from scipy.signal import convolve2d, fftconvolve
from scipy.signal import wiener

# Extract the face region from the image using the bounding box coordinates (y, y+h, x, x+w)
# 'my_face' now contains only the face region from the image
my_face = img[y:y+h, x:x+w]

# Resize the face region to 160x160 pixels as required by FaceNet input size
my_face = cv.resize(my_face, (160,160))

# Apply Gaussian blur to the face region to smooth out the image and reduce noise
# The kernel size is (5, 5) and the standard deviation is set to 0
blurred_face = cv.GaussianBlur(my_face, (5, 5), 0)

# Resize the blurred face image to 160x160 pixels (though it's already 160x160, this step ensures consistency)
face_arr = cv.resize(blurred_face, (160,160))

# Display the blurred and resized face using Matplotlib
plt.imshow(face_arr)
```

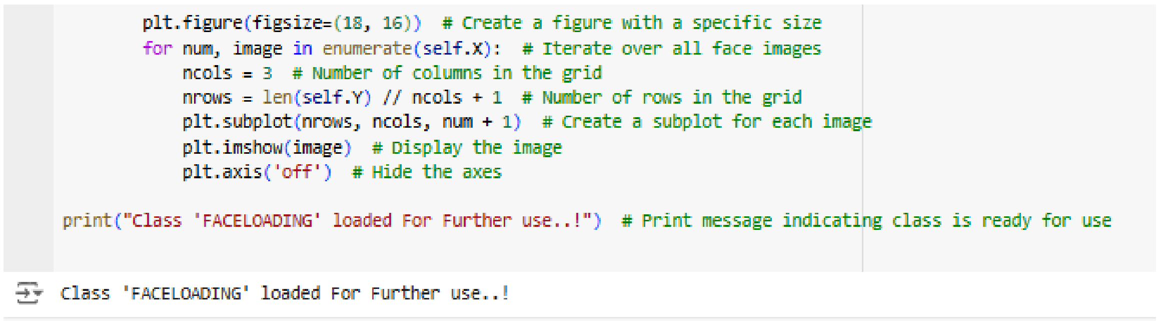
```
my_face
#contains the cropped region of the original image that corresponds to the detected face.
```

```
ndarray (160, 160, 3) show data
```

```
# Define a class for loading and preprocessing face data
class FACELOADING:
    def __init__(self, directory):
        """
        Initialize the FACELOADING class.

        :param directory: The directory containing subdirectories of face images.
        """
        self.directory = directory  # Main directory containing subdirectories of images
        self.target_size = (160, 160)  # Target size for resizing faces
        self.X = []  # List to hold face images
        self.Y = []  # List to hold labels for each face image
        self.detector = MTCNN()  # Initialize the MTCNN face detector

    def extract_face(self, filename):
        """
        Extract and preprocess the face from an image file.

        :param filename: The path to the image file.
        :return: The cropped and resized face image.
        """
        img = cv.imread(filename)  # Read the image from file
        img = cv.cvtColor(img, cv.COLOR_BGR2RGB)  # Convert image from BGR to RGB format

        # Detect faces in the image and extract the bounding box of the first detected face
        x, y, w, h = self.detector.detect_faces(img)[0]['box']
        x, y = abs(x), abs(y)  # Ensure coordinates are positive
        face = img[y:y + h, x:x + w]  # Crop the face from the image
        face_arr = cv.resize(face, self.target_size)  # Resize the face to the target size
        return face_arr

    def load_faces(self, dir):
        """
        Load faces from a given directory.

        :param dir: The directory containing images of faces.
        :return: A list of face images.
        """
```

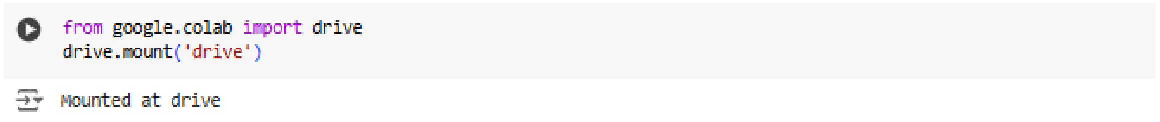


```python
        FACES = []  # List to hold faces from the directory
        for im_name in os.listdir(dir):  # Iterate over all image files in the directory
            try:
                path = os.path.join(dir, im_name)  # Full path to the image file
                single_face = self.extract_face(path)  # Extract face from the image
                FACES.append(single_face)  # Add the face to the list
            except Exception as e:
                pass  # If an error occurs (e.g., no face detected), skip the image
        return FACES

    def load_classes(self):
        """
        Load faces and their corresponding labels from the main directory.

        :return: A tuple of NumPy arrays: (faces, labels)
        """
        for sub_dir in os.listdir(self.directory):  # Iterate over each subdirectory (each representing a class)
            path = os.path.join(self.directory, sub_dir)  # Full path to the subdirectory
            FACES = self.load_faces(path)  # Load faces from the subdirectory
            labels = [sub_dir for _ in range(len(FACES))]  # Create labels for the faces (one label per face)
            print(f"Loaded successfully: {len(labels)}")  # Print the number of faces loaded
            self.X.extend(FACES)  # Add faces to the list of all faces
            self.Y.extend(labels)  # Add labels to the list of all labels

        # Convert lists to NumPy arrays
        return np.asarray(self.X), np.asarray(self.Y)

    def plot_images(self):
        """
        Plot a grid of face images.
        """
```



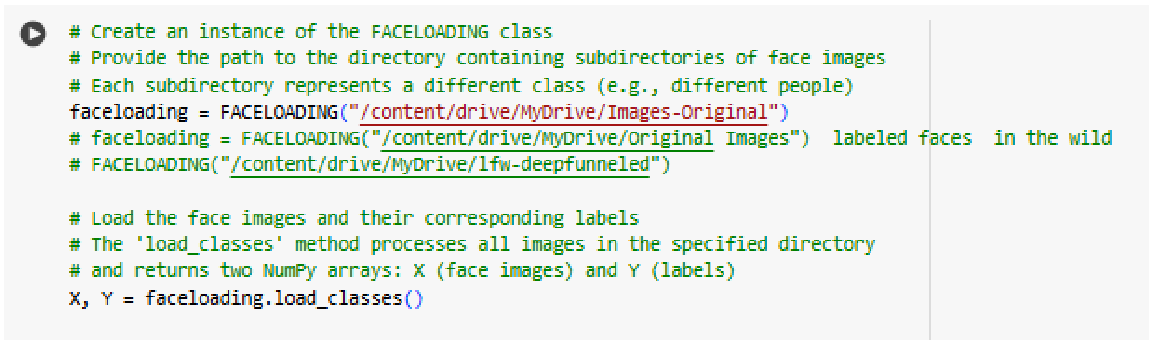
```python
        plt.figure(figsize=(18, 16))  # Create a figure with a specific size
        for num, image in enumerate(self.X):  # Iterate over all face images
            ncols = 3  # Number of columns in the grid
            nrows = len(self.Y) // ncols + 1  # Number of rows in the grid
            plt.subplot(nrows, ncols, num + 1)  # Create a subplot for each image
            plt.imshow(image)  # Display the image
            plt.axis('off')  # Hide the axes

print("Class 'FACELOADING' loaded For Further use..!")  # Print message indicating class is ready for use
```

```
Class 'FACELOADING' loaded For Further use..!
```

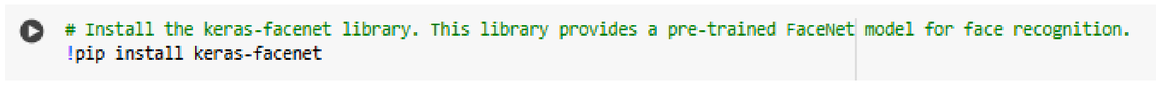


```python
from google.colab import drive
drive.mount('drive')
```

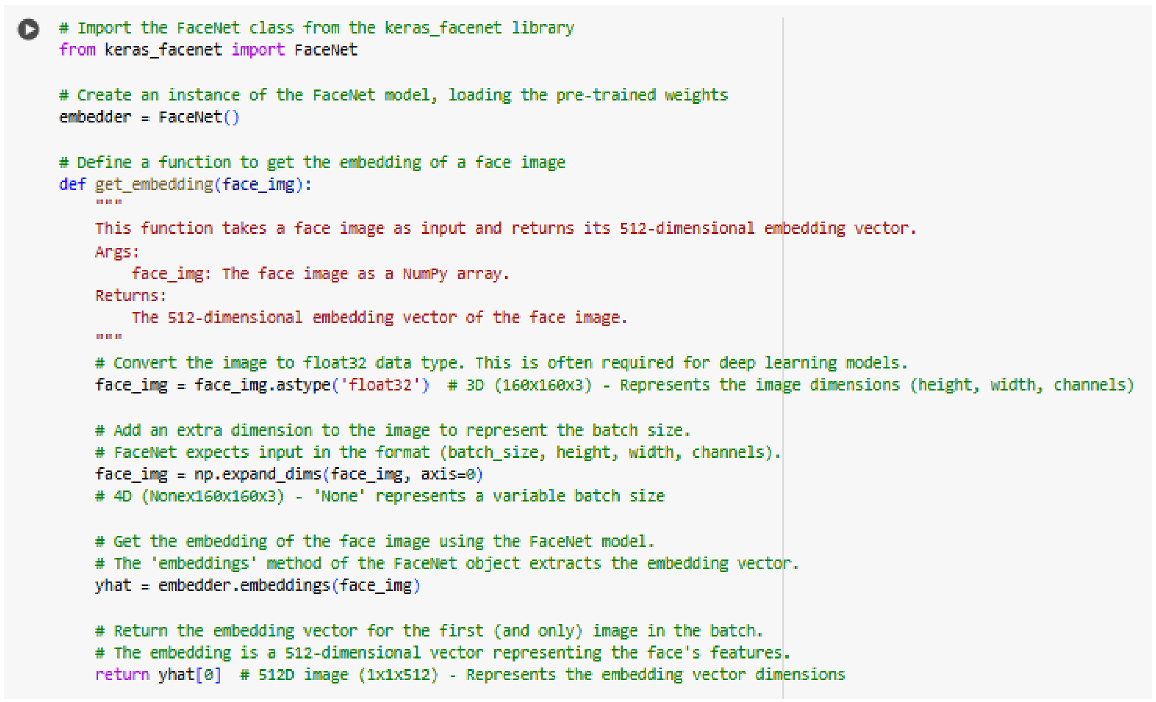
```
Mounted at drive
```



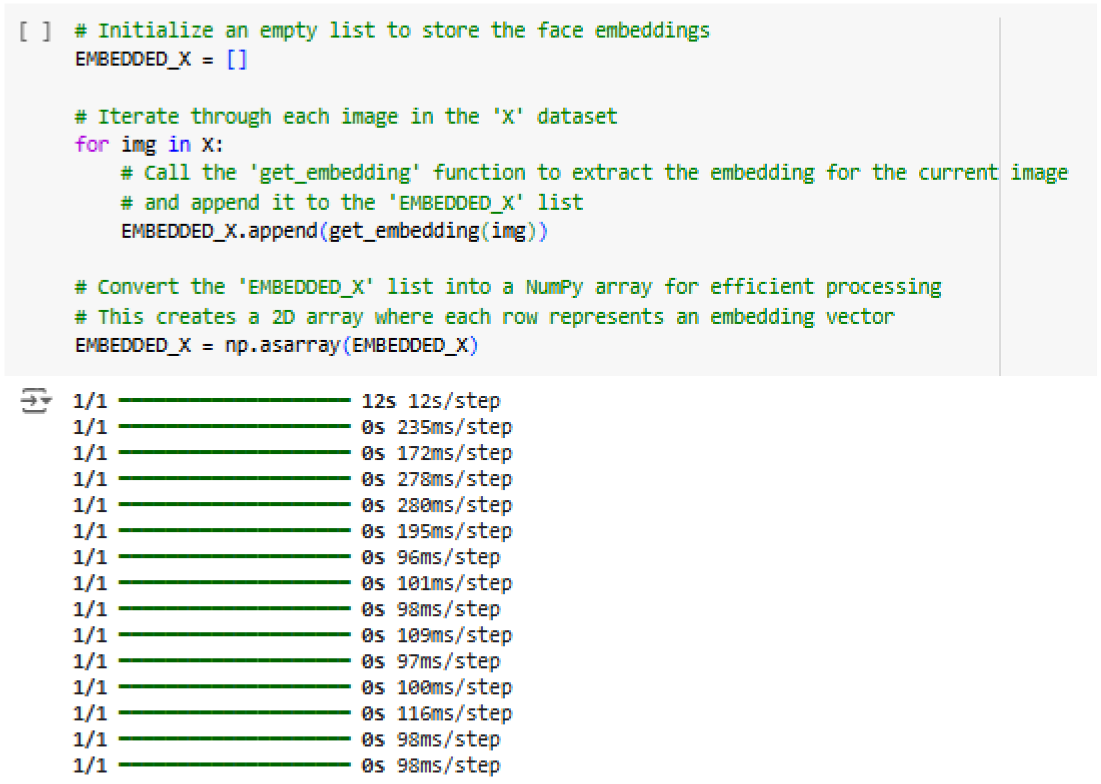
```python
# Create an instance of the FACELOADING class
# Provide the path to the directory containing subdirectories of face images
# Each subdirectory represents a different class (e.g., different people)
faceloading = FACELOADING("/content/drive/MyDrive/Images-Original")
# faceloading = FACELOADING("/content/drive/MyDrive/Original Images")  labeled faces  in the wild
# FACELOADING("/content/drive/MyDrive/lfw-deepfunneled")

# Load the face images and their corresponding labels
# The 'load_classes' method processes all images in the specified directory
# and returns two NumPy arrays: X (face images) and Y (labels)
X, Y = faceloading.load_classes()
```



```python
# Install the keras-facenet library. This library provides a pre-trained FaceNet model for face recognition.
!pip install keras-facenet
```

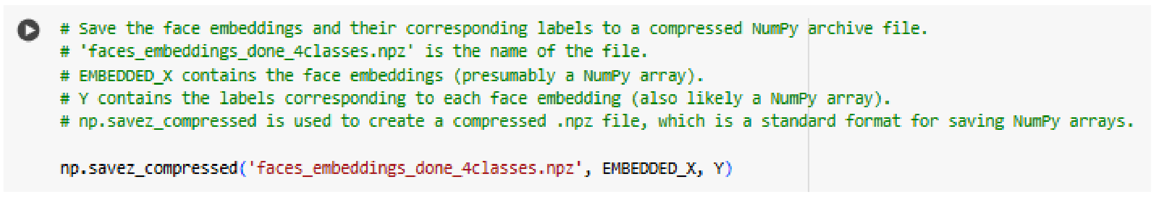


```python
# Import the FaceNet class from the keras_facenet library
from keras_facenet import FaceNet

# Create an instance of the FaceNet model, loading the pre-trained weights
embedder = FaceNet()

# Define a function to get the embedding of a face image
def get_embedding(face_img):
    """
    This function takes a face image as input and returns its 512-dimensional embedding vector.
    Args:
        face_img: The face image as a NumPy array.
    Returns:
        The 512-dimensional embedding vector of the face image.
    """
    # Convert the image to float32 data type. This is often required for deep learning models.
    face_img = face_img.astype('float32')  # 3D (160x160x3) - Represents the image dimensions (height, width, channels)

    # Add an extra dimension to the image to represent the batch size.
    # FaceNet expects input in the format (batch_size, height, width, channels).
    face_img = np.expand_dims(face_img, axis=0)
    # 4D (Nonex160x160x3) - 'None' represents a variable batch size

    # Get the embedding of the face image using the FaceNet model.
    # The 'embeddings' method of the FaceNet object extracts the embedding vector.
    yhat = embedder.embeddings(face_img)

    # Return the embedding vector for the first (and only) image in the batch.
    # The embedding is a 512-dimensional vector representing the face's features.
    return yhat[0]  # 512D image (1x1x512) - Represents the embedding vector dimensions
```

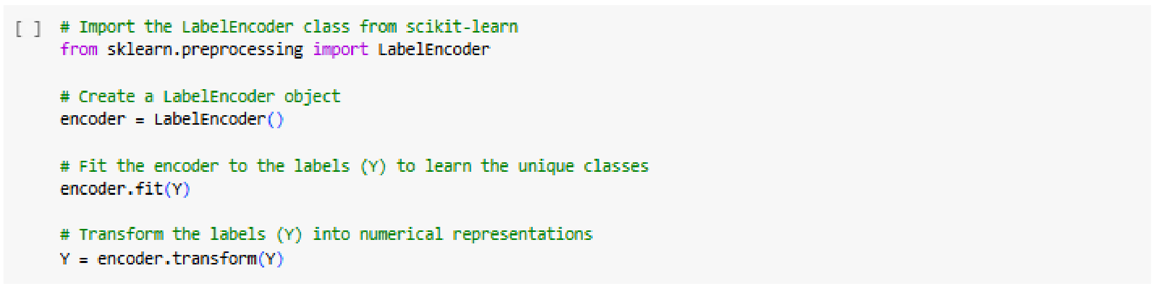


```python
# Initialize an empty list to store the face embeddings
EMBEDDED_X = []

# Iterate through each image in the 'X' dataset
for img in X:
    # Call the 'get_embedding' function to extract the embedding for the current image
    # and append it to the 'EMBEDDED_X' list
    EMBEDDED_X.append(get_embedding(img))

# Convert the 'EMBEDDED_X' list into a NumPy array for efficient processing
# This creates a 2D array where each row represents an embedding vector
EMBEDDED_X = np.asarray(EMBEDDED_X)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 12s 12s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 235ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 172ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 278ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 280ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 195ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
```



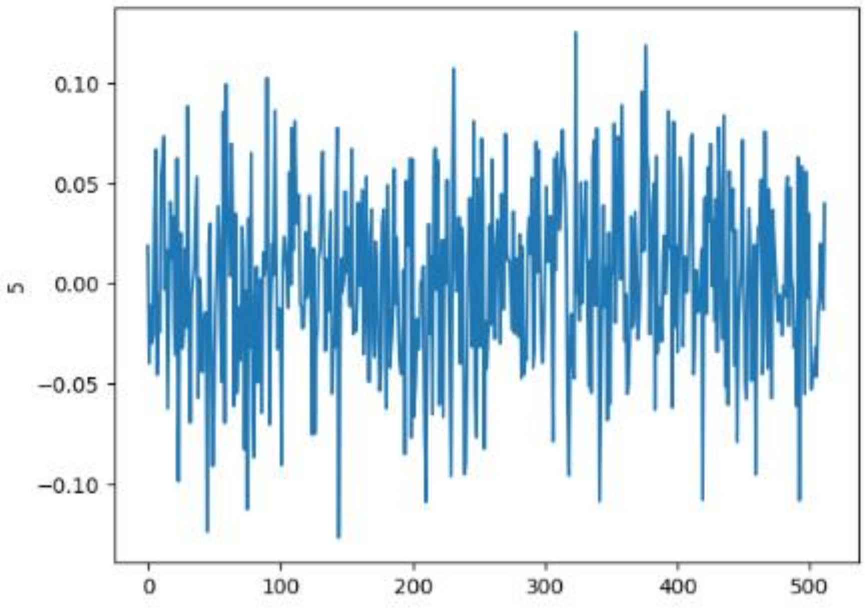
```
# Save the face embeddings and their corresponding labels to a compressed NumPy archive file.
# 'faces_embeddings_done_4classes.npz' is the name of the file.
# EMBEDDED_X contains the face embeddings (presumably a NumPy array).
# Y contains the labels corresponding to each face embedding (also likely a NumPy array).
# np.savez_compressed is used to create a compressed .npz file, which is a standard format for saving NumPy arrays.

np.savez_compressed('faces_embeddings_done_4classes.npz', EMBEDDED_X, Y)
```



```
[ ] # Import the LabelEncoder class from scikit-learn
    from sklearn.preprocessing import LabelEncoder

    # Create a LabelEncoder object
    encoder = LabelEncoder()

    # Fit the encoder to the labels (Y) to learn the unique classes
    encoder.fit(Y)

    # Transform the labels (Y) into numerical representations
    Y = encoder.transform(Y)
```

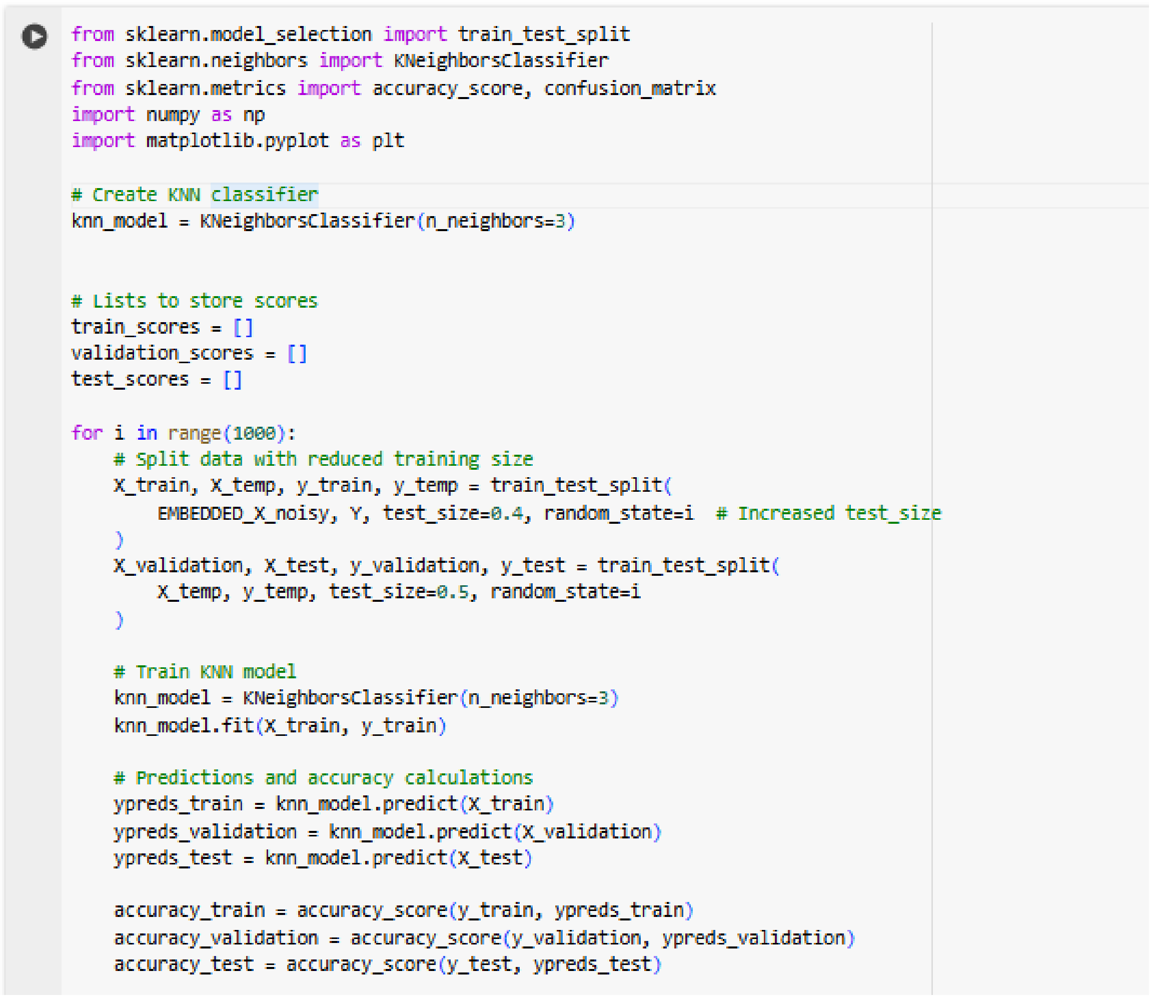
```
# Plot the values of the first face embedding (EMBEDDED_X[0])
plt.plot(EMBEDDED_X[0])

# Set the y-axis label to the corresponding label of the first face embedding (Y[0])
plt.ylabel(Y[0])
```

```
Text(0, 0.5, '5')
```

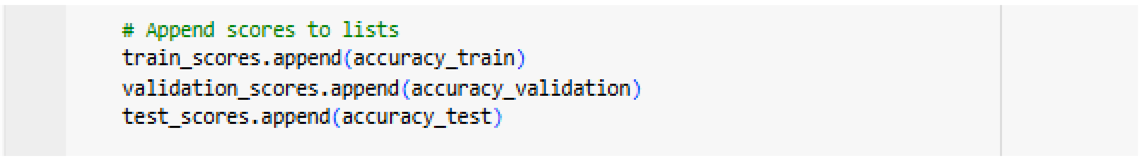


```
[ ] Y
```

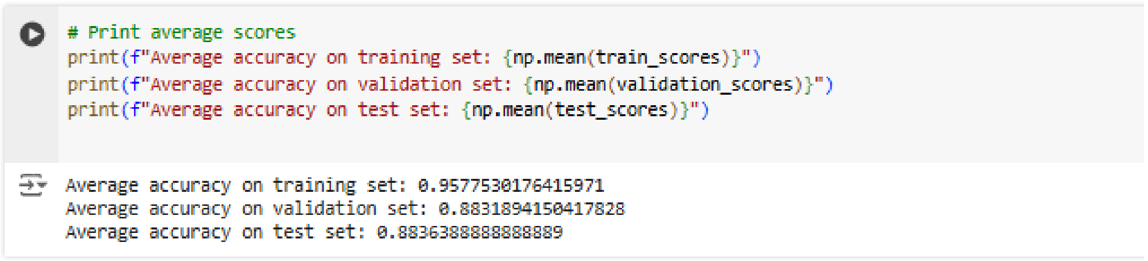
```
array([ 5,  2,  4, ..., 30, 30, 30])
```



```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt

# Create KNN classifier
knn_model = KNeighborsClassifier(n_neighbors=3)


# Lists to store scores
train_scores = []
validation_scores = []
test_scores = []

for i in range(1000):
    # Split data with reduced training size
    X_train, X_temp, y_train, y_temp = train_test_split(
        EMBEDDED_X_noisy, Y, test_size=0.4, random_state=i  # Increased test_size
    )
    X_validation, X_test, y_validation, y_test = train_test_split(
        X_temp, y_temp, test_size=0.5, random_state=i
    )

    # Train KNN model
    knn_model = KNeighborsClassifier(n_neighbors=3)
    knn_model.fit(X_train, y_train)

    # Predictions and accuracy calculations
    ypreds_train = knn_model.predict(X_train)
    ypreds_validation = knn_model.predict(X_validation)
    ypreds_test = knn_model.predict(X_test)

    accuracy_train = accuracy_score(y_train, ypreds_train)
    accuracy_validation = accuracy_score(y_validation, ypreds_validation)
    accuracy_test = accuracy_score(y_test, ypreds_test)
```

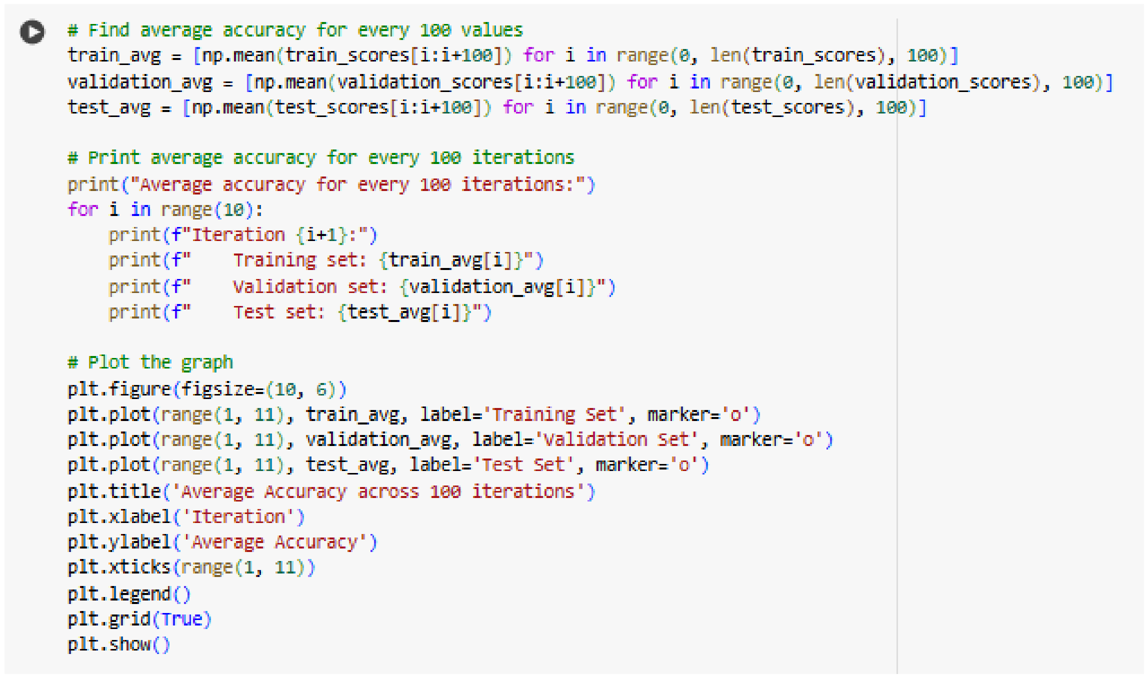


```python
    # Append scores to lists
    train_scores.append(accuracy_train)
    validation_scores.append(accuracy_validation)
    test_scores.append(accuracy_test)
```

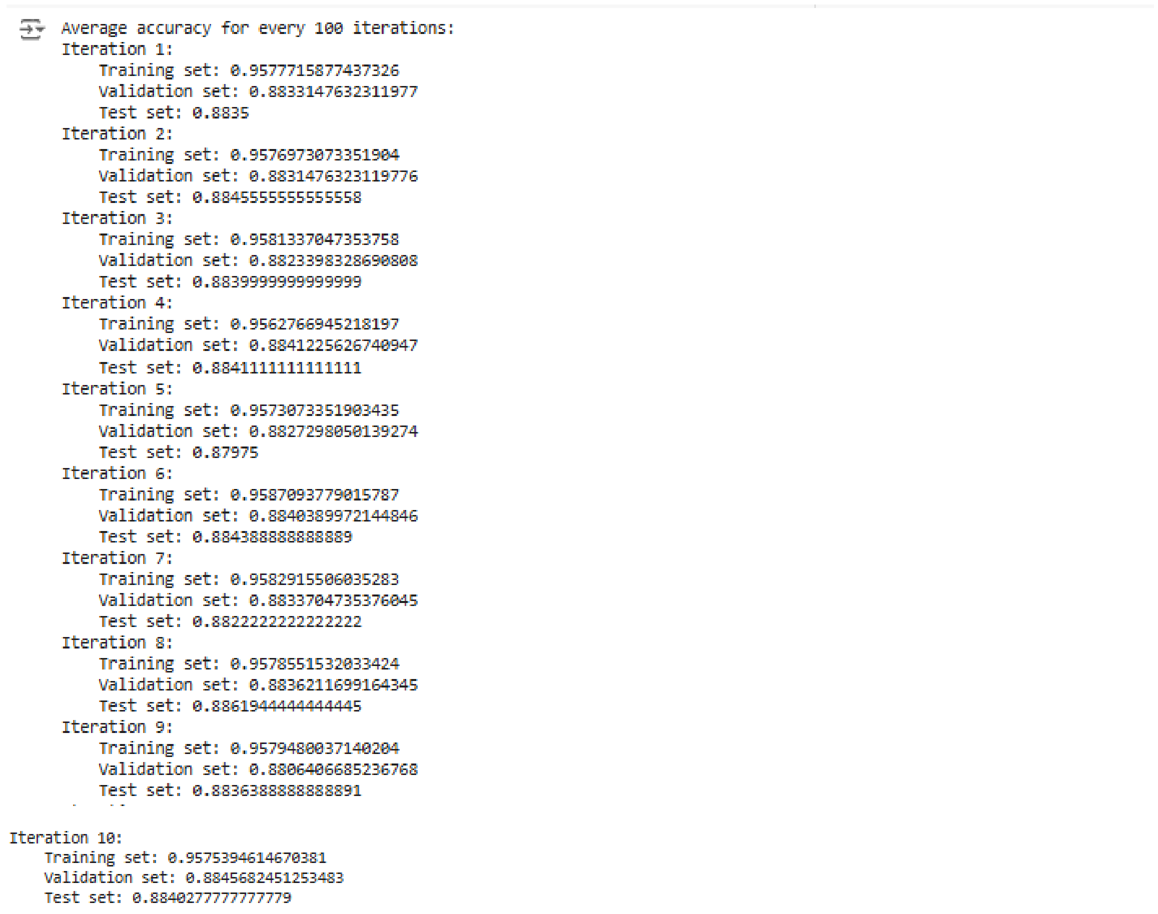


```python
# Print average scores
print(f"Average accuracy on training set: {np.mean(train_scores)}")
print(f"Average accuracy on validation set: {np.mean(validation_scores)}")
print(f"Average accuracy on test set: {np.mean(test_scores)}")
```

```
Average accuracy on training set: 0.9577530176415971
Average accuracy on validation set: 0.8831894150417828
Average accuracy on test set: 0.8836388888888889
```

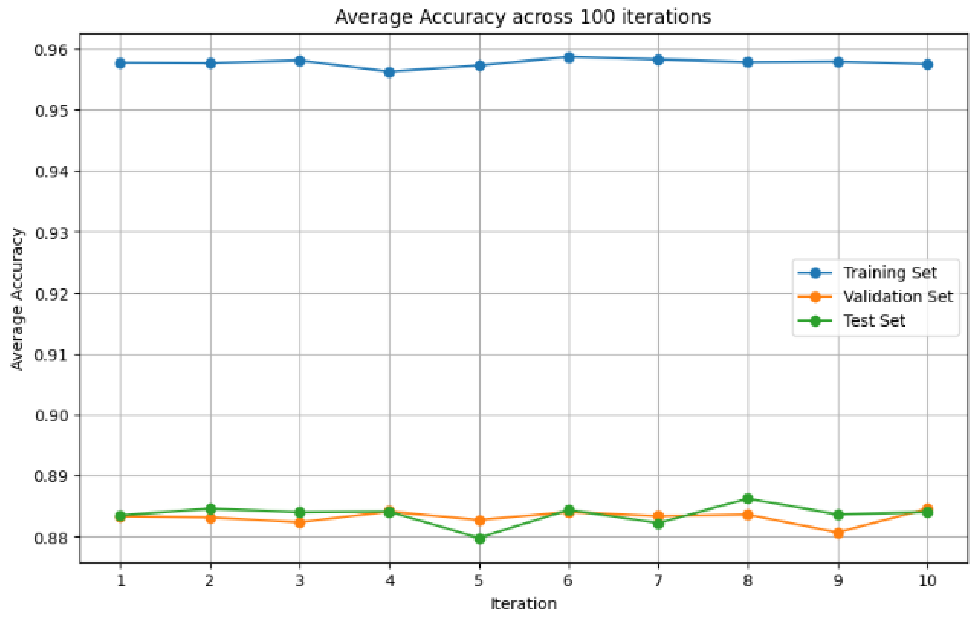


```python
# Find average accuracy for every 100 values
train_avg = [np.mean(train_scores[i:i+100]) for i in range(0, len(train_scores), 100)]
validation_avg = [np.mean(validation_scores[i:i+100]) for i in range(0, len(validation_scores), 100)]
test_avg = [np.mean(test_scores[i:i+100]) for i in range(0, len(test_scores), 100)]

# Print average accuracy for every 100 iterations
print("Average accuracy for every 100 iterations:")
for i in range(10):
    print(f"Iteration {i+1}:")
    print(f"    Training set: {train_avg[i]}")
    print(f"    Validation set: {validation_avg[i]}")
    print(f"    Test set: {test_avg[i]}")

# Plot the graph
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), train_avg, label='Training Set', marker='o')
plt.plot(range(1, 11), validation_avg, label='Validation Set', marker='o')
plt.plot(range(1, 11), test_avg, label='Test Set', marker='o')
plt.title('Average Accuracy across 100 iterations')
plt.xlabel('Iteration')
plt.ylabel('Average Accuracy')
plt.xticks(range(1, 11))
plt.legend()
plt.grid(True)
plt.show()
```

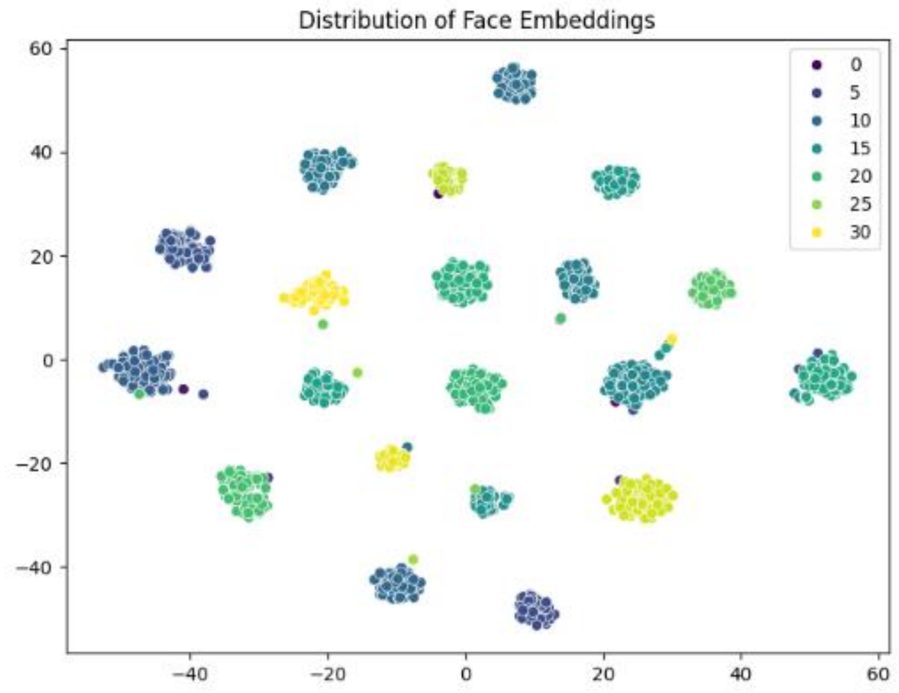


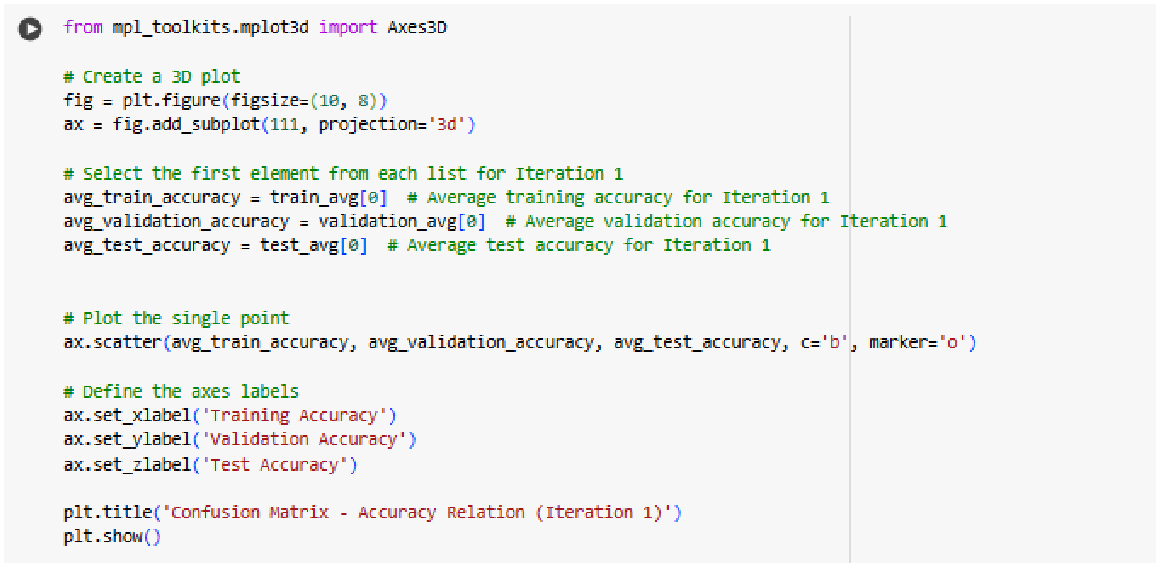
```
Average accuracy for every 100 iterations:
Iteration 1:
    Training set: 0.9577715877437326
    Validation set: 0.8833147632311977
    Test set: 0.8835
Iteration 2:
    Training set: 0.9576973073351904
    Validation set: 0.8831476323119776
    Test set: 0.8845555555555558
Iteration 3:
    Training set: 0.9581337047353758
    Validation set: 0.8823398328690808
    Test set: 0.8839999999999999
Iteration 4:
    Training set: 0.9562766945218197
    Validation set: 0.8841225626740947
    Test set: 0.8841111111111111
Iteration 5:
    Training set: 0.9573073351903435
    Validation set: 0.8827298050139274
    Test set: 0.87975
Iteration 6:
    Training set: 0.9587093779015787
    Validation set: 0.8840389972144846
    Test set: 0.884388888888889
Iteration 7:
    Training set: 0.9582915506035283
    Validation set: 0.8833704735376045
    Test set: 0.8822222222222222
Iteration 8:
    Training set: 0.9578551532033424
    Validation set: 0.8836211699164345
    Test set: 0.8861944444444445
Iteration 9:
    Training set: 0.9579480037140204
    Validation set: 0.8806406685236768
    Test set: 0.8836388888888891
```

```
Iteration 10:
    Training set: 0.9575394614670381
    Validation set: 0.8845682451253483
    Test set: 0.8840277777777779
```

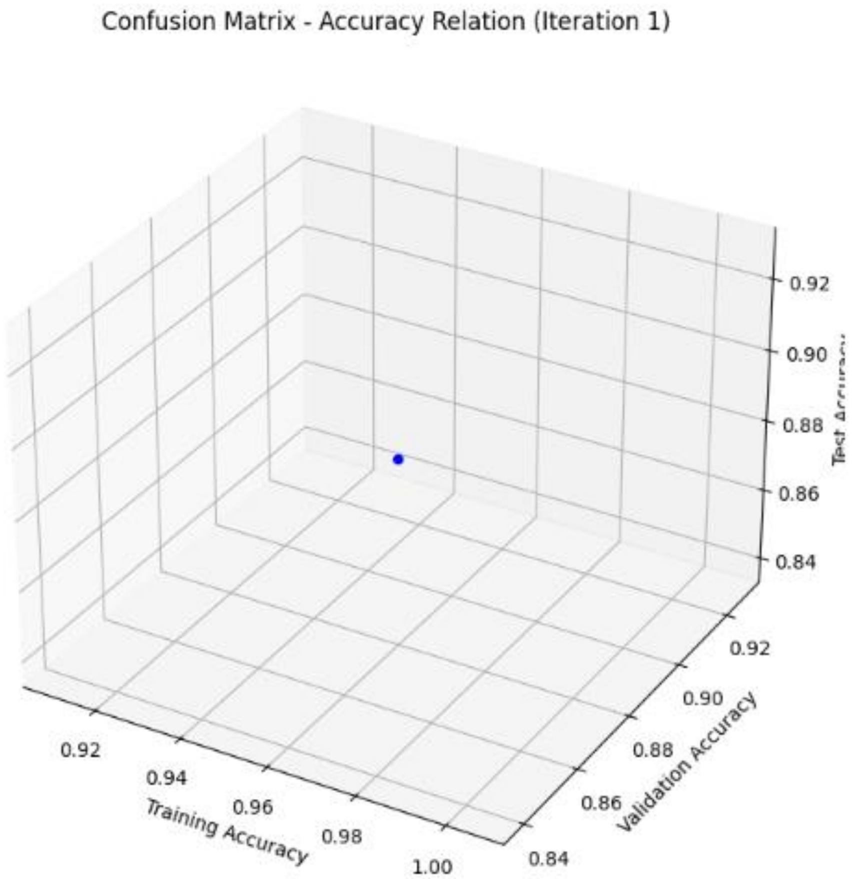


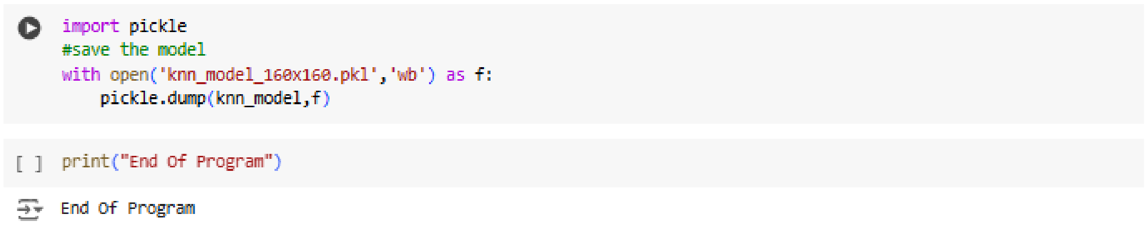
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Using t-SNE for dimensionality reduction (visualize in 2D)
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, random_state=0)
X_2d = tsne.fit_transform(EMBEDDED_X)

# Scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_2d[:,0], y=X_2d[:,1], hue=Y, palette='viridis')
plt.title('Distribution of Face Embeddings')
plt.show()
```





```
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Select the first element from each list for Iteration 1
avg_train_accuracy = train_avg[0]  # Average training accuracy for Iteration 1
avg_validation_accuracy = validation_avg[0]  # Average validation accuracy for Iteration 1
avg_test_accuracy = test_avg[0]  # Average test accuracy for Iteration 1


# Plot the single point
ax.scatter(avg_train_accuracy, avg_validation_accuracy, avg_test_accuracy, c='b', marker='o')

# Define the axes labels
ax.set_xlabel('Training Accuracy')
ax.set_ylabel('Validation Accuracy')
ax.set_zlabel('Test Accuracy')

plt.title('Confusion Matrix - Accuracy Relation (Iteration 1)')
plt.show()
```





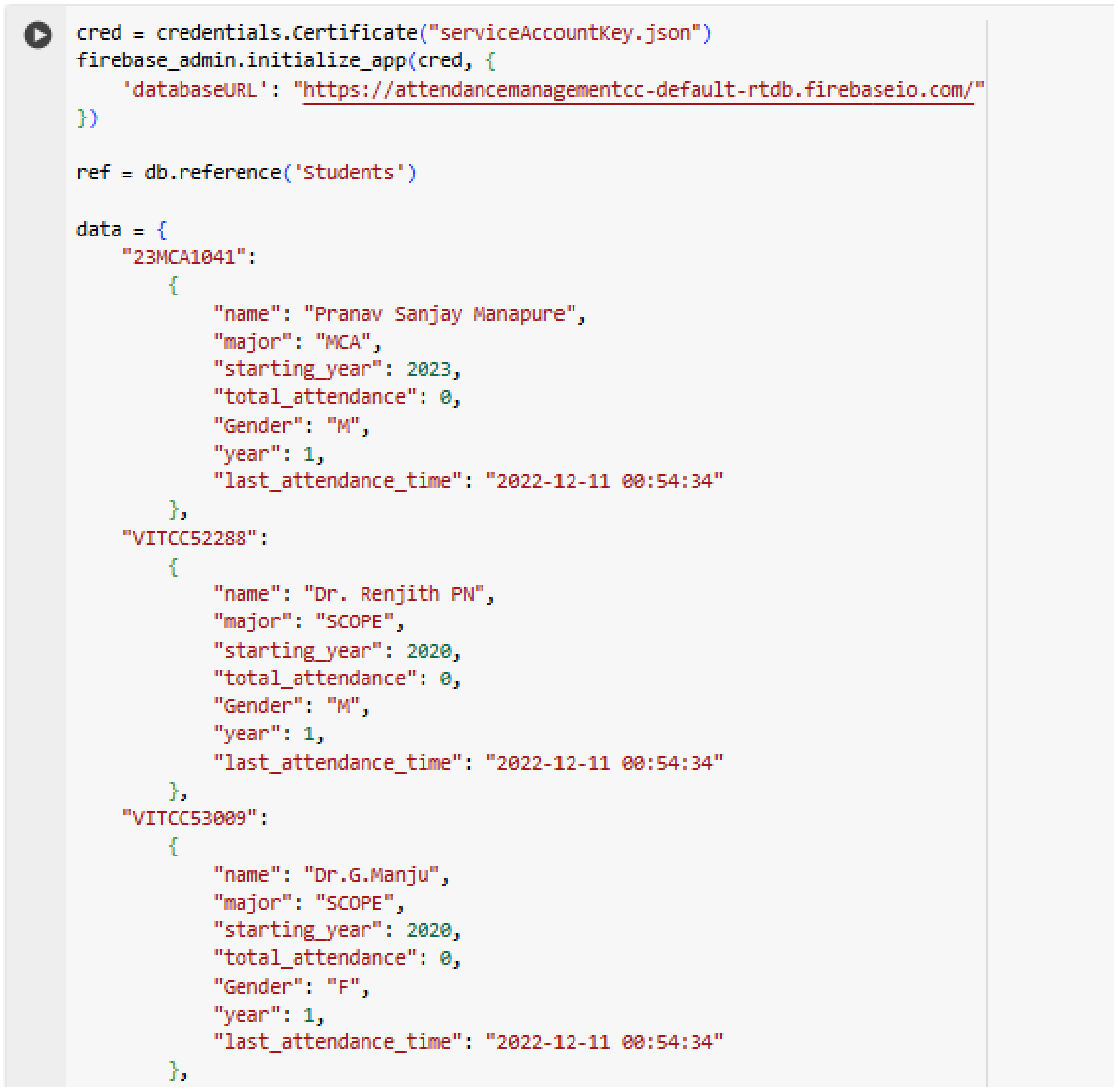
```
import pickle
#save the model
with open('knn_model_160x160.pkl','wb') as f:
    pickle.dump(knn_model,f)
```

```
[ ] print("End Of Program")
```

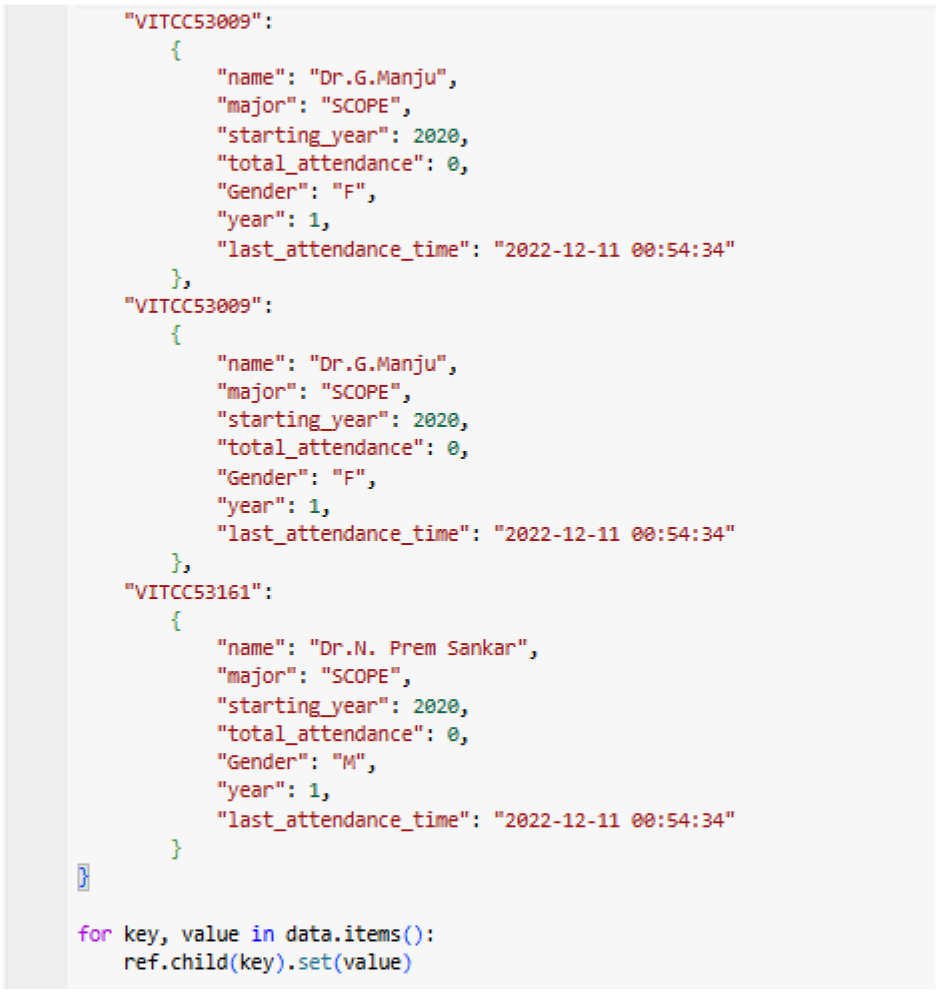
```
End Of Program
```

ADVANCED ATTENDANCE TRACKING THROUGH FACIAL RECOGNITION TECHNOLOGY REALTIME APPLICATION:

Module 1: AddDataToDatabase.

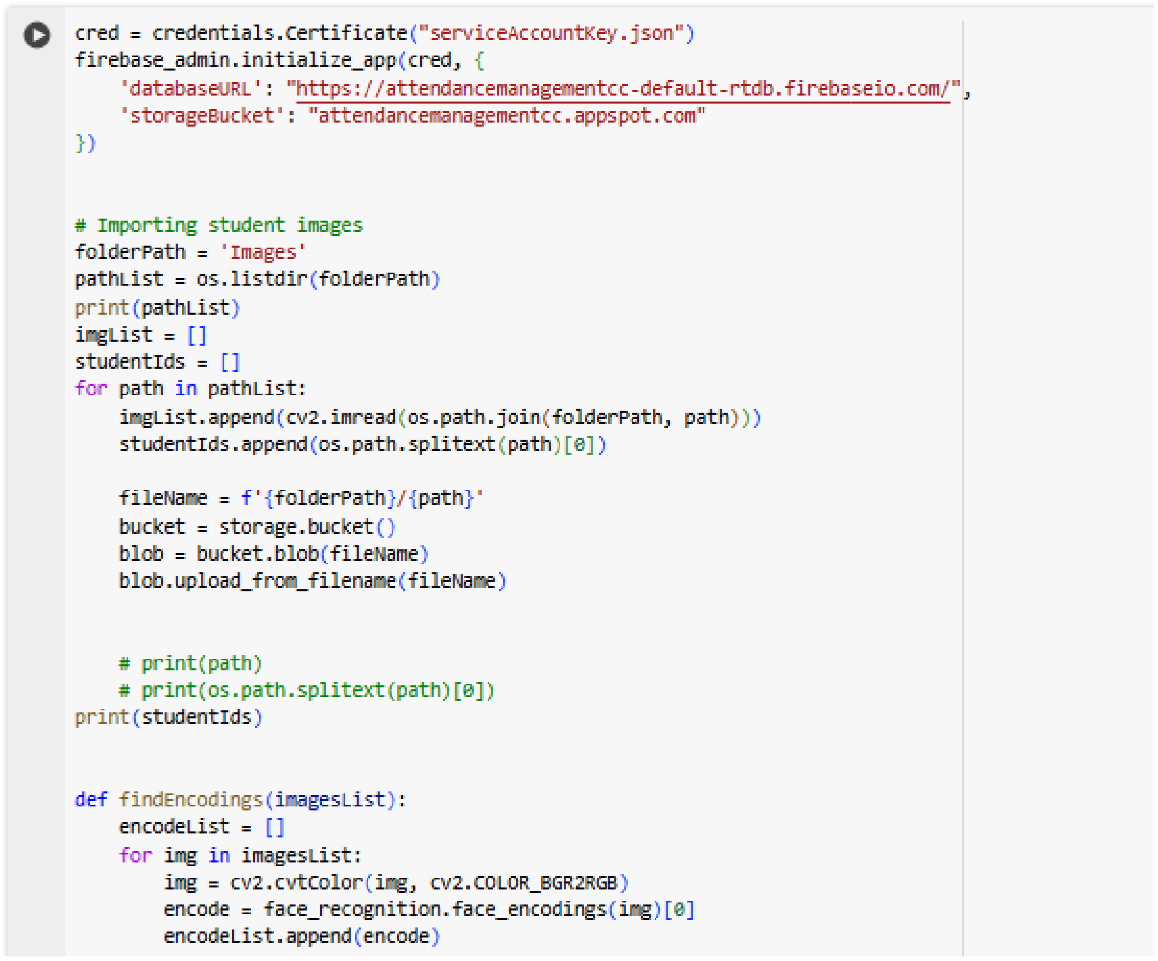


```
cred = credentials.Certificate("serviceAccountKey.json")
firebase_admin.initialize_app(cred, {
    'databaseURL': "https://attendancemanagementcc-default-rtdb.firebaseio.com/"
})

ref = db.reference('Students')

data = {
    "23MCA1041":
        {
            "name": "Pranav Sanjay Manapure",
            "major": "MCA",
            "starting_year": 2023,
            "total_attendance": 0,
            "Gender": "M",
            "year": 1,
            "last_attendance_time": "2022-12-11 00:54:34"
        },
    "VITCC52288":
        {
            "name": "Dr. Renjith PN",
            "major": "SCOPE",
            "starting_year": 2020,
            "total_attendance": 0,
            "Gender": "M",
            "year": 1,
            "last_attendance_time": "2022-12-11 00:54:34"
        },
    "VITCC53009":
        {
            "name": "Dr.G.Manju",
            "major": "SCOPE",
            "starting_year": 2020,
            "total_attendance": 0,
            "Gender": "F",
            "year": 1,
            "last_attendance_time": "2022-12-11 00:54:34"
        },
```

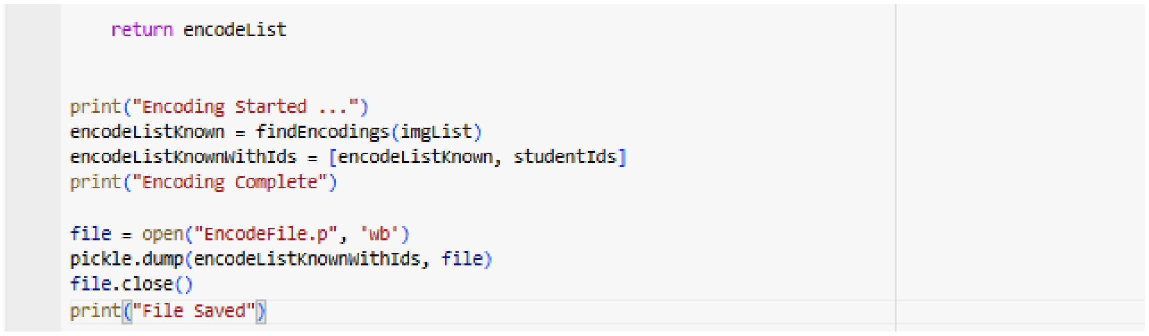


```
    "VITCC53009":
        {
            "name": "Dr.G.Manju",
            "major": "SCOPE",
            "starting_year": 2020,
            "total_attendance": 0,
            "Gender": "F",
            "year": 1,
            "last_attendance_time": "2022-12-11 00:54:34"
        },
    "VITCC53009":
        {
            "name": "Dr.G.Manju",
            "major": "SCOPE",
            "starting_year": 2020,
            "total_attendance": 0,
            "Gender": "F",
            "year": 1,
            "last_attendance_time": "2022-12-11 00:54:34"
        },
    "VITCC53161":
        {
            "name": "Dr.N. Prem Sankar",
            "major": "SCOPE",
            "starting_year": 2020,
            "total_attendance": 0,
            "Gender": "M",
            "year": 1,
            "last_attendance_time": "2022-12-11 00:54:34"
        }
}

for key, value in data.items():
    ref.child(key).set(value)
```

Module 2: Add pictures accordingly to there keys

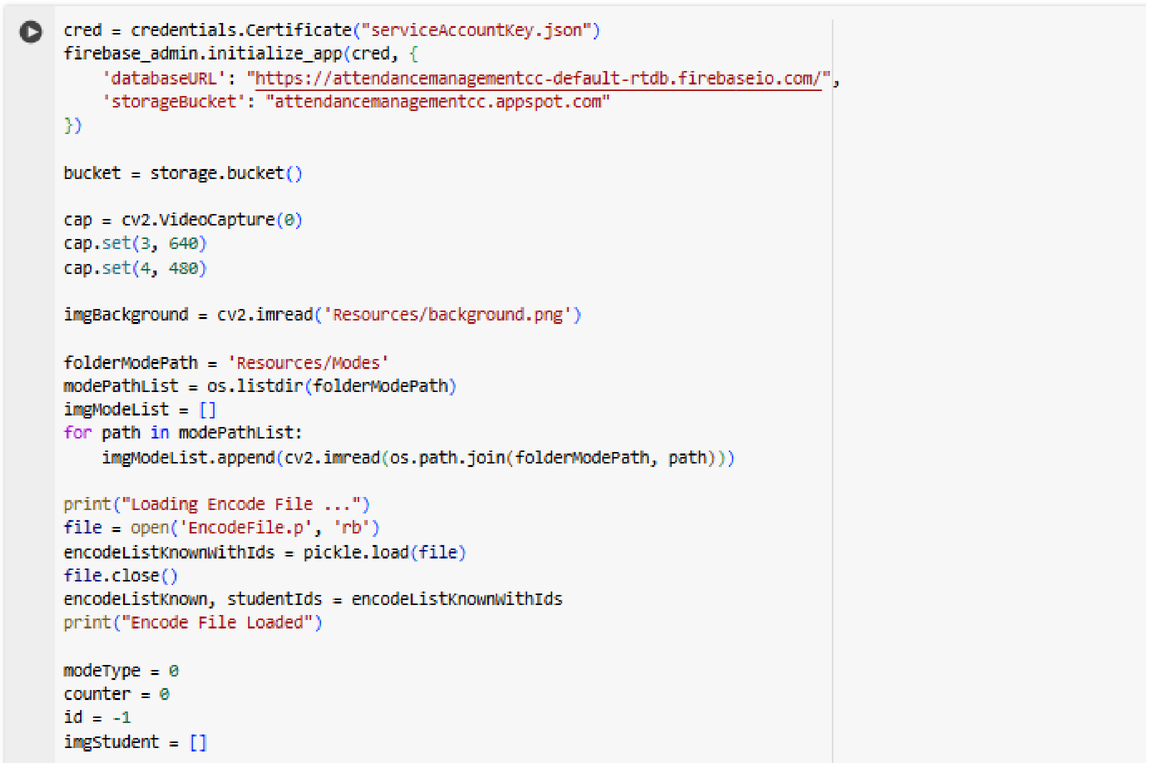


```
cred = credentials.Certificate("serviceAccountKey.json")
firebase_admin.initialize_app(cred, {
    'databaseURL': "https://attendancemanagementcc-default-rtdb.firebaseio.com/",
    'storageBucket': "attendancemanagementcc.appspot.com"
})


# Importing student images
folderPath = 'Images'
pathList = os.listdir(folderPath)
print(pathList)
imgList = []
studentIds = []
for path in pathList:
    imgList.append(cv2.imread(os.path.join(folderPath, path)))
    studentIds.append(os.path.splitext(path)[0])

    fileName = f'{folderPath}/{path}'
    bucket = storage.bucket()
    blob = bucket.blob(fileName)
    blob.upload_from_filename(fileName)


    # print(path)
    # print(os.path.splitext(path)[0])
print(studentIds)


def findEncodings(imagesList):
    encodeList = []
    for img in imagesList:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        encode = face_recognition.face_encodings(img)[0]
        encodeList.append(encode)
```

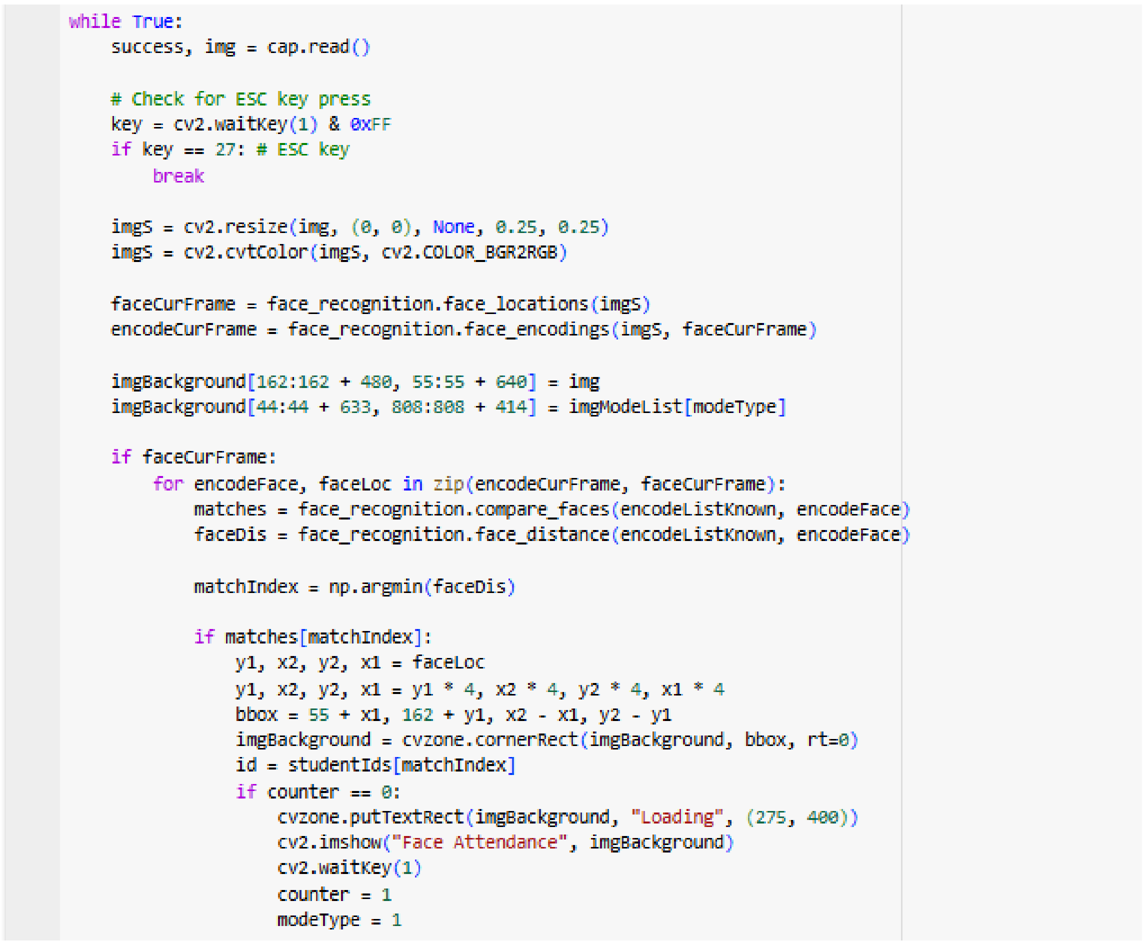


```
    return encodeList


print("Encoding Started ...")
encodeListKnown = findEncodings(imgList)
encodeListKnownWithIds = [encodeListKnown, studentIds]
print("Encoding Complete")

file = open("EncodeFile.p", 'wb')
pickle.dump(encodeListKnownWithIds, file)
file.close()
print("File Saved")
```

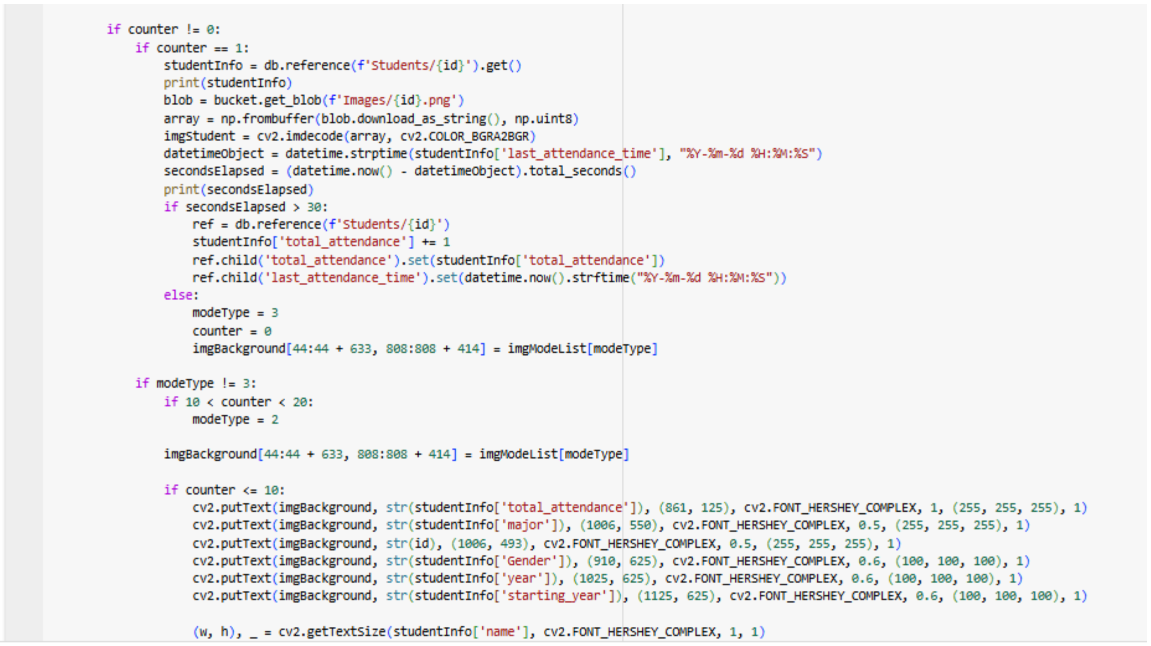
Module 3: Main Application Runner File



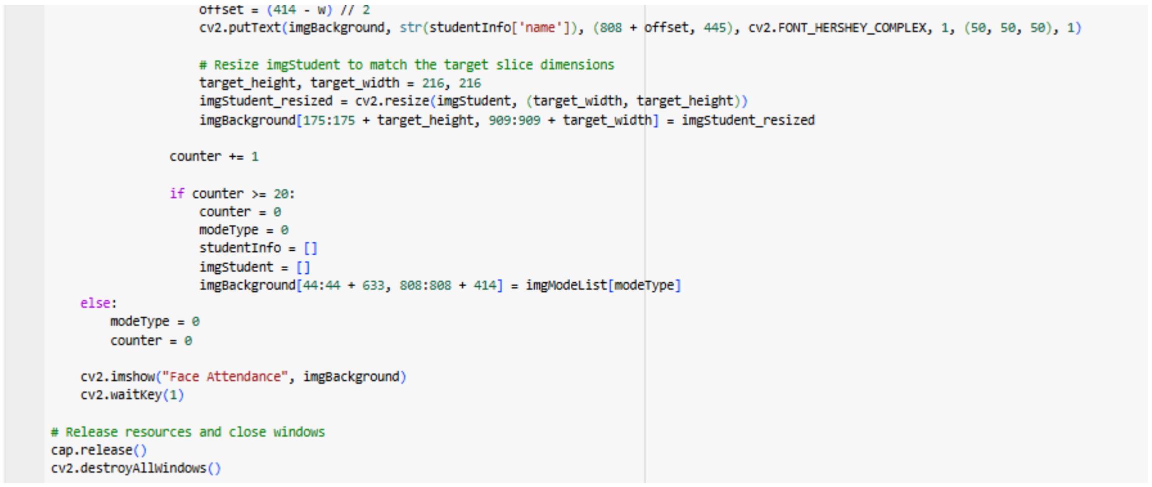
```
cred = credentials.Certificate("serviceAccountKey.json")
firebase_admin.initialize_app(cred, {
    'databaseURL': "https://attendancemanagementcc-default-rtdb.firebaseio.com/",
    'storageBucket': "attendancemanagementcc.appspot.com"
})

bucket = storage.bucket()

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

imgBackground = cv2.imread('Resources/background.png')

folderModePath = 'Resources/Modes'
modePathList = os.listdir(folderModePath)
imgModeList = []
for path in modePathList:
    imgModeList.append(cv2.imread(os.path.join(folderModePath, path)))

print("Loading Encode File ...")
file = open('EncodeFile.p', 'rb')
encodeListKnownWithIds = pickle.load(file)
file.close()
encodeListKnown, studentIds = encodeListKnownWithIds
print("Encode File Loaded")

modeType = 0
counter = 0
id = -1
imgStudent = []
```



```
while True:
    success, img = cap.read()

    # Check for ESC key press
    key = cv2.waitKey(1) & 0xFF
    if key == 27: # ESC key
        break

    imgS = cv2.resize(img, (0, 0), None, 0.25, 0.25)
    imgS = cv2.cvtColor(imgS, cv2.COLOR_BGR2RGB)

    faceCurFrame = face_recognition.face_locations(imgS)
    encodeCurFrame = face_recognition.face_encodings(imgS, faceCurFrame)

    imgBackground[162:162 + 480, 55:55 + 640] = img
    imgBackground[44:44 + 633, 808:808 + 414] = imgModeList[modeType]

    if faceCurFrame:
        for encodeFace, faceLoc in zip(encodeCurFrame, faceCurFrame):
            matches = face_recognition.compare_faces(encodeListKnown, encodeFace)
            faceDis = face_recognition.face_distance(encodeListKnown, encodeFace)

            matchIndex = np.argmin(faceDis)

            if matches[matchIndex]:
                y1, x2, y2, x1 = faceLoc
                y1, x2, y2, x1 = y1 * 4, x2 * 4, y2 * 4, x1 * 4
                bbox = 55 + x1, 162 + y1, x2 - x1, y2 - y1
                imgBackground = cvzone.cornerRect(imgBackground, bbox, rt=0)
                id = studentIds[matchIndex]
                if counter == 0:
                    cvzone.putTextRect(imgBackground, "Loading", (275, 400))
                    cv2.imshow("Face Attendance", imgBackground)
                    cv2.waitKey(1)
                    counter = 1
                    modeType = 1
```

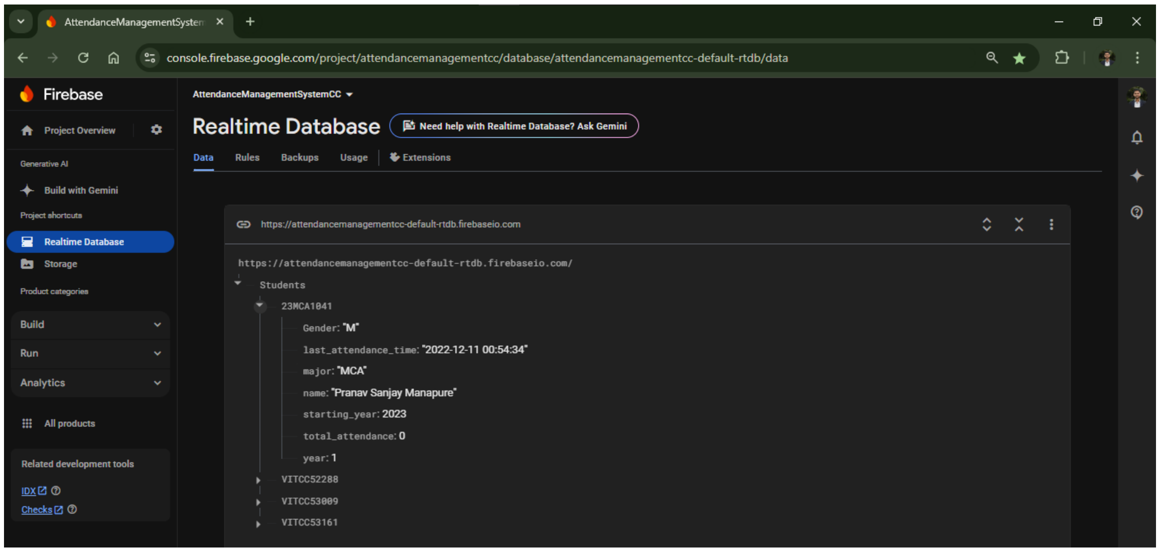


```
        if counter != 0:
            if counter == 1:
                studentInfo = db.reference(f'Students/{id}').get()
                print(studentInfo)
                blob = bucket.get_blob(f'Images/{id}.png')
                array = np.frombuffer(blob.download_as_string(), np.uint8)
                imgStudent = cv2.imdecode(array, cv2.COLOR_BGRA2BGR)
                datetimeObject = datetime.strptime(studentInfo['last_attendance_time'], "%Y-%m-%d %H:%M:%S")
                secondsElapsed = (datetime.now() - datetimeObject).total_seconds()
                print(secondsElapsed)
                if secondsElapsed > 30:
                    ref = db.reference(f'Students/{id}')
                    studentInfo['total_attendance'] += 1
                    ref.child('total_attendance').set(studentInfo['total_attendance'])
                    ref.child('last_attendance_time').set(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
                else:
                    modeType = 3
                    counter = 0
                    imgBackground[44:44 + 633, 808:808 + 414] = imgModeList[modeType]

            if modeType != 3:
                if 10 < counter < 20:
                    modeType = 2

                imgBackground[44:44 + 633, 808:808 + 414] = imgModeList[modeType]

                if counter <= 10:
                    cv2.putText(imgBackground, str(studentInfo['total_attendance']), (861, 125), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 1)
                    cv2.putText(imgBackground, str(studentInfo['major']), (1006, 550), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)
                    cv2.putText(imgBackground, str(id), (1006, 493), cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 1)
                    cv2.putText(imgBackground, str(studentInfo['Gender']), (910, 625), cv2.FONT_HERSHEY_COMPLEX, 0.6, (100, 100, 100), 1)
                    cv2.putText(imgBackground, str(studentInfo['year']), (1025, 625), cv2.FONT_HERSHEY_COMPLEX, 0.6, (100, 100, 100), 1)
                    cv2.putText(imgBackground, str(studentInfo['starting_year']), (1125, 625), cv2.FONT_HERSHEY_COMPLEX, 0.6, (100, 100, 100), 1)

                    (w, h), _ = cv2.getTextSize(studentInfo['name'], cv2.FONT_HERSHEY_COMPLEX, 1, 1)
```



```
                    offset = (414 - w) // 2
                    cv2.putText(imgBackground, str(studentInfo['name']), (808 + offset, 445), cv2.FONT_HERSHEY_COMPLEX, 1, (50, 50, 50), 1)

                    # Resize imgStudent to match the target slice dimensions
                    target_height, target_width = 216, 216
                    imgStudent_resized = cv2.resize(imgStudent, (target_width, target_height))
                    imgBackground[175:175 + target_height, 909:909 + target_width] = imgStudent_resized

                counter += 1

                if counter >= 20:
                    counter = 0
                    modeType = 0
                    studentInfo = []
                    imgStudent = []
                    imgBackground[44:44 + 633, 808:808 + 414] = imgModeList[modeType]
    else:
        modeType = 0
        counter = 0

    cv2.imshow("Face Attendance", imgBackground)
    cv2.waitKey(1)

# Release resources and close windows
cap.release()
cv2.destroyAllWindows()
```

Module 4: Realtime Firebase



console.firebase.google.com/project/attendancemanagementcc/database/attendancemanagementcc-default-rtdb/data

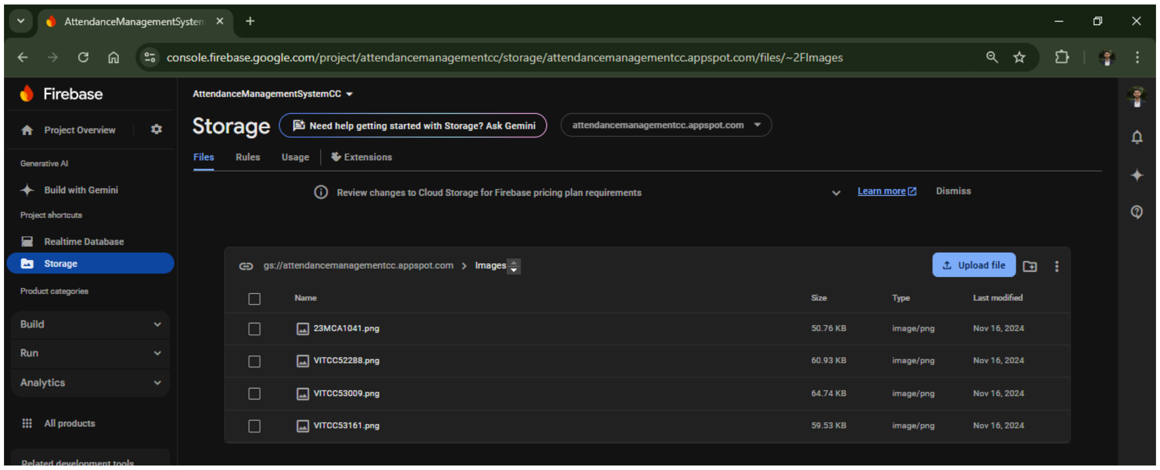
AttendanceManagementSystemCC

Realtime Database

Need help with Realtime Database? Ask Gemini

https://attendancemanagementcc-default-rtdb.firebaseio.com

```
https://attendancemanagementcc-default-rtdb.firebaseio.com/
  Students
    23MCA1041
      Gender: "M"
      last_attendance_time: "2022-12-11 00:54:34"
      major: "MCA"
      name: "Pranav Sanjay Manapure"
      starting_year: 2023
      total_attendance: 0
      year: 1
    VITCC52288
    VITCC53009
    VITCC53161
```



console.firebase.google.com/project/attendancemanagementcc/storage/attendancemanagementcc.appspot.com/files/~2FImages

AttendanceManagementSystemCC

Storage

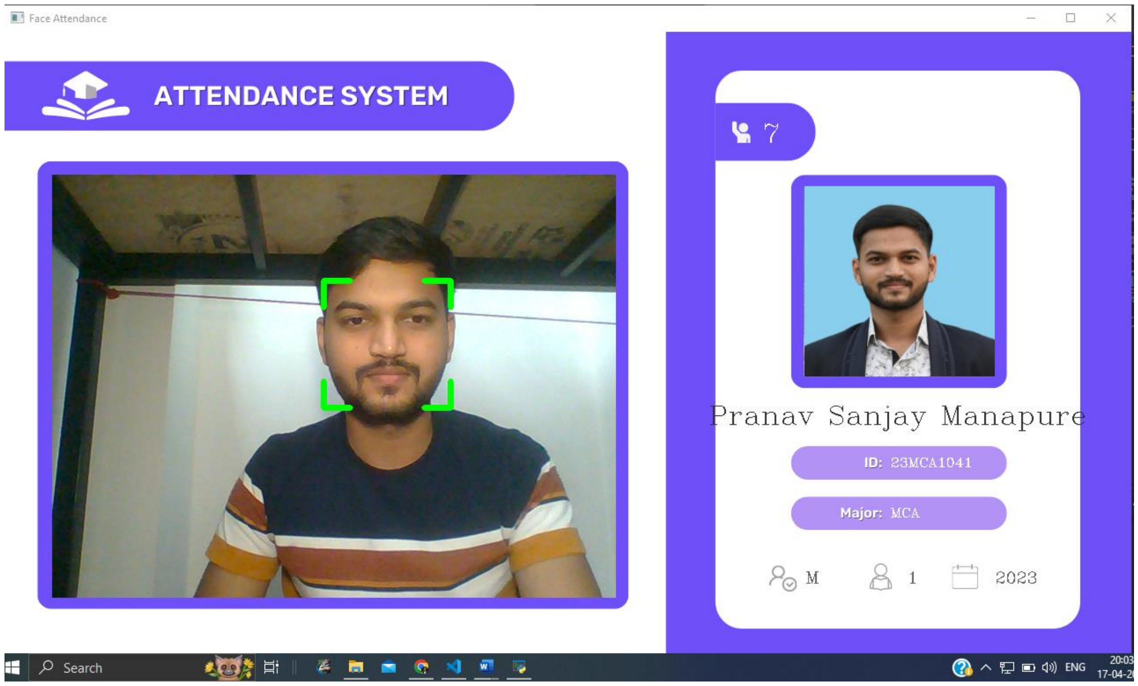
Need help getting started with Storage? Ask Gemini

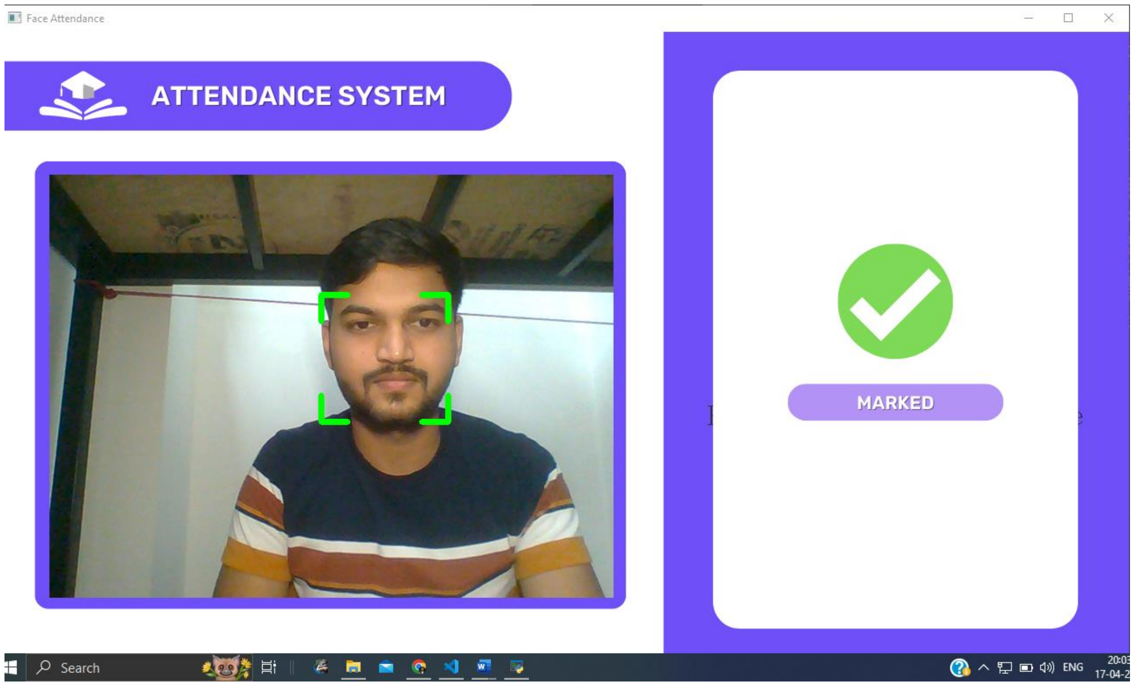
attendancemanagementcc.appspot.com

Review changes to Cloud Storage for Firebase pricing plan requirements

gs://attendancemanagementcc.appspot.com > Images

Name	Size	Type	Last modified
23MCA1041.png	50.76 KB	image/png	Nov 16, 2024
VITCC52288.png	60.93 KB	image/png	Nov 16, 2024
VITCC53009.png	64.74 KB	image/png	Nov 16, 2024
VITCC53161.png	59.53 KB	image/png	Nov 16, 2024


Face Attendance
ATTENDANCE SYSTEM
7
Pranav Sanjay Manapure
ID: 23MCA1041
Major: MCA
M
1
2023
Search
ENG
20:03
17-04-2


Face Attendance
ATTENDANCE SYSTEM
MARKED
Search
ENG
20:03
17-04-2

Chapter 8

Conclusion

8.1 CONCLUSION

In this thesis, we explored an advanced approach to attendance tracking through facial recognition technology, which combines efficiency with cutting-edge artificial intelligence. By implementing models such as FaceNet with K-Nearest Neighbors (KNN) for face recognition and employing a dataset-driven methodology, this project demonstrated how facial recognition can simplify and automate attendance systems. This technology significantly enhances convenience by allowing individuals to mark attendance simply by presenting their faces, which eliminates the need for cards or other identification tools. With the growing demand for secure and reliable attendance solutions in schools, universities, and workplaces, this system holds substantial promise.

The study's findings highlighted the model's ability to achieve reliable accuracy under various conditions, including different angles and lighting scenarios. Additionally, we addressed several challenges typically associated with facial recognition, such as high computational costs and difficulty in handling low-resolution images. By focusing on power efficiency and real-time processing, this project also offers a framework for developing sustainable and cost-effective attendance solutions, making it feasible for large-scale deployment. Our results suggest that facial recognition can meet modern organizational needs for efficient and secure attendance management.

While the proposed system is highly effective, it is essential to consider its limitations, particularly around privacy concerns and ethical implications. Ensuring the responsible use of facial recognition technology is vital, especially in settings where individuals' personal information is involved. Moving forward, it is necessary to adopt robust data privacy and security practices to safeguard users’ data and increase trust in such systems. Further research could explore improved encryption techniques, anonymization processes, and integration of compliance measures to align with data protection regulations.

In conclusion, this thesis demonstrated that facial recognition technology can serve as an innovative solution for attendance tracking, with the potential to transform attendance management across diverse fields. By refining the technology to address challenges such as privacy, computational efficiency, and adaptability to varied environments, facial recognition-based attendance systems can become a widely accepted tool for organizations. Future work could expand upon this research by incorporating more advanced algorithms, exploring 3D face recognition, and enhancing the system's adaptability to further increase its accuracy and robustness.

REFERENCES

[1] Andre Budimana, Fabian, Ricky Aryatama Yaputera, Said Achmad, Aditya Kurniawan, “Student attendance with face recognition (LBPH or CNN): Systematic literature review”, International Conference on Computer Science and Computational Intelligence, 2022

[2] Ahmad Anshari, Sulistyo Aris Hirtranusi, Dana Indra Sensuse, Kautsarina, Ryan Randy Suryono, “Face Recognition for Identification and Verification in Attendance System: A Systematic Review”, IEEE International Conference on Communication, Networks and Satellite (Comnetsat), 2021

[3] Ms. Sarika Ashok Sovitkar, Dr. Seema S. Kawathekar, “Comparative Study of Feature-based Algorithms and Classifiers in Face Recognition for Automated Attendance System”, International Conference on Innovative Mechanisms for Industry Applications, 2020

[4] HUDA.H.Mady, Shadi M.S. Hilles, “Efficient Real Time Attendance System Based on Face Detection Case Study “MEDIU Staff””, International Journal of Contemporary Computer Research, 2017

[5] Smitha, Pavithra S Hegde, Afshin, “Face Recognition based Attendance Management System”, International Journal of Engineering Research & Technology, 2020

[6] Dhanush Gowda H.L, K Vishal, Keertiraj B. R, Neha Kumari Dubey, Pooja M. R., “Face Recognition based Attendance System”, International Journal of Engineering Research & Technology, 2020

[7] Khawla Alhanaeea, Mitha Alhammadia, Nahla Almenhalia, Maad Shatnawia, “Face Recognition Smart Attendance System using Deep Transfer Learning”, International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, 2021

[8] Foteini P. Filippidou and George A. Papakostas, “Single Sample Face Recognition Using Convolutional Neural Networks for Automated Attendance Systems”, IEEE Xplore, 2020

[9] Dr A Manjula, D. Kalpana, Sanjay Guguloth, “Facial Recognition Attendance Monitoring System using Deep Learning Techniques”, International Journal For Innovative Engineering and Management Research, 2023

[10] Anju V Das, Anjana Shyju, and Thomas Varghese and Nisha Mohan P M, “Face Recognition Based Attendance Management System Using Machine Learning”, International Journal Of Innovative Research In Technology, 2019

[11] Aparna Trivedi, Chandan Mani Tripathi, Dr. Yusuf Perwej, Ashish Kumar Srivastava, Neha Kulshrestha, “Face Recognition Based Automated Attendance Management System”, International Journal of Scientific Research in Science and Technology, 2022

[12] Praveen K. Sah, Mamata Garanayak, Sujata Chakravarty, Bijay K. Paikaray, Rakesh Sharma and Suneeta Satpathy, “Student Attendance System Based on Face Recognition and Machine Learning”, Advances in Computation Intelligence, 2022

[13] Soumitra Chowdhury, Sudipta Nath, Ashim Dey and Annesha Das, “Development of an Automatic Class Attendance System using CNN-based Face Recognition”, Emerging Technology in Computing, Communication and Electronics, 2020

[14] Nandhini R, Duraimurugan N, S. P. Chokkalingam, “Face Recognition Based Attendance System”, International Journal of Engineering and Advanced Technology, 2019

[15] Anuj Singh, Nikhil Rawat, Rajan Kesri, “Face Recognition Based Attendance System”, International Journal for Multidisciplinary Research,