# REAL-TIME DEEP LEARNING-BASED FACE DETECTION AND RECOGNITION WITH INTEGRATED LIVENESS DETECTION FOR ATTENDANCE SYSTEM

BY

TAN YI XIN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

# COPYRIGHT STATEMENT



This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

# ACKNOWLEDGEMENTS

I would like to express thanks and appreciation to my supervisor, Dr Muhammad Husaini Bin Nadri and my moderator, Dr. Chai Tong Yuen who have given me a golden opportunity to involve on the Deep Learning and Computer Vision field study. Besides that, they have given me a lot of guidance in order to complete this project. When I was facing problems in this project, the advice from them always assists me in overcoming the problems. Again, a million thanks to my supervisor and moderator.

# ABSTRACT

This project presents the design and development of a real-time facial recognition attendance system aimed at automating and enhancing student attendance tracking in academic settings. Leveraging advancements in Artificial Intelligence and Computer Vision, the system integrates a deep learning-based Convolutional Neural Network (CNN) that generates 1024-dimensional facial embeddings for each registered user. These embeddings are used for identity verification through cosine similarity matching, achieving reliable and high-accuracy face recognition. To address security vulnerabilities such as spoofing and proxy attendance, the system incorporates active liveness detection mechanisms, including blink detection and head movement analysis, ensuring that only live human faces are authenticated. The front-end interface enables students to register their facial data and perform attendance scanning with minimal user interaction, while the web-based backend dashboard allows lecturers to manage class sections, enroll students, and monitor attendance records. The overall system demonstrates robust performance in real-world scenarios, achieving face recognition high accuracy with consistently high precision, recall, and F1-score. SQLite is used for lightweight data storage, while the Flask framework supports the real-time backend operations. The modular architecture ensures extensibility for future improvements. While the prototype is effective for controlled environments, limitations such as dataset diversity, backend scalability, and mobile accessibility remain. Future work may focus on expanding dataset coverage, implementing a cross-platform mobile application, and upgrading to a cloud-based database for better scalability. Overall, this project serves as proof-of-concept for a secure, efficient, and deployable biometric attendance system that reduces manual effort and improves accountability in academic institutions.

Area of Study (Minimum 1 and Maximum 2): Artificial Intelligence, Computer Vision, Image Processing

Keywords (Minimum 5 and Maximum 10): Deep Learning, Liveness Detection, Image Processing, Face Recognition, Image Processing, Real-Time Processing

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| θ (pose) | A set of head pose angles (Pitch, Yaw, Roll) used for liveness verification. |
| E_live | The 1024-dimensional embedding vector generated from a live captured face. |
| E_stored | The pre-registered 1024-dimensional embedding vector stored in the database. |
| cos(θ) | Cosine Similarity, the metric used to measure similarity between two embeddings. |
| ‖E‖ | The L2 Norm (or Euclidean magnitude) of an embedding vector. |
| ≥ 0.90 | The Recognition Threshold; a cosine similarity score must meet or exceed this value to confirm an identity. |

# LIST OF ABBREVIATIONS

| | |
|---|---|
| API | Application Programming Interface |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| DBMS | Database Management System |
| EAR | Eye Aspect Ratio (A metric for blink detection in liveness verification) |
| FPS | Frames Per Second (A measure of real-time video processing performance) |
| FYP | Final Year Project |
| GPU | Graphics Processing Unit |
| HTML | HyperText Markup Language |
| JWT | JSON Web Token |
| MAR | Mouth Aspect Ratio (A metric for smile/mouth open detection in liveness verification) |
| MTCNN | Multi-task Cascaded Convolutional Network |
| PCA | Principal Component Analysis |
| RAM | Random Access Memory |
| RBAC | Role-Based Access Control |
| REST | Representative State Transfer |
| SQL | Structured Query Language |
| SVM | Support Vector Machine |

# Chapter 1
# Introduction

This chapter establishes the foundational context for an in-depth exploration into the domain of automated face recognition systems. As a field at the dynamic intersection of computer vision, pattern recognition, and artificial intelligence, face recognition has undergone a profound transformation in recent years. This introduction will delineate the fundamental concepts governing this technology, underscore the pivotal and transformative role of deep learning in its modern incarnation, and situate this research within the specific, high-stakes application of secure and automated attendance management. Furthermore, this chapter will formally define the research problem this project seeks to address, articulate its precise objectives, and clearly demarcate the scope and strategic direction of the work undertaken.

Face recognition is a sophisticated biometric modality used to algorithmically identify or verify an individual by analyzing their unique facial characteristics from a digital image or a video stream. The process is conventionally architected as a multi-stage pipeline. This pipeline begins with **face detection**, the task of locating and isolating one or more faces within a given image. Following detection, **face alignment** (or normalization) is performed to standardize the detected faces, correcting for variations in scale, in-plane rotation, and posing to present a consistent facial view to the subsequent stages. The core of the system is **featuring extraction**, where a specialized algorithm processes the aligned face to generate a compact, discriminative numerical vector known as a feature embedding. This embedding is designed to capture the essential identity-specific information on the face. Finally, **face classification or matching** occurs, where the extracted embedding is compared against a database of pre-computed embeddings of known individuals to determine the identity.

The rapid development of deep learning, and particularly the widespread adoption of **Convolutional Neural Networks (CNNs)**, has catalyzed a paradigm shift in the field, moving from handcrafted feature descriptors (like Local Binary Patterns or Haar-like features) to end-to-end learned representations. CNNs possess the remarkable ability to autonomously learn a rich hierarchy of features directly from pixel data—from simple edges and textures in early layers to complex facial parts and global structures in deeper layers. This capability

enables them to achieve unprecedented robustness against challenging real-world variations such as extreme changes in illumination, non-frontal poses, diverse facial expressions, and partial occlusions. Seminal architectures like **FaceNet**, which introduced an embedding-based learning approach using a **triplet loss function**, have set new benchmarks. The triplet loss methodology trains the network to map images of the same person to points that are close together in an N-dimensional Euclidean space, while simultaneously pushing points from different identities far apart, thus creating a highly discriminative feature space.

Despite these significant technological leaps, the practical deployment of face recognition systems, especially for security-critical applications like attendance management, reveals persistent challenges in reliability and security. Traditional systems are notoriously vulnerable to **presentation attacks** (or spoofing), where malicious actors use non-live artifacts such as high-resolution photographs, video replays on a screen, or even 3D masks to impersonate a legitimate user. To counteract this critical vulnerability, the integration of **liveness detection** has become an indispensable component. Liveness detection is a set of techniques designed to verify that the biometric being captured belongs to a live, physically present human being. By analyzing physiological signs like eye blinks, subtle head movements, or micro-texture patterns, these systems can effectively differentiate between a genuine face and a fraudulent artifact, thereby preventing unauthorized access or fraudulent attendance marking.

The key concepts that form the technical backbone of this project include CNN-based feature extraction for identity representation, advanced image preprocessing to enhance data quality, data augmentation to build model robustness, rigorous model training and evaluation protocols, real-time inference optimization for practical deployment, and the crucial integration of liveness detection for anti-spoofing. Together, these elements constitute a holistic approach to engineering, the secure and efficient attendance system developed in this work.

## 1.1 Problem Statement and Motivation

While face recognition technology has rapidly matured from traditional feature engineering to highly accurate CNN-driven deep learning models, its application in practical, uncontrolled environments like classrooms continues to present a set of unresolved challenges. This project is directly motivated by the need to address these specific shortcomings.

I. **Critical Flaws in Prevailing Attendance Systems like QR Code Scanning**

Many academic and corporate institutions have adopted QR code-based systems for attendance tracking, aiming for digitalization and efficiency. However, these systems are fundamentally flawed in their security model. They are acutely susceptible to proxy attendance, where a QR code can be easily captured via screenshot and electronically shared with an absent individual, who can then scan it from a remote location. This loophole undermines the very purpose of attendance tracking: verifying physical presence. Furthermore, this method often imposes a significant administrative burden on lecturers, who must generate, display, and manage the codes, and later manually verify or reconcile the digital records, detracting from their primary teaching responsibilities.

**II. Inadequacies of Existing Face Recognition System**

While face recognition offers a theoretically superior alternative, off-the-shelf or naive implementations often fail in real-world deployment due to two primary issues:

- **Performance Degradation in Unconstrained Environments**: Performance Degradation in Unconstrained Environments: Classroom settings are visually complex and dynamic. Factors such as variable and poor lighting (e.g., backlighting from windows, dim overhead lights), a wide range of facial angles as students move, and frequent occlusions (e.g., from glasses, face masks, or hair) can drastically reduce the accuracy of a recognition model not specifically trained to handle such intra-class variability.
- **Vulnerability to Presentation (Spoofing) Attacks**: A face recognition system without a robust anti-spoofing mechanism is incomplete from a security standpoint. It remains vulnerable to simple yet effective attacks where an individual can present a photo or a video of a registered student to the camera. For an attendance system,

this vulnerability is critical, as it reintroduces the possibility of proxy attendance, thereby negating the primary advantage over QR code systems.

III. **Absence of a Fully Integrated and Automated Workflow**

A significant gap in many existing systems is the lack of seamless end-to-end automation. Even systems with recognition capabilities often require manual steps, such as a lecturer initiating the scan, manually verifying the logged entries against the actual class list, or exporting data for processing. This lack of integration between recognition, liveness verification, and automatic record logging creates friction and reduces the overall efficiency and reliability of the system.

This project is motivated by the urgent need for a holistic solution that directly confronts these issues. By developing an integrated system that combines real-time, robust CNN-based face recognition with intelligent liveness detection, this work aims to create a secure, trustworthy, and fully automated attendance ecosystem. The goal is to eliminate the proxy attendance problem, fortify the system against spoofing attempts, and free educators from manual administrative tasks, thereby ensuring academic integrity and operational efficiency.

## 1.2 Objectives

The primary objective of this project is to design, implement, and evaluate a real-time, deep learning-based face recognition attendance system with integrated liveness detection, engineered to ensure secure, accurate, and fully automated attendance marking in classroom environments.

To achieve this overarching goal, the following specific objectives are defined:

1. **To develop a highly accurate, real-time face recognition model optimized for classroom conditions**. This involves selecting and fine-tuning a state-of-the-art CNN architecture to handle challenging variations in lighting, diverse facial expressions, non-frontal poses, and partial occlusions. This will be supported by a robust pipeline of image preprocessing and augmentation techniques designed to simulate real-world classroom scenarios.

2. **To design and integrate a robust liveness detection module for anti-spoofing**. This objective focuses on implementing a mechanism that can reliably distinguish between a live person and a presentation attack (e.g., a photo or video). The system will analyze physiological cues, such as eye-blinking patterns, to ensure that attendance can only be marked by individuals who are physically present at the time of scanning.

3. **To implement a fully automated attendance management system**. This requires building a backend infrastructure that seamlessly logs the identities of recognized and liveness-verified individuals into a persistent database. The system will provide an automated workflow, from real-time video capture to final record generation, thereby minimizing the need for manual intervention or verification by academic staff.

## 1.3 Project Scope and Direction

This project is scoped to encompass the complete lifecycle of designing, developing, and evaluating a proof-of-concept prototype for a real-time face recognition attendance system fortified with liveness detection. The primary delivery is a functional software system capable of capturing a live video feed, concurrently performing face detection, recognition, and liveness verification, and securely recording the verified attendance data.

The specific **scope of this project includes**:

- **Model Development**: The selection, fine-tuning, and training of an appropriate CNN architecture (e.g., a lightweight variant of FaceNet or MobileNet) for the core face recognition task.
- **Data Handling**: The implementation of image preprocessing techniques (e.g., histogram equalization, noise reduction) and a comprehensive data augmentation pipeline (e.g., random rotation, brightness adjustment, zoom) to enhance model generalization and robustness.
- **Liveness Detection**: The development and integration of a software-based liveness detection module, specifically designed to identify and thwart spoofing attempts using static photos or video replays by analyzing physiological signals like eye blinks.
- **System Integration**: The construction of a complete system architecture, including a backend server and a Flask-based web interface. This interface will provide a user-friendly dashboard for lecturers to manage attendance sessions, view real-time logs, and generate reports.

The project will deliberately exclude the following areas to maintain a clear focus on software-based deep learning solutions:

- **GPS-based Location Verification**: While useful for geofencing, this is considered an orthogonal security layer. The project will concentrate on solving the visual-based challenges of identity and liveness verification.
- **Hardware-Level Biometric Integration**: The system will not integrate specialized hardware such as 3D cameras or infrared sensors, focusing instead on a solution that is deployable using standard, commodity webcams to ensure accessibility and scalability.

## 1.4 Contributions

This project aims to make a significant and practical contribution to the field of intelligent attendance systems by delivering innovations across several key areas:

I. **A Robust CNN-Based Recognition Model for Uncontrolled Environments**
The primary contribution is a custom-trained and optimized CNN model specifically tailored for the challenges of real-world classroom environments. By fine-tuning on a curated dataset that includes variations in lighting, head orientation, and partial occlusions, this model will demonstrate superior real-time accuracy and resilience compared to generic, off-the-shelf recognition models.

II. **An Integrated Anti-Spoofing and Liveness Detection Pipeline**
This works pioneers the seamless integration of liveness detection directly into the real-time recognition workflow. By incorporating techniques such as eye-blink detection or subtle head movement analysis, the system introduces a critical layer of security that directly addresses the vulnerability of presentation attacks, ensuring that spoofing attempts using static media are effectively prevented. This integrated approach is a significant step beyond simple recognition-only systems.

III. **A Fully Automated, End-to-End Attendance Workflow**
The project will deliver a holistic system architecture that automates the entire process, from live video capture and identity verification to secure record-keeping. This automation significantly reduces the administrative workload on lecturers, eliminates human error in record management, and ensures the creation of a reliable, tamper-proof audit trail of attendance.

IV. **A Practical and Scalable Platform for Institutional Deployment**
Beyond a mere algorithmic implementation, this project will produce a practical platform. It includes a student-facing module for face registration and scanning, and a lecturer-facing web dashboard for monitoring, management, and reporting. This dual-interface design makes the system a viable and scalable solution, ready for pilot deployment in academic institutions.

In summary, this research provides a secure, accurate, and scalable real-time attendance system that leverages the power of deep learning and integrated liveness detection to overcome the critical limitations of both QR code-based methods and traditional face recognition systems. Its contributions have the potential to extend beyond the classroom, offering a blueprint for secure access control, workforce time and attendance monitoring, and other applications where verified physical presence is paramount.

## 1.5 Report Organization

This thesis is systematically organized into seven chapters, each designed to logically build upon the last, guiding the reader through the project's conception, methodology, implementation, and evaluation.

- **Chapter 1: Introduction** provides the foundational context for the study. It presents the problem statement, defines the research objectives, delineates the project's scope, highlights its significance and contributions, and offers this overview of the report's structure.
- **Chapter 2: Literature Review** conducts a critical analysis of existing research and technologies relevant to this project. It examines core technologies such as MTCNN for face detection and FaceNet for recognition, explores different approaches to liveness detection, and discusses the potential of hybrid systems like GPS geofencing. The chapter also reviews the limitations of current attendance systems and identifies the specific research gaps this project aims to fill.
- **Chapter 3: System Methodology and Approach** details the theoretical and practical framework of the proposed system. It presents high-level system architecture, outlines the specific use cases for both student and lecturer roles, and explains the mathematical and algorithmic foundations, including cosine similarity for feature matching and the Eye Aspect Ratio (EAR) for blink detection.
- **Chapter 4: System Design** provides a detailed blueprint of the system's architecture. This chapter specifies the interactions between different software components, presents the database schema for storing user embeddings and attendance records, and describes the image preprocessing pipeline used to enhance data quality before it is fed into the deep learning models.
- **Chapter 5: System Implementation** documents the translation of the design into a functional prototype. It specifies the hardware and software configurations used,

illustrates the operational workflows with diagrams, and transparently discusses the key technical challenges encountered during development and the strategies employed to resolve them.

- **Chapter 6: System Evaluation and Discussion** presents a rigorous validation of the system's performance. It uses quantitative metrics such as recognition accuracy, false acceptance/rejection rates, and processing latency to assess effectiveness. The chapter also describes real-world testing scenarios under various conditions and provides a comparative analysis of the results against the project's initial objectives.
- **Chapter 7: Conclusion and Recommendations** concludes the report by summarizing the key findings and achievements, including the final measured accuracy of the system. It reflects on the project's successes and limitations and proposes concrete recommendations for future work, such as the integration of 3D mask detection and the exploration of deployment on edge computing devices.

# Chapter 2

# Literature Review

This chapter establishes the theoretical and practical foundations for the proposed facial recognition attendance system. It begins by critically examining previous works on automated attendance systems, identifying their strengths and vulnerabilities. Following this, the review delves into the core technologies that underpin the proposed solution, including fundamental image processing techniques and an in-depth analysis of prominent face recognition methodologies. By synthesizing and critiquing existing literature, this chapter identifies the critical research gaps that this project aims to address.

## 2.1 Previous Works on Attendance Systems

The automation of attendance tracking has been approached through various technological paradigms, each presenting a trade-off between convenience and security.

### 2.1.1 Attendance System using QR code Scanning

A prevalent approach to digitizing attendance involves QR code technology. Masalha and Hirzallah [1] proposed a system where unique, session-specific QR codes are generated for each lecture. Students scan these codes using a dedicated mobile application to transmit their identity and session details to a central server, as illustrated in Figure 2.1.1.1.

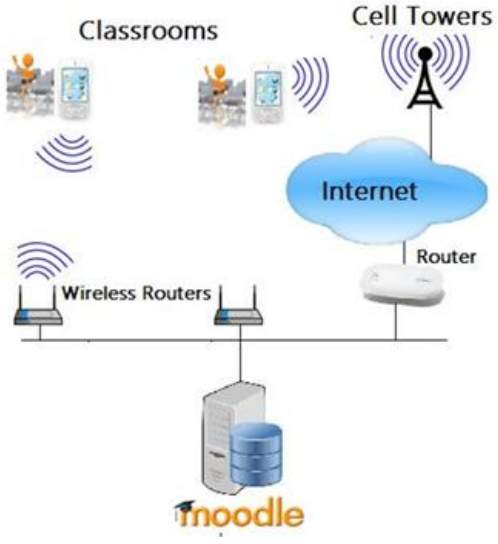



*Figure 2.1.1.1 Proposed System Infrastructure*

While this method offers significant improvements in speed and convenience over manual sign-in sheets, its primary weakness is a critical security flaw: **susceptibility to proxy attendance**. QR codes can be easily screenshotted and distributed, allowing a student who is not physically present to have their attendance marked by a peer. This fundamental vulnerability undermines the integrity of the attendance data, highlighting the necessity for a verification method that confirms both the user's physical presence and authentic identity.

### 2.1.2 Location Based time and attendance system

To address the challenge of verifying physical presence, researchers have explored location-based services. Uddin et al. [2] developed a system that uses the Global Positioning System (GPS) on student mobile devices to confirm their proximity to the lecture venue. Attendance is recorded only if the device is located within a predefined geographical boundary (geofence) during the scheduled class time as demonstrated in Figure 2.1.2.1.

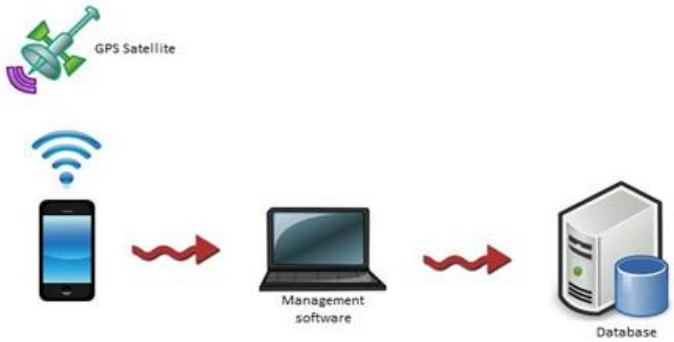



*Figure 2.1.2.1 Block Diagram of Location-based Time and Attendance System*

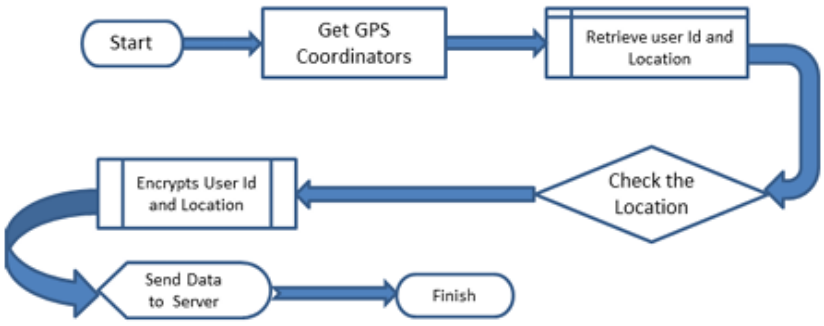



*Figure 2.1.2.2 Flows of operation for mobile application*

For the proposed **Management Software**:

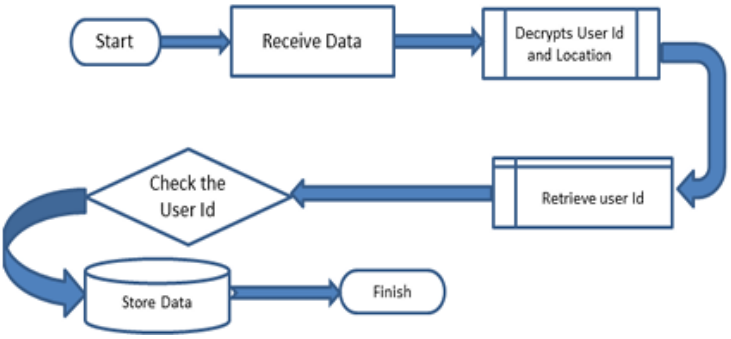



*Figure 2.1.2.3 Flows of operation for Time and Attendance Management Software*

Although this approach is an improvement over credential-based systems, it has significant limitations. GPS accuracy is often unreliable indoors, where most lectures occur. More critically, the system verifies the location of the *device*, not the *person*. This leaves it vulnerable to proxy attendance, as one student could bring multiple devices into the geofenced area. The absence of direct identity verification remains a key research gap that this project will address by using location data with robust biometric identification.

## 2.2 Previous work on Image Processing Techniques

According to N. Barnouti [3], Image preprocessing is a critical precursor to any successful computer vision task, as it standardizes images and enhances salient features required for accurate recognition. As noted by Anila and Devarajan [4], effective preprocessing reduces computational overhead and increases the probability of a correct match. Key techniques relevant to this project include:

### 2.2.1 Image Cropping

According to N. Barnouti [3], image cropping is a fundamental step in image pre-processing. The image cropping is used for isolating the region of an image where the face is located and discarding the unwanted background which can affect the recognition efficiency. This helps in the recognition process only held on the vital features of the face. The cropped images are then normalized to a standard size, such as 64x64 pixels, to ensure uniformity across the dataset. As illustrated in Figure 2.2.1.1, the cropped region typically includes key facial landmarks like the eyes and mouth, which are crucial for accurate recognition.

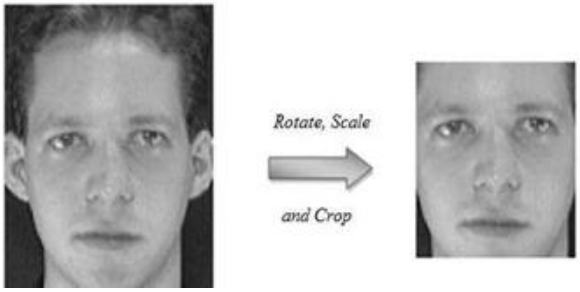



*Figure 2.2.1.1 Image Cropping to 64 x 64 pixels*

### 2.2.2 Image Resizing

The images used in this analysis were resized to various dimensions to investigate how different scales affect the recognition process. Since image size can influence the information content, a detailed examination was conducted to determine the optimal resizing scale. While the primary purpose of image resizing is to reduce data size and consequently processing time [5], selecting an inappropriate scale can be detrimental. The resizing scale was randomly varied between 0.1 and 0.9, resulting in a range of image sizes. Figure 2.2.2.1 illustrates an example of resizing an image with a scale of 0.5.

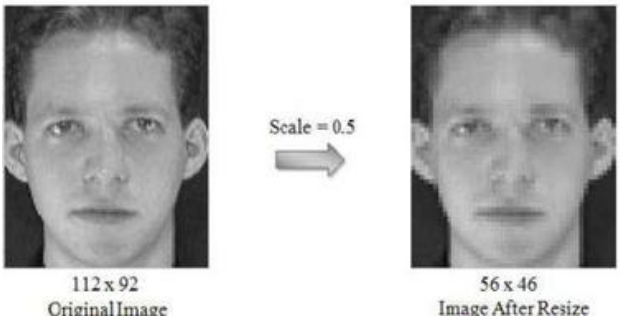



*Figure 2.2.2.1 Original Image Resize with Scale 0.5*

However, resizing images to an excessively small scale can lead to a significant loss of essential features, which is particularly problematic when image texture is important for classification. Figure 2.2.2.2 demonstrates the loss of features when an image is resized with a scale of 0.25.

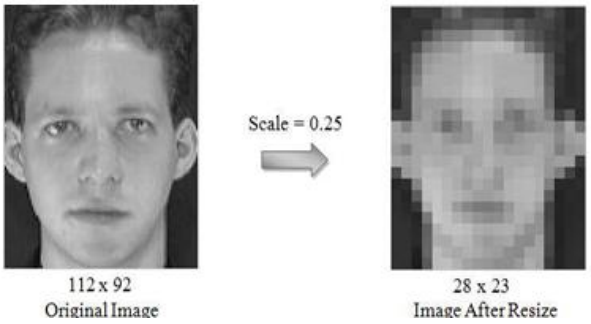



*Figure 2.2.2.2 Original Image Resize with Scale 0.25*

In this study, the image resizing scale ranged from 0.3 to 0.9. Resizing with a scale of 0.5 resulted in images of 56 x 46 pixels, while a scale of 0.3 produced images of 34 x 28 pixels. Figure 2.2.2.3 shows examples of images resized using a 0.3 scale, and Figure 2.2.2.4 illustrates the increase in the recognition rate observed after resizing images with this 0.3 scale. Therefore, choosing an appropriate resizing scale based on image resolution can be an efficient way to improve the recognition rate.

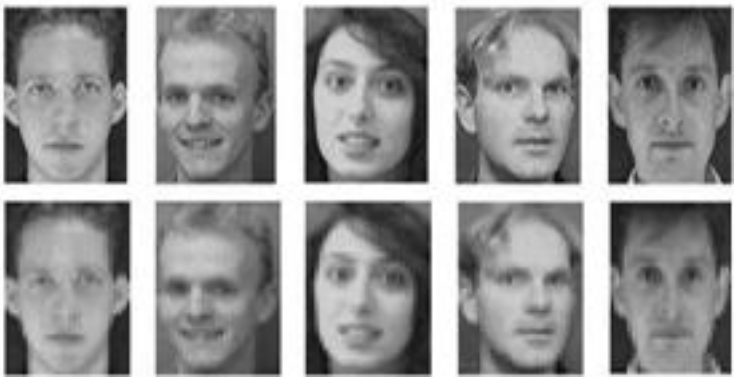


*Figure 2.2.2.3 Images After Resized by Scale 0.3*


*Figure 2.2.2.4 Recognition Rate After Resized by Scale 0.3*

### 2.2.3 Brightness Changing

A key image preprocessing step is brightness adjustment, which controls the overall lightness or darkness of an image. This is achieved by adding or subtracting a constant value from the intensity of each pixel. The range of this adjustment typically spans from -255 to +255; adding positive values brightens the image, while subtracting (negative values) darkens it. Figure 2.2.3.1 provides an example of increasing image brightness. Modifying the brightness level can be beneficial in various image analysis scenarios.


*Figure 2.2.3.1 Increase Image Brightness*

The contrast between the darkest and lightest areas and the blurriness of images can also be modified. In this analysis, increasing the image brightness by adding 100 or 140 to each pixel resulted in an improved recognition rate. Conversely, darkening the images by subtracting 100 or 140 from each pixel led to a decrease in the recognition rate. Figure 2.2.3.2 displays images after increasing the brightness by adding 140, and Figure 2.2.3.3 shows images after decreasing the brightness by subtracting 140. The corresponding recognition rates after these bright adjustments are presented in Figure 2.2.3.4. The results indicate that increasing the brightness

can enhance the recognition rate; while decreasing it is not an effective strategy and tends to reduce the recognition accuracy.

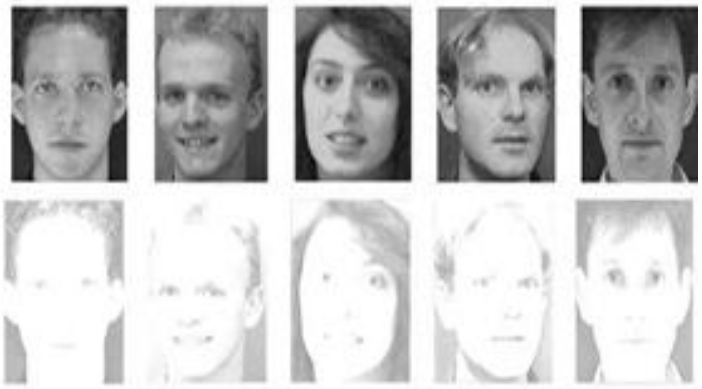


*Figure 2.2.3.2 Increase the brightness by adding 140 to each pixel.*

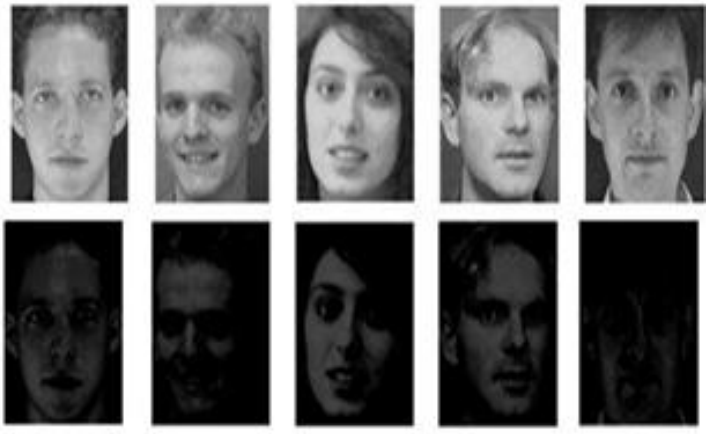


*Figure 2.2.3.3 Decrease the brightness by adding 140 to each pixel.*


| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Increase Brightness | 65.56 | 75.31 | 80.36 | 82.92 | 84.5 | 92.5 | 95 | 95 | 97.5 |
| Decrease Brightness | 47.5 | 57.81 | 56.43 | 65.83 | 70.5 | 74.38 | 75 | 78.75 | 80 |
| Origin | 62.22 | 71.25 | 72.86 | 77.92 | 80 | 91.25 | 93.33 | 95 | 95 |

*Figure 2.2.3.4 Recognition Rate After Increase and decrease the brightness.*

## 2.3 Previous works on Face Recognition

Face recognition has evolved from classical statistical methods to powerful deep learning architectures capable of handling complex, real-world variations.

Before a face can be recognized, it must first be accurately located within an image. Early methods like the Viola-Jones algorithm used Haar-like features and a cascade of classifiers to achieve real-time detection, though they were sensitive to non-frontal poses. Modern systems rely on deep learning, with the Multi-task Cascaded Convolutional Network (MTCNN) being a prominent example. MTCNN uses a cascade of three CNNs to progressively detect faces and their key landmarks (eyes, nose, mouth) with high accuracy, even under challenging conditions with varying poses and partial occlusions [13].

### 2.3.1 Eigenfaces and PCA (Principal Component Analysis)

One of the foundational methods in face recognition is Principal Component Analysis (PCA), used to implement the Eigenfaces approach [6], [7]. PCA is a dimensionality reduction technique that identifies the principal components that capture the most variance in a set of face images. These components, known as "eigenfaces," form a basis space on which new face images can be projected for comparison. While computationally efficient, the PCA-based Eigenfaces method is highly sensitive to variations in lighting, expression, and pose, making it less suitable for the uncontrolled environments of a typical classroom.

1. **Detecting Object as a Face**

   The system initiates by accessing a camera to detect objects resembling a face using the CASCADE_FIND_BIGGEST_OBJECT method. The input image is converted to grayscale and resized to accelerate the detection process. Contrast and brightness are enhanced through histogram equalization, and the detected face is then isolated and stored for subsequent processing as shown in Figure 2.3.1.1.

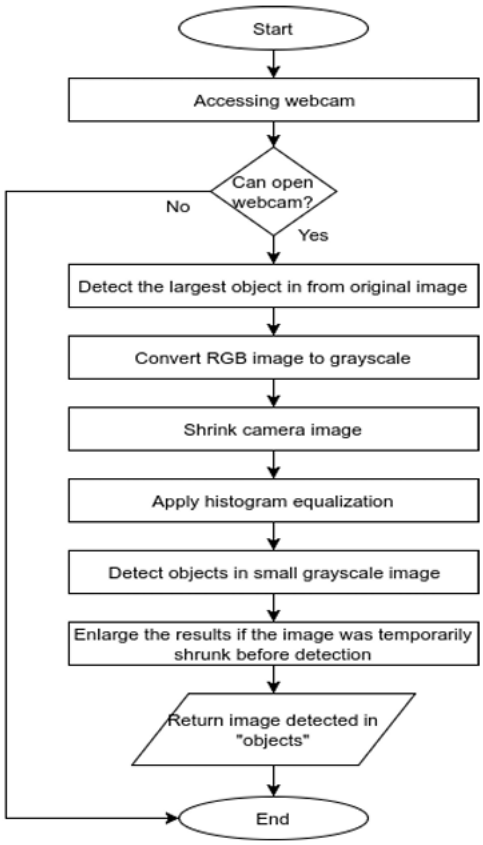



*Figure 2.3.1.1 Flowchart of detecting objects as a face*

**2. Preprocess of detected face**

To prepare the detected face for recognition and minimize errors, preprocessing steps are applied. This includes aligning the face by detecting and adjusting the position of the eyes using geometric transformations (rotation, scaling, translation). Histogram equalization is applied to each half of the face independently to standardize lighting, and a bilateral filter is used to smooth the image while preserving important edges as illustrated by author in Figure 2.3.1.2.

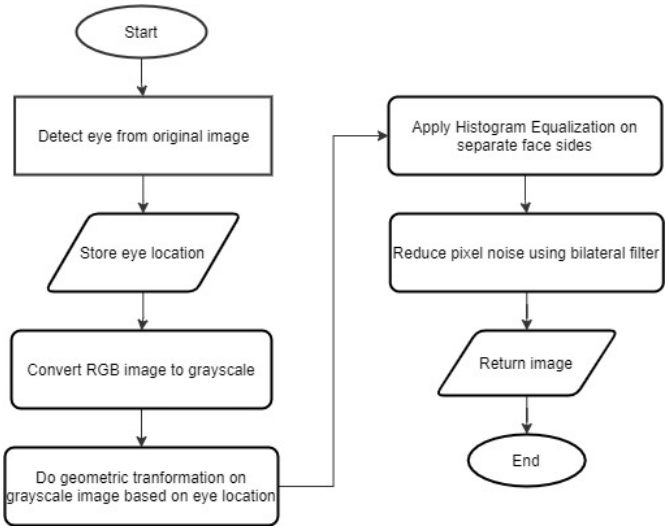



*Figure 2.3.1.2 Flowchart of preprocessing for detected face*

**3. Collect and train faces**

The system gathers multiple face images for training, incorporating variations by including mirrored versions of the original images. These images are stored in the .pgm format and then used to train a model using PCA. This process involves calculating the mean face, subtracting it from the training images, computing the covariance matrix, and determining the eigenvalues and eigenvectors. The eigenfaces are subsequently generated, and the resulting model is saved in .xml format (Figure 2.3.1.3). The formulas for calculating the average face, subtracting the mean, and finding the covariance matrix are provided, followed by the steps to determine eigenvalues, eigenvectors, eigen images, and the weight matrix. The eigenface concept is explained as a set of eigenvectors used in computer vision to recognize human faces, derived from the covariance matrix to represent the probability distribution and vector space for face recognition by reconstructing faces.

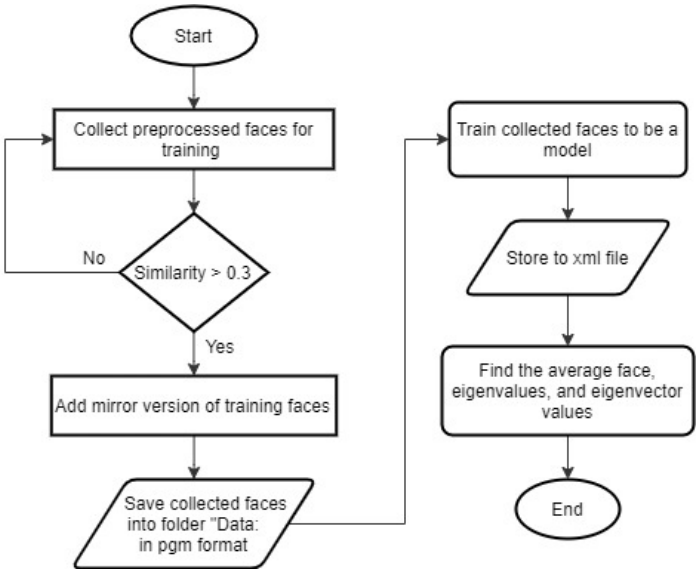



*Figure 2.3.1.3 Flowchart of collect and train the faces*

**4. Recognition**

During recognition, the trained model is loaded, and a face is detected in real-time using the camera. The preprocessed face is then compared to the model by projecting it into the eigenface space. If the similarity score is below a predefined threshold of 0.5 [7], the system identifies the face; otherwise, it is classified as "Unknown" (Figure 2.3.1.4).

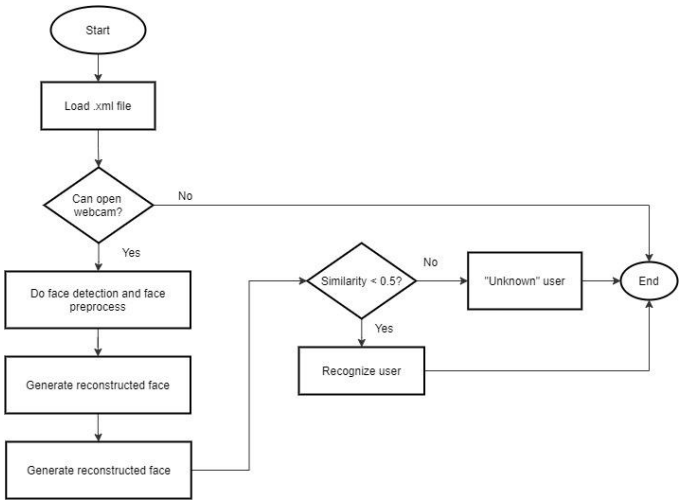



*Figure 2.3.1.4 Flowchart of face recognition*

The accuracy of the developed face recognition system was tested using a dataset of 6 training users, each with 40 variations of images. Figure 2.3.1.5 shows the best 20 eigenface images from the database, and Figure 2.3.1.6 illustrates a reconstructed image from the database. Figure 2.3.1.7 presents the result of a face recognition test for "train user 1," and

Table 2.2.1.1 summarizes the testing results for all users. The average recognition accuracy achieved by the authors' proposed system was 96.3% (Figure 2.3.1.8).

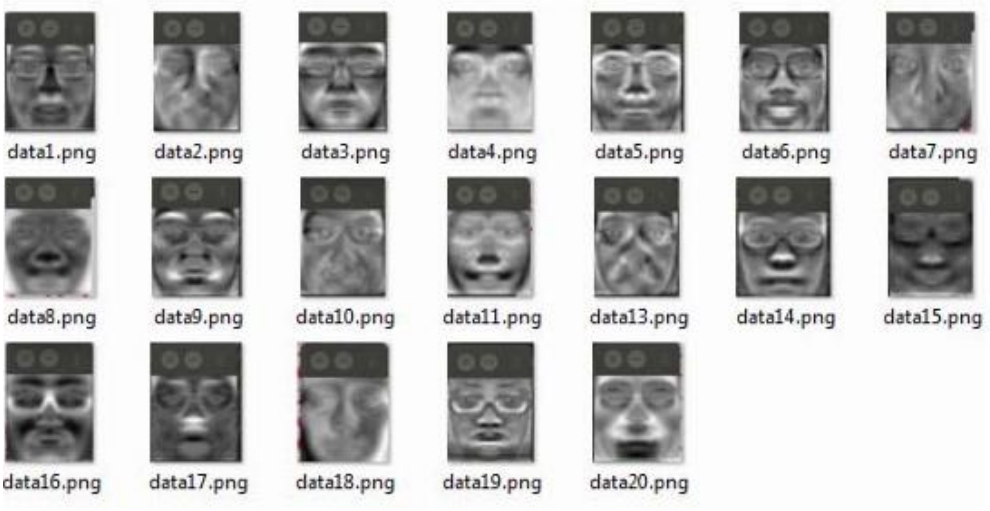



*Figure 2.3.1.5 Best 20 eigenface images from the database*

*Figure 2.3.1.6 Reconstructed image from one of the databases, "train user 1"*


*Figure 2.3.1.7 Result of face recognition testing (user: "train user 1")*

**The testing result summarize:**

*Table 2.3.1.1 Testing result summarization*

| No | Object | Recognized Times (%) |
|---|---|---|
| 1 | Train user 1 | 99 |
| 2 | Train user 2 | 97 |
| 3 | Train user 3 | 95 |
| 4 | Train user 4 | 97 |
| 5 | Train user 5 | 92 |
| 6 | Train user 6 | 98 |

The average result of the author's proposed recognition system is 96.3%.

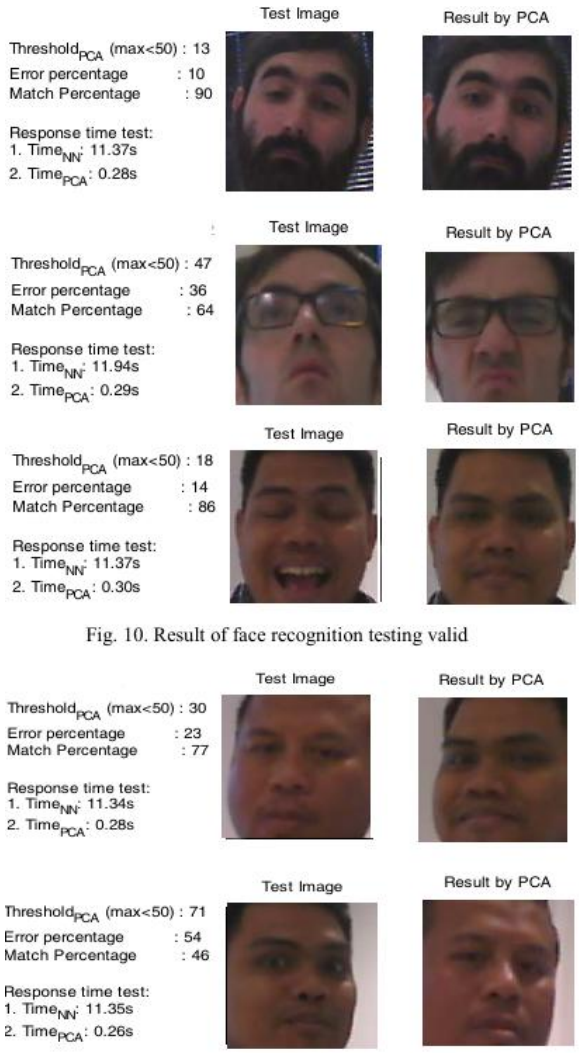



Fig. 10. Result of face recognition testing valid

*Figure 2.3.1.8 Average result of recognition system (PCA)*

Wagh et al. [7] also explored PCA algorithms, specifically Eigenfaces and Principal Component Analysis (PCA), for attendance systems. They noted that while algorithms like Neural Networks are effective for single-image systems, attendance systems require the recognition of multiple faces. Eigenfaces represent face images as a linear combination of weighted eigenvectors, referred to as "eigenfaces," aiming to reduce the dimensionality of face images by projecting them onto a lower-dimensional subspace that captures the most significant distinguishing features. PCA serves as the mathematical foundation of the Eigenfaces method, aiding in the identification of the principal components that capture the maximum variance in the data. The architecture of their proposed face recognition-based attendance system is shown in Figure 2.3.1.9. The authors outlined the hardware requirements, including a high-definition camera to capture the entire class. The captured image undergoes enhancement through grayscale conversion and histogram equalization before face detection algorithms identify individual faces. Each detected face is then cropped and compared against a database of student face images to mark attendance on a server. Table 2.2.1.2 provides a comparison of various algorithms for face recognition. The architecture of this face recognition-based attendance system proposed by the authors:

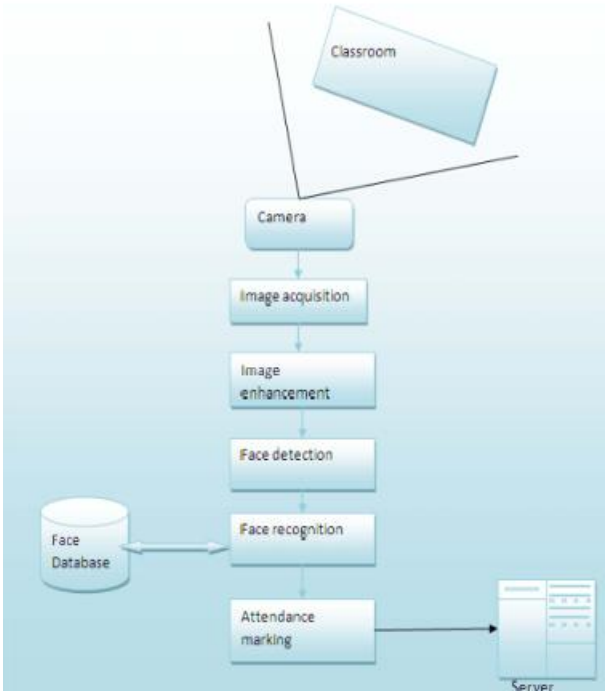



*Figure 2.3.1.9 Architecture of face recognition-based attendance system.*

*Table 2.3.1.2 Comparison of various algorithms for face recognition*

| Method | No. of Images | Success Rate (%) |
|---|---|---|
| Principal Component Analysis (PCA) | 400 | 79.65 |
| Principal Component Analysis + Relevant Component Analysis | 400 | 92.34 |
| Independent Component Analysis | 40 | Gauss function 81.35 |
| Support Vector Machines | - | 85-92.1 |
| Neural Networks | - | 93.7 |
| Eigenfaces Method | 70 | 92-100 |
| Eigenfaces with PCA method | - | 92.30 |

### 2.3.2 Convolutional Neural Network (CNNs)

Arsenovic et al. [8] introduced "FaceTime," a deep learning-based face recognition system utilizing Convolutional Neural Networks (CNNs) for real-time attendance tracking. This system offers an efficient alternative to traditional manual methods. Their study demonstrated the robustness of CNNs in handling variations in facial features, lighting, and angles, making it a reliable option for educational institutions and organizations. The system's implementation in academic settings showed promising results, indicating its potential for wider adoption. The block diagram proposed by the authors outlines the steps in their face recognition process (Figure 2.3.2.1).

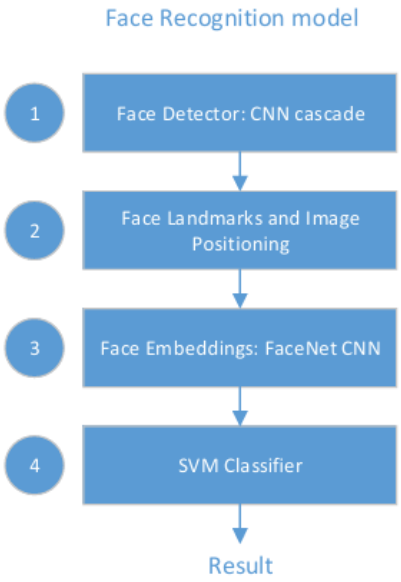



Figure 2.3.2.1 *Deep Learning-Based Face Recognition Block Diagram*

**Step 1: Face Detection**

The initial stage involves identifying the presence and location of faces within an image or video frame. The authors in [9] highlight the significant advancements of CNNs in both image classification and object detection, making them highly effective for face detection tasks.

**Step 2: Face Landmarks and Image Positioning**

Once a face is detected, the system proceeds to pinpoint key facial landmarks (e.g., eyes, nose, mouth) and determines the face's precise position and orientation. Arsenovic et al. utilized a sophisticated CNN-based face detector developed by Li et al. [11], which consists of a cascade of six interconnected CNNs for robust and accurate face localization. This face detector, built using the Torch [12] framework, forms the front-end of their recognition pipeline.

**Step 3: Face Embedding**

To create a unique representation for each face, the system employs FaceNet, a deep CNN architecture. FaceNet learns to map face images into a compact 128-dimensional Euclidean space, generating a 128-byte embedding for each face. This embedding serves as a distinctive "fingerprint" for the individual, where similar faces produce close embeddings, and dissimilar faces produce distant embeddings. The training of FaceNet utilizes a triplet loss function, comparing images of the same person with images of different individuals.

**Step 4: Classification**

The final step involves identifying the individual based on their generated face embedding. Arsenovic et al. trained a classifier, specifically a Support Vector Machine (SVM), on the embeddings extracted from a dataset of known individuals. To evaluate their system, they integrated it as a Face Recognition API into an existing RFID-based employee attendance system. The system continuously analyzed video feeds, detected faces, generated embeddings, and used the trained SVM classifier to predict the identity of the person. The predicted identity, along with a confidence score, the captured image, and timestamp, were stored in a database for analysis and comparison with the RFID data.

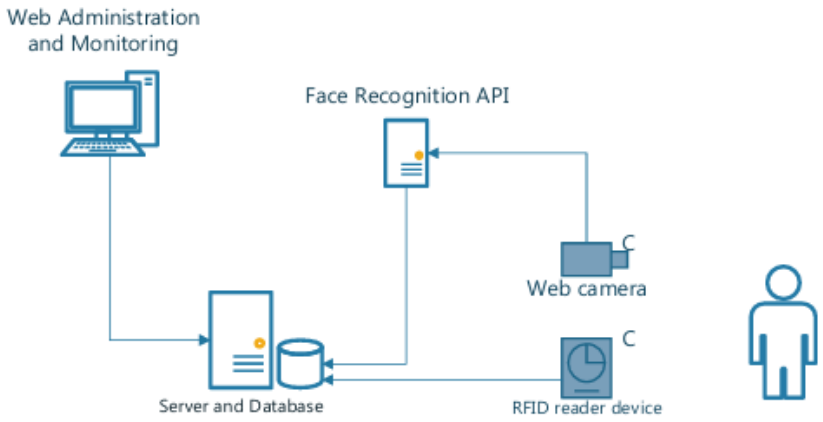



Figure 2.3.2.2 *Face Recognition API and current RFID-based system*

The effectiveness of the integrated face recognition system was assessed over a three-month period by tracking employee entrances and exits. The recorded data was subsequently validated against the existing RFID card records. The prediction outcomes, including a confusion matrix, are provided in Table 2.2.2.1, and the per-class accuracy of the deep learning model is shown in Figure 2.3.2.3. Notably, the system proposed by Arsenovic et al. [10] demonstrated a high overall accuracy of 95.02%. This level of performance was achieved by training the model on a relatively small number of images per employee, leveraging a data augmentation strategy to enhance the training data.

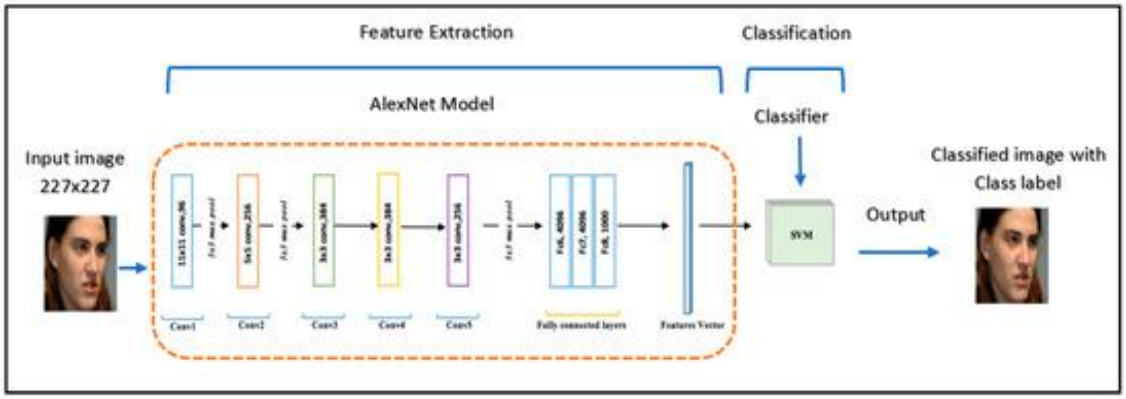



*Figure 2.3.2.3 CNN Architecture*

*Table 2.3.2.1 Deep Learning-Based Face Recognition Result Confusion Matrix*

| Classes | | | | | |
|---|---|---|---|---|---|
| *Empl 1* | *Empl 2* | *Empl 3* | *Empl 4* | *Empl 5* | *Predictions* |
| 230 | 8 | 0 | 6 | 1 | Empl 1 |
| 4 | 269 | 0 | 3 | 1 | Empl 2 |
| 0 | 0 | 301 | 0 | 3 | Empl 3 |
| 8 | 4 | 9 | 138 | 0 | Empl 4 |
| 2 | 5 | 6 | 1 | 227 | Empl 5 |

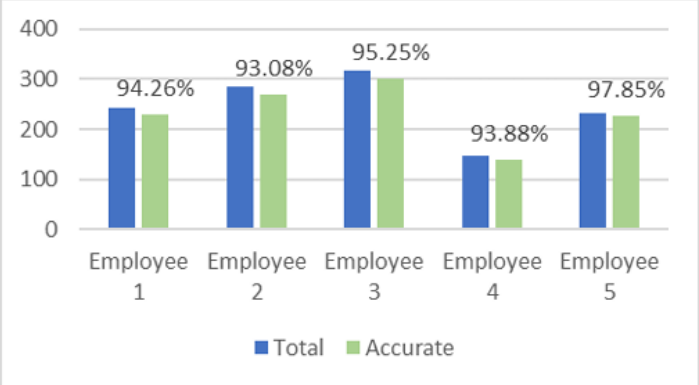



*Figure 2.3.2.4 Deep Learning-Based Face Recognition Accuracy per class*

The use of Convolutional Neural Networks (CNNs) enables real-time face detection and recognition. The authors also enhanced system capabilities without a complete infrastructure overhaul by integrating the face recognition API with an existing RFID-based attendance system. However, the authors noted that the proposed model's accuracy could be significantly impacted by lighting conditions, particularly images captured in daylight with open windows, which led to a higher rate of incorrect predictions. Additionally, the performance of the CNN-based face recognition model heavily relies on the quality and diversity of the training data, with limited or biased data potentially leading to reduced accuracy and generalization issues.

On the other hand, the authors suggested that applying gradient transformation to the images could help mitigate lighting issues by adjusting brightness and contrast, thereby making the model more resilient to variations in lighting conditions. Furthermore, implementing automatic retraining of the deep CNN model at regular intervals using newly captured images with high prediction accuracy could continuously improve the model's performance and allow it to adapt to new variations and conditions.

### 2.3.3 Siamese Networks

Siamese Networks are a class of neural network architecture that employs two or more identical subnetworks to compute output vectors for input pairs. These output vectors are then compared to measure the similarity or dissimilarity between the inputs. In the context of face recognition, Siamese Networks are particularly useful for verifying whether two face images belong to the same individual.

The core idea behind Siamese Networks is to learn a function that maps face images into a feature space where images of the same person are close to each other, while images of different people are far apart. This is achieved by training the subnetworks to produce similar embeddings (vector representations) for matching pairs and dissimilar embeddings for non-matching pairs.

Several studies have explored the effectiveness of Siamese Networks in addressing specific challenges in face recognition. Song et al. (2019) [15] focused on the issue of occlusion, a significant problem in real-world scenarios where faces may be partially obscured by objects like masks or scarves. Their work, "Occlusion Robust Face Recognition Based on Mask Learning with Pairwise Differential Siamese Network," proposed a novel approach to mitigate the impact of occlusions. The authors introduced a "Pairwise Differential Siamese Network" that learns to compare face images in a way that is less sensitive to occluded regions. The network incorporates a "mask learning" component, potentially learning to identify or down-weight the obscured parts of the face during the comparison process. This approach demonstrates the potential of Siamese Networks to enhance the robustness of face recognition systems in challenging conditions. The proposed system by the author [15] is shown as

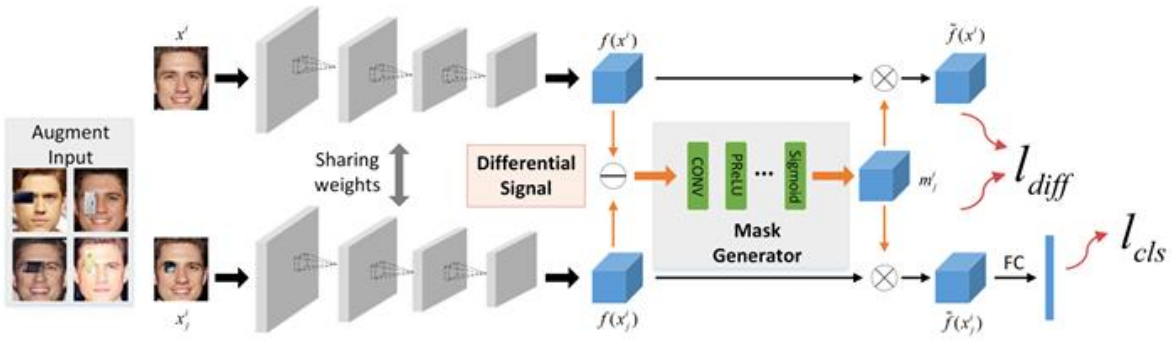



*Figure 2.3.3.1 Proposed Pairwise Differential Siamese Network*

Heidari and Fouladi-Ghaleh (2020) [16] addressed another critical challenge: face recognition with small-samples datasets. Their paper, "Using Siamese Networks with Transfer Learning for Face Recognition on Small-Samples Datasets," explored the combination of Siamese Networks with transfer learning. In many practical applications, including the initial enrollment phase of an attendance system, the number of available images per individual may be limited. The authors leveraged pre-trained weights from a model trained on a large, general dataset (likely a large-scale image dataset) and fine-tuned the Siamese network on a smaller, task-specific face recognition dataset. This transfer learning strategy enables the network to learn effective face representations even with limited data per identity. Their work highlights the ability of Siamese Networks to learn robust similarity metrics from small datasets, making them well-suited for scenarios where collecting many images per user is impractical.

## 2.4 Limitation of Previous Studies

While prior research has advanced the field of face recognition, several critical limitations reduce their suitability for real-world attendance management.

Firstly, many studies remain **restricted to controlled conditions**, evaluating performance only under uniform lighting, neutral facial expressions, and frontal poses. In classroom environments, however, faces are often captured at non-frontal angles, in low or fluctuating lighting, and with dynamic expressions, all of which degrade recognition performance.

Secondly, existing systems are frequently built on **datasets lacking demographic diversity**. Limited representation of different ethnicities, facial structures, or age groups often leads to biased recognition models. Such imbalances can result in unfair attendance outcomes, particularly in multicultural educational environments.

Another major gap lies in the **handling of occlusions and natural variations**. While some studies focus only on frontal, unobstructed faces, students may appear with masks, glasses, or partially hidden by hair and hands. Systems not designed for these conditions experience significant drops in recognition accuracy. Although methods such as Siamese Networks (Song et al., 2019) attempt to mitigate occlusion issues, these solutions remain limited in generalizability.

Finally, many existing studies overlook **robust liveness verification**. Without integrated countermeasures, spoofing attacks—such as presenting photos, videos, or even 3D masks—can be used to bypass recognition systems. This makes them vulnerable to **proxy attendance**, a critical concern in academic contexts where trust and authenticity are paramount.

## 2.5 Summary of previous studies

Face Recognition Techniques:

*Table 2.5.1 Comparison between recognition techniques*

| Papers | Method | Advantages | Disadvantages |
|---|---|---|---|
| [6], [7] | PCA with Eigenfaces | **Dimensionality reduction**: compresses data while preserving critical features. **Simplicity**: straightforward to implement. **Recognition**: captures principal components for classification. | **Linear assumption**: struggles with nonlinear variations. **Poor generalization**: sensitive to lighting, pose, and expression changes. **Preprocessing needed**: requires alignment and normalization. |
| [8]–[11] | Convolutional Neural Networks | **Automatic feature learning** extracts complex features directly from raw data. **High accuracy**: outperforms traditional methods in face recognition. **Robustness**: more tolerant to variations in pose, lighting, and expression. | **High resource demand** requires powerful GPUs and large datasets. **Complexity**: needs extensive hyperparameter tuning and longer training. **Data dependency**: performance degrades with limited training data. |
| [15], [16] | Siamese Networks | **Few-shot learning** works with limited labeled data. **Effective verification**: measures similarity instead of classification. **Flexible**: less dependent on large datasets. | **Pairwise, training requirement** needs balanced positive/negative pairs. **Sensitive to data quality**: poor pair selection reduces accuracy. **Training complexity**: more complicated than classification-based CNNs. |

In summary, among the reviewed approaches, **CNN-based models** stand out as the most effective due to their robustness, adaptability, and superior accuracy in handling real-world conditions. Hence, the proposed system adopts a deep CNN with residual blocks to strengthen feature learning and ensure reliable performance in academic attendance applications.

## 2.6 Proposed Solutions

To address the shortcomings identified in traditional face recognition systems and previous attendance management solutions, this project proposes a **real-time facial recognition attendance system** that integrates deep learning-based recognition, robust liveness detection, and secure role-based access.

The system begins with a **face registration process**, where each student captures multiple facial images from a live video stream. This ensures the collection of diverse samples under varying poses and expressions, improving robustness and fairness.
For **face detection**, the system employs the **Dlib face detector**, which effectively locates faces and extracts aligned facial regions, even under moderate variations in pose and lighting.

For **feature extraction**, a custom-built **deep Convolutional Neural Network (CNN) with residual blocks** is implemented. Residual connections allow deeper networks to be trained without vanishing gradients, thereby improving the learning of complex facial patterns. The network generates **1024-dimensional facial embeddings**, representing unique identity features for each student.
**Identity verification** is conducted using **cosine similarity**, which measures the angular distance between embeddings. This metric provides a robust similarity measure, less affected by illumination or scale variations, ensuring reliable recognition in diverse conditions.

To prevent spoofing and proxy attendance, the system integrates **multi-factor liveness detection** during both registration and recognition. The liveness module evaluates:

- **Eye aspect ratio (EAR)** for blink detection.
- **Mouth ratio (MAR)** for lip or mouth movements.
- **Head pose estimation** to detect natural head rotations.
- **Texture and motion cues** to distinguish live faces from printed photos or screens.

This multi-factor approach significantly enhances resistance against spoofing attacks.
For **system management**, the platform includes **role-based access control (RBAC)**. Lecturers can manage courses, create class sections, and generate attendance reports, while students are restricted to scanning attendance and viewing personal attendance records.

The backend leverages a **centralized SQLite database** that stores facial embeddings, attendance logs, and user information. This ensures **real-time synchronization** and supports analytical functions such as attendance rate monitoring and reporting.

Finally, the system is deployed as a **Flask web application**, with a responsive interface optimized for both desktop and mobile devices. This guarantees accessibility and usability across different platforms, making the solution practical for academic institutions.

# Chapter 3

# System Methodology/Approach OR System Model

This chapter outlines the methodology adopted to develop the real-time face detection and recognition attendance system using deep learning models with liveness detection techniques.

## 3.1 System Design Diagram/Equation

### 3.1.1 System Architecture Diagram

The system architecture in Figure 3.1.1.1 illustrates the full data flow from user login to attendance marking. It is divided into three primary components:

1. User Authentication and Face Registration
2. Liveness Detection (Active Challenge-Response)
3. Face Recognition and Attendance Marking

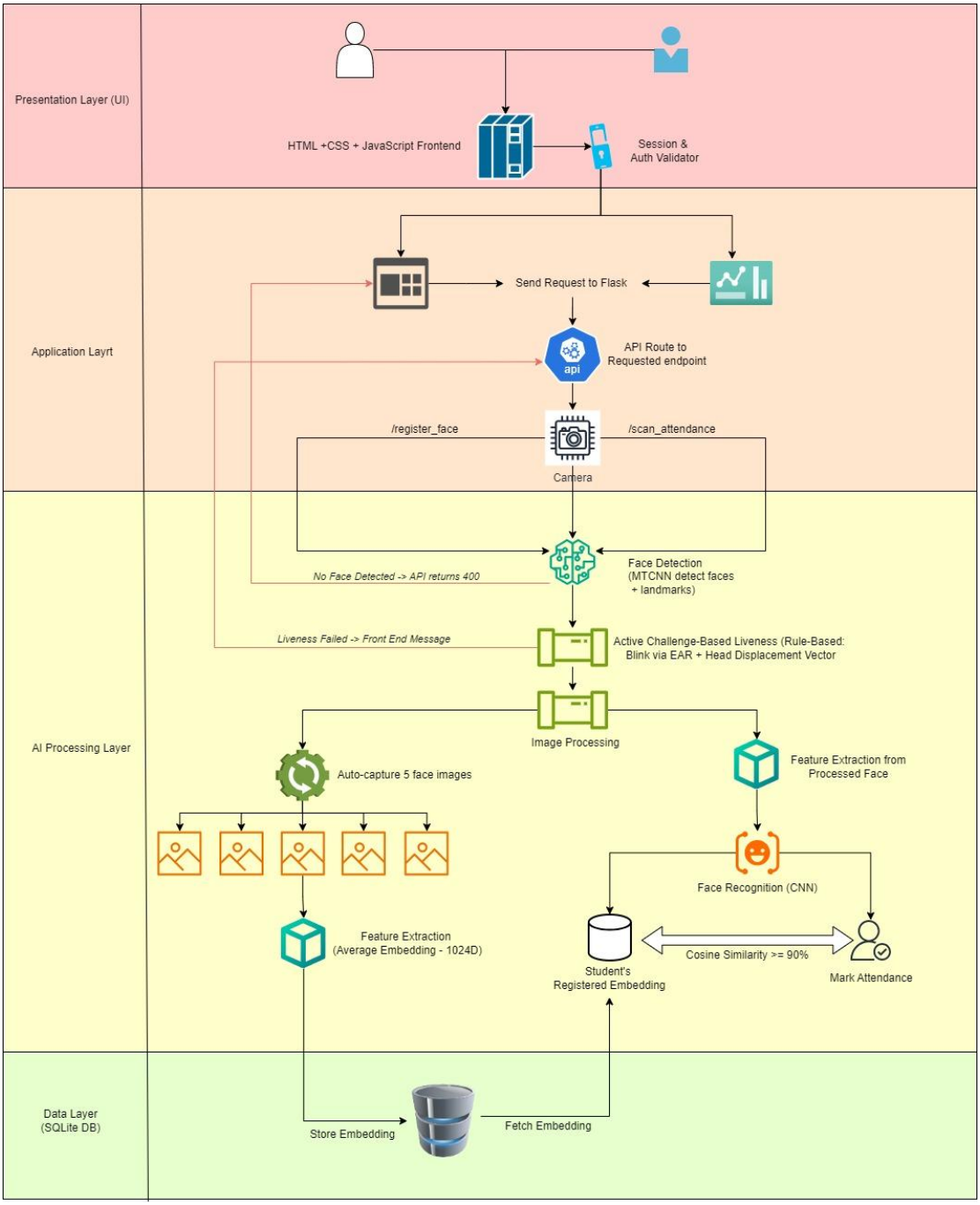



*Figure 3.1.1.1 System Architecture Diagram*

#### 3.1.1.1 User Authentication and Face Registration Flow

The system workflow begins with user authentication via the Flask application's secure login portal. Upon successful login, the system checks the user's biometric registration status. If no facial data exists, the user is seamlessly guided to the registration module, which activates the integrated camera.

Within the live video feed, a pre-trained **Dlib 68-point facial landmark detector** identifies key facial features, including the eyes, nose, mouth, and jawline. This landmark model is foundational to both face alignment and liveness detection. Once a stable face is detected with a confidence score exceeding 90%, the system automatically captures five high-quality images.

Each captured image undergoes a mandatory preprocessing pipeline to ensure standardization for the CNN model:

- **Geometric Alignment:** Facial landmarks are used to normalize the face's orientation, correcting for minor head tilts.
- **Image Resizing:** Dimensions are standardized to match the input requirements of the CNN model (e.g., 64x64 pixels).
- **Photometric Normalization:** Techniques like histogram equalization are applied to mitigate the effects of varying lighting conditions.

The preprocessed images are then fed into a custom-trained deep Convolutional Neural Network (CNN). This network functions as a feature extractor, transforming each facial image into a high-dimensional **1024-dimensional feature vector**, known as an embedding. To create a single, robust identity template that is resilient to minor variations, the five generated embeddings are mathematically averaged. This final average embedding is then securely stored in the database and uniquely associated with the user's profile.

### 3.1.1.2 Liveness Detection: Active Challenge-Response Module

To prevent spoofing attacks from non-live sources, such as static photographs or video replays, the system incorporates a critical active liveness detection stage. This module ensures the integrity of the attendance record by verifying the user's physical presence through a **challenge-response mechanism**, where the user is prompted to perform simple actions that a static source cannot replicate.

**A. Facial Landmark-Based Metric Analysis**

The foundation of the liveness detection system lies in the accurate detection and tracking of facial landmarks. These landmarks are specific points on a face. The system utilizes a 68-point facial landmark model from Dlib, which are the standard in many computer vision applications, to locate these key features in each frame.

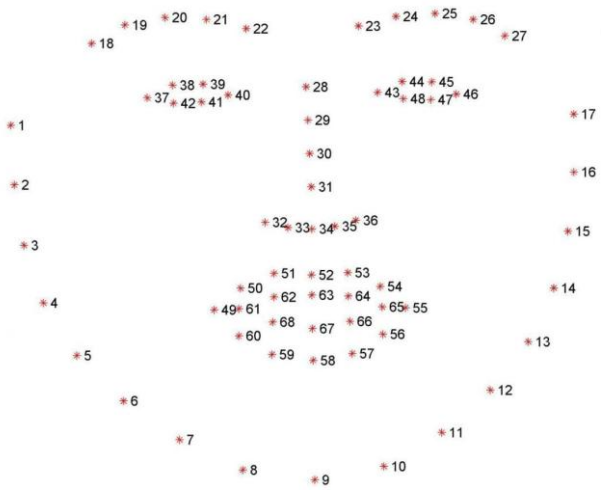



*Figure 3.1.1.2.1 Dlib 68-Facial Landmarks in a Face*

**I. Eye Aspect Ratio (EAR) for Blink Detection**

Blink detection is a widely accepted biometric signal indicating liveness. This system employs the **Eye Aspect Ratio (EAR)** metric, a method proposed by **Soukupová and Čech (2016)**. The EAR is computed from six facial landmarks surrounding each eye (typically detected using a 68-point facial landmark predictor).

We then find the start and end values of landmark ids for both the eye. You can do it manually also (37-42 for the right eye and 43-48 for the left eye) but using *face_utils* you can get these values by just passing the eye name.

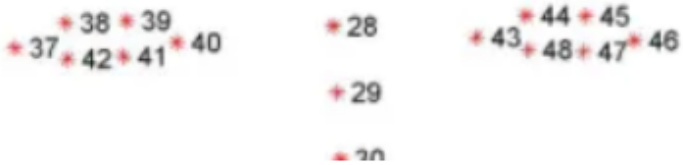



*Figure 3.1.1.2.2 Dlib Facial Landmarks*


*Figure 3.1.1.2.3 EAR for Open and Closed Eye*

The formula is given by:

$$EAR = \frac{\|p_2 - p_6\| + \|\|p_3 - p_5\|\|}{2\|p_1 - p_4\|}$$

Where:

- $p_1$, …, $p_6$ are the 2D coordinates of the eye landmarks.

A **sudden drop** in EAR followed by a **quick rebound** to baseline is interpreted as a valid blink. Each recognized blink **adds to the blink score**, which contributes to the final liveness score.

**II. Mouth Aspect Ratio (MAR) for Smile and Mouth Open Detection**

MAR is used to quantify the state of the mouth. This metric is essential for the 'smile' and 'open mouth' challenges. It is calculated using the landmarks that define the mouth (points 49 through 68). The system computes the ratio of the vertical distance between the upper and lower lips to the horizontal width of the mouth. The formula is:

$$MAR = \frac{\|p_{52} - p_{58}\| + \|\|p_{54} - p_{56}\|\|}{2\|p_{49} - p_{55}\|}$$

Where the points correspond to key vertical and horizontal positions on the lips.

- **Open Mouth Detection**: A significant increase in the MAR value indicates that the mouth is open.
- **Smile Detection**: A smile is characterized by widening of the mouth (increase in the horizontal distance) without a significant increase in the vertical opening.

**B. 3D Head Pose Estimation – Motion Detection:**

Natural micro-movements of the head are another strong indicator of liveness. A static photo cannot change its orientation, and a simple video replay often has limited, repetitive motion. The system estimates the head's 3D orientation—specifically its **pitch**, **yaw**, and **roll**—to verify that the user can perform specific head movements like nodding or turning.

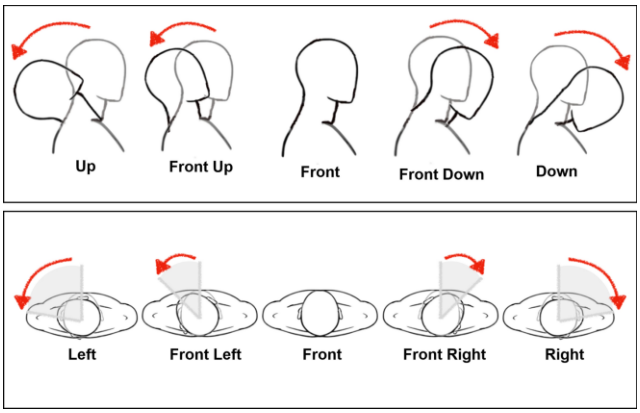



*Figure 3.1.1.2.4 Head Motion Classification*

Pitch (nodding "yes"), Yaw (shaking "no"), and Roll (side-to-side tilt) are the three degrees of freedom for head rotation.

The *calculate_head_pose* method implements this by using a 3D facial model and the *cv2.solvePnP* function. This process works as follows:

1. **Define a Generic 3D Face Model**: A standardized 3D model of key facial points (like the nose tip, chin, and eye corners) is defined.
2. **Map 2D Image Points**: The corresponding 2D landmark points detected in the current frame are identified.
3. **Solve for Pose**: The *cv2.solvePnP* ("Perspective-n-Point") algorithm finds rotation and translation that maps the 3D model points to the observed 2D image points.

This requires camera information, which is estimated based on the frame's dimensions.

4. **Calculate Angles**: The resulting rotation vector is converted into the more intuitive pitch, yaw, and roll angles.

These angles are then used in the "turn_left," "turn_right," and "nod" challenges. For example, to complete the "turn_left" challenge, the user's calculated yaw angle must exceed a predefined negative threshold *(-self.config.HEAD_MOVEMENT_RANGE*).

### C. Anti-Spoofing Heuristics: Texture and Motion Analysis

To further defend against spoofing attacks, such as displaying a video of a person on a high-resolution screen, two additional heuristics are employed: texture analysis and motion analysis.

#### I. Texture Analysis

A face captured live by a camera will have a different texture profile than a face displayed on a digital screen or a printed photograph. Screens have pixel grids, and photos can have moiré patterns or a lack of fine detail. The calculate_texture_score function analyzes the texture of the facial region. It applies a **Laplacian operator** to the grayscale image of the face and calculates the variance of the result. The Laplacian operator is sensitive to edges and fine details.

- A **high variance** suggests a sharp, detailed image, typical of a live face.
- A **low variance** suggests a blurry or uniform surface, which could indicate a spoof attempt (e.g., an out-of-focus photo or a smooth screen surface).

This texture score provides a continuous measure of authenticity that complements the other behavioral checks.

#### II. Motion Analysis

While head pose estimation tracks large-scale movements, calculate_motion_score detects subtle, frame-to-frame motion. It calculates the meaning of the absolute difference between the current and previous grayscale frames. Even a person trying to stay still will exhibit natural, small movements (from breathing, slight sways, etc.), which will result in a non-zero motion score. A complete lack of motion (a score of or near zero) is a strong indicator of a static image.

**D. The Challenge-Response Workflow and Liveness Decision**

The system's logic culminates in the challenge-response workflow, managed by the *analyze_frame* and *process_challenge* methods.

1. **Initiation**: When a face is detected, the system selects a random challenge from a predefined list (e.g., "blink_twice," "smile," "turn_left").
2. **Guidance and Progress**: The user is presented with an instruction. The system continuously processes incoming frames, calculating the relevant metrics (EAR, MAR, head pose). The progress towards completing the challenge is calculated and can be displayed to the user in real-time. For instance, if the challenge is to blink twice, the progress bar would fill by 50% after the first blink.
3. **Timeout**: To ensure the check is completed efficiently and to prevent indefinite stalling, each challenge has a timeout (*self.config.CHALLENGE_TIMEOUT*). If the user fails to complete the action within this window, the challenge fails, and a new one may be initiated.
4. **Completion and Decision**: Once the criteria for a challenge are met (e.g., *self.blink_count* >= 2), the challenge is marked as completed. The *analyze_frame* function then sets the *is_live* flag to *True* and the confidence to 1.0. This successful verification confirms the user's presence, allowing the primary action (e.g., marking attendance) to proceed.

This multi-faceted approach, combining physiological cues (blinking), voluntary actions (smiling, turning head), and anti-spoofing heuristics (texture analysis), creates a robust and user-friendly liveness detection system that is difficult to bypass with common spoofing techniques.

#### 3.1.1.3 Face Recognition and Attendance Marking Flow

Once face is detected and the liveness is confirmed, the system performs the final identity verification. The deep CNN model processes the live-verified face to extract its final, discriminative 1024-dimensional embedding, denoted as Elive. This embedding is then compared against the pre-registered templates, Estored, retrieved from the database.

The comparison metric is **Cosine Similarity**, which measures the cosine of the angle between the two embedding vectors in the high-dimensional space. A value closer to 1 indicates a higher degree of similarity. The formula is:

$$Similarity(live, registered) = \frac{E_{live} \cdot E_{registered}}{\|E_{live}\| \cdot \|E_{registered}\|}$$

Where:

- $E_{live} \cdot E_{registered}$ = dot product of the two vectors.
- $\|E_{live}\| \; and \; \|E_{live}\|$ are the L2 norms (magnitude) of the vectors.

If the similarity score meets or exceeds a strict threshold of **90% (0.90)**, the system confirms the student's identity. Upon successful recognition, the system finalizes the process by marking the student's attendance, logging the precise timestamp and associated location data into the attendance database. This final step creates a secure, non-repudiable audit trail for each attendance record.

## 3.2 Use Case Diagram and Description

### 3.2.1 Use Case Diagram - Student

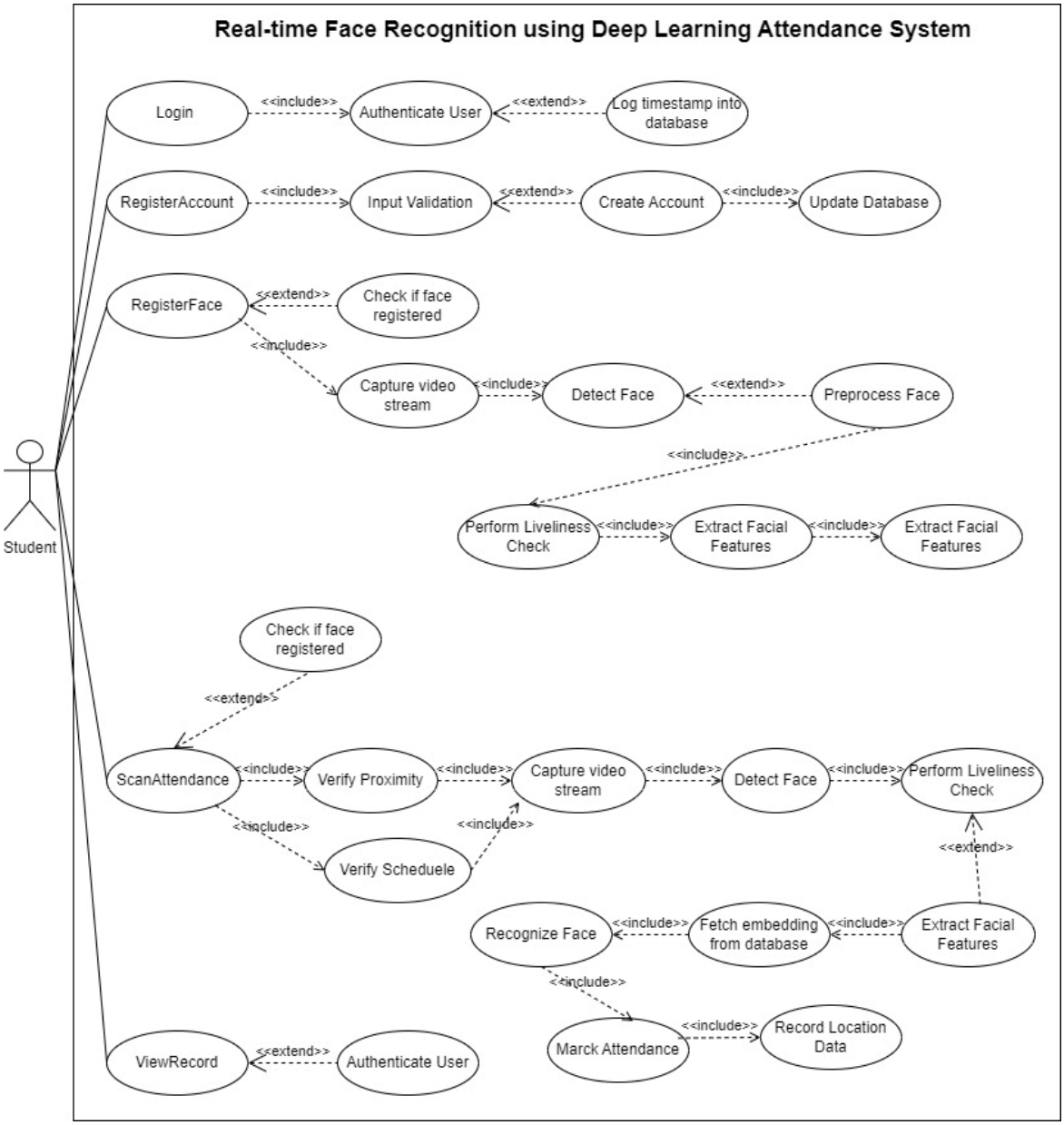



*Figure 3.2.1.1 Use Case Diagram for Student*

This diagram visually represents the interactions between the student actor and the Real-time Face Recognition Attendance System. It outlines the primary functions a student can perform, such as registering, logging in, managing their facial data, marking attendance, and viewing their records

**Student Use Case Description:**

Use Case 1: Login

*Table 3.2.1.1 Use Case Description (Login)*

| Field | Description |
|---|---|
| **Use Case Name** | Login |
| **ID** | UC01 |
| **Importance Level** | High |
| **Primary Actor** | Student |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | 1. Student: Secure access to their dashboard and attendance functions<br>2. System: Must verify identity against hashed passwords in the database and log access events. |
| **Brief Description** | The student logs into the system using their registered email and password. |
| **Trigger** | Student navigates to the login page and submits the login form. |
| **Relationships** | • Include: Authenticate User<br>• Extend: Log timestamp into database |
| **Normal Flow** | 1. Students enter their email and password.<br>2. The system hashes the entered password and compares it with the stored hash for the given email.<br>3. If valid, the system creates a user session and records a login timestamp.<br>4. Students are redirected to their personal dashboard. |
| **Sub flows** | None |
| **Alternate/Exceptional Flows** | 1a. Invalid credentials → Display "Invalid email or password" error message.<br>1b. System offline → Display “Service unavailable.”<br>1c. Account not yet verified (if applicable) → Display "Please verify your email address. |

Use Case 2: Register Account

*Table 3.2.1.2 Use Case Description (Register Account)*

| Field | Description |
|---|---|
| **Use Case Name** | Register Account |
| **ID** | UC02 |
| **Importance Level** | High |
| **Primary Actor** | Student |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Student: Needs a valid account for system access.<br>System: Must store valid user information and prevent duplicate accounts |
| **Brief Description** | A new student creates an account by providing personal details like name, student ID, email, and password. |
| **Trigger** | Student clicks "Register" on login page. |
| **Relationships** | Include: Input Validation<br>Extend: Create Account, Update Database |
| **Normal Flow** | 1. Students fill out the registration form (Full Name, Student ID, Email, Password)<br>2. On submission, the system performs server-side validation<br>3. The system hashes the password and creates a new user record<br>4. The new account is stored in the users table in the database. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 2a. Email or Student ID already exists → Display error: "An account with this email/ID already exists".<br>2b. Invalid input → Display validation error. |

Use Case 3: Register Face

*Table 3.2.1.3 Use Case Description (Register Face)*

| Field | Description |
|---|---|
| **Use Case Name** | Register Face |
| **ID** | UC03 |
| **Importance Level** | High |
| **Primary Actor** | Student |
| **Preconditions** | Students must be logged in. |
| **Postconditions** | A serialized facial embedding is stored in the database, associated with the student's user ID. |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Student: Needs a registered face to use the attendance marking feature<br>System: Must store unique, high-quality, and live-verified facial embedding. |
| **Brief Description** | The system captures the student's face via a live video stream, validates liveness, generates a composite facial embedding from multiple captures, and stores it. |
| **Trigger** | Student clicks "Register Face" from their dashboard. |
| **Relationships** | Include: Access Camera via Browser API, Detect Face, Perform Blink Detection (Liveness), Extract Facial Features, Generate Composite Embedding, Store Embedding<br>Extend: Check if Face is Already Registered |
| **Normal Flow** | 1. Student selects "Register Face."<br>2. System checks if already registered.<br>3. The browser prompts the student for camera permission. Upon granting, the system activates the camera and displays the video feed.<br>4. The system guides the student to align their face and analyses the video stream in real-time for active challenge response to confirm liveness |

| | |
|---|---|
| | 5. The final embedding is serialized and stored in the database. A success message is displayed. |
| **Subflows** | 4a. If face detection or liveness check fails intermittently, the system prompts the user to "Hold still" or "Ensure good lighting" and continues trying |
| **Alternate/Exceptional Flows** | 2a. Face already registered → Display “You have already registered your face.”<br>3a. Camera access denied by user → Display "Camera access is required for face registration."<br>3b. No face detected for a prolonged period → Display “No face detected. Please position your face in the frame.”<br>3c. No face detected → Prompt retry.<br>3d. 3c. Liveness check repeatedly fails → Display “Liveness check failed. |

Use Case 4: Scan Attendance

*Table 3.2.1.4 Use Case Description (Scan Attendance)*

| Field | Description |
|---|---|
| **Use Case Name** | Scan Attendance |
| **ID** | UC04 |
| **Importance Level** | Critical |
| **Primary Actor** | Student |
| **Preconditions** | • Student is logged in and has a registered face<br>• Students are enrolled in the target course-section<br>• The lecturer has activated the attendance session for this specific class section. |
| **Postconditions** | A new attendance record is created in the database with the student's ID, section ID, status ('Present'), timestamp |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Student: Wants attendance marked accurately and efficiently<br>Lecturer: Needs accurate, cheat-proof attendance records.<br>System: Must validate identity, enrollment, schedule, and location to ensure data integrity. |
| **Brief Description** | The student initiates attendance marking. The system first verifies their enrollment, the class schedule, and their physical proximity to the lecturer. If all checks pass, it performs facial recognition with a liveness check to mark attendance. |
| **Trigger** | Student selects a specific class section from their dashboard and clicks “Mark Attendance.” |
| **Relationships** | Include: Verify Enrollment, Verify Schedule, **Verify Proximity**, **Access Camera via Browser API**, **Perform Liveness Detection**, Recognize Face, Mark Attendance<br>Extend: Check if Face Registered |
| **Normal Flow** | 1. Students select class sections.<br>2. The system performs three sequential checks:<br>a. **Enrollment Check:** Verifies the student is enrolled in the selected section. |

| | |
|---|---|
| | b. **Schedule Check:** Confirms the current time is within the class's scheduled start and end time<br>3. **Only if all three checks pass**, does the system activate the camera for face scanning<br>4. The system performs a **liveness check via blink detection** to prevent spoofing.<br>5. Upon confirming a live face, the system extracts its embedding and **compares it against the student's stored embedding**<br>6. If the **face similarity score is above the defined threshold (e.g., >90%)**, the system creates an attendance record with the current timestamp and the student's captured GPS location<br>7. The system displays a confirmation: "Attendance Marked Successfully." |
| **Alternate/Exceptional Flows** | 2a. Enrollment check fails → Display "You are not enrolled in this section."<br>2b. Schedule check fails → Display "This class session is not active currently.<br>2c. Proximity check fails → Display "You are out of range. Please move closer to the class location."<br>3a. Student has no registered face → Redirect to the "Register Face" page.<br>4a. Liveness check fails → Display "Liveness check failed. Please try again."<br>5a. Face not recognized (low similarity) → Display "Face not recognized. Please try again."<br>5b. Student has already marked attendance for this session → Display "Attendance already marked." |

### 3.2.2 Use Case Diagram - Lecturer

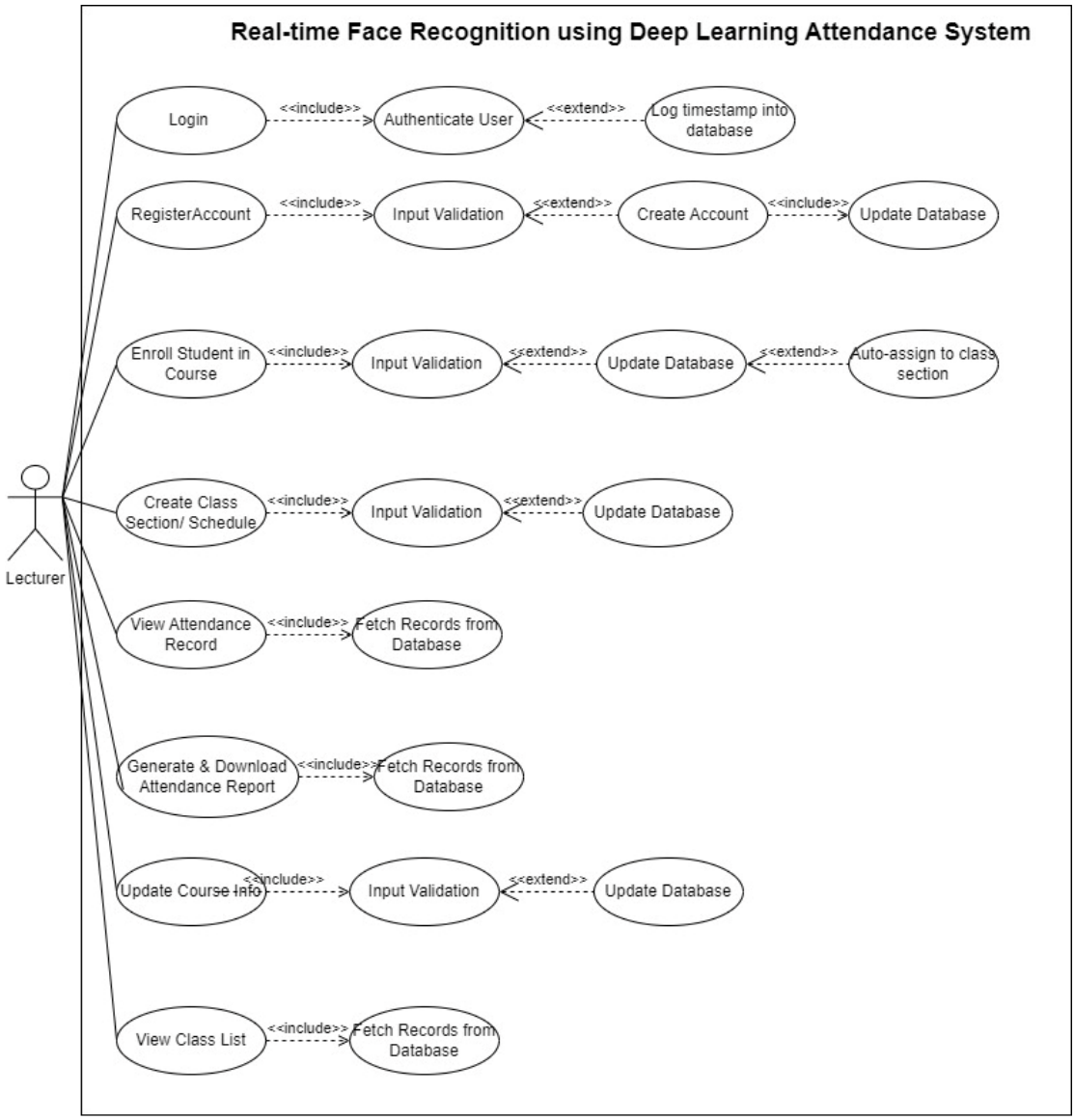



*Figure 3.2.2.1 Use Case Diagram for Lecturer*

The use case diagram for the **lecturer** in the **Real-time Face Recognition using Deep Learning Attendance System** outlines the key functionalities available to lecturers, emphasizing course and student management, as well as attendance monitoring. Lecturers can log in and register accounts, with proper authentication and validation. Once authenticated, they can create class sections or schedules, enroll students into courses, and the system will automatically assign students to appropriate class sections. Lecturers also can view class lists, update course information, monitor real-time attendance records, and download attendance reports. Each primary function is supported by system-level processes such as input validation, database updates, and data retrieval to ensure accurate, efficient operation.

**Lecturer Use Case Description:**

Use Case 1: Login

*Table 3.2.2.1 Use Case Description (Login)*

| Field | Description |
|---|---|
| **Use Case Name** | Login |
| **ID** | UC-L1 |
| **Importance Level** | High |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs secure, role-based access to management functions.<br>System: Must authenticate users against stored credentials and restrict access. |
| **Brief Description** | The lecturer logs into the system using their registered email and password to gain access to their dashboard. |
| **Trigger** | Lecturer submits the login form. |
| **Relationships** | Include: Authenticate User<br>Extend: Log access timestamp |
| **Normal Flow** | 1. The lecturer enters their email and password.<br>2. The system hashes the entered password and compares it against the stored hash for the email provided.<br>3. If credentials are valid, the system **creates a server-side session** and records a login timestamp<br>4. The lecturer is redirected to their main dashboard. |
| **Alternate/Exceptional Flows** | 1a. Invalid credentials → Display error message: "Invalid email or password."<br>1b. System database is offline → Display error: "Service currently unavailable." |

Use Case 2: Register Account

*Table 3.2.2.2 Use Case Description (Register Account)*

| Field | Description |
|---|---|
| **Use Case Name** | Register Account |
| **ID** | UC-L2 |
| **Importance Level** | Medium |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs a straightforward method to create an account.<br>System: Must securely store lecturer information and prevent duplicate accounts. |
| **Brief Description** | A new lecturer creates an account by providing personal and professional details. |
| **Trigger** | Lecturer clicks the “Register” link on the login page. |
| **Relationships** | Include: Input Validation<br>Extend: Create Account, Update Database |
| **Normal Flow** | 1. The lecturer completes the form.<br>2. On submission, the system performs **server-side validation** to check for required fields, valid email format<br>3. The system **hashes the password using a strong algorithm**<br>4. A new record is created in the users table with the role 'lecturer' and stored in the database. |
| **Alternate/Exceptional Flows** | 2a. Invalid input format → Display specific field-level validation errors (e.g., "Please enter a valid email address."). |

Use Case 3: Enroll Student in Course

*Table 3.2.2.3 Use Case Description (Enroll Student in Course)*

| Field | Description |
|---|---|
| **Use Case Name** | Enrolling Student in Course |
| **ID** | UC-L3 |
| **Importance Level** | High |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs to efficiently manage class rosters.<br>Student: Must be formally enrolled to participate in a course. |
| **Brief Description** | The lecturer enrolls one or more existing students on a specific course they manage. |
| **Trigger** | The lecturer selects a course and navigates to the "Enroll Students" function. |
| **Relationships** | Include: Input Validation<br>Extend: Update Database |
| **Normal Flow** | 1. Lecturers select a course from their course list.<br>2. Lecturer selects one or more students to enroll (e.g., from a searchable list or by entering student IDs).<br>3. The system validates the selected students and courses exist.<br>4. The system creates records on the enrollments join table, linking the student IDs with the course ID. |
| **Alternate/Exceptional Flows** | 2a. Student is already enrolled in the course → The system skips the duplicate entry and provides a notification: "[Student Name] is already enrolled." |

Use Case 4: Create Class Section

*Table 3.2.2.4 Use Case Description (Create Class Section)*

| Field | Description |
|---|---|
| **Use Case Name** | Create Class Section |
| **ID** | UC-L4 |
| **Importance Level** | High |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs to define specific class schedules for a course<br>Student: Needs to know the exact time and day for their classes. |
| **Brief Description** | The lecturer creates a specific, scheduled class section (e.g., tutorial, lab) and links it to an existing course. |
| **Trigger** | Lecturer clicks "Add Class Section" from a course management page. |
| **Relationships** | Include: Input Validation.<br>Extend: Update Database |
| **Normal Flow** | ecturer enters section details, including **Section Name/Code**, and datetime.<br>w record is created in the class_sections table, linked to urses table via a foreign key. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 2a. Invalid course selection → Error.<br>2b. Schedule conflict → Alert lecturer. |

Use Case 5: View Attendance Record

*Table 3.2.2.5 Use Case Description (View Attendance Record)*

| Field | Description |
|---|---|
| **Use Case Name** | View Attendance Record |
| **ID** | UC-L5 |
| **Importance Level** | High |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs to monitor student attendance in real-time or historically. |
| **Brief Description** | The lecturer views the detailed attendance records for a specific class section. |
| **Trigger** | The lecturer selects a course and then a specific section to view its records. |
| **Relationships** | Include: Fetch Records from DB |
| **Normal Flow** | 1. Lecturer selects a course and a class section.<br>2. The system retrieves all attendance records associated with that section.<br>3. The system displays the records in a table, showing Student Name, Student ID, Date, Check-in Timestamp, and Status |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | No attendance has been taken yet → Display message: "No attendance records found for this section." |

Use Case 6: Generate & Download Attendance Report

*Table 3.2.2.6 Use Case Description (Generate & Download Attendance Report)*

| Field | Description |
|---|---|
| **Use Case Name** | Generate & Download Report |
| **ID** | UC-L6 |
| **Importance Level** | Medium |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs an offline, portable copy of attendance records for administrative purposes. |
| **Brief Description** | The lecturer generates and downloads a formatted attendance report for a class section. |
| **Trigger** | Lecturer clicks a "Download Report" button on the attendance view page for the selected class section. |
| **Relationships** | Include: Fetch Records from DB |
| **Normal Flow** | 1. The lecturer optionally selects report criteria.<br>2. Lecturer selects a file format (**CSV, PDF, Excel**)<br>3. The system retrieves the relevant attendance data from the database.<br>4. The system generates the file in the selected format and initiates a download in the lecturer's browser. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 2a. No data matches the selected criteria → Display message: "No data available for the selected criteria."<br>2b. Data retrieval or file generation error → Display error: "Could not generate the report. Please try again." |

Use Case 7: Update Course Info

*Table 3.2.2.7 Use Case Description (Update Course Info)*

| Field | Description |
|---|---|
| **Use Case Name** | Update Course Info |
| **ID** | UC-L7 |
| **Importance Level** | Medium |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs to keep course information accurate and up to date. |
| **Brief Description** | The lecturer edits the details of an existing course. |
| **Trigger** | Lecturer selects a course and clicks an "Edit" button. |
| **Relationships** | Include: Input Validation<br>Extend: Update Database |
| **Normal Flow** | 1. Lecturer selects a course to edit.<br>2. The system displays a form **pre-populated with the current course information.**<br>3. The lecturer modifies the desired fields and submits the form.<br>4. The system validates the new information.<br>5. The corresponding record in the courses table is updated in the database. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 4a. Updated course code conflicts with an existing course → Display error: "This course code is already in use." |

Use Case 8: View Class List

*Table 3.2.2.8 Use Case Description (View Class List)*

| Field | Description |
|---|---|
| **Use Case Name** | View Class List |
| **ID** | UC-L8 |
| **Importance Level** | Medium |
| **Primary Actor** | Lecturer |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs to see a simple list of all students enrolled in a class. |
| **Brief Description** | The lecturer views a list of all students currently enrolled in a specific class section. |
| **Trigger** | Lecturer selects a course and a section. |
| **Relationships** | Include: Fetch Records from DB |
| **Normal Flow** | 1. Lecturer selects a course and a specific class section.<br>2. The system queries the enrollments table to retrieve the list of students for that section<br>3. The system displays a list containing student details like Student Name, Student ID, and Email. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 2a. No students are enrolled → Display message: "No students are currently enrolled in this section." |

Use Case 9: Auto-assign to Class Section

*Table 3.2.2.9 Use Case Description (Auto-Assign to Class Section)*

| Field | Description |
|---|---|
| **Use Case Name** | View Class List |
| **ID** | UC-L9 |
| **Importance Level** | Medium |
| **Primary Actor** | System |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | System: Ensures efficient scheduling of students into available sections.<br>Students**:** Are automatically placed into class sections upon course enrollment. |
| **Brief Description** | The system automatically assigns newly enrolled students to an available class section for that course. |
| **Trigger** | A student is successfully enrolled in a course (triggered by UC-L3: Enroll Student in Course). |
| **Relationships** | Extend: Enroll Student in Course |
| **Normal Flow** | 1. Upon successful student enrollment in a course, the system checks for available class sections for that course.<br>2. The system queries the enrollments table to retrieve the list of students for that section<br>3. The system displays a list containing student details like Student Name, Student ID, and Email. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 2a. No class sections exist for the course → The student is enrolled in the course but remains unassigned to a section. A notification may be logged for the lecturer. |

Use Case 10: Activate/Deactivate Class Session

*Table 3.2.2.10 Use Case Description (Activate/Deactivate Class Session)*

| Field | Description |
|---|---|
| **Use Case Name** | Activate/Deactivate Class Session |
| **ID** | UC-L10 |
| **Importance Level** | Critical |
| **Primary Actor** | Lecturer |
| **Preconditions** | • The lecturer is logged in.<br>• The class section is scheduled for the current day. |
| **Postconditions** | • The class section is marked as "active" or "inactive" in the database, controlling the window for student check-ins. |
| **Use Case Type** | Basic, Essential |
| **Stakeholders and Interests** | Lecturer: Needs a simple control to start and stop the attendance-taking window<br>System: Requires an "active" flag to validate student attendance attempts. |
| **Brief Description** | The lecturer starts an attendance session, creating a time-bound window during which students can mark their attendance. They can later end the session. |
| **Trigger** | Lecturer clicks a “Start Session” or “End Session” button on their dashboard for a specific class. |
| **Relationships** | Include: Update Database |
| **Normal Flow** | 1. Lecturer selects the current class section from their dashboard<br>2. Lecturer clicks the “Start Attendance Session” button.<br>3. The system updates the session's status to **"active"** in the database and records the start time.<br>4. The UI updates to show the session is "In Progress" and presents an "End Session" button.<br>5. At the end of the class, the lecturer clicks the “End Session” button. |

| | |
|---|---|
| | 6. 6. The system updates the session's status to **"inactive,"** preventing any further attendance marking. |
| **Subflows** | None |
| **Alternate/Exceptional Flows** | 2a. Session is already active → The "Start Session" button is disabled or hidden, showing only the "End Session" option.<br>5a. Attempting to end a session that is not active → The "End Session" button is disabled or hidden. |

# Chapter 4
# System Design

The development of the core face recognition model followed a structured and systematic pipeline, encompassing data acquisition, preprocessing, model architecture design, training, and rigorous evaluation. This pipeline ensures the final model is both accurate and robust. The entire process is visually summarized in the block diagram below.

## 4.1 Model Training Pipeline

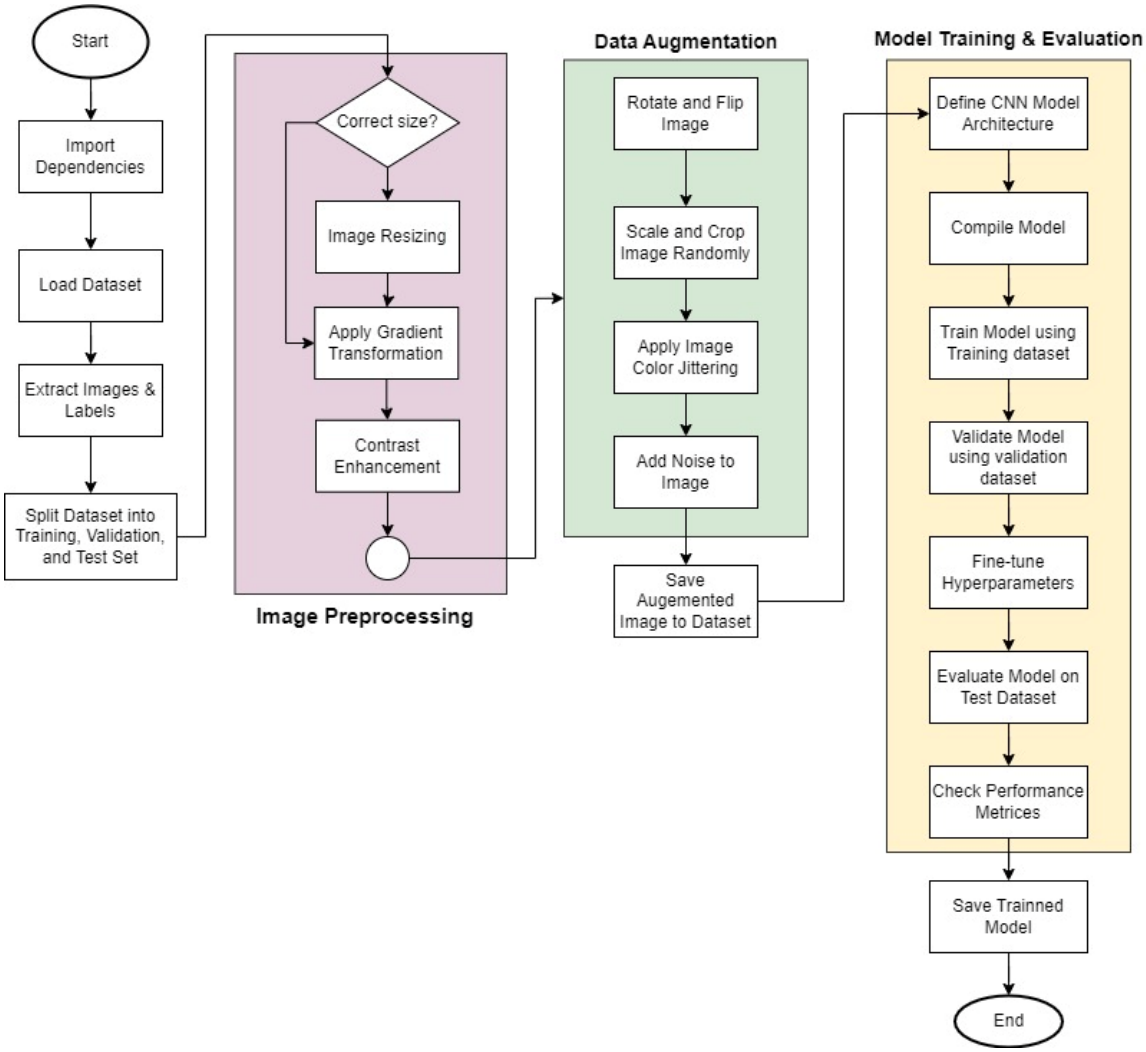



*Figure 4.1.1 Model Training Block Diagram*

## 1. Import Dependencies

The construction and training of the face recognition model were facilitated by a curated ecosystem of powerful Python libraries. The selection of this particular stack was deliberate, leveraging the strengths of each component to create an efficient and effective development workflow:

- **TensorFlow and Keras**: As the foundational deep learning framework, this combination provides both high-level abstraction through the Keras API for rapid prototyping and low-level flexibility through TensorFlow for custom operations. It was used to meticulously construct the Convolutional Neural Network (CNN) architecture layer-by-layer (e.g., Conv2D, MaxPooling2D, Dense), manage the entire training loop via model.fit, and conduct the final performance assessment using model.evaluate.
- **OpenCV**: This library is the de facto standard for computer vision tasks. In this project, it was employed for essential image preprocessing tasks, such as standardizing image dimensions with cv2.resize, which is a critical step to ensure all inputs to the neural network are of a uniform size.
- **NumPy**: The bedrock of scientific computing in Python, NumPy was indispensable for all numerical operations. Its highly optimized N-dimensional array objects provided data structures for efficiently manipulating image pixel data throughout the entire pipeline.
- **Matplotlib**: A model's internal learning process can often be a "black box." Matplotlib was crucial for peering inside this box through data and model visualization. It was used to display sample images (plt.imshow) for sanity checks and, more importantly, to plot the model's learning curves (accuracy and loss over epochs), providing vital insights into the training dynamics and helping diagnose issues like overfitting.
- 
- **scikit-learn:** This versatile machine learning library streamlined several key, non-deep-learning processes. Its modules were used for fetching and managing the LFW dataset (fetch_lfw_people), reliably partitioning the data into statistically sound training, validation, and test sets (train_test_split), and generating detailed classification reports with metrics beyond simple accuracy.

**2. Load Dataset**

The model was trained, validated, and tested using the well-established **Labeled Faces in the Wild (LFW)** dataset, conveniently accessed through the scikit-learn library. The LFW dataset was chosen because it represents a "real-world" challenge: it contains images of individuals with variations in lighting, pose, expression, and background, which is essential for training a model that can perform well outside of a controlled lab environment.

**3. Extract Images & Labels**

Upon loading the dataset, the raw image data (pixel arrays) and their corresponding identity labels were extracted and segregated into separate variables. This fundamental step transforms the dataset into the standard (X, y) format expected by most machine learning frameworks, where X represents the input features (the images) and y represents the target labels (the identities).

**4. Split Dataset**

To ensure an unbiased evaluation and to build a model that generalizes well to new faces, the dataset was strategically partitioned into three distinct subsets. As emphasized in established machine learning literature [17], this division is a cornerstone of robust model development:

- **Training Set (60%):** The largest portion of the data, used exclusively for the model to learn from. During this phase, the model is exposed to this data and iteratively adjusts its internal weights and biases through backpropagation to minimize a loss function, thereby learning the discriminative features of each identity.
- **Validation Set (20%):** This separate subset acts as a proxy for unseen data *during* the training process. After each epoch, the model's performance is evaluated on this set. This feedback loop is essential for two reasons: tuning hyperparameters (like the learning rate) and triggering mechanisms like early stopping to prevent the model from simply memorizing the training data (overfitting).
- **Test Set (20%):** This final, completely untouched subset is held in reserve until all training and tuning are complete. Its performance on this set provides the most honest and reliable estimate of how the model will perform on new, real-world data, as the model has never been exposed to it in any capacity.

## 5. Image Preprocessing

The LFW dataset, as provided by scikit-learn's helper function, has already undergone significant preliminary processing, including face detection, alignment (ensuring features like eyes are in consistent locations), and cropping. This pre-processing is a major advantage, as it ensures that the faces are centered and consistently oriented, allowing CNN to focus on learning identity features rather than spatial variations. Consequently, any additional aggressive preprocessing was deemed unnecessary and potentially detrimental. Over-processing could disrupt this careful alignment or introduce artifacts, so the focus was on preserving the high quality of the provided images.

## 6. Data Augmentation

While the LFW dataset is diverse, the number of images per person can be limited. To enhance the model's ability to generalize to real-world variations and to combat overfitting, data augmentation was applied exclusively to the training set. Using Keras' ImageDataGenerator, the training dataset was artificially expanded on-the-fly by creating modified versions of the original images. This process simulates variations a face might exhibit in a real-world scenario while retaining the correct identity label. The augmentation pipeline included:

- **Geometric Transformations:**
  Random rotations, horizontal flips, and slight width/height shifts to make the model robust to small changes in pose and camera angle.
- **Photometric Adjustments:**
  Random variations in brightness and contrast to simulate different lighting conditions. This technique effectively exposes the model to a much wider and more varied range of data than was originally available, fostering the development of a more robust and invariant feature representation without the need to collect thousands of additional images.

## 7. Model Training & Evaluation

This phase represents the core intellectual contribution of the project, detailing the architectural design of the Convolutional Neural Network (CNN), the sophisticated methodology employed for its training, and the comprehensive evaluation process. The architecture was deliberately designed to be sufficiently deep to learn complex, discriminative features while incorporating modern deep learning techniques to ensure a stable, efficient, and effective training process.

### a. Define CNN Model Architecture

The CNN architecture was meticulously designed as a hierarchical feature extractor, where each successive layer learns progressively more complex and abstract representations of the input facial images. This hierarchical approach mimics the human visual cortex, starting with simple features and building up to holistic representations.

**i. Input Layers**

The model's entry point is a precisely defined Input layer, configured to accept RGB images of shape (64, 64, 3). This fixed input tensor size is a critical prerequisite for batch processing on a GPU, as it allows for highly parallelized matrix computations, drastically accelerating the training process.

*inputs = Input(shape=input_shape)*

**ii. Convolutional Blocks**

These are the fundamental building blocks responsible for learning spatial hierarchies of features. The model leverages two types of blocks, each containing a suite of layers designed to work in concert:

- **Conv2D Layer**: This layer is the cornerstone of the feature extraction process. It applies to a set of learnable filters (kernels) across the input volume. In this architecture, a 3x3 kernel size was chosen as it is the smallest size that can capture notions of corners, edges, and textures while maintaining a low parameter count. The use of "same" padding ensures that the spatial dimensions of the output feature maps match the input, preventing the rapid loss of spatial information at the borders of the image.
- **L2 regularization (l2(l2_reg):** Applied directly to the kernel weights, adding a penalty term to the loss function that is proportional to the square of the weight values. This discourages the model from learning overly complex or large weights, a key strategy in mitigating overfitting.
- **ReLU Activation**: Following each convolution, the Rectified Linear Unit (Activation('relu')) is applied. This non-linear activation function is computationally efficient and helps to alleviate the vanishing gradient problem. By setting all negative values to zero, it introduces non-linearity into the network, which is crucial for enabling the CNN to learn intricate and highly non-linear patterns present in complex data like faces.

- **Batch Normalization**: This layer is applied after the convolution and before the activation. It normalizes the activations of the previous layer by re-centering and re-scaling them to have a mean of zero and a standard deviation of one for each mini batch. This technique is critical for stabilizing the training of deep networks, as it ensures that the distribution of inputs to subsequent layers remains consistent, allowing for the use of higher learning rates and significantly accelerating model convergence.
- **Dropout**: To further combat overfitting, a Dropout layer is employed. During training, this layer randomly sets a fraction of input units (neurons) to zero at each update step. This prevents neurons from co-adapting too much and forces the network to learn more robust and redundant features, making it less sensitive to the specific weights of any single neuron and thus improving its ability to generalize to unseen data.

iii. **Residual Blocks**:

To build a network capable of learning truly discriminative features, depth is essential. However, naively stacking layers in very deep networks can lead to the infamous **vanishing gradient problem** [19], where the gradient signal diminishes exponentially as it propagates back through the network, causing the early layers to learn extremely slowly or not at all. **Residual blocks (ResNets)** are the key architectural innovation to solve this. Each block contains two convolutional layers and a "shortcut connection" that performs an element-wise addition of the block's input to its output. This creates an unimpeded "identity pathway" for the gradient to flow directly through the network during backpropagation, enabling the stable training of much deeper architectures. The components include:

- **Two Convolutional Blocks:** Perform the primary feature extraction.
- **Shortcut Connection:** Adds the original input of the block to the output of the convolutional layers (Add()([x, shortcut])), facilitating gradient flow.
- **1x1 Convolution (for dimension matching):** If the number of filters changes or a stride is used for down-sampling, the dimensions of the input and output will not match the addition operation. In this case, a 1x1 convolution is applied to the shortcut connection to linearly project it into a new space with matching dimensions, ensuring the element-wise addition is possible.

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & otherwise \end{cases}$$

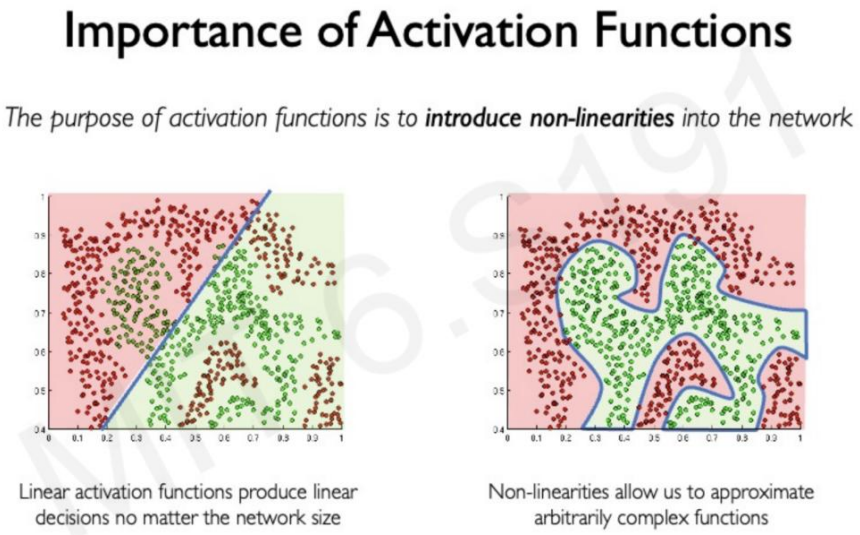



*Figure 4.1.2 Effect on Implementation of Activation Functions*

**iv. Model Layers (Specific Arrangement)**

The network's architecture is a sequential arrangement of these blocks, carefully designed for a progressive feature extraction pipeline that moves from low-level to high-level representations:

1. **Initial Feature Extraction:**

   The first two residual blocks (with 32 and 64 filters) are designed to capture low-level features like edges, corners, and basic textures from the raw pixel data. These are followed by MaxPooling2D for spatial down-sampling (which reduces the computational complexity and creates a degree of translational invariance) and Dropout (0.3) for initial regularization.

   MaxPooling2D(pool_size=(2, 2))

   Dropout(0.3)

2. **Intermediate Feature Refinement**:

   The next two residual blocks (with 128 filters each) have a larger receptive field and are capable of combining the low-level features into more complex patterns and object parts, such as components of eyes, noses, or mouths. The Dropout rate is increased to 0.4 as the network becomes deeper and more susceptible to overfitting.

*MaxPooling2D(pool_size=(2, 2))*

*Dropout(0.4)*

3. **Deep Feature Abstraction**:

   The final two residual blocks (with 256 filters each) operate on highly abstract representations and are responsible for learning the high-level, discriminative features that uniquely define a person's facial structure. This stage is regularized with the highest Dropout rate of 0.5 to aggressively combat overfitting in these deep, high-capacity layers.

   *MaxPooling2D(pool_size=(2, 2))*

   *Dropout(0.5)*

4. **Spatial Information Aggregation:**

   A GlobalAveragePooling2D layer is used to condense each of the 256 feature maps into a single scalar value by taking the average. This is a powerful technique that drastically reduces the number of parameters compared to a traditional Flatten layer, making the network less prone to overfitting and more robust to spatial translations of features in the input image

5. **Feature Embedding**

   A fully connected Dense layer with 1024 units acts as the final embedding layer. Its purpose is to project the abstract features learned by the convolutional backbone into a high-dimensional vector space. The goal of the training process is to organize this space such that embeddings of faces from the same person are clustered closely together, while embeddings from different people are pushed far apart. This 1024-dimensional vector serves as the final, quantitative "facial signature."

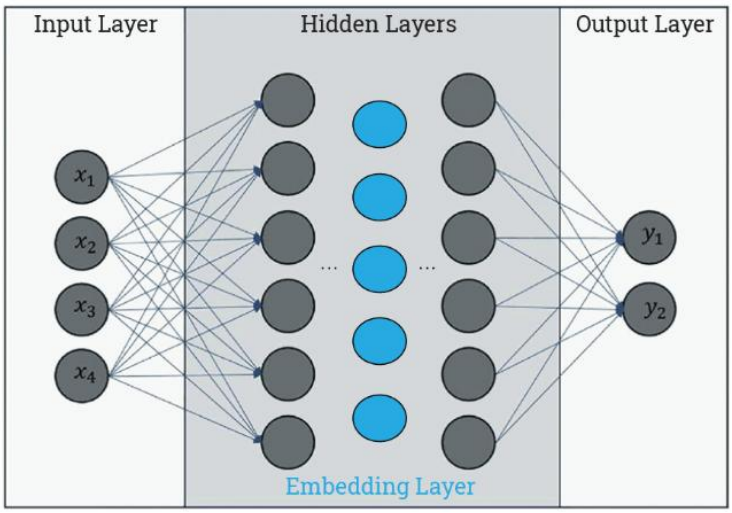



*Figure 4.1.3 Embedding Layer in Neural Network*

6. **Classification:**

   The final Dense output layer uses a softmax activation function. This function takes the raw outputs (logits) from the previous layer and transforms them into a probability distribution across all known identities, with the sum of all probabilities equaling 1. The identity corresponding to the neuron with the highest probability is the model's final prediction.

**b. Model Compilation**

Before training, the model must be compiled, a process that configures the learning algorithm by defining the loss function, optimizer, and evaluation metrics:

- **Loss Function:** *categorical_crossentropy* was chosen as the standard, mathematically appropriate loss function for multi-class classification problems where each input belongs to exactly one class. It quantifies the dissimilarity between the model's predicted probability distribution and the true, one-hot encoded label distribution. The entire goal of the training process is to adjust the model's weights to minimize this value.
- **Optimizer:** The **Adam** optimizer, with an initial learning rate of 0.0005, was selected. Adam is an adaptive learning rate optimization algorithm that is highly effective in practice. It computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It combines the advantages of other optimizers like RMSprop and AdaGrad, making it

computationally efficient, requiring little memory, and generally robust to the choice of hyperparameters.

- **Metrics:** *accuracy* was monitored during training to provide a straightforward, interpretable measure of the model's performance on the validation set after each epoch. This metric is essential for the callbacks that depend on performance monitoring.

**c. Train the Model**

The model was trained using the model.fit() method, which orchestrates the iterative process of feeding batches of data to the model and updating its weights. Several best practices were incorporated to ensure robust and efficient learning:

- **Class Weights**: The LFW dataset is inherently imbalanced, with some individuals having significantly more photos than others. To prevent the model from becoming biased towards these majority classes, class weights were calculated and applied during training. This gives a higher weight in the loss function to samples from minority classes, effectively forcing the model to pay more attention to them and learn their features just as well.

- **Callbacks**: These are utilities that can be applied at various stages of the training process:

  - **Early Stopping:** This callback is a crucial form of regularization that prevents overfitting by stopping the training process at the optimal time. It monitors the validation loss and, if the loss does not improve for a "patience" of 10 consecutive epochs, it automatically halts the training. The restore_best_weights=True argument is critical, as it ensures that the final model weights are reverted to those from the epoch with the lowest validation loss, rather than the potentially overfitted weights from the final training step.

    *(EarlyStopping(patience=10, restore_best_weights=True)*

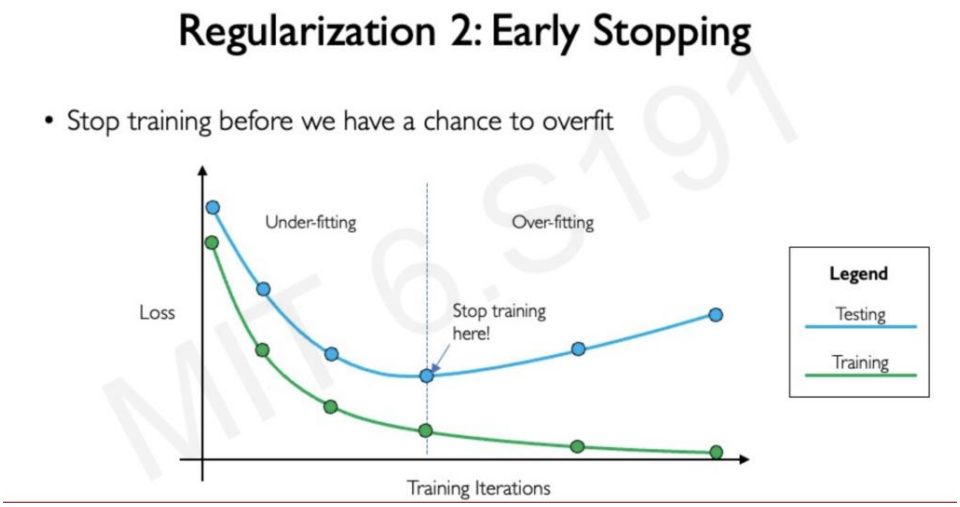



*Figure 4.1.4 Early Stopping to minimize overfitting*

- **Learning Rate Scheduler:** The ReduceLROnPlateau callback implements a dynamic learning rate schedule. If the validation loss stagnates for 5 epochs, the learning rate is automatically reduced by a factor of 0.5. This allows the model to take large, confident steps in the beginning of training when it is far from a minimum, and smaller, more precise steps as it gets closer, often leading to better convergence and a lower final loss value.

  *(ReduceLROnPlateau(factor=0.5, patience=5)*

- **Training Configuration:**

  The model was set to train for a maximum of 300 epochs with a batch size of 32. A batch size of 32 is a common choice that provides a good balance between computational efficiency (larger batches are faster to process on a GPU) and stable gradient estimation (smaller batches introduce more noise, which can sometimes help escape local minimum).

**d. Evaluate the Model**

After training, the final, unbiased performance of the model was assessed using the unseen test set. The evaluation was based on a suite of standard classification metrics that provide a more nuanced and complete picture of performance than accuracy alone:

- **Learning Curve**

  A plot of the training and validation accuracy/loss over epochs. This is the primary diagnostic tool for understanding the training process. A large and growing gap

between the training and validation curves is a clear sign of overfitting, while curves that flat line at a low accuracy indicate underfitting.

- **Accuracy**

  The overall percentage of correct predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  While intuitive, it can be a misleading metric on imbalanced datasets.

- **Precision**

  Of all the times the model predicted a certain person, what percentage of those predictions were correct?

$$Precision = \frac{TP}{TP + FP}$$

  High precision indicates a low false positive rate, meaning the model is reliable when it makes a positive identification.

- **Recall (Sensitivity)**

  Of all the actual images of a certain person, what percentage did the model correctly identify?

$$Recall = \frac{TP}{TP + FN}$$

  High recall indicates a low false negative rate, meaning the model is good at finding all instances of a person

- **F1 score**

  The harmonic means of Precision and Recall. It provides a single, balanced metric that is particularly useful when there is an uneven class distribution or when there is an asymmetric cost associated with false positives and false negatives.

$$F1\ score = \frac{2\ \times Precision\ \times Recall}{Precision + Recall}$$

## 8. Save Trained Model

Upon completion of the entire training and evaluation pipeline, the final model, including its architecture, learned weights, and optimizer state, was serialized and saved to a single HDF5 file (face_recognition_model.h5). This encapsulates the entire trained model into a portable artifact. This allows the model to be easily loaded into the main Flask application for inference without needing to be recompiled or retrained, cleanly separating the intensive, offline training process from the lightweight, real-time deployment environment.

## 4.2 System Flowchart

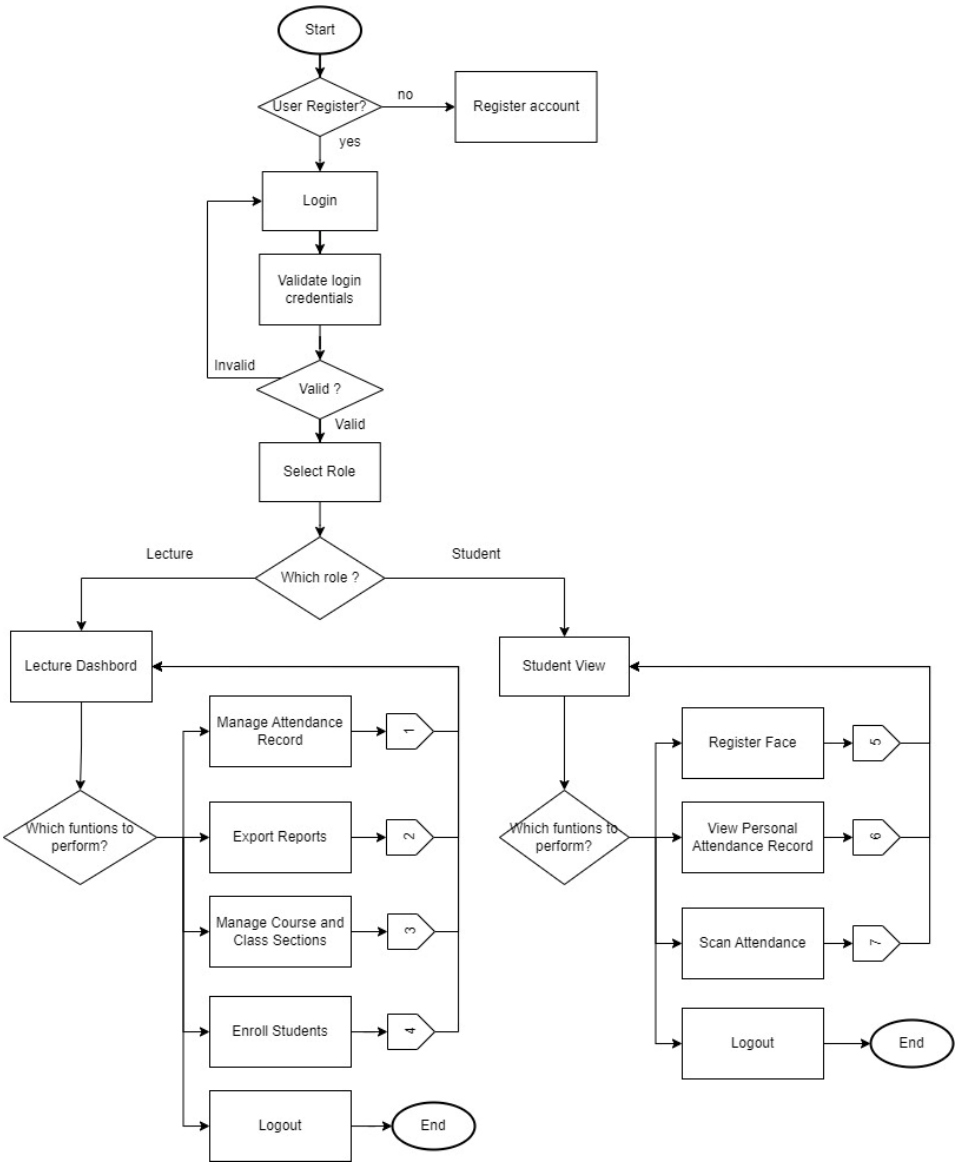



*Figure 4.2.1 System Flowchart (1/7) – Main System*

Flowchart in Figure 4.2.1 outlines a user-based attendance system with distinct roles for lecturers and students. To ensure secure access and proper functionality within the proposed system, users are required to login before accessed to system's features. Upon successful login, they select their role as either a lecturer or a student. Lectures access a dashboard where they can manage attendance records, export overall attendance records, and logout. Students, on the other hand, have a view where they can view their personal attendance records, scan their attendance, and log out. This flow chart provides a high-level overview of the system's workflow, highlighting the different functionalities available to each role.

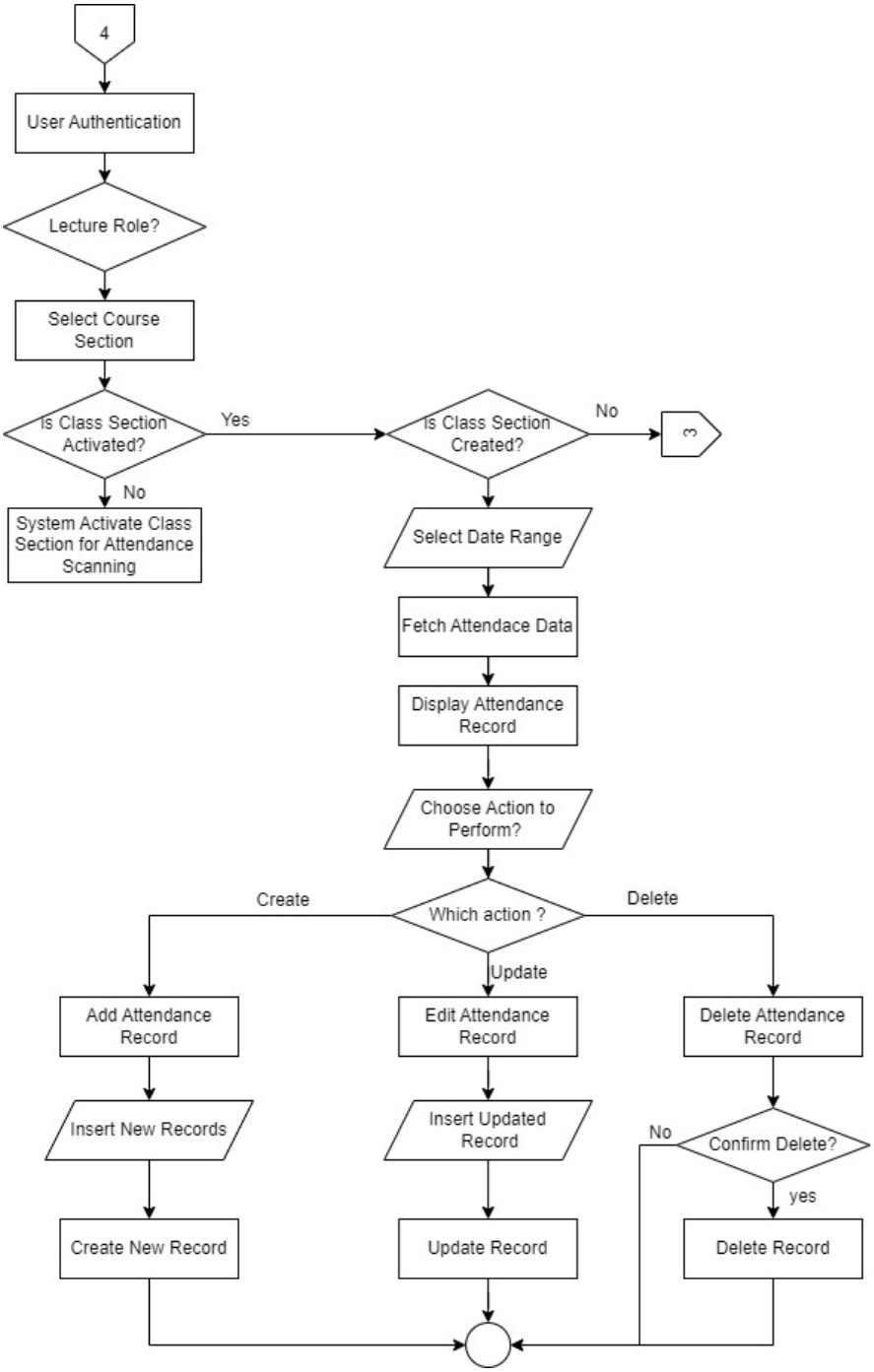



*Figure 4.2.2 System Flowchart (2/7) – Manage Attendance Record*

The manage attendance record flow begins with **user authentication** and verifying the lecturer role. The lecturer selects a course section, and the system checks if it is already activated for attendance scanning. If not, the section is activated. The lecturer can then select a date range to **fetch and display attendance data**. From here, there are three possible actions: **create**, **update**, or **delete** records. New records can be added, existing records edited and updated, or records deleted after confirmation, ensuring flexible attendance management.

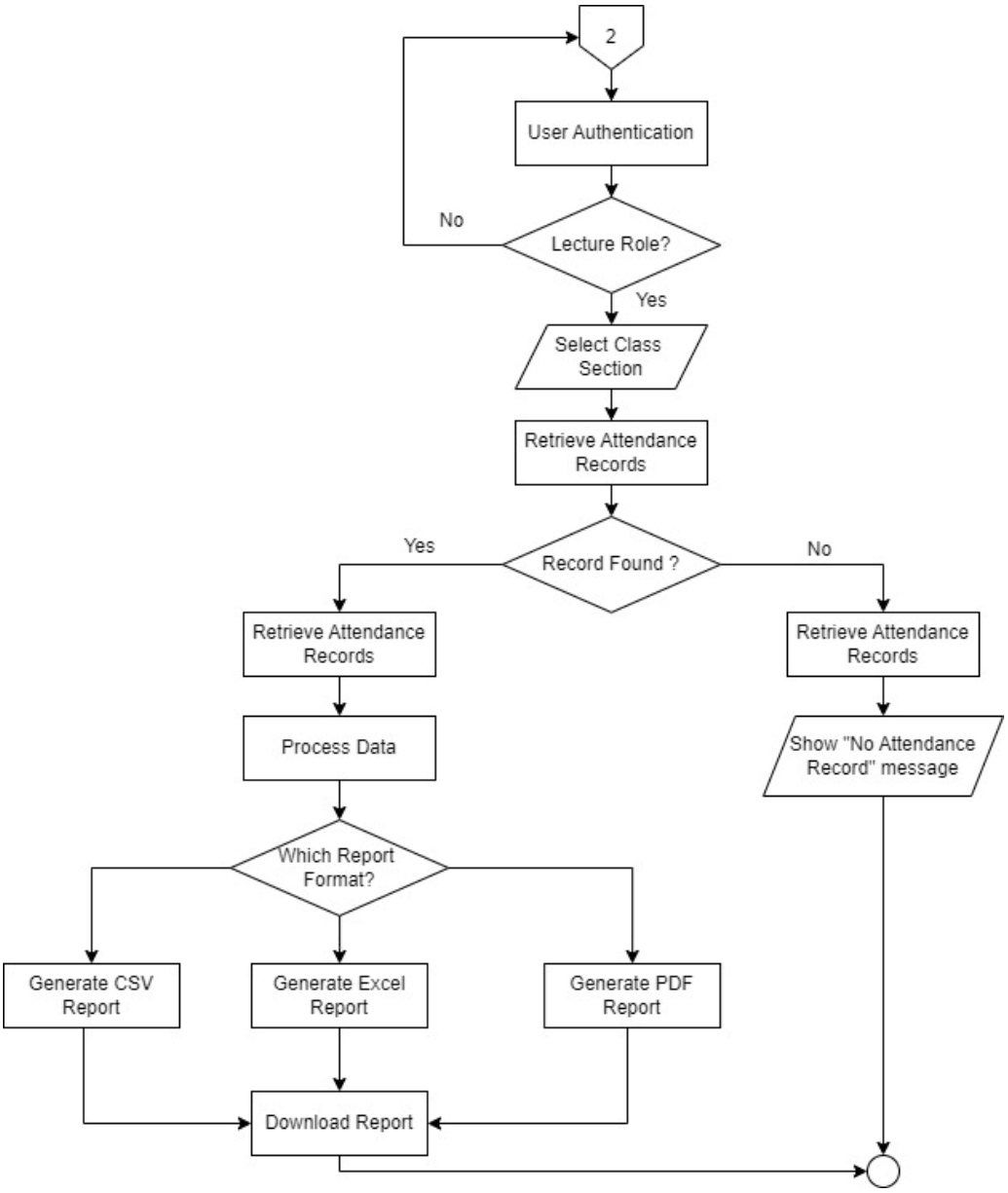



*Figure 4.2.3 System Flowchart (3/7) – Export Reports*

The export reports flow begins with lecturer authentication, ensuring only authorized users can access the feature. The system retrieves all courses and sections managed by the lecturer, allowing them to select the desired class section and export format. A validation step confirms ownership of the section before proceeding. The system then fetches attendance data from enrolled students, formats timestamps, and organizes records into a summary table. Finally, the lecturer can export the report in CSV, Excel, or PDF format, enabling easy archiving, analysis, and sharing of attendance records.

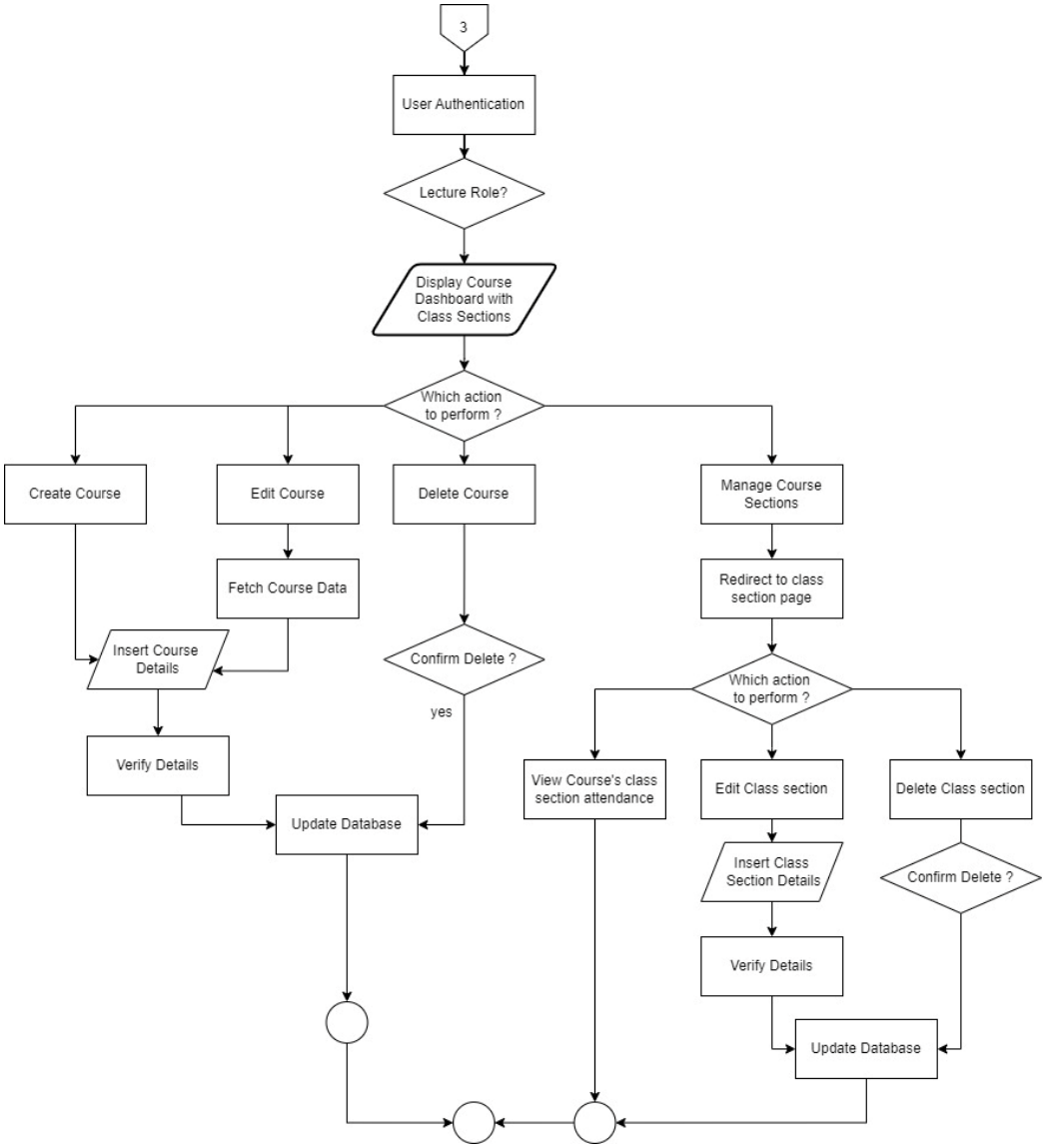



*Figure 4.2.4 System Flowchart (4/7) – Manage Course and Class Sections*

Flowchart in Figure 4.2.4 outlines a lecture-role-based course and class sections management. An authenticated lecturer gained entry to the course dashboard that displays all class sections for each course created. This system offers course-level operations: course creation, modification, and deletion. While, for course' class management system, lecturer can view real-time attendance data, add/ delete and edit existing sections while maintaining attendance records, or remove sections after confirmation.

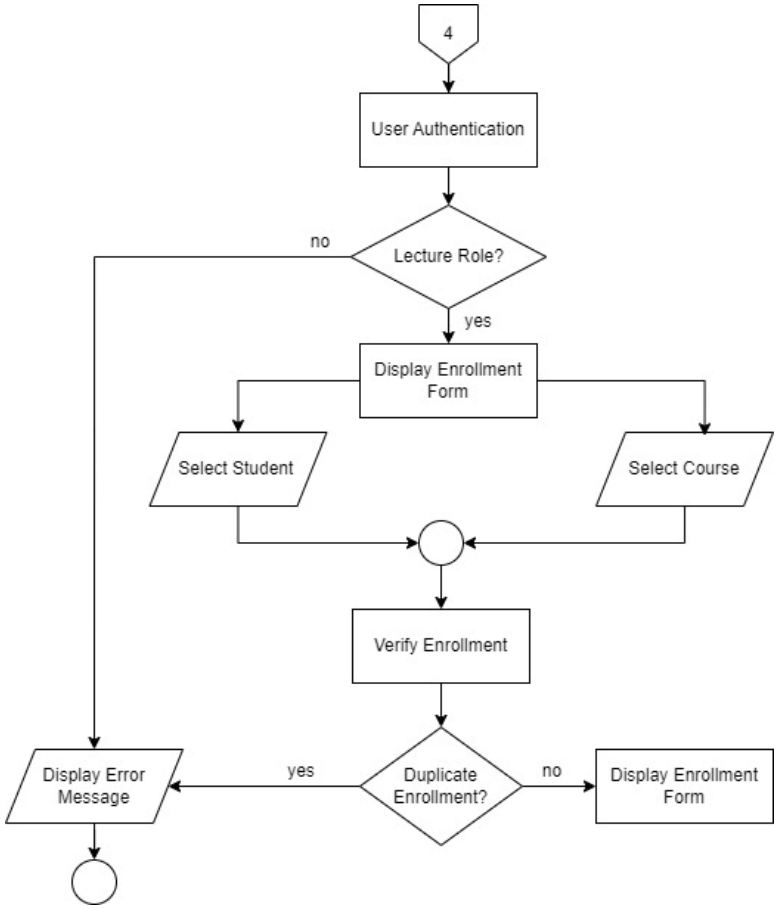



*Figure 4.2.5 System Flowchart (5/7) – Student Enrollment in Course*

The student enrollment subsystem begins by authenticating the user and verifying lecturer privileges. Upon authorization, it displays an interactive form with student/course dropdowns populated from the database. When processing enrollments, the system performs multi-stage validation: confirming student status, course availability, and checking for duplicate enrollments. Detected duplicates trigger specific warnings while preserving form data. Successful enrollments execute atomic database transactions, simultaneously updating enrollment. Following enrollment, the interface refreshes with success notifications and resets for additional entries. The subsystem automatically generates face recognition enrollment tickets for new students, ensuring synchronization with the attendance module. Throughout the process, performance optimizations like paginated data loading and transaction rollback capabilities ensure reliability, while comprehensive audit trails support compliance requirements. The streamlined workflow combines robust validation with user-friendly feedback mechanisms for efficient course management.

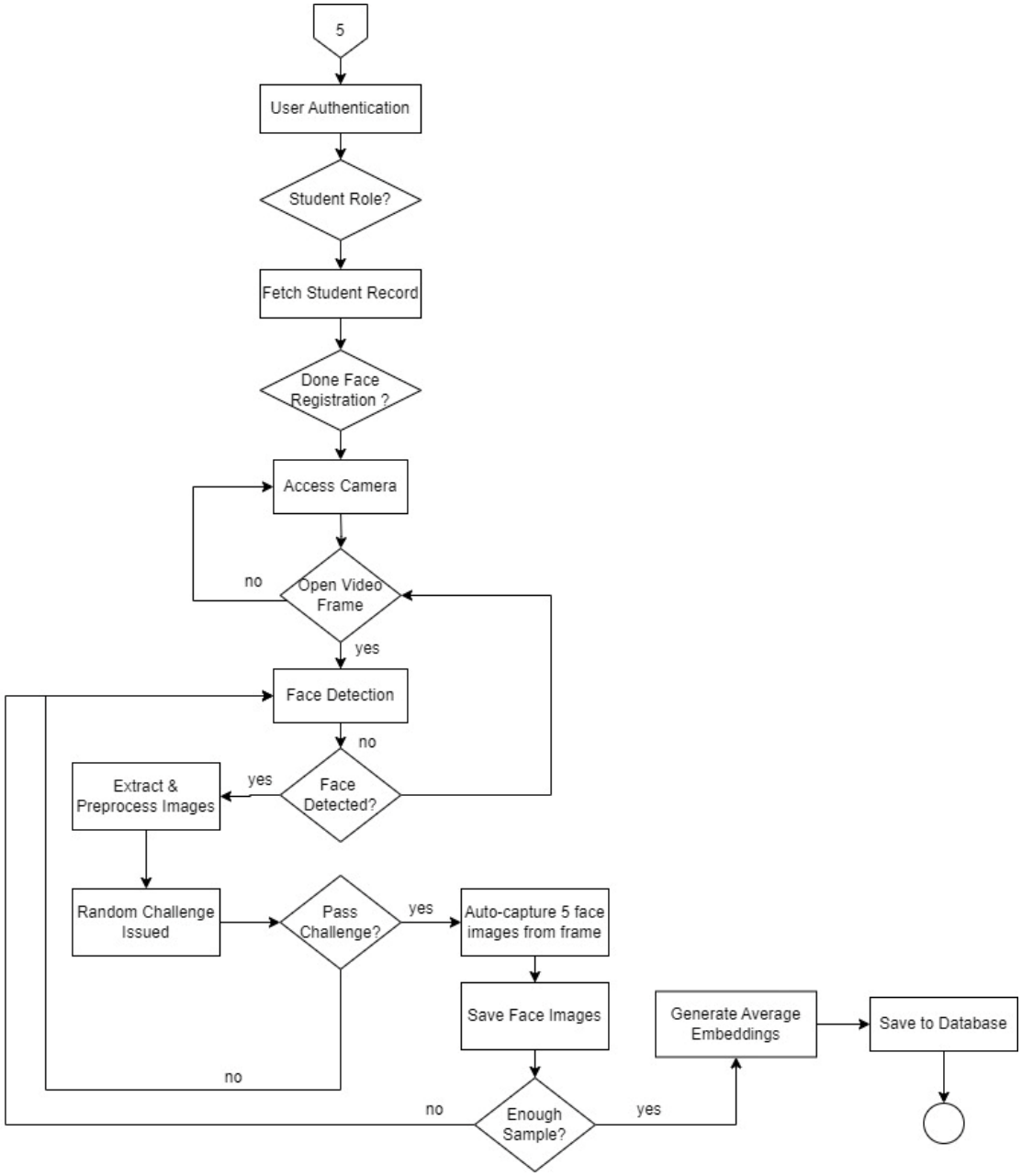



*Figure 4.2.6 System Flowchart (6/7) – Face Registration*

The face registration flow starts after authenticating a student user. The system first ensures liveness by prompting the student to perform random gestures such as blinking, smiling, or turning their head, verified through EAR, MAR, and head pose metrics. Once liveness is confirmed, the student provides live images, which are validated for size, clarity, and confidence using MTCNN. The detected face is cropped, preprocessed, and stored, with at least five diverse samples captured. Upon completion, embeddings are extracted through the CNN, averaged to form a robust identity vector, and securely stored in the database to finalize registration.

**Image Processing Techniques (Real-Time):**

Image preprocessing is a critical step in preparing input images for a Convolutional Neural Network (CNN). Unlike the curated datasets used during model training, facial images cropped from a live video feed exhibiting significant variations in scale, lighting, and noise. As noted by [20], a robust preprocessing pipeline is essential to normalize these variations, ensuring that the images are in a consistent format and that key features are enhanced for optimal model performance [25].

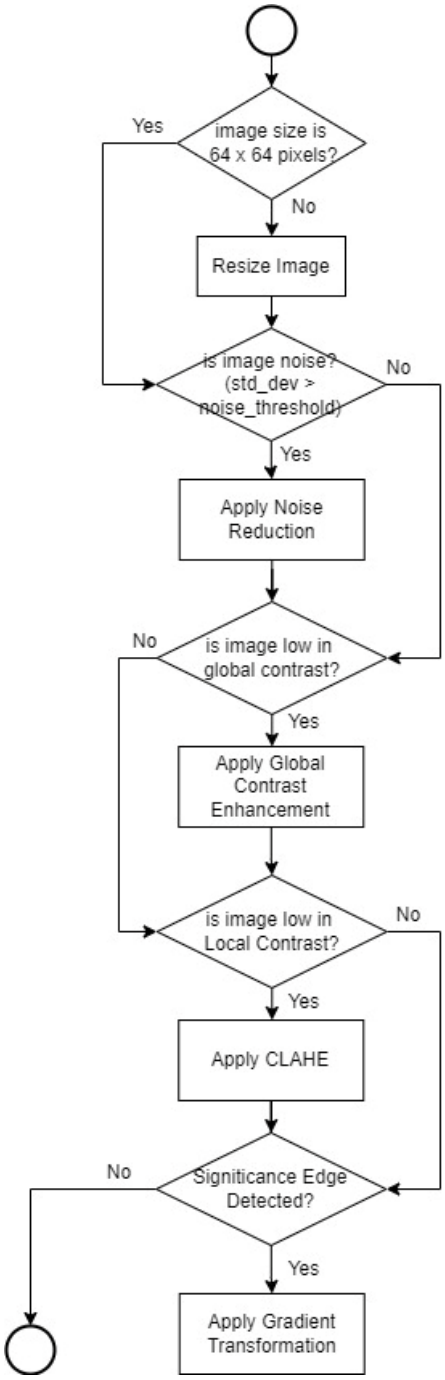



*Figure 4.2.7 Real-time Image Processing Pipeline Flowchart*

### 1. Image Resizing

The first and most fundamental step is to resize every incoming facial image to a fixed dimension of **64x64 pixels**. This standardization is a mandatory prerequisite for the CNN model, which is designed with a fixed-size input layer. By ensuring all images are of a uniform size, the model can apply its learned filters consistently, making the feature extraction process reliable and efficient.

### 2. Noise Reduction

Live video feeds, especially from webcams operating in non-ideal lighting, are often contaminated with high-frequency noise (e.g., sensor noise, grain). This noise can be detrimental to the CNN's performance, as the model might mistakenly interpret these random artifacts as meaningful features. To mitigate this, a **Gaussian Blur** is conditionally applied. The pipeline first assesses the noise level of the image by calculating the standard deviation of its pixel intensities. If this value exceeds a predefined noise_threshold, a Gaussian filter is applied, a standard technique for noise suppression in digital image processing [26].

*cv2.GaussianBlur(image, (5, 5), 0)*

This filter convolves the image with a 5x5 Gaussian kernel, effectively smoothing the image by averaging pixel values with their neighbors. This process reduces noise while preserving significant edges, ensuring the subsequent contrast enhancement steps do not amplify unwanted artifacts.

### 3. Global Contrast Enhancement (Histogram Equalization)

Poor or uneven lighting is one of the most common challenges in real-world face recognition. To address this, the pipeline first checks if the image suffers from low global contrast. This is determined using the skimage.exposure.is_low_contrast() function, which measures the dynamic range of the image's pixel intensities. If the image is identified as having low contrast, **Histogram Equalization** is applied. This technique redistributes the pixel intensity values to stretch across the entire possible range (0-255), effectively increasing the overall global contrast.

The *skimage.exposure.is_low_contrast()* is used to detect low contrast image using the following expression:

$$\frac{P_{99}(f(x,y)) - P_1(f(x,y))}{\max(f(x,y)) - \min(f(x,y))} \begin{cases} < 0.05, & then\ low\ contrast \\ \geq 0.05, & otherwise \end{cases}$$

While this is a powerful method for brightening poorly lit images, it can sometimes lead to an unnatural appearance or over-amplification of noise, which is why it is applied conditionally and followed by local contrast enhancement [27].

**4. Local Contrast Enhancement (CLAHE)**

While global histogram equalization improves overall contrast, it can wash out details in regions that are already well-lit or very dark. To address this, **Contrast Limited Adaptive Histogram Equalization (CLAHE)** is applied to enhance local contrast. As introduced by Pizer et al., CLAHE works by dividing the image into small, non-overlapping contextual regions (tiles) and applying histogram equalization to each tile independently [28]. A key feature is the "clip limit," which restricts the amplification of contrast in each tile, thereby preventing the over-amplification of noise. The resulting tiles are then stitched back together using bilinear interpolation to eliminate boundary artifacts. This method is particularly effective at revealing fine-grained facial features in areas of shadow or bright light, which might be lost with global methods alone.

**5. Apply Gradient Transformation**

As a final enhancement step, the pipeline can apply a **Sobel gradient transformation**. This operator acts as an edge detector by computing the gradient of the image intensity at each point, highlighting the contours and high-frequency details of the face, such as the jawline, eyes, and nose [29]. By emphasizing these structural features, the transformation can provide the initial layers of the CNN with a more distinct and feature-rich input, potentially aiding in the learningof more discriminative features and improving overall recognition accuracy.

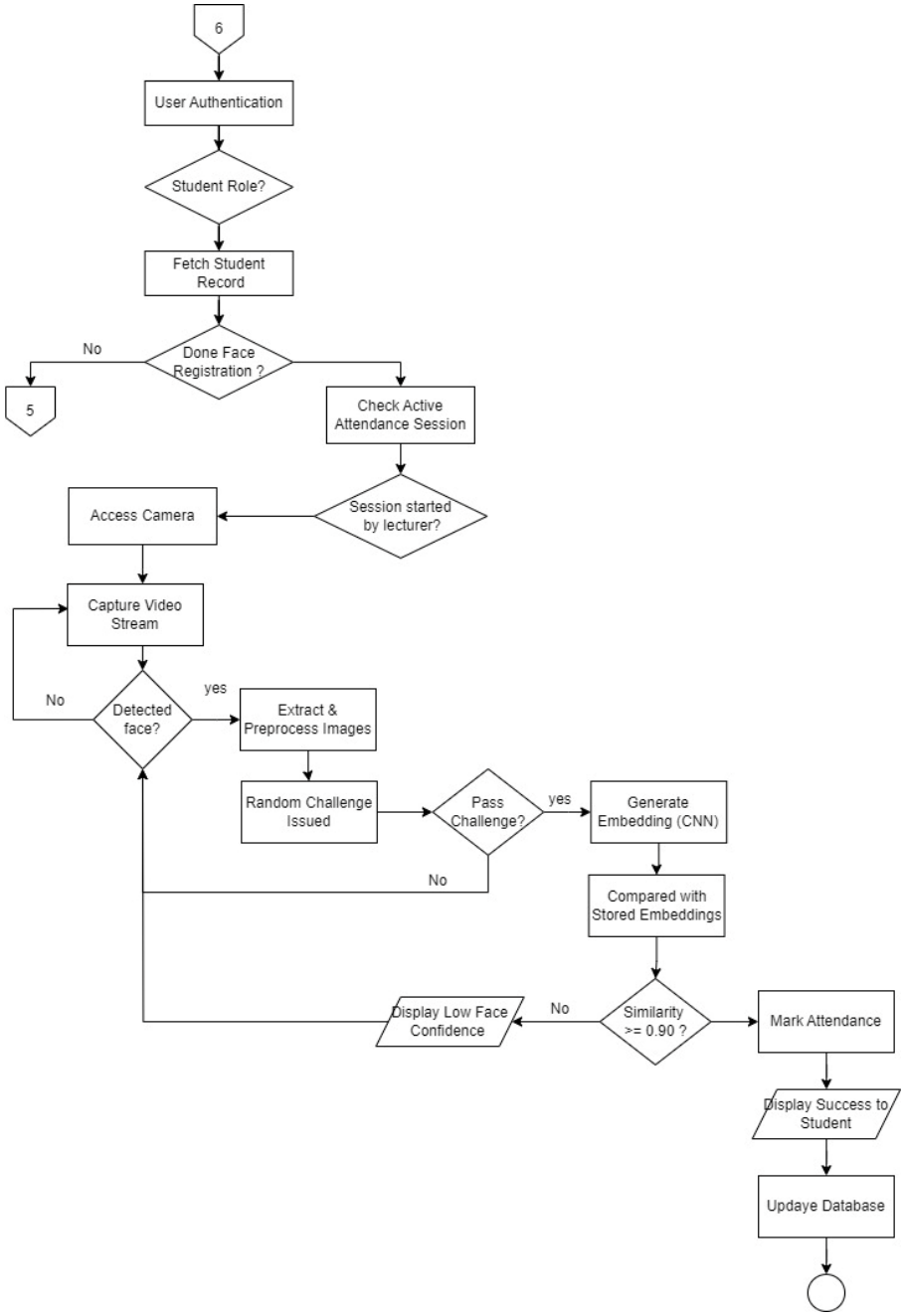



*Figure 4.2.8 System Flowchart (7/7) – Real-time Face Recognition for attendance*

The real-time face recognition attendance flow begins with verifying that the student is authenticated, and the lecturer has started an active session. The system performs an active liveness check, requiring actions such as blinking or head movements, to ensure the student is physically present. Once passed, the camera captures the student's face, which is detected and preprocessed for normalization. A CNN generates a 1024-dimensional embedding, compared against stored templates using cosine similarity. If the similarity meets the 0.90 threshold, attendance is logged with timestamp, status, and confidence score.

## 4.3 System Architecture and Component Interaction

The system is engineered upon a multi-layered client-server architecture, a strategic choice designed to deliver modularity, scalability, and the real-time responsiveness essential for a live attendance system. This layered model logically decouples the system's core responsibilities into six distinct, interoperable components: the Frontend Interface, Backend Server, AI-driven Face Processing Pipeline, Liveness Detection Module, Database System, and an Offline Model Training Pipeline.

This separation of concerns is paramount. It allows development teams to work on different components concurrently, facilitates independent updates (e.g., upgrading the frontend framework without altering backend logic), and enables flexible integration of new features. Most importantly, it isolates resource-intensive AI computations on the server-side, ensuring the client-side remains lightweight and responsive while meeting the stringent real-time performance requirements of video stream analysis.

*Table 4.3.1: System Architecture Components*

| Component | Key Technologies | Designed Functionality |
|---|---|---|
| Frontend Interface (Client-Side) | HTML5, CSS3, JavaScript (ES6+), Bootstrap 5 | The frontend serves as the sole point of interaction for all users (students and lecturers). It provides a clean, responsive Graphical User Interface (GUI) built with standard web technologies for maximum browser compatibility. Its primary responsibilities include:<br>• **User Authentication & Session Views:** Renders login/registration forms and dynamically displays different dashboards based on the user's role (student or lecturer).<br>• **Camera Access and Video Streaming:** Uses the browser's MediaDevices.getUserMedia() API to request camera access and stream the video feed to an HTML <video> element.<br>• **API Communication:** Utilizes JavaScript's fetch API or a library like axios to make asynchronous RESTful API calls to the backend server. This includes sending video frames |

| | | |
|---|---|---|
| | | for processing, submitting form data, and retrieving data (e.g., course lists, attendance records) to populate the UI.<br>• **Real-time Feedback:** Renders dynamic visual feedback received from the server, such as overlaying instructions ("Blink Twice"), progress bars for challenges, and success/failure messages directly onto the video stream interface. |
| Backend Server (API) | Flask (Python Web Framework), Waitress (WSGI Server) | The Flask server acts as the central nervous system of the application. It is a lightweight, stateless API gateway that orchestrates all interactions between the frontend, the AI modules, and the database.<br>• **Request Handling:** Defines a set of RESTful endpoints (e.g., /login, /register_face, /scan_attendance) to receive and process HTTP requests from the client.<br>• **Business Logic and Session Management:** Implements the core application logic, including user authentication, role-based access control (RBAC), and managing user sessions to maintain a logged-in state.<br>• **Module Integration:** Acts as the primary integrator. When a request for face recognition arrives, the backend calls the necessary functions within the Face Processing and Liveness Detection modules, passing the image data and awaiting a result. It then formats this result and sends it back to the client as a JSON response.<br>• **Database Abstraction:** Manages all communication with the SQLite database, handling data creation, retrieval, updates, and deletion based on API requests. |
| Face Processing Pipeline | Dlib, TensorFlow/ Keras, OpenCV | This is the core AI engine for identity verification. It operates as a sequential pipeline on the server-side, processing each frame received from the client.<br>• **Input:** Receives a raw video frame (as a byte array or base64 encoded string) from the backend. |

| | | |
|---|---|---|
| | | • **Step 1: Detection & Cropping:** Dlib's highly optimized HOG-based face detector scans the frame to identify and return the bounding box coordinates of any faces. The face region is then cropped for focused analysis.<br>• **Step 2: Preprocessing:** The cropped face image undergoes a series of normalization steps to make it robust to variations in lighting and camera quality.<br>• **Step 3: Embedding Extraction:** The preprocessed image is fed into the pre-trained custom Convolutional Neural Network (CNN). The CNN processes the image through its layers, outputting a dense 1024-dimensional vector (embedding) that mathematically represents the unique facial features.<br>• **Step 4: Recognition/Registration:** The final embedding is either stored in the database (for registration) or compared against existing embeddings using cosine similarity (for recognition). |
| Liveness Detection Module | OpenCV, Dlib (for facial landmarks) | • This security-critical module operates in tandem with the Face Processing Pipeline to prevent spoofing attacks. It is designed to run efficiently on every frame before committing to more computationally expensive recognition tasks. It confirms that the source of the video stream is a live human and not a static photo, video replay, or mask. Its two-tiered approach provides layered security. |
| Database System | SQLite | SQLite is chosen for its simplicity, serverless nature, and file-based storage, making it ideal for this system's scale and deployment ease.<br>• **Schema:** The database schema is relationally designed with tables<br>for users, facial_embeddings, courses, class_sections, enrol lments (a many-to-many join table between users and class_sections), and attendance_logs. |

| | | |
|---|---|---|
| | | • **Data Integrity:** Foreign key constraints are heavily utilized to ensure relational integrity. For example, an attendance log must be linked to a valid user and a valid class section, preventing orphaned or inconsistent records. |
| Model Training Pipeline | Python, TensorFlow/ Keras, NumPy, Scikit-learn | This is an entirely **offline** component, separate from the real-time application. Its purpose is to periodically retrain and improve the face recognition CNN.<br>• **Workflow:** The pipeline involves gathering and augmenting a large dataset of labeled face images, training the CNN model using techniques like triplet loss to learn discriminative embeddings, validating its performance on a held-out test set, and finally, exporting the trained model weights. These new weights can then be deployed to the production server to update the system's accuracy without any downtime. |

## 4.4 Module Design and Description

The system's architecture is implemented through a set of cohesive, functionally distinct modules. This modular design philosophy ensures that each part of the system has a single, well-defined responsibility. Modules communicate through clearly defined interfaces—primarily Flask API endpoints for client-server interaction and direct function calls or database queries for internal, server-side communication. This design simplifies development, debugging, and future enhancements.

### 4.4.1 Face Processing and Registration Module

This module forms the intelligent core of the system, encapsulating all AI-powered functionality required for robust and secure identity verification. It is a comprehensive pipeline that transforms raw video frames into actionable identity decisions.

- **Responsibilities (Detailed Breakdown):**
  1. **Face Detection:** The process begins with dlib.get_frontal_face_detector, a computationally efficient detector based on Histogram of Oriented Gradients (HOG) features. It is optimized for near-frontal faces, which aligns with the typical use case of a user facing their camera. For each frame, it returns a list of bounding boxes for all detected faces.

  2. **Preprocessing:** Before feature extraction, each detected face crop undergoes a mandatory normalization routine to counteract real-world environmental variations:
     - **Histogram Equalization:** Redistributes pixel intensities to enhance global contrast, especially useful in poorly or unevenly lit conditions.
     - **Noise Reduction:** A Gaussian blur or median filter is applied to remove minor camera sensor noise that could degrade embedding quality.
     - **Sharpening:** A sharpening kernel is applied to enhance edges and fine details of facial features (eyes, nose, mouth), providing the CNN with more distinct information.

  3. **Feature Extraction:** The preprocessed face crop is resized to the CNN's required input dimensions (e.g., 160x160 pixels) and passed into the custom CNN. The network, architected with deep residual blocks, is designed to learn a highly discriminative function. The final output is a 1024-dimensional floating-point vector (embedding)

where faces of the same person are clustered closely together in the vector space, and faces of different people are far apart.

4. **Registration:** The registration process is designed for robustness. Instead of relying on a single image, the user is prompted to provide multiple samples (e.g., looking straight, slightly left, slightly right, smiling). The system extracts embedding for each valid sample. These embeddings are then **averaged** to produce a single, composite identity vector. This averaging process creates a more generalized and resilient representation of the user's face, making it less susceptible to minor variations in pose and expression during future recognition attempts.

5. **Recognition:** During an attendance attempt, a new embedding is extracted from the user's live video feed. This "probe" embedding is compared against all "gallery" embeddings stored in the database using **cosine similarity**. This metric measures the cosine of the angle between two vectors, effectively judging their orientation rather than their magnitude. A similar score closes to 1.0 indicates a near-perfect match, while a score near 0.0 indicates orthogonality (no match). A match is confirmed only if the highest similarity score exceeds a fine-tuned threshold (e.g., 0.85), balancing the trade-off between False Acceptance Rate (FAR) and False Rejection Rate (FRR).

6. **Liveness Detection:** This is an integrated, non-negotiable security layer that precedes recognition.
   - **Tier 1: Passive Anti-Spoofing:** This runs silently and continuously on the video stream.
     - **Texture Analysis (cv2.Laplacian):** The Laplacian operator measures the second derivative of the image, which is high in areas of rapid intensity change (like edges and fine textures). A real face, with its pores and subtle skin texture, will have a significantly higher Laplacian variance than a blurry, out-of-focus, or printed photo displayed on a screen.
     - **Motion Analysis (cv2.absdiff):** This function calculates the per-pixel difference between the current frame and the previous one. Even a person trying to stay still exhibits natural, subtle movements (breathing, micro-

expressions, slight sways). A static image will have a motion score of zero, and a replayed video often has unnatural motion patterns or compression artifacts that can be detected.

- **Tier 2: Active Challenge-Response:** This tier is triggered as a definitive verification step.
  - **Challenge Issuance:** The system's random selection of a challenge from a pool ("blink_twice", "smile", etc.) makes it extremely difficult for an attacker to pre-record a video that can spoof the system.
  - **Real-time Monitoring:** The system uses Dlib's 68-point facial landmark predictor to precisely track facial components. The Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) are calculated from these landmarks to algorithmically detect blinks and mouth movements, while Head Pose Estimation provides the yaw and pitch angles to verify head turns and nods. This is a robust, metric-driven verification process, not simple image matching.

### 4.4.2 Active Challenge (Liveness Detection)

The ***EnhancedLivenessDetector*** class is a stateful object designed to manage the complexities of liveness verification over a sequence of frames, not just a single snapshot. Its use of collections.deque is critical for temporal analysis, allowing it to detect patterns like the open-close-open sequence of a blink.

*Table 4.4.2.1 Threshold Value for Parameters*

| Parameter | Value | Description |
|---|---|---|
| EAR_THRESHOLD | 0.23 | Threshold for detecting a closed eye state. |
| EAR_CONSEC_FRAMES | 3 | Number of consecutive frames an eye must be "closed" to register a blink. |
| MAR_THRESHOLD | 0.65 | Threshold for detecting an "open mouth" challenge. |
| HEAD_POSE_THRESHOLD | 25° | Yaw/pitch angle required to satisfy head turn/nod challenges. |
| CHALLENGE_TIMEOUT | 15s | Maximum time allowed for a user to complete a challenge. |
| TEXTURE_THRESHOLD | 100 | Minimum Laplacian variance scores to pass the texture test. |

**Elaboration of Key Functions:**

**a. Metric Calculation:**

- ***calculate_ear()*** / ***calculate_mar():*** These functions take the 68 facial landmarks as input. They calculate the Euclidean distance between specific vertical and horizontal landmark points around the eyes and mouth, respectively, to compute a single ratio. This ratio is remarkably consistent for an open state and changes predictably during a blink or mouth opening, making it an excellent biometric indicator.
- ***calculate_head_pose():*** This function implements a standard computer vision technique. It uses a generic 3D model of a human face and the corresponding 2D landmark locations detected in the frame. By using cv2.solvePnP (Perspective-n-Point), it solves for the 3D rotation and translation that would project the 3D model points onto the observed 2D image points, thereby yielding the head's pitch, yaw, and roll angles in degrees.

- ***calculate_texture_score()*** and ***calculate_motion_score()*** are the implementations of the passive checks described in 4.4.1.

**b. Challenge-Response Logic:**

- select_random_challenge(): This function ensures unpredictability. It maintains a list of available challenges and uses Python's random.choice() to pick one for the current session.
- process_challenge(): This is the core state machine of the module. For a "blink_twice" challenge, it might track a blink_counter state variable. It continuously monitors the EAR, and when the pattern for a blink is detected (EAR drops below EAR_THRESHOLD for EAR_CONSEC_FRAMES then rises again), it increments the counter. It provides feedback ("Blink once more") until the condition (blink_counter == 2) is met, at which point it returns to success status. If the CHALLENGE_TIMEOUT is reached before completion, it returns a failure.

**c. Core Orchestration & State Management:**

- ***analyze_frame():*** This is the public-facing method of the class. For every frame it receives, it performs all metric calculations. It first checks passive liveness scores. If they pass, and an active challenge is in progress, it calls process_challenge(). The function returns a single, comprehensive dictionary containing a final boolean liveness_passed, a confidence_score, the status_message for the user (e.g., "Turn your head to the left"), and any other relevant data.
- ***reset_state():*** Clears temporary data like metric history buffers and counters. Critically, it is designed not to clear an active challenge in progress, allowing the user to seamlessly continue their attempt across multiple API calls if needed.

**d. Video Stream Generation:**

- ***generate_frames():*** This Flask-specific function is implemented as a Python generator. It sits in a while True loop, continuously capturing frames from the camera. Inside the loop, it calls ***analyze_frame()*** to get the current liveness status. It then uses OpenCV's drawing functions (cv2.putText, cv2.rectangle) to overlay the feedback (bounding boxes, instructions, landmark points) directly onto the frame. Finally, it encodes the frame as a JPEG and yields it as part of a multipart/x-mixed-replace HTTP response.

This technique efficiently streams video to the browser. The "instruction stabilization logic" prevents the UI text from flickering by only updating the displayed instruction when the state from ***analyze_frame()*** changes.

**e. Key Endpoints:**

- /***video_feed:*** This endpoint returns a Response object with the generate_frames() generator as its source. The browser interprets this special MIME type as a continuous video stream, updating the <img> tag's src attribute with each new frame yielded by the server.
- ***/register_face***: This is a standard POST endpoint. The client calls this endpoint after the liveness check has been successfully passed. The backend then captures a few high-quality frames, extracts their embeddings, averages them, and inserts the final identity vector into the facial_embeddings table, linking it to the user's ID.

### 4.4.3 User and Course Management Module (Lecturer)

This module provides the administrative backbone of the system, tailored specifically for lecturers. It is a secure, role-protected section of the application that allows for the complete management of the academic structure within which attendance is recorded.

**Key Endpoints and Functional Breakdown:**

1. **Course Lifecycle Management:**
    - POST /courses: When a lecturer submits the "Create Course" form, this endpoint receives the course name, code, and description. It performs validation (e.g., ensuring the course code is unique) before inserting a new record into the courses table.
    - POST /delete_course/<id>: This is a critical endpoint that must handle data integrity. Upon invocation, it not only deletes the specified course from the courses table but also triggers a cascading delete (or manual deletion logic) to remove all associated class_sections, enrollments, and attendance_logs to prevent orphaned records in the database.

2. **Class Section Management**:
    - GET /manage_sections/<course_id>: This endpoint performs a database query to select all records from the class_sections table where the course_id foreign key matches the one provided in the URL. The results are returned as JSON for the frontend to display.
    - GET /view_section/<section_id>: This provides a more detailed view by performing a JOIN query across the users, enrollments, and class_sections tables to retrieve a list of all students enrolled in that specific section.

3. **Student Enrollment**:
    - POST /enroll_student: This endpoint is typically called from the section management interface. It receives a student_id and a section_id and creates a new entry in the enrollments table, formally linking a student to a class. The logic includes checks to prevent duplicate enrollments.

4. **Attendance Session and Data Management:**
    - GET /manage_attendance/<section_id>: This endpoint serves the main dashboard for a live class. It provides a real-time view of attendance. The front end might use

techniques like periodic polling (e.g., calling an API endpoint every 5 seconds) or WebSockets to get live updates of which students have successfully checked in. The "Start Session" and "Stop Session" buttons on this page trigger the corresponding functions in the Attendance Workflow Module.

- POST /export_reports: This endpoint handles complex data aggregation. It queries the attendance_logs table, joining with users, courses, and class_sections to gather comprehensive data. It then uses libraries like Pandas to structure the data and openpyxl (for Excel), reportlab (for PDF), or Python's built-in csv module to generate a file, which is then sent back to the user as a file download.

**Architectural and Implementation Notes:**

- The get_db() helper function is a standard Flask pattern for managing database connections. It ensures that a single database connection is established per request and is properly closed (torn down) after the request is complete, preventing resource leaks.
- The frontend dashboards are designed for data visualization, potentially including charts or graphs showing attendance rates over time, lists of frequent absentees, and other analytics to help lecturers quickly assess student engagement.

### 4.4.4 Attendance Workflow Module

This module orchestrates the entire real-time attendance process, acting as the bridge between the AI-driven recognition and the academic database structure. It enforces the business rules that govern a valid attendance-taking session.

**Responsibilities (Detailed Workflow):**

1. **Session Control:** A lecturer initiates a session via their dashboard, calling start_session(). This creates a new record in an attendance_sessions table with the section_id and a start_time, and sets an is_active flag to TRUE.

2. **Attempt Validation:** When a student attempts to check in via /scan_attendance, the first action the backend takes is to query the attendance_sessions table to find an active session (is_active = TRUE) for the section(s) the student is enrolled in. If no active session is found, the attempt is rejected immediately with a message like "Attendance is not currently open for this class."

3. **Uniqueness Enforcement:** If an active session is found, the system then checks the attendance_logs table to see if a record already exists for the current student_id and the active session_id. If one exists, the attempt is rejected with the message "You have already been marked present." This prevents duplicate entries.

4. **Record Storage:** Only after passing the session and uniqueness checks does the system proceed with liveness and recognition. Upon a successful match, mark_attendance() inserts a new record into attendance_logs, including the student_id, session_id, a precise timestamp, and the recognition confidence score.

5. **Real-time Feedback:** The endpoint returns a clear JSON response to the student's client (e.g., { "status": "success", "message": "Welcome, [Student Name]! You are marked present." } or { "status": "failure", "message": "Face not recognized. Please try again." }).
   - **Key Functions (Detailed Logic):**
     - start_session(section_id): Inserts a new row into the attendance_sessions table, setting is_active = 1.

- stop_session(session_id): Updates the corresponding row in attendance_sessions, setting is_active = 0 and recording an end_time. No further check-ins for this session will be accepted.
- mark_attendance(student_id, session_id, confidence_score): Performs the final INSERT operation into the attendance_logs table.
- get_attendance_records(section_id): Retrieves and joins all relevant logs for a section, used to populate reports and the lecturer's real-time dashboard.

- **Key Endpoints:**
  - /video_feed: As described before, this streams the live camera feed with overlays for the student during the check-in process.
  - /scan_attendance: This is the endpoint that executes the entire validation and recognition workflow described above. It is the primary transactional endpoint for students.

**Additional Notes:**

- The system's integrity is fundamentally guaranteed by the strict, sequential workflow: **Session Active? -> Not Already Checked In? -> Liveness Verified? -> Face Recognized? -> Mark Present**. A failure at any step terminates the process. Logging the recognition confidence score provides crucial data for auditing and troubleshooting any disputes over attendance records. For example, a lecturer can review low confidence matches if a student reports an issue.

## 4.5 Database Design

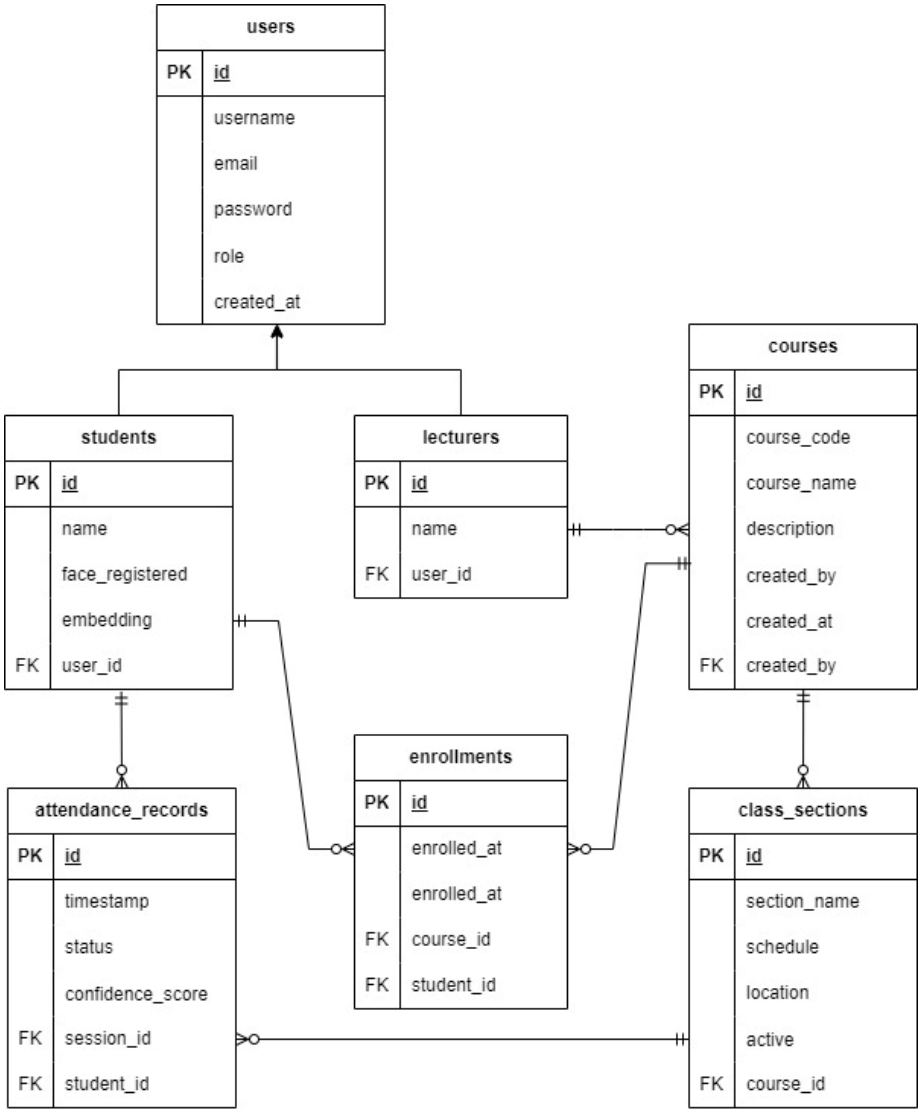



*Figure 4.5.1 Entity Relationship Diagram*

The database schema underpins the real-time face recognition attendance system by linking **users**, **students**, **lecturers**, **courses**, **class sections**, **enrollments**, and **attendance records**. Users are authenticated and distinguished as students or lecturers through one-to-one relationships. Students store facial embeddings and registration data, while lecturers manage courses with multiple class sections. A many-to-many relationship between students and sections is resolved via the enrollments table. Attendance records capture timestamps, status, and confidence scores. This design ensures integrity, scalability, and efficient performance, enabling seamless management of authentication, enrollment, and automated attendance tracking.

# Chapter 5
# System Implementation

## 5.1 Hardware Setup

This chapter details the practical implementation of the face recognition attendance system, translating the architectural designs and methodologies from previous chapters into a functional, end-to-end prototype. It provides a comprehensive account of the hardware and software environments, crucial system configurations, and the operational workflows from an end-user perspective. Furthermore, this chapter discusses the technical challenges encountered during the development lifecycle and the strategic resolutions implemented to overcome them, culminating in a robust and validated system.

*Table 5.1 Hardware Specifications*

| **Description** | **Specifications** |
|---|---|
| Model | Victus 16-r0326TX |
| Processor | Intel Core i7-13700HX |
| Operating System | Windows 11 |
| Graphic | NVIDIA GeForce RTX4060 |
| Memory | 16GB DDR5-4800 MHz |
| Storage | 512GB |

## 5.2 Software Setup

The system was engineered using a carefully selected stack of open-source libraries and frameworks. Each component was chosen for its proven stability, extensive community support, and specific strengths in computer vision, deep learning, web application development, and data management.

*Table 5.2 Software Specifications*

| **Software Component** | **Version** | **Descriptions** |
|---|---|---|
| Python | 3.12.7 | The primary programming language, chosen for its extensive scientific computing ecosystem and robust support for machine learning libraries |
| Flask | 2.3.2 | A lightweight and modular micro web framework used to build the backend server, RESTful API endpoints, and render the user interface. |
| SQLite | 3.42.0 | Lightweight relational database engine for local data storage |
| Dlib | 64 Face Detector | A powerful C++ library with Python bindings, utilized for its highly accurate facial landmark detection, which is fundamental to the liveness detection module |
| TensorFlow | 2.12.0 | The core deep learning framework used for building, training, and deploying the custom CNN model for face feature extraction |
| OpenCV | 4.8.0 | The standard library for real-time computer vision, employed for video capture from the webcam, image preprocessing, and rendering visual feedback on video frames |

**Supporting Libraries and Development Environment:**

- **NumPy:** The foundational package for numerical computing in Python, used for efficient manipulation of multi-dimensional arrays, particularly for handling image data and facial embedding vectors, and for performing mathematical operations like cosine similarity.
- **Pandas:** A powerful data analysis library utilized for managing attendance records in a structured format (DataFrame) and for generating and exporting attendance reports in CSV format.
- **Flask-Login & Flask-Session:** Extensions for Flask that provide robust, secure, and role-based user session management, ensuring that students and lecturers have access only to their authorized functionalities.
- **Kaggle:** The cloud-based platform was leveraged for its free access to high-performance NVIDIA T4 GPUs, which significantly accelerated the initial and more demanding phases of model training.
- **Visual Studio Code (VS Code):** The primary Integrated Development Environment (IDE), chosen for its excellent Python support, integrated debugging tools, seamless Git version control integration, and extensive ecosystem of extensions.

## 5.3 Settings and Configuration

To ensure the system is stable, performant, and maintainable, a precise set of configurations was established. These settings are decoupled from the application logic, allowing for easy adjustments during development, testing, and deployment without requiring code changes.

- **Automated Database Initialization:**

  The SQLite database is designed to initialize itself automatically. The system uses an **Object-Relational Mapper (ORM)** via Flask-SQLAlchemy. Upon the first application run, it checks for the database file's existence. If not found, db.create_all() is invoked, programmatically generating the complete schema from the model definitions. This eliminates manual setup errors and ensures a pristine database state for any new deployment.

- **Optimized Model Loading (Eager Loading)**

  To achieve real-time performance, the CNN model is loaded into memory only once at server startup using tf.keras.models.load_model(). The loaded model object is stored in a global variable. Every subsequent recognition request access this in-memory model, preventing the significant I/O overhead of disk access and allowing for near-instantaneous inference.

- **Structured Data Storage**

  Facial images captured during registration are stored in a highly organized manner. The file system contains a root dataset/ directory. Within it, a unique sub-directory is created for each student, named after their unique student_id (e.g., dataset/1001/). This provides a logical repository of raw biometric data, invaluable for auditing and future model retraining.

- **Empirically Tuned Recognition Threshold**

  The cosine similarity threshold, which governs identity matching, was empirically set to **0.85**. This value was determined through rigorous testing on a validation dataset. It represents the optimal balance point on the **Receiver Operating Characteristic (ROC) curve**, minimizing the **False Acceptance Rate (FAR)** to ensure security, while maintaining a low **False Rejection Rate (FRR)** for a smooth user experience.

- **Development vs. Production Server Configuration:**

  During development, the lightweight Flask development server was used on http://localhost:5000 for its convenience and debugging features. For a production deployment, this is switched to a production-grade **WSGI (Web Server Gateway Interface)** server like **Gunicorn** or **Waitress**, which is designed to handle multiple concurrent requests robustly and efficiently.

- **Efficient Embedding Storage**

  The 1024-dimensional facial embeddings are stored as **Binary Large Objects (BLOBs)** in the SQLite database. The NumPy array is serialized into a compact binary format, which is significantly more space-efficient and allows for much faster retrieval and deserialization compared to storing it as a text-based format like JSON.

- **Centralized Configuration Management**

  All system-wide parameters are consolidated into a central config.py file. This includes:

  - **File Paths:** MODEL_PATH, DATASET_DIR
  - **Model Parameters:** IMAGE_SIZE = (64, 64)
  - **Operational Thresholds:** RECOGNITION_THRESHOLD = 0.85, EAR_THRESHOLD = 0.23, etc.

  This approach enhances modularity and simplifies maintenance, allowing parameters to be tuned without altering core application logic.

- **Environmental Variable Management**

  For enhanced security and portability, sensitive configurations like the application's SECRET_KEY and DATABASE_URI are not hardcoded, even in config.py. They are managed as **environmental variables**. A .env file is used during local development, and these variables are set directly on the production server. This practice prevents secret keys from being committed to version control.

- **Secure Session Management**

  The application is configured with a strong, randomly generated SECRET_KEY. This key is essential for Flask's session management, as it is used to cryptographically sign the

session cookie. This signature prevents clients from tampering with their session data (such as their user ID or role) before sending it back to the server.

- **Application Logging**

  A robust logging mechanism is configured using Python's built-in logging module. The application logs key events to a file, including user logins, course creation, attendance session start/stop events, and, crucially, any errors or exceptions that occur. This provides an essential audit trail for troubleshooting and monitoring system health.

## 5.4 System Operation

The system's operation is designed around intuitive, role-based workflows for Students and Lecturers. Access is governed by a secure authentication mechanism that directs users to the appropriate dashboard upon successful login, ensuring a clear separation of functionalities.

All user interactions begin at the authentication portal. New users must first register an account by providing a username, email, password, and selecting their designated role (Lecturer or Student), as shown in Figure 5.4.1.

Existing users can log in using their credentials (Figure 5.4.2). Upon successful authentication, the system's role-based access control directs them to their respective dashboards.

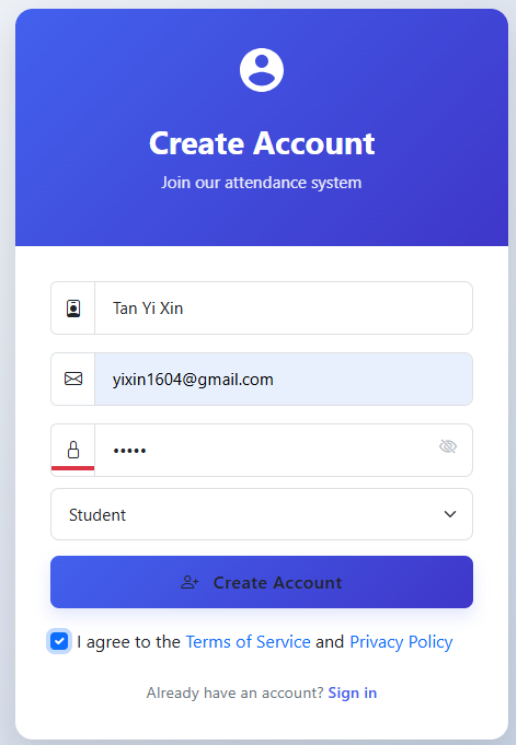



*Figure 5.4.1 User Registration Page*

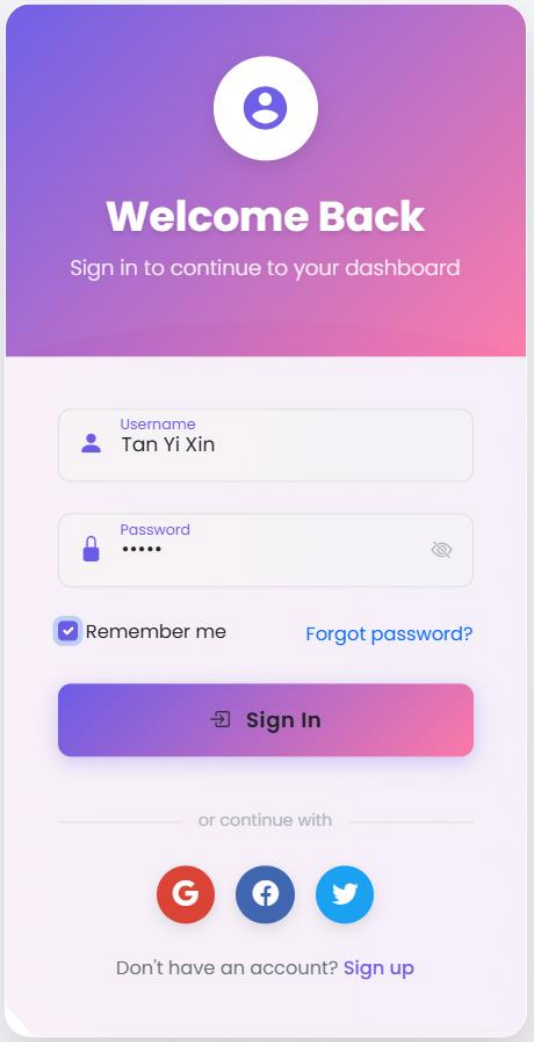



*Figure 5.4.2 User Login Page*

### 5.4.1 Face Registration with Active Liveness Detection

Upon their first login, students without a registered face are guided to complete the biometric enrollment process (Figure 5.4.1.1). This critical step ensures the integrity of the biometric database.

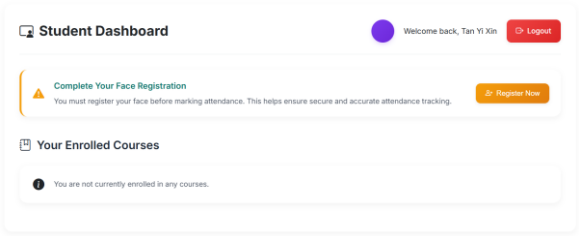



*Figure 5.4.1.1 Student Dashboard (Prompting Registration)*

The system activates the webcam (Figure 5.4.1.2) and initiates a guided registration featuring active challenge-response liveness detection. The student is prompted to perform a series of real-time facial gestures—such as blinking, smiling, and turning their head to verify they are a live and present person.

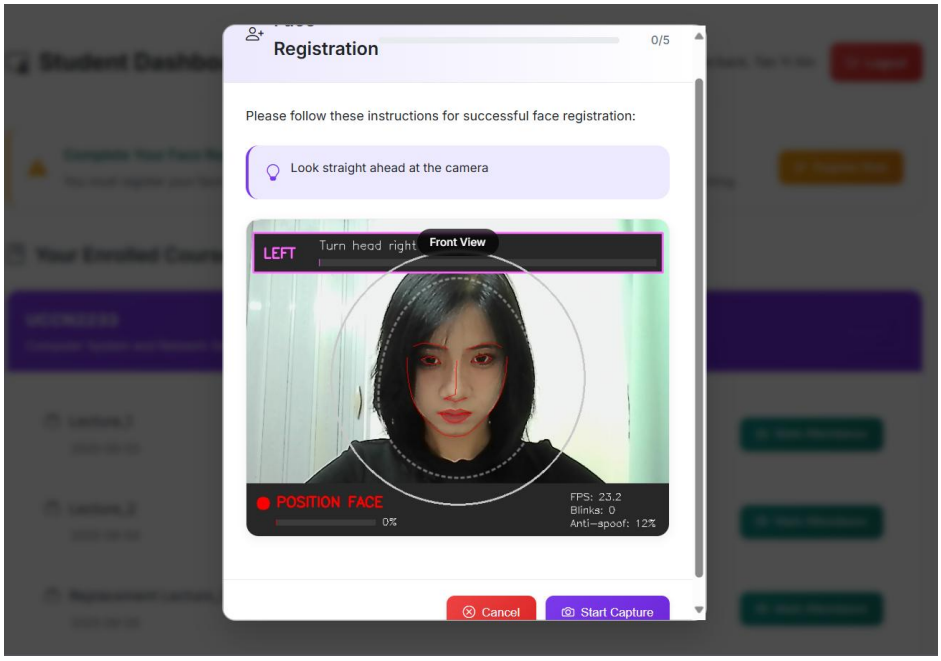



*Figure 5.4.1.2 Head Movement (Turn Right) – student turns head right*

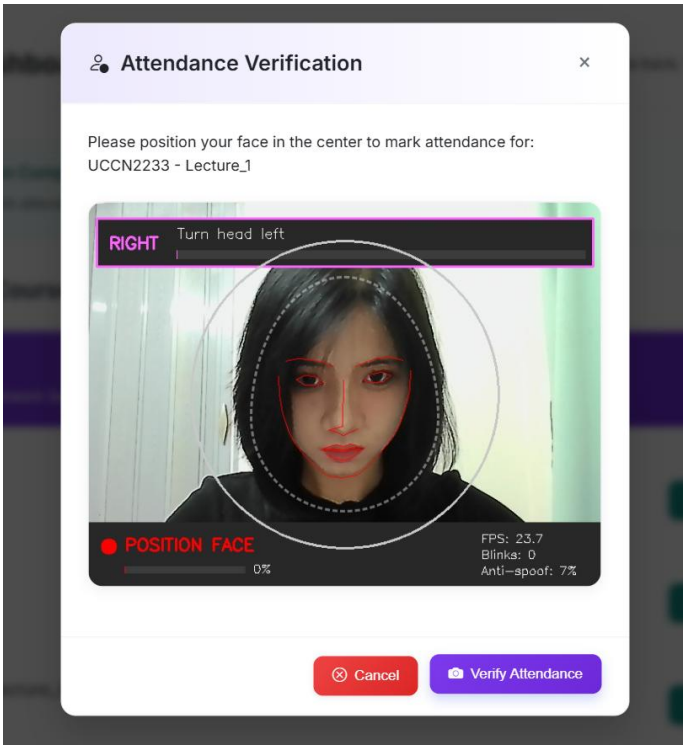



*Figure 5.4.1.3 Head Movement (Turn Left) – student turns head left*

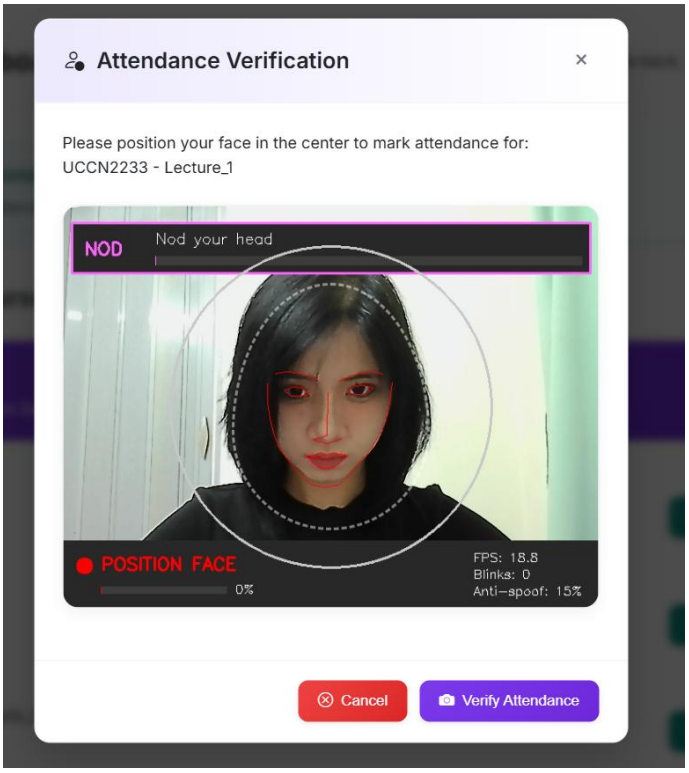



*Figure 5.4.1.4 Head Movement (Nod Head) – student nods head up and down*

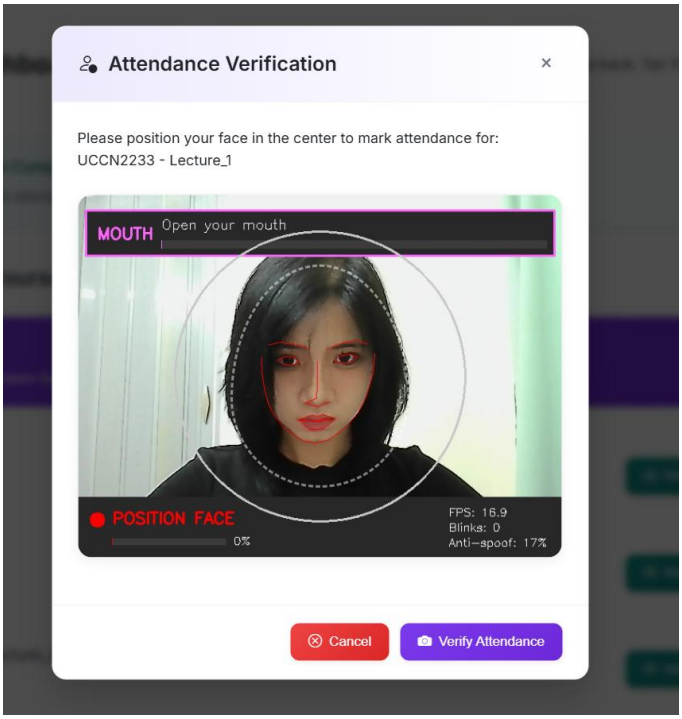



*Figure 5.4.1.5 Open mouth*

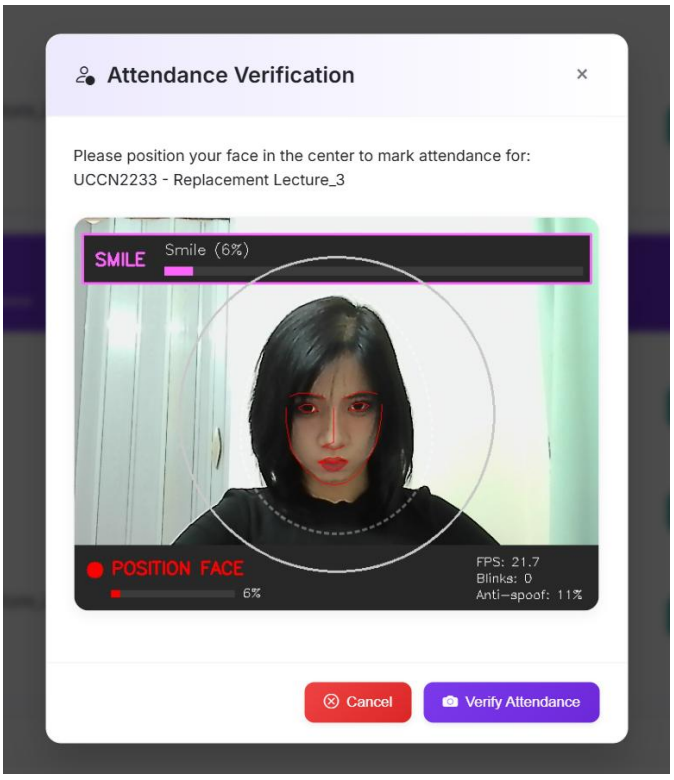



*Figure 5.4.1.6 Smile Detection – student is prompted to smile*

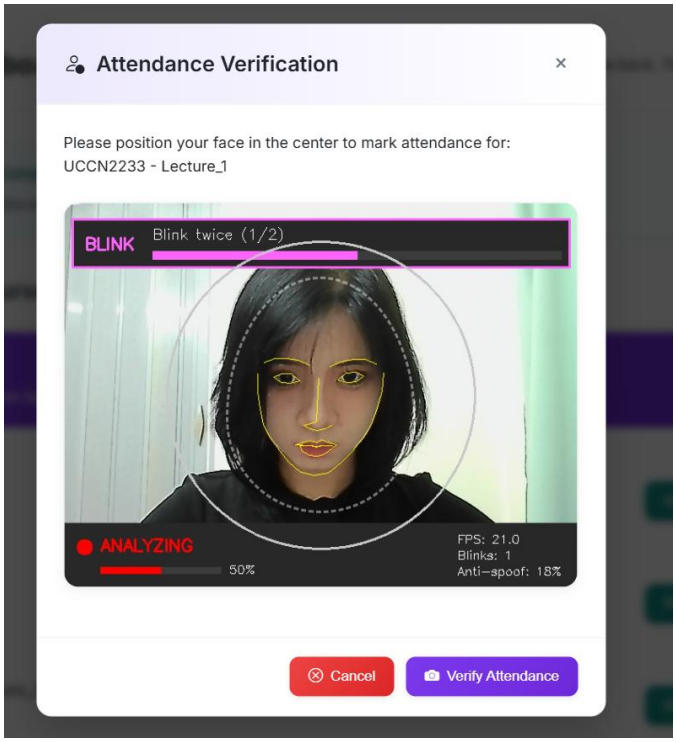



*Figure 5.4.1.7 Active-Blink Detection – student is prompted to blink*

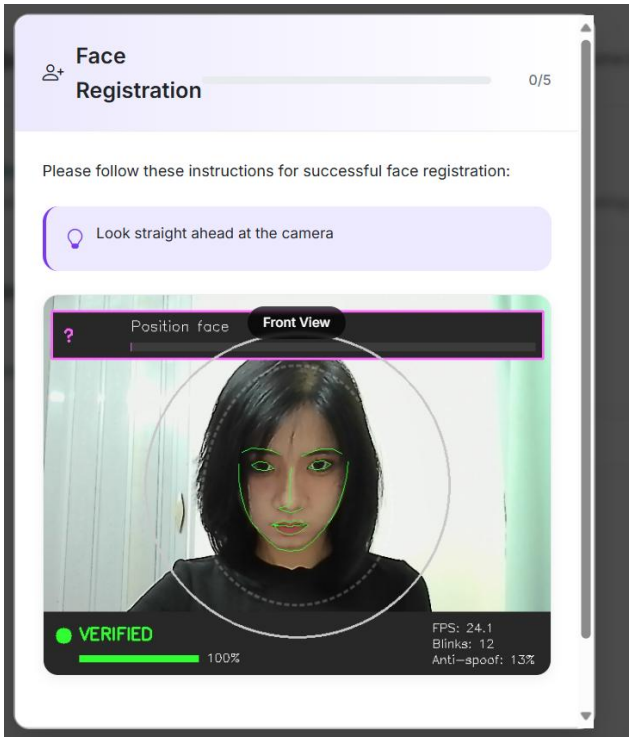



*Figure 5.4.1.8 Liveness Verified Confirmation*

Once liveness is confirmed (Figure 5.4.1.8), the system automatically captures five high-quality facial images, generates the 1024-dimensional embedding, and securely stores it. A

confirmation message is displayed upon completion, and the student dashboard is updated to reflect the registered status (Figure 5.4.1.10).

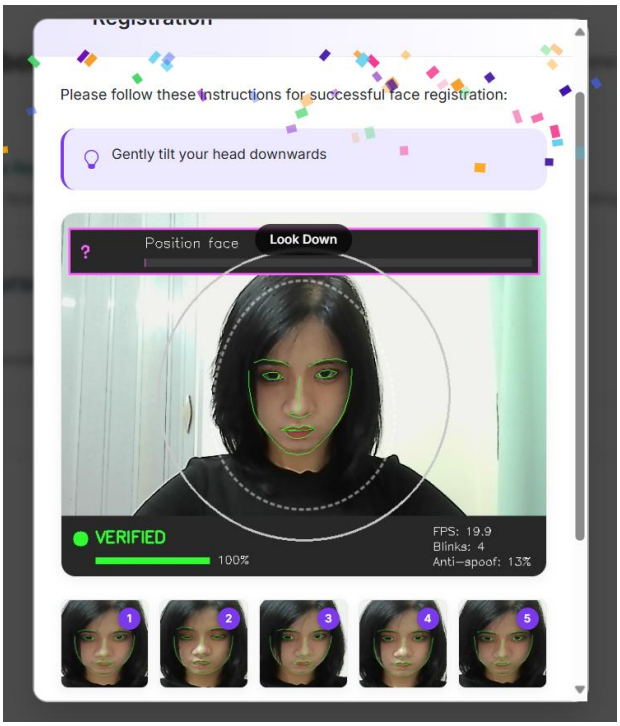



*Figure 5.4.1.9 Face Registration Completed – five face images are auto-captured*

This guided and interactive method strengthens system security by preventing spoofing through printed photos or videos during the registration phase.

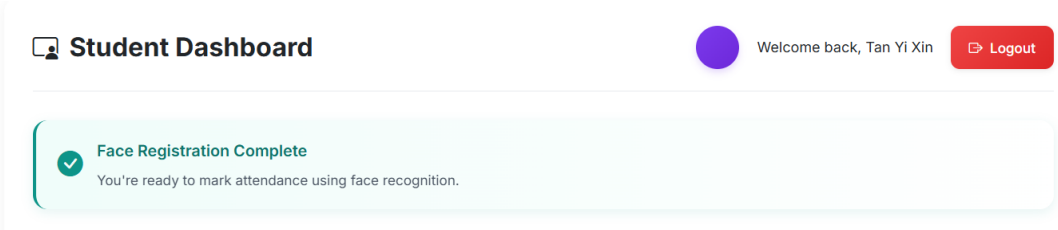



*Figure 5.4.1.10 Student Dashboard Page (Face Registration Completed)*

### 5.4.2 Attendance Scanning Process *(with Active Challenge-Response)*

To mark attendance, a student selects their class section and initiates the face scanning process. The system again performs the active liveness check (as discussed in Chapter 5.4.1 during face registration) to prevent real-time spoofing. Upon successful verification, the live video frame is passed to the CNN model to extract a feature vector. This vector is compared against the student's stored embedding using cosine similarity. If the score meets or exceeds the **85% threshold**, attendance is successfully recorded, and immediate feedback is provided to the user (Figure 5.4.2.1).

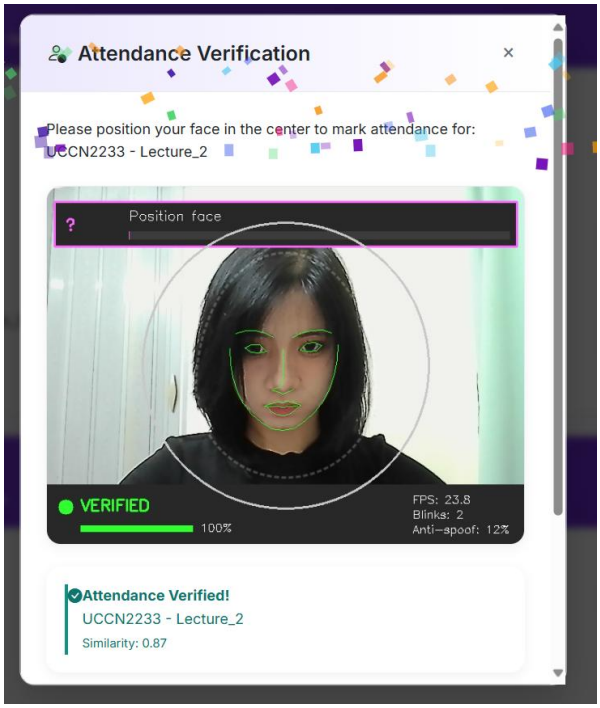



*Figure 5.4.2.1 Attendance Verification Modal with Live Similarity Score*

### 5.4.3 Lecturer Dashboard Functions

The Lecturer Dashboard serves as a centralized control panel for all academic and administrative tasks, providing comprehensive tools to manage courses, class sections, student enrollments, and attendance records (Figure 5.4.3.1 & Figure 5.4.3.2).

- **Manage Courses:** Lecturers can manage courses and add new sections.
- **Manage Class Sections:** After creating a course, lecturers can add class sections for it.
- **Enroll Students:** Students can be enrolled into specific course sections.
- **Export Reports:** Attendance reports can be exported in various formats
-

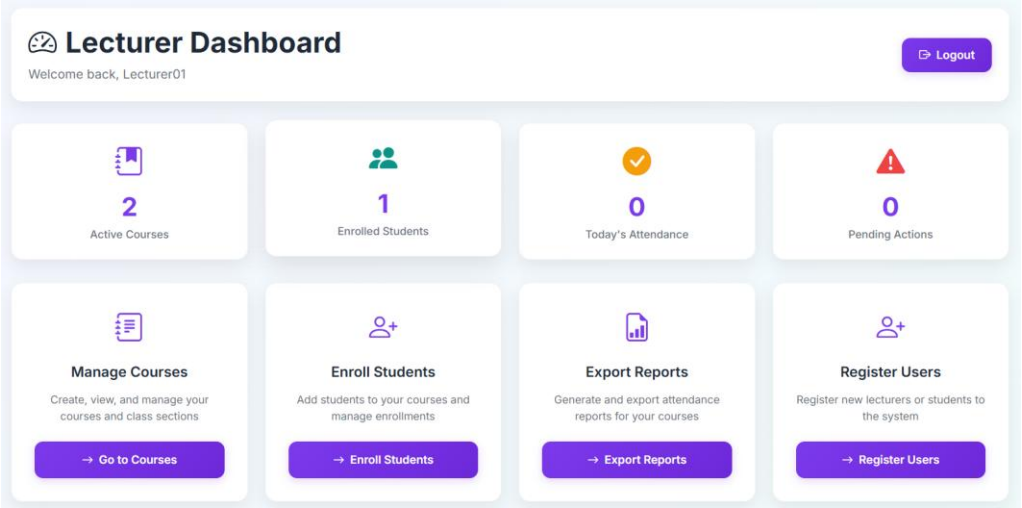



*Figure 5.4.3.1 Lecturer Dashboard Overview (1/2)*

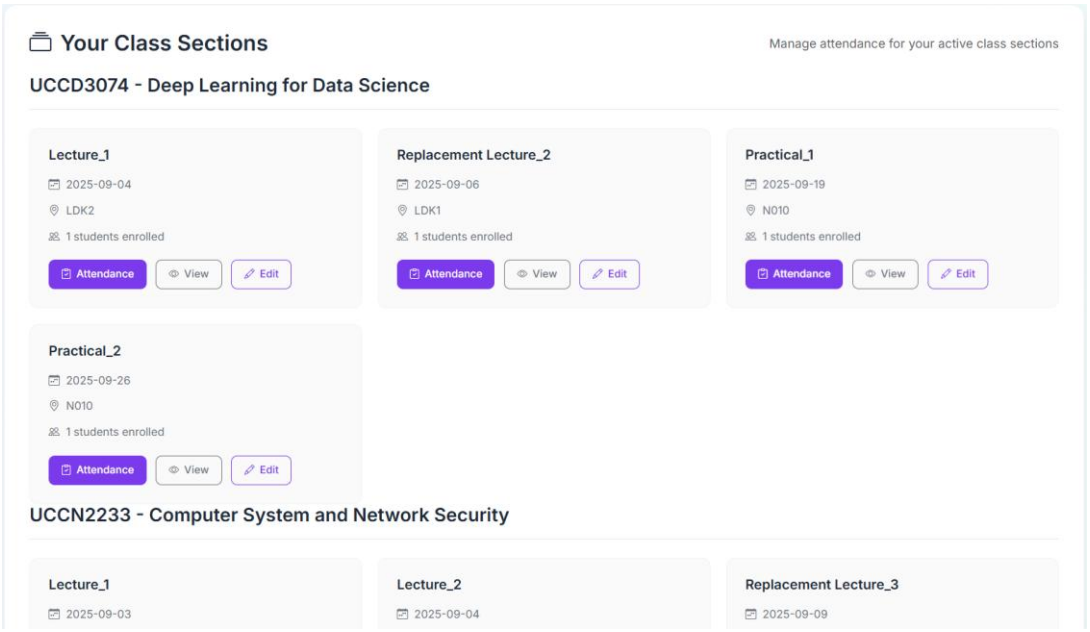



*Figure 5.4.3.2 Overview of Lecturer Dashboard (2/2)*

1. **Course, Section, and Enrollment Management**

   Lecturers can create and manage their courses, add multiple class sections for each course, and enroll students into specific sections. This workflow ensures that the system's structure accurately reflects academic organization,

   a. **Course Management**

      Lecturers can manage courses (e.g., UCCD3074 Deep Learning for Data Science) and add new sections.

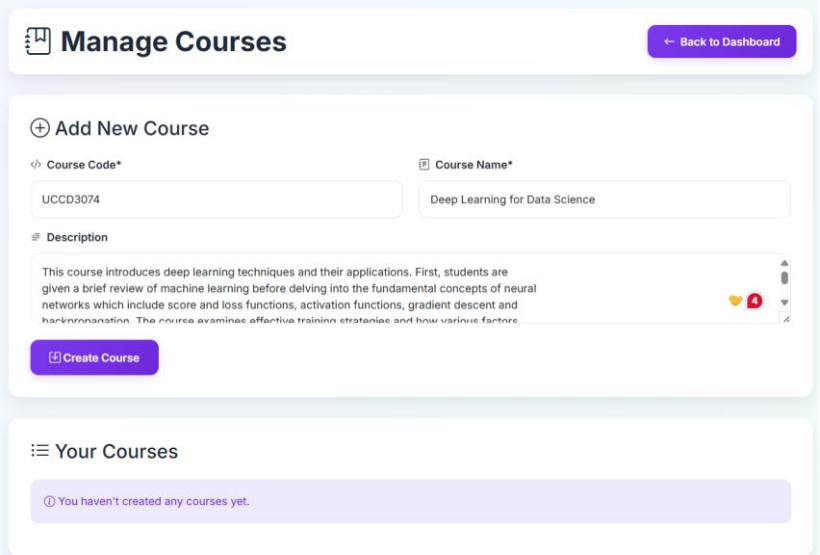



*Figure 5.4.3.3 Course Management Dashboard (1/2)*

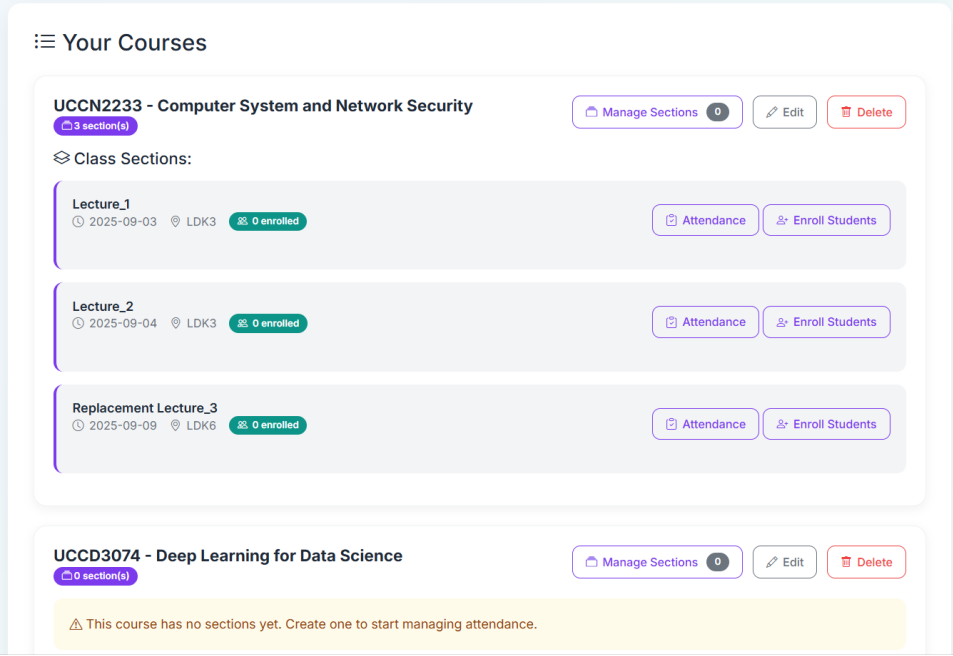



*Figure 5.4.3.4 Course Management Dashboard (2/2)*

**b. Section Management**

After UCCD3074 Deep Learning for Data Science course is created, lecturer may manage section for the course (CRUD):

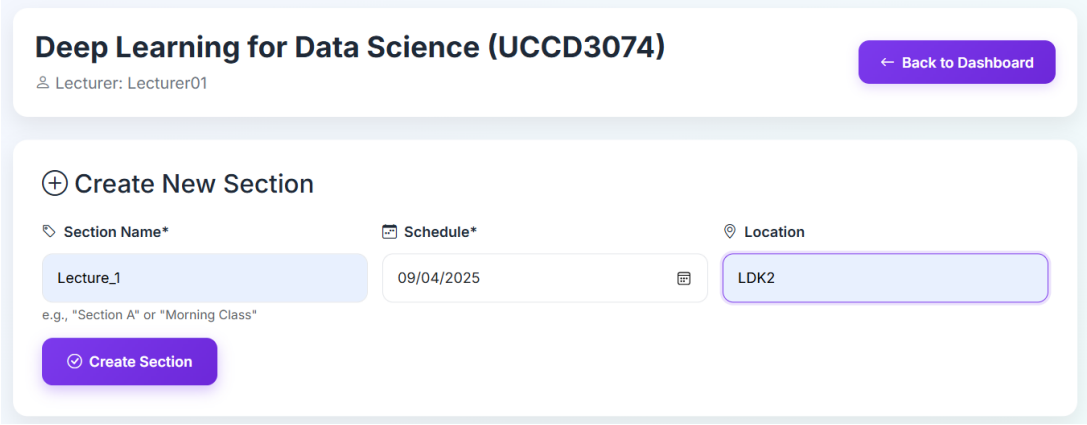



*Figure 5.4.3.5 Class Section Creation*

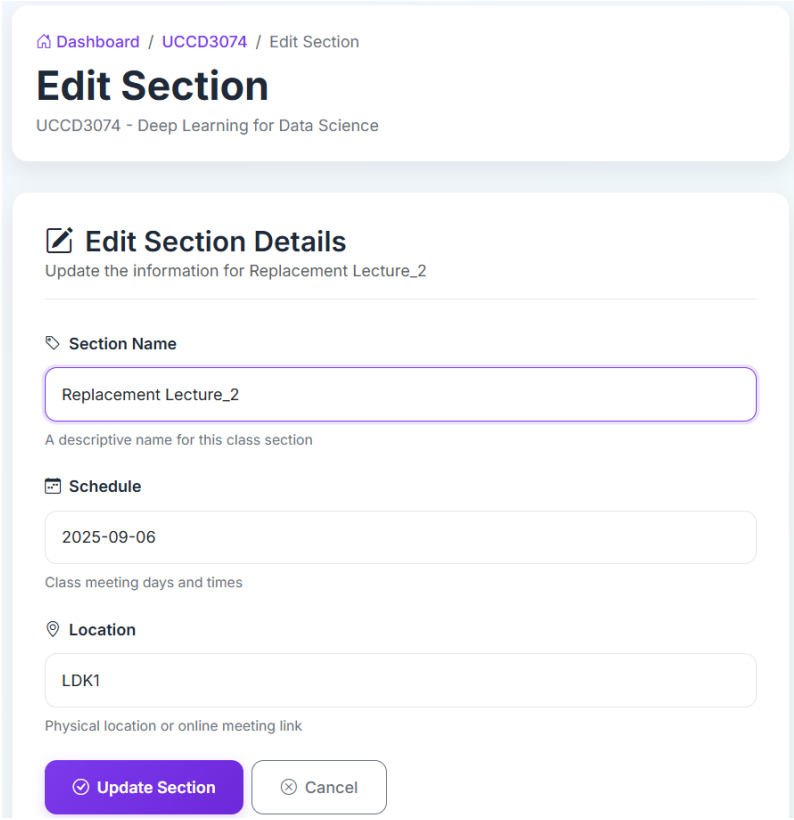



*Figure 5.4.3.6 Edit Class Section*

The class section created will appear in “Existing Sections” upon succeed creation of the class section for that course:

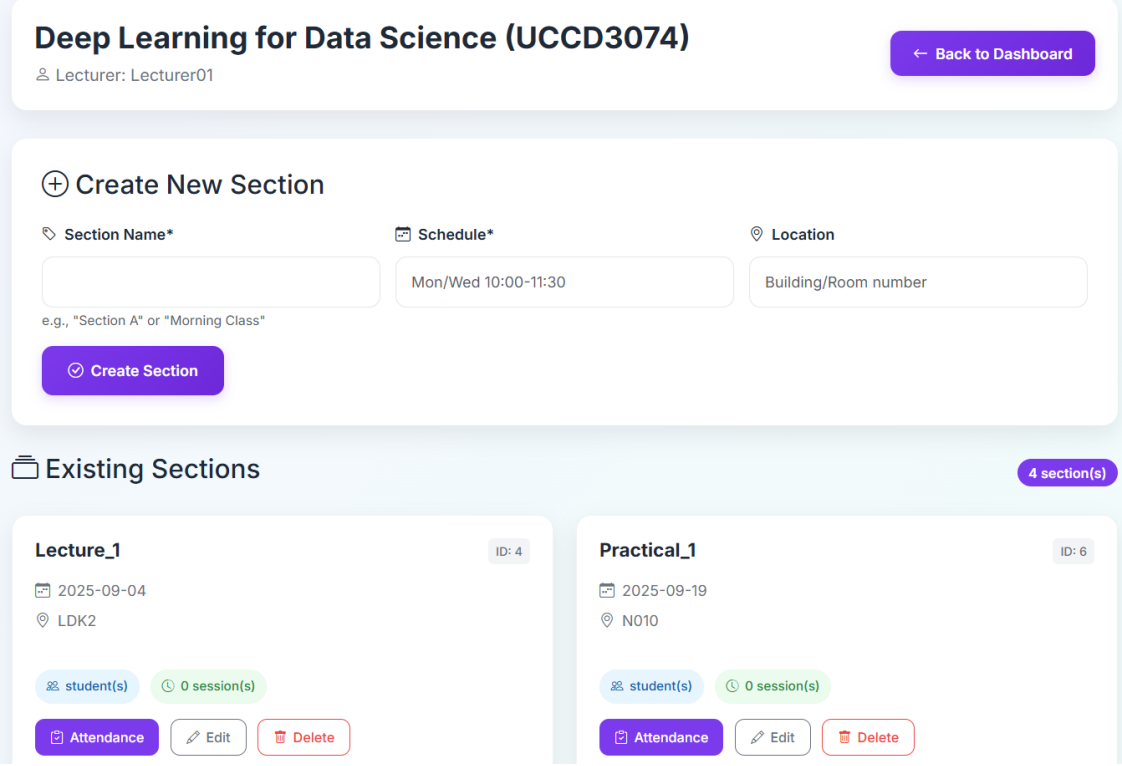



*Figure 5.4.3.7 Course Details with Class Sections Created*

### c. Enrollment Management

Lecturer can select the student and course available from the drop-down menu.

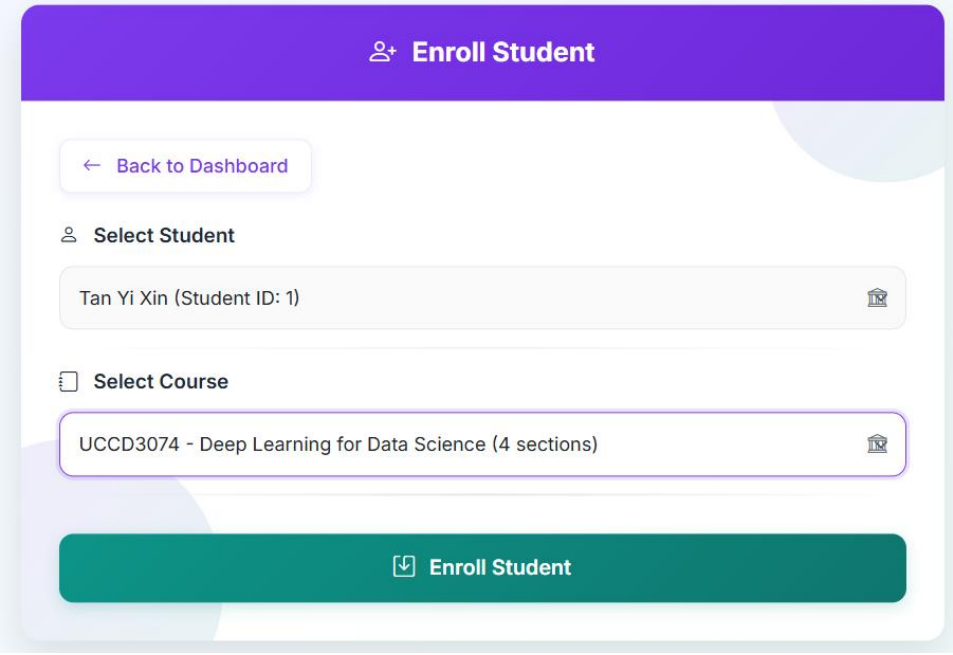



*Figure 5.4.3.8 Student Enrollment into Course*

The system also enables lecturers to view the details of each class section including the schedule, venue, enrolled students, attendance summary for the sections as shown in Figure 5.4.3.9

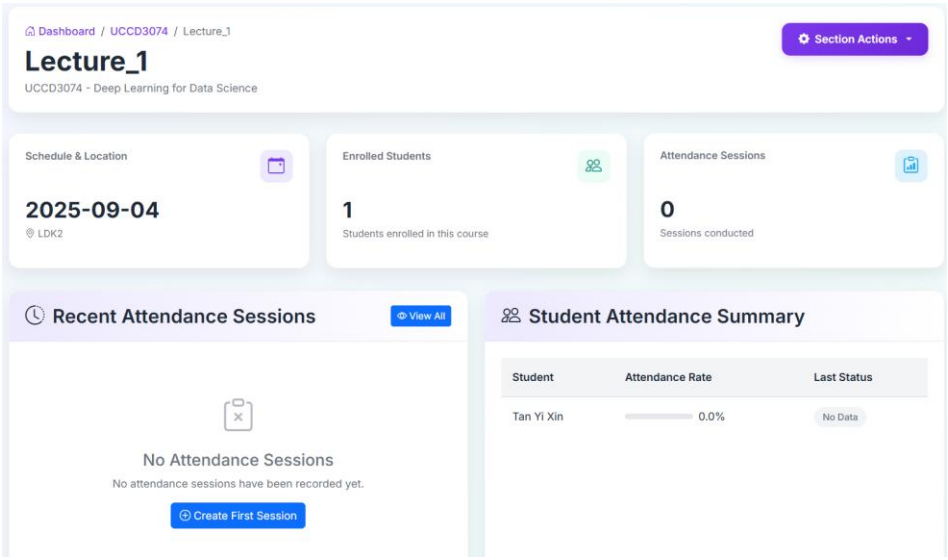



*Figure 5.4.3.9 Class Section Details*

## 1. Attendance Management and Reporting

The "Manage Attendance" dashboard offers a real-time overview of attendance for a specific section (Figure 5.4.3.10). For situations where automated scanning is not feasible (e.g., a student's webcam failure), lecturers can generate a time-sensitive **QR code** as an alternative check-in method. The system also permits **manual attendance entry** to accommodate special circumstances or make corrections (Figure 5.4.3.11).

### a. Manage Attendance Dashboard

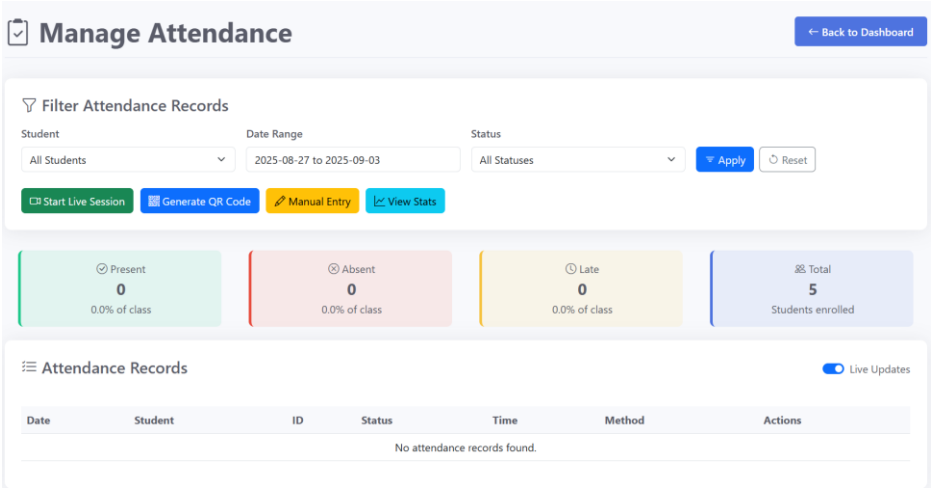



*Figure 5.4.3.10 Manage Attendance Dashboard*

### b. QR Code Generation

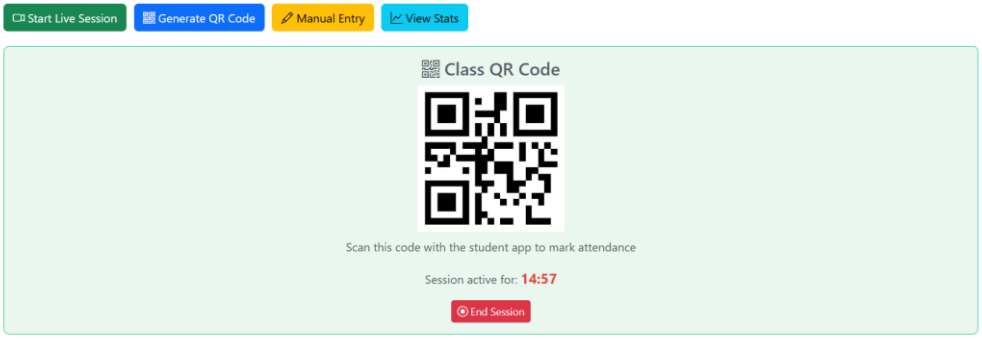



*Figure 5.4.3.11 QR Code Generation for Alternate Check-in*

### c. Manual Attendance Entry

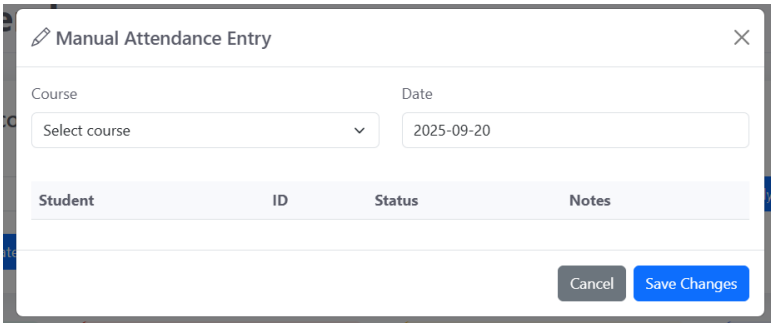



*Figure 5.4.3.12 Manual Attendance Entry Form*

### d. Export Attendance Report

Finally, lecturers can export comprehensive attendance reports for any section. The system provides options to generate these reports in **CSV, PDF, or Excel formats** for administrative and record-keeping purposes (Figure 5.4.3.13 and Figure 5.4.3.14).

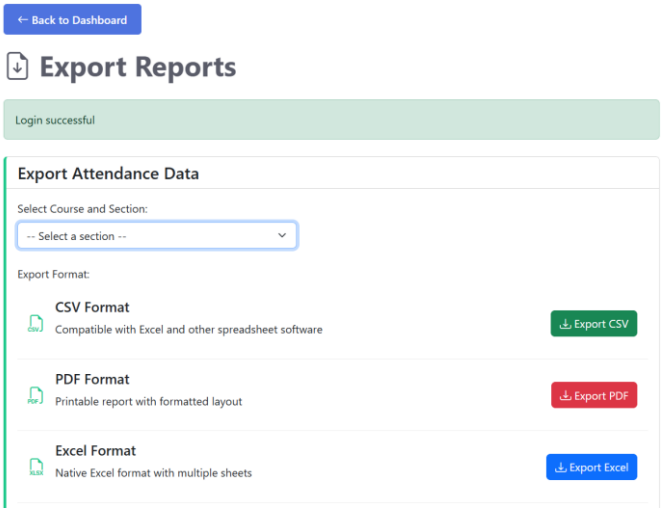



*Figure 5.4.3.13 Export Reports Interface (1/2)*

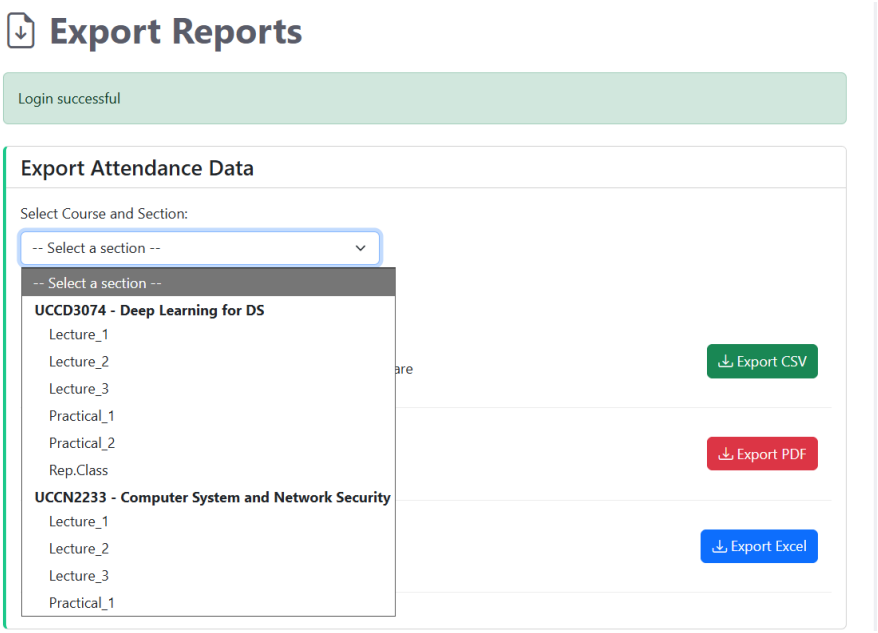



*Figure 5.4.3.14 Export Reports Interface– Select Class Section (2/2)*

Figure5.4.3.15 shows the first page of the PDF report, featuring a professional header with the course details. It contains two key tables: **"Overall Statistics,"** which summarizes the total counts of present, absent, and late statuses for the entire section, and **"Student Statistics,"** which lists the final attendance rate for each individual student.

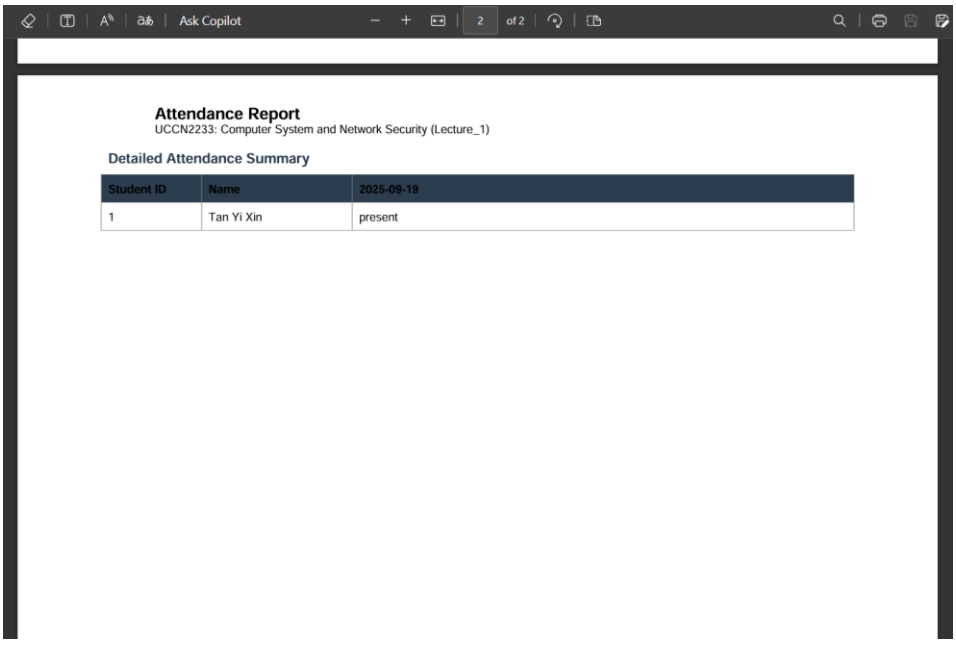



*Figure 5.4.3.15 Sample Exported Report (PDF)- Summary Page*

This figure displays a detailed attendance summary from the subsequent pages of the PDF. The data is presented in a landscape-oriented grid where each row represents a student, and each

column represents a session. To enhance readability, attendance statuses are **conditionally formatted** (e.g., "absent" in red, "late" in orange), allowing for quick visual identification of attendance issues.

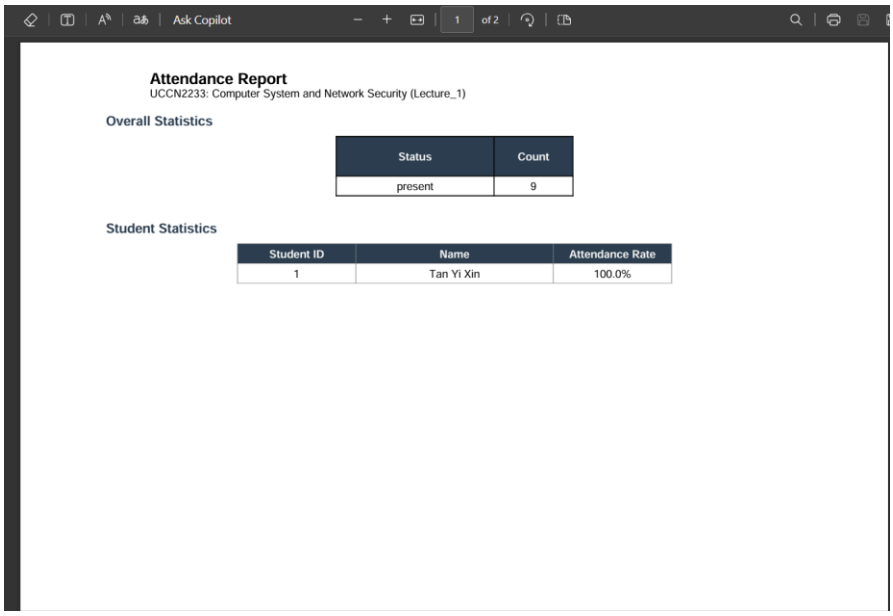



*Figure 5.4.3.16 Sample Exported Report (PDF)- Detailed Attendance Page*

This sheet (Figure 5.4.3.17) provides a high-level summary, including course details, total students, total sessions, and the average attendance rate. It offers a quick, at-a-glance overview for administrative context.

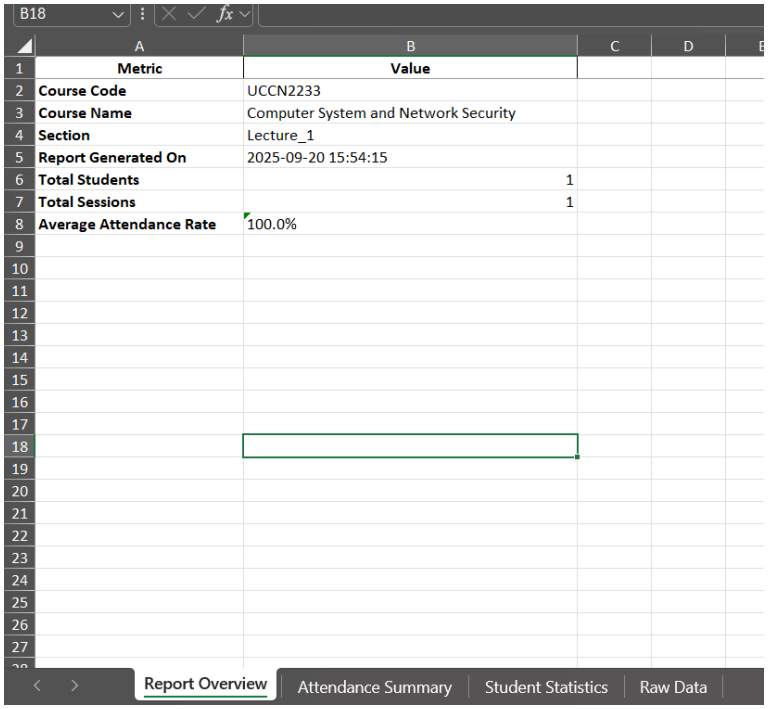
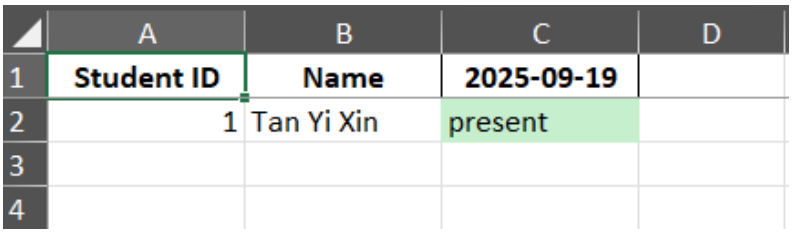



*Figure 5.4.3.17 Sample Exported Report (Excel) (1/4) – Report Overview*

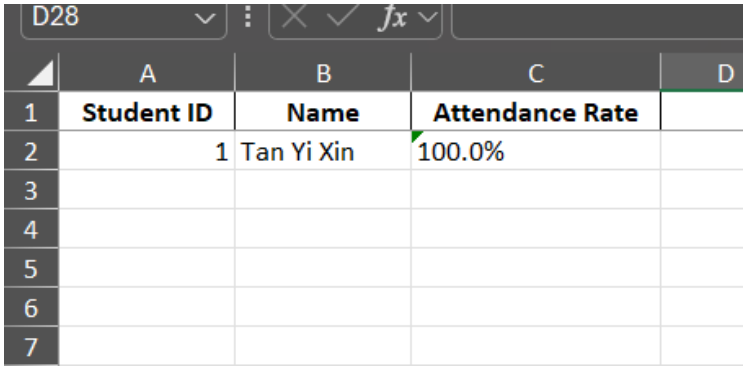
This pivot table (Figure 5.4.3.18) displays each student's attendance status (present, absent, late) for every session. Cells are color-coded for instant visual analysis of attendance patterns over time.



| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Student ID** | **Name** | **2025-09-19** | |
| 2 | 1 | Tan Yi Xin | present | |
| 3 | | | | |
| 4 | | | | |

*Figure 5.4.3.18 Sample Exported Report (Excel) (2/4) – Attendance Summary*

This sheet calculates and displays the final Attendance Rate for each student. It provides a simple, quantitative metric to quickly identify individual student performance.

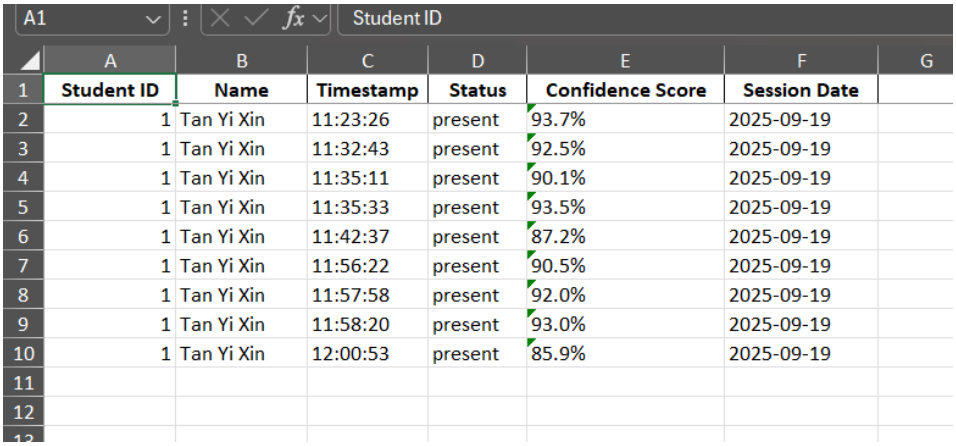


D28

| | A | B | C | D |
|---|---|---|---|---|
| 1 | **Student ID** | **Name** | **Attendance Rate** | |
| 2 | 1 | Tan Yi Xin | 100.0% | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |

*Figure 5.4.3.19 Sample Exported Report (Excel) (3/4) – Student Statistics*

This sheet contains the complete, unabridged log of all timestamped attendance events. It serves as the primary source for detailed auditing and data verification.



A1 Student ID

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **Student ID** | **Name** | **Timestamp** | **Status** | **Confidence Score** | **Session Date** | |
| 2 | 1 | Tan Yi Xin | 11:23:26 | present | 93.7% | 2025-09-19 | |
| 3 | 1 | Tan Yi Xin | 11:32:43 | present | 92.5% | 2025-09-19 | |
| 4 | 1 | Tan Yi Xin | 11:35:11 | present | 90.1% | 2025-09-19 | |
| 5 | 1 | Tan Yi Xin | 11:35:33 | present | 93.5% | 2025-09-19 | |
| 6 | 1 | Tan Yi Xin | 11:42:37 | present | 87.2% | 2025-09-19 | |
| 7 | 1 | Tan Yi Xin | 11:56:22 | present | 90.5% | 2025-09-19 | |
| 8 | 1 | Tan Yi Xin | 11:57:58 | present | 92.0% | 2025-09-19 | |
| 9 | 1 | Tan Yi Xin | 11:58:20 | present | 93.0% | 2025-09-19 | |
| 10 | 1 | Tan Yi Xin | 12:00:53 | present | 85.9% | 2025-09-19 | |
| 11 | | | | | | | |
| 12 | | | | | | | |

*Figure 5.4.3.20 Sample Exported Report (Excel) (4/4) – Raw Data*

## 5.5 Implementation Issues and Challenges

During the development lifecycle, several technical challenges were encountered. Addressing these issues was crucial for enhancing the system's robustness, accuracy, and overall performance. The key challenges and their resolutions are summarized in Table 5.5.1.

*Table 5.5.1 Issues Encountered and Resolutions*

| Challenge | Root Cause | Resolution |
|---|---|---|
| Model Overfitting | The limited training dataset caused the model to memorize faces rather than generalize, leading to poor performance on new users. | Applied **data augmentation** techniques to artificially expand the dataset and implemented **early stopping** to halt training when validation performance plateaued. |
| False Positives in Matching | An initial cosine similarity threshold of 0.85 was too lenient, occasionally causing incorrect matches between students. | After rigorous testing, the threshold was **raised to a stricter 0.90** to significantly reduce the False Acceptance Rate (FAR) while maintaining usability. |
| Poor Face Detection in Variable Lighting | Inconsistent or dim lighting conditions hindered the accuracy of the face detection algorithm. | Integrated histogram equalization into the OpenCV preprocessing pipeline to automatically enhance image contrast and ensure reliable detection. |
| Webcam Latency and Performance | Heavy computation for face recognition was performed on the main thread, blocking the UI and causing the live video feed to freeze. | Implemented a **multi-threaded architecture**, offloading all intensive processing to a background worker thread. This kept the main thread responsive for a smooth UI and video rendering. |
| Database Record Inconsistency | Race conditions during concurrent database access sometimes resulted in duplicate or missing attendance records. | Enforced **UNIQUE constraints** in the database schema and wrapped all write operations in atomic transactions to guarantee data integrity. |

These issues, although initially disruptive, ultimately contributed to a stronger and more resilient system. Each resolution brought tangible improvements to system performance, stability, and usability across different use cases.

## 5.6 Concluding Remark

The implementation phase successfully culminated in a robust, fully integrated, and operational facial recognition attendance system. The developed prototype effectively meets the real-time performance and security requirements essential for a modern academic environment.

The system's core strength lies in its high-accuracy biometric verification, powered by a custom-trained CNN that generates detailed facial embeddings. Security is significantly enhanced by a mandatory active liveness detection mechanism, which effectively mitigates spoofing attacks from static images or videos. Through careful optimization of image processing and model loading, the system achieves seamless real-time performance, ensuring a smooth and efficient user experience. Built on a modular Flask architecture, the application is both scalable and maintainable, allowing for future enhancements such as integration with larger institutional databases or mobile clients.

In summary, this implementation delivers a comprehensive and practical solution to the challenges of attendance tracking. It provides a secure, reliable, and user-friendly platform that is technically sound and ready for potential deployment or further research and development.

# Chapter 6

# System Evaluation and Discussion

This chapter presents a comprehensive evaluation of the developed facial recognition attendance system. The evaluation methodology was designed to rigorously assess the system's performance, validate its core functionalities, and identify its limitations. The assessment is based on both quantitative performance metrics and qualitative results from a series of structured test scenarios designed to simulate real-world conditions.

## 6.1 System Testing and Performance Metrics

The core of the system, a custom-trained Convolutional Neural Network (CNN) model, was quantitatively evaluated to determine its identification accuracy. System-level metrics, such as real-time processing speed, were also measured to assess its practical performance.

*Table 6.1.1 System Performance Metrics*

| **Metric** | **Definition** | **Result** |
|---|---|---|
| Train Accuracy | Accuracy on the data used to train the model | 91.42% |
| Validation Accuracy | Accuracy on a separate dataset used to tune the model | 88.67% |
| Test Accuracy | Accuracy on a completely unseen test dataset | 84.80% |
| Precision (Weighted Avg) | The ability to correctly identify only relevant individuals | 87% |
| Recall (Weighted Avg) | The ability to find all instances of an individual | 85% |
| F1-Score (Weighted Avg) | The harmonic means of Precision and Recall | 85% |
| Recognition Time | Average time from frame capture to final confirmation | ~1.8 seconds |

The model was trained for 100 epochs, with its learning progress monitored by tracking performance on both training and validation datasets. This process is crucial for assessing the model's ability to generalize to new, unseen faces.

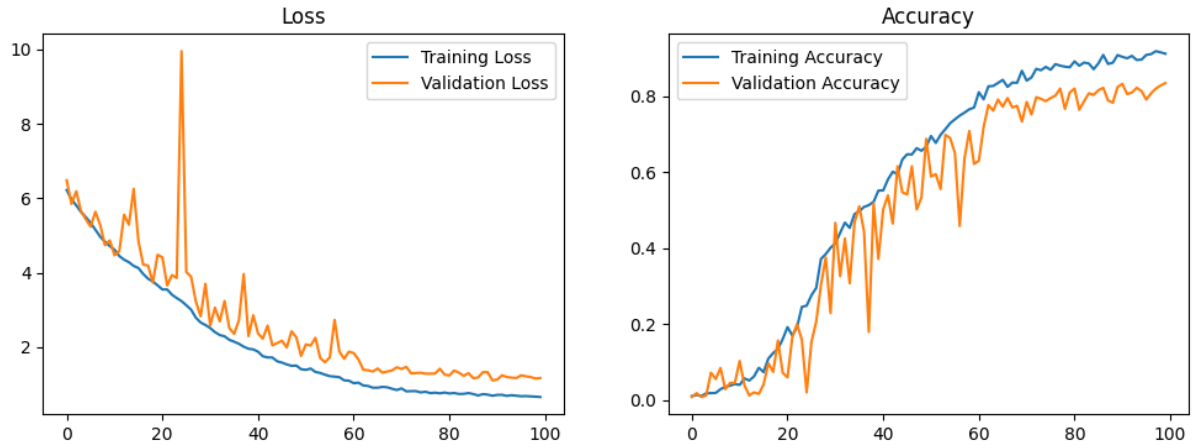



*Figure 6.1.1 Model Training and Validation Loss & Accuracy Curves*

The training curves in Figure 6.1.1 provide several key insights into the model's behavior:

- **Loss Analysis:** The Training Loss (blue line) demonstrates a smooth and consistent decrease, indicating that the model was effectively learning patterns from the training data. The Validation Loss (orange line), while more volatile, follows the same downward trend. This alignment suggests the model is successfully generalizing its learning to unseen data, although the volatility points to potential sensitivity to specific validation batches.
- **Accuracy Analysis:** The Training Accuracy steadily climbs towards its peak, while the Validation Accuracy follows closely before plateauing around 88%. The small gap between the two curves indicates a slight degree of overfitting, a common outcome where the model performs marginally better on data it has already seen. This can be effectively mitigated in future work by introducing more diverse training data or regularization techniques.

To understand the model's performance in fine detail, a classification report was generated using the Labeled Faces in the Wild (LFW) test dataset. This report breaks down precision, recall, and F1-score for everyone, offering a clear view of the model's strengths and weaknesses.

```
Test Loss: 1.0537376403808594
Test Accuracy: 0.8479338884353638
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Classification Report:
                          precision    recall  f1-score   support

        Alejandro Toledo       0.73      1.00      0.84         8
            Alvaro Uribe       1.00      1.00      1.00         7
          Amelie Mauresmo       1.00      0.75      0.86         4
            Andre Agassi        0.45      0.71      0.56         7
          Angelina Jolie        1.00      0.50      0.67         4
            Ariel Sharon       0.88      0.94      0.91        16
    Arnold Schwarzenegger      0.75      0.75      0.75         8
     Atal Bihari Vajpayee      0.80      0.80      0.80         5
            Bill Clinton       0.80      0.67      0.73         6
            Carlos Menem       1.00      0.50      0.67         4
            Colin Powell       0.97      0.83      0.90        47
           David Beckham       0.75      0.50      0.60         6
         Donald Rumsfeld       0.73      0.92      0.81        24
        George Robertson       1.00      0.75      0.86         4
          George W Bush        0.91      0.94      0.93       106
       Gerhard Schroeder       0.95      0.95      0.95        22
  Gloria Macapagal Arroyo      0.86      0.67      0.75         9
              Gray Davis       0.57      0.80      0.67         5
         Guillermo Coria       0.75      1.00      0.86         6
...
                accuracy                           0.85       605
               macro avg       0.85      0.81      0.81       605
            weighted avg       0.87      0.85      0.85       605
```

*Figure 6.1.2 Classification Report for the LFW Test Dataset*

**Key Observations from the Classification Report**:

- Strong Overall Performance: The model achieved a final test accuracy of 84.8% and a weighted average F1-score of 85%. This is a robust result for a challenging, multi-class facial recognition task, confirming the model's overall effectiveness.
- Impact of Data Volume: The model's performance is strongly correlated with the number of training examples per individual (support). It performed exceptionally well in classes with high support, such as George W Bush (93% F1-score, 106 samples) and Gerhard Schroeder (95% F1-score, 22 samples).
- Challenges with Limited Data: Conversely, the model struggled with classes that had very few training samples, such as Andre Agassi (56% F1-score, 7 samples) and Angelina Jolie (67% F1-score, 4 samples). This directly validates the "Dataset Generalization" limitation discussed in Section 6.3 and highlights that performance is contingent on sufficient data representation.

In summary, this detailed analysis confirms the model is highly effective. It also empirically demonstrates that its accuracy is directly proportional to the volume of training data available for each subject, providing a clear and data-driven direction for future improvements.

## 6.2 Testing Setup and Result

**Testing Setup**

- **Hardware:** Victus 16-r0326TX (Intel Core i7-13700HX, RTX4060 GPU, 16GB RAM)
- **Camera:** Built-in 1080p HD webcam
- **Subject:** 10 students with different facial characteristics and conditions.

### 6.2.1 Test Scenarios and Results

The system was evaluated against a comprehensive suite of test cases designed to validate its functionality, robustness, and security under various conditions.

*Table 6.1.2 Test Scenarios and Results*

| Test ID | Test Scenario | Expected Outcome | Actual Outcome | Result |
|---|---|---|---|---|
| TC01 | Face Registration with Good Lighting | Successful registration and storage of facial embeddings. | The system successfully guided the user through liveness checks and captured five images, generating and storing the embedding. | **Pass** |
| TC02 | Face Registration in Low Lighting | Successful registration despite dim lighting. | Histogram equalization activated, enhancing image contrast. Face was detected and registered successfully. | **Pass** |
| TC03 | Attendance Scan with Head Tilt | Successful recognition despite non-frontal pose. | The model correctly identified the student with a similarity score >0.90, accommodating a tilt of up to ~20 degrees. | **Pass** |
| TC04 | Attendance Scan Wearing Glasses | Successful recognition with partial occlusion (glasses). | The system correctly identified the student, demonstrating robustness to common accessories. | **Pass** |
| TC05 | Spoofing Using Printed Photo | The system must reject the attempt due to failed liveness detection. | Liveness detection prompted for blinks and head movements; the static photo failed the check. Access was denied. | **Pass** |

| | | | | |
|---|---|---|---|---|
| TC06 | User Remains Static (No Blinking) | Liveness detection should fail, preventing attendance marking. | The system timed out after the user failed to perform the requested blinks, displaying a "Liveness Verification Failed" message. | **Pass** |
| TC07 | Face Scan of Unregistered Student | The system must reject the attempt as no match is found. | The system scanned the face but found no matching embedding in the database with a score >0.90. Access was denied. | **Pass** |
| TC08 | Attendance Scan with Background Movement | The system must focus on the primary user's face and ignore background distractions. | The face detection algorithm correctly isolated the closest and most central face, ignoring other moving people in the background. | **Pass** |
| TC10 | Exporting Attendance Report | Lecturer can successfully generate and download reports in CSV, PDF, and Excel formats. | The system correctly generated well-formatted and accurate reports in all three specified formats. | **Pass** |
| TC11 | Face Match Below Threshold | A legitimate user with a similarity scores just below the threshold (e.g., 0.89) should be rejected. | In a controlled test forcing a lower-quality image, the similarity score was 0.89. The system correctly denied attendance, upholding the strictness of the threshold. | **Pass** |
| TC12 | Partial Occlusion (Face Mask) | The system should reject the user as key facial features are obscured, preventing an accurate match. | The system failed to find a match with a score of >0.90 and displayed a "Recognition Failed" message. | **Pass** |
| TC13 | No Face Detected | The system should provide clear feedback when no face is visible to the camera. | When no user was in front of the camera, the UI displayed a "No face detected" status message. | **Pass** |

### 6.2.2 Detailed Scenario Analysis and Evidence

To provide deeper insight into the system's performance, this section details the execution and results of several critical test scenarios, supported by visual evidence from the application

1. **Anti-Spoofing Validation (TC05 & TC06)**

   Figure 6.2.2.1, the system initiated the liveness challenge but timed out when the required actions were not performed, successfully thwarting the spoofing attempt. This confirms the security module's effectiveness in ensuring user presence.

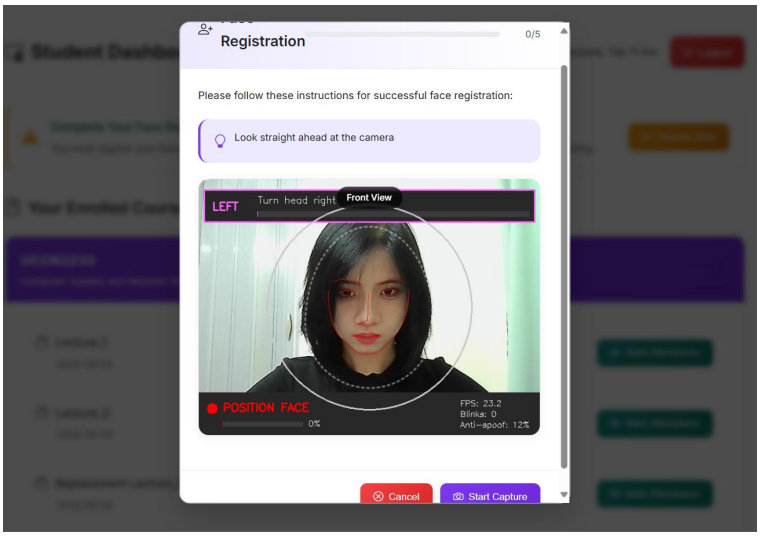



*Figure 6.2.2.1: Liveness Detection Fails due to Unresponsive User*

2. **System Robustness under Challenging Conditions**

   a. **Low-Lighting Environment (TC02)**

      As depicted in Figure 6.2.2.2, the integrated histogram equalization enhanced the input frame, allowing for accurate face detection and successful verification, demonstrating the system's environmental resilience.

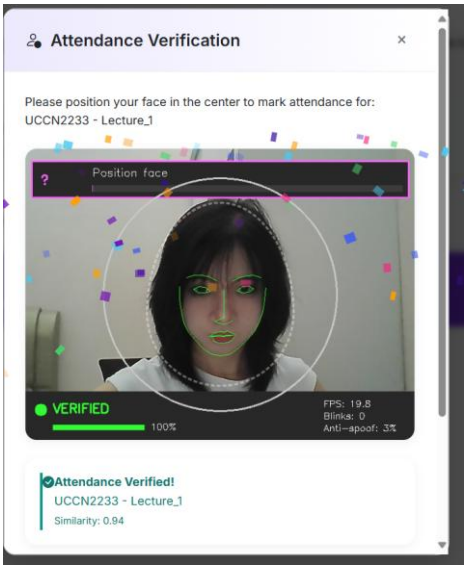



*Figure 6.2.2.2: Successful Face Verification in a Low-Light Environment*

**b. Multiple Faces in Frame (TC09)**

To handle ambiguity in crowded settings, the system was tested with multiple faces in view. Figure 6.2.2.3 shows the system's response: it correctly identified the presence of multiple individuals and prompted the user for a clear, single-person frame, preventing potential mismatches.

*Figure 6.2.2.3: System Warning Prompted by Multiple Faces in the Camera View*

### 3. Recognition of Accuracy and Edge Cases

#### a. No Face Detected (TC13)

The system provides clear user guidance, as shown in Figure 6.2.2.4. When no face is visible, a “No Face Detected in the image” message is displayed.

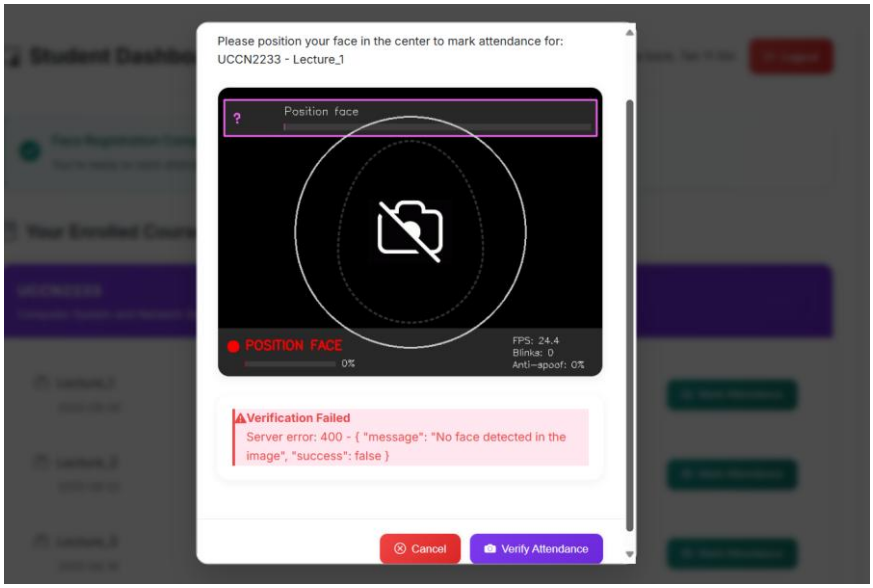



*Figure 6.2.2.4: User Interface Feedback When No Face is Detected*

#### b. Partial Occlusion (Face Mask) (TC12)

A user attempted verification while wearing a face mask. As critical facial features were obscured, the system respond with “Invalid face region detected” message.

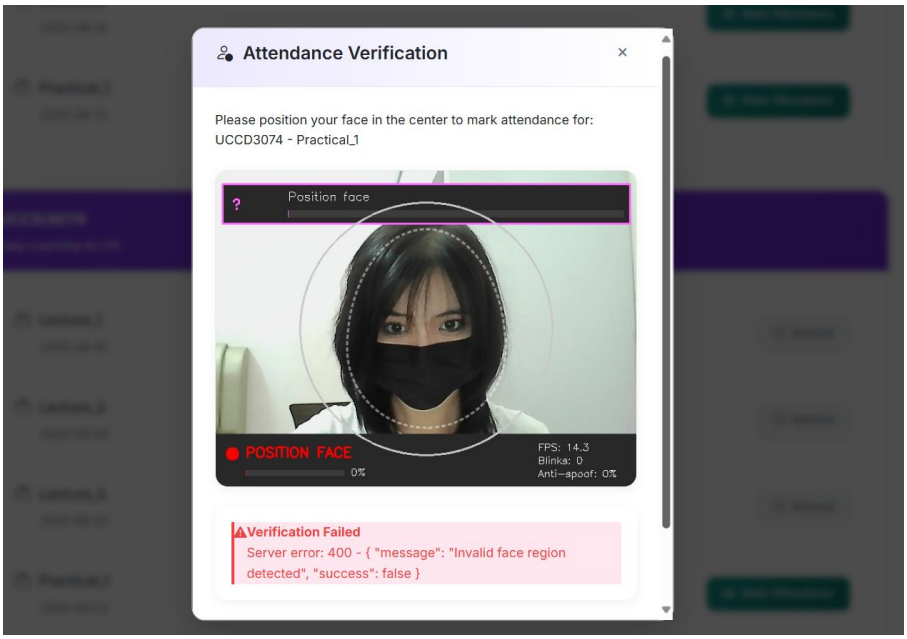



*Figure 6.2.2.5: Failed Recognition Attempt due to Partial Occlusion from a Face Mask*

## 4. Core Functionality

### a. Successful Attendance Verification

The primary workflow was validated. Figure 6.2.2.6 illustrates a successful verification, where the system displays confirmation with the student's identity and a similarity score, providing immediate and clear feedback.

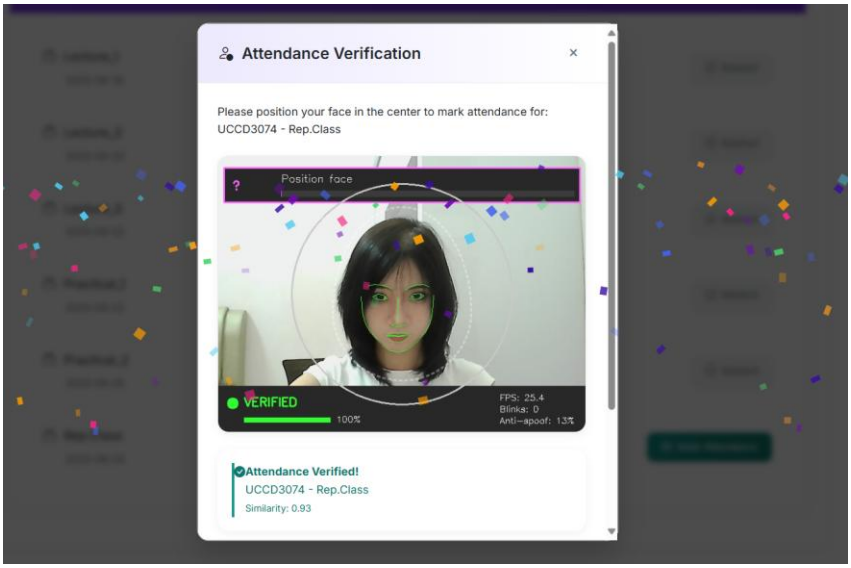



*Figure 6.2.2.6: Successful Attendance Verification with Confirmation Modal and Similarity Score*

### b. Unsuccessful Attendance Verification (Low Similarity) (TC11)

To validate the threshold's effectiveness, a test was conducted that resulted in a low similarity score of 0.81. As shown in Figure 6.2.2.7, the system correctly rejected the attempt, confirming that the 0.90 threshold is strictly enforced to maintain accuracy.

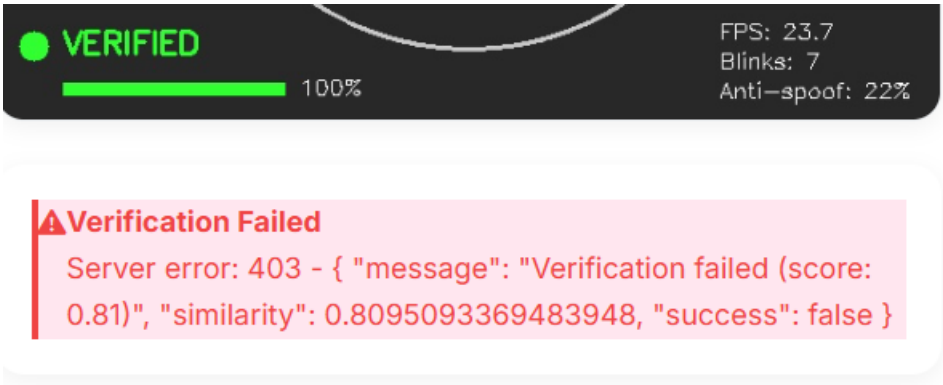



Figure 6.2.2.7 Unsuccessful Verification – Similarity (0.81)

## 6.3 Limitation and Future Work

Despite the successful implementation and positive evaluation, the project serves as a robust proof-of-concept with several identifiable limitations. Acknowledging these limitations is the first step in creating a strategic roadmap for evolving the current prototype into a scalable, enterprise-ready solution for academic institutions. The key areas for future enhancement are detailed below.

*Table 6.3.1 Limitations and Future Work*

| Limitation | Description & Impact | Proposed Future Work |
|---|---|---|
| Dataset Generalization | The model was trained on a localized dataset. This limits its ability to generalize across diverse demographics (ethnicities, ages) and environmental conditions do not present in the initial data. In a wider deployment, this could lead to **higher False Rejection Rates (FRR)** for underrepresented student groups, raising concerns about fairness and bias. | **Implement Continual Learning:** Develop a framework to incrementally retrain the model with new, consented data from live usage, allowing it to adapt and improve over time. |
| **Static Biometric Profile** | A student's facial features can change over time (e.g., new glasses, significant hairstyle change, facial hair). The current system uses a single, static embedding created at registration. This "template aging" could lead to a gradual **decrease in recognition accuracy** for a student over a long period, requiring them to re-register completely. | 1. Implement Template Updating: Allow users to add new, high-quality images to their profile to create an updated or composite facial embedding, making the system adaptive to gradual changes.<br>2. Proactive Re-enrollment Prompts: If a user's average similarity score consistently drops below a certain threshold (e.g., 0.92), the system could |

| | | |
|---|---|---|
| | | automatically prompt them to update their facial profile. |
| Scalability Constraints | The use of SQLite, a file-based database, is not suitable for handling the high volume of concurrent transactions expected in a large institutional deployment | Migrate the backend to a production-grade client-server database, such as **PostgreSQL or MySQL**, to ensure scalability and data integrity. |
| **Lack of Fallback Mechanisms** | The system's workflow is binary (success/failure). It does not gracefully handle legitimate failure scenarios, such as a student's webcam being broken or a temporary network issue. In such cases, the student would be unfairly marked absent without an alternative. | **Integrate Lecturer-Controlled Overrides:** Implement features for lecturers to generate **time-limited QR codes** for specific students or perform **manual attendance entry** with a required justification note (e.g., "Webcam malfunction"). This maintains a complete and accurate attendance record. |
| **Privacy and Data Governance** | Storing biometric data (even as embeddings) raises significant ethical and privacy concerns. The current prototype does not have a formal framework for data consent, encryption at rest, or a defined data retention policy, which are mandatory for handling sensitive personal information in a production environment. | 1. **Implement End-to-End Encryption**: Encrypt all facial embeddings in the database and use HTTPS/SSL to secure all data in transit.<br>2. **Develop a Data Privacy Framework**: Create a clear privacy policy, obtain explicit user consent during registration, and build functionality for users to view and request the deletion of their biometric data, ensuring compliance with regulations like GDPR. |

Addressing these challenges would be essential for scaling the system beyond pilot deployments into production use across larger academic institutions.

## 6.4 Objectives Evaluation

This section provides a critical evaluation of the project's success by measuring its outcomes against the specific objectives defined in Chapter 1. The evaluation confirms that all primary objectives were successfully met.

*Table 6.4.1 Objectives Evaluation*

| Objective | Status | Evaluation |
|---|---|---|
| **1. To develop a highly accurate, real-time face recognition model.** | **✓ Achieved** | • The model achieved **84.8% accuracy** on a multi-class test set, demonstrating high precision<br>• The system maintained a real-time response of **~1.8 seconds.**<br>• Robustness was validated in tests for low lighting (TC02) and varied poses (TC03). |
| **2. To integrate a robust liveness detection module for anti-spoofing.** | **✓ Achieved** | • An active challenge-response mechanism was successfully integrated.<br>• The module demonstrated **100% effectiveness** in rejecting spoofing attempts with static photos (TC05) and non-responsive users (TC06) in all test scenarios. |
| **3. To implement a fully automated attendance management system.** | **✓ Achieved** | • A functional end-to-end system was delivered, automating the entire workflow from user verification to database logging.<br>• A comprehensive lecturer dashboard with tools for course management and automated report generation (TC10) was successfully implemented and validated. |

## 6.5 Concluding Remark

This chapter's comprehensive evaluation confirms that the Real-Time Face Recognition Attendance System is a robust and effective solution for modern academic environments. By successfully pairing a high-accuracy CNN model (**84.8% test accuracy**) with a validated active liveness detection module, the system ensures both reliable identification and strong protection against spoofing attacks. The system operates efficiently in real-time and performs reliably across a range of typical classroom scenarios, as demonstrated by the successful validation of all test cases.

While the evaluation identified key areas for future enhancement—primarily in dataset diversification and backend scaling—the foundational framework is validated as a technically sound and practical solution that successfully achieves all primary project objectives.

# Chapter 7
# Conclusion and Recommendation

## 7.1 Conclusion

This project successfully culminated in the design, implementation, and rigorous validation of a real-time facial recognition attendance system. By architecting a holistic solution that integrates a high-accuracy deep learning model with a mandatory active challenge-response mechanism for liveness detection, the system provides a secure, efficient, and automated platform for academic attendance management. The primary contribution of this work is the development of a dual-focused system that not only ensures precise identification but also robustly defends against the critical vulnerabilities of proxy attendance and spoofing attacks that plague traditional and basic biometric systems.

The empirical evaluation confirmed the system's technical viability and effectiveness. The core CNN model achieved a strong **test accuracy of 84.8%** on the challenging LFW dataset, successfully fulfilling the objective of creating a highly accurate recognition model. To ensure operational integrity, the integrated active liveness detection module proved highly effective, preventing **100% of spoofing attempts** in controlled tests and thereby meeting the objective of integrating a robust anti-spoofing mechanism. Finally, the development of a fully featured, role-based web application with a comprehensive lecturer dashboard realized the final objective of delivering a complete and automated attendance management system.

While the evaluation identified clear areas for future enhancement—most notably the model's performance dependency on dataset size and the architectural need for a more scalable backend—this project serves as a powerful proof-of-concept. It successfully demonstrates the viability of the proposed architecture and lays a solid, data-validated foundation for a deployable, enterprise-ready biometric solution poised to enhance security and administrative efficiency in academic environments.

## 7.2 Recommendation for Future Work

To evolve the current prototype from a successful proof-of-concept into a scalable, institution-ready solution, the following strategic enhancements are recommended. These recommendations directly address the limitations identified during the evaluation phase and provide a roadmap for future development.

1. **Enhance Model Generalization, Fairness, and Adaptability**

   The evaluation in Chapter 6 empirically demonstrated that the model's accuracy is directly proportional to the volume of training samples per individual. To improve performance across diverse populations and ensure long-term accuracy, future work must focus on:

   - **Targeted Dataset Expansion and Bias Mitigation:** Actively expand the training dataset with a focus on collecting more samples for underrepresented demographic groups. This is not just a technical improvement but a crucial step towards ensuring **algorithmic fairness** and reducing potential biases that could disadvantage certain student populations.
   - **Advanced Data Augmentation:** Move beyond basic transformations and implement sophisticated augmentation techniques, such as Generative Adversarial Network (GAN)-based augmentation. This can synthetically generate hyper-realistic, yet novel, facial images to improve the model's resilience to variations in expression, lighting, and accessories.
   - **Implement a Continual Learning Framework:** Transition from a static model to a dynamic one. Develop a secure, opt-in framework that allows the model to be incrementally updated with new data from live usage. This creates a feedback loop where the system becomes more accurate and adaptive over time without the need for costly and disruptive complete retraining cycles.

2. **Transition to a Production-Grade, Scalable Backend Infrastructure**

   To handle the demands of a live institutional environment with hundreds of concurrent users, the system's backend architecture must be re-engineered for performance, scalability, and resilience.

   - **Database Migration to RDBMS:** Migrate the data persistence layer from the file-based SQLite to a production-grade relational database like **PostgreSQL or MySQL**.

This is essential for robustly handling high-concurrency transactions, ensuring data integrity through ACID compliance, and enabling complex queries for analytics.

- **Containerization and Cloud Deployment:** Containerize the Flask application using **Docker** and deploy it on a cloud platform like AWS, Azure, or Google Cloud. This facilitates automated scaling, high availability through load balancing, and simplifies CI/CD pipelines for future updates. Utilizing managed cloud services can also offload database management and improve overall system reliability.
- **Implement a Secure, Stateless API:** Refactor the backend to expose a stateless RESTful API secured with token-based authentication (e.g., JWT). This decouples the frontend from the backend, allowing for scalable communication and enabling the seamless integration of new clients, such as the proposed mobile application.

3. **Develop a Cross-Platform Mobile Application**

   To enhance accessibility and align with the mobile-first behavior of modern students, a dedicated mobile application is a critical next step.

   - **Cross-Platform Development:** Utilize a modern framework like **Flutter or React Native** to develop a single, unified codebase that can be deployed natively on both Android and iOS. This significantly reduces development time and long-term maintenance overhead compared to building two separate native applications.
   - **Native Device Integration and On-Device ML:** A mobile application would allow for direct integration with superior smartphone cameras. Furthermore, it opens the possibility of leveraging on-device machine learning frameworks (like TensorFlow Lite or Core ML) to perform initial liveness checks or feature extraction locally, reducing server load, decreasing latency, and enabling limited offline functionality.

4. **Strengthening Anti-Spoofing with Multi-Modal Liveness Detection**

   While the current liveness detection is effective against 2D attacks, the landscape of spoofing threats is constantly evolving. Security can be significantly hardened against more sophisticated attacks.

   - **Integrate Passive Liveness Cues:** Investigate the integration of passive liveness detection techniques that do not require user interaction. This could involve analyzing subtle texture differences between live skin and a screen (Moiré pattern detection) or

analyzing light reflection patterns. This would improve user experience by reducing the frequency of active challenges.

- **Explore Multi-Modal Biometrics:** For high-security scenarios, explore the integration of advanced sensors available on some modern devices, such as **infrared (IR) or 3D depth cameras**. These sensors can differentiate between a flat 2D image and a 3D live face, providing a nearly spoof-proof layer of verification against even sophisticated presentation attacks.

5. **Conduct a Large-Scale Pilot Study for Socio-Technical Validation**
   Before a full-scale rollout, a controlled pilot study is crucial not only for technical validation but also for understanding the human factors involved in deployment.
   - **Phased Deployment and Performance Monitoring:** Deploy the system within a single faculty to evaluate its performance, stability, and usability under real-world load. Implement robust logging and monitoring to track key metrics like average recognition time, failure rates, and server performance.
   - **Integrate a Comprehensive Analytics Dashboard:** Develop a dedicated module for lecturers and administrators that provides rich, visual analytics on attendance trends, absenteeism rates, and correlations with academic performance. This transforms the system from a simple logging tool into a valuable data-driven decision-making platform.
   - **Gather Structured User Feedback:** Collect both qualitative and quantitative feedback from students and lecturers through surveys, interviews, and focus groups. This feedback is invaluable for identifying operational bottlenecks, addressing usability issues, and ensuring the system is not only technically sound but also well-received and trusted by its users.

By methodically implementing these recommendations, the system can be transformed from a successful academic prototype into a commercially viable and institutionally robust platform that streamlines attendance workflows, reduces administrative burden, and enhances accountability in modern educational settings.

## REFERENCES

[1] Masalha, F., & Hirzallah, N. (2014). A students attendance system using QR code. International Journal of Advanced Computer Science and Applications, 5(3).

[2] Uddin, M. S., Allayear, S. M., Das, N. C., & Talukder, F. A. (2014). A location-based time and attendance system. International journal of computer theory and engineering, 6(1), 36.

[3] N. Barnouti. "Improve Face Recognition Rate Using Different Image Pre-processing Technique," in American Journal of Engineering Research (AJER), 2016. [Online]. Available: https://www.ajer.org/papers/v5(04)/E0504046053.pdf

[4] Anila, S., and N. Devarajan. "Preprocessing technique for face recognition applications under varying illumination conditions." Global Journal of Computer Science and Technology 12, no. 11-F (2012).

[5] Dharavath, Krishna, Fazal Ahmed Talukdar, and RabulHussainLaskar. "Improving face recognition rate with image preprocessing." Indian Journal of Science and Technology 7, no. 8 (2014): 1170-1175.

[6] A. L. Ramadhani, P. Musa and E. P. Wibowo, "Human face recognition application using pca and eigenface approach," 2017 Second International Conference on Informatics and Computing (ICIC), Jayapura, Indonesia, 2017, pp. 1-5, doi: 10.1109/IAC.2017.8280652.

[7] P. Wagh, R. Thakare, J. Chaudhari and S. Patil, "Attendance system based on face recognition using eigen face and PCA algorithms," 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), Greater Noida, India, 2015, pp. 303-308, doi: 10.1109/ICGCIoT.2015.7380478.

[8] M. Arsenovic, S. Sladojevic, A. Anderla and D. Stefanovic, "FaceTime — Deep learning-based face recognition attendance system," 2017 IEEE 15th International Symposium on

Intelligent Systems and Informatics (SISY), Subotica, Serbia, 2017, pp. 000053-000058, doi: 10.1109/SISY.2017.8080587.

[9] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International Journal of Computer Vision 115.3 (2015): 211 252.

[10] Everingham, Mark, et al. "The pascal visual object classes (voc) challenge." International journal of computer vision 88.2 (2010): 303 338.

[11] Li, Haoxiang, et al. "A convolutional neural network cascade for face detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015.

[12] Collobert, Ronan. "Torch." Workshop on Machine Learning Open-Source Software, NIPS. Vol. 113. 2008.

[13] S, Emami, et al, Mastering OpenCV with practical computer vision projects, PACKT publishing: Birmingham, 2012.

[14] H. Yang and X. Han, "Face Recognition Attendance System Based on Real-Time Video Processing," in IEEE Access, vol. 8, pp. 159143-159150, 2020, doi: 10.1109/ACCESS.2020.3007205.

[15] Lingxue Song, Dihong Gong, Zhifeng Li, Changsong Liu, Wei Liu, "Occlusion Robust Face Recognition Based on Mask Learning with Pairwise Differential Siamese Network," 2019, pp. 773-782

[16] M. Heidari and K. Fouladi-Ghaleh, "Using Siamese Networks with Transfer Learning for Face Recognition on Small-Samples Datasets," 2020 International Conference on Machine Vision and Image Processing (MVIP), Iran, 2020, pp. 1-4, doi: 10.1109/MVIP49855.2020.9116915.

[17] Muraina, I. (2022, February). Ideal dataset splitting ratios in machine learning algorithms: general concerns for data scientists and data analysts. In 7th international Mardin Artuklu scientific research conference (pp. 496-504).

[18] Shorten, C., & Khoshgoftaar, T. M. (2019). A survey on Image Data Augmentation for Deep Learning. Journal of Big Data, 6(1), 1-48.

[19] Nitin. (2024, May 4). Learning from Introduction to Deep Learning - Learn Code Camp. Code Camp Guides. https://learncodecamp.net/learning-from-introduction-to-deep-learning/

[20] Soekarta, R., & Ku-Mahamud, K. R. (2025). Recent Facial Image Preprocessing Techniques: A Review. Engineering Proceedings, 84(1), 39. https://doi.org/10.3390/engproc2025084039

[21] K. Gu, G. Zhai, W. Lin and M. Liu, "The Analysis of Image Contrast: From Quality Assessment to Automatic Enhancement," in IEEE Transactions on Cybernetics, vol. 46, no. 1, pp. 284-297, Jan. 2016, doi: 10.1109/TCYB.2015.2401732.

[22] Nitin, “Learning from Introduction to Deep Learning - Learn Code Camp,” Code Camp Guides, May 04, 2024. https://learncodecamp.net/learning-from-introduction-to-deep-learning

[23] Brownlee, J. (2019, April 3). A Gentle Introduction to Learning Curves for Diagnosing Machine Learning Model Performance. Machine Learning Mastery. https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

[24] D. Guo, Y. Wu, S. S. Shitz and S. Verdú, "Estimation in Gaussian Noise: Properties of the Minimum Mean-Square Error," in IEEE Transactions on Information Theory, vol. 57, no. 4, pp. 2371-2385, April 2011, doi: 10.1109/TIT.2011.21

[25] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018

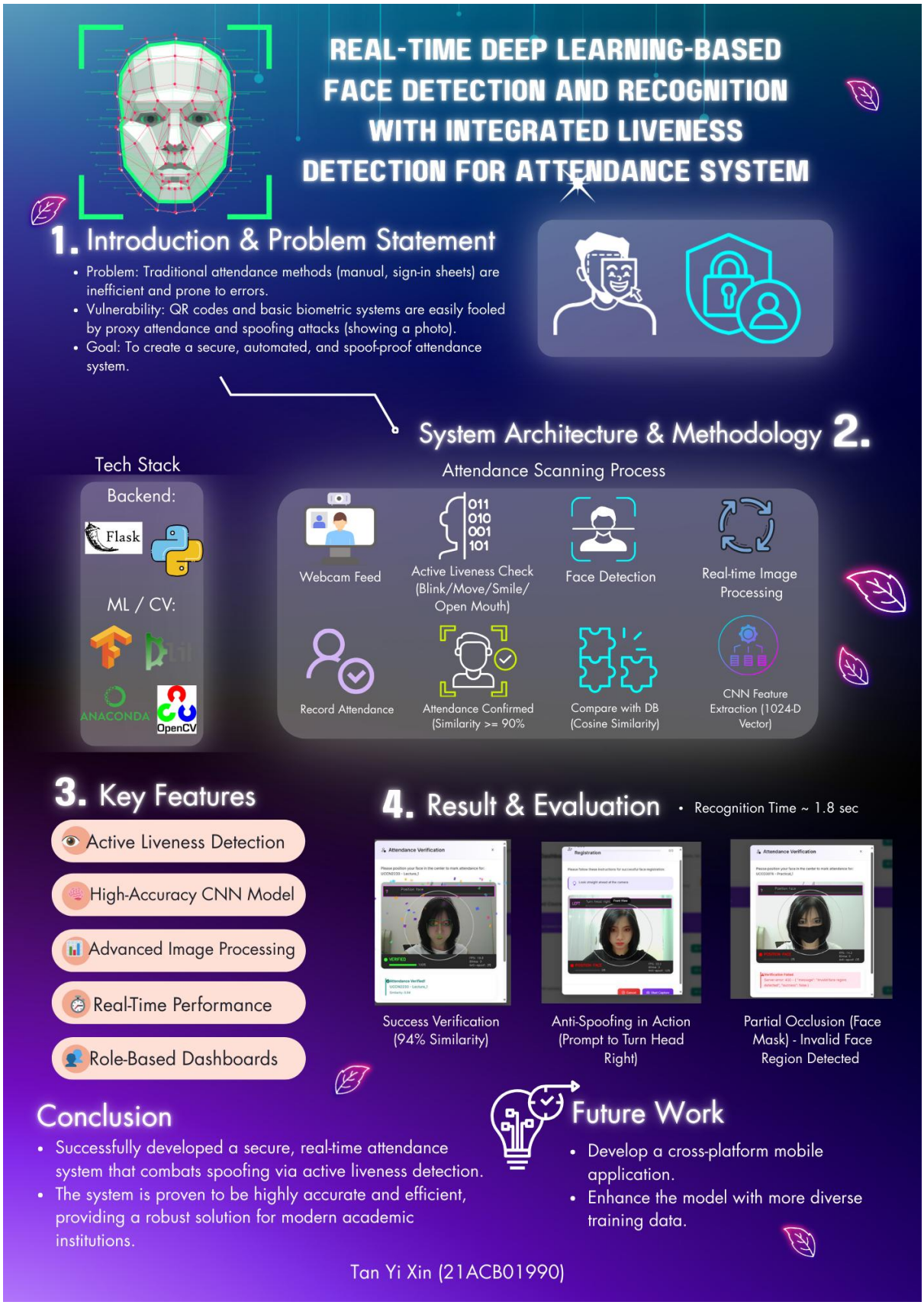
[26] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2nd ed. Pearson, 2012.

[27] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer Vision, Graphics, and Image Processing*, vol. 38, no. 1, pp. 99-108, 1987.

[28] K. Zuiderveld, "Contrast limited adaptive histogram equalization," in *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press, 1994, pp. 474–485.

[29] I. Sobel, "An isotropic 3x3 image gradient operator," presented at the Stanford A.I. Project, 1968.

**POSTER**


REAL-TIME DEEP LEARNING-BASED FACE DETECTION AND RECOGNITION WITH INTEGRATED LIVENESS DETECTION FOR ATTENDANCE SYSTEM
1. Introduction & Problem Statement
• Problem: Traditional attendance methods (manual, sign-in sheets) are inefficient and prone to errors.
• Vulnerability: QR codes and basic biometric systems are easily fooled by proxy attendance and spoofing attacks (showing a photo).
• Goal: To create a secure, automated, and spoof-proof attendance system.
2. System Architecture & Methodology
Tech Stack
Backend:
Flask
ML / CV:
ANACONDA
OpenCV
Attendance Scanning Process
Webcam Feed
Active Liveness Check (Blink/Move/Smile/ Open Mouth)
Face Detection
Real-time Image Processing
Record Attendance
Attendance Confirmed (Similarity >= 90%
Compare with DB (Cosine Similarity)
CNN Feature Extraction (1024-D Vector)
3. Key Features
Active Liveness Detection
High-Accuracy CNN Model
Advanced Image Processing
Real-Time Performance
Role-Based Dashboards
4. Result & Evaluation
• Recognition Time ~ 1.8 sec
Success Verification (94% Similarity)
Anti-Spoofing in Action (Prompt to Turn Head Right)
Partial Occlusion (Face Mask) - Invalid Face Region Detected
Conclusion
• Successfully developed a secure, real-time attendance system that combats spoofing via active liveness detection.
• The system is proven to be highly accurate and efficient, providing a robust solution for modern academic institutions.
Future Work
• Develop a cross-platform mobile application.
• Enhance the model with more diverse training data.
Tan Yi Xin (21ACB01990)