COVENTRY
UNIVERSITY

Faculty of Engineering and Computing

Department of Aerospace, Electronic and Electrical Engineering

6040CEM Individual Project Realization

Medium Range Facial Recognition for Attendance Recording

Student Name: Chuah Hui Suen

Student ID (CU): 13466486

Supervisor: Dr Solahuddin Yusuf Fadhlullah

Submitted in partial fulfillment of the requirements of the Degree of Bachelor of Electrical and Electronic Engineering

Session: Apr 2024

ACKNOWLEDGEMENT

This page is solely dedicated to those involved in their utmost effort in the contribute to the successful completion of this project. First and foremost, I would like to thank my family and friends for their moral support and the much-needed motivation throughout the process of this final-year project.

Next, I would also like to take this opportunity to express my gratitude to the staff of INTI International College Penang, especially my project supervisor, Dr. Solahuddin for his time, guidance, mentorship, and unwavering support throughout the completion of this project. His expertise and insights have been significant in shaping its direction. Despite his demanding schedule, he consistently provided me with valuable information and guidance.

DECLARATION OF ORIGINALITY AND EXCLUSIVENESS

I hereby declare that the dissertation is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Coventry or other institutions.

[illegible]

(Chuah Hui Suen ,13466486)

Date: 20 May 2024

Supervised by: Dr Solahuddin Yusuf bin Fadhlullah

[illegible]

(Dr Solahuddin Yusuf bin Fadhlullah)

Date: 20 May 2024

Abstract

The Smart Classroom Attendance System developed in this project represents a solution designed to enhance the efficiency and accuracy of attendance tracking in educational settings. This system outlines the attendance-taking process by adding advanced technologies such as CNN (Convolutional Neural Network) and the dlib library for face detection, feature extraction, and recognition. The system aims to address specific objectives: achieving high accuracy in identifying multiple individuals simultaneously from a distance greater than three meters, verifying ten individuals in a single frame, and implementing real-time processing with a target time of 3 to 6 seconds.

This project investigates the performance of the facial recognition system under various parameters, including distance and the number of faces in the frame. The system addresses the challenges faced in typical classroom settings for taking attendance by detecting and recognizing multiple faces in real-time. It efficiently manages student identities, aligns faces for recognition, and updates attendance records in real time, contributing to a more interactive and data-driven educational environment.

Despite its advantages, the system faces limitations related to computational complexity and hardware constraints, which impact the processing time and accuracy, especially under varying distances and different distributions of faces detected. These challenges require ongoing refinement and adaptation to ensure the system's reliability and accuracy. Moreover, this project lays the foundation for the development of smarter classrooms, where technology optimizes administrative tasks, allowing educators to focus on their core mission of teaching. As the system continues to evolve through user feedback and performance improvements, it has the potential to revolutionize attendance tracking and contribute to the ongoing transformation of educational technology.

Table of Contents

List of Tables

List of Figures

1-Introduction

1.1 Project Background

In the age of digital transformation, educational institutions are looking for creative ways to improve efficiency, simplify administrative procedures, and create a more engaging, data-driven learning environment. The effective monitoring of student attendance, previously done manually and prone to errors, is an essential component of this change. Conventional techniques for recording attendance, including manually calling names or keeping paper records, are both fraught with difficulties. They are labor-intensive, prone to human mistakes, and do not have the real-time functionality required in today's classrooms. The need for an automated and accurate attendance recording system becomes even more crucial in the typical classroom setting, where a small group of students may be present. Instructors often use clickers or register manually for attendance recording, which has led to inefficiencies and potentially inaccurate data a few decades ago.

With the evolution of digital technology, there are now more ways to handle these issues, such as face recognition technology. Many institutions still encounter the same issues with their current attendance record systems, even though some of them are switching to digital attendance solutions. These restrictions on the attendance systems often include poor accuracy, delays in updating records, and a deficiency of up-to-date information for instructors. Moreover, these systems are less effective because they frequently need human assistance. Stakeholders in this project include educational institutions, instructors, and students. Accurate attendance tracking is crucial for institutions to meet regulatory requirements and improve resource allocation. Instructors benefit from having real-time attendance data, allowing them to adapt their teaching methods as needed. For students, the system ensures fairness and accountability in attendance recording.

The "Medium Range Facial Recognition for Attendance Recording" project aims to address the urgent need for an automated, effective, and precise solution for attendance recording in educational institutions. This project aims to build a system that can identify many people at once from a distance of more than 3 meters using facial recognition technology. The system integrates with a web server using the Flask framework, which manages HTTP requests and routing data. Instructors and administrators can review and manage the attendance records in an accessible format by automatically recording the recognized faces in an Excel spreadsheet.

This project represents a significant step toward the digital transformation of educational institutions' attendance recording systems. It aims to optimize administrative tasks, reduce the burden on educators, and provide a reliable and scalable solution for attendance tracking. In addition, the project also intends to increase data accuracy, lessen administrative burden, and improve the overall educational experience for instructors and students by automating attendance tracking and incorporating real-time capabilities.

1.2 Problem Statement

The majority of student attendance in nowadays educational settings is still recorded by hand using conventional methods. These traditional techniques have been around for a long time and still rely on name-calling, paper registers, and rudimentary clickers. They have proven helpful in monitoring attendance, but they have certain inherent issues that prohibit educational institutions from advancing into the digital era. The major problem with traditional methods of taking attendance is time-consuming and laborious. Time spent teaching is not supposed to be wasted on the task of manually recording every student's presence or absence in a classroom. This labor-intensive procedure is prone to error, which makes it challenging for institutions to administratively reconcile inequalities and produces erroneous attendance records.

Moreover, the current state of attendance tracking is out of step with the needs of current educational environments. It lacks the real-time functionality that modern learning environments require. It is common for instructors to encounter delays in updating attendance records, which impedes their ability to obtain immediate insights regarding student engagement. This knowledge is essential for effectively tailoring instructional strategies to each student's needs. Furthermore, the existing attendance tracking is not up to date with the changing requirements of educational settings. It doesn't have the necessary real-time features that contemporary learning environments demand. Inefficiency and resource misallocation might result from inaccurate recordkeeping. The major educators, the instructors, are also impacted by the existing state of attendance tracking. Taking attendance manually removes them from their primary responsibility, which is to deliver excellent education to students. They are unable to evaluate student participation in class and adjust their teaching methods in real-time due to inaccurate or delayed attendance data which brings substantial effects.

Apart from that, the end users of education are the students who are impacted. There has never been a greater need for an automatic, accurate, and effective way to check student attendance in the classroom. The "Medium Range Facial Recognition for Attendance Recording" project aims to solve these issues by automating attendance tracking and including real-time capabilities. The suggested solution makes use of face recognition technology to identify many people at once from a distance larger than three meters with an approximate accuracy rate of 93%. However, there are substantial technological obstacles in maintaining this precision in different classroom settings and at different distances.

Additionally, the system aims to verify ten individuals in a single frame which requires strong image processing methods and a significant amount of processing power. Targeted processing times of 3 to 6 seconds were set for multi-face detection and recognition; however, preliminary investigations reveal that this goal is difficult to meet because of the computational burden associated with processing high-resolution photographs over long distances. The system is always being optimized to speed up processing. To address these technical requirements, the system interfaces with a web server using the Flask framework, enabling efficient data handling and real-time processing. The system gives instructors and administrators a user-friendly approach to monitor attendance by automatically recording recognized faces into an Excel spreadsheet.

1.3 Problem Objectives

Under the problem stated in Chapter 1.2, a set of project objectives to address the matter has been formulated. These objectives are as shown below:

1. To develop a facial recognition system that can identify multiple individuals simultaneously at 93% accuracy with greater distances than 3 meters.

2. To establish a facial recognition system that can verify 10 individuals in one frame.

3. To implement a real-time facial recognition system that has a processing time of 3 to 6 seconds for multi-face detection and recognition.

2-Literature Review

2.1 Student Attendance System

Attendance tracking is a significant aspect of the educational system to ensure that students are actively participating in classes and fulfilling their academic obligations. Traditional attendance-taking methods often rely on manual processes, which can be labor-intensive, error-prone, and inefficient in adapting to the demands of modern education. This section explores various types of student attendance systems.

Next, various types of student attendance systems have been widely applied worldwide. All of those systems are named biometric attendance systems including facial recognition, fingerprint recognition, iris recognition, and voice recognition. They offer high accuracy and eliminate the possibility of proxy attendance. However, they have pros and cons at the same time.

Fingerprint recognition gives high accuracy and is widely accepted. However, it takes time for the verification process so the user has to line up and perform the verification one by one. Thus, it is inefficient in this case, especially when quick or mass verification is needed. After that, iris recognition is accurate but requires the collection of detailed information, which has invaded user privacy. The need for such detailed information can make users hesitant to use this technology due to privacy invasion concerns. Hereafter, voice recognition is less accurate compared to other recognition systems. Thus, it is not considered as the suitable method to apply as attendance tracking for students while the RFID card system can be implemented due to its simplicity. However, the user might tend to help their friends to check in as long as they have their friend's ID card. Hence, facial recognition is suggested to be implemented in the student attendance system with human faces exposed and contains less detailed information compared to iris recognition.

System type	Advantage	Disadvantage
Fingerprint recognition	High accuracy, widely accepted and used, fast recognition process	Fingerprint can be altered/ damaged, sensitive to environmental conditions, privacy concerns
Facial recognition	User-friendly, applicable, no physical contact needed	Accuracy can be affected according to facial appearance changes, vulnerable to spoofing, privacy concerns
Iris recognition	High accuracy and security, stable, reliable with age, unique and difficult to forge	High cost, invasive, limited use in low-light environment
Voice recognition	Non-intrusive, convenient for user, can be combined with other biometrics system, applicable in phone-based and IoT applications	Vulnerable to audio recordings, accuracy affected by background voice, user variability
RFID card system	Easy to use and implement, cost-effective, no privacy concerns related to biometric data	Cards can be lost or stolen, potential for card sharing or "buddy punching", requires distribution and maintenance of physical cards

Table 1: Comparison table between various biometric systems

2.2 Deep learning

Deep learning, a subset of machine learning and artificial intelligence (AI), mimics the way humans acquire knowledge, particularly in recognizing patterns across diverse data types like photos, text, and audio. It excels in automating tasks that traditionally require human intelligence, such as image description and audio transcription. This approach proves pivotal in data science, offering data scientists a faster and more efficient means of collecting, analyzing, and interpreting large datasets.

In essence, deep learning constructs neural networks with multiple interconnected layers, akin to the human brain's network of neurons. The process involves data collection and pre-processing, employing deep convolutional neural networks for feature extraction, data augmentation to enhance diversity, and training/validation phases for model refinement. Ensemble learning, combining multiple models, further boosts accuracy and robustness.

Deep learning’s significance lies in its applications, including digital assistants, fraud detection, and facial recognition, with high accuracy critical for safety-centric applications like autonomous cars and medical devices. The methodology involves training models on vast labeled datasets, and its efficacy is demonstrated through advancements in face recognition, and handling variations in lighting, pose, and expression.

While deep learning benefits from automatic feature learning and pattern discovery, challenges arise from biases in training data, learning rate management, and demanding hardware requirements. Despite limitations, deep learning finds applications in customer experience, text generation, aerospace, military, industrial automation, colorization of media, and computer vision. Ongoing advancements in the field continue to shape its potential applications, making deep learning a dynamic force in various industries. [6]

2.3 Transfer learning

Transfer learning, a widely adopted technique in deep learning, involves repurposing a pre-trained model for a new problem. This approach is particularly popular in scenarios where obtaining large labeled datasets is challenging, which is often the case in real-world applications. Essentially, transfer learning leverages the knowledge gained from solving one task to enhance the generalization capabilities for another related task. For instance, a classifier trained to recognize backpacks can use its knowledge to identify other objects like sunglasses. In practice, the method involves transferring the learned weights from one task (Task A) to a new task (Task B), allowing the model to build upon previously acquired patterns. Transfer learning finds extensive application in computer vision and natural language processing tasks, such as sentiment analysis, owing to its efficiency in handling complex models that demand substantial computational power. It stands out for its ability to save training time, enhance neural network performance, and operate effectively with limited data. The decision to employ transfer learning arises when there is insufficient labeled data, and a pre-trained network on a related task with ample data already exists. The technique proves valuable when the inputs for both tasks are the same. Approaches to transfer learning include training a model to reuse it, using a pre-trained model, and feature extraction. Popular pre-trained models, like Inception-v3, ResNet, and AlexNet, further facilitate the adoption of transfer learning in various applications. [7]

2.4 Comparison between deep learning and transfer learning performance

Deep learning and transfer learning are two prominent techniques within the field of machine learning that have revolutionized the way models are developed and deployed for various tasks. Both approaches have their unique advantages and use cases. In this section, we will make a comparison between deep learning and transfer learning in a paragraph and a table. Deep learning, a subset of machine learning, involves training complex neural networks on vast datasets to learn representations and patterns directly from the data. It is a data-hungry approach, requiring substantial amounts of labelled data and significant computational resources. Deep learning models are including deep neural networks (DNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). They are designed to handle specific tasks, ranging from image classification to natural language understanding. While deep learning models excel at tasks they were originally trained for, they often require lengthy training times and massive datasets to achieve high accuracy.

Transfer learning enhances the power of deep learning by utilizing knowledge gained from pre-trained models. These models undergo initial training on diverse and extensive datasets, enabling them to encompass a wide range of features and representations. Transfer learning adapts these pre-trained models to new, related tasks with smaller datasets. It reduces the need for extensive training and large datasets, resulting in faster convergence and often superior performance. By fine-tuning pre-trained models, or simply using their features as a starting point, transfer learning empowers machine learning practitioners to address specific problems efficiently, from image recognition to natural language processing.

Aspect	**Deep Learning**	**Transfer Learning**
Training data	Requires large labeled datasets specific to the target task	Can adapt to face recognition tasks with smaller datasets, leveraging pre-trained models

Computational Resources	Demands substantial computational power and training time for custom face recognition models	Reduces computational requirements by building on pre-trained models, enabling faster model development
Model Complexity	Employs custom deep neutral networks designed for face recognition, which can be complex and resource-intensive	Adapts pre-trained models as a starting-point, simplifying model architecture while maintaining performance
Generalization	Achieves high accuracy with extensive training data but might struggle with limited data or variations	Excels at face recognition tasks with limited data and challenging variations, thanks to pre-trained models
Data Efficiency	Inefficient when the target face recognition task has a scarcity of labeled data	Efficiently handles limited labeled data by leveraging pre-trained models and faster convergence
Real-World Use Cases	Applied in various face recognition scenarios, such as security, attendance systems, and access control	Beneficial in real-world scenarios where labeled face data is scarce or fast model deployment is essential

Table 2: Comparison table between performance of deep learning and transfer learning

In summary, deep learning models developed for face recognition are powerful when they have access to large, task-specific datasets and computational resources. They can achieve high accuracy in ideal conditions. However, transfer learning in face recognition excels when dealing with challenging real-world scenarios, limited labeled data, or when rapid deployment is a priority. By leveraging pre-trained models, transfer learning bridges the gap between the need for high accuracy and the constraints of real-world applications, making it a valuable approach in the field of face recognition.

2.5 Face detection techniques

Face detection is a computer vision task that involves identifying and locating human faces within digital images or video frames. The primary goal of face detection is to determine the presence, position, and often the size and orientation of one or multiple faces in a given visual input. It's important to distinguish face detection from face recognition. Face detection focuses on finding faces in an image or video, whereas face recognition involves identifying and matching those detected faces to specific individuals.

In addition to detecting faces, face detection typically provides information about the position of the detected faces, often expressed as bounding boxes (rectangular regions) around each face. These bounding boxes indicate where the faces are located in the image. The picture below shows the example of how bounding boxes look like.

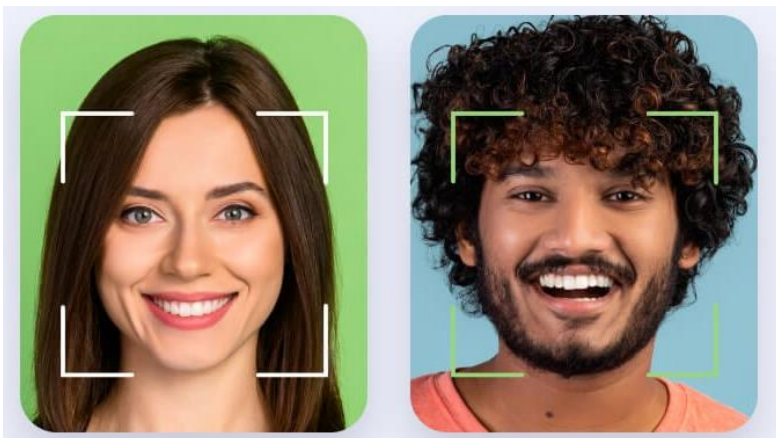


Figure 1: Example of face detection looks like

However, face detection can be a challenging task due to variations in lighting, pose, expression, and occlusion (when part of the face is obscured). Advanced face detection algorithms are designed to handle these challenges. Various types of algorithms and techniques are used for face detection, ranging from traditional methods like Haar cascades to more advanced deep learning-based approaches using convolutional neural networks (CNNs). Those algorithms and techniques are discussed in the small section following. Popular deep learning-based face detection models include Single Shot MultiBox Detector (SSD) and You Only Look Once (YOLO). [8]

2.5.1 Viola-Jones Algorithm

Object detection stands as a pivotal domain within computer vision, finding applications in diverse fields such as security systems, human-computer interaction, and image and video editing. A prominent framework in this realm is the Viola-Jones algorithm, specifically tailored for face and eye detection. Conceived by Paul Viola and Michael Jones in 2001, this algorithm is distinguished for its speed and efficiency, anchored in the ingenious combination of Haar-like features and the AdaBoost machine learning algorithm.
At its core, Haar-like features serve as the foundation of the Viola-Jones algorithm. These features are simplistic, rectangular calculations derived by subtracting the sum of pixel intensity values in a white region from the sum of intensity values in a black region. AdaBoost, on the other hand, plays a pivotal role as a machine learning algorithm that amalgamates multiple weak classifiers into a robust and accurate classifier.

The operational workflow of the Viola-Jones algorithm unfolds through a systematic series of steps. The process commences with the generation of an extensive set of Haar-like features, systematically computed across varied scales and locations within an image. Subsequently, AdaBoost is employed to discern the most salient features from this set and to train a classifier, harnessing the power of these selected features. The trained classifier then navigates through the image, utilizing a sliding window technique to evaluate the presence of an object. Upon detection, the window undergoes resizing, and the iterative process continues until the object is precisely located.

A compelling feature of the Viola-Jones algorithm lies in its expeditious execution, attributed to the utilization of an integral image representation. This representation enables rapid calculation of Haar-like features, contributing to the algorithm's impressive speed. Additionally, the incorporation of the AdaBoost learning algorithm further enhances efficiency by facilitating the training of a potent classifier with a reduced number of features, thereby minimizing computational demands.

In essence, the Viola-Jones algorithm stands as a cornerstone in object detection methodologies, showcasing a harmonious blend of Haar-like features and AdaBoost to achieve rapid and accurate detection of faces and facial features in images. Its versatile applications across various domains underscore its significance in advancing computer vision capabilities, offering a glimpse into the nuanced intricacies of this pioneering algorithm. [9]

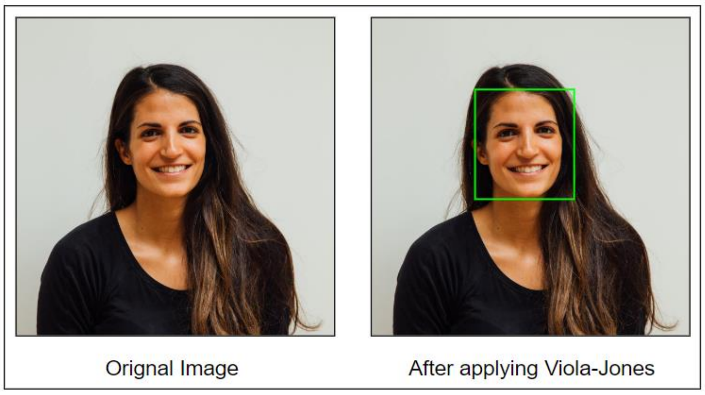


Figure 2: A demonstration of Viola-Jones algorithm

2.5.2 Histogram of Oriented Gradients (HOG)

The Histogram of Oriented Gradients (HOG) stands as a pivotal feature extraction technique within the realms of computer vision and image processing, finding extensive applications in the domains of object detection and image recognition. In the intricate landscape of computer vision tasks, where the representation of complex visual data in a meaningful and concise manner is paramount, HOG emerges as a robust solution by honing in on the distribution of gradient orientations within an image. This focus on capturing local intensity gradients and their orientations proves instrumental in characterizing object shapes and structures.

The HOG algorithm, a multi-step process, commences with image pre-processing to bolster its resilience against lighting variations and noise. This often involves converting the input image to grayscale, normalizing pixel intensities, and applying contrast normalization. The subsequent step involves the computation of gradient magnitudes and orientations of image pixels, facilitating the identification of edges and texture boundaries crucial for subsequent analysis.

Following gradient computation, the image is strategically partitioned into small, overlapping cells, typically covering regions of 8x8 pixels. Within each of these cells, a histogram of gradient orientations is computed, wherein the orientations undergo quantization into bins. This histogram encapsulates the distribution of gradient orientations within the given cell. Subsequently, these cells are grouped into larger blocks, often comprising 2x2 or 3x3 cells, and normalization is applied within each block. This normalization step significantly enhances the algorithm's robustness to changes in lighting conditions and contrast, contributing to its adaptability in diverse scenarios.

The culmination of the HOG algorithm involves the formation of a comprehensive descriptor. The normalized histograms from all blocks are concatenated to construct the final HOG descriptor for the image. This descriptor serves as a powerful representation, capturing the spatial distribution of gradients and their orientations across the entire image. The resulting HOG descriptor provides a condensed yet rich characterization of the image's structural features, making it particularly well-suited for subsequent tasks such as object detection and image recognition.

In essence, the Histogram of Oriented Gradients algorithm unfolds as a meticulous and systematic approach to feature extraction, leveraging the nuanced distribution of gradient orientations to distill complex visual data into a form that is both meaningful and conducive to diverse computer vision applications. [10]

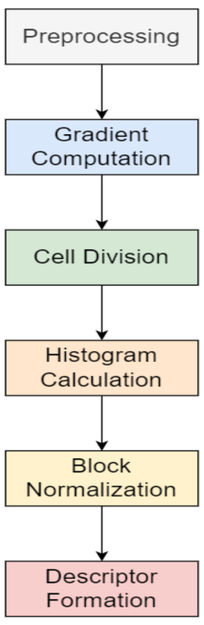


Figure 3: Steps involved when computing HOG

2.5.3 Cascade Classifier

A Haar cascade classifier is a type of cascade classifier that is a machine learning object detection program, that identifies objects in images and videos through four stages: calculating Haar features, creating integral images for efficient computation, using Adaboost for feature selection and training, and implementing cascading classifiers for efficient detection. This algorithm necessitates a substantial dataset of positive and negative images for training. Haar features involve calculations on adjacent rectangular regions within a detection window. Integral images expedite these calculations by reducing operations, using sub-rectangles and array references. Adaboost selects and trains the best features, combining weak classifiers into strong classifiers. Cascading classifiers consist of stages with trained weak learners, quickly rejecting negatives. Minimizing the false negative rate is crucial for effective object detection, and Haar cascades, though effective, require careful hyperparameter tuning. [11]

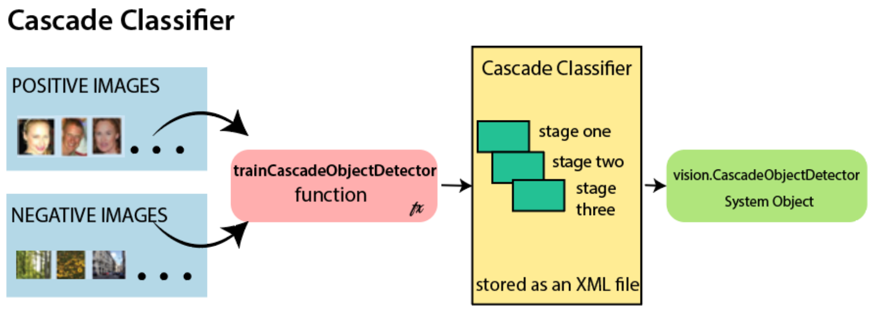


Figure 4: Process of Cascade Classifier

2.5.4 Scale-Invariant Feature Transform (SIFT)

Scale Invariant Feature Transform (SIFT), introduced by D. Lowe in 2004, stands as a pivotal feature extraction method in the realm of computer vision, specifically geared towards image matching and object detection applications. This algorithm operates on key terminologies such as Feature Extraction, which endeavors to reduce dataset features via a mapping function, Key Points that denote spatially invariant locations highlighting significant image pixels, and Descriptors, vectors delineating local surroundings around key points to establish associations between images. The inclusion of Gaussian Blur, a method adept at noise reduction in images, aids in efficient key point detection.

The advantages of SIFT underscore its locality, ensuring resilience against noise and clutter, distinctiveness, enabling comparison with extensive datasets, quantity generation even from diminutive objects, and noteworthy efficiency comparable to real-time performance. The execution of SIFT involves several stages, commencing with Building the Scale-Space. This entails applying Gaussian Blur to different scales of the original image, creating a multi-scale representation that mitigates scale dependency.

The subsequent stage, the Difference of Gaussian (DoG), amplifies features by subtracting higher-blurred from lower-blurred versions, resulting in a set of images for each octave. Key Point Localization involves comparing pixel values in localities, classifying them as potential key points only if they exhibit local extremum characteristics. To refine the generated key points, criteria such as contrast and edge alignment are applied, yielding a set of legitimate key points.

Orientation Assignment follows, involving the calculation of magnitude and orientation for each key point. Histograms are then created to represent orientation against magnitude, eliminating rotation and illumination dependencies. The final stage, Key Point Descriptor, involves forming a 16x16 grid around each key point, generating histograms for sub-blocks, and creating a feature vector. To eliminate rotation dependence, the gradient orientation difference is computed, while illumination dependency is mitigated through thresholding and normalization.

The culmination of SIFT lies in Key Point Matching, where the extracted key points serve as robust elements for pattern matching in other images, underscoring the algorithm's significance in object detection and image matching domains. Through its intricate steps, SIFT provides a comprehensive and detailed methodology for feature extraction, contributing to the robustness and adaptability of computer vision systems. [12]

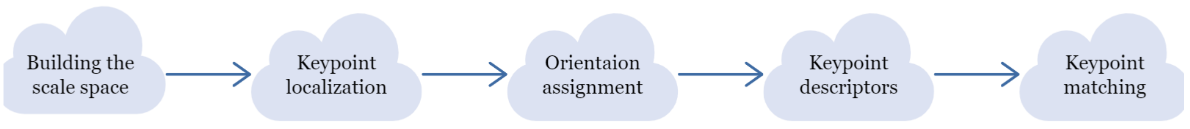


Figure 5: SIFT algorithm overview

2.5.5 Local Binary Patterns (LBP)

Recognizing faces is a complex task for computers due to various challenges such as illumination variation, low resolution, and occlusion. In the realm of computer-based face recognition, the Local Binary Pattern (LBP) algorithm has emerged as a noteworthy technique. First introduced in 1994, LBP combines statistical and structural methods to represent a facial image. The step-by-step architecture of LBP involves four key parameters: Neighbors (defining the number of samples to build the circular local binary pattern), Radius (representing the radius around the central pixel), Grid X (determining the number of cells in the horizontal direction), and Grid Y (specifying the number of cells in the vertical direction).

The process begins with the training of the algorithm using a dataset containing facial images, each associated with a unique identifier. This identifier, whether a number or a name, plays a crucial role in the algorithm's ability to recognize individuals. During training, photos of the same person are assigned the same ID. Once trained, the LBP algorithm employs a sliding window concept based on the specified radius and neighbor parameters to create an intermediate image that accentuates facial characteristics.

In practical terms, the grayscale image is divided into pixels, and a window (e.g., 3x3) is applied. This window is represented as a matrix, with the central pixel serving as the threshold value. The neighboring eight pixels are assigned binary values (1 if greater than the threshold, 0 if less), resulting in a binary matrix. Converting this binary value into a decimal value yields the central pixel's new intensity value, creating an image that better captures the original's features.

Following this, the Grid X and Grid Y parameters are employed to divide the image into multiple grids. As the generated image is grayscale, each pixel possesses a histogram with 256 positions. Concatenating these histograms produces a more comprehensive final histogram that accurately represents the characteristics of the original image. This final histogram, a unique signature for each image, facilitates the face recognition process.

In the face recognition stage, the trained algorithm utilizes the generated histograms from the training dataset to represent each image. When presented with a new input image, the algorithm repeats the process, creating a histogram that encapsulates its distinctive features. By comparing this histogram with those from the training dataset, the algorithm identifies and recognizes the individual depicted in the input image, showcasing the efficacy of the Local Binary Pattern algorithm in tackling the complexities of facial recognition in computer vision. [15]

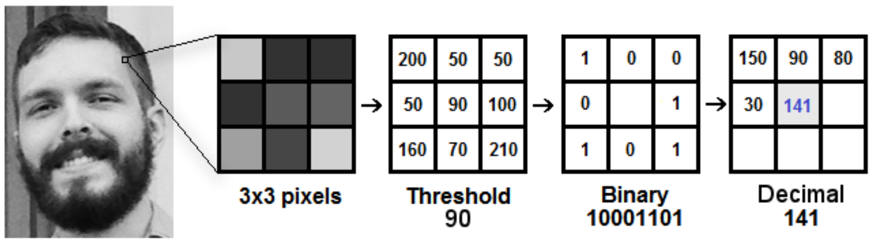


Figure 6: Applying LBP operation

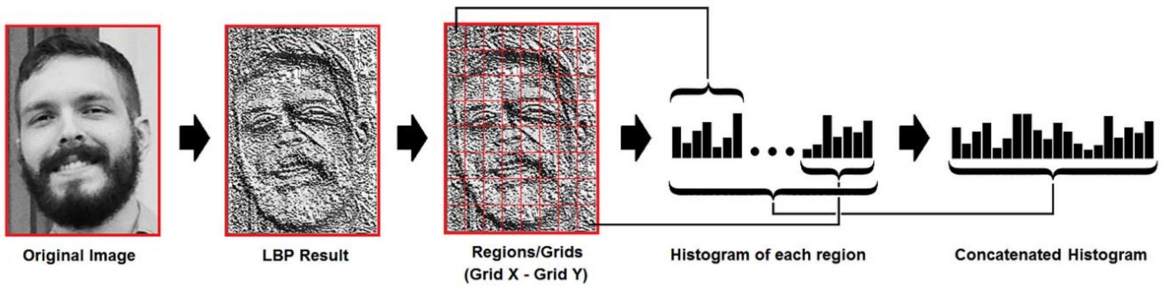


Figure 7: Extracting histograms

2.6 Face recognition techniques

Face recognition, also known as facial recognition, is a computer vision and biometric technology that involves identifying and verifying individuals based on their facial features. Unlike face detection, which focuses on locating faces in images or videos, face recognition goes a step further by associating the detected faces with specific individuals.

Face recognition serves two primary purposes which are identification and verification. It determines the identity of an individual by comparing their face to a database of known faces. Besides, it also verifies whether the person claiming to be a specific individual matches the stored reference face. In addition, face recognition systems analyze and extract facial features, such as the distance between the eyes, the shape of the nose, and the arrangement of facial landmarks such as eyes, nose, and mouth. These features are used to create a unique face template or facial signature for each individual.

However, face recognition systems must deal with variations in lighting, pose, facial expressions, and occlusions. There are various types of advanced algorithms are designed to handle these challenges including traditional methods such as eigenfaces, as well as deep learning-based approaches using convolutional neural networks (CNNs). Notable deep learning models for face recognition include FaceNet and VGGFace. [16]

2.6.1 Eigenfaces

Eigenfaces, a representation learning method within the realm of computer vision with a specific focus on facial images, operates on the premise of expressing a facial image as a linear combination of fundamental images termed eigenfaces. The crux of eigenfaces lies in the ability to discern optimal eigenfaces, thereby enabling the representation of any facial image through a linear combination of these fundamental components.

To embark on the journey of computing eigenfaces, a substantial set of facial images serves as the training dataset. A pivotal pre-processing step involves aligning facial features such as eyes, nose, and mouth, coupled with normalizing lightness and pose. This strategic pre-processing ensures that the algorithm can concentrate on the regions of the image pertinent to appearance, discarding irrelevant facial elements. Subsequently, the images undergo transformation into feature vectors encoding visual information. The covariance matrix of these vectors reveals correlations among various facial features, forming the basis for deriving eigenfaces.

The algorithm delves into the computation of eigenvectors from the covariance matrix, defining the latent space encapsulating facial variations. These eigenvectors then serve as the foundational images for the eigenface representation of subsequent test facial images. The training pipeline involves this intricate process, as illustrated in the provided diagram.

In the test phase, a new facial image undergoes transformation into a feature vector, followed by projection into the space defined by the eigenvectors. The resulting projection coefficients offer a concise representation of the face as a linear combination of the eigenfaces, as depicted in the corresponding diagram.

Eigenfaces boast several advantages, notably their efficiency and compactness. The coefficients derived from the eigenvectors are minimal, capturing the essential facial variations, facilitating the representation of new images using a sparse set of numbers. This efficiency proves valuable in diverse applications like search engines and security systems. Furthermore, eigenfaces exhibit robustness to lighting and pose variations, a critical attribute in classification tasks.

However, the method of eigenfaces is not without limitations. Its simplicity is accompanied by a dependence on the quality of the training set, rendering it less robust in representing a broad spectrum of facial images. The method excels in portraying faces akin to those in the training set, urging the need for a diverse training dataset to account for this inherent characteristic. Despite these limitations, the efficiency and compactness of eigenfaces position them as valuable tools in various facial recognition applications. [17]

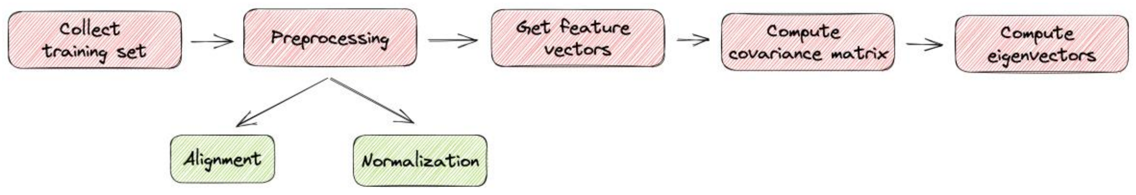


Figure 8: Training pipeline of eigenfaces

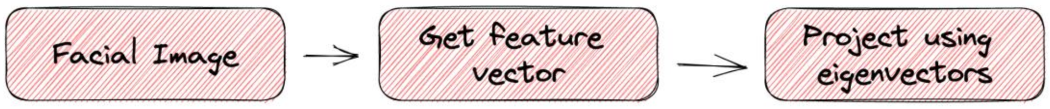


Figure 9: Test pipeline of eigenfaces

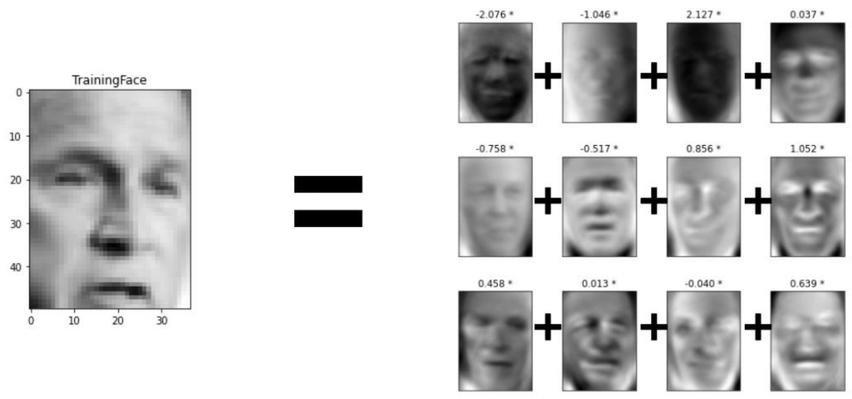


Figure 10: Example of an image represented as linear combinations of eigenfaces

2.6.2 Fisherfaces (Linear Discriminant Analysis)

In this comprehensive exploration, we delve into the intricacies of FisherFaces, an advanced technique for face recognition that builds upon the foundation laid by EigenFaces. FisherFaces leverages the combined power of Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) to enhance the accuracy and robustness of facial recognition systems. The overarching steps in face recognition, including capturing, feature extraction, and comparison, are outlined as the fundamental stages of this process.

OpenCV, a powerful computer vision library, provides three built-in face recognizers, namely EigenFaces, FisherFaces, and Local Binary Patterns Histograms (LBPH). Our focus in this article centers on FisherFaces, recognizing it as an improvement over EigenFaces. Before delving into FisherFaces, a brief background on EigenFaces elucidates its algorithmic approach. EigenFaces operates on the premise that not all parts of a face are equally crucial for recognition, emphasizing regions of maximum variation, such as those between the nose and eyes.

However, EigenFaces exhibits limitations, particularly in handling illumination as a significant feature, leading to potential inaccuracies. To address these shortcomings, the article introduces FisherFaces as an evolved version of the EigenFaces algorithm. FisherFaces acknowledges the significance of illumination variations and strives to extract individual features separately, preventing one person's facial data from unduly affecting others.

The FisherFaces algorithm, in essence, extracts principal components that distinguish one individual from another. By employing Fisher Linear Discriminant (FLD) or Linear Discriminant Analysis (LDA), it seeks to model the differences between classes, enhancing the separation between individuals in the feature space. LDA, a dimensionality reduction technique, aims to maximize the ratio of between-class scatter matrix to within-class scatter matrix, exhibiting resilience to varying illumination conditions.

The FisherFaces algorithm proves advantageous in not explicitly capturing illumination variations, offering a more refined approach compared to EigenFaces. Its core algorithmic steps involve calculating scatter matrices, seeking a projection matrix to maximize class separability, and solving the General Eigenvalue Problem to derive transformation matrices. Pseudocode provides a structured representation of the FisherFaces algorithm, emphasizing its implementation details.

To exemplify the application of FisherFaces, the article introduces the 'Yale Face Database' for training, featuring grayscale images with varying facial poses. The process involves data retrieval, image processing encompassing preprocessing and feature generation, and, finally, the recognition process. The recognition phase hinges on successfully matching test images with their corresponding training images, with the system exhibiting varying degrees of accuracy based on the similarity between the two. In cases where the training and testing images correspond, the system achieves a 100% recognition rate, while in scenarios where the images differ but belong to the same person, recognition rates of up to 90% are attained.

In summary, FisherFaces emerges as a sophisticated solution, addressing the limitations of EigenFaces in facial recognition. Its nuanced approach to feature extraction and recognition positions it as a valuable tool in computer vision applications, particularly in scenarios with varying illumination conditions and diverse datasets. [18]

2.6.3 Local Feature-Based Methods (SURF)

The Speeded Up Robust Features (SURF) method represents a significant advancement in the realm of image feature detection and matching, renowned for its speed and robustness. This algorithm, introduced in the Ph.D. thesis of H. Bay at ETH Zurich in 2009, builds upon the principles of Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). The primary focus of SURF is on providing a quick and accurate means of generating local, similarity-invariant representations for images, facilitating real-time applications like object recognition and tracking.

The SURF framework comprises two fundamental steps: feature extraction and feature description. In the feature extraction phase, interest points are detected using a Hessian matrix approximation. Integral images, introduced in 1984, play a crucial role in the computation of box-type convolution filters, enabling fast and efficient calculations of pixel value sums within rectangular regions. The Hessian matrix-based interest point selection relies on the determinant of the Hessian matrix, integrating information about both location and scale. Despite discretization and cropping limitations associated with Gaussian filters, SURF overcomes these challenges by employing box filters and maintaining fast convolution capabilities.

Scale-space representation, a common implementation in image pyramids, is achieved differently in SURF. The algorithm, leveraging box filters and integral images, foregoes iterative filtering and applies filters of varying sizes directly to the original image. This unique approach allows for parallel processing and efficient up-scaling of filter sizes without the need for iterative image size reduction. The feature extraction step involves non-maximum suppression in a 3×3×3 neighborhood to localize interest points across the image and scales.

Moving to feature description, SURF's creation of descriptors occurs in two distinct steps. The first step involves determining a reproducible orientation for the interest points by calculating Haar-wavelet responses in both x and y directions within a circular neighborhood around the keypoint. The orientation with the maximum sum of responses is chosen as the main orientation. The second step entails constructing a square region centered around the keypoint and aligned with the chosen orientation. This region is divided into smaller sub-regions, and for each sub-region, simple features are computed at regularly spaced sample points. The responses are weighted with a Gaussian to enhance robustness, and the resulting descriptor vector captures the intensity structure of each sub-region.

In essence, SURF's prowess lies in its ability to achieve speed and robustness through innovative techniques such as box filters, integral images, and parallel processing. This method significantly improves computation efficiency while maintaining accuracy, making it a valuable tool in various computer vision applications. [20]

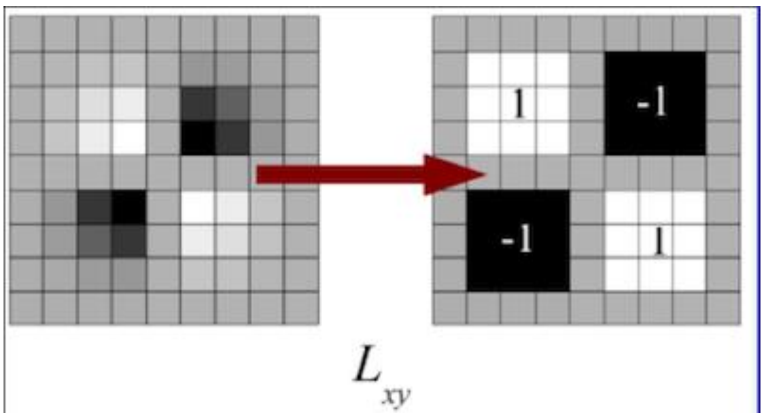


Figure 11: Gaussian partial derivative in xy

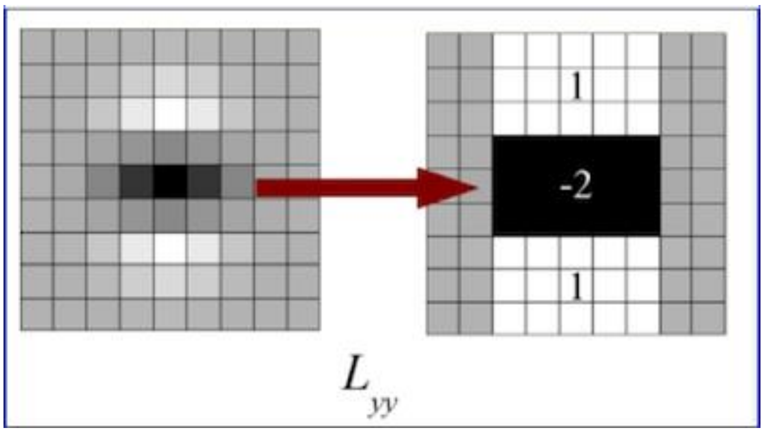


Figure 12: Gaussian partial derivative in y

2.6.4 Facial Landmark-Based Methods

Facial landmark detection algorithms play a pivotal role in automatically pinpointing key facial landmark points, such as the nose tip, eye corners, eyebrows, and chin tip, within facial images or videos. These algorithms find application in a variety of tasks, including face swap, head pose detection, detecting facial gestures, and gaze direction determination. The process of landmark detection involves two primary steps: face detection and landmark detection within the identified face bounding rectangle.

These algorithms are categorized based on facial appearance and shape patterns into three major types: holistic methods, Constrained Local Model (CLM) methods, and regression-based methods. Holistic methods explicitly model overall facial appearance and global facial shape patterns, while CLMs rely on local facial appearance and global shape patterns. Regression-based methods use holistic or local appearance information and may implicitly embed global facial shape patterns for joint landmark detection. Recent advancements involve combining deep learning models with global 3D shape models for more accurate landmark detection.

Several popular models are employed for facial landmark detection, with notable examples being the FacemarkLBF model from OpenCV, Dlib model, MTCNN model (Multi-task Cascaded Convolutional Networks), and the Mediapipe model developed by Google. The FacemarkLBF model returns 68 landmarks for each detected face, facilitating tasks such as face alignment and feature localization. Dlib offers a pre-built model, shape_predictor_68_facemarks.dat, providing 68 feature points. MTCNN, a deep learning architecture, employs cascaded convolutional networks for both face detection and landmark localization, detecting five key facial landmarks. Mediapipe, developed by Google, uses a holistic model to detect face and hand landmarks, providing 468 face landmarks along with hand landmarks.

Despite the progress in facial landmark detection, challenges persist due to factors such as diverse facial expressions, varying head poses, environmental conditions like illumination changes, and occlusion by other objects. Ongoing research endeavors aim to address these challenges and further enhance the accuracy and robustness of facial detection and landmark localization algorithms. As the field continues to evolve, the quest for a model that excels in performance across diverse conditions remains an active area of exploration. [21]

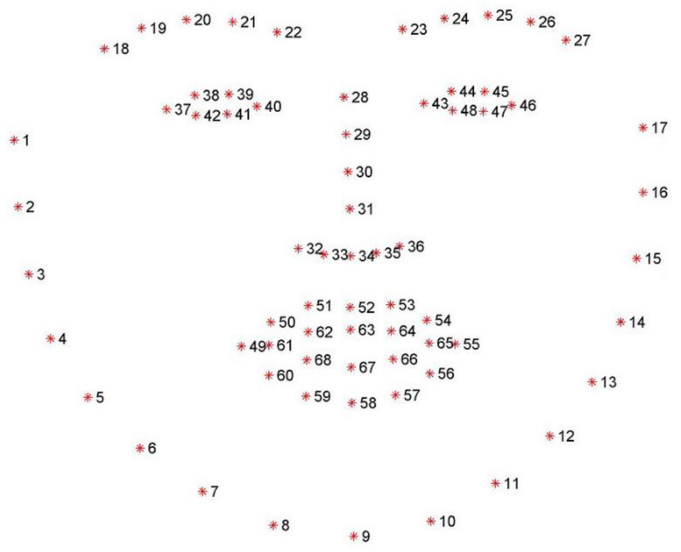


Figure 13: DLib landmark points on face.

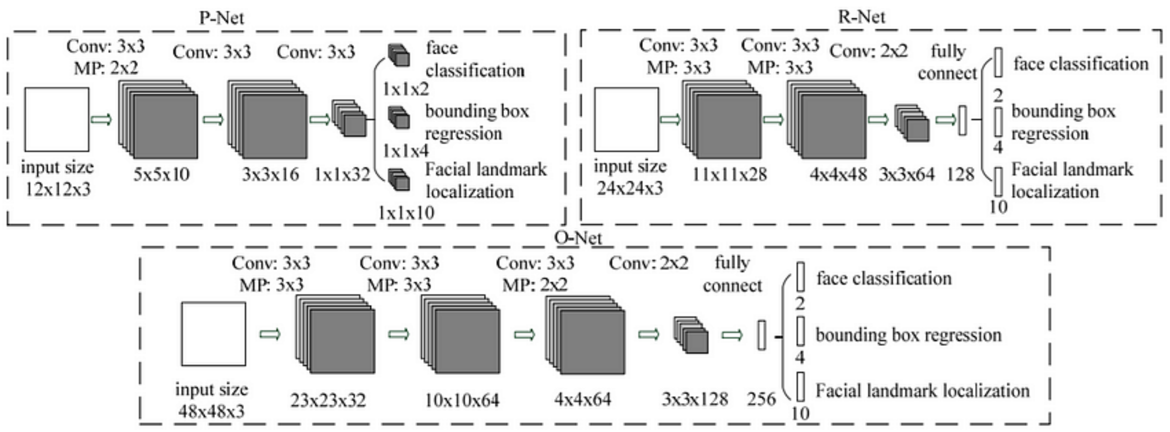
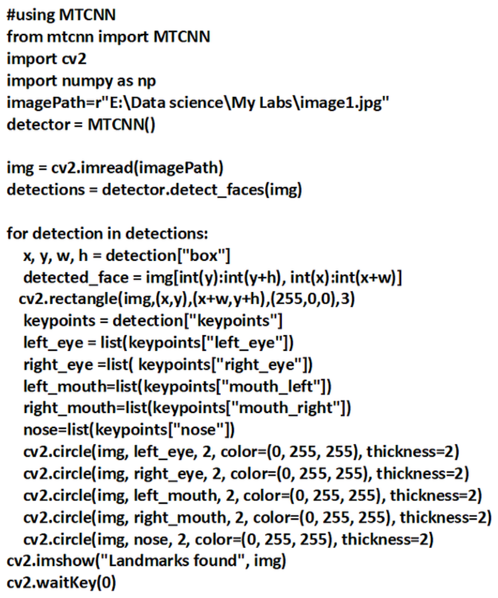
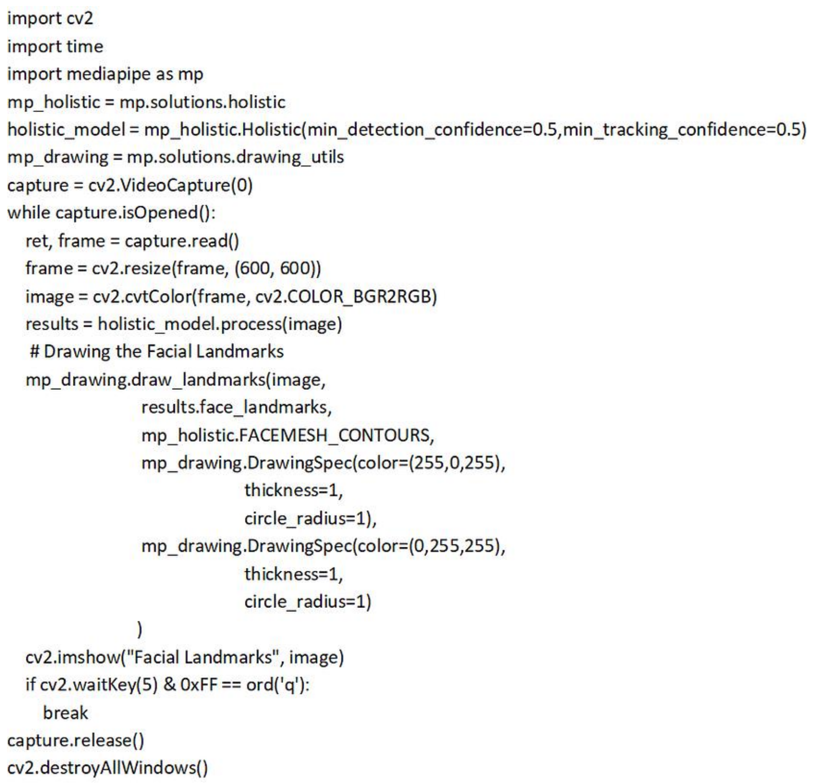


Figure 14: MTCNN architecture



```
#using MTCNN
from mtcnn import MTCNN
import cv2
import numpy as np
imagePath=r"E:\Data science\My Labs\image1.jpg"
detector = MTCNN()

img = cv2.imread(imagePath)
detections = detector.detect_faces(img)

for detection in detections:
    x, y, w, h = detection["box"]
    detected_face = img[int(y):int(y+h), int(x):int(x+w)]
   cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),3)
    keypoints = detection["keypoints"]
    left_eye = list(keypoints["left_eye"])
    right_eye =list( keypoints["right_eye"])
    left_mouth=list(keypoints["mouth_left"])
    right_mouth=list(keypoints["mouth_right"])
    nose=list(keypoints["nose"])
    cv2.circle(img, left_eye, 2, color=(0, 255, 255), thickness=2)
    cv2.circle(img, right_eye, 2, color=(0, 255, 255), thickness=2)
    cv2.circle(img, left_mouth, 2, color=(0, 255, 255), thickness=2)
    cv2.circle(img, right_mouth, 2, color=(0, 255, 255), thickness=2)
    cv2.circle(img, nose, 2, color=(0, 255, 255), thickness=2)
cv2.imshow("Landmarks found", img)
cv2.waitKey(0)
```

Figure 15: Landmark detection using MTCNN



```
import cv2
import time
import mediapipe as mp
mp_holistic = mp.solutions.holistic
holistic_model = mp_holistic.Holistic(min_detection_confidence=0.5,min_tracking_confidence=0.5)
mp_drawing = mp.solutions.drawing_utils
capture = cv2.VideoCapture(0)
while capture.isOpened():
    ret, frame = capture.read()
    frame = cv2.resize(frame, (600, 600))
    image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = holistic_model.process(image)
     # Drawing the Facial Landmarks
    mp_drawing.draw_landmarks(image,
                          results.face_landmarks,
                          mp_holistic.FACEMESH_CONTOURS,
                          mp_drawing.DrawingSpec(color=(255,0,255),
                                           thickness=1,
                                           circle_radius=1),
                          mp_drawing.DrawingSpec(color=(0,255,255),
                                           thickness=1,
                                           circle_radius=1)
                          )
    cv2.imshow("Facial Landmarks", image)
    if cv2.waitKey(5) & 0xFF == ord('q'):
        break
capture.release()
cv2.destroyAllWindows()
```

Figure 16: Mediapipe face landmark detection

2.6.5 3D Face Recognition

In the contemporary era where our lives are intricately interwoven with digital platforms, the imperative of safeguarding personal information has never been more pronounced. With projections estimating cybercrime to cost the world $6 trillion annually by 2021, conventional security measures like passwords and PINs are proving insufficient. The protection of sensitive information and personal identities has become paramount, underscored by the escalating incidences of identity fraud and spoof attacks. In response to these challenges, 3D face recognition emerges as a transformative technology in the realm of digital security.

Three-dimensional face recognition, also referred to as 3D facial recognition, stands out as an innovative method within the broader landscape of facial recognition. It leverages the inherent 3D structure of the human face to establish a definitive identity, setting it apart from traditional 2D approaches. Unlike 2D methods, 3D face recognition involves a meticulous process of 3D face reconstruction, introducing a heightened level of accuracy and performance that even surpasses fingerprint recognition.

Consider the unique facial features such as ridges, nose bridges, and indentations. While 2D images may capture facial patterns, they fall short in representing these depth-related intricacies. The efficacy of the 3D face recognition system hinges on its adept utilization of data. Researchers employ 3D face scans and extensive databases to craft detailed 3D face models capable of sophisticated pattern analysis, encapsulating intricate facial details like lighting conditions, poses, and expressions.

The superiority of 3D face recognition over its 2D counterpart is rooted in its prowess in pattern recognition, capturing and analyzing the depth of the human face. Unlike 2D methods reliant on flat images, 3D face recognition utilizes a comprehensive 3D face model, thereby capitalizing on the intrinsic 3D geometry of the human face. This approach provides a formidable defense against challenges such as changing lighting conditions, diverse facial expressions, and varying head angles.

Central to the success of 3D face recognition is the pivotal role played by artificial intelligence (AI). Advanced algorithms conduct intricate pattern analysis on the 3D facial data, ensuring robust recognition even amidst variations in pose and changes in illumination. A noteworthy study from the University of York underscores the substantial improvement in facial recognition results achieved through 3D models compared to 2D images. The amalgamation of depth analysis and AI algorithms renders the system highly adaptable and accurate.

Imagine a scenario involving twins with nearly identical facial features attempting to access a security system. While a 2D system might falter due to their striking resemblances, 3D face recognition discerns between subtle facial depth differences, ensuring correct identification.

The operational workflow of 3D face recognition encompasses face detection, facial landmarks identification, feature extraction, and verification. From capturing 3D facial data and analyzing depth and curvature features to mapping every facial detail into a digital signature for verification, the process is swift and secure. In scenarios such as airport security equipped with 3D face recognition, a traveler's identity is seamlessly validated by cross-referencing it with global databases in milliseconds.

The transition from 2D to 3D face recognition signifies a seismic shift in authentication accuracy. Unlike 2D approaches reliant solely on visual characteristics, 3D face reconstruction employs the intricate geometry of the face for identification, resulting in enhanced accuracy. This shift brings forth a multitude of benefits, including resilience to spoofing, improved robustness in challenging conditions, and the strength of 3D liveness detection.

Facial recognition, while powerful, is not foolproof, with sophisticated cyber-attacks using high-resolution photos or videos occasionally tricking systems. Here, the strength of 3D liveness detection becomes pivotal. By adding an extra layer of facial geometry and monitoring real-time presence, it ensures interactions with a real, live human, thwarting attempts based on photographs or videos.

In real-world applications, 3D face recognition finds substantial utility across diverse sectors. In the financial sector, banks like HSBC streamline authentication processes and reduce fraudulent access attempts using facial recognition. In e-commerce and retail, exemplified by Amazon's Go stores, 3D facial recognition enhances frictionless shopping experiences, ensuring accurate billing without manual intervention. Moreover, in aviation, airports like Changi in Singapore lead the way by pioneering facial recognition for smoother boarding processes, ensuring the person boarding is the rightful ticket holder.

When considering a partner for facial recognition surveillance, FACIA emerges as a leader in implementing biometric solutions for enhanced security. The seamless integration of face recognition with 3D liveness detection ensures the authenticity of users' identities. Their solutions, driven by artificial intelligence, encompass AI-powered facial recognition, liveness detection with 3D face search capabilities, and next-generation face-matching technology offering 1:1 face matching and 1:N face verification, age verification, and on-premises solutions for various sectors.

As we gaze into the future, security, in the era of digital transformation, transcends luxury to become a necessity. Significant investments by entities ranging from Apple to national governments underscore the prominence of 3D face recognition in the convergence of technology and security. The synergy of face recognition and 3D liveness detection offers robust protection in a dynamic landscape. FACIA leads the way, providing tools to confidently navigate this new territory and ensure uncompromising security.

In conclusion, the challenges posed by the digital age are myriad, but with tools like 3D face recognition, we're not merely reacting to threats; we're preempting them. Standing on this technological precipice prompts not a question of whether to adopt such technologies, but rather how soon we can integrate them. FACIA's infusion of AI-driven solutions, including 3D face recognition and liveness detection, offers innovative solutions that resonate across sectors, reflecting a proactive stance in addressing the multifaceted challenges of the digital age. [22]

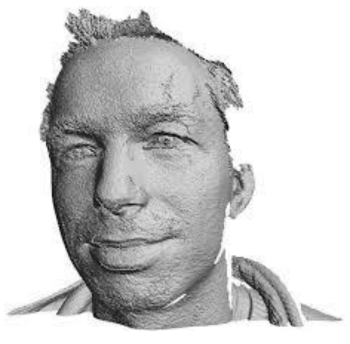


Figure 17: 3D face model

2.6.6 Multimodal (Fusion) Approaches

In recent years, machine learning algorithms, particularly neural networks, have gained substantial prominence owing to their remarkable accuracy in training models. Neural networks, inspired by the human brain, have become a focal point in both academic research and industrial applications due to their superior performance within a single domain dataset. However, contemporary research is increasingly delving into the realm of multimodal input data, marking a shift from unimodal datasets. Multimodality, as defined by Lahal et al. [3], involves systems observed by multiple sensors. The primary objective of leveraging multimodality is to extract and amalgamate essential information from individual sensors, creating a composite feature set to address a given problem. This approach aims to endow the output with a richer representation and enhanced performance compared to individual modalities. Multimodal data analysis finds practical applications across diverse fields such as medicine, business, driverless technology, and gaming, where common remote sensing devices like cameras, LIDAR, radar, and ultrasonic sensors are frequently fused [4].

In the realm of multimodal data fusion, three distinct techniques are commonly employed [5] [6]. The first technique is early fusion or data-level fusion, a traditional method that involves combining multiple datasets before analysis. Early fusion, also known as input-level fusion, poses challenges related to data pre-processing, synchronization of data sources with varying sampling rates, and assumptions of conditional independence between data sources. The process entails either removing the correlation between sensors or fusing data at a lower-dimensional common space, facilitated by statistical methods such as principal component analysis (PCA) and canonical correlation analysis. Despite its historical significance, early fusion has drawbacks, including a substantial reduction in data volume and the complexity of synchronizing timestamps across modalities.

Conversely, the second technique, late fusion or decision-level fusion, operates by independently using data sources and subsequently fusing them during the decision-making stage. Inspired by ensemble classifiers, late fusion proves advantageous when dealing with significantly varied data sources in terms of sampling rate, data dimensionality, and measurement units. Late fusion often outperforms early fusion, particularly as errors from multiple models are treated independently, mitigating correlated errors. However, debates persist regarding whether late fusion consistently surpasses early fusion in performance.

Various rules, such as Bayes rules, max-fusion, and average-fusion, guide the optimal combination of independently trained models.

The third technique, intermediate fusion, aligns with the architecture of deep neural networks. This method stands out as the most flexible, allowing data fusion at different stages of model training. Intermediate fusion involves transforming input data into a higher-level representation through multiple layers, incorporating both linear and nonlinear functions. In the context of deep learning multimodal fusion, intermediate fusion entails merging representations of different modalities into a shared representation layer, facilitating the learning of a joint representation. This fusion can occur simultaneously or gradually, utilizing different modalities at different stages of the model. Unlike early and late fusion, intermediate fusion provides the flexibility to fuse features at various depths, enhancing adaptability. Dimensionality reduction techniques, such as principal component analysis (PCA) and autoencoders, are often employed to optimize performance and prevent overfitting.

Research efforts, exemplified by Karpathy et al. [18], explore "slow-fusion" networks, gradually fusing features across multiple layers for improved performance in video stream classification problems. Similarly, progressive fusion methods, as proposed by other studies [19], prioritize highly correlated input modalities before gradually incorporating less correlated ones. These approaches showcase state-of-the-art performance, especially in complex tasks like communicative gesture recognition. In essence, multimodal data fusion techniques play a pivotal role in addressing the intricacies of diverse datasets, offering a spectrum of strategies to enhance the robustness and adaptability of machine learning models across various applications. [23]

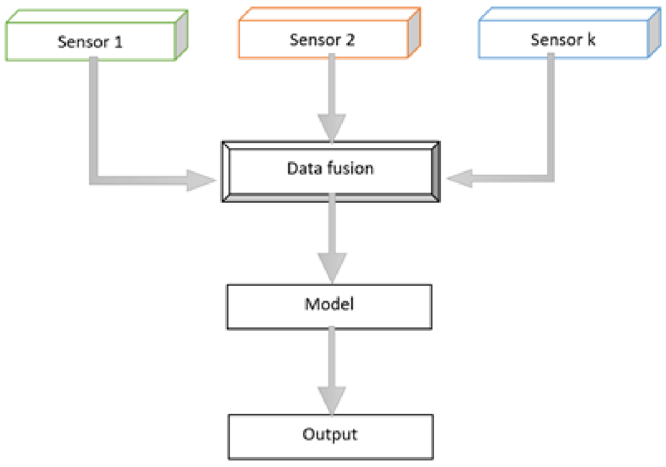


Figure 18: Early fusion or data-level fusion

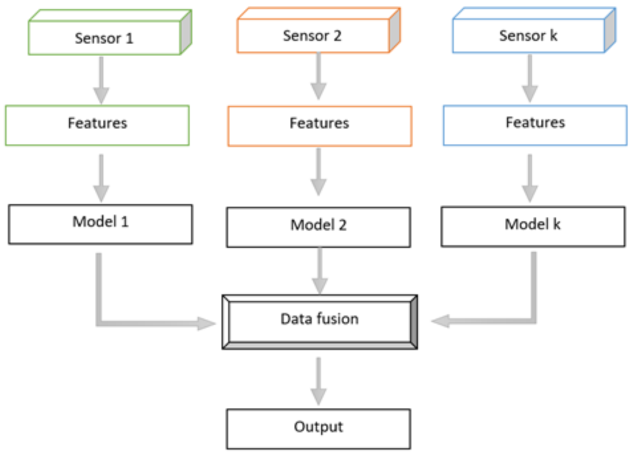


Figure 19: Late fusion or decision fusion

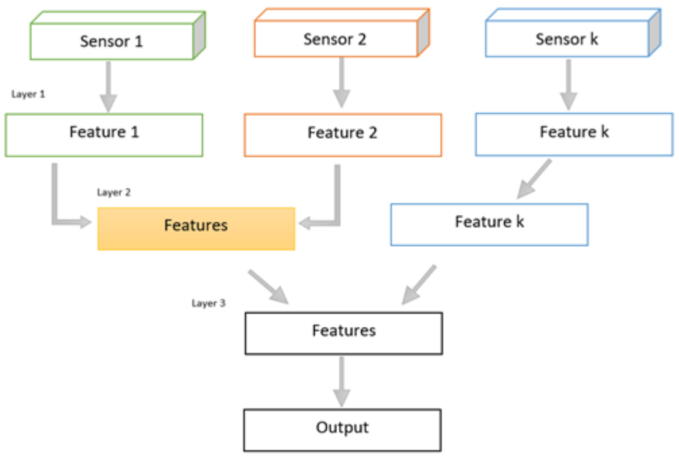


Figure 20: Intermediate fusion

2.7 Existing Project Methodology

This section studies the existing project methodology that has been proposed by the others for facial recognition applications with brief explanation.

The first project starts from “Multi-face recognition for the detection of prisoners in jail using a modified cascade classifier and CNN” proposed by IGSM Diyasa, A Fauzi, M Idhom, and A Setiawan in 2021. They introduced a method that combines deep neural networks that are Convolutional Neural Networks (CNN) and Haar Cascade Classifier as real-time applications to solve the difficulty of classification problems. This method is implemented with the guide of the OpenCV library for multi-face detection and 5MP CCTV camera devices. Furthermore, this method has been proven to be very efficient in face classification since its facial recognition system performance achieves an 87% accuracy rate. [24]

The second project continues to “Multi-face recognition process using Haar Cascade and Eigenface methods.” This method was proposed by Teddy Mantoro*, Media A.Ayu, and Suhendi in 2018. The proposed face recognition process was done using a hybrid process of Haar Cascade and Eigenface methods which can detect multiple faces in a single detection process. This facial recognition system can detect 55 individuals in a single detection process. This improved face recognition approach was able to recognize multiple faces with a 91.67% accuracy level. [25]

The third project title is “University Classroom Attendance System Using FaceNet and Support Vector Machine” which was proposed by Thida Nyein and Aung Nway Oo in 2019. This attendance system provides multi-face recognition with the combination of FaceNet and Support Vector Machine (SVM). In this proposed system, FaceNet is used for feature extraction by embedding 128 dimensions per face and SVM is used to classify the given training data with the extracted feature of FaceNet. The experimental result shows that the proposed approach with an accuracy of 98.66% within 55 students detection in a classroom. [26]

2.8 Summary

This section summarizes and compares the advantage and disadvantage of various types of face detection techniques and face recognition techniques in a table. Besides, this section also provides the comparison of existing methodology in a table from different aspects.

Technique type	Advantage	Disadvantage
Viola-Jones Algorithm	Fast, computationally efficient, works well for detecting frontal faces, suitable for real-time applications	Sensitive to variations in lighting conditions, limited accuracy, requires training phase for classifier creation
Histogram of Oriented Gradients (HOG)	Effective, robust to variations in lighting and orientation, can used in combination between SVM for improved accuracy	Requires additional techniques for precise facial feature localization, may produce false positives in complex scenes, computationally intensive in high resolution images
Deep learning (CNN)	Achieves state-of-the-art accuracy, can handle various poses, expressions, occlusions, support end-to-end training for feature extraction and classification	Requires large datasets for training, demanding resource-limited devices, prone to overfitting with limited data
Cascade classifier (eg: OpenCV Haar Cascades)	Efficient, suitable for real-time applications, can be trained for specific objects, well-suited for resource-constrained devices	Less accurate compared to deep learning model, may produce false positives in cluttered scene, limited adaptability to complex scenarios
Scale-Invariant Feature Transform (SIFT)	Robust to scale, rotation, affine transformations, effective for detecting faces under various conditions, can be combined with other techniques for improved accuracy	Slower and less suitable for real-time applications, sensitive to changes in viewpoint and occlusions, may require additional post-processing for face localization
Local Binary Patterns (LBP)	Efficient, lightweight, robust to the changes in lighting conditions, suitable for resource-constrained devices	Limited accuracy in complex scenes and under extreme variations, may struggle with occluded and non-frontal face, limited feature representation for detailed face analysis

Table 3: Comparison table between various face detection techniques

Technique type	Advantage	Disadvantage
Eigenfaces	Simple, efficient, effective for small to medium-sized databases, low memory requirements	Limited to variations in lighting and pose, less accurate with significant variations in face expressions, requires feature extraction and dimensionality reduction
Fisherfaces (Linear Discriminant Analysis)	Discriminative feature extraction, suitable for face recognition, handles within-class variability effectively, can be combined with other techniques for improved accuracy	Sensitive to variations in illumination and pose, requires relatively large dataset for training, limited robustness to extreme variations
Local Binary Patterns (LBP)	Efficient, lightweight, robust to the changes in lighting conditions, suitable for resource-constrained devices	Limited accuracy in complex scenes and under extreme variations, may struggle with occluded and non-frontal face, limited feature representation for detailed face analysis
Deep learning (CNN)	Achieves state-of-the-art accuracy, can handle various poses, expressions, occlusions, support end-to-end training for feature extraction and classification	Requires large datasets for training, demanding resource-limited devices, prone to overfitting with limited data
Local Feature-Based Methods (SIFT, SURF)	Robust to scale, rotation, affine transformations, effective for detecting faces under various conditions, can be combined with other techniques for improved accuracy	Slower and less suitable for real-time applications, sensitive to changes in viewpoint and occlusions, may require additional post-processing for face localization
Facial Landmark-Based Methods	Effective for precise face alignment and feature localization, enables accurate pose estimation and facial expression analysis, useful in combination with other techniques for enhanced accuracy	May not be suitable for full-scale face recognition on its own, relies on accurate landmark detection, intensive for real-time applications

3D face recognition	Provides depth information, effective in distinguishing identical twins and similar-looking individuals, less susceptible to variations in lighting and 2D photo spoofing	Requires specialized hardware, limited availability of 3D facial data, increased complexity
Multimodal (Fusion) Approaches	Combines multiple techniques for improved accuracy, provides redundancy and robustness, useful for applications with stringent security requirements	Increased complexity, more complex to maintain and implement, requires synchronized data from different modules, increased computational and storage requirements

Table 4: Comparison table between various face recognition techniques

Author name	Feature, Method & Applications	Accuracy
I G S M Diyasa, A Fauzi, M Idhom and A Setiawan (2021)	❖ Modified Cascade Classifier and CNN ❖ Application: Multi-face recognition for the detection of prisoners in jail	87.1%
Teddy Mantoro* Media A. Ayu Suhendi (2018)	❖ Multi-face recognition process using Haar Cascades and Eigenface methods ❖ Application: Multi-face recognition	91.67%
Thida Nyein Aung Nway Oo	❖ FaceNet and Support Vector Machine (SVM) ❖ Application: University classroom attendance system	98.66

Table 5: Comparison table between existing methodologies

3- Methodology

3.1 Overall System Design

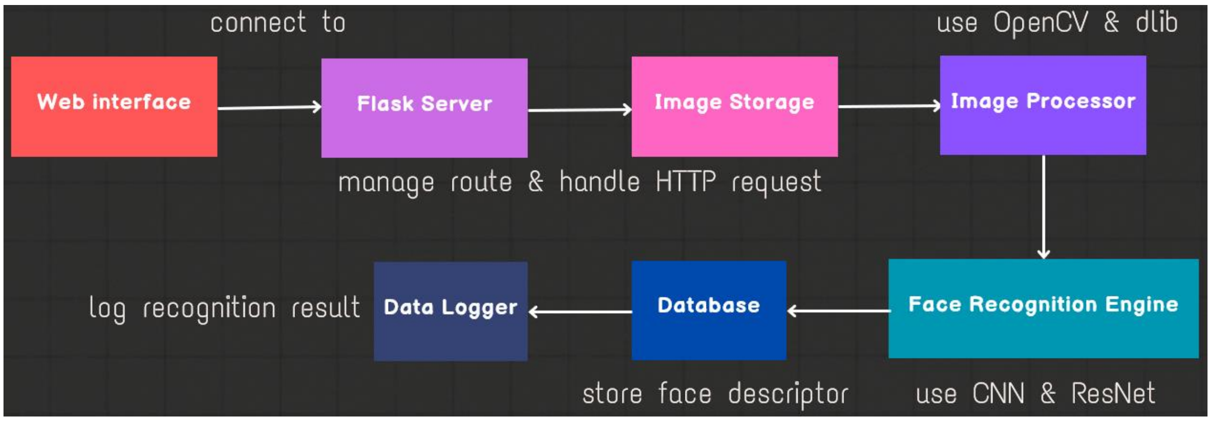


Figure 21: Overall system block diagram

The figure above illustrates the overall design of the facial recognition system. The input of the system is the image of students captured by a smartphone in real-time. The process is separated into many key components: web interface, Flask server, image storage, image processor, face recognition engine, database, and data logger. The output of the system is an updated attendance tracking record of the students in an Excel file. A detailed explanation of the overall system working is provided below.

The web interface serves as the primary access point for users, such as instructors to interact with the system. Users can capture images of students who are present in the classroom in real time using a smartphone through this interface. The interface facilitates the connection to the Flask server, allowing for the upload of captured images and the retrieval of updated attendance records. It ensures that users can easily manage and monitor attendance without needing direct access to the backend process.

Next, the Flask server acts as the central hub for the system, managing routes and handling HTTP requests. The Flask server receives this image and routes it to the appropriate components when an image is captured and uploaded via the web interface. It first directs the image to image storage for temporary holding. Additionally, the Flask server manages data flow between various components, ensuring seamless operation and integration of the system.

After that, image storage is the place that temporarily holds the images that are captured and uploaded through the Flask server. It provides a structured location for storing these images before they are processed. Moreover, image storage also ensures that images are readily available for the next stage of processing by acting as a buffer. Thus, this will maintain the efficiency and organization of the data flow within the system.

Afterward, the image processor will retrieve stored images from the image storage and perform necessary pre-processing tasks. The image processor detects and localizes multiple faces within each image using libraries including OpenCV and dlib. This pre-processing step includes resizing and normalizing the images to prepare them for accurate face recognition. The image processor is crucial for ensuring that the images are in the correct format and quality for subsequent recognition tasks.

Hereafter, the face recognition engine is responsible for identifying the faces detected by the image processor. Leveraging advanced neural networks like CNN (Convolutional Neural Networks) and ResNet (Redisual Networks), the engine compares the detected faces with a database of known face descriptors. This stage involves feature extraction and matching, where the unique features of each face are identified and compared to existing records are determine the identity of each student.

Apart from that, the database component stores the face descriptors and other necessary data for face recognition. The face recognition engine retrieves the corresponding face descriptor from the database for comparison when it identifies a face from the images. Additionally, new face descriptors can be added to the database as needed. In other words, the database is crucial for maintaining an up-to-date repository of face descriptors that the system can reference during the recognition process.

From there on, the data logger logs the results of the recognition process, including recognized faces and timestamps. It provides a record of the recognition events, which can be used for auditing and tracking purposes.

Lastly, the final stage of the system is updating the attendance records. Based on the recognition results logged by the data logger, the facial recognition system will then update the attendance records in an Excel file after completing the recognition process. This file serves as the official record of student attendance, which instructors can access and manage through the web interface. The real-time update of attendance records ensures that the system provides accurate and current data to educational institutions.

3.2 Hardware Design

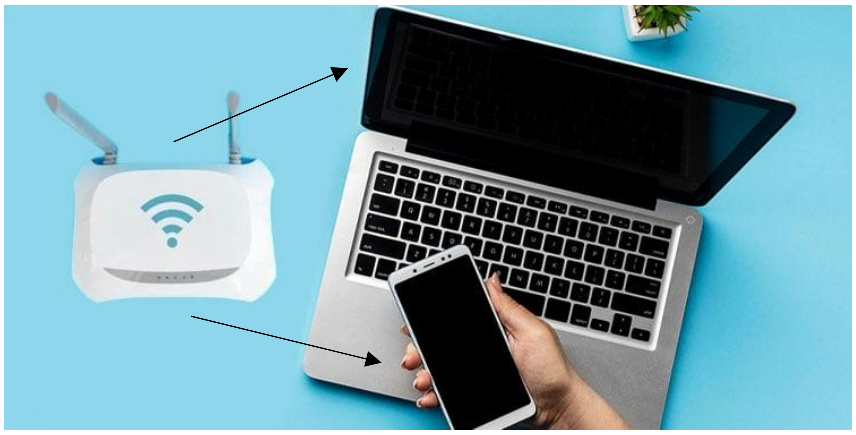


Figure 22: System hardware design

The figure above shows the conceptual hardware design of the project. The hardware used in this project is a smartphone, a laptop, and a Wi-Fi router. The purpose of using the router is to provide a Wi-Fi network and enable wireless communication between smartphone and laptop at the same time. The camera on the smartphone serves as an input source since it is used to capture images of the students. After the image is taken, the photo is sent to the laptop over the Wi-Fi network, triggering the program. The laptop is used to run the backend program which is our facial recognition system implemented in Python using OpenCV. Moreover, the laptop also acts as the main development tool in the project since it is used to write the code and debug the program. The primary functions of the laptop in this project are receiving images, processing images, rendering confirmation, executing the facial recognition program, storing and logging recognition results, and saving it into an Excel spreadsheet.

3.3 Software Design

In this project, the software design is divided into three main scripts, each responsible for different aspects of the facial recognition system. These scripts include “Flask.py” for handling web server functionalities, “feature_extraction.py” for extracting facial features from images, and “face_recognize.py” for processing images and performing face recognition. This section provides a detailed explanation of the responsibilities, key functions, and interactions of these scripts. Furthermore, the face detection algorithm, face recognition algorithm, and data interfacing also will be discussed in this section.

3.3.1 Flask Web Server (“Flask.py”)

Figure 23: Web Framework for Python

The “Flask.py” script uses the Flask framework to create a simple web server that provides file uploads, processes the uploaded images, and provides feedback to the user through a response page. This script comprises three main functionalities: file upload interface, file processing, and response page. Firstly, the file upload interface provides a user-friendly web page where users can upload images for processing. Users can select and upload multiple files at once, making the process efficient and straightforward. The Flask application initializes the necessary configurations to handle these file uploads.

3.3.2 Feature Extraction (“feature extraction.py”)

Figure 24: Dlib + OpenCV

The “feature_extraction.py” script focused on extracting and storing facial features data. The primary purpose of this script is to handle the extraction of facial features from detected faces using pre-trained models from the “dlib” library. The process involves identifying key landmarks on the face and computing a high-dimensional feature vector (or descriptor) that uniquely represents the facial characteristics of each individual. These feature vectors are then stored in a structured format which is a CSV file, for easy access and manipulation. By organizing the extracted data in a CSV file, the system ensures that it can efficiently retrieve and compare facial features during the recognition process, facilitating accurate and reliable identification of individuals.

3.3.3 Image Processing and Face Recognition (“face recognize.py”)

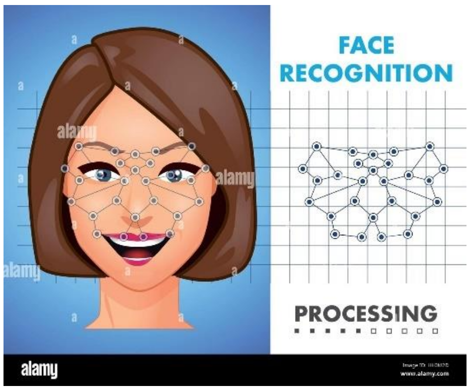


Figure 25: Face Recognition

The face_recognize.py script utilizes several key technologies and libraries to perform face detection and recognition. It employs OpenCV for general image processing tasks and dlib for specialized functions such as CNN-based face detection and recognition. The script uses three primary models: the mmod_human_face_detector, a pre-trained dlib CNN model for detecting faces; the shape_predictor_68_face_landmarks model for identifying key points on each detected face; and the dlib_face_recognition_resnet_model_v1 model for extracting facial features and generating unique descriptors for each face. These models work together to detect faces, identify landmarks, and create descriptors that are used for comparison and recognition.

Despite its effectiveness, the system has a notable limitation: it is unable to detect masked persons because the models selected rely heavily on facial landmarks that are typically obscured by masks. When masks cover significant portions of the face, such as the nose and mouth, the model's ability to identify and verify individuals is significantly reduced.

The script's functionality includes loading a face database from a CSV file containing known face descriptors and associated metadata (names and IDs), and performing detection and recognition by resizing images, detecting faces, checking confidence levels, extracting features using facial landmarks, and matching these against known descriptors for identification. Additionally, the script prompts the user to key in additional information such as the course code, subject name, instructor name, and any remarks. This information is then associated with the recognized faces and stored alongside the recognition results.

3.3.4 Image Pre-Processing

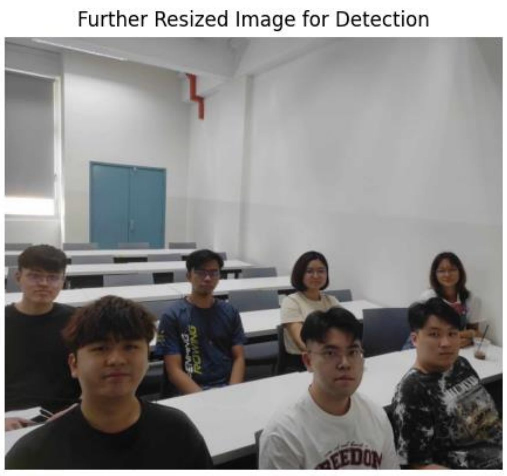
Before an image is fed into the face detection algorithm, it must undergo a series of image processing steps designated to enhance the effectiveness of detection by standardizing the image. The pre-processing steps used include:

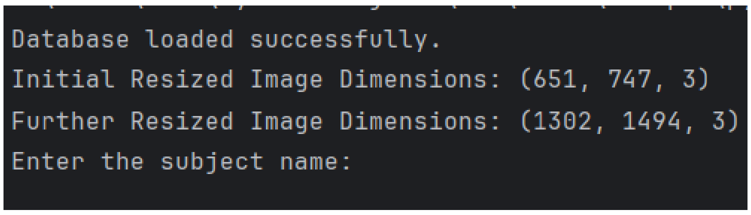
Steps	**Description**
Image Loading	➢ The image is read from the file system using OpenCV's "cv2.imread()" ➢ This step converts the image file into a format that can be manipulated by the OpenCV library ➢ It is the initial step that brings the image data into the processing pipeline
Initial Image Resizing	➢ The loaded image is resized into a smaller dimension using a custom "resize_image" function

	➢ This function scales the image by a specified percentage (50% in this case) ➢ Resizing images helps to standardize the input dimensions, and reduce computational load, making subsequent operations faster and more efficient
Further Resizing for Face Detection	➢ The resized image is further scaled by a factor of 2 ➢ This additional resizing is applied using OpenCV's "cv2.resize()" function ➢ This step improves the accuracy of face detection ➢ Scaling the image up helps in detecting smaller faces that might not be detected at lower resolutions

Table 6: Image pre-processing steps and description

The following images show the sequence and output of each step based on a given sample face image and the proof that has shown the dimension image has been increased with a similar appearance.





```
Database loaded successfully.
Initial Resized Image Dimensions: (651, 747, 3)
Further Resized Image Dimensions: (1302, 1494, 3)
Enter the subject name:
```

Figure 26: Image after pre-processing

3.3.5 Face Detection Algorithm

The face detection technique used in this project is based on a Convolutional Neural Network (CNN) model provided by the dlib library, utilizing the pre-trained model “mmod_human_face_detector.dat”. This CNN model is designed to detect human faces with high accuracy and robustness. The process begins by loading the CNN face detection model, using dlib’s “cnn_face_detection_model_v1” class. This model, specifically trained for detecting human faces, is recognized for its effectiveness in various conditions.

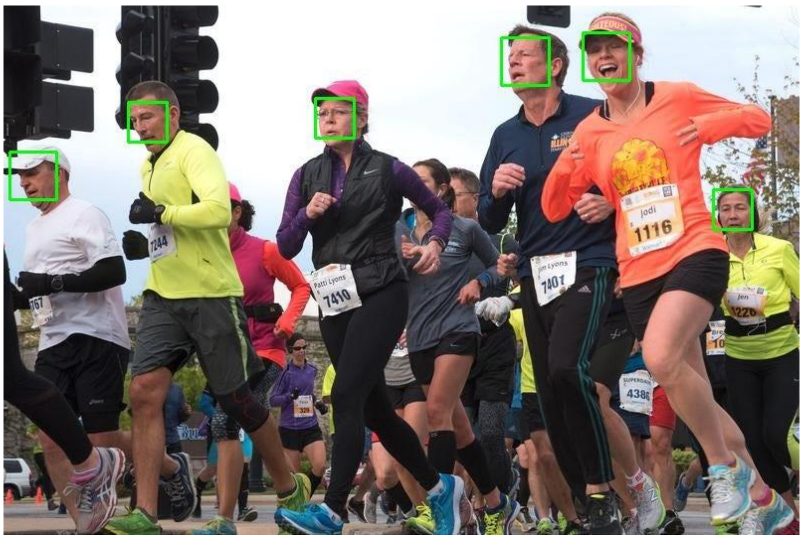


Figure 27: Example of CNN-based face detection using dlib

Before the image is processed by the detection algorithm, it undergoes a series of pre-processing steps to enhance detection accuracy. The initial step involves resizing the input image to standardize the dimensions and reduce computational load. The resized image, which is scaled by a factor of 2, is then passed to the CNN face detector. The detector returns a list of detected faces along with their confidence scores, where each face is represented by a rectangle indicating its coordinates within the image.

Detected faces are filtered based on their confidence scores, ensuring only faces with a confidence score above a certain threshold (0.5 in this case) are considered for further processing. This step is crucial in reducing the number of false positives. For each face that passes this confidence check, the script predicts 68 facial landmarks using another pre-trained dlib model, “shape_predictor_68_face_landmarks.dat”. These landmarks are key points on the face that are used for further processing.

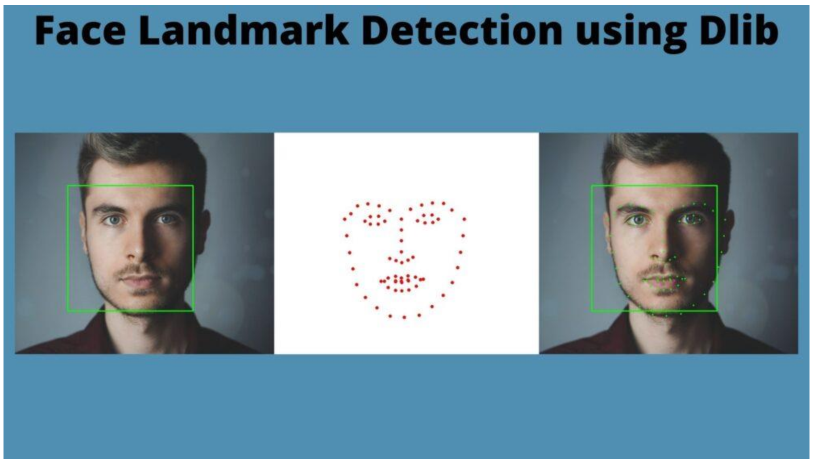


Figure 28: Example of detected facial landmarks on faces

The identified facial landmarks are then used to compute a 128-dimensional descriptor for each detected face using the dlib face recognition model,“dlib_face_recognition_resnet_model_v1”. This descriptor is a compact representation of the face's features and is essential for the subsequent face recognition process. The entire face detection process, from resizing the input image to computing the facial descriptors, ensures high accuracy and robustness in detecting and recognizing faces under various conditions.

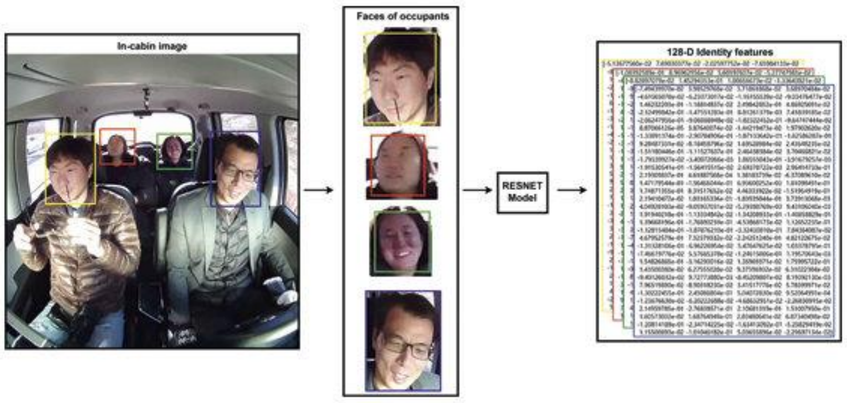


Figure 29: Example of visual representation of 128-dimensional descriptors computed for faces

3.3.6 Face Recognition Algorithm

Face recognition using dlib's CNN architecture involves a series of steps, starting with the utilization of a pre-trained CNN model provided by the dlib library. This CNN model is specifically designed for face detection and recognition tasks, renowned for its accuracy and robustness. In the context of face recognition, the primary objective is to leverage the learned features of this model to create meaningful embeddings for individual faces.

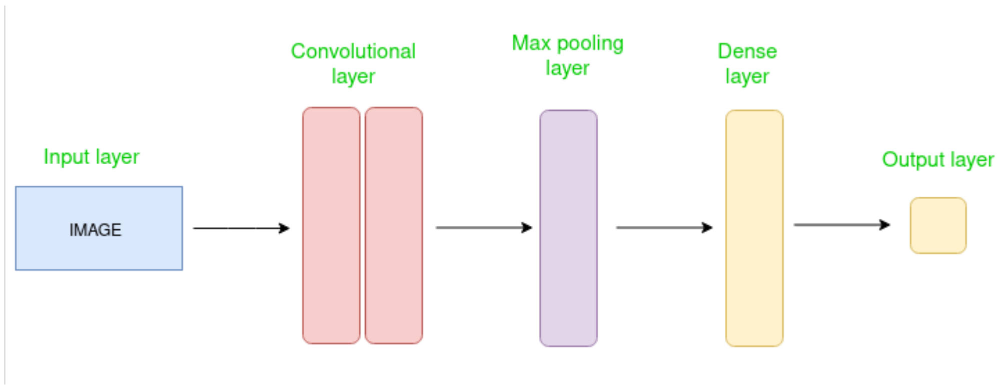


Figure 30: Simple architecture of CNN

The first step entails acquiring a pre-trained CNN model, mmod_human_face_detector.dat, typically available through the dlib library. This model is pre-trained on extensive face datasets, enabling it to capture high-level features from various facial characteristics. Once the CNN model is obtained, the face dataset undergoes meticulous pre-processing. This involves resizing each face image and ensuring the dataset is prepared for detection and recognition tasks. The subsequent phase involves feature extraction, where the pre-trained CNN model is employed to extract relevant features from the face images. The top layers of the CNN model usually consist of fully connected layers that are used for classification purposes.

An embedding layer is introduced to the CNN model to further refine the feature extraction process. This additional layer aids in reducing the dimensionality of the extracted features and fosters the creation of a meaningful representation of faces within a feature space. Afterward, the training phase follows during which the CNN model is trained on the prepared face dataset. A crucial aspect of this training is the selection of an appropriate loss function, with the triplet loss being a common choice for face recognition tasks. The triplet loss enforces that the distance between embeddings of faces belonging to the same person (positive pair) is minimized while the distance between embeddings of faces from different individuals (negative pairs) is maximized.

ResNet (Residual Network) is a type of deep neural network that addresses the problem of vanishing gradients in deep networks by introducing residual connections. These connections allow the model to learn residual functions regarding the input layer, enabling the training of very deep networks. ResNet architectures consist of multiple residual blocks, each containing convolutional layers, batch normalization, and ReLU activations. The key innovation is the shortcut connection that bypasses one or more layers, adding the input directly to the output of the stacked layers. This design helps in maintaining gradient flow, thus enabling the construction of extremely deep networks without performance degradation.

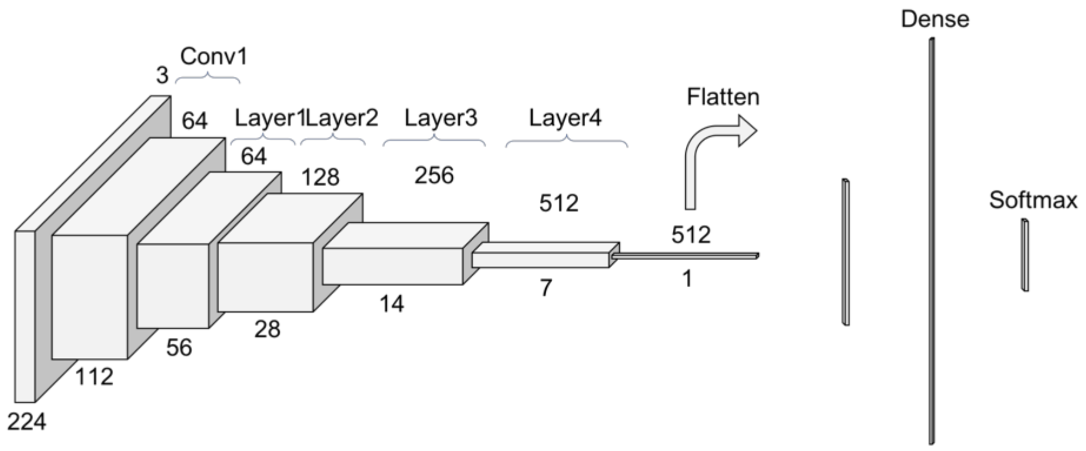
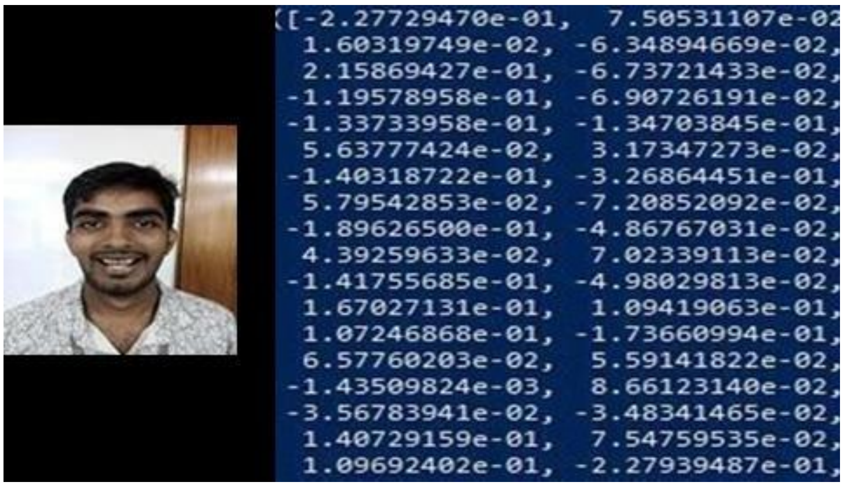


Figure 31: Visualization of ResNets

The dlib face recognition model, dlib_face_recognition_resnet_model_v1, uses a variant of the ResNet architecture to compute 128-dimensional descriptors for faces. These descriptors are compact feature representations that capture the unique characteristics of each face, making them suitable for recognition tasks.



```
([-2.27729470e-01,  7.50531107e-02
  1.60319749e-02, -6.34894669e-02,
  2.15869427e-01, -6.73721433e-02,
 -1.19578958e-01, -6.90726191e-02,
 -1.33733958e-01, -1.34703845e-01,
  5.63777424e-02,  3.17347273e-02,
 -1.40318722e-01, -3.26864451e-01,
  5.79542853e-02, -7.20852092e-02,
 -1.89626500e-01, -4.86767031e-02,
  4.39259633e-02,  7.02339113e-02,
 -1.41755685e-01, -4.98029813e-02,
  1.67027131e-01,  1.09419063e-01,
  1.07246868e-01, -1.73660994e-01,
  6.57760203e-02,  5.59141822e-02,
 -1.43509824e-03,  8.66123140e-02,
 -3.56783941e-02, -3.48341465e-02,
  1.40729159e-01,  7.54759535e-02,
  1.09692402e-01, -2.27939487e-01,
```

Figure 32: Example of 128 embeddings value for an image

Fine-tuning can optionally be performed on the model to adapt it to the specific characteristics of the target faces in the dataset. This fine-tuning process refines the model's parameters to enhance its performance on the given face recognition task. After that, a new face image is passed through the trained CNN-based model during face recognition inference to obtain its embedding in the learned feature space. The following step involves comparing the embedding of the input face with the embeddings of known faces in the dataset. Lastly, a threshold is set for similarity scores to make a final determination of whether the input face matches any known face in the dataset. If the similarity score surpasses the established threshold, the input face is recognized as belonging to a known individual.

3.3.7 Data Interfacing

Integrating data between the processing unit and the mobile device in the context of face recognition involves the mobile device's camera serving as the input mechanism, capturing facial images for subsequent processing on a laptop. The image captured by the smartphone is sent to the laptop, where it triggers the backend program to initiate the facial recognition process. The laptop, running a local server created using the Flask framework, handles the image processing tasks using Python and OpenCV libraries. This process includes detecting faces, extracting features, and recognizing individuals using pre-trained models from the dlib library. Once the image processing is completed, the recognition results are logged and saved.

A local area network (LAN) uses Wi-Fi technology to enable seamless wireless communication between the mobile device and the laptop. This setup allows both the smartphone and laptop to connect to the same Wi-Fi network, facilitating the transfer of data between them. The local server hosted on the laptop manages HTTP requests and routes the necessary data to and from the mobile device. This ensures that the facial recognition results can be effectively processed and stored on the laptop, enhancing the accuracy and efficiency of the system.

In summary, the system employs a smartphone for capturing images and a laptop for processing them, with a Flask server facilitating the communication between the devices over a Wi-Fi network. This configuration ensures efficient and accurate face recognition, with results that can be easily accessed and managed on the laptop.

4- Implementation and Testing

4.1 UML Diagram

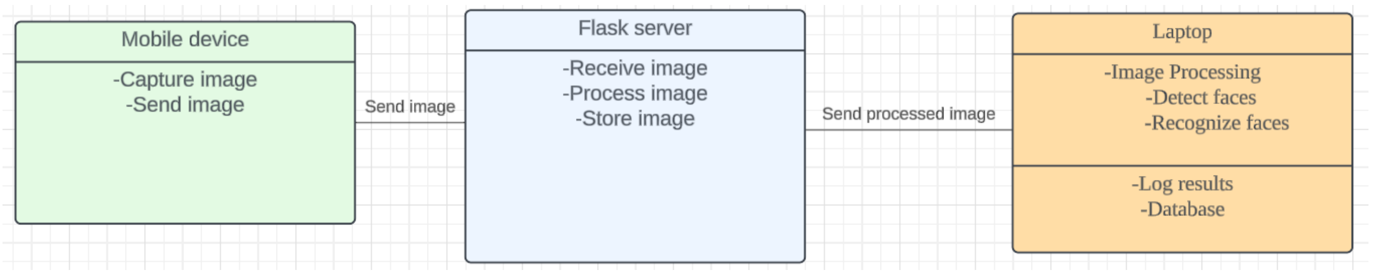


Figure 33: UML Diagram

The figure above shows the UML diagram of this project. The laptop serves as the processing unit of this system and is used for the implementation of image processing algorithms, acting as the interface between mobile devices and data storage. The mobile device's camera captures photos of individuals' faces and transmits them to the laptop through a local area network (LAN). Before capturing an image, the setup is adjusted to ensure optimal lighting and positioning for better recognition results.

The mobile device captures images and sends them to the Flask server over the LAN for further processing. This transmission is facilitated by a web interface that connects the mobile device to the Flask server. The Flask server receives the images and manages the uploaded images. The uploaded images are temporarily stored on the Flask server before being sent to the laptop for detailed analysis.

Upon receiving the processed images from the Flask server, the laptop runs specialized algorithms to detect faces within the images. This involves using pre-trained models to identify and locate faces accurately. The detected faces are then recognized by matching them with known faces stored in the database, a process that includes extracting features from the detected faces and comparing them against the database entries. The recognition results, including identified faces and their corresponding data, are logged and stored in the database for future reference, ensuring that attendance and identification records are maintained accurately.

In summary, the laptop serves as the central processing unit, implementing the image processing algorithms and acting as the interface between the mobile device and data storage.

The mobile device captures photos of faces and transmits them to the laptop through the Flask server using a LAN. The Flask server receives and processes the images, which are then analyzed on the laptop to detect and recognize faces. The results are logged in a database for future reference, providing a comprehensive understanding of the system's workflow as depicted in the UML diagram.

4.2 Flowchart

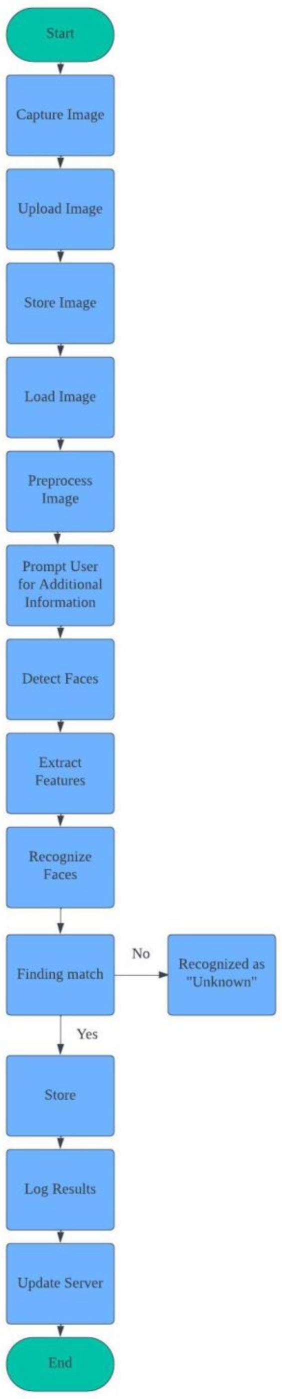


Figure 34: Flowchart of the overall system

The figure above illustrates the main function of the face recognition system. The workflow begins with the initialization of the system, preparing it to capture and process images. The first step involves the mobile device capturing an image of the students who are present. This image serves as the raw input for the system. Once captured, the image is uploaded to the Flask server, facilitating further processing on a more powerful machine (the laptop).

Upon receiving the image, the Flask server stores it in the image storage directory, ensuring its availability for subsequent operations. The stored image is then loaded into the server's memory for processing. In the pre-processing stage, the image undergoes several transformations: it is resized to a standard dimension for the face detection process.

Before starting face detection, the system prompts the user to enter additional information such as the course code, subject name, instructor name, and any remarks. This information is crucial for associating the recognized faces with the relevant metadata.

Next, the pre-processed image is passed through the feature extraction process using dlib. This involves identifying facial landmarks and computing face descriptors, which are unique representations of each face. The system detects faces within the image, identifying their locations using OpenCV and dlib. The face recognition engine then attempts to recognize the detected faces by comparing the extracted descriptors against a database of known faces, utilizing models like CNN and ResNet for matching.

At the decision point, the system checks if the detected face matches any face in the database. If a match is found, the face descriptors of the recognized faces are stored in the database, and the recognition results, including names and IDs, are logged. If no match is found, the faces are verified as unknown and it will not proceed to the storing process. Finally, the facial recognition module updates the new attendance data, showing which students have been recognized and logged into the file directory. The process completes with all recognized faces logged and the user interface updated, enhancing the accuracy and efficiency of attendance tracking in educational settings.

4.3 Hardware Implementation

In the implementation of this project, a Redmi Note 9 Pro serves as the primary device for capturing images of students upon their entry into the classroom. The smartphone operates on the Android 10 platform with MIUI 12, featuring a robust processing capability with its 2.32GHz octa-core configuration. The device boasts substantial storage capacity, offering 128GB of internal storage complemented by 6GB of RAM, providing ample space for efficient image storage and processing. Notably, the Redmi Note 9 Pro is equipped with an advanced camera system, featuring a 64 MP wide lens, an 8 MP ultrawide lens, a 5 MP macro lens, and a 2 MP depth sensor. This camera setup ensures high-quality image capture, enabling precise facial recognition. [30]

On the processing end, an Asus TUF F15 2022 model takes charge of executing the facial recognition program. Powered by the 12th Gen Intel® Core™ i5-12500H Processor clocked at 2.5 GHz, this laptop exhibits formidable computational capabilities. With a total of 12 cores, comprising 4 powerful P-cores and 8 efficient E-cores, the processor ensures swift and efficient handling of the facial recognition algorithms. The system is further enhanced by 8GB DDR4-3200 SO-DIMM memory, supporting dual-channel memory for optimal multitasking performance. Storage needs are met by a 512GB PCIe® 3.0 NVMe™ M.2 SSD, offering high-speed data access for seamless program execution.

Noteworthy is the inclusion of the NVIDIA® GeForce RTX™ 3050 Laptop GPU in the Asus TUF F15, operating at 1790MHz* with a 95W power profile (including a 50MHz OC and 15W Dynamic Boost). This dedicated GPU, with its 4GB GDDR6 memory, significantly contributes to the efficiency of the facial recognition program, ensuring accelerated image processing and accurate identification. The combination of a powerful smartphone and a high-performance laptop underscores the project's commitment to leveraging cutting-edge technology for effective and reliable facial recognition in the classroom environment.[31]

Figure 35: Redmi Note 9 Pro

Figure 36: ASUS TUF Gaming F15 2022

4.4 Software Implementation

Figure 37: Python logo

Python, a high-level programming language, served as the foundation for developing the face recognition system. Leveraging its high-level characteristics, Python provides a user-friendly environment with enhanced code readability and less stringent code formatting compared to languages like C or C++. This quality contributes to reduced costs associated with program maintenance. Furthermore, Python's extensive standard library facilitates code modularity and reusability, fostering versatility in applications, including interfaces for operating systems and machine learning within the realm of face recognition.

Figure 38: OpenCV logo

OpenCV, aptly named for "Open Source Computer Vision," stands as a versatile open-source library extensively utilized in the realm of face recognition, visual image processing, and machine learning systems. This library boasts compatibility with four programming languages, namely C++, Python, Java, and MATLAB, across various platforms. Equipped with over 2500 optimized algorithms, OpenCV plays a pivotal role in enabling advanced face recognition algorithms. The implementation of image processing techniques, readily available in OpenCV, encompasses a spectrum of functionalities, including gray scaling, thresholding, and image feature extraction, crucial for tasks such as contour determination for precise face localization. An exhaustive online documentation resource provides comprehensive insights into the syntax and functions of these face recognition algorithms, serving as a valuable reference and guide for developers and researchers alike.

Figure 39: Flask logo

Flask is a lightweight web framework for Python, designed to simplify the development of web applications. It provides developers with the tools to create robust and scalable web applications quickly and efficiently. Flask follows a minimalistic approach, allowing developers to use only the components they need while maintaining the flexibility to extend and customize the application as required. In the context of the face recognition system, Flask serves as the backbone for managing image uploads, processing images, and providing a user-friendly interface. It acts as the intermediary between the client devices (such as mobile devices) and the processing unit (laptop), facilitating seamless communication and data transfer.

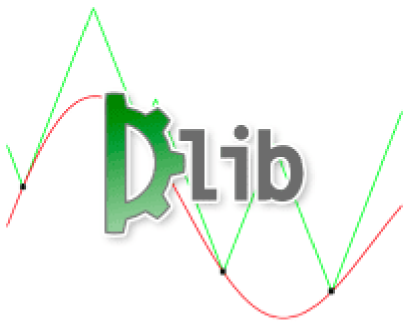


Figure 40: Dlib logo

Dlib is a toolkit containing machine learning algorithms and tools for creating complex software, and it is particularly well-suited for tasks related to face detection and face recognition. The library includes several pre-trained models that are specifically designed to enhance accuracy and robustness in these applications. One such model is the cnn_face_detection_model_v1, a pre-trained CNN model that enables the system to efficiently identify faces in images. Another essential model provided by dlib is the shape_predictor_68_face_landmarks, which predicts 68 facial landmarks, crucial for precise face localization and alignment. Additionally, the face_recognition_model_v1 computes face descriptors, which are essential for recognizing and distinguishing between different faces. The functionality offered by dlib, combined with its ease of integration into Python applications, makes it an invaluable tool for implementing advanced face recognition systems.

Figure 41: Numpy logo

NumPy is a fundamental package for scientific computing with Python, widely used for array and matrix operations essential in image processing and machine learning tasks. In the face recognition system, NumPy is utilized to handle and manipulate large datasets of image data and facial descriptors efficiently. It provides support for multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays, making it a cornerstone library for numerical computations in Python.

Figure 42: Pandas logo

Pandas is a data manipulation and analysis library used to handle and process data in CSV files. In the context of the face recognition system, Pandas is employed to load and manage known face descriptors and associated metadata, such as names and IDs. This library simplifies the process of reading, writing, and manipulating structured data, enabling efficient data handling and preprocessing tasks required for accurate face recognition.

Figure 43: Scipy logo

scipy is a library used for scientific and technical computing. In this script, it is used specifically for computing the cosine distance between face descriptors, a crucial step in the face recognition process. The library offers a range of functions for mathematical algorithms and statistical operations, making it a vital tool for implementing advanced scientific computations in Python.

Figure 44: Tkinter logo

Tkinter is a standard GUI (Graphical User Interface) toolkit in Python. It is used here to create a simple file dialog for selecting image files. Tkinter provides a fast and easy way to create GUI applications, allowing users to interact with the face recognition system through a graphical interface. This integration enhances the usability of the system by providing a user-friendly method for selecting and processing images.

Figure 45: Python Time module

The time module provides various time-related functions, used in this script to measure the processing time of the operations. By recording the start and end times of the face recognition process, it allows for the calculation of the total time taken to complete the task, which is useful for performance evaluation and optimization.

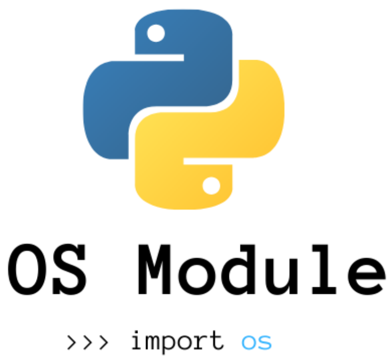


Figure 46: Python OS module

The os module provides a way of using operating system-dependent functionality, such as handling file paths and saving uploaded images. In the face recognition system, the os module is used to manage file operations, ensuring that images are correctly saved and accessed during the processing workflow. This module’s functionality is crucial for maintaining an organized file structure and handling file-related tasks efficiently.

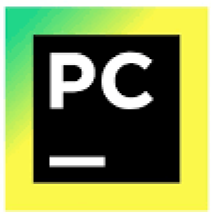


Figure 47: JetBrains Pycharm logo

PyCharm plays a pivotal role in the development of face recognition systems by providing a robust and versatile integrated development environment (IDE). As a feature-rich code editor, PyCharm streamlines the writing and editing of code for face recognition algorithms, offering functionalities such as syntax highlighting, auto-completion, and linting to enhance code correctness and efficiency. Its integrated version control, particularly through Git support, facilitates collaborative development in large-scale face recognition projects. With a vast array of extensions, developers can augment the IDE's functionality to include features related to Python development, machine learning frameworks, and Git integration, aligning the environment with the specific needs of face recognition projects. The IDE's powerful debugging tools, task automation capabilities, and an integrated terminal contribute to efficient troubleshooting and automation of repetitive tasks, critical for ensuring the accuracy and reliability of face recognition algorithms. Additionally, PyCharm provides a seamless interactive development environment for Python, making it well-suited for face recognition systems implemented in this language. Its cross-platform compatibility ensures that developers can work seamlessly across different operating systems, fostering flexibility in the development process. In summary, PyCharm serves as a comprehensive and adaptable tool that significantly contributes to the development, testing, and maintenance of the software aspects of face recognition applications.

4.5 Testing

In this section, we examine the performance of the Convolutional Neural Network (CNN) model for face detection and recognition across various distances between the camera and the subject. Evaluating the model's effectiveness at different distances is crucial for applications that involve varying proximity. By testing the model under these conditions, we aim to understand its robustness and reliability in real-world scenarios.

Additionally, we assess the model's accuracy and processing time duration when detecting and recognizing faces with different numbers of people. This evaluation helps determine the scalability of the model and its efficiency in handling images with varying complexity and crowd density. To ensure consistency and control over the testing variables, all tests are conducted under the same conditions except for 2-meter tests. The images are captured at the same height, from the same angle, in the same room, and using the same group of students as subjects for 3-meter tests to 5-meter tests. This controlled setup helps to isolate the impact of distance and the number of people on the model's performance, providing more accurate and reliable results. The parameters set to test the accuracy and processing time duration of the face recognition system are the different distances from the camera and the different numbers of students in the images.

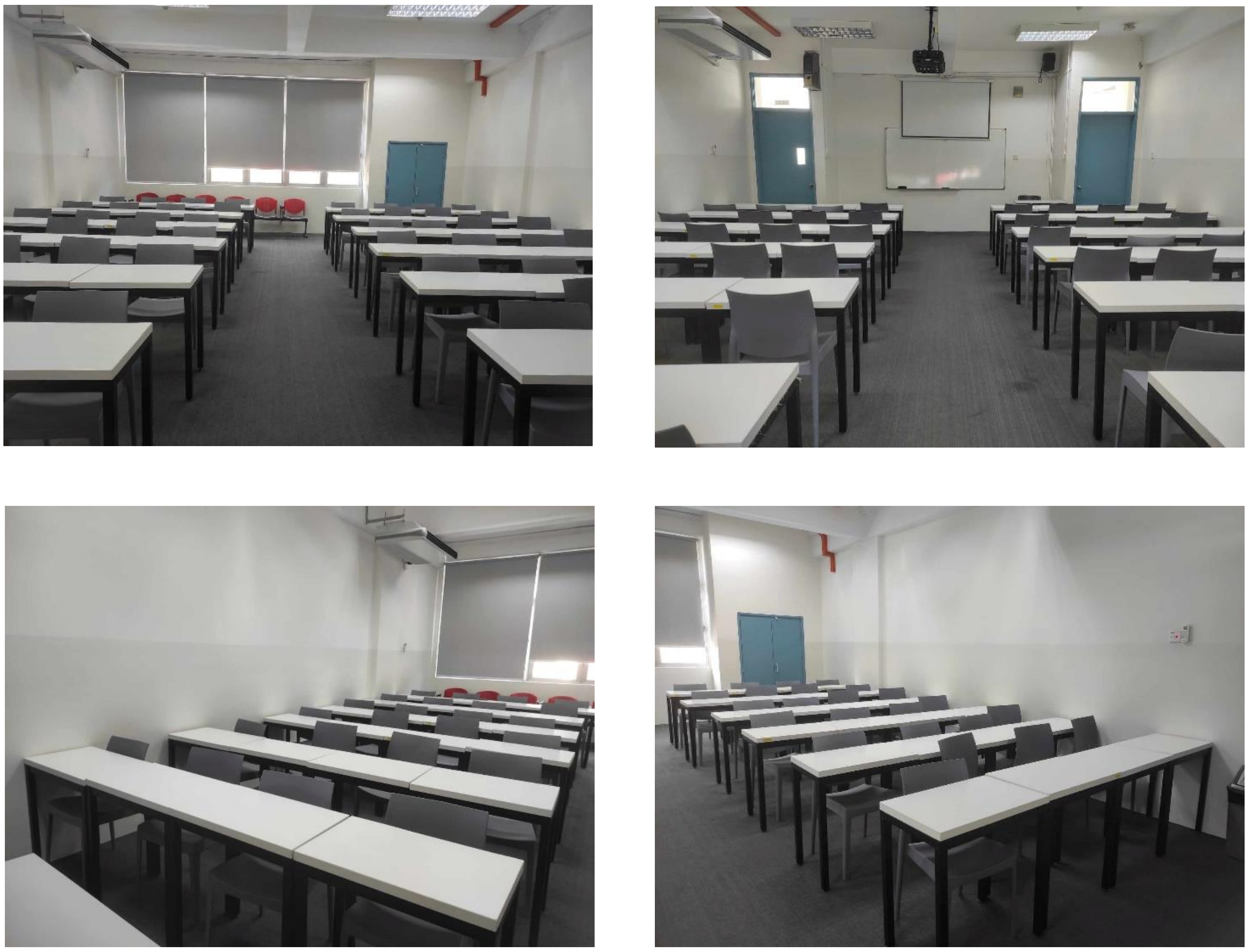


Figure 48: Classroom used for testing

4.5.1 Face Detection and Recognition with Different Distances

Each face detection and recognition effectiveness are dependent on the distance from which the image was taken. To test the effective distance for face detection and recognition, images of faces are taken starting from 2 meters up to 5 meters. Four samples are taken at each distance to ensure the accuracy and reliability of the data, with each row in the setup representing a 1-meter increment. The tests are conducted under controlled conditions, maintaining the same height, same angle, same room, and the same group of students for each distance. It is worth noting that the performance at 2 meters is particularly good, which suggests that slight variations at this distance do not significantly affect the overall results. Sample images taken at distances of 2 meters to 5 meters are shown in the table below:

Distance (m)	Sample image
2	
3	

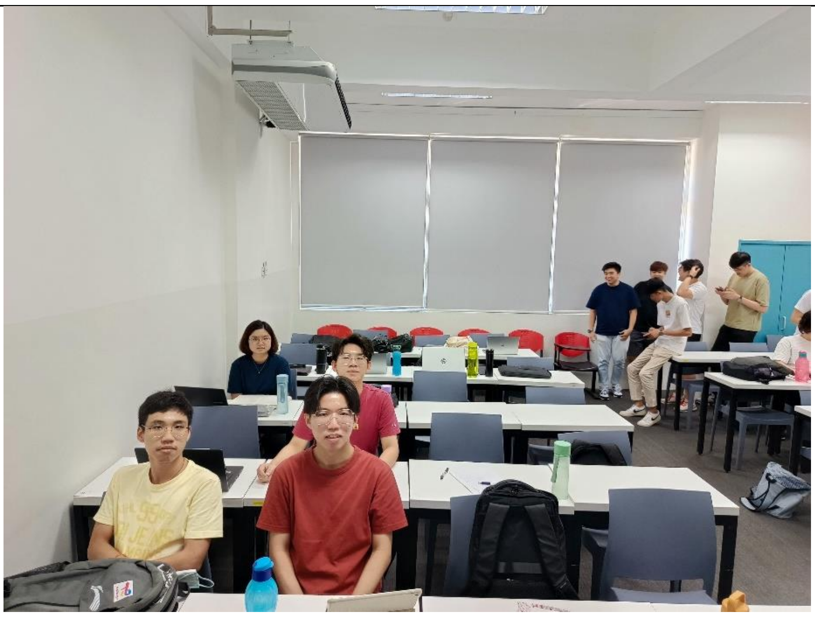
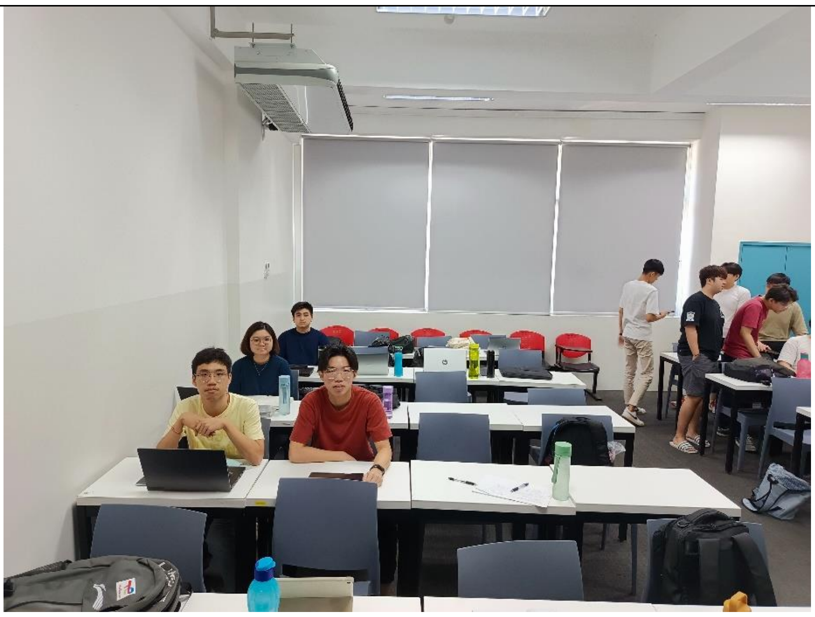
4	
5	

Table 7: Sample images for distance test

Next, the figure below demonstrates the classroom setup for testing face detection and recognition at various distances. Each row represents a distance from the camera, starting from 1 meter to 5 meters. The labels indicate the distance from the camera to the row where the subjects will be seated while the other figure shows the measurement of the distance to the fifth row, which is considered as 5 meters, confirming it is approximately 4.90 meters. This ensures accurate distance labeling for the testing setup.

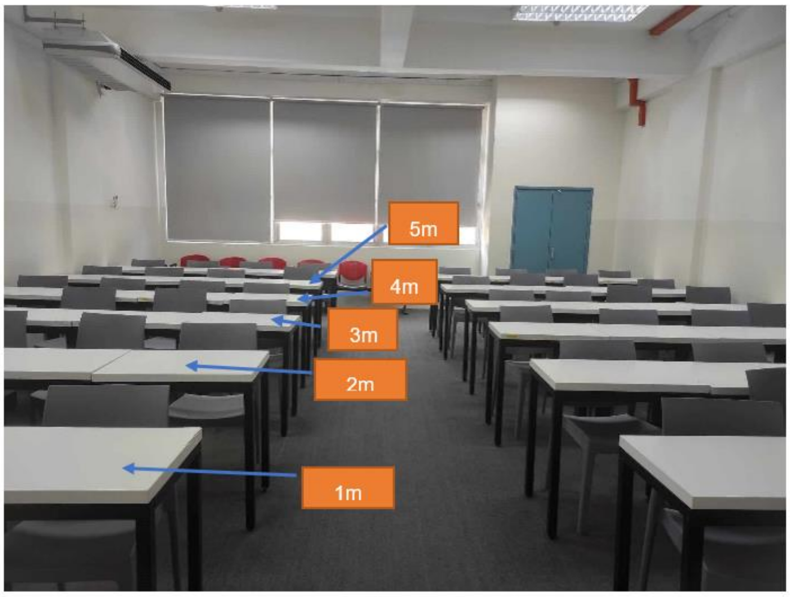


Figure 49: Classroom distance setup

Figure 50: Example of distance measurement

4.5.2 Face Detection and Recognition with Different Numbers of Students

The accuracy and processing time of the face detection and recognition system depend on the number of people present in the image. To investigate this, images containing different numbers of students, ranging from 4 to 10 individuals, were captured. Four images were taken for each group size to ensure data accuracy and reliability. All tests were performed under controlled conditions, maintaining consistent height, angle, room, and using the same group of students. Below are examples of images with varying numbers of students:

Number of students captured	Sample image
4	
6	

8	
10	

Table 8: Sample images for number of students test

4.5.3 Face Detection and Recognition Results

The following images show the results of face detection and recognition at different distances:

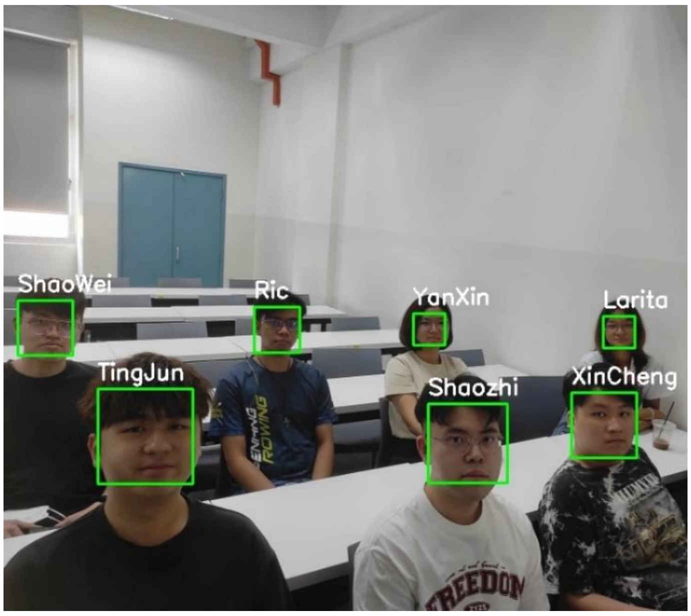


Figure 51: Sample output image for 2m

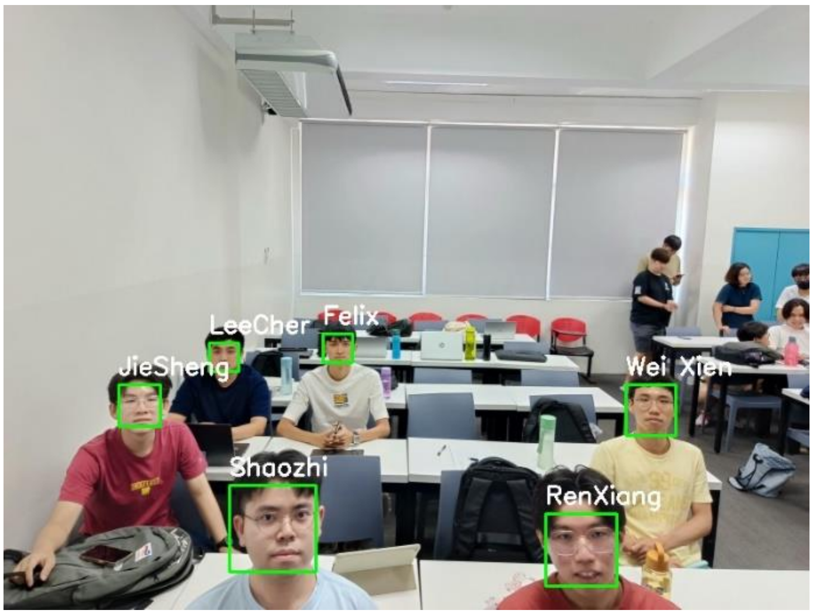


Figure 52: Sample output image for 3m

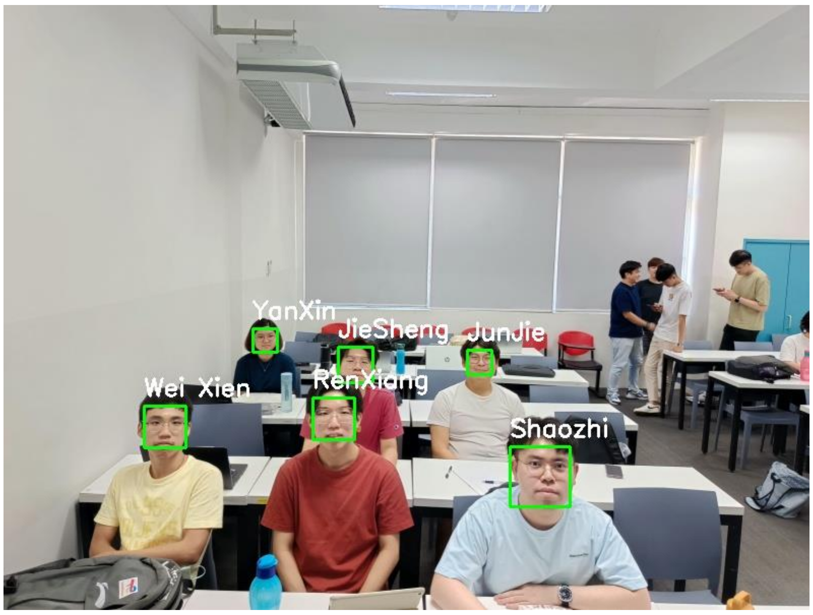


Figure 53: Sample output image for 4m

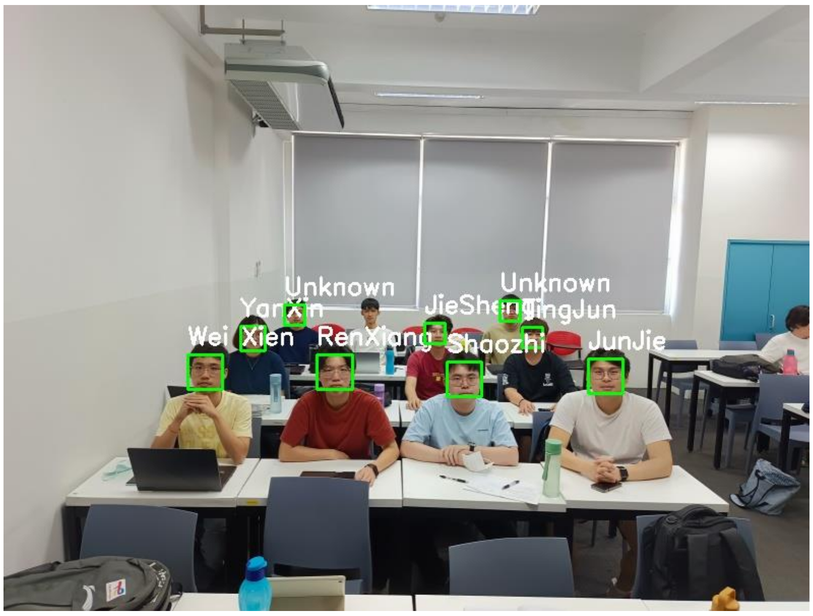


Figure 54: Sample output image for 5m

Next, the following images show the results of face detection and recognition with different numbers of students:

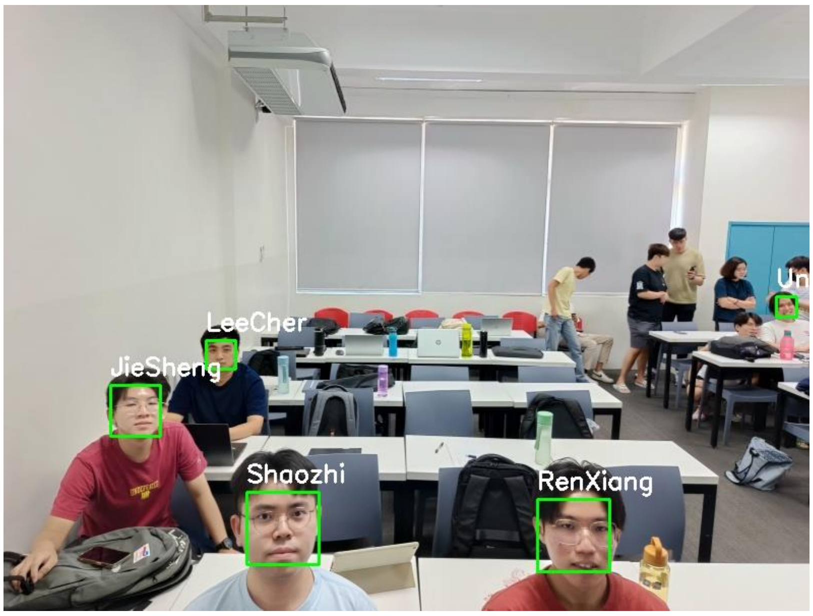


Figure 55: Sample output image for 4 students

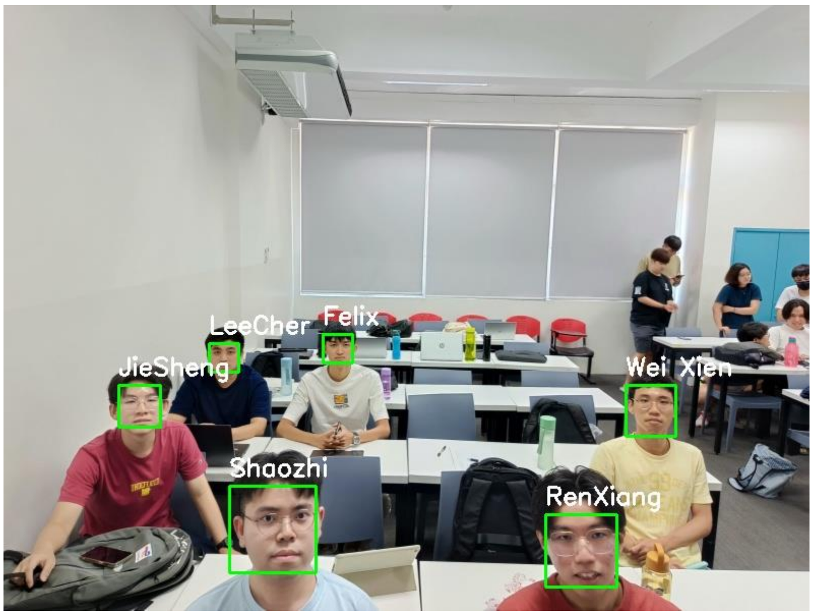


Figure 56: Sample output image for 6 students

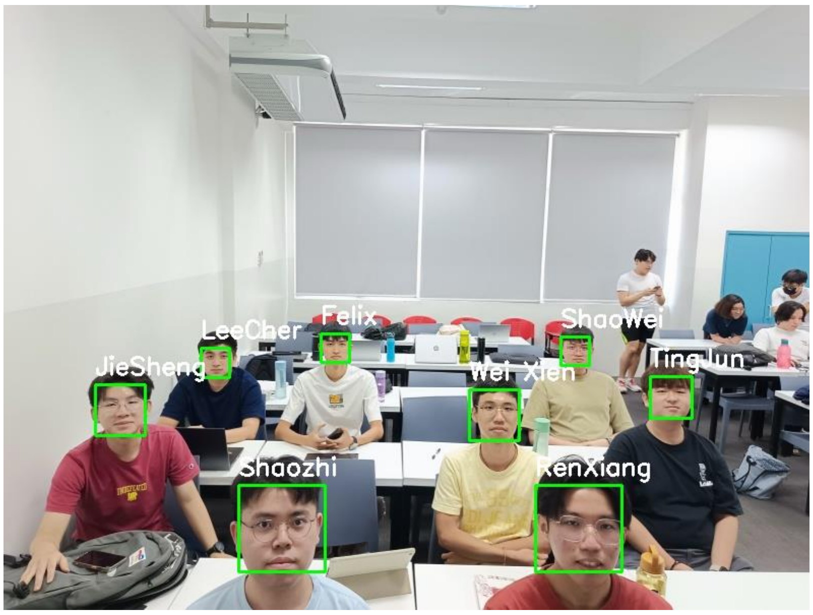


Figure 57: Sample output image for 8 students

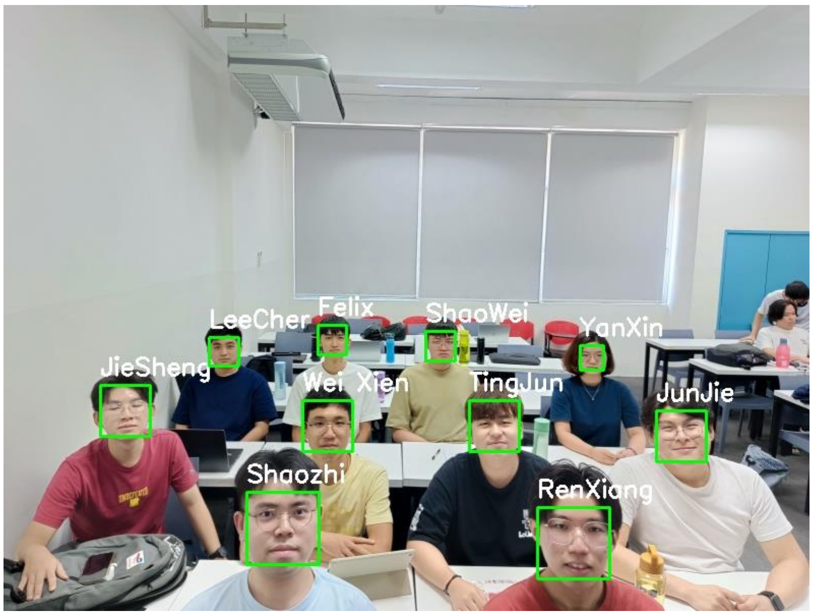
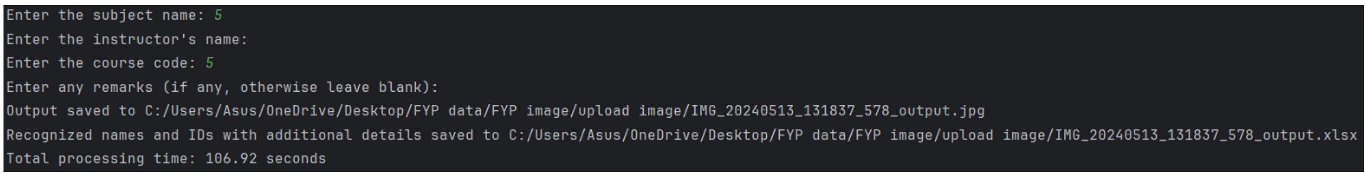


Figure 58: Sample output image for 10 students

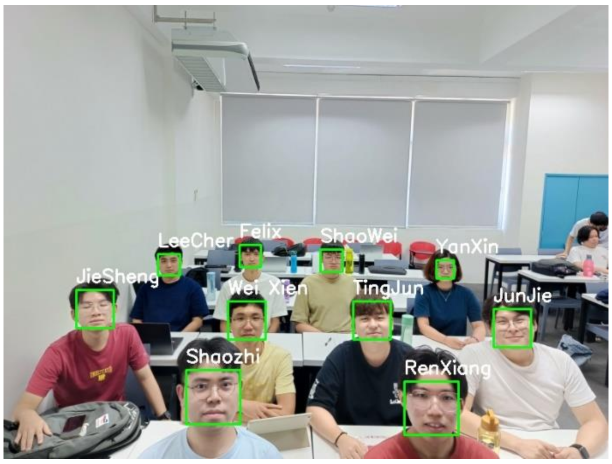
Before the face detection and recognition process, the system prompts the user to enter additional information such as the subject name, instructor name, course code, and any remarks. This information is then associated with the recognized faces and saved in the output files.

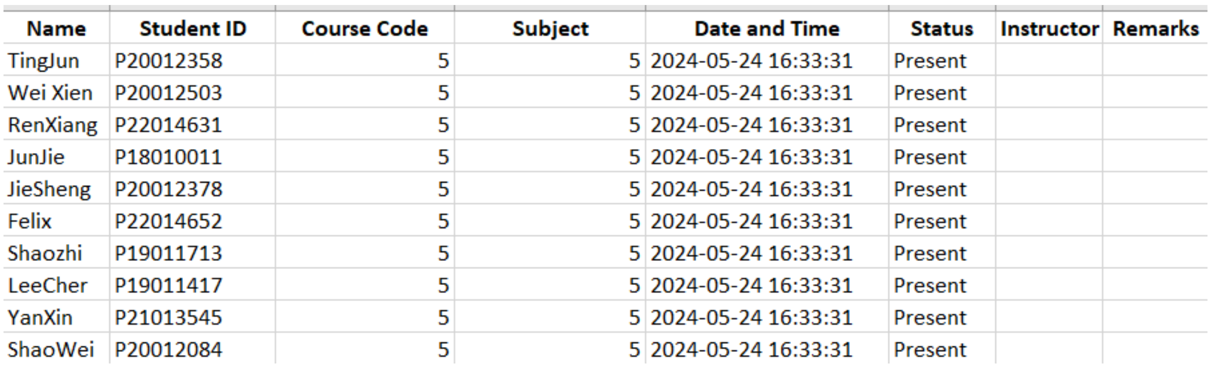


```
Enter the subject name: 5
Enter the instructor's name:
Enter the course code: 5
Enter any remarks (if any, otherwise leave blank):
Output saved to C:/Users/Asus/OneDrive/Desktop/FYP data/FYP image/upload image/IMG_20240513_131837_578_output.jpg
Recognized names and IDs with additional details saved to C:/Users/Asus/OneDrive/Desktop/FYP data/FYP image/upload image/IMG_20240513_131837_578_output.xlsx
Total processing time: 106.92 seconds
```

Figure 59: Sample user prompt output image for 10 students

Apart from that, the figure above shows the console output where the user is prompted to enter the subject name, instructor name, course code, and remarks. After processing, the system saves the output image and an Excel file containing the recognition results and additional details. The processed image with recognized faces is saved to a specified directory while ten recognized names, IDs, and additional information are saved to an Excel file for record-keeping.





Name	Student ID	Course Code	Subject	Date and Time	Status	Instructor	Remarks
TingJun	P20012358	5	5	2024-05-24 16:33:31	Present		
Wei Xien	P20012503	5	5	2024-05-24 16:33:31	Present		
RenXiang	P22014631	5	5	2024-05-24 16:33:31	Present		
JunJie	P18010011	5	5	2024-05-24 16:33:31	Present		
JieSheng	P20012378	5	5	2024-05-24 16:33:31	Present		
Felix	P22014652	5	5	2024-05-24 16:33:31	Present		
Shaozhi	P19011713	5	5	2024-05-24 16:33:31	Present		
LeeCher	P19011417	5	5	2024-05-24 16:33:31	Present		
YanXin	P21013545	5	5	2024-05-24 16:33:31	Present		
ShaoWei	P20012084	5	5	2024-05-24 16:33:31	Present		

Figure 60: Output image and generated Excel file from the sample of 10 students

4.6 User Interface

4.6.1 Mobile Device Interface

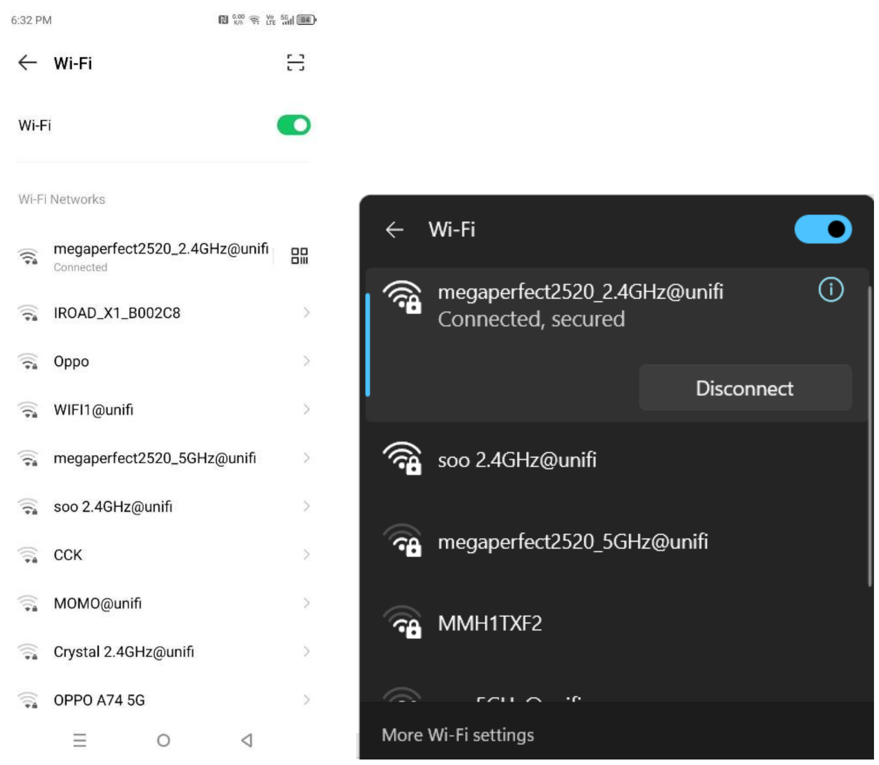
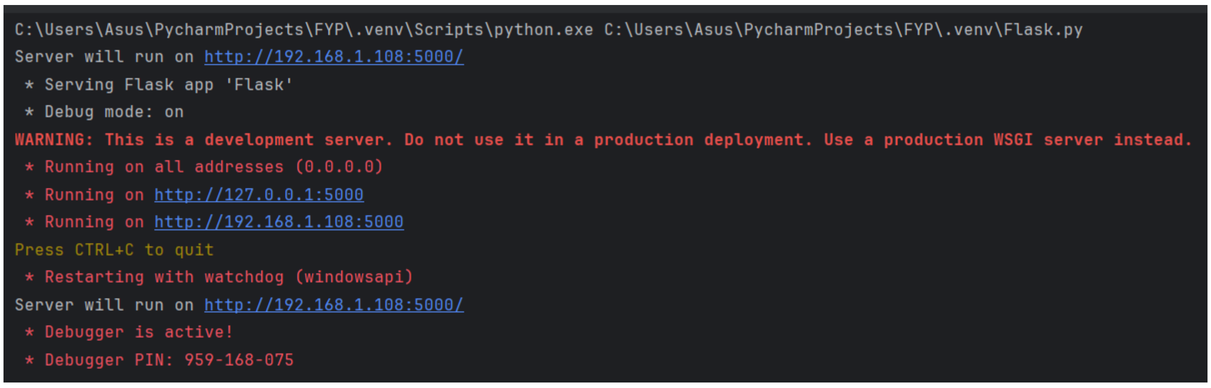


Figure 61: Wi-Fi connection in laptop and mobile device

Before connecting the mobile device with the laptop, both devices must connect to the same Wi-Fi as shown in the figure above. The Flask server needs to be running to handle incoming requests from the mobile device.



```
C:\Users\Asus\PycharmProjects\FYP\.venv\Scripts\python.exe C:\Users\Asus\PycharmProjects\FYP\.venv\Flask.py
Server will run on http://192.168.1.108:5000/
 * Serving Flask app 'Flask'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.108:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
Server will run on http://192.168.1.108:5000/
 * Debugger is active!
 * Debugger PIN: 959-168-075
```

Figure 62: Starting the Flask server

The Flask server can be started from the “Flask.py” script and it will show output as shown in the figure above. Ensure that the server is running and listening for requests.

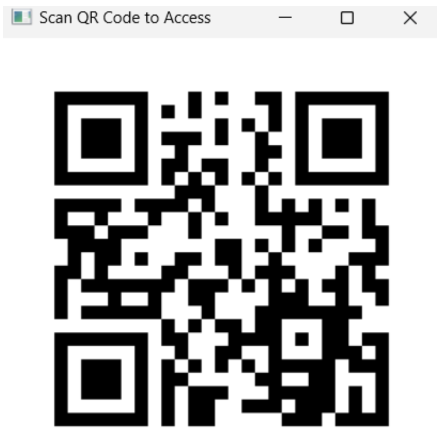


Figure 63: QR code for IP address

The IP address of the laptop is generated in a QR code format. The mobile device can scan the QR code to access the webpage directly, as shown in the figure above.

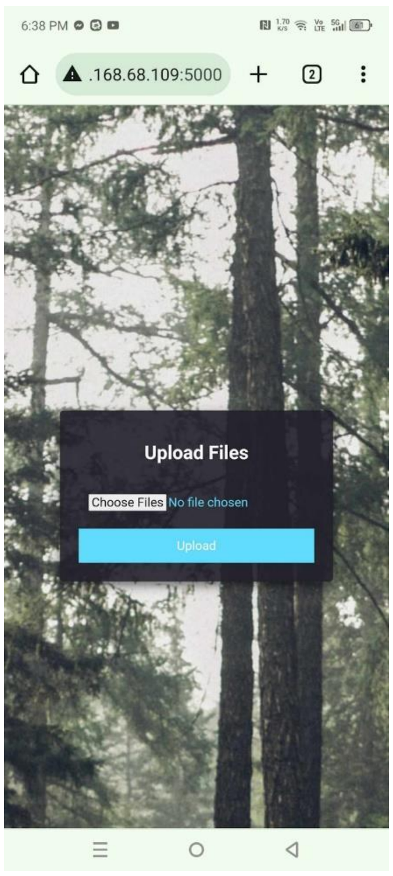


Figure 64: Homepage design (mobile view)

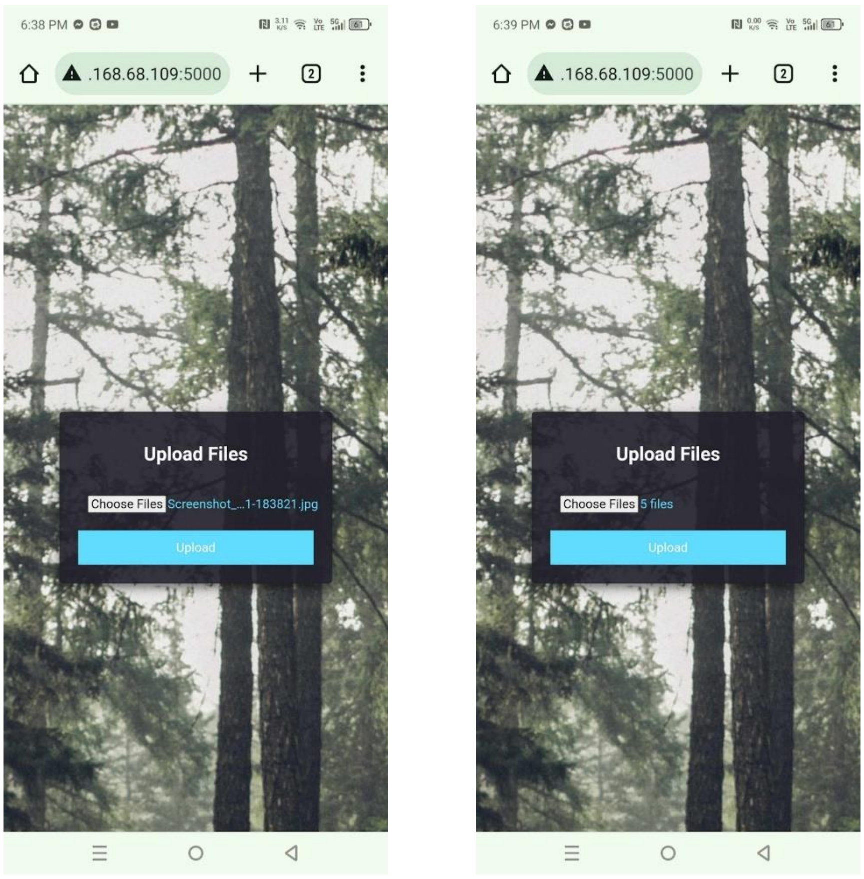
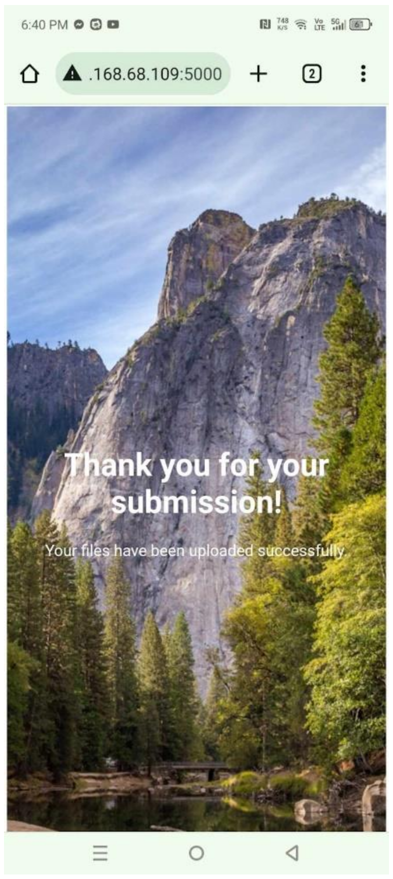


Figure 65: Application page with single or multiple selection (mobile view)

After entering the webpage, the layout is shown in the figure above. The user is required to select the “choose file” button to choose the image(s) they want to process and press the “upload” button to send the image(s) to the laptop. The user can upload both single and multiple files for processing. After receiving the image(s) from the mobile device, the laptop will execute the face recognition program and save the output recognition result to a specified directory.



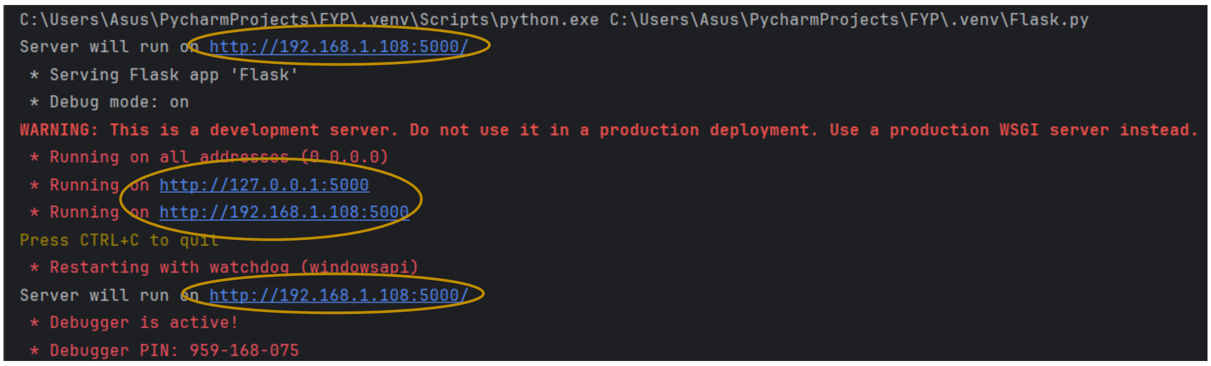
Thank you for your submission!

Your files have been uploaded successfully.

Figure 66: Output displayed on webpage (mobile view)

Once the upload is complete, the webpage will display a message saying "Thank you for your submission" as shown in the figure above.

4.6.2 Laptop Interface



```
C:\Users\Asus\PycharmProjects\FYP\.venv\Scripts\python.exe C:\Users\Asus\PycharmProjects\FYP\.venv\Flask.py
Server will run on http://192.168.1.108:5000/
 * Serving Flask app 'Flask'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.108:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
Server will run on http://192.168.1.108:5000/
 * Debugger is active!
 * Debugger PIN: 959-168-075
```

Figure 67: Output of Flask server

The figure above shows the output of the Flask server running on the laptop. The IP address displayed (circled up in yellow) can be clicked or entered in the web browser on the laptop to access the same webpage hosted by the Flask server which is shown below. This allows both the laptop and mobile device to access the web application for face detection and recognition, ensuring seamless interaction and processing.

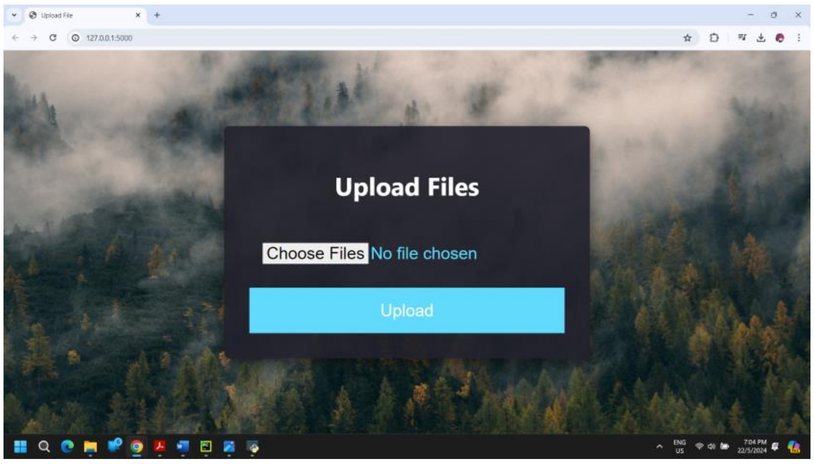


Figure 68: Homepage design (laptop view)

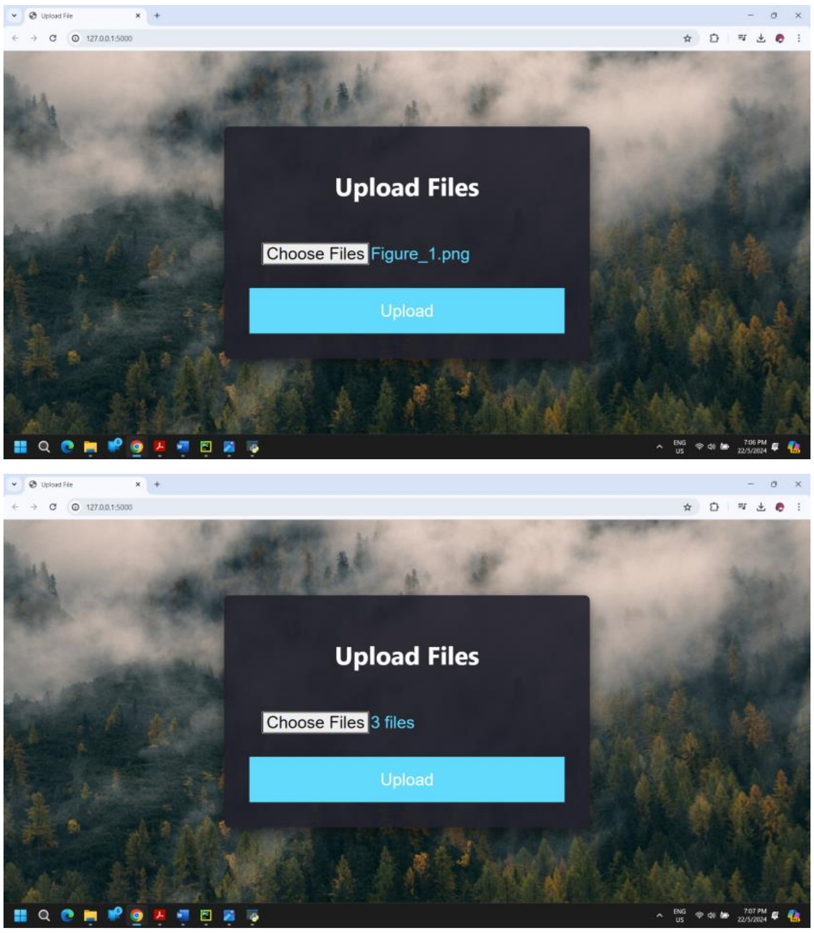
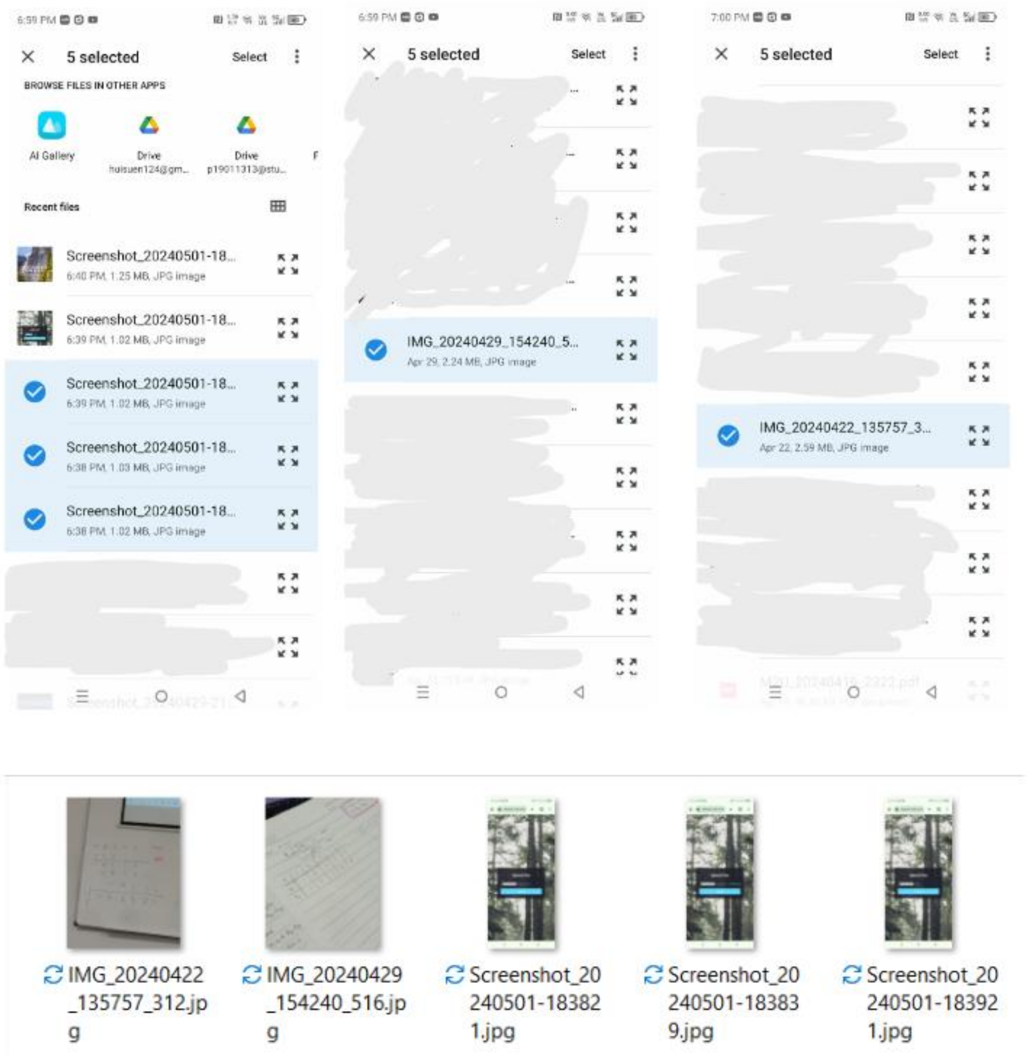


Figure 69: Application page with single or multiple selection (laptop view)

The webpage and function of the webpage are the same as the mobile view which is shown in the figure above.


6:59 PM

5 selected Select

BROWSE FILES IN OTHER APPS

AI Gallery · Drive huisuen124@gm... · Drive p19011313@stu...

Recent files

Screenshot_20240501-18...
6:40 PM, 1.25 MB, JPG image

Screenshot_20240501-18...
6:39 PM, 1.02 MB, JPG image

✓ Screenshot_20240501-18...
6:39 PM, 1.02 MB, JPG image

✓ Screenshot_20240501-18...
6:38 PM, 1.03 MB, JPG image

✓ Screenshot_20240501-18...
6:38 PM, 1.02 MB, JPG image

[redacted]

6:59 PM

5 selected Select

[redacted]

✓ IMG_20240429_154240_5...
Apr 29, 2.24 MB, JPG image

[redacted]

7:00 PM

5 selected Select

[redacted]

✓ IMG_20240422_135757_3...
Apr 22, 2.59 MB, JPG image

[redacted]

IMG_20240422_135757_312.jpg IMG_20240429_154240_516.jpg Screenshot_20240501-183821.jpg Screenshot_20240501-183839.jpg Screenshot_20240501-183921.jpg

Figure 70: Proof that all images selected have been saved in the folder of laptop

All the images sent to the laptop have been saved in a folder so that they can be accessed on the laptop anytime which is proved by the figure above.

5- Results and Discussion

5.1 Results and Analysis

5.1.1 Distance Test

The summary of the system performance at a distance range of 2m to 5m is shown in the table below:

Distance (m)	Accuracy (%)
2	100.00
3	94.64
4	90.48
5	88.65
Average accuracy = 93.44%	

Table 9: Accuracy table with sample test images according to distances

The accuracy according to distance and the average accuracy are calculated as follows:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

$$\begin{aligned}&where\\ &\quad Number\ of\ Correct\ Predictions = True\ Positives\ (faces\ correctly\ recognize),\\ &\quad Total\ Number\ of\ Prediction\\ &\quad = True\ Positives + False\ Positives + False\ Negatives\ (total\ faces\ recognized)\end{aligned}$$

$$Average\ accuracy = \frac{Total\ accuracy}{Total\ Number\ of\ Distances\ Tested}$$

The system demonstrates effective face recognition capabilities within a range of 2 to 4 meters, achieving a success rate of at least 90%. This observation highlights a decline in accuracy as the distance increases, as illustrated in the table for objective 1, which presents the relationship between distances and accuracy. Several technical factors contribute to this decline.

As the distance between the camera and the faces increases, the number of pixels representing each face decreases. This reduction in pixel density leads to a significant loss of detail, making it more challenging for face recognition algorithms to accurately identify distinguishing features such as the contours of facial landmarks, skin texture, and subtle expressions. Additionally, the decline in resolution at greater distances affects the clarity of captured images. Lower resolution means that the finer details are not as discernible, which can reduce the effectiveness of the facial feature extraction process. As a result, the computed descriptors used for recognition are less precise, increasing the likelihood of errors.

5.1.2 Number of Students Test

The following table summarizes the detection and recognition capabilities under different groups of students that are captured in a single frame for objective 2. The number of students is grouped by 4 settings starting from 4 students, 6 students, 8 students, and 10 students.

Number of students	Processing time(s)	Accuracy (%)
4	101.93	97.92
6	104.65	95.74
8	105.37	93.18
10	106.94	90.33

Table 10: Performance metrics of facial recognition system by number of students

The processing time and accuracy according to number of students are calculated as follows:

$$Processing\ time = \frac{Total\ Processing\ Time\ per\ Image\ of\ Number\ of\ Students}{Total\ Image\ of\ Number\ of\ Students}$$

$$Accuracy = \frac{Total\ Accuracy\ per\ Image\ of\ Number\ of\ Students}{Total\ Image\ of\ Number\ of\ Students}$$

From the table, it is observed that as the number of individuals in the frame increases from 4 to 10, the processing time increases from 101.93 seconds to 106.94 seconds, and the accuracy decreases from 97.92% to 90.33%. This decline in accuracy can be attributed to resolution decline, where more faces in the frame mean each face occupies fewer pixels, leading to a loss of detail and making it harder for the algorithm to accurately identify distinguishing features. Additionally, the complexity of accurately matching each face to the correct identity in the database increases, leading to a higher chance of errors. More individuals in the frame can also introduce variability and background noise, complicating the recognition process.

These performance metrics indicate that while the system performs well with a smaller number of individuals, its efficiency and accuracy decrease as the number of individuals increases. This highlights the need for continued refinement in feature extraction and algorithm tuning to improve scalability and robustness. Optimizing hardware resources and enhancing algorithm efficiency can help achieve consistent accuracies closer to the ideal target of 100%.

5.1.3 Processing Time Test

The summary of the relationship between distance and processing time required for facial recognition is shown in the table below:

Distance (m)	Processing time (s)
2	104.94
3	106.74
4	107.00
5	109.74
Average processing time = 107.11s	

Table 11: Relationship between distance and processing time

The processing time and average processing time according to distances are calculated as follows:

$$Processing\ time = \frac{Total\ Processing\ Time\ per\ Image\ of\ Distance}{Total\ Image\ of\ Distance}$$

$$Average\ Processing\ Time = \frac{Total\ Processing\ Time}{Total\ Number\ of\ Distances\ Tested}$$

The table illustrates the relationship between distance and processing time required for facial recognition. As the distance from the camera increases from 2 meters to 5 meters, there is a noticeable increase in processing time. At a distance of 2 meters, the processing time is 104.94 seconds. This time increases progressively with distance, reaching 106.74 seconds at 3 meters, 107.00 seconds at 4 meters, and 109.74 seconds at 5 meters. The average processing time across these distances is calculated to be 107.11 seconds.

This trend suggests that as the distance between the camera and the subjects increases, the system requires more time to process and recognize faces. This increase in processing time can be attributed to several factors, including the decrease in resolution and detail of the faces captured at greater distances, which necessitates more computational effort to accurately identify distinguishing features. Additionally, the increased complexity of matching faces against the database and potential variability in background noise at longer distances also contribute to the longer processing times. Overall, these results highlight the need for optimizing the system's performance to maintain efficiency and accuracy across varying distances.

5.2 Discussion

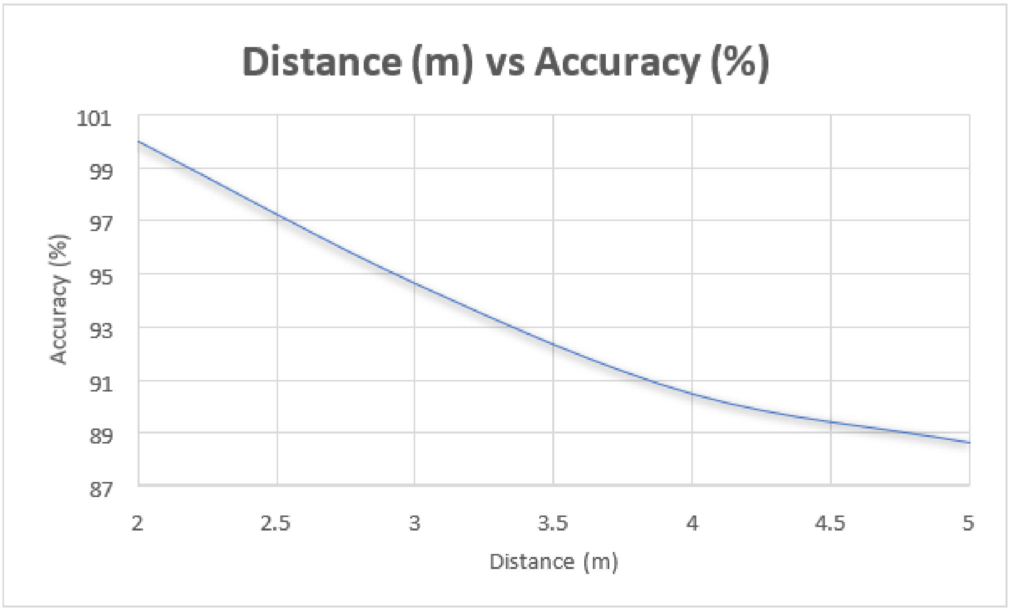


Figure 71: Graph of distance vs accuracy

The graph illustrates the relationship between distance and accuracy in the facial recognition system. As the distance from the camera increases, there is a clear and steady decline in accuracy. At a distance of 2 meters, the system achieves nearly perfect accuracy at approximately 100%. This high accuracy indicates that the system performs exceptionally well when faces are captured within close proximity, where the resolution and detail of the facial features are highest. As the distance increases to 3 meters, the accuracy drops to around 95%. This decline, although slight, marks the beginning of the system's decreased effectiveness due to the increased distance. The resolution begins to decrease, resulting in less detailed facial features for the recognition algorithm to process accurately. Further increasing the distance to 4 meters results in a more pronounced drop in accuracy to approximately 91%. At this distance, the resolution and detail of the faces are significantly lower, leading to a more challenging recognition process. The system struggles more to extract and match facial features accurately, which is reflected in the reduced accuracy. At 5 meters, the accuracy declines to around 89%. This continued decline underscores the impact of increased distance on the system's performance. At greater distances, not only is the resolution lower, but other factors such as increased background noise and variability in lighting conditions also come into play, further complicating the recognition task.

The overall trend depicted in the graph highlights a critical challenge in facial recognition systems: maintaining high accuracy at greater distances. The decrease in accuracy with increasing distance can be attributed to the reduction in the number of pixels representing each face, leading to a loss of detail. Additionally, the increased complexity of distinguishing between faces and matching them accurately in the database adds to the decline in performance.

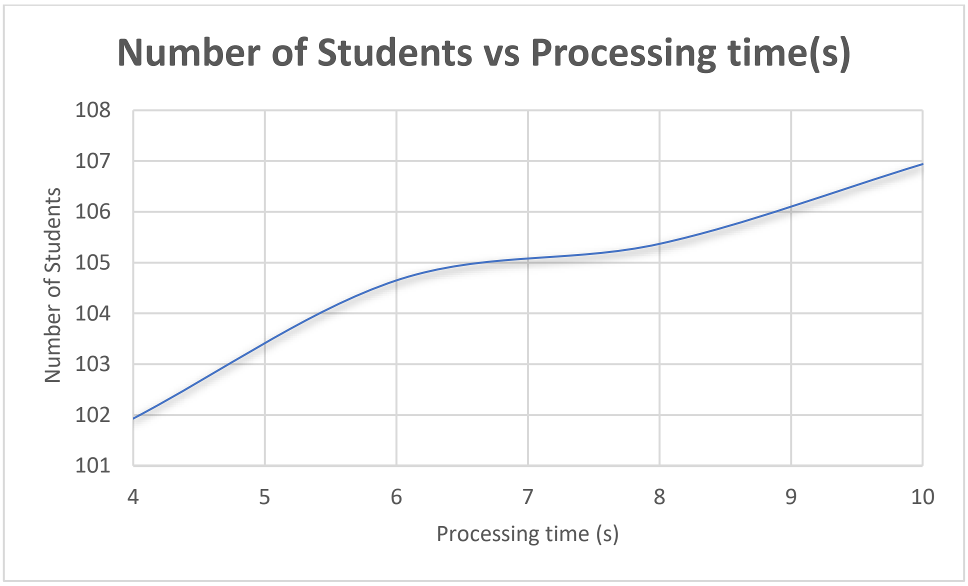


Figure 72: Graph of the number of students captured vs processing time

The graph illustrates the relationship between the number of students in the frame and the processing time required by the facial recognition system. As the number of students increases, there is a noticeable increase in processing time. When there are 4 students in the frame, the processing time is approximately 101.93 seconds. This serves as the baseline, reflecting the system's performance with a relatively small number of faces to process. As the number of students increases to 6, the processing time rises to about 104.65 seconds. This increase can be attributed to the additional computational load required to process and distinguish between more faces. The system must analyze more data, which naturally extends the processing time. With 8 students, the processing time further increases to approximately 105.37 seconds. The trend indicates that the system continues to require more time as the number of faces increases. This is due to the complexity involved in accurately identifying and matching each face against the database. When the number of students reaches 10, the processing time peaks at around

106.94 seconds. At this point, the system is handling a significantly larger volume of data, resulting in the longest processing time recorded. The increased number of faces in the frame requires more intensive computation to maintain accuracy in facial recognition.

The overall trend depicted in the graph demonstrates a clear correlation between the number of individuals and processing time. As more students are added to the frame, the processing time increases steadily. This is indicative of the system's computational limitations and the scalability challenges associated with processing multiple faces simultaneously.

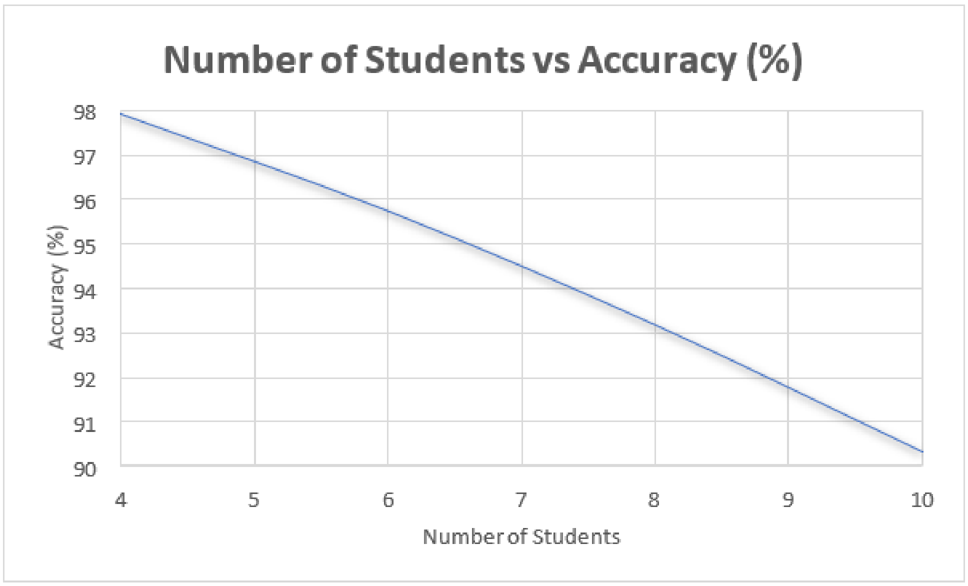


Figure 73: Graph of the number of students captured vs accuracy

The graph illustrates the relationship between the number of students in the frame and the accuracy of the facial recognition system. As the number of students increases, there is a clear and steady decline in accuracy. When there are 4 students in the frame, the system achieves an accuracy of approximately 97.92%. This high accuracy reflects the system's effectiveness when dealing with a smaller number of faces, where it can dedicate more computational resources to accurately identifying each individual. As the number of students increases to 6, the accuracy drops to about 95.74%. This decline, although moderate, indicates that the system begins to face challenges in maintaining high accuracy as the number of faces increases. The additional faces introduce more complexity in distinguishing and matching each face correctly. With 8 students in the frame, the accuracy further decreases to approximately 93.18%. At this point, the system has to manage even more faces, which reduces the available computational resources per face. This leads to a greater chance of recognition errors. When the number of students reaches 10, the accuracy falls to around 90.33%. This significant drop highlights the system's difficulty in handling a large number of faces simultaneously. The increased volume of data and the complexity of accurately identifying each face resulted in a noticeable decline in performance.

The overall trend depicted in the graph shows a clear inverse relationship between the number of individuals and the accuracy of the facial recognition system. As more students are added to the frame, the accuracy steadily decreases. This is indicative of the system's limitations in processing multiple faces with high precision.

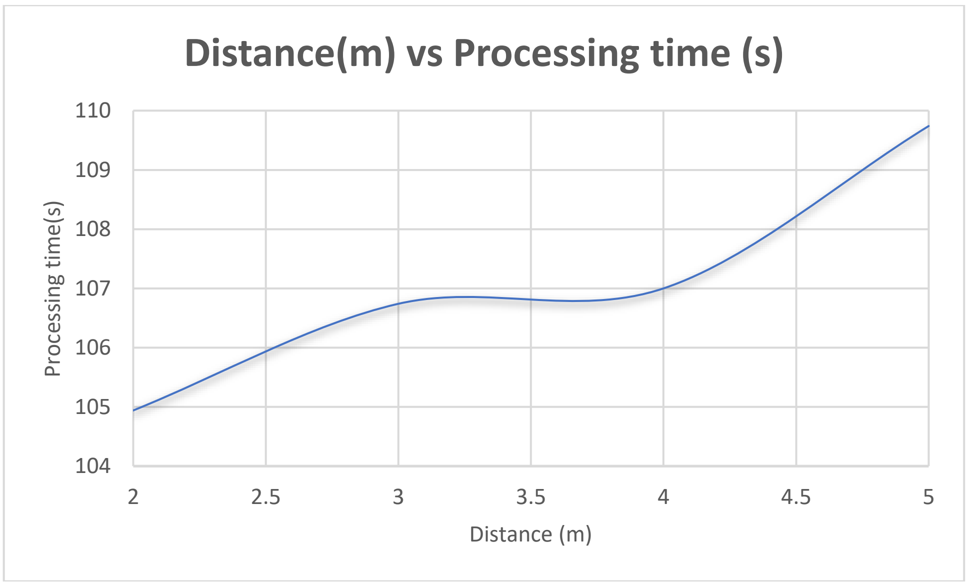


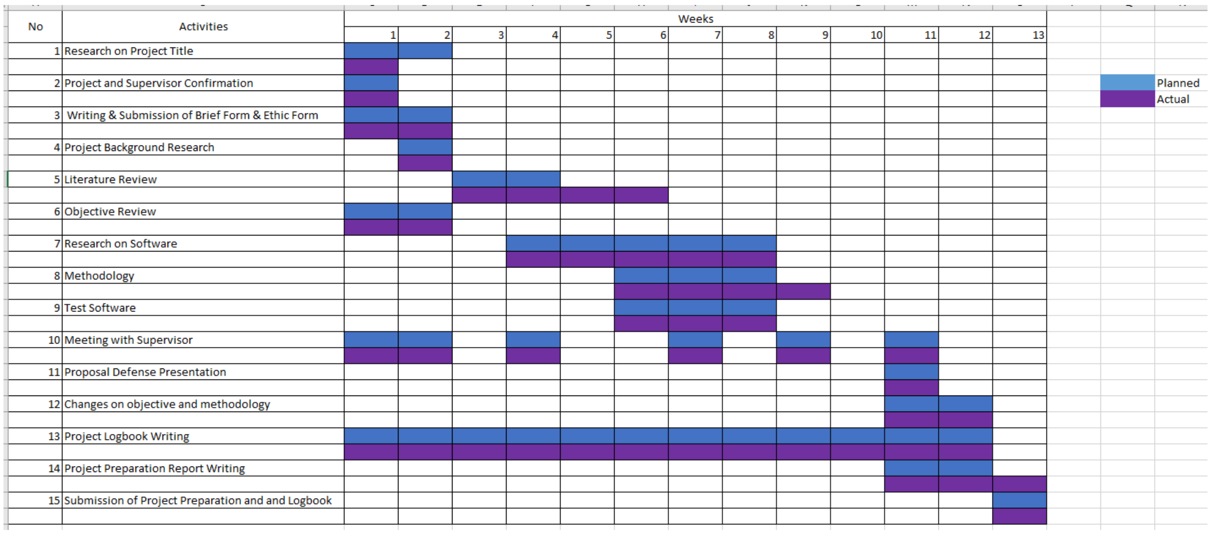
Figure 74: Graph of distance vs processing time

The graph illustrates the relationship between the distance from the camera and the processing time required by the facial recognition system. As the distance increases, there is a noticeable increase in processing time, albeit with some variability. At a distance of 2 meters, the processing time is approximately 104.94 seconds. This serves as the baseline, reflecting the system's performance when faces are captured at proximity, where the resolution and detail of facial features are highest. As the distance increases to 3 meters, the processing time rises to about 106.74 seconds. This increase can be attributed to the additional computational effort required to process faces captured at a slightly lower resolution, where the system needs to work harder to maintain accuracy. With a further increase in distance to 4 meters, the processing time experiences a slight dip to approximately 107.00 seconds. The near-plateauing of processing time at this distance suggests that the system's performance might be stabilizing temporarily despite the increased distance. However, the overall trend indicates that processing times continue to rise. When the distance reaches 5 meters, the processing time peaks at around 109.74 seconds. This significant increase highlights the system's challenges in processing faces captured at greater distances. At this point, the system is dealing with lower-resolution images, which require more intensive computation to accurately extract and match facial features.

The overall trend depicted in the graph shows that as the distance between the camera and the subjects increases, the processing time required by the facial recognition system also increases. This is indicative of the system's computational limitations and the additional effort needed to maintain accuracy at greater distances.

6- Project Management

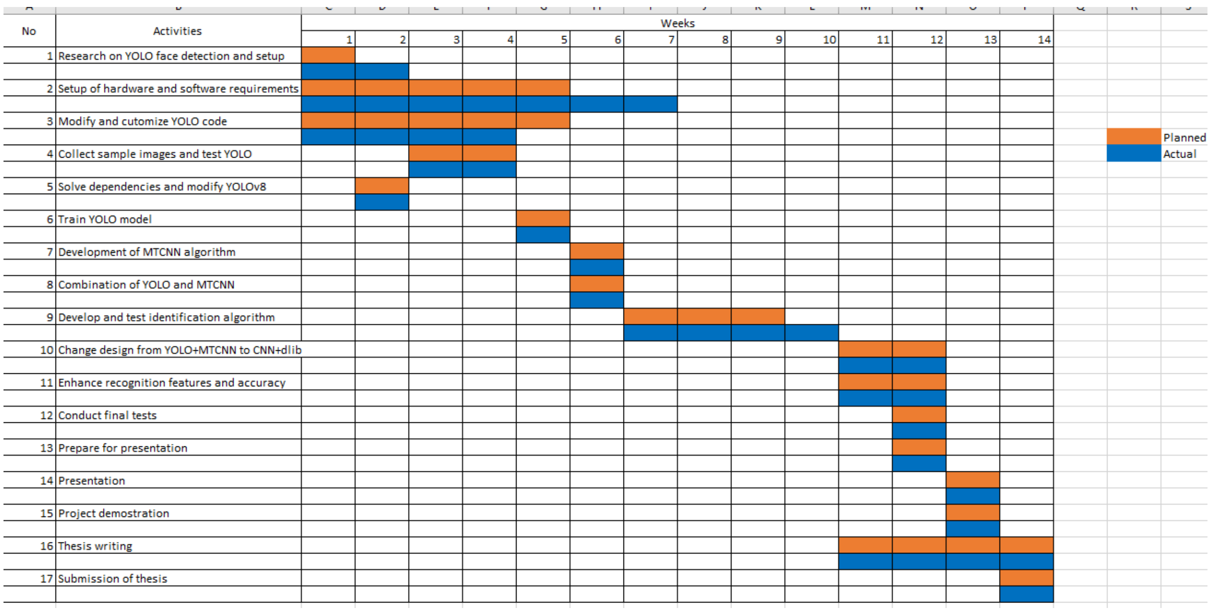
A Gantt chart is utilized to outline the project plan and oversee the project's progression by continuously monitoring its progress. The chart enumerates the planned activities and the estimated time required for each. The status of the tasks is indicated on the chart and updated every week.



No	Activities	1	2	3	4	5	6	7	8	9	10	11	12	13
1	Research on Project Title	🟦	🟦											
		🟪												
2	Project and Supervisor Confirmation	🟦												
		🟪												
3	Writing & Submission of Brief Form & Ethic Form	🟦	🟦											
		🟪	🟪											
4	Project Background Research		🟦											
			🟪											
5	Literature Review			🟦	🟦									
				🟪	🟪	🟪	🟪							
6	Objective Review	🟦	🟦											
		🟪	🟪											
7	Research on Software				🟦	🟦	🟦	🟦	🟦					
					🟪	🟪	🟪	🟪	🟪					
8	Methodology						🟦	🟦	🟦					
							🟪	🟪	🟪	🟪				
9	Test Software						🟦	🟦	🟦					
							🟪	🟪	🟪					
10	Meeting with Supervisor	🟦	🟦		🟦			🟦		🟦		🟦		
		🟪	🟪		🟪			🟪		🟪				
11	Proposal Defense Presentation											🟦		
												🟪		
12	Changes on objective and methodology											🟦	🟦	
												🟪	🟪	
13	Project Logbook Writing	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	
		🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	🟪	
14	Project Preparation Report Writing											🟦	🟦	
												🟪	🟪	🟪
15	Submission of Project Preparation and and Logbook													🟦
														🟪

🟦 Planned
🟪 Actual

Figure 75: Gannt chart of the project preparation



No	Activities	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Research on YOLO face detection and setup	🟧													
		🟦	🟦												
2	Setup of hardware and software requirements	🟧	🟧	🟧	🟧	🟧									
		🟦	🟦	🟦	🟦	🟦	🟦	🟦							
3	Modify and cutomize YOLO code	🟧	🟧	🟧	🟧	🟧									
		🟦	🟦	🟦	🟦										
4	Collect sample images and test YOLO			🟧	🟧										
				🟦	🟦										
5	Solve dependencies and modify YOLOv8		🟧												
			🟦												
6	Train YOLO model					🟧									
						🟦									
7	Development of MTCNN algorithm						🟧								
							🟦								
8	Combination of YOLO and MTCNN						🟧								
							🟦								
9	Develop and test identification algorithm							🟧	🟧	🟧					
								🟦	🟦	🟦	🟦				
10	Change design from YOLO+MTCNN to CNN+dlib											🟧	🟧		
												🟦	🟦		
11	Enhance recognition features and accuracy											🟧	🟧		
												🟦	🟦		
12	Conduct final tests												🟧		
													🟦		
13	Prepare for presentation												🟧		
													🟦		
14	Presentation													🟧	
														🟦	
15	Project demostration													🟧	
														🟦	
16	Thesis writing											🟧	🟧	🟧	🟧
												🟦	🟦	🟦	🟦
17	Submission of thesis														🟧
															🟦

🟧 Planned
🟦 Actual

Figure 76: Gannt chart of the project realisation

During the first session, the focus was on planning and conducting research. Key activities included researching the project title, confirming the project and supervisor, writing and submitting brief and ethics forms, and conducting background research. The planned and actual progress were closely aligned, with minor deviations in some tasks.

The second session primarily concentrated on building the model, writing the program, testing the system, analyzing data, and writing the thesis. Significant tasks included research on YOLO face detection and setup, which was completed on time, and the setup of hardware and software requirements, which experienced delays due to software changes affecting subsequent tasks. Modifications and customizations to the YOLO code were made to enhance performance, and sample images were collected and tested as planned. Dependency issues with YOLOv8 were resolved, allowing for continued modifications and training of the YOLO model. The development of the MTCNN algorithm was initially pursued but later revised to use CNN and dlib for improved accuracy. This change was implemented after accuracy tests showed better results with CNN and dlib. Throughout this session, continuous improvements were made to enhance recognition features and accuracy, culminating in final tests to ensure the system met project objectives. Preparations for the project presentation were completed on time, and the project was successfully demonstrated. The thesis writing was completed, and the final submission was made as scheduled.

Throughout both sessions, documentation processes, including brief form preparation, logbook updates, data collection, and report and thesis writing, were consistently carried out. Despite some delays caused by the extended time required for software implementation due to a lack of image processing knowledge, most tasks were completed on time. Overall, the project was completed within the planned timeline.

7- Conclusion

Throughout this project, both the face detection process and face recognition process have been accomplished using Python programming language and the dlib image processing library along with the OpenCV library as well. The results obtained from all experiments have been recorded and analyzed.

In short, the aim of the project was achieved by completing the medium-range facial recognition system with an average accuracy of 93.77%. The effects of distance and the number of faces in the frame on the accuracy and processing time duration of the system have been explored. Overall, the system functions well most of the time, except under extreme conditions such as images captured in very low brightness and images with significant occlusions or motion blur. However, several future improvements can be made to enhance the design and performance of the facial recognition system.

7.1 Achievements

Technical Objective	Status and Explanation
To develop a facial recognition system that can identify multiple individuals simultaneously at 93% accuracy with greater distances than 3 meters.	Achieved. The overall system accuracy achieved is 93.44%.
To establish a facial recognition system that can verify 10 individuals in one frame.	Achieved. The system successfully verifies 10 individuals in a single frame.
To implement a real-time facial recognition system that has a processing time of 3 to 6 seconds for multi-face detection and recognition.	Not achieved. The system currently exceeds the target processing time, taking 107.11s for multi-face detection and recognition.

Table 12: Achievements of objectives

In this project, significant progress was made towards developing an effective medium-range facial recognition system. Two out of the three technical objectives were successfully achieved. The system can identify multiple individuals simultaneously at a high accuracy rate of 93.44% even at distances greater than 3 meters, and it can verify 10 individuals in one frame.

However, the third objective, which aimed to implement a real-time facial recognition system with a processing time of 3 to 6 seconds, was not met. The system currently takes longer than the target time for multi-face detection and recognition. This shortfall can be attributed to several factors, including the high computational complexity of the algorithms used, especially Convolutional Neural Networks (CNN) used in this project, which are known for their high accuracy but also require long processing times. Additionally, the limitations of the current hardware setup are not powerful enough to handle the intensive computational load within the desired time frame. Further optimization of the algorithm is required to reduce processing time without compromising accuracy.

Future work should focus on addressing these issues to achieve the desired real-time performance. This could involve optimizing the algorithms for faster execution, upgrading hardware components, and exploring more efficient face detection and recognition techniques. By tackling these challenges, the system's performance can be improved to meet the real-time processing objectives.

7.2 Recommendations for Future Work

The medium-range facial recognition system has been developed with the specifications stated in the objective. Despite the successful completion of this project, there are several recommendations are suggested below to further enhance the performance and capabilities of the medium-range facial recognition system:

- Reduce computational overhead by streamlining and simplifying the existing CNN-based algorithms
- Utilize multi-core processors or GPUs more effectively through parallel processing techniques
- Reduce the model size and complexity using pruning and quantization techniques without significantly compromising accuracy
- Improve input image quality by implementing noise reduction, contrast adjustment, and normalization
- Enhance interoperability with various software platforms and devices by developing API endpoints and ensuring compatibility with standard data formats

8- List of references

[1] AGN Business Internet BV, “5 common biometric techniques compared,” *Recogtech.com*, 2020. https://www.recogtech.com/en/knowledge-base/5-common-biometric-techniques-compared

[2] B. Vidyapeeth, “A Comparative Study of Biometric Technologies with Reference to Human Interface K P Tripathi Lecturer (MCA Programme),” *International Journal of Computer Applications*, vol. 14, no. 5, pp. 975–8887, 2011, Accessed: Oct. 16, 2022. [Online]. Available: https://www.ijcaonline.org/volume14/number5/pxc3872493.pdf

[3] Kavita Manral, “RFID: What are its Advantages and Disadvantages?,” *Schneider Electric Blog*, Jun. 20, 2021. https://blog.se.com/industry/machine-and-process-management/2021/06/20/rifd-what-are-its-advantages-and-disadvantages/

[4] https://www.facebook.com/jason.brownlee.39, “How to Perform Face Detection with Deep Learning,” *Machine Learning Mastery*, Jun. 02, 2019. https://machinelearningmastery.com/how-to-perform-face-detection-with-classical-and-deep-learning-methods-in-python-with-keras/

[5] S. Kostadinov, “What Is Deep Transfer Learning and Why Is It Becoming So Popular?,” *Medium*, Nov. 16, 2019. https://towardsdatascience.com/what-is-deep-transfer-learning-and-why-is-it-becoming-so-popular-91acdcc2717a

[6] E. Burns, “What is deep learning and how does it work?,” *SearchEnterpriseAI*, Mar. 2021. https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network

[7] N. Donges, “What is transfer learning? Exploring the popular deep learning approach,” *Built In*, Aug. 25, 2022. https://builtin.com/data-science/transfer-learning

[8] “What is Face Detection and How Does It Work?,” *SearchEnterpriseAI*. https://www.techtarget.com/searchenterpriseai/definition/face-detection

[9] “What is the Viola-Jones algorithm?,” *Educative: Interactive Courses for Software Developers*. https://www.educative.io/answers/what-is-the-viola-jones-algorithm

[10] “Educative Answers - Trusted Answers to Developer Questions,” *Educative*. https://www.educative.io/answers/what-is-histogram-of-oriented-gradients-hog

[11] A. Mittal, “Haar Cascades, Explained,” *Medium*, Dec. 21, 2020. https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d

[12] “What is SIFT?,” *Educative: Interactive Courses for Software Developers*. https://www.educative.io/answers/what-is-sift

[13] “10 Best Face Recognition APIs,” *www.banuba.com*. https://www.banuba.com/blog/best-face-recognition-apis (accessed Nov. 17, 2023).

[14] “Face Recognition and Face Detection using OpenCV - javatpoint,” *www.javatpoint.com*. https://www.javatpoint.com/face-recognition-and-face-detection-using-opencv

[15] Tashmit, “Coding Ninjas Studio,” *www.codingninjas.com*. https://www.codingninjas.com/studio/library/local-binary-pattern-algorithm (accessed Nov. 17, 2023).

[16] Kaspersky, “What is Facial Recognition – Definition and Explanation,” *Kaspersky*, Jan. 13, 2021. https://www.kaspersky.com/resource-center/definitions/what-is-facial-recognition

[17] P. Antoniadis, “How Do Eigenfaces Work?” https://www.baeldung.com/cs/author/panagiotisantoniadis (accessed Jun. 17, 2023).

[18] “Face Recognition using Fisherfaces,” *OpenGenus IQ: Learn Computer Science*, Oct. 13, 2019. https://iq.opengenus.org/face-recognition-using-fisherfaces/

[19] “What is face recognition?,” *PyImageSearch*, May 01, 2021. https://pyimagesearch.com/2021/05/01/what-is-face-recognition/

[20] D. Tyagi, “Introduction to SURF (Speeded-Up Robust Features),” *Medium*, Apr. 07, 2020. https://medium.com/@deepanshut041/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e

[21] R. Lini, “Facial Landmark Detection Algorithms,” *CodeX*, Sep. 27, 2021. https://medium.com/codex/facial-landmark-detection-algorithms-5b2d2a12adaf

[22] L. Oliver, “3D Face Recognition The Ultimate Guide For Greater Security,” *Facia.ai*, Sep. 08, 2023. https://facia.ai/blog/3d-face-recognition/ (accessed Nov. 17, 2023).

[23] H. T, “DATA FUSION,” *Haileleol Tibebu*, Feb. 03, 2020. https://medium.com/haileleol-tibebu/data-fusion-78e68e65b2d1

[24] G. S. M. Diyasa, A. Fauzi, M. Idhom, and A. Setiawan, “Multi-face Recognition for the Detection of Prisoners in Jail using a Modified Cascade Classifier and CNN,” *Journal of Physics: Conference Series*, vol. 1844, no. 1, p. 012005, Mar. 2021, doi: https://doi.org/10.1088/1742-6596/1844/1/012005.

[25] T. Mantoro, M. A. Ayu, and Suhendi, “Multi-Faces Recognition Process Using Haar Cascades and Eigenface Methods,” *2018 6th International Conference on Multimedia Computing and Systems (ICMCS)*, May 2018, doi: https://doi.org/10.1109/icmcs.2018.8525935.

[26] “University Classroom Attendance System Using FaceNet and Support Vector Machine | IEEE Conference Publication | IEEE Xplore,” *ieeexplore.ieee.org*. https://ieeexplore.ieee.org/document/8921316 (accessed Nov. 17, 2023).

[27] “YOLO Object Detection Explained: A Beginner’s Guide,” *www.datacamp.com*. https://www.datacamp.com/blog/yolo-object-detection-explained

[28] R. Gradilla, “Multi-task Cascaded Convolutional Networks (MTCNN) for Face Detection and Facial Landmark Alignment,” *Medium*, Jul. 27, 2020. https://medium.com/@iselagradilla94/multi-task-cascaded-convolutional-networks-mtcnn-for-face-detection-and-facial-landmark-alignment-7c21e8007923

[29] G. Learning, “Everything you need to know about VGG16,” *Medium*, Sep. 23, 2021. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[30] “Xiaomi Redmi Note 9 Pro 5G - Full phone specifications,” *www.gsmarena.com*. https://www.gsmarena.com/xiaomi_redmi_note_9_pro_5g-10582.php (accessed Nov. 17, 2023).

[31] “Buy ASUS TUF Gaming F15 (FX507Z-C4HN027W) | For-Gaming | Laptops,” *eStore Malaysia*. https://shop.asus.com/my/asus-tuf-gaming-f15-2022-fx507z-c4hn027w.html (accessed Nov. 17, 2023).

[32] “Face detection with dlib (HOG and CNN),” *PyImageSearch*, Apr. 19, 2021. https://pyimagesearch.com/2021/04/19/face-detection-with-dlib-hog-and-cnn/

[33] T. Shastrakar, “How to do Face detection with dlib (HOG and CNN),” *www.linkedin.com*, Apr. 08, 2024. https://www.linkedin.com/pulse/how-do-face-detection-dlib-hog-cnn-tejas-shastrakar-lsaue (accessed May 24, 2024).

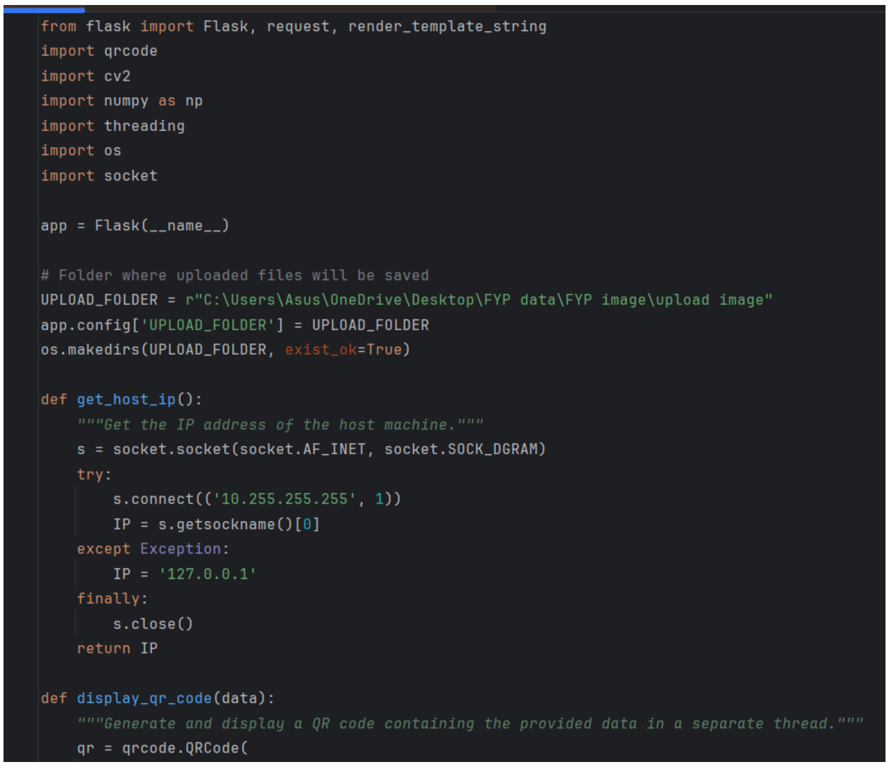
[34] S. Singh, “A Step-by-Step Guide to Face Detection with the dlib Library,” *Medium*, Oct. 02, 2023. https://medium.com/@sukanyasingh303/a-step-by-step-guide-to-face-detection-with-the-dlib-library-2e8f6429e632

[35] S. R. Rath, “Face Detection with Dlib using CNN,” *DebuggerCafe*, Jul. 05, 2021. https://debuggercafe.com/face-detection-with-dlib-using-cnn/ (accessed May 24, 2024).

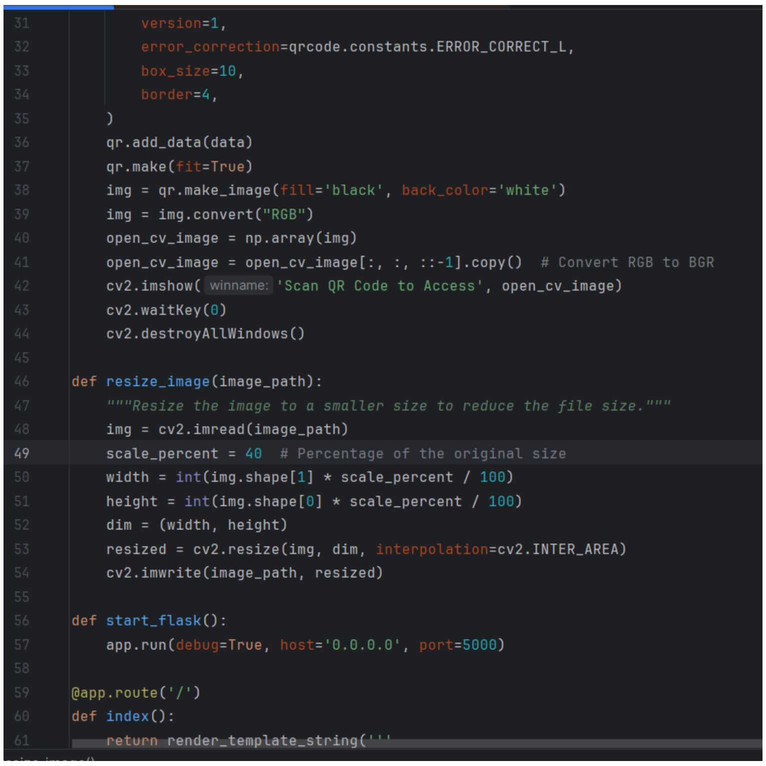
[36] SPARKLERS : We Are The Makers, “Face Recognition Based Complete Attendance System with Database and Webpage using PC or Raspberry Pi,” *YouTube*, Sep. 05, 2023. https://www.youtube.com/watch?v=qeHXHphI9cg (accessed May 24, 2024).

8- Appendix

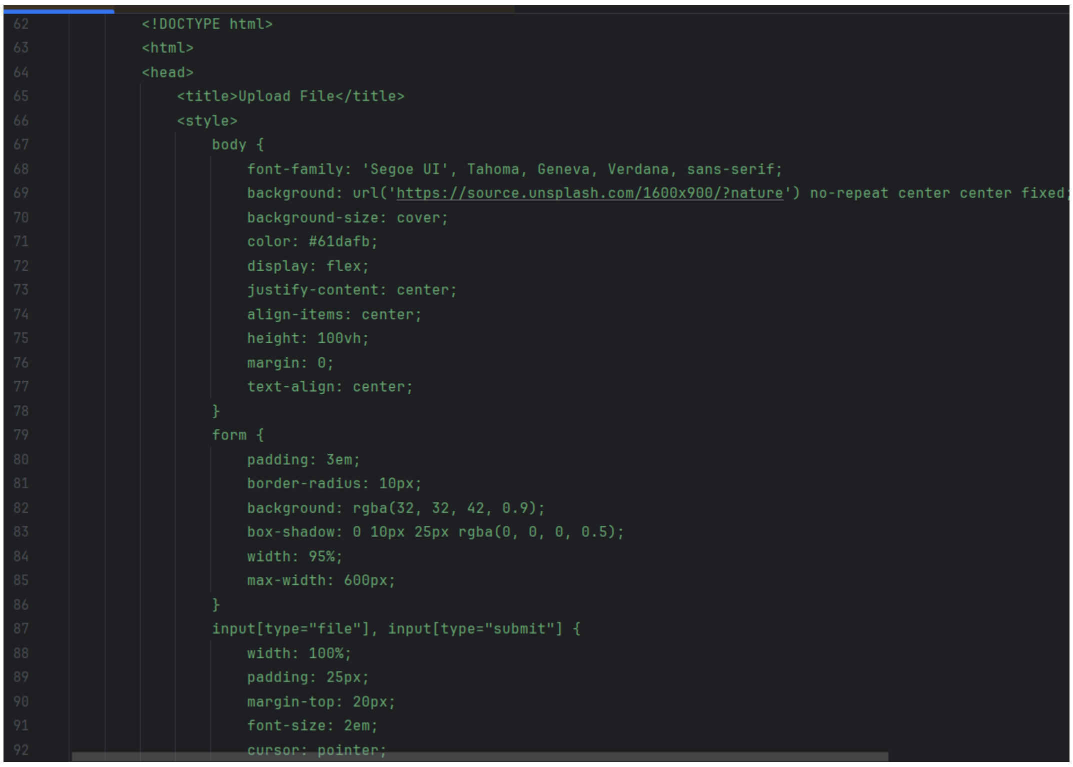
Appendix 1 (“Flask.py”):



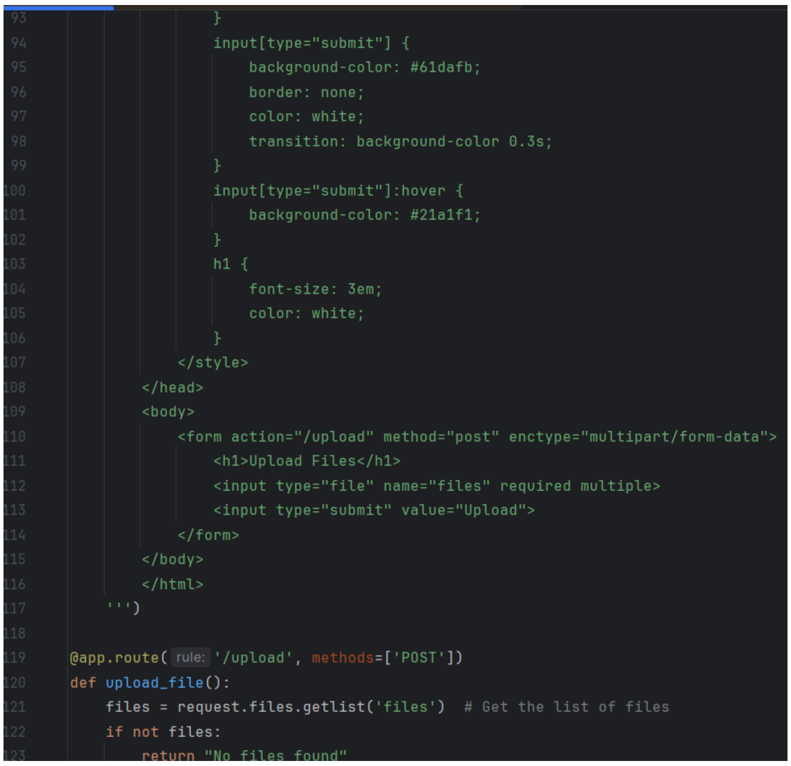
```python
from flask import Flask, request, render_template_string
import qrcode
import cv2
import numpy as np
import threading
import os
import socket

app = Flask(__name__)

# Folder where uploaded files will be saved
UPLOAD_FOLDER = r"C:\Users\Asus\OneDrive\Desktop\FYP data\FYP image\upload image"
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

def get_host_ip():
    """Get the IP address of the host machine."""
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(('10.255.255.255', 1))
        IP = s.getsockname()[0]
    except Exception:
        IP = '127.0.0.1'
    finally:
        s.close()
    return IP

def display_qr_code(data):
    """Generate and display a QR code containing the provided data in a separate thread."""
    qr = qrcode.QRCode(
```



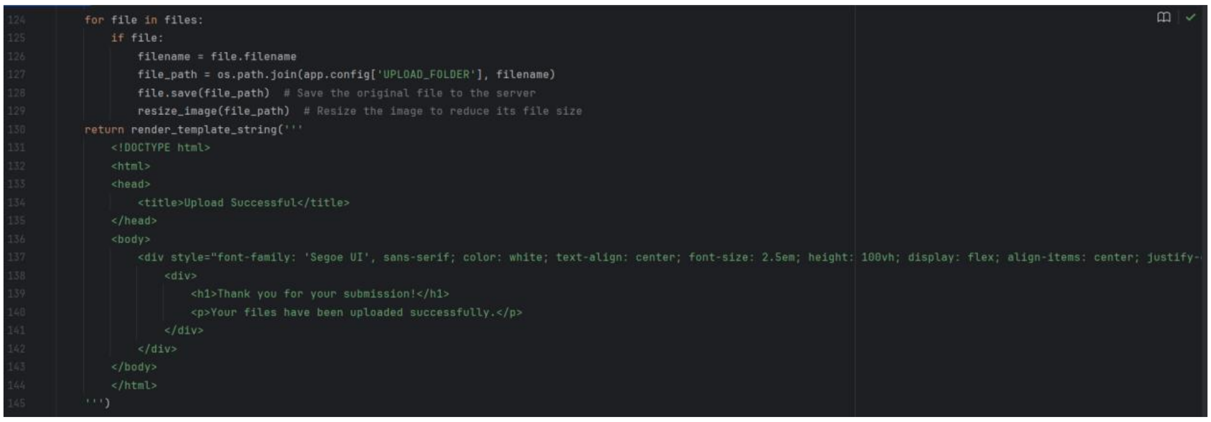
```python
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    qr.add_data(data)
    qr.make(fit=True)
    img = qr.make_image(fill='black', back_color='white')
    img = img.convert("RGB")
    open_cv_image = np.array(img)
    open_cv_image = open_cv_image[:, :, ::-1].copy()  # Convert RGB to BGR
    cv2.imshow( winname: 'Scan QR Code to Access', open_cv_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def resize_image(image_path):
    """Resize the image to a smaller size to reduce the file size."""
    img = cv2.imread(image_path)
    scale_percent = 40  # Percentage of the original size
    width = int(img.shape[1] * scale_percent / 100)
    height = int(img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
    cv2.imwrite(image_path, resized)

def start_flask():
    app.run(debug=True, host='0.0.0.0', port=5000)

@app.route('/')
def index():
    return render_template_string('''
```



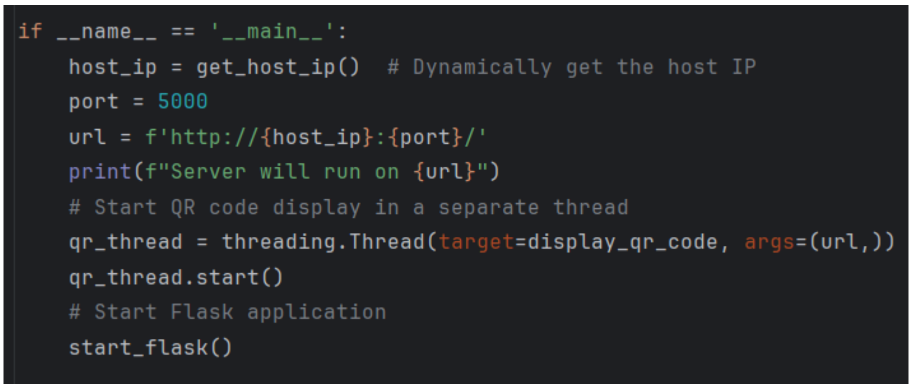
```
        <!DOCTYPE html>
        <html>
        <head>
            <title>Upload File</title>
            <style>
                body {
                    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
                    background: url('https://source.unsplash.com/1600x900/?nature') no-repeat center center fixed;
                    background-size: cover;
                    color: #61dafb;
                    display: flex;
                    justify-content: center;
                    align-items: center;
                    height: 100vh;
                    margin: 0;
                    text-align: center;
                }
                form {
                    padding: 3em;
                    border-radius: 10px;
                    background: rgba(32, 32, 42, 0.9);
                    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.5);
                    width: 95%;
                    max-width: 600px;
                }
                input[type="file"], input[type="submit"] {
                    width: 100%;
                    padding: 25px;
                    margin-top: 20px;
                    font-size: 2em;
                    cursor: pointer;
```



```
                }
                input[type="submit"] {
                    background-color: #61dafb;
                    border: none;
                    color: white;
                    transition: background-color 0.3s;
                }
                input[type="submit"]:hover {
                    background-color: #21a1f1;
                }
                h1 {
                    font-size: 3em;
                    color: white;
                }
            </style>
        </head>
        <body>
            <form action="/upload" method="post" enctype="multipart/form-data">
                <h1>Upload Files</h1>
                <input type="file" name="files" required multiple>
                <input type="submit" value="Upload">
            </form>
        </body>
        </html>
    ''')

@app.route('/upload', methods=['POST'])
def upload_file():
    files = request.files.getlist('files')  # Get the list of files
    if not files:
        return "No files found"
```

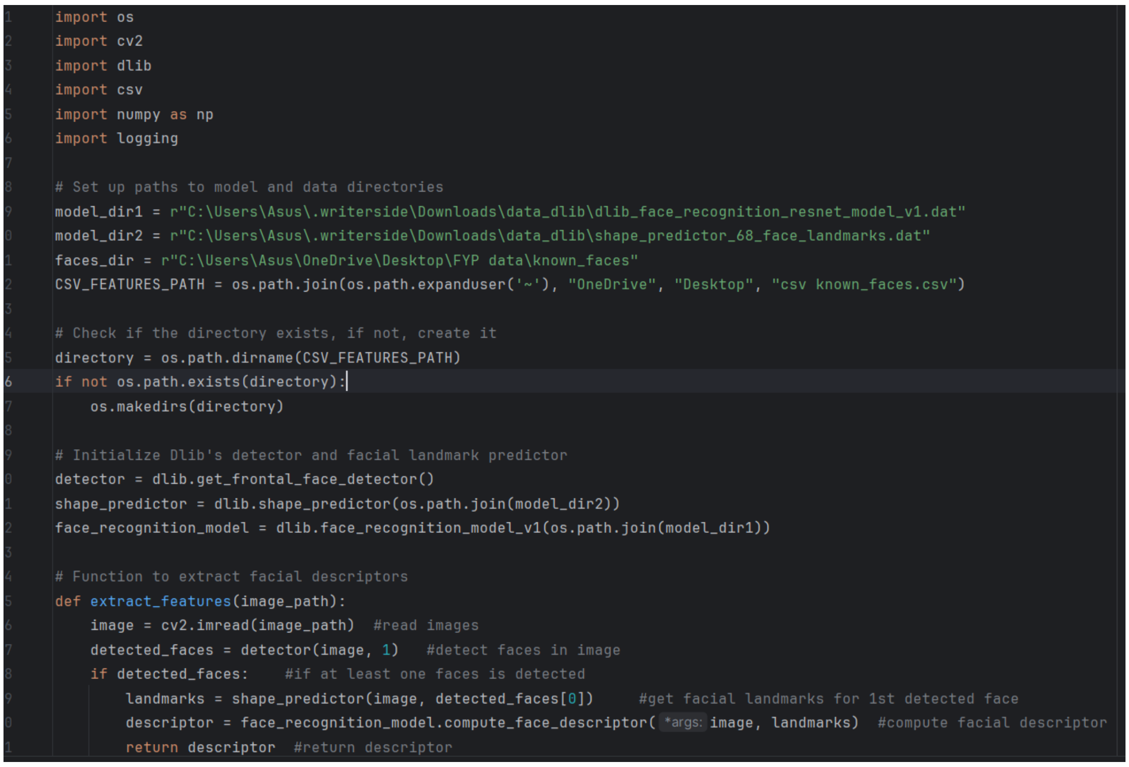


```python
    for file in files:
        if file:
            filename = file.filename
            file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            file.save(file_path)  # Save the original file to the server
            resize_image(file_path)  # Resize the image to reduce its file size
    return render_template_string('''
        <!DOCTYPE html>
        <html>
        <head>
            <title>Upload Successful</title>
        </head>
        <body>
            <div style="font-family: 'Segoe UI', sans-serif; color: white; text-align: center; font-size: 2.5em; height: 100vh; display: flex; align-items: center; justify-
                <div>
                    <h1>Thank you for your submission!</h1>
                    <p>Your files have been uploaded successfully.</p>
                </div>
            </div>
        </body>
        </html>
    ''')
```

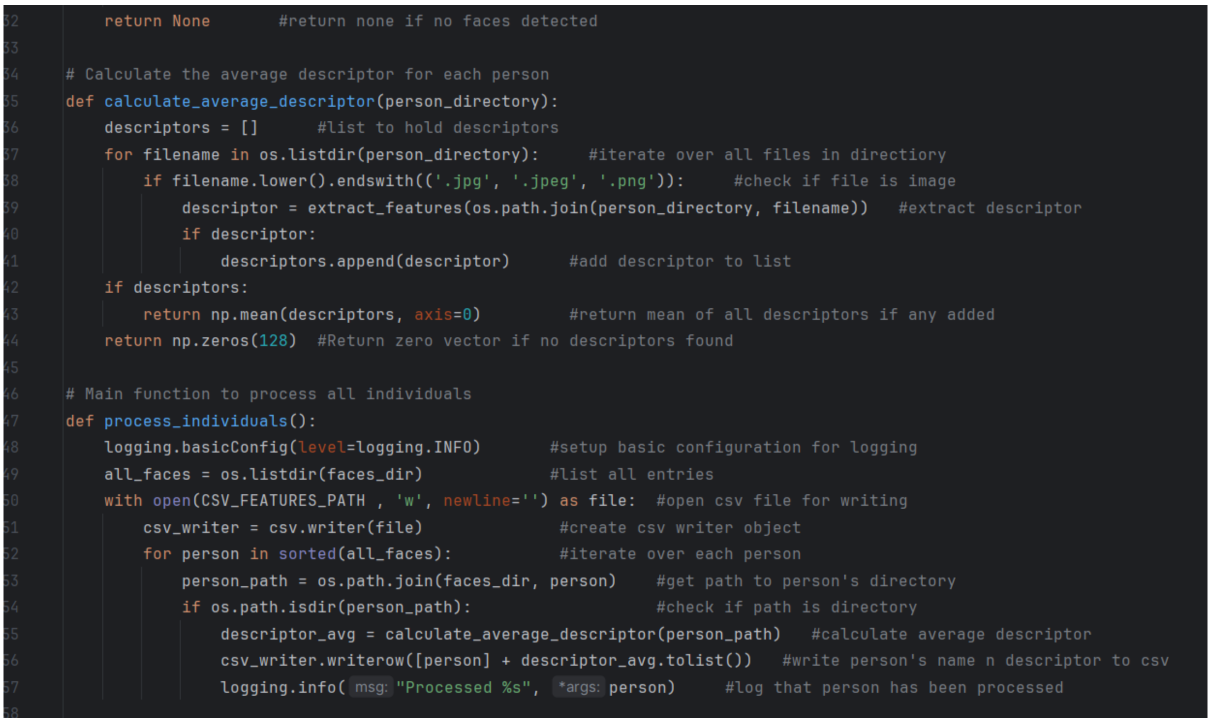


```python
if __name__ == '__main__':
    host_ip = get_host_ip()  # Dynamically get the host IP
    port = 5000
    url = f'http://{host_ip}:{port}/'
    print(f"Server will run on {url}")
    # Start QR code display in a separate thread
    qr_thread = threading.Thread(target=display_qr_code, args=(url,))
    qr_thread.start()
    # Start Flask application
    start_flask()
```

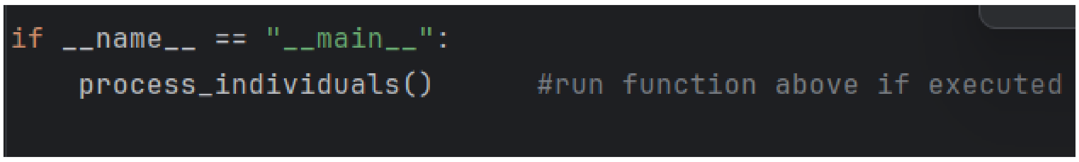
Appendix 2 ("feature extraction.py"):



```python
import os
import cv2
import dlib
import csv
import numpy as np
import logging

# Set up paths to model and data directories
model_dir1 = r"C:\Users\Asus\.writerside\Downloads\data_dlib\dlib_face_recognition_resnet_model_v1.dat"
model_dir2 = r"C:\Users\Asus\.writerside\Downloads\data_dlib\shape_predictor_68_face_landmarks.dat"
faces_dir = r"C:\Users\Asus\OneDrive\Desktop\FYP data\known_faces"
CSV_FEATURES_PATH = os.path.join(os.path.expanduser('~'), "OneDrive", "Desktop", "csv known_faces.csv")

# Check if the directory exists, if not, create it
directory = os.path.dirname(CSV_FEATURES_PATH)
if not os.path.exists(directory):
    os.makedirs(directory)

# Initialize Dlib's detector and facial landmark predictor
detector = dlib.get_frontal_face_detector()
shape_predictor = dlib.shape_predictor(os.path.join(model_dir2))
face_recognition_model = dlib.face_recognition_model_v1(os.path.join(model_dir1))

# Function to extract facial descriptors
def extract_features(image_path):
    image = cv2.imread(image_path)  #read images
    detected_faces = detector(image, 1)   #detect faces in image
    if detected_faces:    #if at least one faces is detected
        landmarks = shape_predictor(image, detected_faces[0])     #get facial landmarks for 1st detected face
        descriptor = face_recognition_model.compute_face_descriptor( *args: image, landmarks)  #compute facial descriptor
        return descriptor  #return descriptor
```

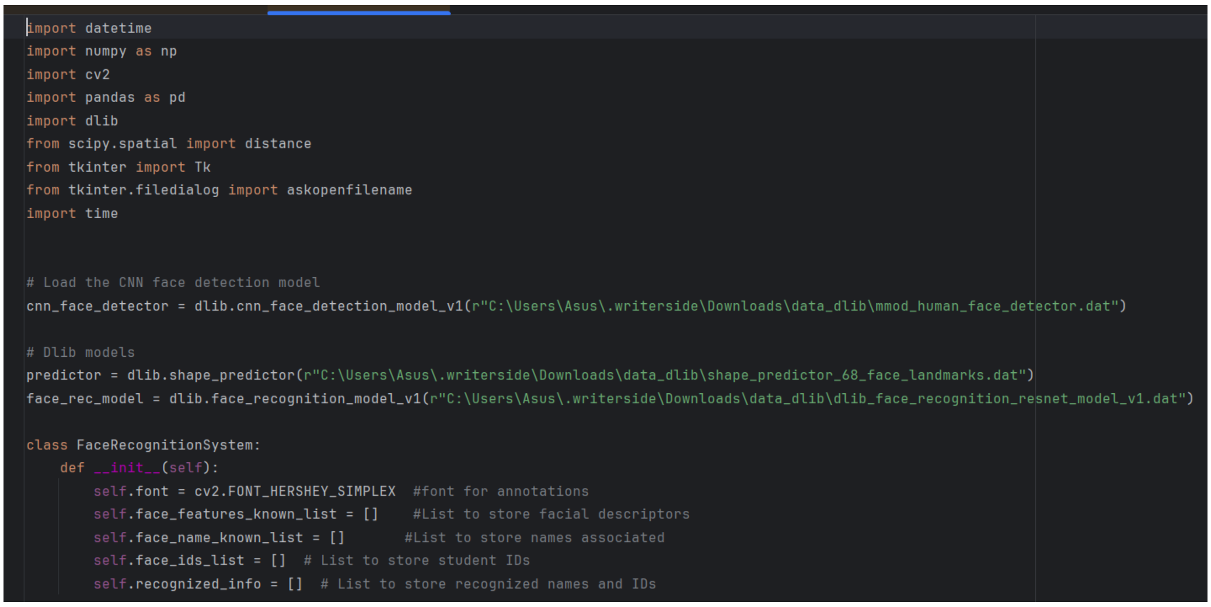


```python
        return None       #return none if no faces detected

# Calculate the average descriptor for each person
def calculate_average_descriptor(person_directory):
    descriptors = []      #list to hold descriptors
    for filename in os.listdir(person_directory):     #iterate over all files in directory
        if filename.lower().endswith(('.jpg', '.jpeg', '.png')):     #check if file is image
            descriptor = extract_features(os.path.join(person_directory, filename))    #extract descriptor
            if descriptor:
                descriptors.append(descriptor)      #add descriptor to list
    if descriptors:
        return np.mean(descriptors, axis=0)         #return mean of all descriptors if any added
    return np.zeros(128)  #Return zero vector if no descriptors found

# Main function to process all individuals
def process_individuals():
    logging.basicConfig(level=logging.INFO)       #setup basic configuration for logging
    all_faces = os.listdir(faces_dir)             #list all entries
    with open(CSV_FEATURES_PATH , 'w', newline='') as file:  #open csv file for writing
        csv_writer = csv.writer(file)              #create csv writer object
        for person in sorted(all_faces):           #iterate over each person
            person_path = os.path.join(faces_dir, person)    #get path to person's directory
            if os.path.isdir(person_path):                   #check if path is directory
                descriptor_avg = calculate_average_descriptor(person_path)   #calculate average descriptor
                csv_writer.writerow([person] + descriptor_avg.tolist())   #write person's name n descriptor to csv
                logging.info( msg: "Processed %s",  *args: person)     #log that person has been processed
```

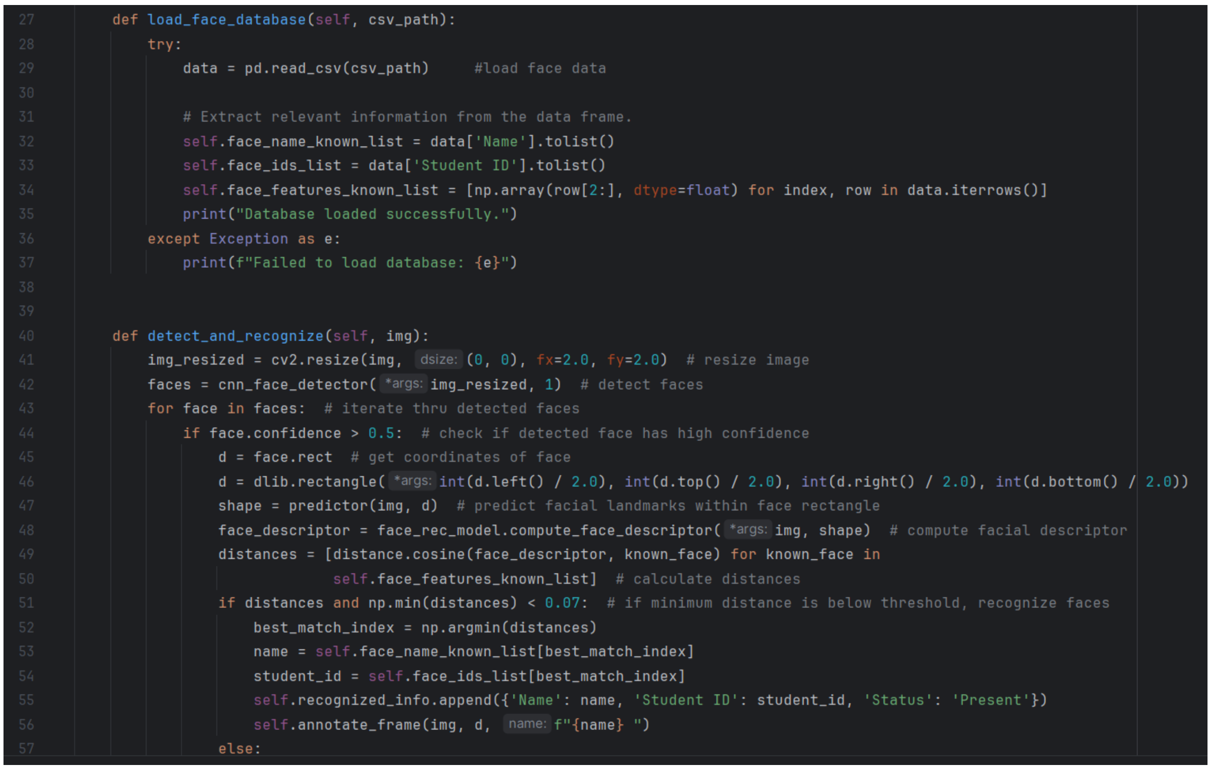


```python
if __name__ == "__main__":
    process_individuals()      #run function above if executed
```

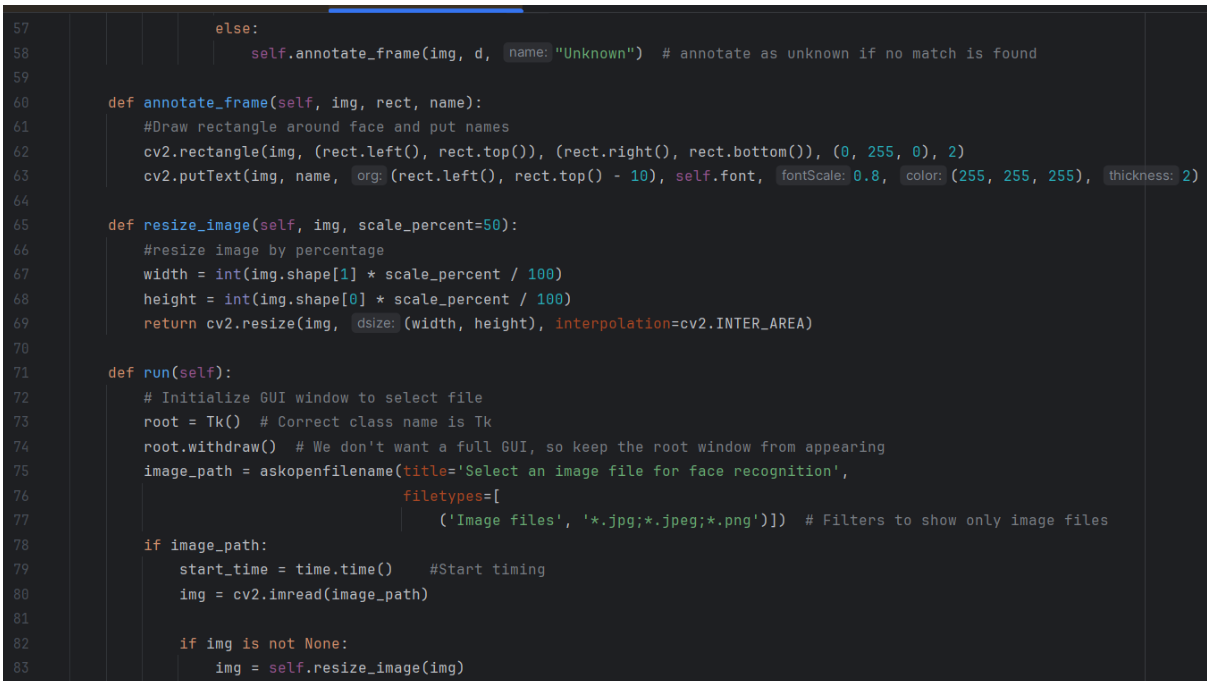
Appendix 3 (“face recognize.py”):



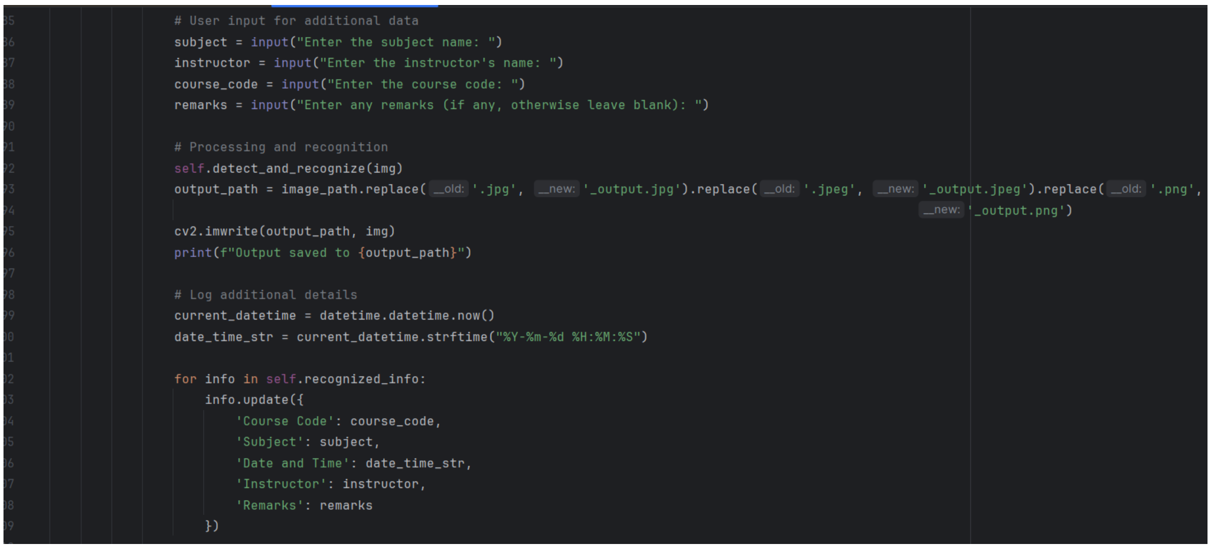
```python
import datetime
import numpy as np
import cv2
import pandas as pd
import dlib
from scipy.spatial import distance
from tkinter import Tk
from tkinter.filedialog import askopenfilename
import time


# Load the CNN face detection model
cnn_face_detector = dlib.cnn_face_detection_model_v1(r"C:\Users\Asus\.writerside\Downloads\data_dlib\mmod_human_face_detector.dat")

# Dlib models
predictor = dlib.shape_predictor(r"C:\Users\Asus\.writerside\Downloads\data_dlib\shape_predictor_68_face_landmarks.dat")
face_rec_model = dlib.face_recognition_model_v1(r"C:\Users\Asus\.writerside\Downloads\data_dlib\dlib_face_recognition_resnet_model_v1.dat")

class FaceRecognitionSystem:
    def __init__(self):
        self.font = cv2.FONT_HERSHEY_SIMPLEX  #font for annotations
        self.face_features_known_list = []    #List to store facial descriptors
        self.face_name_known_list = []       #List to store names associated
        self.face_ids_list = []  # List to store student IDs
        self.recognized_info = []  # List to store recognized names and IDs
```



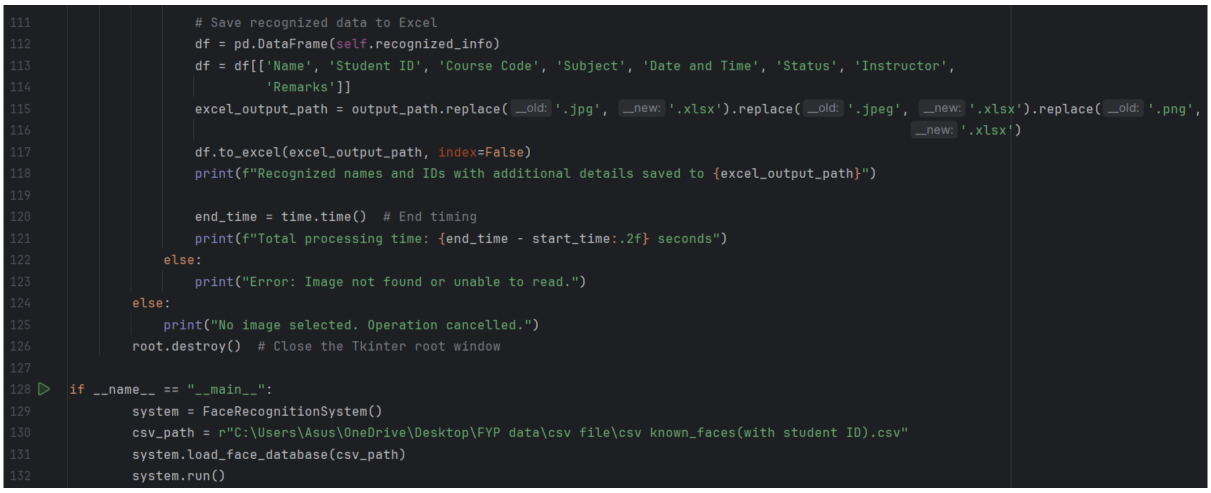
```python
    def load_face_database(self, csv_path):
        try:
            data = pd.read_csv(csv_path)     #load face data

            # Extract relevant information from the data frame.
            self.face_name_known_list = data['Name'].tolist()
            self.face_ids_list = data['Student ID'].tolist()
            self.face_features_known_list = [np.array(row[2:], dtype=float) for index, row in data.iterrows()]
            print("Database loaded successfully.")
        except Exception as e:
            print(f"Failed to load database: {e}")


    def detect_and_recognize(self, img):
        img_resized = cv2.resize(img,  dsize: (0, 0), fx=2.0, fy=2.0)  # resize image
        faces = cnn_face_detector( *args: img_resized, 1)  # detect faces
        for face in faces:  # iterate thru detected faces
            if face.confidence > 0.5:  # check if detected face has high confidence
                d = face.rect  # get coordinates of face
                d = dlib.rectangle( *args: int(d.left() / 2.0), int(d.top() / 2.0), int(d.right() / 2.0), int(d.bottom() / 2.0))
                shape = predictor(img, d)  # predict facial landmarks within face rectangle
                face_descriptor = face_rec_model.compute_face_descriptor( *args: img, shape)  # compute facial descriptor
                distances = [distance.cosine(face_descriptor, known_face) for known_face in
                             self.face_features_known_list]  # calculate distances
                if distances and np.min(distances) < 0.07:  # if minimum distance is below threshold, recognize faces
                    best_match_index = np.argmin(distances)
                    name = self.face_name_known_list[best_match_index]
                    student_id = self.face_ids_list[best_match_index]
                    self.recognized_info.append({'Name': name, 'Student ID': student_id, 'Status': 'Present'})
                    self.annotate_frame(img, d,  name: f"{name} ")
                else:
```



```python
                else:
                    self.annotate_frame(img, d,  name: "Unknown")  # annotate as unknown if no match is found

    def annotate_frame(self, img, rect, name):
        #Draw rectangle around face and put names
        cv2.rectangle(img, (rect.left(), rect.top()), (rect.right(), rect.bottom()), (0, 255, 0), 2)
        cv2.putText(img, name,  org: (rect.left(), rect.top() - 10), self.font,  fontScale: 0.8,  color: (255, 255, 255),  thickness: 2)

    def resize_image(self, img, scale_percent=50):
        #resize image by percentage
        width = int(img.shape[1] * scale_percent / 100)
        height = int(img.shape[0] * scale_percent / 100)
        return cv2.resize(img,  dsize: (width, height), interpolation=cv2.INTER_AREA)

    def run(self):
        # Initialize GUI window to select file
        root = Tk()  # Correct class name is Tk
        root.withdraw()  # We don't want a full GUI, so keep the root window from appearing
        image_path = askopenfilename(title='Select an image file for face recognition',
                                     filetypes=[
                                         ('Image files', '*.jpg;*.jpeg;*.png')])  # Filters to show only image files
        if image_path:
            start_time = time.time()     #Start timing
            img = cv2.imread(image_path)

            if img is not None:
                img = self.resize_image(img)
```

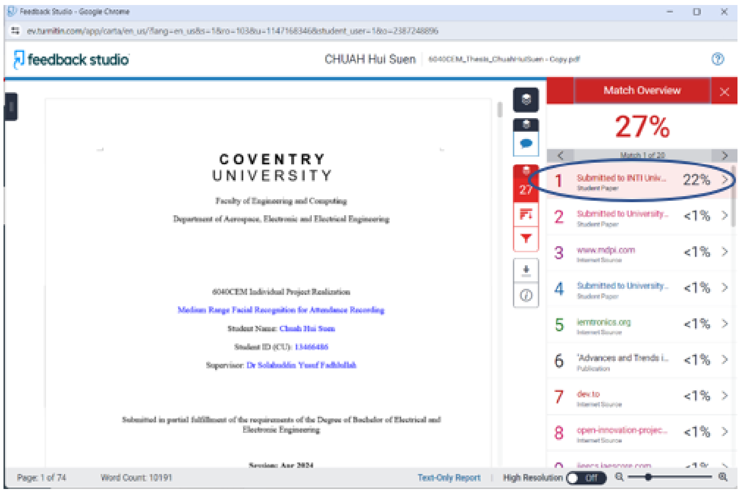


```
            # User input for additional data
            subject = input("Enter the subject name: ")
            instructor = input("Enter the instructor's name: ")
            course_code = input("Enter the course code: ")
            remarks = input("Enter any remarks (if any, otherwise leave blank): ")

            # Processing and recognition
            self.detect_and_recognize(img)
            output_path = image_path.replace( __old: '.jpg',  __new: '_output.jpg').replace( __old: '.jpeg',  __new: '_output.jpeg').replace( __old: '.png',
                                                                                                                                   __new: '_output.png')
            cv2.imwrite(output_path, img)
            print(f"Output saved to {output_path}")

            # Log additional details
            current_datetime = datetime.datetime.now()
            date_time_str = current_datetime.strftime("%Y-%m-%d %H:%M:%S")

            for info in self.recognized_info:
                info.update({
                    'Course Code': course_code,
                    'Subject': subject,
                    'Date and Time': date_time_str,
                    'Instructor': instructor,
                    'Remarks': remarks
                })
```



```
                # Save recognized data to Excel
                df = pd.DataFrame(self.recognized_info)
                df = df[['Name', 'Student ID', 'Course Code', 'Subject', 'Date and Time', 'Status', 'Instructor',
                         'Remarks']]
                excel_output_path = output_path.replace( __old: '.jpg',  __new: '.xlsx').replace( __old: '.jpeg',  __new: '.xlsx').replace( __old: '.png',
                                                                                                                             __new: '.xlsx')
                df.to_excel(excel_output_path, index=False)
                print(f"Recognized names and IDs with additional details saved to {excel_output_path}")

                end_time = time.time()  # End timing
                print(f"Total processing time: {end_time - start_time:.2f} seconds")
            else:
                print("Error: Image not found or unable to read.")
        else:
            print("No image selected. Operation cancelled.")
        root.destroy()  # Close the Tkinter root window

if __name__ == "__main__":
        system = FaceRecognitionSystem()
        csv_path = r"C:\Users\Asus\OneDrive\Desktop\FYP data\csv file\csv known_faces(with student ID).csv"
        system.load_face_database(csv_path)
        system.run()
```

Appendix 4 (Turnitin plagiarism in percentage = 22% since others one is reference link provided in reference part):



Feedback Studio - Google Chrome

ev.turnitin.com/app/carta/en_us/?lang=en_us&s=1&ro=103&u=1147168346&student_user=1&o=2387248896

feedback studio

CHUAH Hui Suen | 6040CEM_Thesis_ChuahHuiSuen - Copy.pdf

COVENTRY UNIVERSITY

Faculty of Engineering and Computing

Department of Aerospace, Electronic and Electrical Engineering

6040CEM Individual Project Realization

Medium Range Facial Recognition for Attendance Recording

Student Name: Chuah Hui Suen

Student ID (CU): 13466486

Supervisor: Dr Solahuddin Yusuf Fadhlullah

Submitted in partial fulfillment of the requirements of the Degree of Bachelor of Electrical and Electronic Engineering

Session: Apr 2024

Match Overview

27%

Match 1 of 20

1	Submitted to INTI Univ... Student Paper	22%
2	Submitted to University... Student Paper	<1%
3	www.mdpi.com Internet Source	<1%
4	Submitted to University... Student Paper	<1%
5	iemtronics.org Internet Source	<1%
6	'Advances and Trends i... Publication	<1%
7	dev.to Internet Source	<1%
8	open-innovation-projec... Internet Source	<1%
9	ieeecs.iaescore.com	<1%

Page: 1 of 74 Word Count: 10191 Text-Only Report High Resolution Off