Tribhuvan University

Institute of Science and Technology

Prime College

Nayabazar, Kathmandu, Nepal

A Project Report on
"Personalized Gesture Recognition System"

Submitted by:

Sujal Shakya (23506/076)

Susit Ratna Tuladhar (23511/076)

Umang Shakya (23513/076)

A Project Report Submitted in Partial fulfilment of the requirement of

Bachelor of Science in Computer Science & Information Technology (BSc. CSIT)

7th Semester of Tribhuvan University, Nepal

Falgun, 2080

PERSONALIZED GESTURE RECOGNITION SYSTEM
[CSC 412]

A project report submitted for the partial fulfillment of the requirement for the degree of Bachelor of Science in Computer science & Information Technology awarded by Tribhuvan University.

Submitted By

Sujal Shakya (23506/076)

Susit Ratna Tuladhar (23511/076)

Umang Shakya (23513/076)

Submitted To

Prime College

Department of Computer science

Affiliated to Tribhuvan University

Khusibun, Nayabazar, Kathmandu

Falgun, 2080

Date: 21st Falgun, 2080

SUPERVISOR’S RECOMMENDATION

It is my pleasure to recommend that a report on “**PERSONALIZED GESTURE RECOGNITION SYSTEM**” has been prepared under my supervision by **Sujal Shakya, Susit Ratna Tuladhar** and **Umang Shakya** in partial fulfillment of the requirement of the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT). Their report is satisfactory to process for the future evaluation.

…………………………………

Mr. Sudan Prajapati

Supervisor

Department of Computer Science & IT

Prime College

Date: 21st Falgun, 2080

CERTIFICATE OF APPROVAL

This is to certify that this report has been read and recommended to the Department of Computer Science and Information Technology for acceptance of report entitled "**PERSONALIZED GESTURE RECOGNITION SYSTEM**" submitted by **Sujal Shakya, Susit Ratna Tuladhar** and **Umang Shakya** in partial fulfillment for the degree of Bachelor of Science in Computer Science and Information Technology (BSc.CSIT), Institute of Science and Technology, Tribhuvan University.

………………………	**………………………**
Mr. Narayan Prasad Sharma	**Ms. Rolisha Sthapit**
Principal	Program Coordinator
………………………	**……………………...**
Mr. Sudan Prajapati	**Mr. Sarbin Sayami**
Supervisor	External Examiner

ACKNOWLEDGMENT

We would like to express our sincere thanks to the Department of Computer Science and Information Technology, Prime College, Khusibu, Kathmandu for providing us with this opportunity to explore our thoughts, curiosity and interest in the field of computer science through a thorough Final Year Project on "**Personalized Gesture Recognition System**".

We would like to express our deepest sense of gratitude and sincere thanks to our project supervisor, **Er. Sudan Prajapati** for his valuable guidance, suggestions, encouragement, critical comments and help for completing this project. His constant support and motivation encouraged us to bring the project to life.

We would like to express our outmost gratitude to our coordinator, **Er. Rolisha Sthapit** for giving us this valuable opportunity to develop the project. Her useful suggestions for this whole work and co-operative behavior are sincerely acknowledged. In addition, we are also thankful to our friends, seniors, and all the staff members of B.Sc. CSIT department for their timely support and guidance throughout the project.

With respect,

Sujal Shakya (23506/076)

Susit Ratna Tuladhar (23511/076)

Umang Shakya (23513/076)

ABSTRACT

Personalized Gesture Recognition System is an application that primarily focuses on the recognition of static hand gestures and implementing the gestures for Human-Computer Interaction (HCI). The project emphasizes the importance of natural and intuitive user-computer communication by allowing users to train the system with personalized static hand gestures. Technology is now more accessible and boundaries are removed. Real-time gesture recognition is achieved by leveraging the K-Nearest Neighbours (KNN) algorithm, which is advantageous in that it can classify inputs with less information. By utilising a common webcam or built-in camera, it does away with the requirement for extra imaging and tracking equipment. The implemented system, created with Python using OpenCV and Mediapipe, is designed to identify, recognize, and interpret personalized static hand gestures. The system integrates the detected and classified gestures to trigger keyboard events, enhancing user-computer interaction.

Keywords: *Human-Computer Interaction (HCI), K-Nearest Neighbors (KNN), Real-Time Gesture Recognition, Python, OpenCV, MediaPipe, Custom Hand Gestures, Keyboard Events*

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

API -	Application Programming Interface
AR -	Augmented Reality
ASL -	American Sign Language
CNN -	Convolutional Neural Network
CPU -	Central Processing Unit
CSV -	Comma Separated Value
EMG –	Electromyography
FPS -	Frames Per Second
GPU -	Graphics Processing Unit
HCI -	Human-Computer Interaction
HGR -	Hand Gesture Recognition
KNN -	K-Nearest Neighbors
ML -	Machine Learning
NLP-	Natural Language Processing
SVM -	Support Vector Machine
UML-	Unified Model Language
UI -	User Interface
VR -	Virtual Reality

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Introduction

Gesture recognition offers a path for computer devices to better understand and interpret various human hand gestures. Gesture recognition, a captivating aspect of Human-Computer Interaction (HCI), seamlessly connects the physical and digital worlds, enabling users to communicate with machines through intuitive movements. One of the key advantages of gesture recognition is its natural and intuitive interface. Instead of relying on traditional input methods like keyboards or mice, users can interact with computers in a more fluid and expressive manner, using gestures that mimic real-world actions. Incorporating these natural gestures into digital interfaces enhances user interaction, creating a more natural and immersive computing experience [1].

Human hand gestures serve as a non-verbal means of communication, encompassing a spectrum from basic actions like pointing and moving objects to intricate expressions of emotions and communication [2]. These gestures fall into two categories: static, where users hold a specific pose, and dynamic, which involves distinct pre-stroke, stroke, and post-stroke phases. [1]. This project specifically focuses on the recognition of static hand gestures, acknowledging their prevalence and significance in real-life scenarios.

In the realm of HCI, where the goal is to make computer systems more user-friendly and responsive to human behavior, personalized gesture recognition plays a pivotal role. By enabling users to interact with devices through personalized gestures, HCI breaks down barriers and makes technology more accessible [3]. Our project aligns with HCI principles, prioritizing users to train the system with personalized static gestures for intuitive interaction. By utilizing gestures as input, our project streamlines interactions, reducing the need for additional devices and enhancing user mobility.

In the field of gesture recognition, various methods like Convolutional Neural Networks (CNN) and Support Vector Machines (SVM) have been investigated [4]. The selection of K-Nearest Neighbors (KNN) and Decision Tree for this project was based on their effectiveness in predicting gestures with minimal data. KNN and Decision Tree provide quicker outcomes than the alternative algorithms [5], rendering them well-suited for real-time gesture recognition applications, where users have the capability to train the application to recognize their individualized gestures.

Additionally, practical implementation often requires imaging and tracking devices, such as gloves, body suits, and optical tracking markers. [1]. This project simplifies gesture recognition by utilizing the simplicity of a standard web or built-in camera, eliminating the need for additional devices.

The project focuses on developing a system that detects, recognizes, and interprets customizable static hand gestures using Python, OpenCV, and MediaPipe. It remains robust across variations in light intensity, pose, or hand orientation. The system utilizes a laptop's built-in camera to capture and classify static gestures using the KNN and Decision Tree algorithm. The system integrates the detected and classified gestures to trigger keyboard events, enhancing user-computer interaction.

1.2 Problem Statement

The realm of human-computer interaction is marked by challenges in both hand gesture recognition (HGR) and the efficiency of traditional input methods. Existing HGR systems are constrained by predefined sets of gestures, limiting users and developers from effectively recognizing new, unseen gestures due to the impracticality of including the vast diversity of plausible hand shapes in a predefined list. Simultaneously, traditional input methods, reliant on keyboards and peripherals, lack the desired intuitiveness and fluidity.

Users face hurdles in customizing and deploying their own gesture recognition pipelines, particularly when lacking machine learning expertise. Moreover, conventional input methods pose accessibility issues for users with physical limitations. To address these challenges, this project aims to introduce a user-friendly application that allows to seamlessly customize and deploy gesture recognition models. By mapping key bindings to gestures, the project seeks to enhance the overall user experience, streamline the input process, provide an alternative and accessible method for users, and empower users to personalize their interaction paradigms, thus significantly improving productivity. The project aims to redefine how users engage with and control their digital environments, responding to the need for a more adaptable, accessible, and personalized human-computer interaction solution.

1.3 Objectives

- To develop a gesture recognition system based on KNN and Decision Tree (Gini) algorithms that enables users to train the application to accurately recognize personalized static hand gestures.
- To develop a desktop interface enabling users to employ personalized hand gestures to trigger keyboard events.

1.4 Scope and Limitation

This project aims to develop a gesture recognition system using the K-Nearest Neighbors (KNN) and Decision Tree algorithm, with a primary focus on recognizing personalized static hand gestures. The goal is to enhance Human-Computer Interaction (HCI) by providing a seamless and intuitive user experience. The project acknowledges the prevalence and significance of static gestures in real-life scenarios, emphasizing their importance in the development of the system. The project aims to provide a user-friendly desktop interface that allows users to train the system to accurately recognize their individualized static hand gestures. Utilizing Python, OpenCV, and Mediapipe, the system integrates the detected and classified gestures to trigger keyboard events, providing an alternative to conventional user-computer interaction.

The limitations of the project may include:

Gesture Complexity: The system focuses on static hand gestures, and while it caters to a wide range of gestures, highly intricate or dynamic gestures may pose challenges in recognition.
Hardware Dependency: The system relies on a laptop's built-in camera for gesture recognition. Variations in camera quality and specifications may impact the accuracy and robustness of the recognition system.
Limited Gesture Training: The success of personalized gesture recognition depends on the user's ability to effectively train the system. Users must adhere to training guidelines for optimal results.
Real-Time Processing: While KNN is chosen for its efficiency, real-time processing of gestures may still be influenced by the hardware capabilities of the user's device.
Single-Hand Gestures: The current implementation of the system restricts hand gesture training to the right hand only, limiting the system's ability to recognize gestures performed by the left hand.
Mouse Controls: The system's current configuration does not include comprehensive support for gesture-based mouse controls, potentially restricting users who rely heavily on this input method.

1.5 Development Methodology

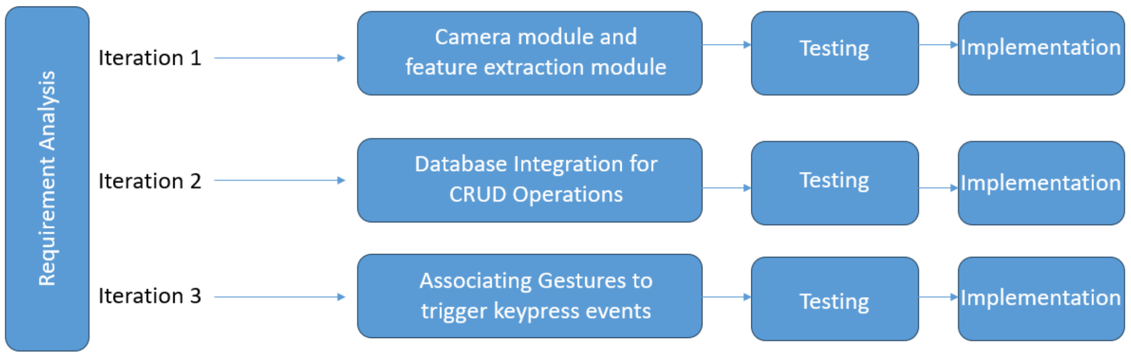

Requirement Analysis
Iteration 1
Camera module and feature extraction module
Testing
Implementation
Iteration 2
Database Integration for CRUD Operations
Testing
Implementation
Iteration 3
Associating Gestures to trigger keypress events
Testing
Implementation

Figure 1.1: Incremental Development Methodology

To design our system, we intended to adopt the incremental development paradigm. The concept behind the incremental model was that new features should be added to an existing system via "increments" rather than starting from scratch all at once. The following benefits that the incremental development methodology offers led us to choose to employ it:

a. **Progressive Refinement:** Progressive system improvement is made possible via incremental development. The system was developed in manageable, modest steps, with each step adding new capability or improving already-existing capabilities. This method aids in controlling complexity and concentrating on particular system components at once.

b. **Early Delivery:** Delivery of incomplete but potentially functional versions of the system to end users or stakeholders is made possible by incremental models. This implies that some system features could be implemented and used sooner, giving users and stakeholders benefits even as development goes on.

c. **Parallel Development:** Development can be sped up by allowing separate teams or people to work on various increments concurrently. In bigger projects, this parallel development can be especially helpful.

1.6 Report Organization

The comprehensive report on the “Personalized Gesture Recognition System” project organizes its findings into a cohesive and well-structured format.

Chapter 1, “Introduction”, introduces the project’s context and importance, highlighting the challenges addressed by the gesture recognition system. It provides clear objectives, outlines the project’s scope and limitations, and establishes the chosen Incremental development methodology as the guiding framework. This chapter also offers readers a preview of the report’s organizational structure.

Chapter 2, “Background Study and Literature Review”, presents a detailed background study on gesture recognition technology and a thorough review of existing literature.

Chapter 3, “System Analysis”, conducts a detailed examination and documentation of the existing system that serves as the basis for the project. This chapter examines the functional and non-functional requirements of the system and evaluates the feasibility of the project, encompassing technical, operational, economic, and scheduling considerations.

Chapter 4, “System Design”, focuses on the intricate details of building the gesture recognition system. It constitutes of detailed diagrams and nature of the algorithm used.

Chapter 5, “System Implementation and Test”, progresses the project from design to reality. This chapter outlines the implementation process and details the tools and techniques employed. It seamlessly transitions into testing, where validation and reliability become important. Unit testing, System testing and Performance Testing are elaborated upon, ensuring the system’s functionality and robustness.

Chapter 6, “Conclusion and Recommendation”, marks the culmination of the project’s journey. It encapsulates the outcomes and lessons gleaned from the endeavor, reinforcing its significance and contributions. Future recommendations chart a path for potential enhancements and extensions of the gesture recognition system, providing a forward-looking perspective.

CHAPTER 2
BACKGROUND STUDY AND LITERATURE REVIEW

2.1 Background Study

The field of Human-Computer Interaction (HCI) focuses on exploring how humans and computers interact effectively. Among various methods, hand gestures offer a natural and intuitive way for people to communicate with technology [1]. This has led to a surge in the development of Hand Gesture Recognition (HGR) systems in recent years, finding applications in diverse domains like virtual reality, sign language recognition, gaming, and more.

Hand Gesture Recognition (HGR) systems use vision-based (cameras, depth sensors) and sensor-based (data gloves, EMG sensors) approaches. Vision-based focuses on appearance (color, texture) and model-based (3D hand models) methods. Sensor-based includes accurate data gloves (cumbersome) and less intrusive EMG sensors (require calibration). Trade-offs exist between accuracy, wearability, and computational demands in HGR systems [6].

In the project, the vision-based methodology employs MediaPipe, an open-source framework developed by Google. This framework streamlines the creation of multimodal machine learning pipelines, providing pre-trained models for tasks such as face, hand, and pose detection. This approach enables the recognition of custom hand gestures without requiring any external devices, except for a camera.

The project employs K-Nearest Neighbors (KNN) and Decision Tree algorithms for classification. KNN assigns labels to new data points by considering the majority vote from their k closest neighbors within the training dataset [7]. In contrast, the Decision Tree, as a supervised machine learning algorithm, constructs a tree-shaped framework through recursive dataset splitting based on key features. The objective is to establish a set of rules for categorizing instances into distinct categories.

2.2 Literature Review

Hand gesture recognition (HGR) has emerged as a powerful tool for natural and intuitive human-computer interaction (HCI), enabling users to control devices and applications through hand movements. Over the past years, significant research efforts have been dedicated to developing robust and efficient HGR systems, exploring diverse algorithms, applications, and advancements. This literature review delves into the state-of-the-art in HGR, examining key research papers that shed light on various approaches, applications, and future directions in this dynamic field. By analyzing these contributions, we aim to gain a comprehensive understanding of the current landscape of HGR and situate our own project, which utilizes KNN and MediaPipe for gesture recognition and key press triggering, within this broader context.

In the paper titled "Gesture Recognition: A Survey," it is mentioned that gesture recognition entails the identification of meaningful human expressions of motion and is essential for crafting intelligent and efficient human-computer interfaces. The applications of gesture recognition span a diverse range, encompassing areas such as sign language, medical rehabilitation, and virtual reality [1].

In the paper titled "A Review of Hand Gesture Recognition with a Focus on Human-Computer Interaction," it is asserted that the visual interpretation of hand gestures plays a crucial role in attaining the desired ease and naturalness for Human-Computer Interaction (HCI). This perspective has stimulated a highly active research domain dedicated to the analysis and interpretation of hand gestures using computer vision [2].

In the paper titled "Performance Analysis of KNN, SVM, and ANN Techniques for Gesture Recognition System," the research focuses on the development of a system for identifying gestures in American Sign Language (ASL), which is an active and evolving area within computer vision and machine learning. The proposed method utilizes Kinect cameras for capturing gesture signals and extracting features through the application of KNN, SVM, and ANN for classification purposes. The system attains a notable recognition accuracy of 97.10% specifically for static ASL numerical signs, with SVM demonstrating superior performance compared to the other techniques [7].

Existing-applications:

Minorva Falk/KNN Alphabet:

This project utilizes MediaPipe for hand landmark detection and KNN for recognition of American Sign Language (ASL) alphabets. It achieves accurate results within a limited alphabet scope, demonstrating the potential of the approach for specific gesture recognition tasks.

Hand Gesture Recognition for American Sign Language using K-Nearest Neighbor with Mediapipe:

This project combines MediaPipe hand detection with KNN for ASL gesture classification. It showcases the capabilities of the method for real-time gesture recognition.

Gesture recognition using Mediapipe Framework and KNN algorithm:

This project demonstrates the use of MediaPipe hand landmark extraction coupled with KNN for general gesture recognition. It highlights the suitability of this approach for simpler gesture recognition tasks.

CHAPTER 3
SYSTEM ANALYSIS

3.1 System Analysis

Systems analysis is a process of studying a system or organization in order to understand its components, how they interact and how they can be improved. It is a holistic approach that looks at the system as a whole and identifies the relationships between its parts. The goal of systems analysis is to identify problems and inefficiencies in the current system and to propose solutions for improvement.

3.1.1 Requirement Analysis

Requirement analysis is a critical phase in software development that involves gathering, analyzing, and documenting the needs and constraints of the project. For the Personalized Gesture Recognition project, the requirement analysis process aimed to identify the key features and functionality of the system, as well as any constraints or limitations that needed to be considered during development.

3.1.1.1 Functional Requirements

The functional requirement of this project describes different features and behavior of the system which will be fulfilling the purpose which was intended and to meet the needs of the users. Here, the system will track and capture the hand gestures through the camera in real-time. It extracts the essential features from the hand's position and movement, which enables swift identification of the gestures through real-time prediction. The result is promptly displayed to the user, facilitating seamless interaction through hand gestures.

Table 3.1: **Use Case Scenario Table: Training a classification model**

Use Case Scenario: Training a classification model	
Primary Actor: User	
Secondary Actor: System	
Precondition: The device needs to have a camera installed.	
Flow of Events:	
1.	User starts the application
2.	User switches to 'Features' tab to add a new model
3.	User selects the created model and clicks the 'Add Features' button
4.	System turns on the camera to detect the hand gesture
5.	System extracts the landmarks from the hand and saves it in a .csv file
6.	User clicks the 'Train' button
7.	System trains the model with captured features and generates a .sav file
Post Condition: System should generate a model ready for real time classification	

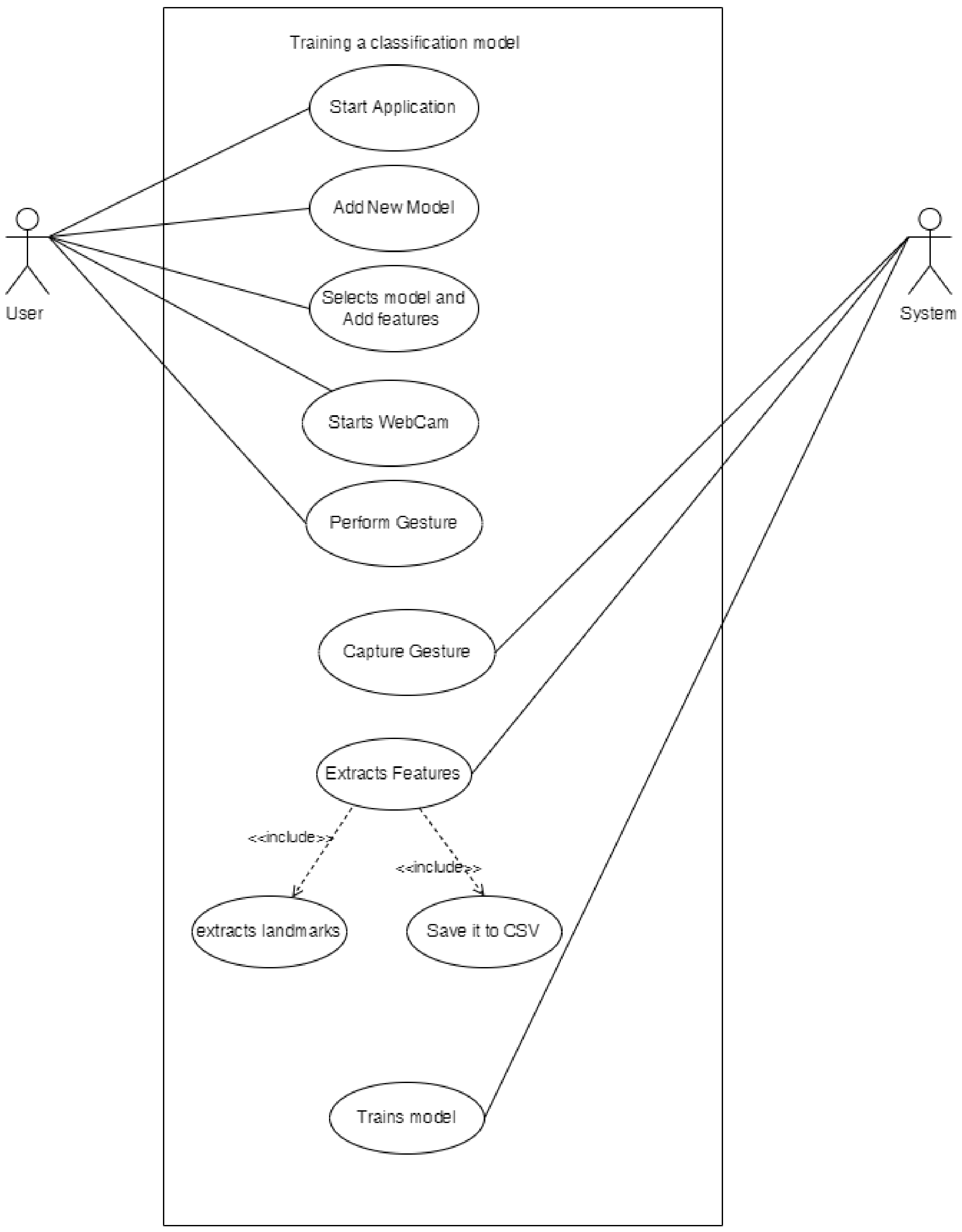


Figure 3.1: Use Case Diagram: Training Classification Model

Table 3.2: Use Case Scenario Table: Real Time Prediction

Use Case Scenario: Real Time Prediction using a trained classification model	
Primary Actor: User	
Secondary Actor: System	
Precondition: The device needs to have a trained classification model	
Flow of Events:	
1.	User starts the application.
2.	User selects a trained model and an available profile.
3.	User clicks the 'Turn On' button to start the camera.
4.	System detects the hand gesture
5.	System extracts the landmarks from the hand and feeds it to the selected model
6.	System then performs corresponding key press event mapped to the classified gesture
Post Condition: System should detect the hand gesture and perform corresponding key press event	

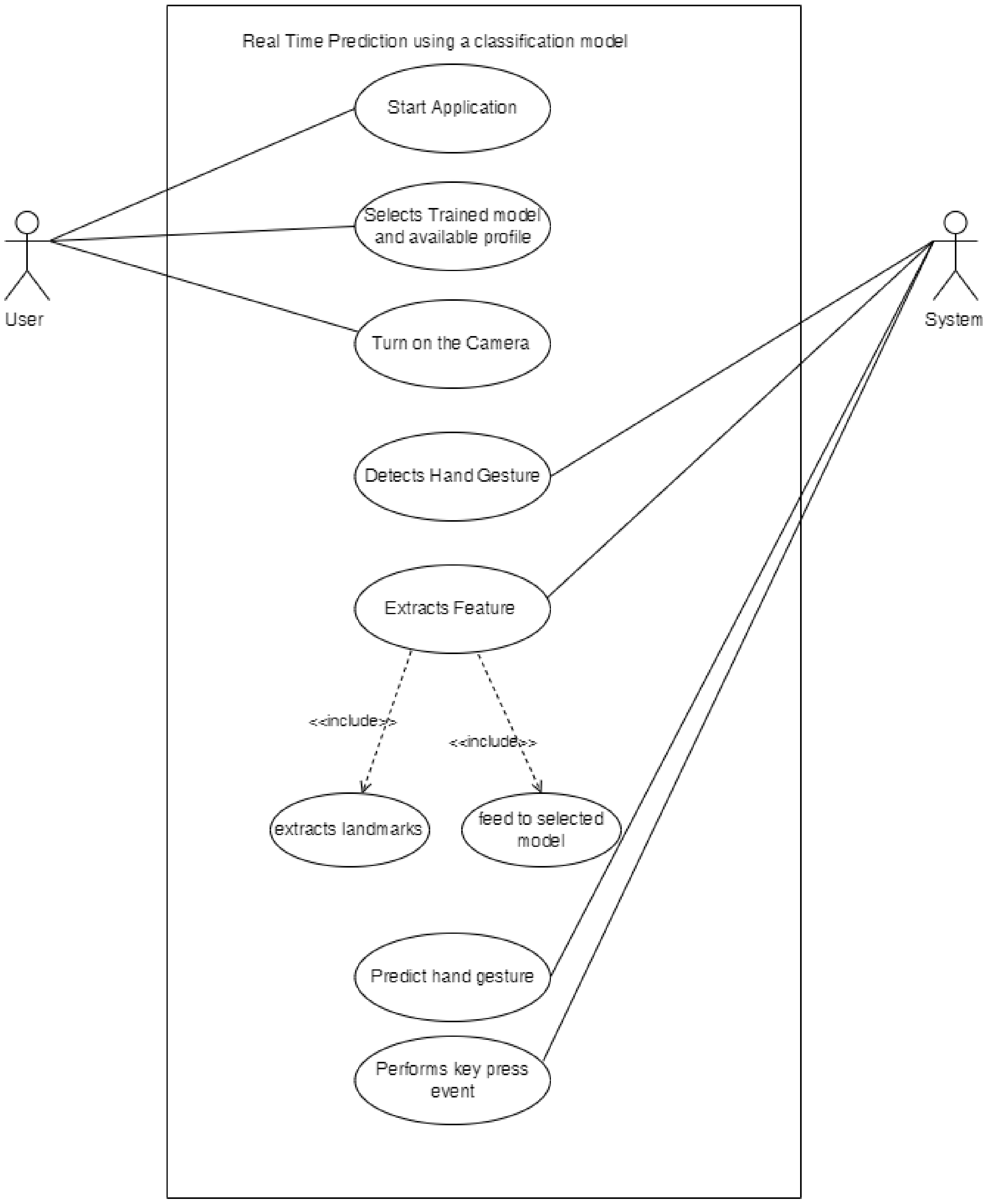


Figure 3.2: Use Case Diagram: Real Time Prediction using Classification Model

3.1.1.2 Non-functional Requirements

Non-functional requirements define the quality attributes, constraints, and characteristics that shape how a system behaves, performs, and interacts, beyond just its core functionality.

Performance: The system should be highly responsive, with gestures recognition occurring within milliseconds. It should be capable of managing a substantial variety of gestures without sacrificing performance.

Accuracy: The level of accuracy in gesture recognition should be high to minimize false positives and negatives.

Scalability: The system architecture should be designed to be scalable, which allows easy interaction with new gestures and commands in future.

Usability: The gestures should have an intuitive user interface, guiding users to perform gestures and customize the gesture mapped operations.

Reliability: The system should be stable and reliable, capable of prolonged use without frequent crashes or interruptions.

3.1.2 Feasibility Analysis

Feasibility analysis entails looking at a variety of aspects to see if the project is feasible while taking technical, financial, and time constraints into account. The goal is to determine whether the project is worth pursuing considering its technical, operational, economic, legal, schedule aspects.

3.1.2.1 Technical Feasibility

- **Software Requirement:**

  Python is used for this project which consists of various modules which can be used easily and it is easier and faster compared to other languages. The OpenCV tool was developed for image processing and computer vision tasks and it is used for capturing features and display frames. “Mediapipe”, an opensource framework is used for defining pre-trained modules which will be used for detecting and tracking of hands, capturing handedness of the hand and extracting 21 key points features from the hand.

- **Hardware requirement:**

  It requires a webcam which captures gestures at the frame rate of 30 frames per second or more, with 8GB memory, 10GB secondary memory and a dedicated GPU, such as an NVIDIA GeForce GTX or higher, would enhance parallelized computations. The project would benefit from a multicore CPU with a clock speed of at least 2.5 GHz or higher.

3.1.2.2 Operational Feasibility

Our application instantly activates the webcam, once the application is launched. The webcam of the user will be taking the input by capturing the gestures. The application will recognize the hand motion once the gestures have been captured. The recognized gestures will be compared with the datasets and equivalent results is generated. The output will be according to the gestures we have trained.

3.1.2.3 Economic Feasibility

Economic feasibility measures the cost effectiveness of a project to determine if the completion of a project is possible or not with available economic resources. For this project, a laptop is required if possible, having a GPU enabled processor and it should be able to use python, OpenCV library and NumPy library. The additional cost for the project would be for the purchasing and installation of cameras if one doesn’t have a camera. With all these available resources, our project can be built economically.

3.1.2.4 Schedule Feasibility

Schedule feasibility evaluates the project’s likelihood of timely completion. By utilizing the Gantt chart, tasks, durations, and dependencies are visually mapped, aiding efficient resource allocation and project management. This well-structured schedule ensures the timely implementation of our gestures-based system, enhancing the project feasibility.

Table 3.3: Project Schedule

Task	Start Date	End Date	Days to complete
Requirement Gathering	9/18/2023	9/28/2023	10
Analysis	10/1/2023	10/7/2023	6
System Design	10/8/2023	11/8/2023	31
Implementation	11/9/2023	12/15/2023	36
Testing	12/16/2023	1/18/2024	33
Documentation	9/18/2023	1/22/2024	126

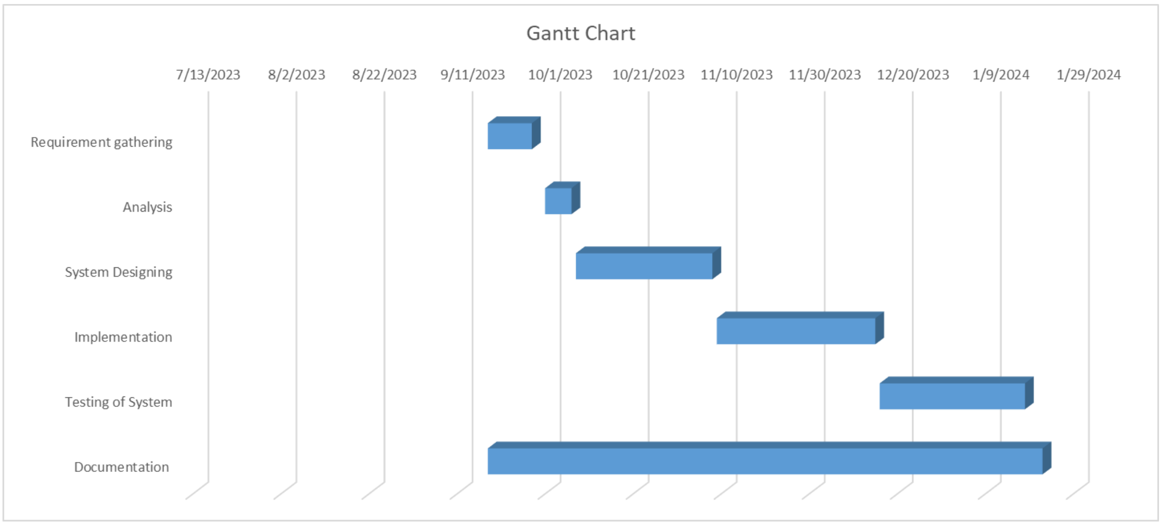


Figure 3.3: Gantt Chart

3.1.3 Object Oriented Analysis

3.1.3.1 Class Diagram

Class diagram is the static diagram. It illustrates the structure and relationships within a system or software application using classes, interfaces and their associations. Basically, it includes the information about the attributes and methods of each class, as well as associations and dependencies between each class.

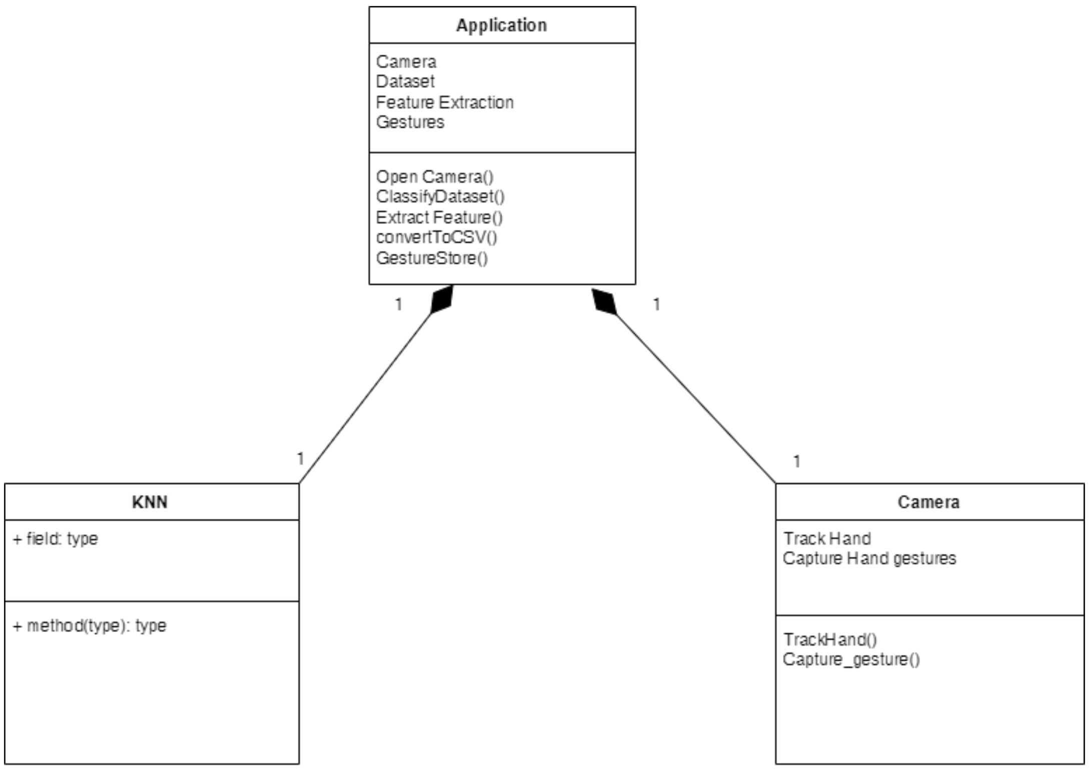


Figure 3.4: Class Diagram

The figure shows a block diagram of a system that uses a camera to capture hand gestures. This system utilizes a camera to capture hand gestures and classify them. The camera first captures video frames containing the gestures. These frames are then analyzed to track and extract the hand movements. Features are subsequently extracted from the identified gestures and converted into a CSV format for storage. Finally, a KNN classifier analyzes the features to categorize the hand gestures.

3.1.3.2 Sequence Diagram

Sequence diagram capture high-level interaction between the user of the system and the system, between the systems and other systems or between subsystems which is also known as system sequence diagrams.

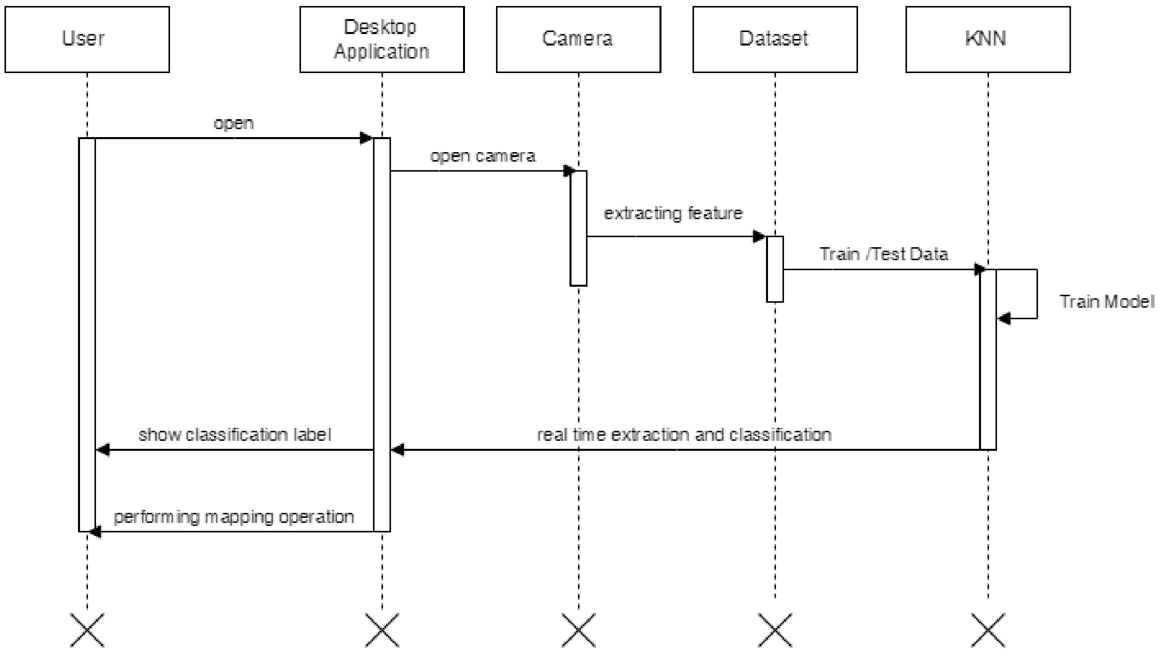


Figure 3.5: Sequence Diagram

User initiates the process by opening the desktop application. User then utilizes the camera to capture images of gestures. The captured images or retrieved data are then processed, and features are extracted. These features are subsequently used to train a model. Once trained, the model can classify new data presented to it.

3.1.3.3 Activity Diagram

Activity diagram are useful for modelling workflows, business processes and system behaviors.

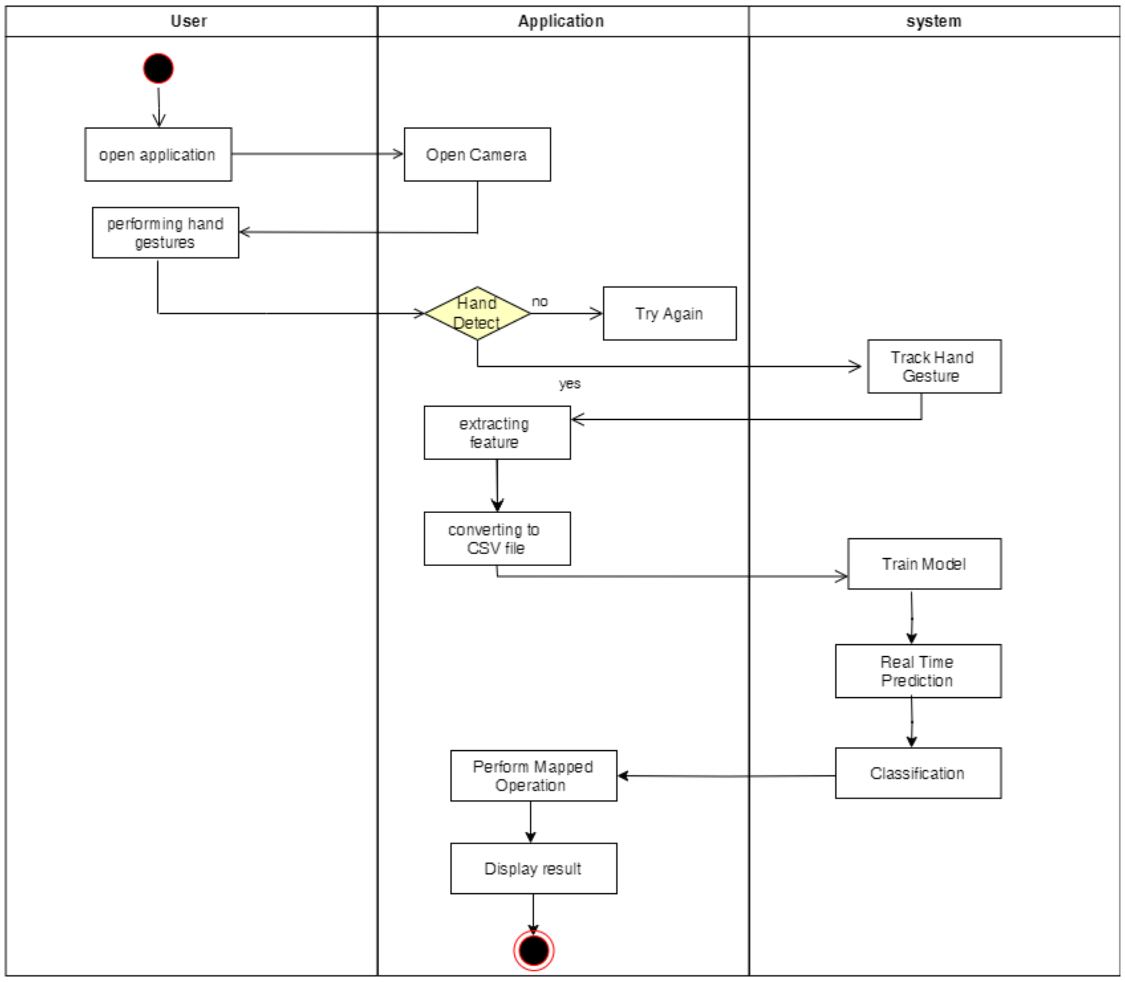


Figure 3.6: Activity Diagram

This system utilizes hand gestures for user interaction. Upon opening the application, the camera activates to capture hand movements. The system then detects the presence of a hand. If no hand is detected, the user is prompted to try again. If a hand is identified, the system tracks its movements and extracts the gestures from the video frames. Features are subsequently extracted from these gestures and saved in a CSV format. This data is then used to train a model for real-time hand gesture prediction. Based on the predicted gesture, the system performs a corresponding action and displays the result to the user.

CHAPTER 4
SYSTEM DESIGN

4.1 Design

System design is a process of designing the elements of a system such as architecture, modules, and components, various interfaces of those components and the data that goes through the system.

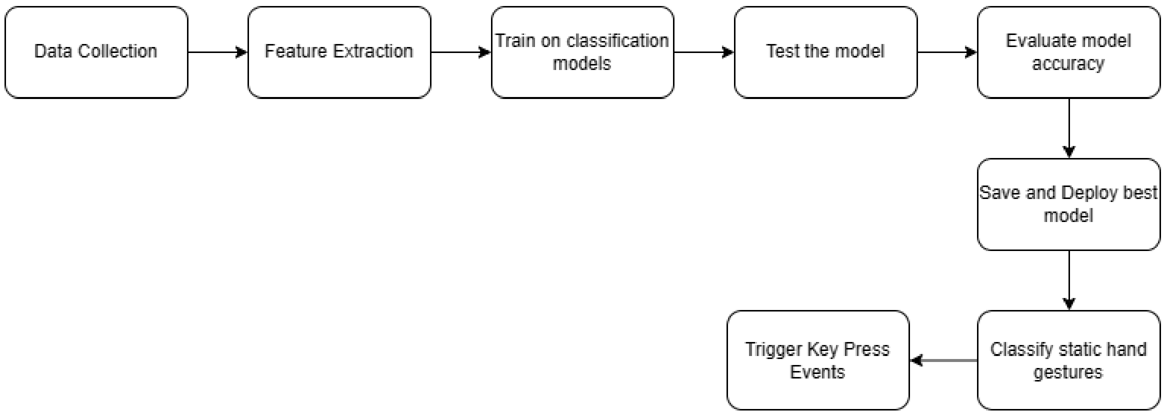


Figure 4.1: System Architecture for Personalized Gesture Recognition

The system architecture is shown in the figure 4.1. The procedure involved in the system architecture are:

- **Data Collection:**

The process begins with collecting data from the live video feed taken by the camera. This process captures the video as frames and sends the frames one at a time to the next module continuously for feature extraction.

- **Feature Extraction**

Identifying the most important and informative characteristics (features) from the raw data. Feature extraction is done by the Google's Mediapipe which provides hand landmarks points of hands on each frame. Following feature extraction, the application performs normalization to ensure all features operate on a uniform scale. This standardization step enhances the comparability of the features and bolsters the performance of the machine learning model used later.

- **Model Training**

An appropriate machine learning algorithm (e.g., decision trees, neural networks, support vector machines) based on the nature of the data and the type of problem being solved is chosen. The algorithm is fed with the data extracted in the Feature Extraction stage for training and testing purposes.

- **Model Testing**

The trained model is tested on a separate set of data (not used during training) to assess its performance. Testing is necessary for model evaluation and selection of best model.

- **Model Evaluation**

Model evaluation involves calculating metrics like accuracy, precision, recall, or others relevant to the problem to gauge the model's effectiveness.
Validation also plays a crucial role in ensuring that the model generalizes unseen data well doesn't just memorize the training set.

- **Model Deployment**

The best-performing model is stored for future use. The best performing model is then integrated with the desktop application.

- **Classification**

The model analyzes a static image of the hand and classifies it into a predefined gesture category.

- **Trigger Key Press Event**

A specific key press associated with the detected gesture is initiated by the system. This supports the effective integration and implementation of the hand gesture classification system.

4.1.1 Refinement of class diagram

Refined class diagram is the extension or modification or detailed form of a basic class diagram that provides a more comprehensive and detailed view of the system's architecture.

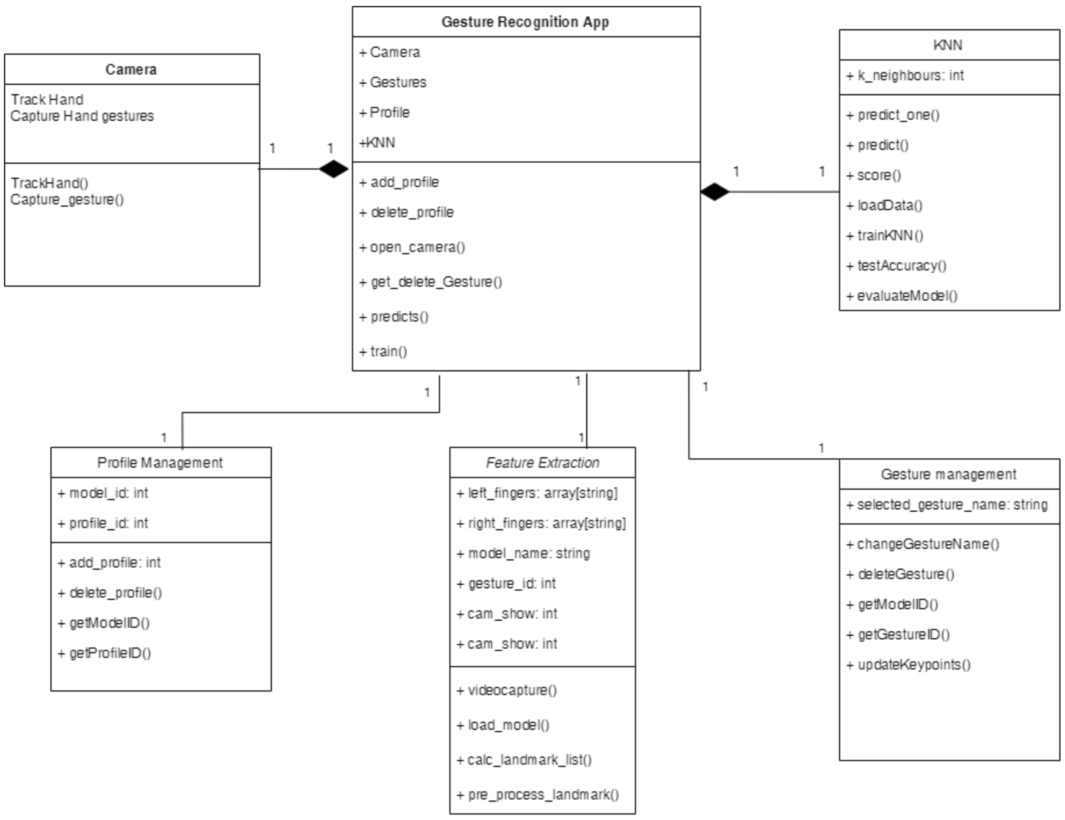


Figure 4.2: Refined Class Diagram

The class diagram depicts a gesture recognition application. It showcases the various classes and their functionalities involved in recognizing gestures. The classes and their relationships are as follows:

Classes:

GestureRecognitionApp: This class represents the main application responsible for managing the overall gesture recognition process.

Camera: This class handles interactions with the camera device, capturing video frames containing hand gestures.

Profile: This class stores user profiles including information about user preferences and recognition models.

Gestures: This class manages the different gestures the application can recognize. It also stores information about each gesture, such as its name and associated actions.

KNN: This class represents the K-nearest neighbors algorithm, a machine learning model used for gesture classification.

GestureRecognitionApp class represents the main application responsible for managing the overall gesture recognition process. It is the interface that enables the users to interact with and use the system. Profile has an "add_profile()" operation, allowing users to create profiles. Profile has a "delete_profile()" operation, enabling users to delete profiles. Profile has a "get_profile()" operation for retrieving profile information. Gestures has an "add_gesture()" operation for adding new gestures to the system's repertoire. Gestures also has a "delete_gesture()" operation for removing gestures from the system. KNN has a "train()" operation, signifying its role in training the gesture recognition model based on captured data. KNN has a "predict()" operation, enabling it to predict gestures based on new input data. Here the camera and KNN is associated with gesture recognition app.

4.1.2 Refinement of Sequence Diagram

Refined sequence diagram in UML (Unified Modelling Language) is defined as an improved or detailed form or version of the initial sequence diagram. Here, a sequence diagram is defined as the diagram which represents interactions between objects or components in a system over time.

The sequence diagram in Figure 4.3 illustrates the process of feature extraction within the project. As the user runs the application, it initiates the entire workflow. Upon opening, the application activates the camera, which continuously feeds frames to MediaPipe's landmark model. This model then extracts features from the captured frames. The application subsequently normalizes these features and saves them in a CSV file for further use.

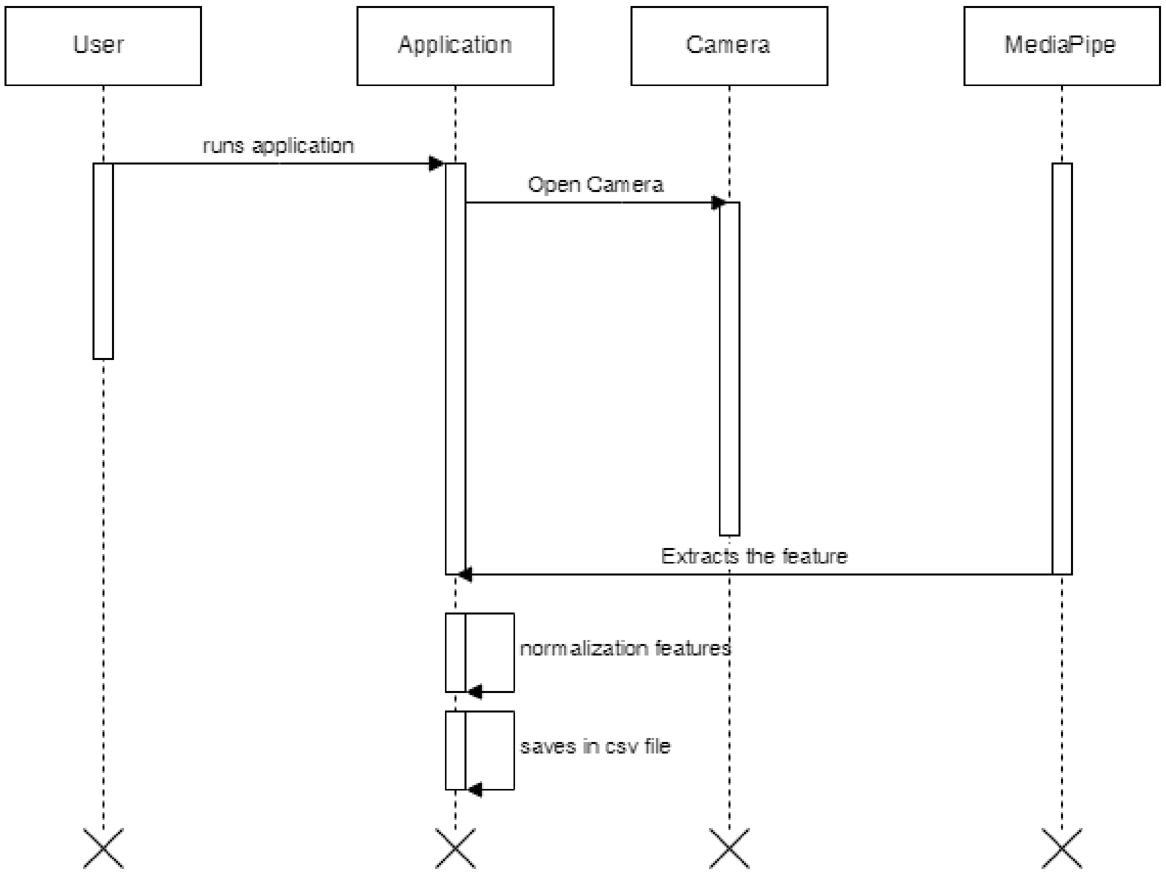


Figure 4.3: Feature Extraction Sequence Diagram

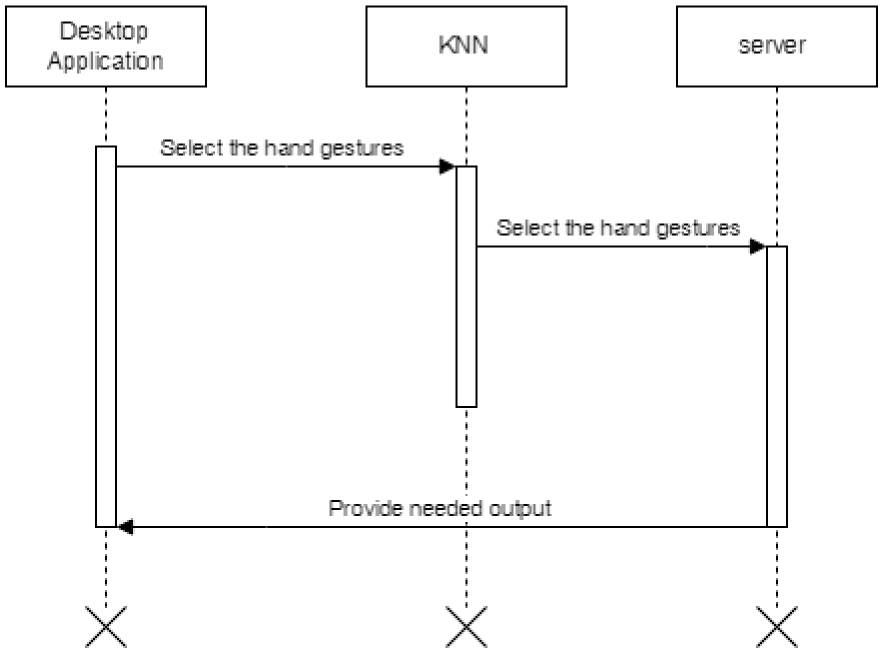


Figure 4.4: Training Sequence Diagram

The sequence diagram in Figure 4.4 depicts the model training process. As the user runs the application, it feeds the CSV file containing extracted features to the KNN algorithm, which is responsible for classification. The KNN algorithm then trains the model and generates a SAV file containing the trained model. Finally, the KNN sends this SAV file back to the application and gets stored on local disk for further use.

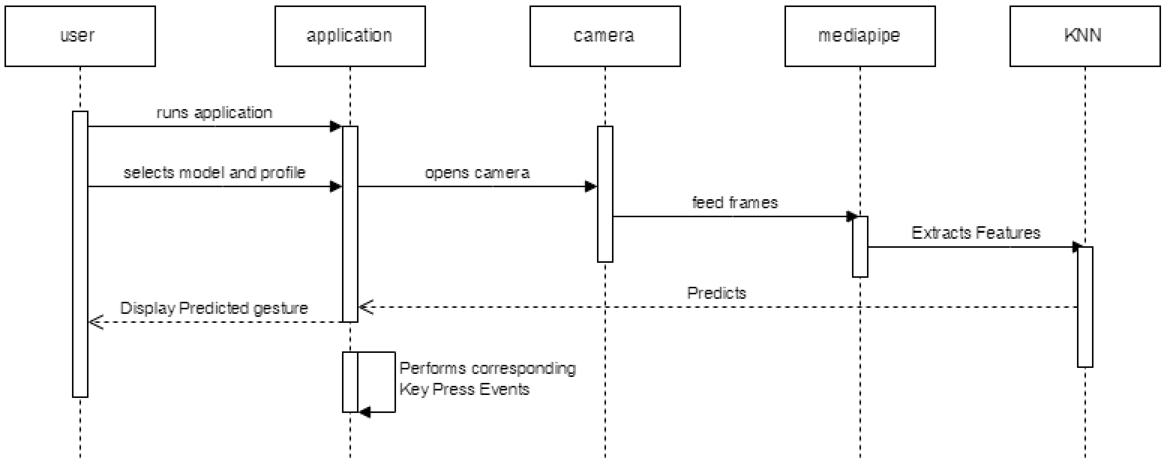


Figure 4.5: Real Time Prediction Sequence Diagram

Figure 4.5 showcases the real-time prediction sequence in the application. Upon user initiation by running the application, it activates the camera to capture video frames. These frames are then fed to MediaPipe's landmark model for feature extraction. Once features are extracted, the application sends them to the KNN algorithm for prediction. Based on the predicted gesture, the application displays the results to the user and triggers corresponding key press event. This continuous loop enables real-time interaction based on user gestures.

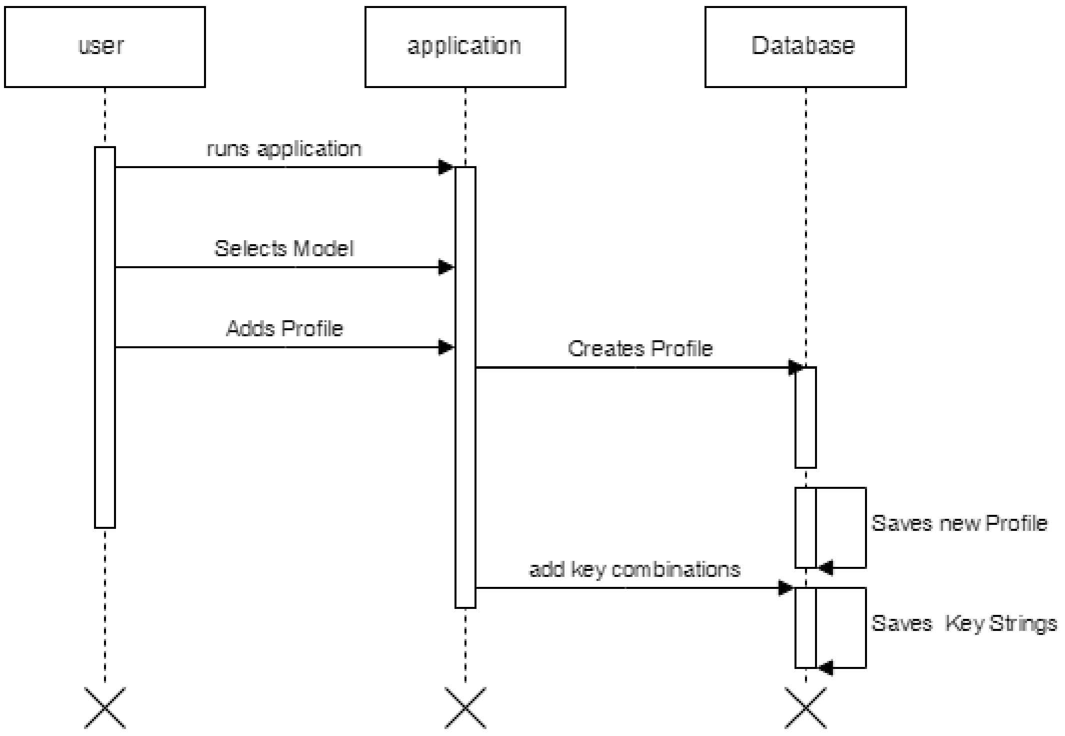


Figure 4.6: Profile Management Sequence Diagram

Figure 4.6 depicts the user profile management sequence within the application. The process begins with the user running the application. Once launched, the user can select the model to use and proceed to add a new profile to that model. This profile creation involves entering all necessary user data, which is then saved to the database. Additionally, the user can define key combinations associated with the profile, with the corresponding key strings being stored in the database alongside the profile information.

4.1.3 Refinement of Activity Diagram

Refined activity diagram in UML (Unified Modelling Language) is defined as adding more details and precision to the initial diagram. Activity diagram is used to model workflows and business processes. Refining the activity diagram is basically providing more details and precisions involved in the particular workflows and business processes.

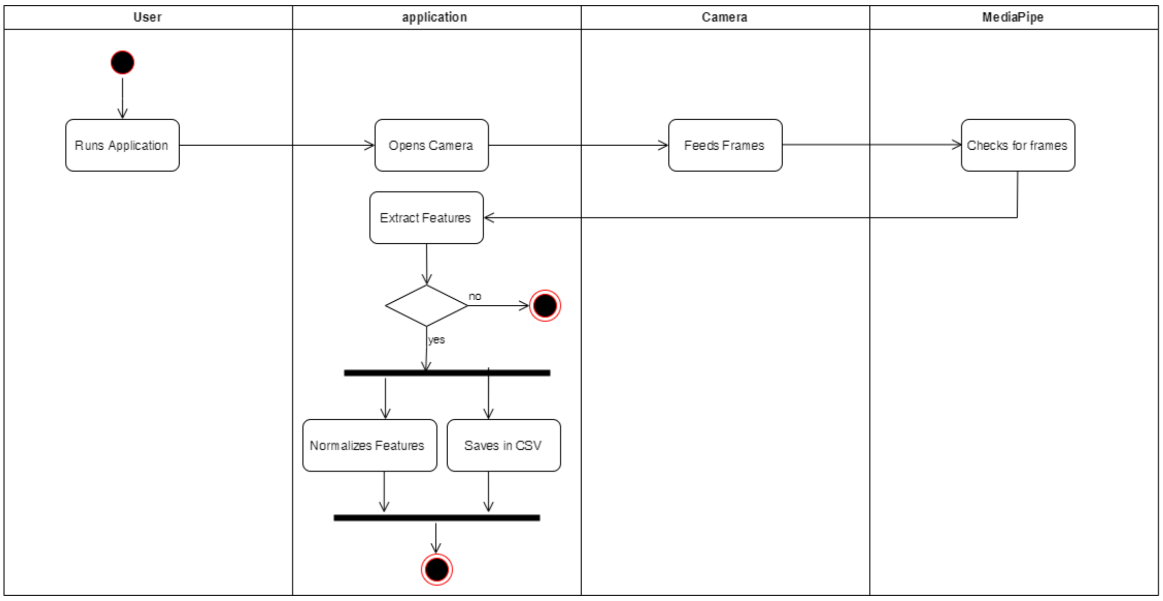


Figure 4.7: Feature Extraction Activity Diagram

This activity diagram outlines the process of extracting features from hand gestures for further analysis. The journey commences with the user initiating the application, prompting it to activate the camera for capturing hand movements. Each captured video frame is then fed to MediaPipe, a powerful tool responsible for extracting features from the image data. These features might encompass the precise locations of key hand landmarks or the angles between various hand segments.

Following feature extraction, the application performs normalization to ensure all features operate on a uniform scale. This standardization step enhances the comparability of the features and bolsters the performance of the machine learning model used later. Finally, the normalized features are saved in a CSV file, creating a valuable dataset for training a machine learning model capable of recognizing diverse hand gestures. In essence, this activity diagram captures the steps involved in extracting and preparing hand gesture features for subsequent machine learning applications.

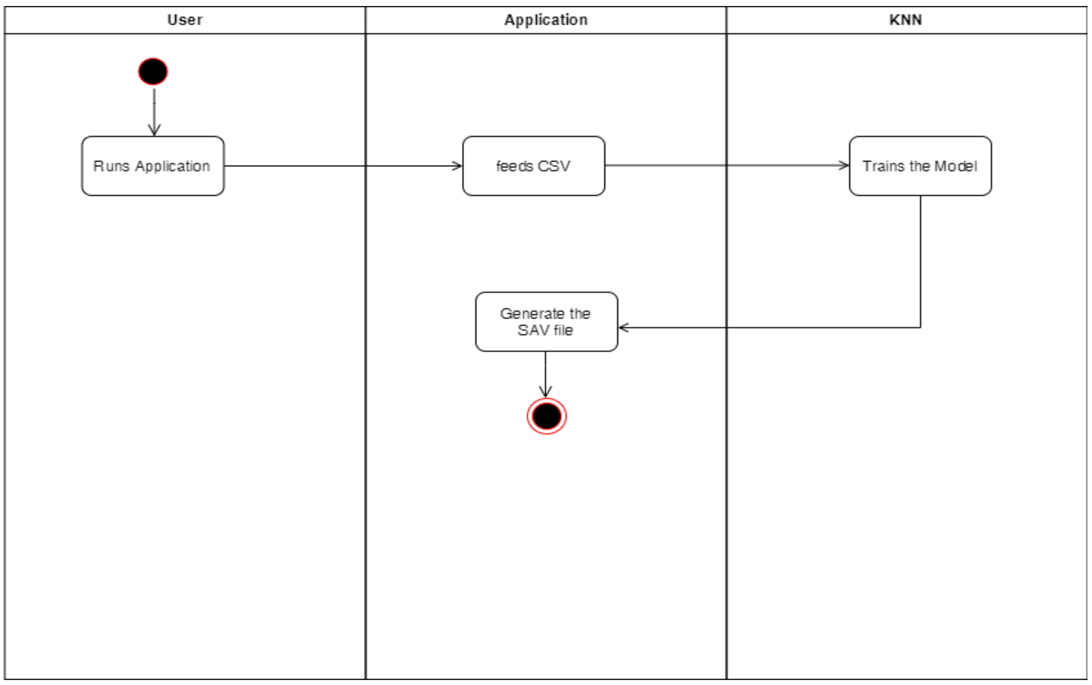


Figure 4.8: Training Model Activity Diagram

This activity diagram outlines the training process for a KNN model in gesture recognition. The process starts with the user initiating the application, prompting it to load training data from a CSV file. This data holds extracted features from various hand gestures, along with corresponding labels signifying the specific gesture each data point represents. Next, the loaded data is strategically divided into two sets: a training set and a testing set. The training set plays a vital role in training the KNN model, allowing it to learn the intricate relationships between the features and their corresponding gesture labels. Once trained, the model's performance is evaluated using the testing set. This evaluation employs metrics like accuracy, precision, and recall to assess the model's effectiveness. Finally, the model is saved to a file for future use. This process ensures the KNN model is equipped to accurately recognize various hand gestures based on the knowledge it acquired during the training phase.

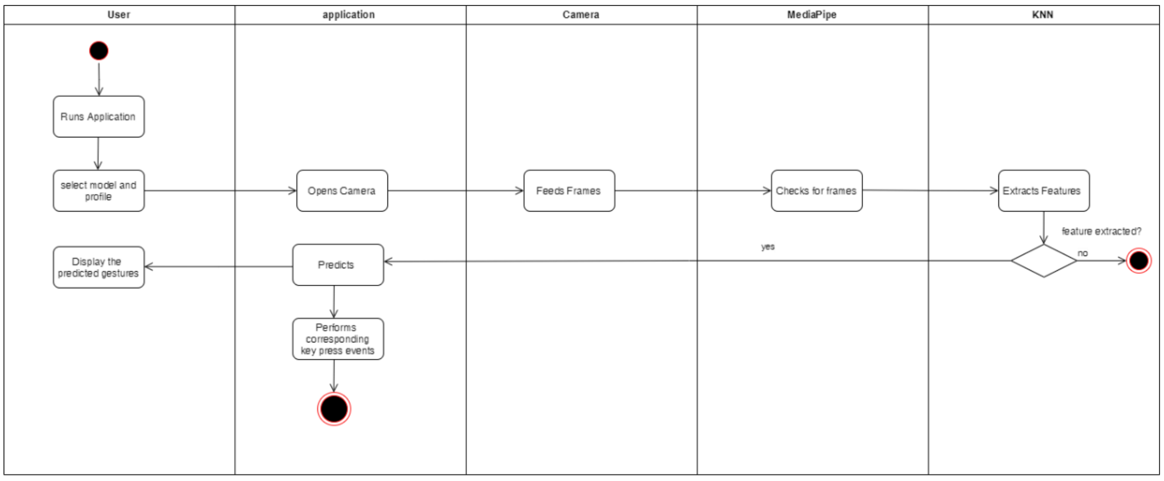


Figure 4.9: Real Time Activity Diagram

This activity diagram showcases the real-time hand gesture classification process using a K-Nearest Neighbors (KNN) algorithm. Upon user initiation, the application activates the camera to capture video frames potentially containing hand gestures. The camera continuously captures frames, which are then fed to the MediaPipe's landmarks model for feature extraction. The extracted features are then fed into the KNN algorithm. The algorithm compares the features against a vast repository of training data, identifying the nearest neighbors based on a predefined distance metric. The gesture class associated with the majority of these nearest neighbors is then predicted as the recognized gesture. Finally, the application displays the recognized gesture to the user and also triggers the corresponding key press event.

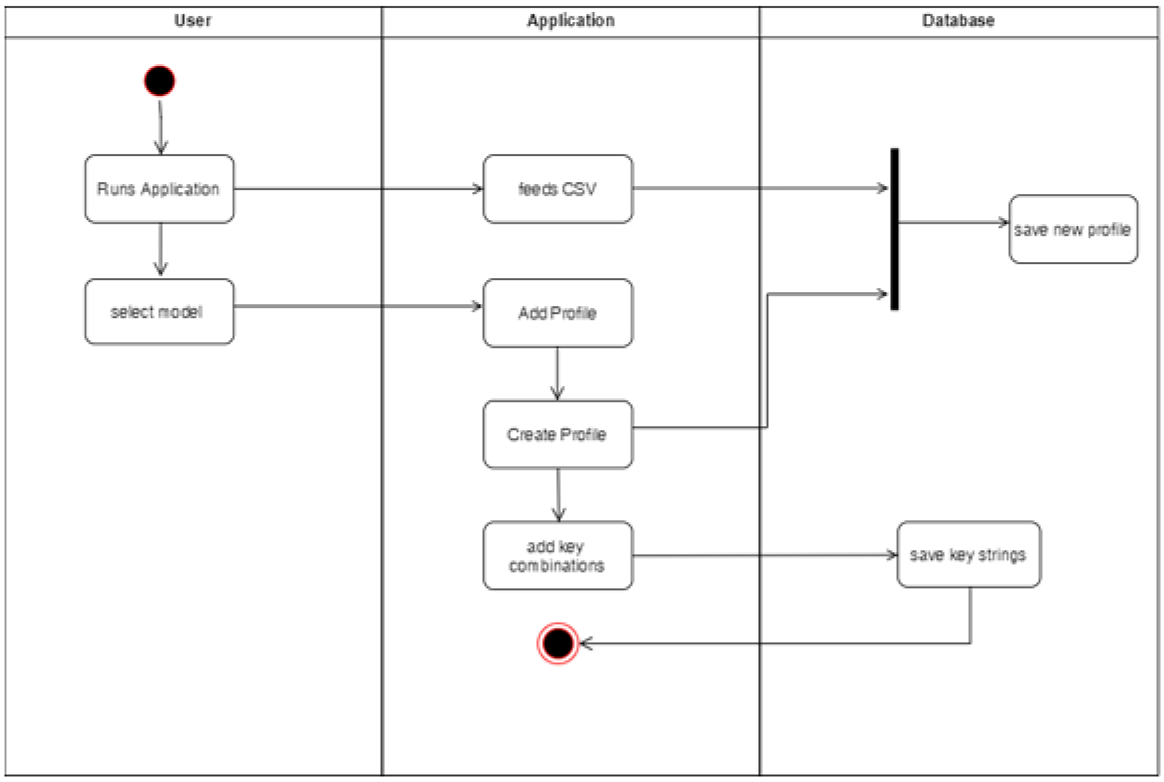


Figure 4.10: Profile Management Activity Diagram

Figure 4.10 depicts the process of managing user profiles within the application. The user initiates by launching the application, which then prompts them to select a model and proceed to create a new profile associated with that model. This profile creation involves entering essential user data alongside a selected CSV file. Both the profile information and the CSV data are subsequently saved in the database. Additionally, the user has the option to define key combinations for the profile, with the corresponding keystrokes being stored alongside the profile data within the database.

4.1.4 Component Diagram

A component diagram plays a crucial role in visualizing the static structure and interactions between different parts of a system. It depicts components as rectangular boxes, representing self-contained, reusable modules with well-defined interfaces. Lines connecting these boxes, known as connectors, symbolize the relationships and dependencies between them.

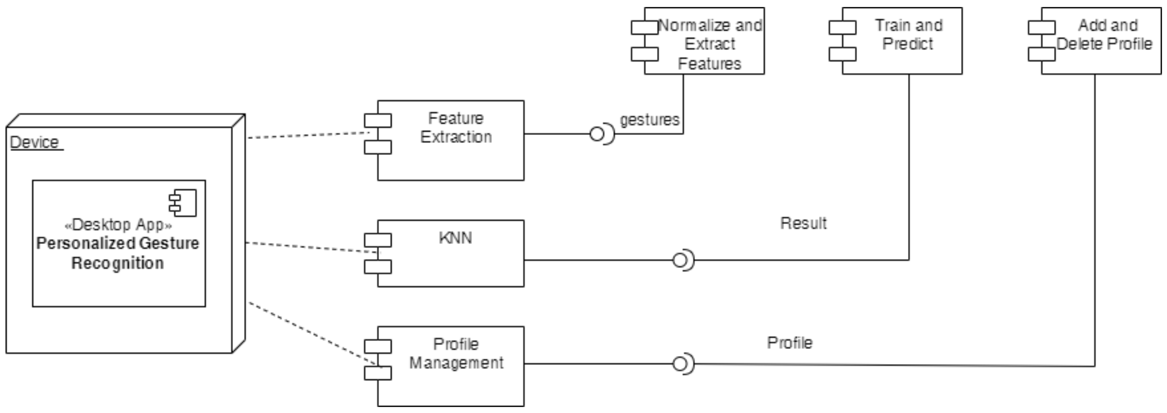


Figure 4.11: Component Diagram

The gesture recognition system captures user gestures through a device like a camera or sensor. The Desktop App, the core of the system, receives this data, extracts features, trains a personalized recognition model, and uses it to predict new gestures. Feature Extraction prepares the data for training, while Train and Add Profile creates user profiles with trained models. Prediction utilizes the models to recognize incoming gestures. Profile Management allows users to control their profiles. Finally, the KNN algorithm classifies the gestures based on the extracted features, and normalization ensures the features are on a common scale for better analysis. This diagram provides a high-level view of the system's architecture, outlining the components and their interactions.

4.1.5 Deployment Diagram

A deployment diagram offers a visual representation of a system's architecture, but from a deployment perspective. Instead of focusing on internal components and their interactions, it depicts the physical nodes (hardware, software execution environments) on which the system's software components reside and how they communicate.

Figure 4.12 showcases a personalized gesture recognition system running solely on a single desktop computer. The system itself, encompassing the gesture recognition software, resides on the desktop, interacting with a database that stores user gestures and trained recognition models. This straightforward deployment, often used for smaller-scale systems, demonstrates the software's functionality on a single machine.

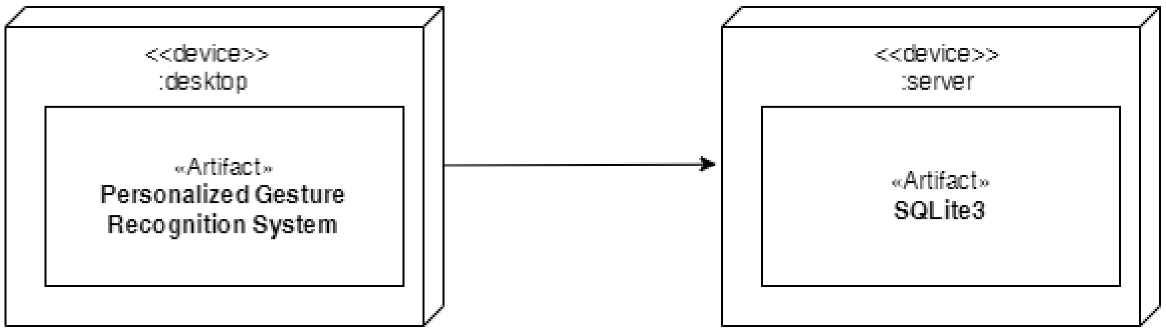


Figure 4.12: Deployment Diagram

4.2 Algorithm Description

The project implements the following algorithms:

KNN for Classification:

K-Nearest Neighbors (KNN) is a versatile tool used for both classifying and predicting values in machine learning. Unlike other methods, it doesn't rely on assumptions about the data's underlying structure. Instead, it works by finding the closest data points (k neighbors) in the training set to a new, unknown data point. The class label or average value of these neighbors is then assigned to the new point.

To use KNN effectively, two key factors to consider are:

- **Number of neighbors (k):** This value determines how many neighbors contribute to the prediction. Choosing a high k leads to smoother classifications but can be prone to errors, while a low k makes the model more sensitive to outliers. Finding the right balance is crucial.
- **Distance metric:** This metric defines how similarity between data points is measured. Euclidean distance, which calculates the straight-line distance between points, is a popular choice.

$$\boldsymbol{d(A,B)} = \sqrt{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{A_i} - \boldsymbol{B_i})^{\boldsymbol{2}}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

- **Finding the closest matches:** For a new data point to be classified, KNN searches through the training data, calculating the distance between the new point and every other point using the chosen distance metric. It identifies the k data points from the training set that are most similar to the new point.

- **Voting for a category:** Once KNN has identified the closest neighbors, the new point's category needs to be decided. Each neighbor "votes" for its own category, and the category with the most votes wins.

- **Assigning the label:** Based on the majority vote, KNN assigns the winning category label to the new data point. This means the new point is classified into the category that its closest neighbors belong to.

Decision Tree for Classification:

Decision trees are versatile machine learning algorithms used for both classification and regression tasks. They work by recursively partitioning the input space into regions, with each region associated with a specific class label. Decision trees are constructed through a process of selecting the best feature to split on at each node, based on certain criteria.

To use a Decision Tree effectively, the key components and steps involved are:

Tree Construction:

The decision tree starts with the root node, which represents the entire dataset. At each node, the algorithm selects the feature that provides the best split. This decision is made based on a measure of impurity or information gain. The dataset is split into subsets based on the chosen feature and threshold. This process is repeated recursively for each subset until a stopping criterion is met.

Impurity Measure:

Common impurity measures include Gini impurity. Gini impurity measures the likelihood of misclassifying a randomly chosen element. The algorithm aims to minimize impurity at each node, leading to more homogeneous subsets.

The Gini impurity for a node t with K classes is calculated as follows:

$$\boldsymbol{G(t) = 1 - \sum_{i=1}^{k} (p_i)^2} \quad \text{................................(2)}$$

Where p_i is the proportion of samples in class i at node t.

$$\boldsymbol{Impurity\ Reduction = Current\ Impurity - \frac{\sum_{i=1}^{m} Impurity(T_i) * |T_i|}{|T|}} \quad (3)$$

Where Impurity(T_i) is the impurity of the i-th child node, |(T_i)| is the number of samples in the ith child node, and |T| is the total number of samples.

Stopping Criteria:

The tree-building process stops when a predefined stopping criterion is met. This can include reaching a maximum depth, having a minimum number of samples in a node, or achieving pure leaf nodes (all samples belong to the same class).

Prediction:

To classify a new data point, it traverses the tree from the root to a leaf node based on the feature values of the data point. Each leaf node represents a class label, and the majority class in that leaf is assigned to the new data point.

CHAPTER 5
IMPLEMENTATION AND DESIGN

5.1 Implementation Overview

Incremental delivery is a project management approach that involves dividing a project into smaller, more manageable parts and delivering these components gradually over time. Each increment builds upon the previous one, introducing new features and functionality until the final product is fully developed. As part of the Agile methodology, incremental delivery places a strong emphasis on collaboration among developers, stakeholders, and customers. The development team closely engages with the customer to identify their needs and requirements, enabling the delivery of functional software in a quick and frequent manner. This iterative process allows customers to witness ongoing progress and provide feedback, which is then incorporated into subsequent iterations.

5.1.1 Tools Used

5.1.1.1 Front-End Tools

The front-end of the Gesture Recognition project was developed using Python and PyQt5, a robust GUI toolkit. PyQt5 was chosen for its capability to create interactive and visually appealing user interfaces. Leveraging Python's versatility, the project achieved a user-friendly front-end with PyQt5 handling the design and layout aspects. Additionally, the implementation involved making use of Python scripts for the core functionality, ensuring an efficient and responsive user experience. The integration of PyQt5 in this project facilitates seamless interaction and visualization for users engaging with the Gesture Recognition application.

5.1.1.2 Back-End Tools

The Gesture Recognition project's backend utilized a robust toolset for optimal functionality. Python served as the primary programming language, providing the flexibility and power required for backend operations. SQLite3 was employed for efficient data storage and retrieval in database management.

In hand detection and feature extraction, the project leveraged Mediapipe, a comprehensive computer vision library. Mediapipe facilitated precise hand detection and extraction of crucial features for gesture recognition.

OpenCV, the Open Source Computer Vision Library, played a crucial role in image processing tasks. Its functionalities were key to enhancing the accuracy and efficiency of various computer vision operations, contributing significantly to the overall success of the Gesture Recognition project.

5.1.2 Modules Description

This project implements following modules:

Process Module:

The process module is responsible for the feature extraction process which utilizes the hand detection and landmarks models provided by the Google's Mediapipe framework. The process module captures the frames from the live video captured through a web cam or a built-in camera. The frames captured are feed to the pre-trained models to detect the hands and to provide the coordinates of the landmarks detected on the hand by the model. The two-dimensional landmarks are flattened and then normalized.

The figure 5.1 showcases the 21 landmarks localized by the landmarks model on the detected hands.

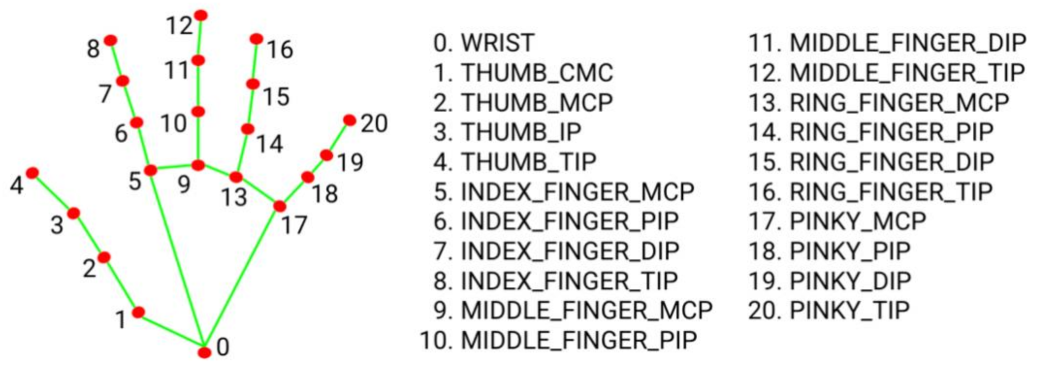
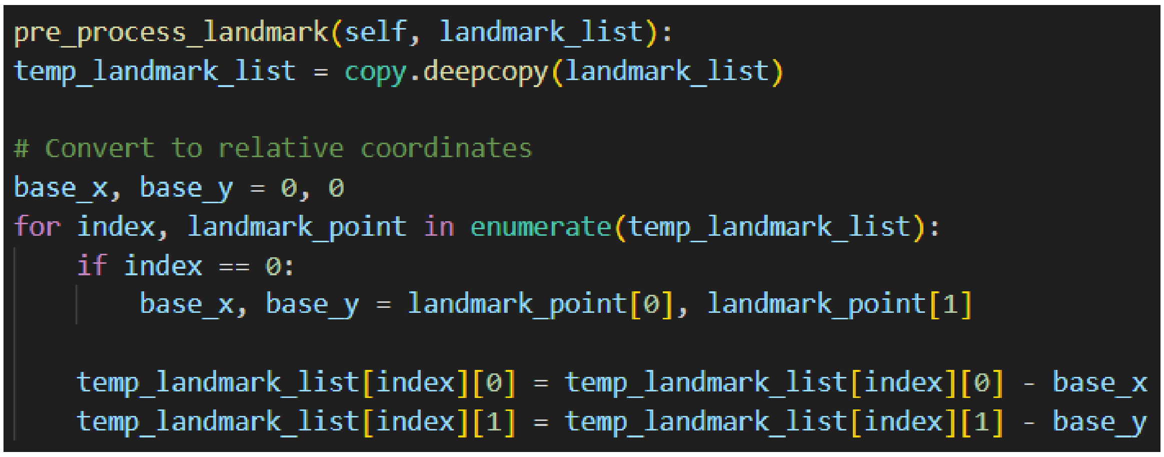


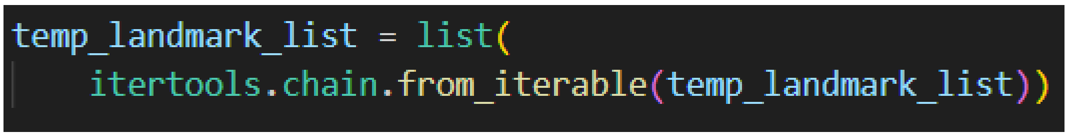
Figure 5.1: Landmarks model on detected hands

The preprocessing is done in three steps. First, the landmarks points are converted to relative coordinates. Then the relative coordinates are further processed to convert the two-dimensional points into one-dimensional list and then normalized.



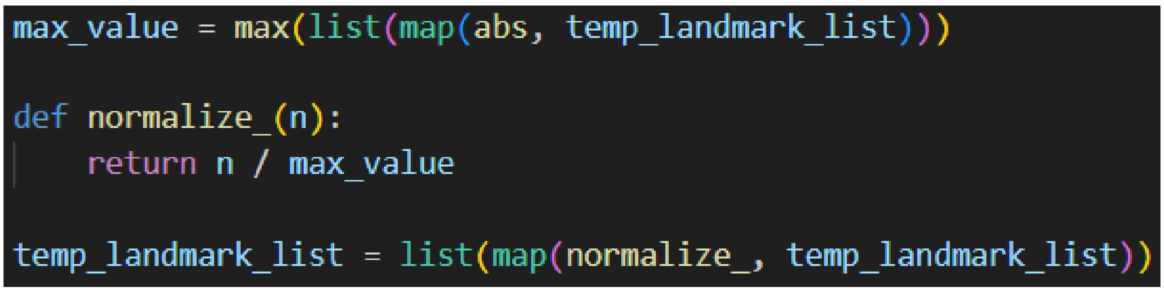
```
pre_process_landmark(self, landmark_list):
temp_landmark_list = copy.deepcopy(landmark_list)

# Convert to relative coordinates
base_x, base_y = 0, 0
for index, landmark_point in enumerate(temp_landmark_list):
    if index == 0:
        base_x, base_y = landmark_point[0], landmark_point[1]

    temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
    temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y
```

Figure 5.2: Converting to relative coordinates



```
temp_landmark_list = list(
    itertools.chain.from_iterable(temp_landmark_list))
```

Figure 5.3: Converting to single dimensional list



```
max_value = max(list(map(abs, temp_landmark_list)))

def normalize_(n):
    return n / max_value

temp_landmark_list = list(map(normalize_, temp_landmark_list))
```

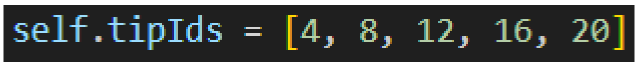
Figure 5.4: Final normalization process

After preprocessing the dataset, the dataset is saved into a csv file. This process is repeated for each frame captured by the camera.

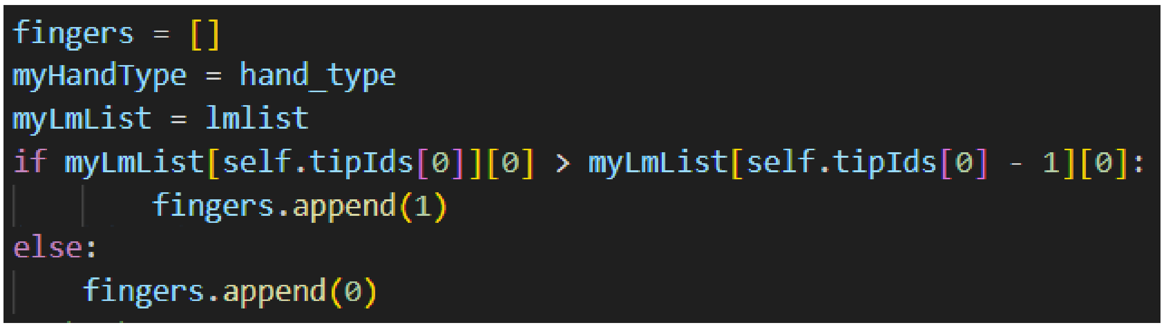
Fingers Tracking Module:

Fingers Tracking Module serves a very important role in the project. This module allows tracking of fingers that are lifted up. The current implementation only allows tracking of left-hand fingers. The significance of this model is that it allows one to track the numbers of fingers lifted and the order that the fingers are lifted which is utilized in the project for gesture recognition activation. The reason why this module is included in the project is to only allow gesture recognition when intended by the user.

To track if thumb of left hand is up, x coordinate of 4th landmark point (tip of thumb) is compared with 3rd landmark point and the status of thumb is calculated.



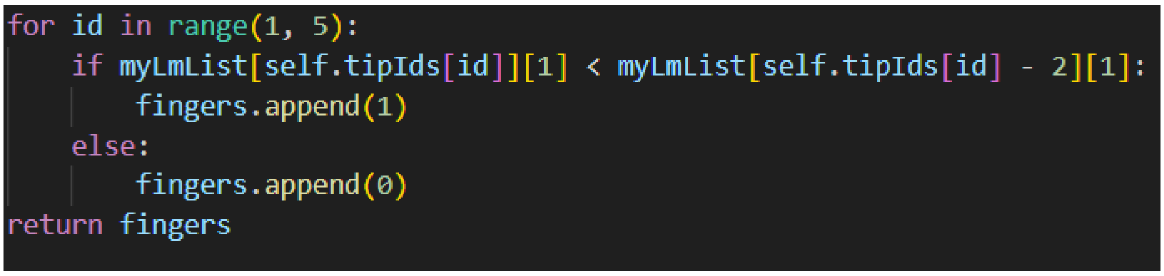
```
self.tipIds = [4, 8, 12, 16, 20]
```



```
fingers = []
myHandType = hand_type
myLmList = lmlist
if myLmList[self.tipIds[0]][0] > myLmList[self.tipIds[0] - 1][0]:
        fingers.append(1)
else:
    fingers.append(0)
```

Figure 5.5: Thumb Tracking

To track if other fingers of left hand are up, y coordinate of corresponding tips of the fingers are compared with corresponding coordinates of lower landmarks.



```
for id in range(1, 5):
    if myLmList[self.tipIds[id]][1] < myLmList[self.tipIds[id] - 2][1]:
        fingers.append(1)
    else:
        fingers.append(0)
return fingers
```

Figure 5.6: Finger Tracking

The module will output a list of 5 binary numbers where 1 represents the finger is up, 0 represents the finger is down.

r

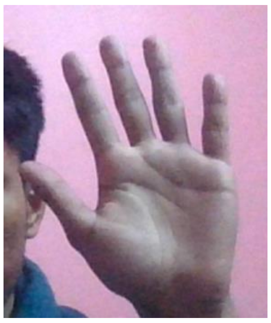


Figure 5.7: [1, 1, 1, 1, 1] shows all fingers are up

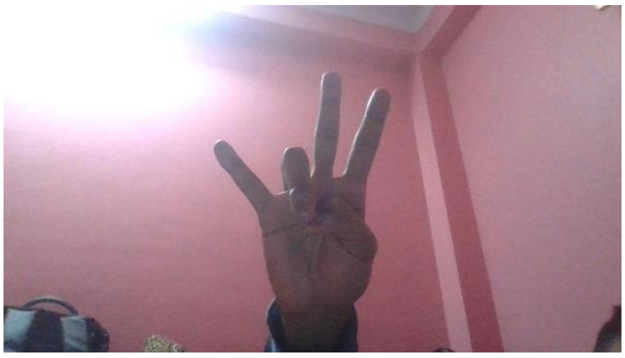


Figure 5.8: [0, 1, 1, 0, 1] shows respective fingers are up

- **Home-Tab:**

This module allows user to select the model, the profile and the camera option to whether display or not to display the captured camera frames on the screen. This module is used to start the overall program of recognizing gestures**.**

Selecting Profiles: Profiles are model dependent. It means that each model can have multiple profiles with multiple key configurations assigned to each gesture. This allows reusability of model for different key configurations.

Camera Options: Camera options allows users to turn their camera on or off during the runtime of the application. Allowing turning on and off the camera adds to the user customizability. Turning camera on is recommended for the project for the ease of use.

- **Features-Tab:**

This module allows user to navigate through available models, add new models, delete existing models, add features to existing models and train the models. Adding new model creates a csv file for saving extracted features. Training the model creates a .sav file.

Add Model: Add Model option allows users to name their model before adding features and training the model. User is prompted with a dialog box where the user needs to enter the name of their new model. Validations are included such that only unique names for a new model are accepted by the system

Delete Model: Delete Model option allows users to delete an existing model, whether it be a trained model or an untrained model. Deleting a model deletes all it profiles, gestures and key configurations as well. Before deleting a model, user needs to select the model from the list.

Add Features: Add Features option allows adding new gestures to a new model or an existing model. Clicking on the add features button after selecting a model turns on the camera for frame capture process. The frames are captured when user presses the 'q' button on their keyboard. This frame capture process is followed by feature extraction and preprocessing of the landmarks done by the 'Process' module discussed in the previous sections. The features extracted are continuously being updated on a csv file. After a satisfactory number of frames are captured, the user is prompted with providing a gesture name to the recent gesture captured. If user selects 'Cancel', the gesture features are discarded and the process is terminated.

Train: The train option allows training of model if the features have been added to the model. The training of model starts by feeding the values in the respective csv file to the KNN algorithm which then generates a trained model within less than a second with evaluation details in the console panel. This model is then ready to use.

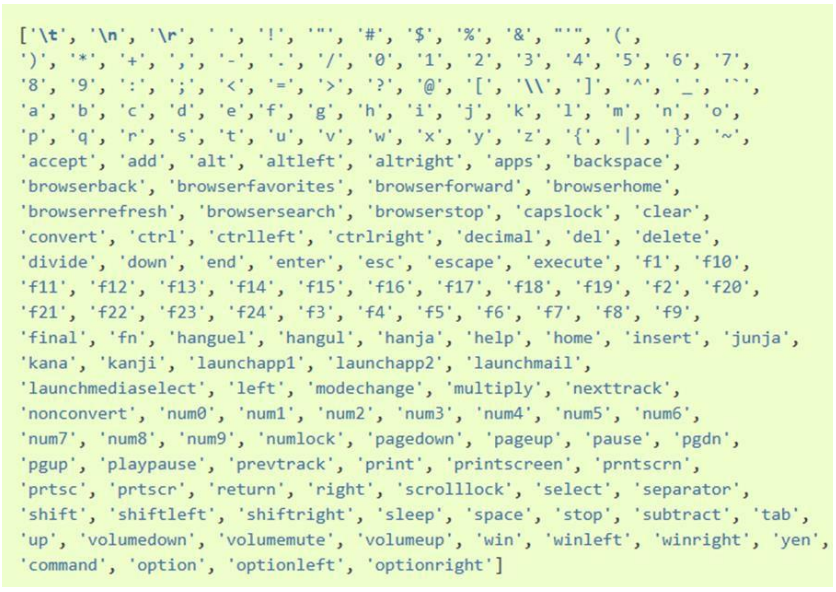
- **Profiles-Tab:**

This module allows users to view all the models and their corresponding profiles, add new profiles to a model, add key combinations to the available gestures allowing triggering the key combinations when performing gestures. This allows customizability to the user.

Select Model: Select model displays all the trained models available.

Select Profile: If the selected model is a new model, then the model will not have any profiles. User needs to create a new profile before proceeding forward to setting key combination to gestures.

Key Combinations: Key Combinations refers to the set of keys that will be pressed when a gesture is detected. Specific gestures can carry unique key combinations. Key combinations are to be assigned one key at a time with a space in between the keys. The following keys are supported:
To set a key combination, select the gesture from the list with which the key combination is to be configured. For example: If one wants to type the name "RAM" using a gesture, key should be entered in the following format: R A M, with a space in between every letter. After clicking on set button, the key combination is set for the gesture. In a similar fashion, all the available gestures can be assigned with corresponding key combinations.



```
['\t', '\n', '\r', ' ', '!', '"', '#', '$', '%', '&', "'", '(',
')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`',
'a', 'b', 'c', 'd', 'e','f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~',
'accept', 'add', 'alt', 'altleft', 'altright', 'apps', 'backspace',
'browserback', 'browserfavorites', 'browserforward', 'browserhome',
'browserrefresh', 'browsersearch', 'browserstop', 'capslock', 'clear',
'convert', 'ctrl', 'ctrlleft', 'ctrlright', 'decimal', 'del', 'delete',
'divide', 'down', 'end', 'enter', 'esc', 'escape', 'execute', 'f1', 'f10',
'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f2', 'f20',
'f21', 'f22', 'f23', 'f24', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9',
'final', 'fn', 'hanguel', 'hangul', 'hanja', 'help', 'home', 'insert', 'junja',
'kana', 'kanji', 'launchapp1', 'launchapp2', 'launchmail',
'launchmediaselect', 'left', 'modechange', 'multiply', 'nexttrack',
'nonconvert', 'num0', 'num1', 'num2', 'num3', 'num4', 'num5', 'num6',
'num7', 'num8', 'num9', 'numlock', 'pagedown', 'pageup', 'pause', 'pgdn',
'pgup', 'playpause', 'prevtrack', 'print', 'printscreen', 'prntscrn',
'prtsc', 'prtscr', 'return', 'right', 'scrolllock', 'select', 'separator',
'shift', 'shiftleft', 'shiftright', 'sleep', 'space', 'stop', 'subtract', 'tab',
'up', 'volumedown', 'volumemute', 'volumeup', 'win', 'winleft', 'winright', 'yen',
'command', 'option', 'optionleft', 'optionright']
```

Figure 5.9: Various Key Combinations

- **Gestures-Tab:**

This module allows user update and delete the existing gestures in a model. Updating the gestures renames the gesture and deleting the gesture deletes the gesture from the database requiring to re-train the edited model again

5.2 Testing

Testing involves assessing and confirming the functionality of developed software or applications. It aims to determine whether there is alignment between the actual outcomes and the anticipated results. This evaluation process occurs throughout the software development stages.

5.2.1 Unit Testing

Unit testing is a component of the testing methodology that involves the examination of individual software modules and the components constituting the entire software. The objective is to verify the performance of each unit within the software code, ensuring it functions as anticipated.

Table 5.1: Unit Testing

S. N.	Module	Test Cases	Test Description	Action Performed	Expected Result	Actual Result	Remarks
1	Home_Tab	TC-1	Check whether the application runs with camera option on 'Hide'	Chose camera option 'Hide' and hit the 'Turn On' button	Application runs without displaying captured camera frames on the screen	Application ran without displaying captured camera frames on the screen	Pass
2	Home_Tab	TC-2	Check whether the application runs with camera	Chose camera option 'Show' and hit	Application runs by displaying captured camera	Application ran by displaying captured camera	Pass

			option on 'Show'	the 'Turn On' button	frames on the screen	frames on the screen	
3	Home_Tab	TC-3	Check whether all the trained models and profiles are displayed in the drop down options	Clicked on both dropdowns to check if any model and their profiles are missing	All trained models and profiles are displayed in the drop down	All trained models and profiles were displayed in the drop down	Pass
4	Feature_Tab	TC-4	Check whether adding new model with duplicate name validations work or not	Clicked on 'Add New Model' button and entered the name of the new model	New model is added added with validation checks	New model was added with validation checks	Pass
5	Feature_Tab	TC-5	Check whether deleting any model from the list	Clicked on 'Delete Model' button	The selected model is deleted	The selected model was deleted	Pass

			deletes the model from the database		from the database	from the database	
6	Feature_Tab	TC-6	Check whether hitting the 'Add Features' button opens up camera for capturing and extracting features	Clicked on 'Add Features' button	The camera runs and displays the captured camera frames on screen	The camera ran and displayed the captured camera frames on screen	Pass
7	Feature_Tab	TC-7	Check whether the features are captured and stored in a csv after opening the camera to add features	Click on 'q' button to capture features and add to the corresponding model's csv	The extracted features are added to the corresponding csv along with the corresponding gesture_id	The extracted features were added to the corresponding csv along with the corresponding gesture_id	Pass

8	Feature_Tab	TC-8	Check whether hitting the 'Train' button works if the csv with extracted features is empty	Hit the 'Train' button	A warning message dialogue box appears displaying an error message	A warning message dialogue box appeared displaying an error message	Pass
9	Feature_Tab	TC-9	Check whether hitting the 'Train' button generates a .sav file and stores it in the predefined path	Hit the 'Train' button	A .sav file is generated and saved in the predefined path along with the training output in the console	A .sav file was generated and saved in the predefined path along with the training output in the console	Pass
10	Profile_Tab	TC-10	Check whether all the trained models and their profiles are displayed in the drop down	Clicked on both dropdowns to check if any model and their profiles	All trained models and their profiles are displayed in the drop down	All trained models and their profiles were displayed in the drop down	Pass

				are missing			
11	Profile_Tab	TC-11	Check whether adding a new profile with validations works	Clicked on the 'Add New Profile' button and entered the name of the new profile	New profile is added with validation checks	New profile was added with validation checks	Pass
12	Profile_Tab	TC-12	Check whether Key Combinations can be set to the selected gesture with validation checks	Typed in the Key combination field and hit the 'Set' button to set the key combination to the selected gesture	Key Combination is assigned to the selected gesture with validation checks	Key Combination was assigned to the selected gesture with validation checks	Pass

13	Gestures_ Tab	TC-13	Check whether selected gesture name can be updated	Selected the gesture from the table and typed the new gesture name and hit the 'Update' button	Selected gesture is renamed with validation checks	Selected gesture was renamed with validation checks	Pass
14	Gestures_ Tab	TC-14	Check whether selected gesture is deleted from database when the 'Delete' button is hit	Selected the gesture from the table and hit the 'Delete' button	Selected gesture is deleted from the database	Selected gesture was deleted from the database	Pass

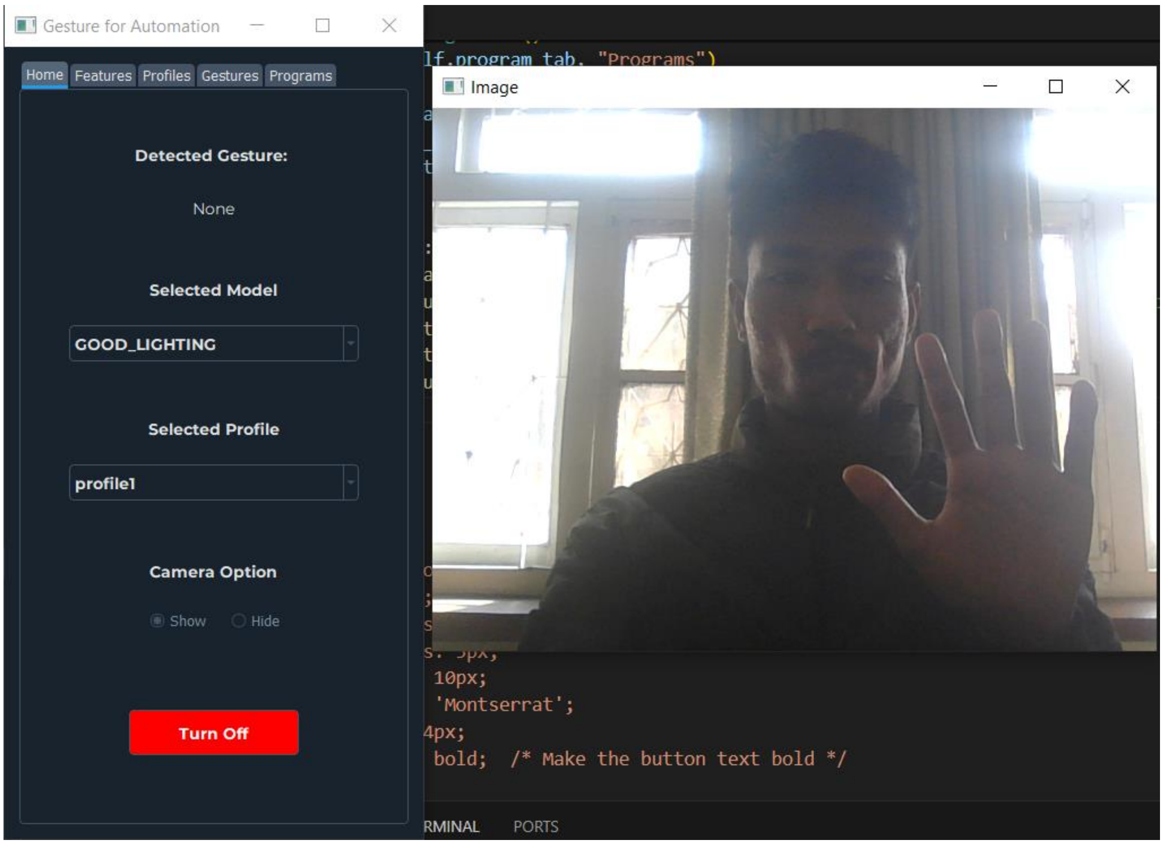
5.2.2 System Testing

System testing is a procedure to confirm that a software system aligns with the designated requirements and operates as intended. This assessment encompasses the entire system, ensuring its accurate and proper functioning.

Table 5.2: System Testing

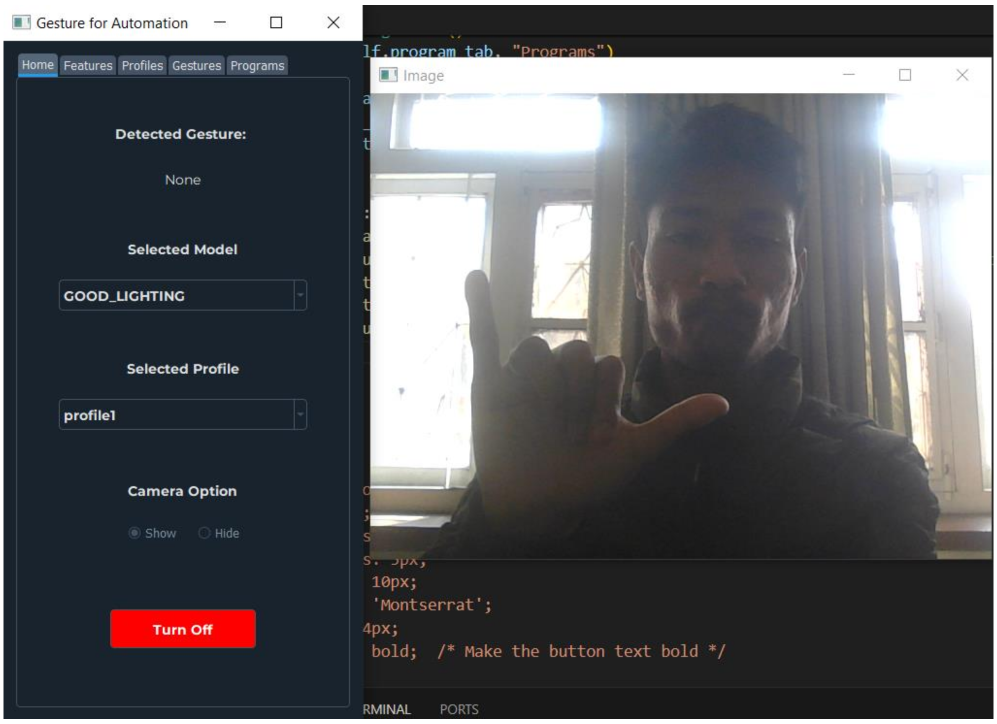
S.N.	Module	Test Cases	Test Description	Expected Outcome	Actual Outcome
1	Home_Tab	TC-1	User Flow: Select Model, Select Profile, Select Camera Option, Turn On	Opens up the camera with selected options for gesture recognition	Opened up the camera with selected options for gesture recognition
2	Features_Tab	TC-2	User Flow: Add New Model, Name the model	Saves the new model in the database after validation checks	Saved the new model in the database after validation checks
3	Features_Tab	TC-3	User Flow: Select Model, Delete model, Confirm choice	Deletes the selected model from the database	Deleted the selected model from the database
4	Features_Tab	TC-4	User Flow: Select Model, Add Feature, Provide new Gesture name		Opened up the camera for feature extraction and provided option to name the new gesture after feature extraction

5	Features_Tab	TC-5	User Flow: Select Model, Train	Selected model is trained and new .sav file is generated in predefined path	Selected model was trained and new .sav file was generated in the predefined path
6	Features_Tab	TC-6	User Flow: Select model, Name the profile, set profile	New profile of selected model is saved	New profile of selected mode was saved
7	Features_Tab	TC-7	User Flow: Select Model, Select Profile, Select Gesture, Set key combination	Key combination for selected gesture is added	Key combination for selected gesture was added
8	Gestures_Tab	TC-8	User Flow: Select Model, Select Gesture, Update	New name is assigned to selected gesture	New name was added to selected gesture
9	Gestures_Tab	TC-9	User Flow: Select Model, Select Gesture, Delete	Selected gesture is deleted from the database	Selected gesture was deleted from the database



```
lf.program_tab, "Programs")
 10px;
 'Montserrat';
4px;
 bold;  /* Make the button text bold */
```

Figure 5.10: System Testing for STC-1 (system runs correctly but no gesture is detected as intended)



Detected Gesture:

None

Selected Model

GOOD_LIGHTING

Selected Profile

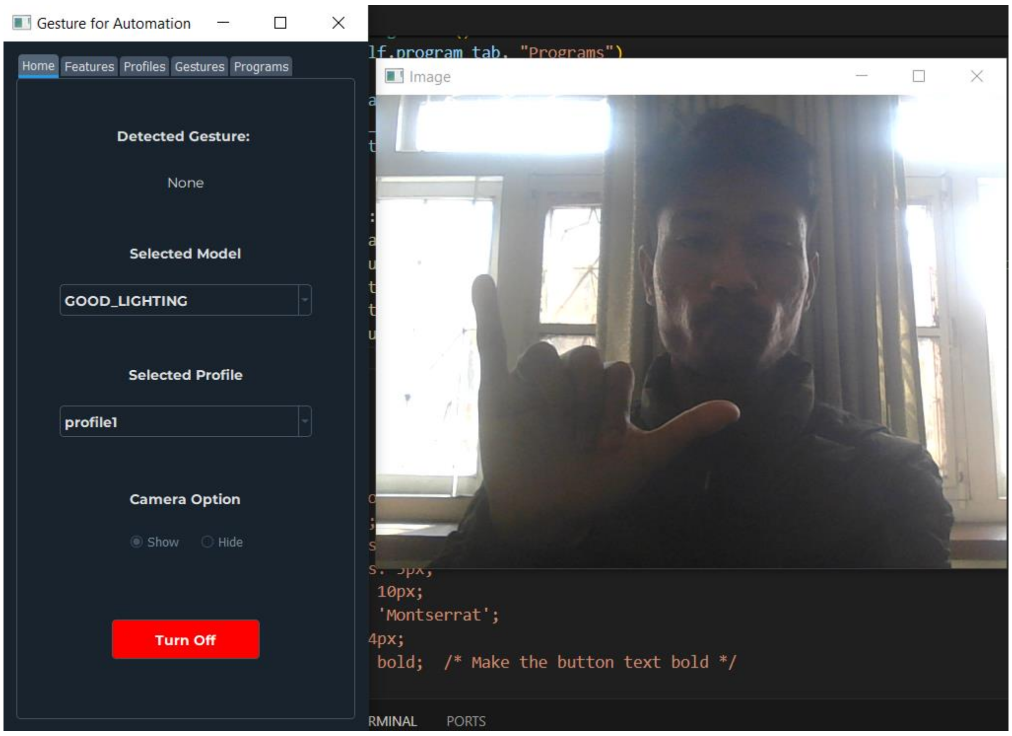
profile1

Camera Option

[✓ Show] Hide

```
lf.program_tab, "Programs")
 10px;
 'Montserrat';
4px;
 bold;  /* Make the button text bold */
```

Figure 5.11: System Testing for user showing activation gesture for gesture recognition.



Detected Gesture:

None

Selected Model

GOOD_LIGHTING

Selected Profile

profile1

Camera Option

[✓ Show] Hide

```
lf.program_tab, "Programs")
 10px;
 'Montserrat';
4px;
 bold;  /* Make the button text bold */
```

Figure 5.12: System Testing for gesture detection after activation

5.2.3 Performance Testing

5.2.3.1 Performance Testing Under Good Light Condition:

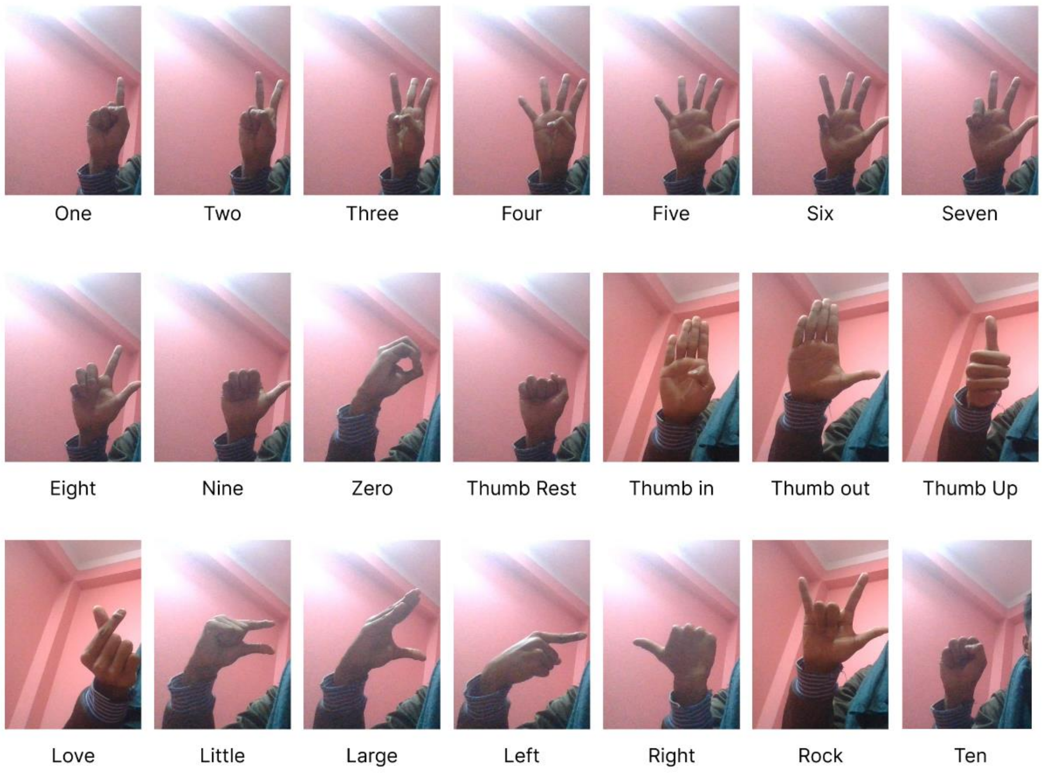


Figure 5.13: Performance Testing Under Good Light Condition

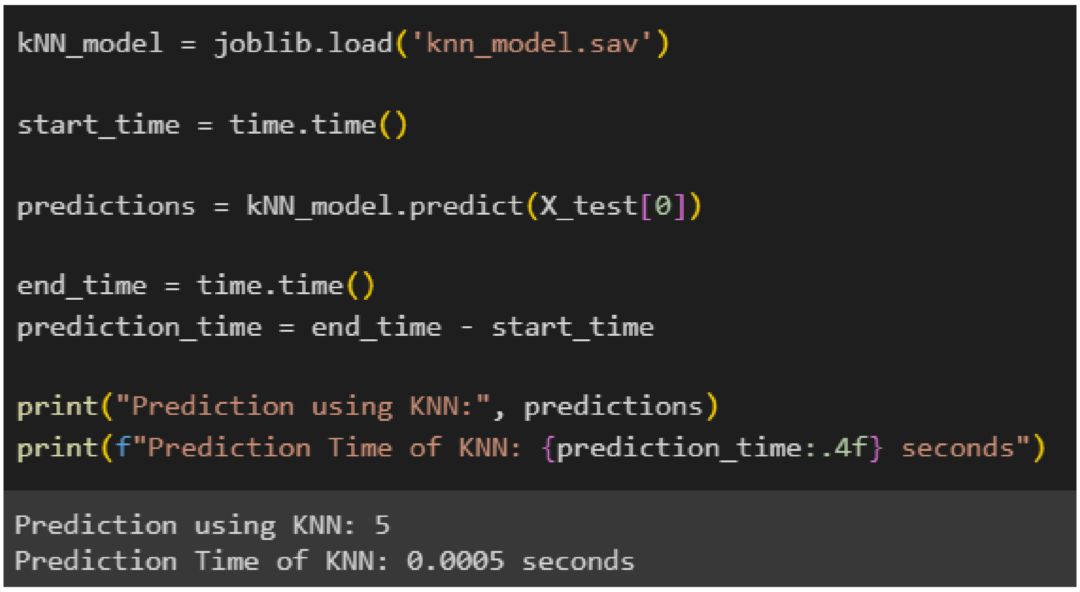
Each class consists of approximately 45 instances where 75% of the dataset is separated for training purpose and 25% of the dataset is used for testing purposes. These images are captured under good lighting conditions and the prediction results are compared between the two algorithms in the next section.

5.2.3.2 Comparison Under Good Lighting

For training the models to predict accurately under good lighting, Mediapipe's hand detection confidence was set to 0.7 – 1.0 and hand tracking confidence was set to 0.5.

```
y_test[0]
```

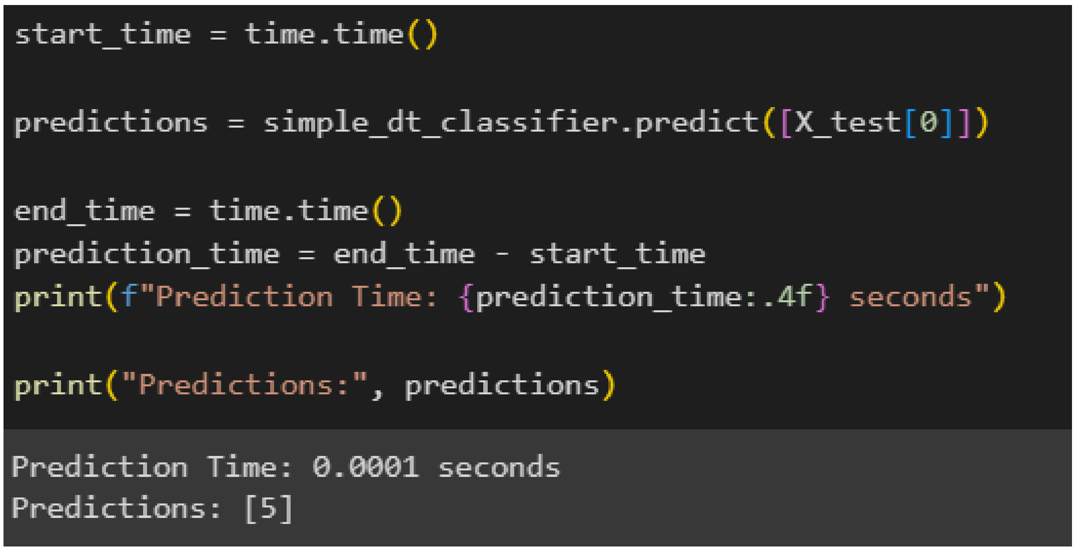
```
5
```



```
kNN_model = joblib.load('knn_model.sav')

start_time = time.time()

predictions = kNN_model.predict(X_test[0])

end_time = time.time()
prediction_time = end_time - start_time

print("Prediction using KNN:", predictions)
print(f"Prediction Time of KNN: {prediction_time:.4f} seconds")
```

```
Prediction using KNN: 5
Prediction Time of KNN: 0.0005 seconds
```

Figure 5.14: Average Prediction Time of KNN



```
start_time = time.time()

predictions = simple_dt_classifier.predict([X_test[0]])

end_time = time.time()
prediction_time = end_time - start_time
print(f"Prediction Time: {prediction_time:.4f} seconds")

print("Predictions:", predictions)
```

```
Prediction Time: 0.0001 seconds
Predictions: [5]
```

Figure 5.15: Average Prediction Time of Decision Tree

Table 5.3: Comparison table under good light condition

Algorithm	Hand Detection Confidence	Hand Tracking Confidence	Training Time (seconds)	Prediction Time (seconds)
K Nearest Neighbor	0.7 - 1.0	0.5	0.0	0.0005
Simple Decision Tree	0.7 – 1.0	0.5	56.4586	0.0001

From the table 5.3, we can conclude that under good lighting, the accuracy of KNN is better than that of Decision Tree. No training time is needed for KNN as KNN stores and compares the unknown dataset with the known so no actual training is done. The prediction time of decision tree is slightly better than KNN but the difference is negligible.

5.2.2.3 Performance Testing Under Mediocre Light Condition:

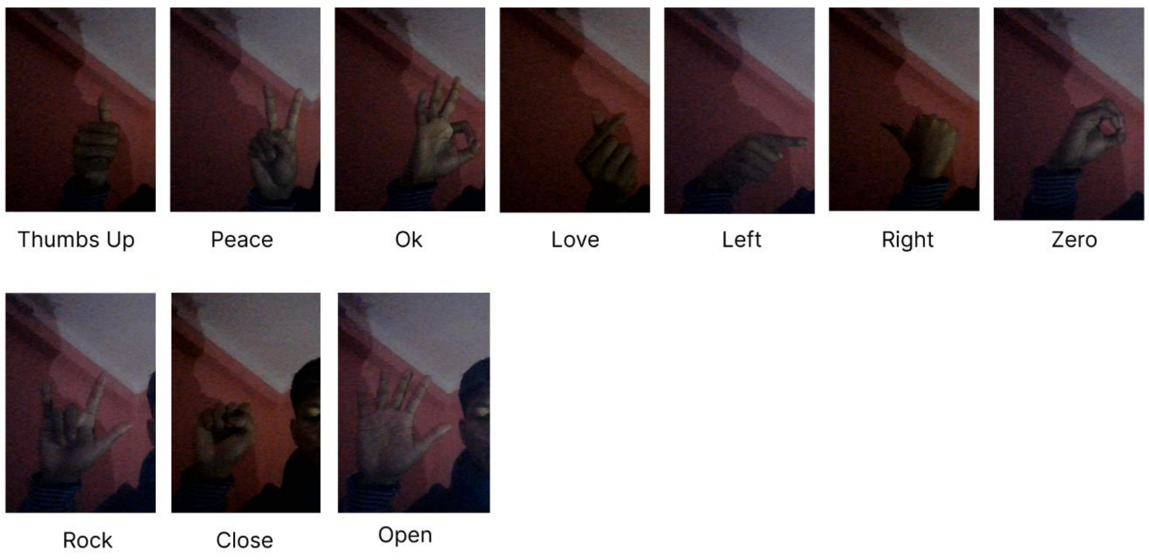


Figure 5.16: Dataset with mediocre light condition

Each class consists of approximately 45 instances where 75% of the dataset is separated for training purpose and 25% of the dataset is used for testing purposes. These images are captured under mediocre lighting conditions and the prediction results are compared between the two algorithms in the next section.

5.2.2.4 Comparison Table Under Mediocre Lighting

For training the models to predict accurately under mediocre lighting, Mediapipe's hand detection confidence was set to 0.1 and hand tracking confidence was set to 0.5.

Table 5.4: Comparison table under mediocre light condition

Algorithm	Hand Detection Confidence	Hand Tracking Confidence	Training Time (seconds)	Prediction Time (seconds)
K Nearest Neighbor	0.1	0.5	0.0	0.0005
Simple Decision Tree	0.1	0.5	49.7030	0.0002

From the table 5.4, we can conclude that under mediocre lighting, the accuracy of KNN decreased as compared to when lighting condition was good. The accuracy of Decision Tree doesn't seem to have a significant change under different light conditions. Training time of KNN is 0 as KNN stores and compares the unknown dataset with the known so no actual training is required. The prediction time of decision tree is slightly better than KNN but the difference is negligible.

5.3 Result Analysis

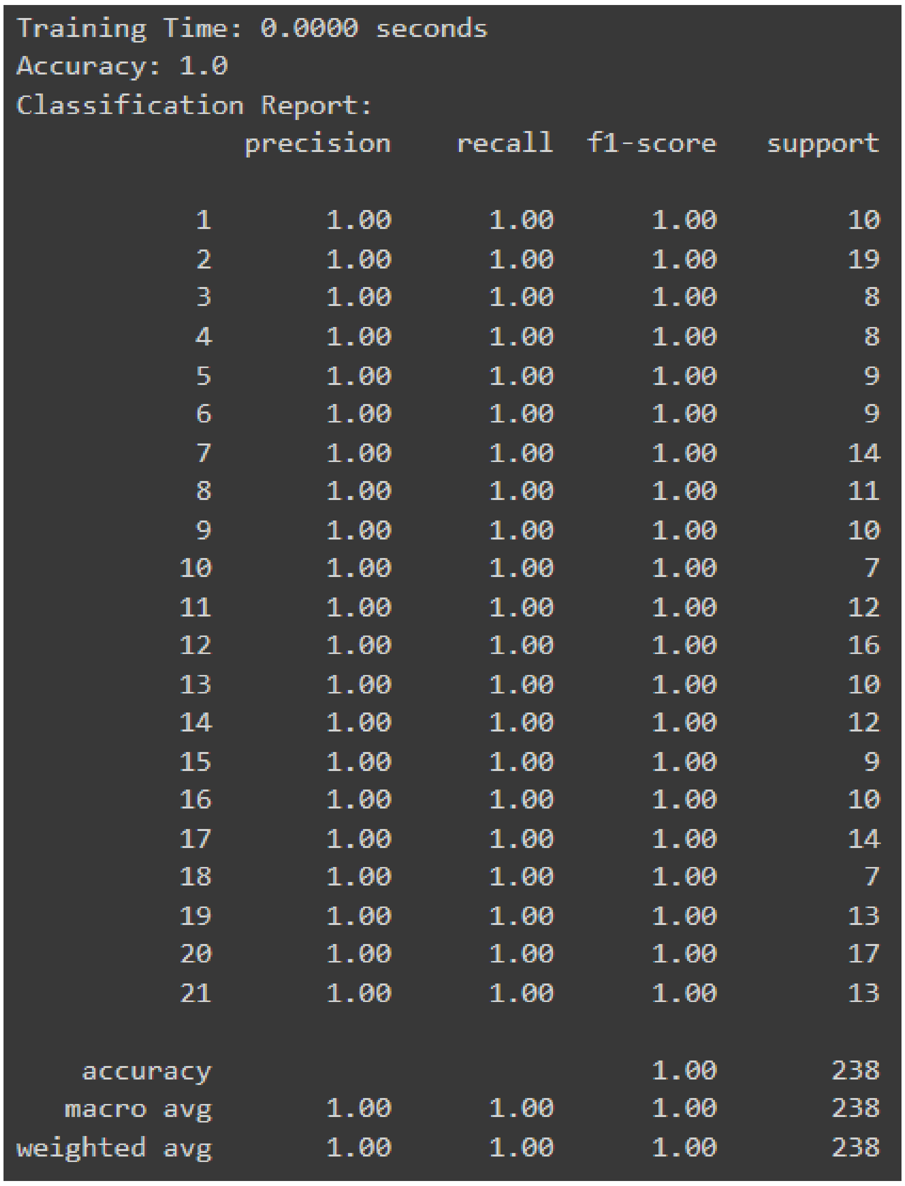
Unit testing and system testing was used to test the system, and the results showed that it could successfully carry out its intended tasks. The project's objectives were met, according to the results, while there is still opportunity for development in terms of raising community involvement and adding to the system's capabilities.

5.3.1 Evaluating Accuracy

One popular indicator in machine learning is accuracy, which is used to assess how well a classifier model is performing. The percentage of correctly categorized occurrences among all instances in the dataset is known as accuracy. The dataset must first be split into a training set and a test set in order to calculate accuracy. The test set is used to assess the model's performance, whereas the training set is used to train the model.

5.3.1.1 Evaluating Accuracy for KNN

The best hyper parameter was found to be n neighbors = 3. The accuracy was 100% for both training and test dataset.



```
Training Time: 0.0000 seconds
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        19
           3       1.00      1.00      1.00         8
           4       1.00      1.00      1.00         8
           5       1.00      1.00      1.00         9
           6       1.00      1.00      1.00         9
           7       1.00      1.00      1.00        14
           8       1.00      1.00      1.00        11
           9       1.00      1.00      1.00        10
          10       1.00      1.00      1.00         7
          11       1.00      1.00      1.00        12
          12       1.00      1.00      1.00        16
          13       1.00      1.00      1.00        10
          14       1.00      1.00      1.00        12
          15       1.00      1.00      1.00         9
          16       1.00      1.00      1.00        10
          17       1.00      1.00      1.00        14
          18       1.00      1.00      1.00         7
          19       1.00      1.00      1.00        13
          20       1.00      1.00      1.00        17
          21       1.00      1.00      1.00        13

    accuracy                           1.00       238
   macro avg       1.00      1.00      1.00       238
weighted avg       1.00      1.00      1.00       238
```

Figure 5.17: Classification Report of KNN

In the figure 5.17, showing the classification report of KNN, the overall accuracy, is 100% which tells us about the model predicting class labels for all the instances.

All the precision, F1-score and recall have values equal to 1, which indicates the perfect performance for each class.

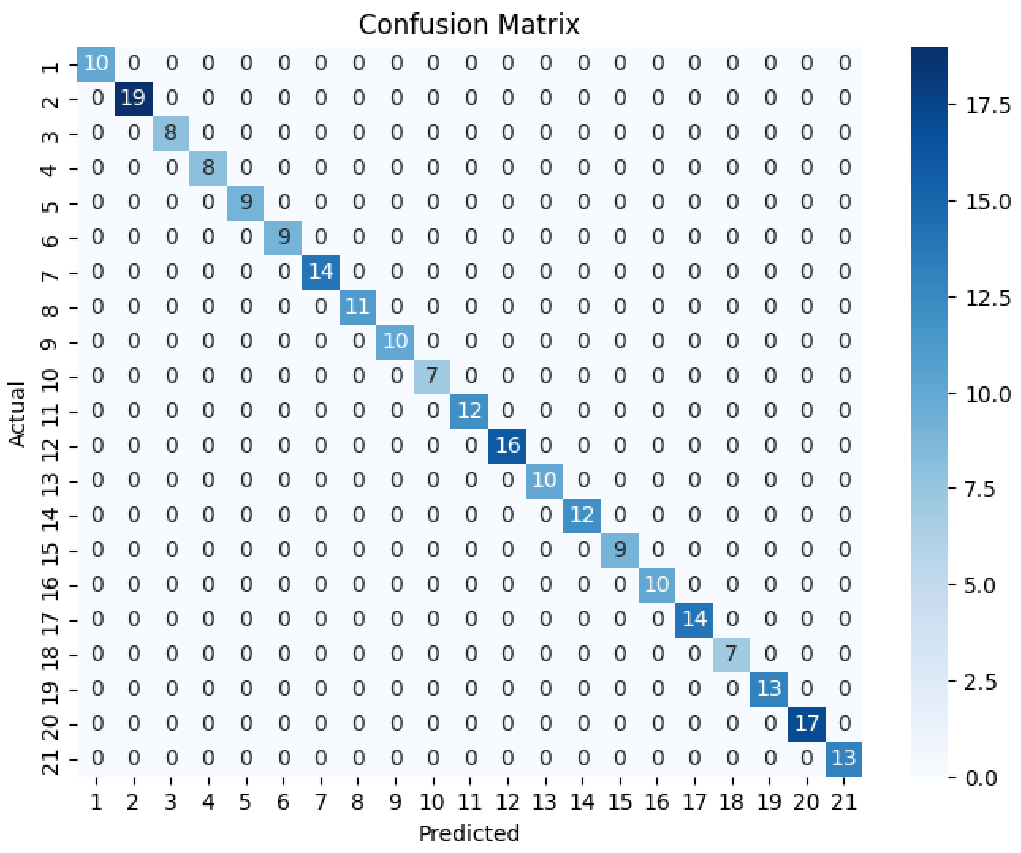


Figure 5.18: Confusion Matrix of KNN

In the figure 5.18, the confusion matrix of KNN, where rows represent the actual classes and the column represents predicted classes.

- **Diagonal Element:** Diagonal elements represents the number of instances where true class is equal to predicted class. Here for example, the top-left corner consists of 10 instances where both actual and predicted classes are 0.

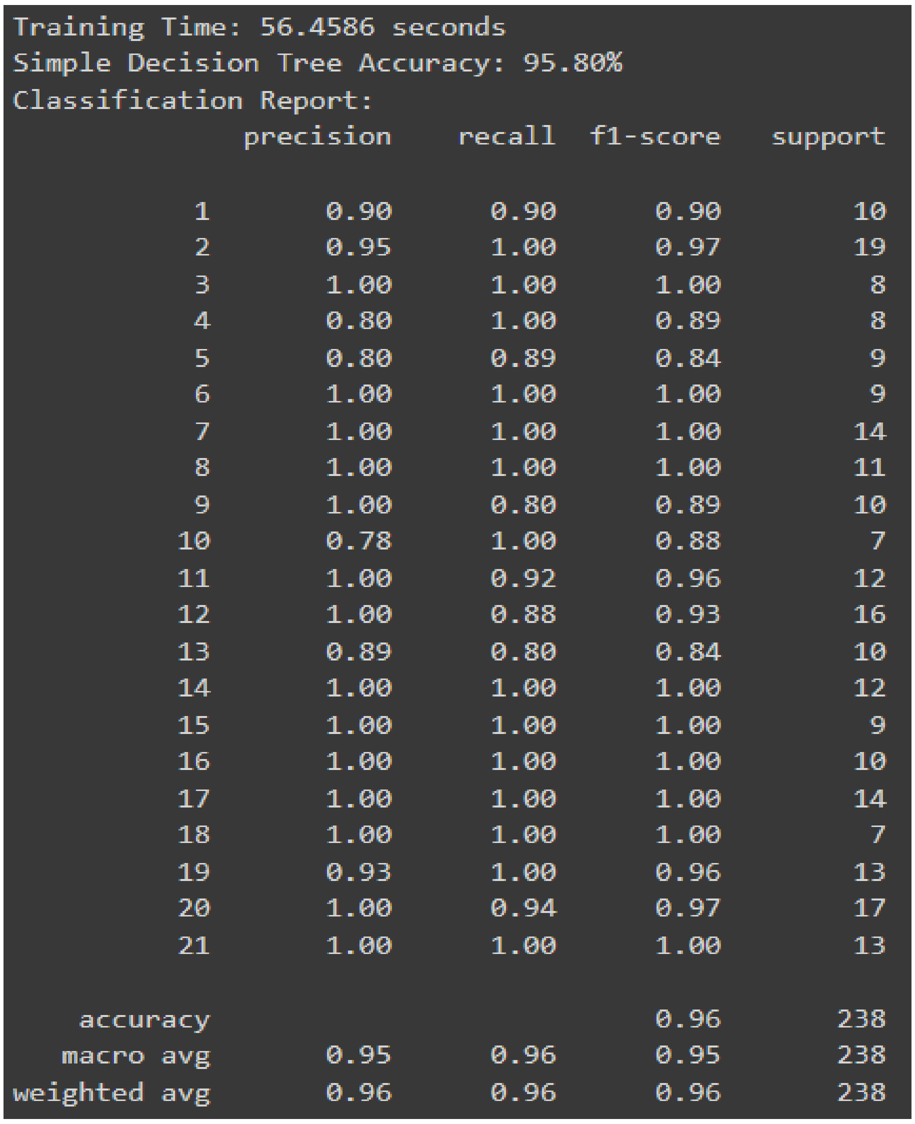
- **Off Diagonal Element:** Off diagonal basically represents the instances that are misclassified. Such elements represent the misclassification of data.

- **Class Imbalances:** Class Imbalances refers to the imbalance in the number of instances in different classes. In the above figure, class 2 has the highest instances i.e. 19 whereas class 18 has the lowest instances i.e. 7 causing class imbalance between the two classes.

- **Misclassification:** Here, misclassifications in the confusion matrix refer to incorrect predictions made by the model. The above model is 100% accurate so there are no misclassifications.

5.3.1.2 Evaluating Accuracy of Decision Tree

The Simple Decision Tree model was trained and assessed on the test dataset. During the hyperparameter tuning process, the maximum depth of the tree was left unrestricted (max_depth = None).



```
Training Time: 56.4586 seconds
Simple Decision Tree Accuracy: 95.80%
Classification Report:
              precision    recall  f1-score   support

           1       0.90      0.90      0.90        10
           2       0.95      1.00      0.97        19
           3       1.00      1.00      1.00         8
           4       0.80      1.00      0.89         8
           5       0.80      0.89      0.84         9
           6       1.00      1.00      1.00         9
           7       1.00      1.00      1.00        14
           8       1.00      1.00      1.00        11
           9       1.00      0.80      0.89        10
          10       0.78      1.00      0.88         7
          11       1.00      0.92      0.96        12
          12       1.00      0.88      0.93        16
          13       0.89      0.80      0.84        10
          14       1.00      1.00      1.00        12
          15       1.00      1.00      1.00         9
          16       1.00      1.00      1.00        10
          17       1.00      1.00      1.00        14
          18       1.00      1.00      1.00         7
          19       0.93      1.00      0.96        13
          20       1.00      0.94      0.97        17
          21       1.00      1.00      1.00        13

    accuracy                           0.96       238
   macro avg       0.95      0.96      0.95       238
weighted avg       0.96      0.96      0.96       238
```

Figure 5.19: Classification Report of Decision Tree

The resulting accuracy on both the training and test datasets was found to be 95.80%. This metric indicates the proportion of correctly classified instances out of the total test dataset. The classes like class 5, class 13 have lesser number of F1-score, which can be because of challenges in distinguishing classes.

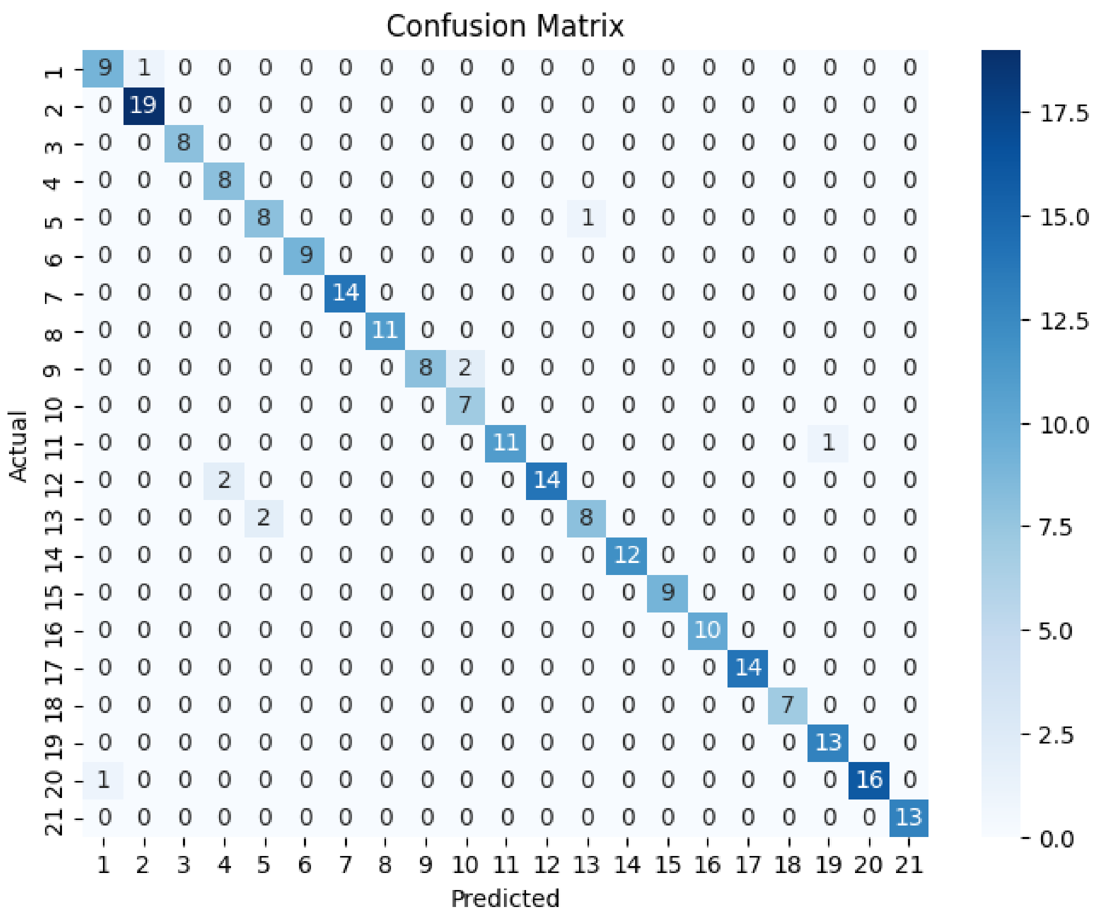


Fig 5.20: Confusion Matrix of Decision Tree

In the figure 5.20, the confusion matrix of KNN, where rows represent the actual classes and the column represents predicted classes.

- **Diagonal Element:** Diagonal elements represents the number of instances where true class is equal to predicted class. Here for example, the bottom-right corner consists of value 13 which represents the number of instances correctly predicted as class 21 is 13 with no misclassifications.

- **Off Diagonal Element:** Off diagonal basically represents the instances that are misclassified.

- **Class Imbalances:** Class Imbalances refers to the imbalance in the number of instances in different classes. In the above figure, class 2 has the highest instances i.e. 19 whereas class 18 has the lowest instances i.e. 7 causing class imbalance between the two classes

- **Misclassification:** Here, misclassifications in the confusion matrix refer to incorrect predictions made by the model. Here, 14 instances of class 12 are correctly predicted whereas the remaining 2 instances are predicted as class 4.

CHAPTER 6
CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

In conclusion, the gesture recognition system uses the KNN algorithm to classify and recognize custom gestures once trained. The model trained is able to identify gestures accurately based on their structure, allowing users to interact with the system more intuitively and efficiently.

Furthermore, the system also incorporates a feature to distinguish between individual hand movements, providing an additional layer of precision and accuracy in recognizing gestures. This enhanced capability allows the system to better adapt to the diverse needs and preferences of users.

Overall, the gesture recognition system offers a robust and effective solution for enhancing human-computer interaction through gesture recognition and analysis.

6.2 Future Recommendations

There are several potential future recommendations that could be implemented in the gesture recognition system project:

• Multi-hand support: Currently, the system only supports gesture recognition from a single hand. Adding support for multi-hand gesture recognition could make the system more accessible and adaptable to users with varying hand mobility.

• Fine-grained gesture recognition: Enhancing the system's capability to recognize subtle differences in hand movements could lead to more accurate and detailed gesture recognition. This improvement would enable users to interact with the system in a more intuitive and efficient manner.

• Integration with natural language processing (NLP) systems: Combining the system's gesture recognition capabilities with advanced NLP systems could facilitate more advanced interactions between users and the system. This integration would allow users to interact with the system using a combination of gestures and spoken language, further enhancing the overall user experience.

• Hand pose tracking: Implementing real-time hand pose tracking algorithms could enhance the system's ability to recognize gestures and adapt to users with varying hand sizes and gestural styles. This improved capability would enable the system to provide a more personalized and adaptive gesture recognition experience for users.

By implementing these future recommendations, the gesture recognition system project could continue to evolve and improve, offering a more versatile, intuitive, and accessible solution for gesture recognition and interaction in various applications and settings.

REFERENCES

Citations:

[1] Mitra, Sushmita, and Tinku Acharya. "Gesture recognition: A survey." *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (2007): 311-324.

[2] Pavlovic, Vladimir I., Rajeev Sharma, and Thomas S. Huang. "Visual interpretation of hand gestures for human-computer interaction: A review." *IEEE Transactions on pattern analysis and machine intelligence* 19.7 (1997): 677-695.

[3] Haria, Aashni, et al. "Hand gesture recognition for human computer interaction." Procedia computer science 115 (2017): 367-374.

[4] Agarap, Abien Fred. "An architecture combining convolutional neural network (CNN) and support vector machine (SVM) for image classification." *arXiv preprint arXiv:1712.03541* (2017).

[5] Sheth, Vraj, Urvashi Tripathi, and Ankit Sharma. "A Comparative Analysis of Machine Learning Algorithms for Classification Purpose." *Procedia Computer Science* 215 (2022): 422-431.

[6] Oudah, Munir, Ali Al-Naji, and Javaan Chahl. "Hand gesture recognition based on computer vision: a review of techniques." *journal of Imaging* 6, no. 8 (2020): 73.

[7] Kumar, B. P., and M. Manjunatha. "Performance analysis of KNN, SVM and ANN techniques for gesture recognition system." *Indian Journal of Science and Technology* 9, no. 1 (2016): 1-8.

BIBLIOGRAPHY

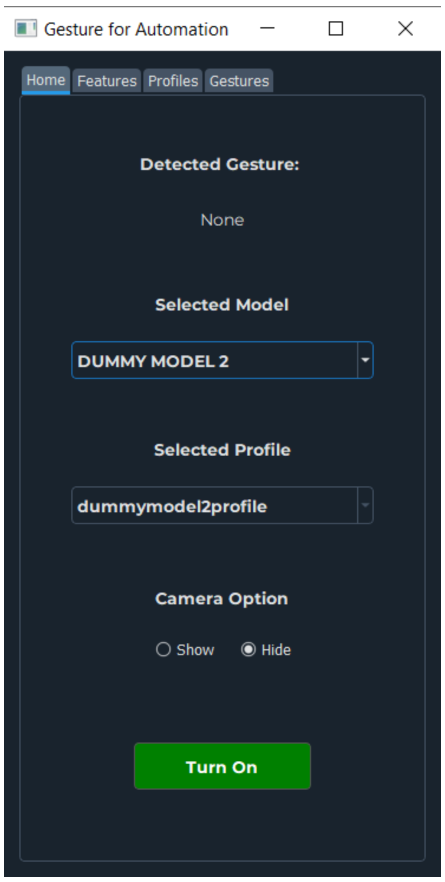
[Online].(2024) Available: https://docs.python.org/3/

[Online]. (2024) Available: OpenCV - Open Computer Vision Library

[Online]. (2024) Available: https://developers.google.com/mediapipe

APPENDICES

Screenshots:


Gesture for Automation

Detected Gesture:

None

Selected Model

DUMMY MODEL 2

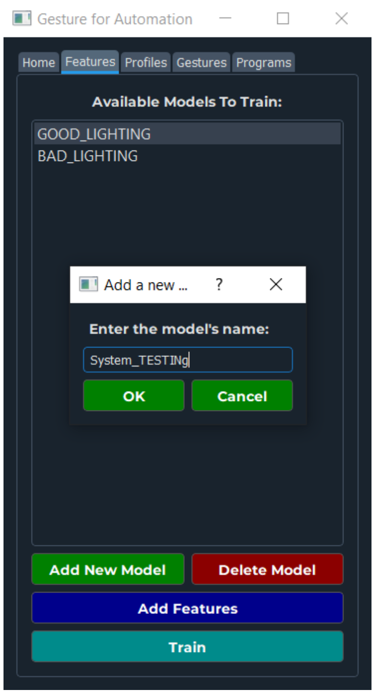
Selected Profile

dummymodel2profile

Camera Option

Show [✓ Hide]

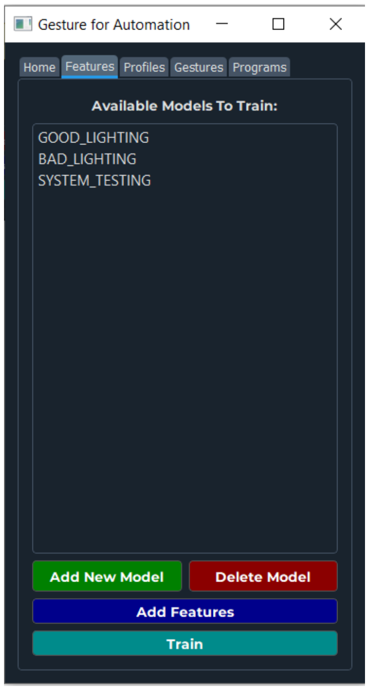
Screenshot1: Home Tab of Application



GOOD_LIGHTING
BAD_LIGHTING

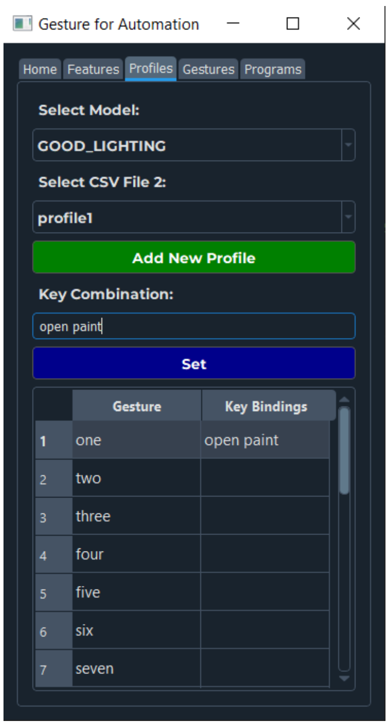
System_TESTINg

Screenshot2: Feature Tab of Application adding new model



GOOD_LIGHTING
BAD_LIGHTING
SYSTEM_TESTING

Screenshot3: Feature Tab of Application model created



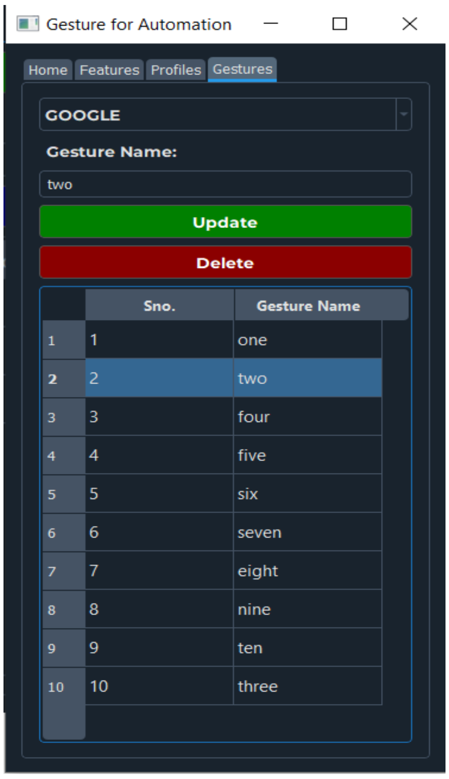
Select Model: GOOGLE_LIGHTING

Select CSV File 2: profile1

Key Combination: open paint

	Gesture	Key Bindings
1	one	open paint
2	two	
3	three	
4	four	
5	five	
6	six	
7	seven	

Screenshot3: Profile Tab of Application adding key combination

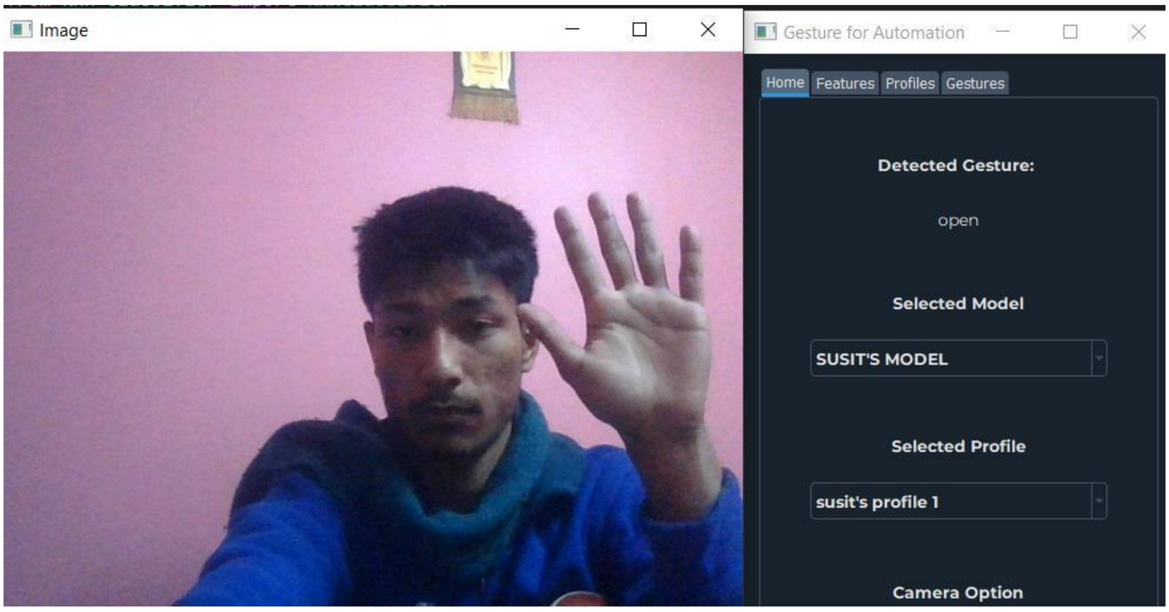


GOOGLE

Gesture Name: two

	Sno.	Gesture Name
1	1	one
2	2	two
3	3	four
4	4	five
5	5	six
6	6	seven
7	7	eight
8	8	nine
9	9	ten
10	10	three

Screenshot4: Gesture Tab of Application



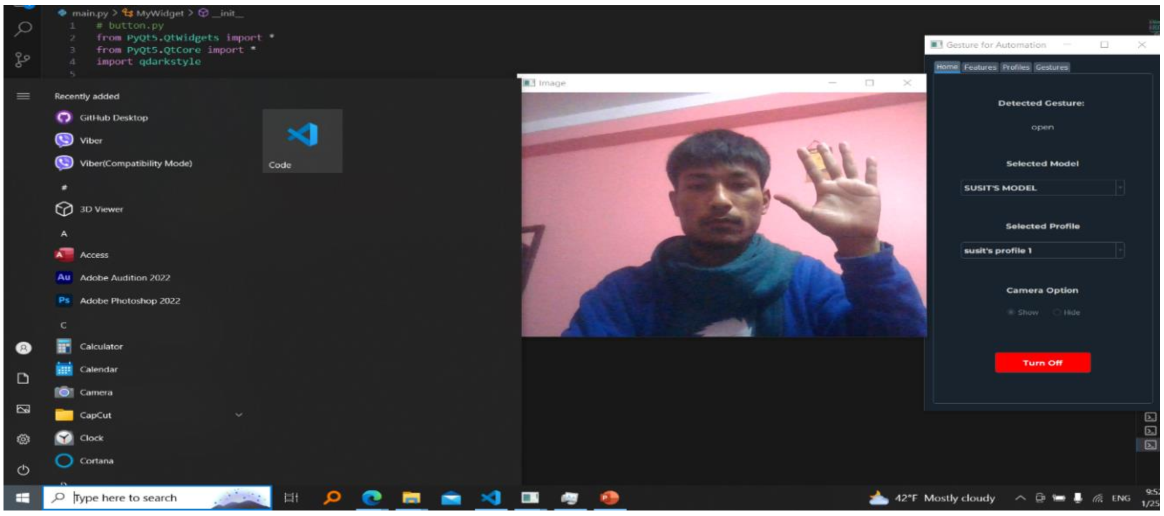
Detected Gesture:
open

Selected Model
SUSIT'S MODEL

Selected Profile
susit's profile 1

Camera Option

Screenshot5: Detection of hand gesture ‘Open’



```
main.py > MyWidget > __init__
# button.py
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
import qdarkstyle
```

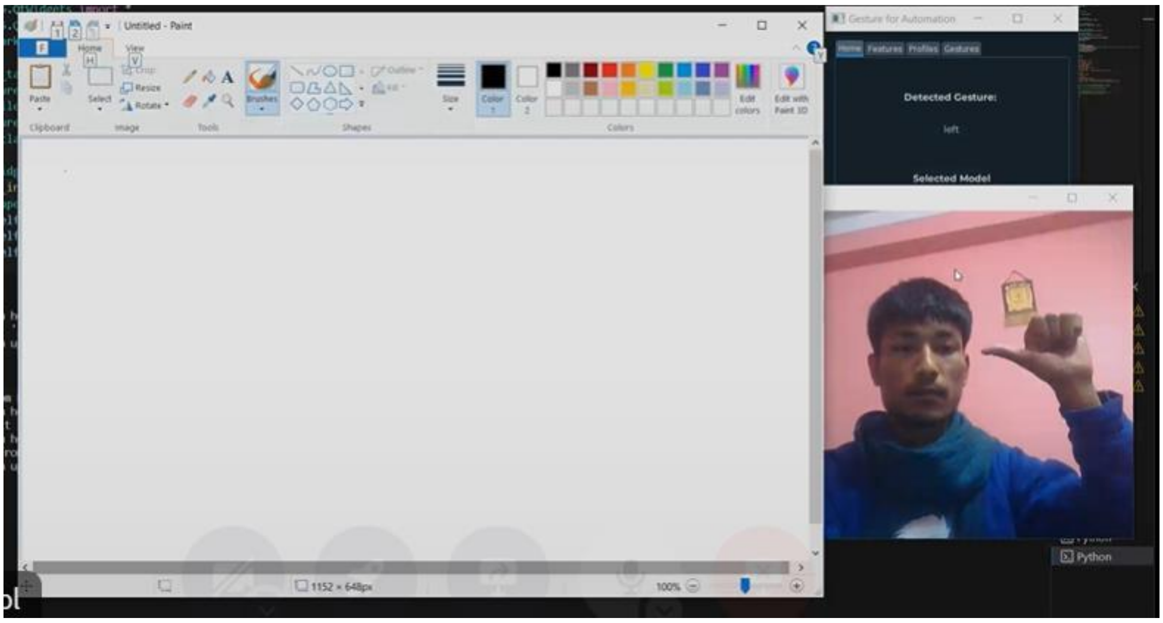
Detected Gesture:
open

Selected Model
SUSIT'S MODEL

Selected Profile
susit's profile 1

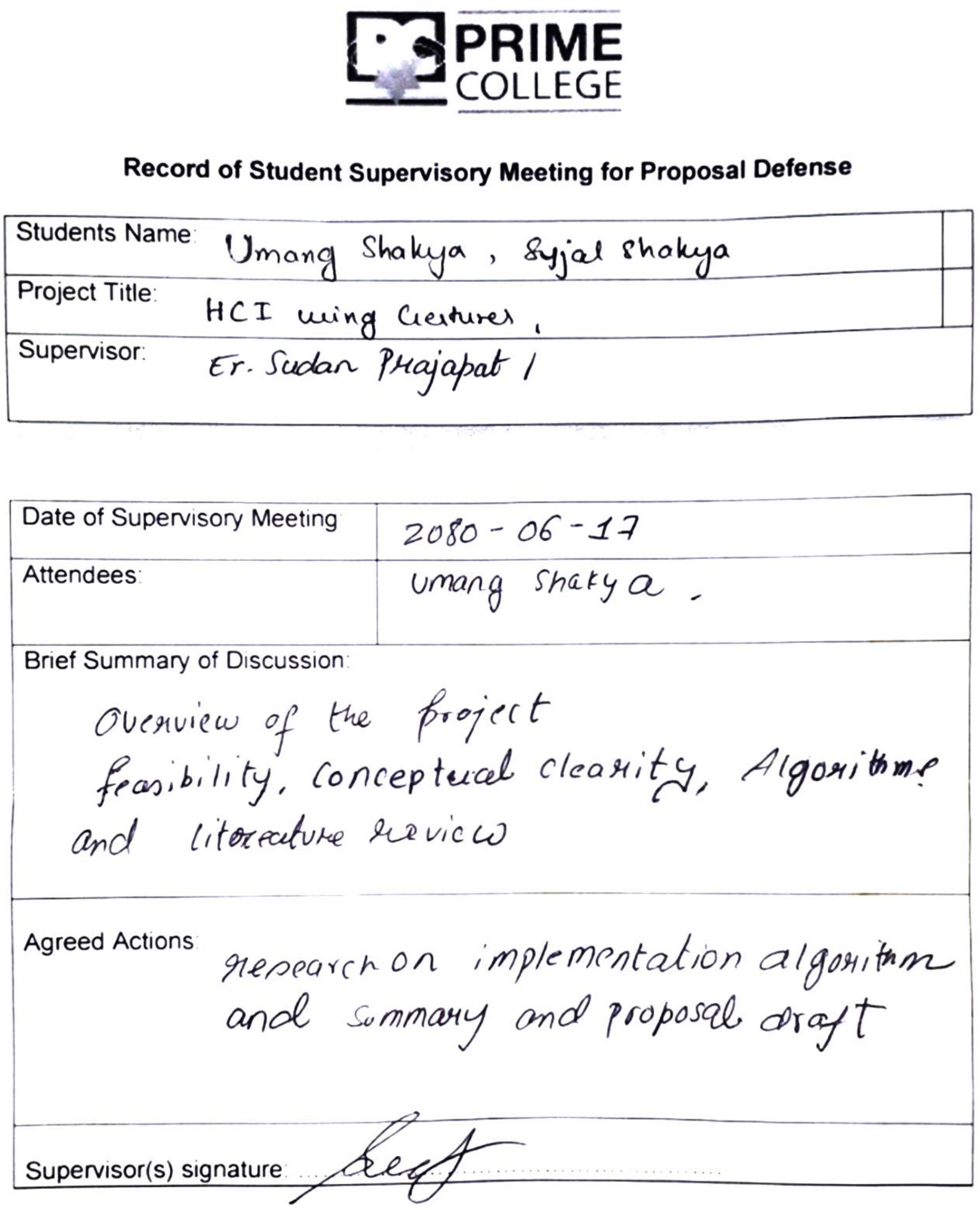
Camera Option
Show Hide

Screenshot6: Opening window tab to open the paint app



Screenshot6: Detection of hand gesture 'left' and opening Paint Application performing some gestures

Log Sheets:


PRIME COLLEGE

Record of Student Supervisory Meeting for Proposal Defense

Students Name:	Umang Shakya , Sujal Shakya
Project Title:	HCI using Gestures.
Supervisor:	Er. Sudan Prajapati

Date of Supervisory Meeting:	2080 - 06 - 17
Attendees:	Umang Shakya .

Brief Summary of Discussion:

Overview of the project
feasibility, conceptual clearity, Algorithms
and literature review

Agreed Actions: research on implementation algorithm and summary and proposal draft

Supervisor(s) signature: [illegible]

PRIME COLLEGE

Record of Student Supervisory Meeting for Proposal Defense

Students Name:	Umang Shakya, Sujal Shakya
Project Title:	HCI using Gestures.
Supervisor:	Er. Sudan Prajapati

Date of Supervisory Meeting:	2080-06-24
Attendees:	Umang Shakya, Sujal Shakya

Brief Summary of Discussion:

proposal Report finalization
Abstract level discussion
Presentation points discussion.
IEEE format for Referencing

Agreed Actions:

Explore of feed forward back propogation Algorithm

Supervisor(s) signature: [illegible]

PRIME COLLEGE
Affiliated to Tribhuvan University

CSIT 7th Semester Project Details

Project Title: Personalized Gesture Recognition

Project Members: Sujal Shakya , Susit Ratna Tuladhar, Umang Shakya

Supervisor Name: Sudan Prajapati

DATE	TASK	REMARKS	SIGNATURE
2080/08/12	Feature Extraction of Gesture Training set	Reseach more on algoritms.	[illegible]
2080/08/17	Research on SVM algorithm & implementation	SVM with 70% accuracy	[illegible]
2080/08/23	Implement Naive Bayes, KNN, Decision Tree	KNN & Decision Tree with more 90% acc	[illegible]
2080/08/26	Included the training module on program	Training Module included with CRUD operation	[illegible]
2080/09/11	Added more features to desktop app	Profile, Gesture Management added	[illegible]
2080/09/16	Switch to SQLite DB	Migrated data from CSV to Database	[illegible]
2080/09/21	Implementation of User Interface	PyQt5 for UI implemented	[illegible]
2080/09/25	Add figures and tables to document	improve doc follow IEEE Reference	[illegible]
2080/09/29	Refinement Diagrams	Use UML diagrams	[illegible]
2080/10/04	Testing & Evaluation of KNN, Decision Tree	Use More Test Sets and improve accuracy	[illegible]

PRIME COLLEGE

Record of Student Supervisory Meeting for Final Acceptance

Students Name: Sujal Shakya, Susit Ratna Tuladhar, Umang Shakya	
Project Title: Personalized Gesture Recognition	
Supervisor: Sudan Prajapati	

Date of Final Supervisory Meeting:	16th Falgun
Attendees:	Sujal Shakya, Susit Ratna Tuladhar, Umang Shakya

Brief Summary of Discussion: Accepted for final presentation. Signature:....[illegible]..........