TRIBHUVAN UNIVERSITY
FACULTY OF SCIENCE AND TECHNOLOGY

A Project Report on

"SignScript: Real-time Sign Language Recognition System"

Submitted to:
Department of Statistics and Computer Science
Patan Multiple Campus

In partial fulfilment of the requirements for bachelor's degree in computer science and information technology

Submitted By:
Kaushal Lamichhane (25539/077)
Nishesh Pokharel (25549/077)
Pratik Rai (25555/077)

Under The Supervision Of
Er. Roshan Kumar Nandan

Certificate of Declaration

Award Title: BSc. Computer Science and Information Technology

Declaration Sheet

(Presented in partial fulfilment of the assessment requirements for the above award)

This work, in whole or in part, has no way been submitted to the university or any other institutional body in any manner for assessment or other purposes. We attest that the intellectual content of the work was entirely the product of our own sweats.

We affirm that the intellectual content of the work is entirely the product of our own hard work. It’s conceded that the innovator always owns the brand in all design work. It's conceded that the brand in all design works is possessed by the author. still, by submitting similar copyrighted work for evaluation, the author grants the university a perpetual kingliness-free license to do all acts appertained to in section 16(I) of the Brand, Designs and Patents Act 1988(i.e. Copy the work; make a dupe available for public information; intimately perform, parade, or reproduce the work; broadcast or acclimatize the work.

Declared By:

Kaushal Lamichhane Symbol No: 25539/077	Nishesh Pokharel Symbol No: 25549/077	Pratik Rai Symbol No: 25555/077

Date: 03/03/ 2025

Patan Multiple Campus

Tribhuvan University

Supervisor's Recommendation

I hereby recommend that this project, prepared under my supervision entitled **"SignScript: Real-time Sign Language Recognition System"**, a real time sign language recognition system using machine learning, in partial fulfillment of the requirements for the degree of B.Sc. in Computer Science and Information Technology, is processed for evaluation.

…….

Er. Roshan Kumar Nandan

Project Supervisor

Patan Multiple Campus

Patandhoka, Lalitpur

Patan Multiple Campus

Tribhuvan University

Approval Page

The project "**SignScript: Real-time Sign Language Recognition System**" has been submitted to partial fulfillment of the requirements for the B.Sc. in Computer Science and Information Technology degree. It is a real time sign language recognition system which uses machine learning to predict the correct labels for a given hand gesture.

The following people have assessed and approved this project:

…….	……..
Coordinator	Internal Examiner
Mr. Dadhi Ram Ghimire	
…….	……..
Supervisor	External Examiner
Er. Roshan Kumar Nandan	

Acknowledgement

We would like to expand our ardent appreciation to everybody who played a part in bringing this venture to realization. Their unflinching back, direction, and support have been important all through this travel. An extraordinary thank you goes to Er. Roshan Kumar Nandan, our supervisor, whose ability and shrewd criticism significantly shaped the course of this venture. His mentorship given a solid establishment and essentially upgraded the quality of our work.

We are also deeply thankful to our companions and family for their consistent back, understanding, and support amid the challenging stages of this extend. Their conviction in our capacities has been a colossal source of inspiration.

Our earnest appreciation goes to Patan Different Campus for advertising the essential assets and a strong environment that empowered the improvement and effective completion of this extend.

In conclusion, we are thankful to all people who contributed, straightforwardly or in a roundabout way, to this venture. Your experiences and endeavours have been instrumental in its victory. Thank you to everybody for being portion of this travel and for making a difference turn this venture into a reality.

We are honoured.

Kaushal Lamichhane
Nishesh Pokharel
Pratik Rai

Date: 03/03/2025

ABSTRACT

The identification of hand gestures and sign language for human-computer interaction is a current area of research in computer vision and machine learning. One of its primary goals is to create systems that can identify certain gestures and use them to send data or operate things. Hand postures, on the other hand, are the hand's static structure; gestures, on the other hand, are its dynamic mobility, and they must be explained in terms of both space and time. Data glove techniques and vision-based techniques are the two primary approaches for hand gesture recognition.

The main objective of this endeavor is to develop a vision-based system that can recognize sign language in real time. simple and organic ways for a human and a machine to communicate.

Research on human-machine interaction through gesture recognition has led to the use of such technology in a wide range of applications, including touch screens, video game consoles, virtual reality, medical applications, and sign language recognition. This is because the human hand is one of the most important communication tools in daily life and because image and video processing techniques are constantly improving.

Sign language is the most natural way for deaf people to communicate with one another, yet it has been observed that they struggle to engage with hearing people in everyday situations. Sign language has a vocabulary of signs, just as spoken language has a vocabulary of words.

Keywords: *Computer Vision, Machine Learning, Sign Language, Hand Gesture Recognition*

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

HCI	Human Computer Interface
ML	Machine Learning
AI	Artificial Intelligence
SDK	Software Development Kit
CV	Computer Vision
NSL	Nepali Sign Language
ISL	Indian Sign Language
NLP	Natural Language Processing
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network

CHAPTER 1

INTRODUCTION

1.1. Introduction

Sign dialects were made to a great extent to help the hard of hearing and dumb. In arrange to specific exact data, they utilize a concurrent and particular blend of hand movements, hand shapes, and introduction. This extends falls inside the HCI (Human Computer Interface) segment and looks for to recognize different letter sets (a-z), digits (0-9) and a few commonplace ISL family hand movements such as Thank you, Hi, and so on. Hand-gesture acknowledgment could be a troublesome issue, and ISL recognition is especially troublesome owing to the utilize of both hands. Numerous ponders have been exhausted the past utilizing sensors (such as glove sensors) and different picture preparing strategies (such as edge discovery, Hough Change, and so on), but they are very expensive, and numerous individuals cannot manage them. [1]

1.2. Problem Statement

Normal people find it difficult to understand the verbally impaired since they communicate through hand signs. Therefore, it is necessary to have technologies that can differentiate between different signals and notify the public. As of present, there is no appropriate framework in place to deal with the problem of people who suffer from deaf and speaking disorders. Furthermore, there aren't many dual systems (sign to speech and vice versa) implemented, and there are a lot of useless videos on websites.

1.3. Objectives

The main objective of an sign language detection system are:

- To create a reliable system for detecting sign language in real time.
- To incorporate a mechanism for translating recorded sign gestures into text so that people can comprehend sign language.
- To guarantee optimal system performance over a range of sign motions.

1.4. Scope and Limitation

By offering a user-friendly and effective platform for translating Nepali sign language into text in real-time, SignScript seeks to eliminate the communication gap between the hearing and non-hearing communities.

Scopes:

- The system makes it possible to translate Nepali sign language into text output with ease.
- It makes it easier for people to connect in real time in dynamic settings like meetings, social gatherings, and classrooms.
- The system is made to perform on commonly available gadgets, such as desktops with simple cameras and cellphones and tablets.
- It has the potential to be integrated into assistive technologies, improving accessibility for people with disabilities.

Limitations:

- Individual hand movements, pace, and variances in signing styles can all impact identification accuracy.
- The Nepali sign language is the only one where the system works well.
- Its dependability may be lowered by external elements like dim illumination, background noise, or obscured hand movements.
- Because real-time recognition necessitates consistent hardware and software performance, it is less efficient on devices with lower specifications.

1.5. Development Methodology

In organize to develop our amplify, we have utilized the waterfall illustrate. The waterfall appear can be a successive program advancement handle in which stages like prerequisites gathering, arrange, utilization, testing, sending, and upkeep are all taken after in a tireless descending stream.

Each course of action ought to be wrapped up a number of times some time recently moving on to another step, much like a waterfall that keeps slipping. This strategy gives an

organized system for meander organization, guaranteeing that requests are completely caught on some time recently usage and including clear breakthroughs to screen advance.

Without a doubt, the waterfall approach can be especially successful for ventures with well-defined needs, in spite of the reality that it may not be as adaptable as iterative approaches.

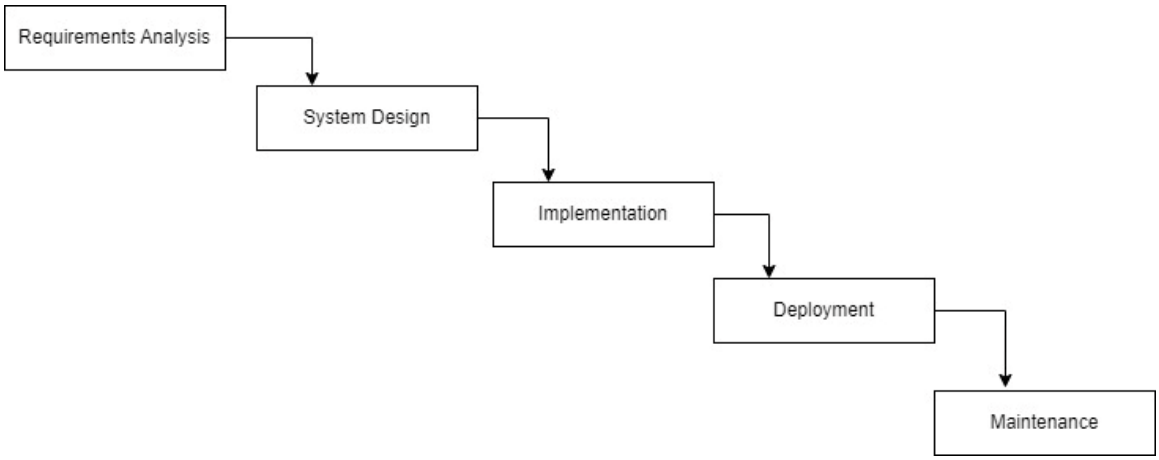


Figure 1.1: Waterfall Model for SignScript

1.6. Report Organization

Chapter 1 gives the common presentation almost the framework and the targets, scope, and confinements.

Chapter 2 investigates the writing audit, crucial hypothesis, phrasings, and existing frameworks.

Chapter 3 audits related works, prerequisite investigation, and achievability examination of our venture.

Chapter 4 indicates the framework plan of the extend and depicts the fundamental handle show.

Chapter 5 bargains with the usage and testing of the framework.

Chapter 6 draws conclusions examined in.

CHAPTER 2:

BACKGROUND STUDY AND LITERATURE REVIEW

2.1. Literature Review

Real-time sign language recognition advances have altogether upgraded communication between the hearing and non-hearing communities by leveraging progresses in computer vision and profound learning. Different investigate endeavors have investigated distinctive procedures for motion acknowledgment, counting machine learning, profound learning, and multimodal approaches.

Automatic sign language recognition framework is based on deep learning and repetitive neural systems (RNNs). Several research have been conducted into the field of sign language recognition highlighting the superiority of CNN-RNN architectures. Koller et al. [2], proposed a programmed sign dialect acknowledgment framework based on profound learning and repetitive neural systems (RNNs). The investigate illustrated that consolidating worldly conditions in sign signals made strides acknowledgment exactness, especially in complex sentence structures. Zhang et al. [3], utilized a combination of Convolutional Neural Systems (CNNs) and Long Short-Term Memory (LSTM) systems for dynamic gesture recognition. Their approach accomplished high precision by integrating spatiotemporal highlights, enabling the system to recognize full sentences in sign language instead of isolated gestures. In addition, research conducted by Joze and Koller presented a dataset-centric approach by utilizing large-scale sign dialect video datasets. Their framework utilized self-supervised learning methods to enhance sign dialect recognition without requiring broad labeled data [4].

Sign language recognition has too profited from multimodal combination methods, where analysts combine hand signals, facial expressions, and body movements to achieve higher recognition exactness. A study by Pigou et al. [5], illustrated those consolidating numerous modalities essentially progressed execution in real-time sign dialect acknowledgment applications.

Cooper et al. [6], investigated different methods for sign dialect acknowledgment, centering on the application of machine learning models for hand signal classification. Their inquire

about highlighted the challenges of energetic motion acknowledgment and the significance of strong highlight extraction strategies. Additionally, Prasanna et al. [7], given a comprehensive study on hand motion acknowledgment, examining progressions in profound learning and their effect on sign dialect interpretation systems. Their ponder emphasized the require for real-time, proficient models that can generalize well over diverse endorsers and situations. These considers fortify the importance of utilizing CNN-based designs for precise and versatile sign dialect acknowledgment.

One approach for recognizing signs in Nepali sign language (NSL) was developed [8]. These systems convert visual sign language motions into text or voice by utilizing cutting-edge methods in computer vision, machine learning, and gesture recognition. These developments could improve accessibility in a number of areas, such as customer service, healthcare, and education. [9]

Sign Language Recognition Categories

- **Static Gesture Recognition:** usually used to recognize alphabets or particular words, this technique focuses on reading individual signals or hand shapes separately.
- **Dynamic Gesture Recognition:** focuses on hand movement patterns, making it possible to recognize words and sentences. For this method to effectively comprehend gesture transitions, strong temporal modeling is necessary.
- **Multimodal Recognition:** incorporates more clues for a thorough interpretation, including hand gestures, body posture, and facial emotions. Although encouraging, it makes coordinating various input modalities more difficult.

2.2. Terminologies

- **Sign Language Recognition:** SLR, or sign language recognition is communication between signers and non-signers through the interpretation of body language, facial expressions, and hand gestures to convert sign language into spoken words or text.
- **Gesture Frame:** One camera-captured image that shows a particular hand position, gesture, or emotion that makes up a sign.

- **Feature Extraction:** The process of separating important attributes, including hand shape, orientation, and motion, from input data in order to facilitate precise sign identification is known as "feature extraction."
- **Real-time Translation:** The system's capacity to process and decipher signs instantly, producing text or spoken output almost instantly for efficient communication.

2.3. Existing Systems

2.3.1. Manual Interpretation

In the past, sign language interpretation relied on human interpreters who acted as a bridge between signers and non-signers. Although this method works well in one-on-one or small-group situations, it has some drawbacks:

- **Scalability Problems**: Human interpreters are unable to handle numerous simultaneous or large-scale interactions in real time across several sites.
- **Accessibility Restrictions**: In distant or underdeveloped locations, professional sign language interpreters are not always available.
- **Subjectivity**: Depending on the interpreter's experience and ability level, interpretations may differ, which could result in inconsistent results.

2.3.2. Machine Learning-Based Systems

Sign language recognition advanced significantly with the advent of machine learning (ML) techniques. These systems used machine learning algorithms to extract and classify features.

Some of its key features are as follows:

- Motion and Hand Tracking: tracks the movements, locations, and orientations of hands using camera-based systems or sensor technologies (such as gloves or depth sensors).
- Feature Extraction: uses algorithms to identify important characteristics in input data, like hand shape, motion, details of color, and depth.

2.3.3. Deep Learning and Neural Networks

By mechanizing include extraction and expanding precision for complex movements, profound learning has made major progressions in sign dialect acknowledgment frameworks conceivable.

Key highlights of these frameworks incorporate:

- **Convolutional Neural Systems (CNNs):** Convolutional neural systems, or CNNs, are utilized to distinguish inactive signals by extricating spatial highlights from video outlines.
- **Repetitive Neural Systems (RNNs) and LSMTs:** Energetic sign acknowledgment is made conceivable by the utilize of LSTMs and Repetitive Neural Systems (RNNs), which are utilized to comprehend transient associations in ceaseless movements.
- **Real-Time Frameworks:** Improved preparing control to supply real-time distinguishing proof.

CHAPTER 3:

SYSTEM ANALYSIS

3.1. System Analysis

3.1.1. Requirement Analysis

The outline for making program is called a Software Requirement Specification, or SRS archive. Like a outline, it portrays what the program ought to do, how it ought to work, and for whom. Highlights, client desires, specialized determinations, and non-functional necessities like security and execution are all secured.

There are two essential categories of necessity examination:

3.1.1.1. Functional Requirements

Functional requirements contain a description of the work flows that the system follows, the functions that particular screens carry out, and other business compliance standards that the system needs to fulfill.

- **Image Interpretation and Identification:** The system must use computer vision techniques to reliably evaluate real-time photos and recognize hand gestures.
- **Caption Labeling:** For each recognized hand gesture, the system must produce text captions in Nepali or English that are precise and appropriate given the situation.
- **User Interface:** The system needs to have an easy-to-use interface that can recognize hand gestures in real time and produce relevant phrases based on the motion.
- **Multilingual Support:** Users should be able to choose between Nepali and English as their preferred language on the system.
- **Real-Time Processing:** To give users feedback right away, the system should analyze photos and produce relevant phrases in real time.

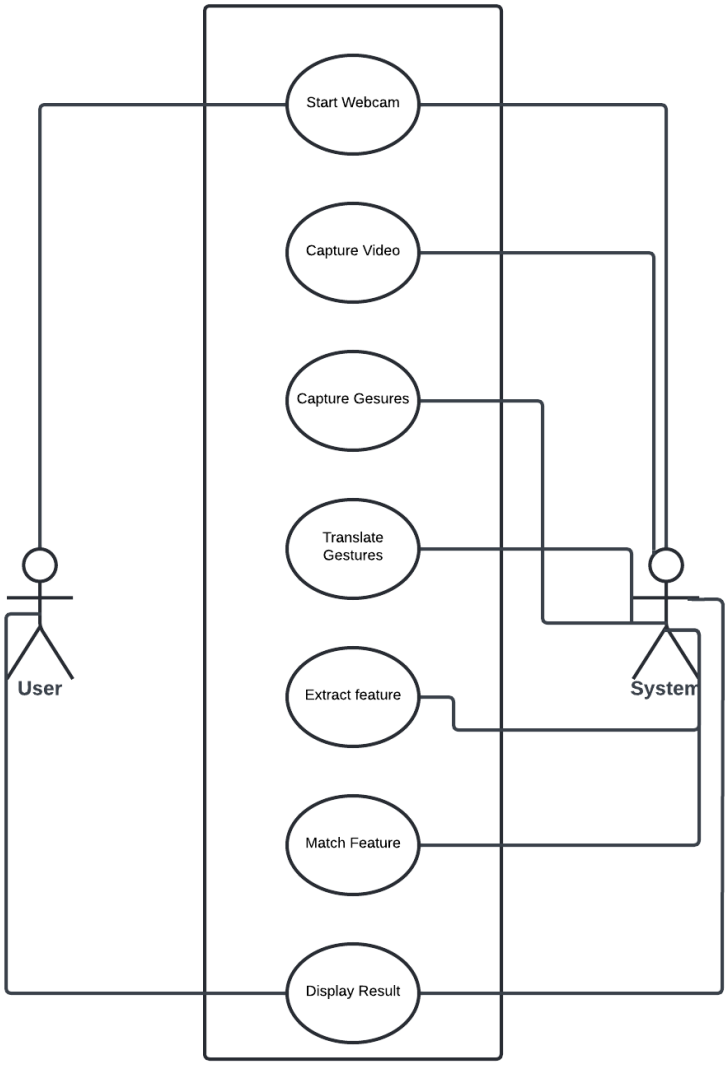


Figure 3.1: Use Case Diagram for SignScript

Actors:

- **User:** Can show hand gestures that need to be translated
- **System:** Captures images, analyses, and recognizes the hand gestures.

Use Cases:

- **Real-time video input:** used by users.
- **Captures gestures:** used by system.
- **Translates gestures:** Exclusive to system, triggered by system after gestures are processed.
- **Display result:** The system displays appropriate labels for processed gestures.

3.1.1.2. Non-Functional Requirements

The non-functional requirements in content to the project are as follows:

- **Performance:** To guarantee seamless and effective operation, the system should have minimal latency and great responsiveness, particularly when identifying hand movements in real-time.
- **Scalability:** It should be possible for the system to increase with the number of users and load variations without seeing a decrease in performance.
- **Accuracy:** The system should maintain a high level of accuracy in gesture recognition and label generation.
- **Security:** The system needs to protect user privacy and prevent unwanted access to data.

3.1.2. Feasibility Analysis

The goal of feasibility studies is to logically and objectively identify the advantages and disadvantages of an existing company or planned endeavor, as well as environmental dangers and possibilities, the resources needed to succeed, and, finally, the likelihood of success. To ascertain whether developing a new or enhanced system is compatible with cost, benefits, operation, technology, and time, a feasibility study is required. The feasibility studies that follow are:

3.1.2.1. Technical Feasibility

The concept is technically feasible given the current state of web technologies. It is possible to build the application efficiently using IDEs like Jupiter Notebook and VSCode. Machine learning libraries like Scikit-learn, CV2, Cvzone and Numpy are easy to integrate and widely available.

3.1.2.2. Operational Feasibility

The platform will offer a smooth user experience with an intuitive graphical user interface and real-time object detection for sign recognition. The platform is compatible with all current operating systems. Hence, the project is operationally feasible.

3.1.2.3. Economic Feasibility

Because open-source tools and libraries like TensorFlow will be used, the project is financially sustainable. Using pre-trained models and free-tier IDEs like Jupiter Notebook for testing and development can lower development costs. .

3.1.2.4. Schedule

The following Gantt chart shows a visual representation of our project tasks displayed against time. It allows us to see the start and finish dates of individual tasks within the project, as well as any dependencies between tasks.

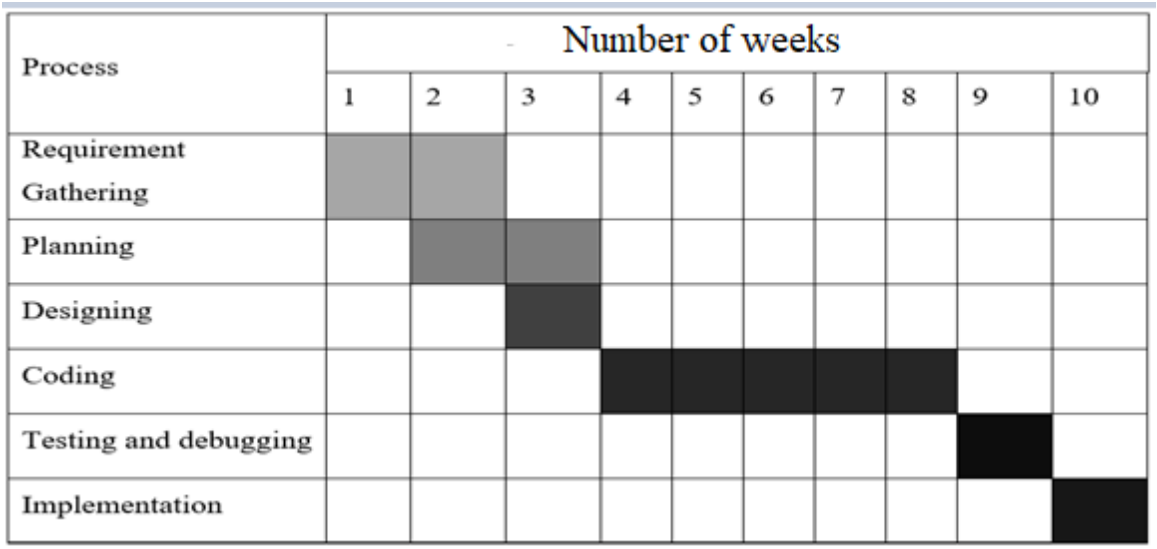


Figure 3.2: Gantt Chart for Scheduling of Project

3.1.3. Analysis

3. 1.3.1. Object Modelling Using Class and Object Diagram

A class diagram system consists of three main classes: Hand, Features, and Sign. The Hand class represents the detected hand and includes attributes such as hand_id, image_path, and timestamp, along with a method detect_hand() to perform hand detection using OpenCV. The Features class stores the extracted hand landmarks, with attributes like feature_id, hand_id, coordinate_x, and coordinate_y, and a method extract_features() to extract these landmarks. The Sign class represents the classified hand sign, containing attributes such as sign_id, feature_id, and label, along with a method classify_sign() to classify the sign using

a Random Forest model. The relationships between the classes are clearly defined: a Hand can contain multiple Features , and each set of Features is classified into one Sign.

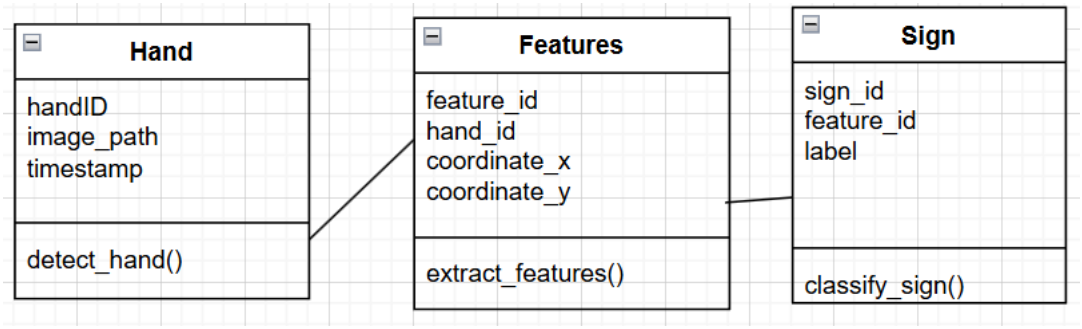


Figure 3.3: Class Diagram for SignScript

The object diagram of the signscript system has the Hand1 object represents a detected hand with attributes such as hand_id = 101, image_path = "/images/ok.png", and timestamp = "2024-08-12 12:23:56". This hand object is associated with two sets of features: Features1 and Features2, which store the extracted hand landmarks with coordinates like coordinate_x = 0.45, coordinate_y = 0.563 for Features1 and coordinate_x = 0.50, coordinate_y = 0.893 for Features2. Each set of features is classified into a corresponding sign: Features1 is classified as Sign1 with the label "Hudaina".

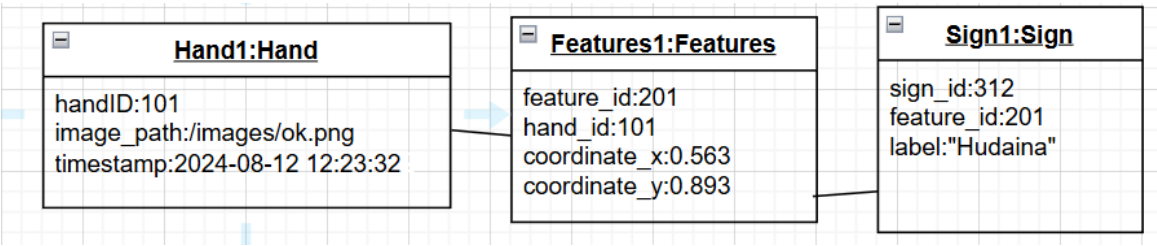


Figure 3.4: Object Diagram for SignScript

3.1.3.2. Dynamic Modelling Using State and Sequence Diagram

The sequence diagram for SignScript represents a basic flow of interactions between the user and the system. The user initiates the process by capturing a hand image, which is then passed to the Hand Detection module. The Hand Detection module uses OpenCV to detect the hand and forwards the detected hand to the Feature Extraction module. The Feature Extraction module extracts the hand landmarks (features) and sends them to the Sign

Classification module. Finally, the Sign Classification module uses a Random Forest model to classify the hand sign and displays the result to the user.

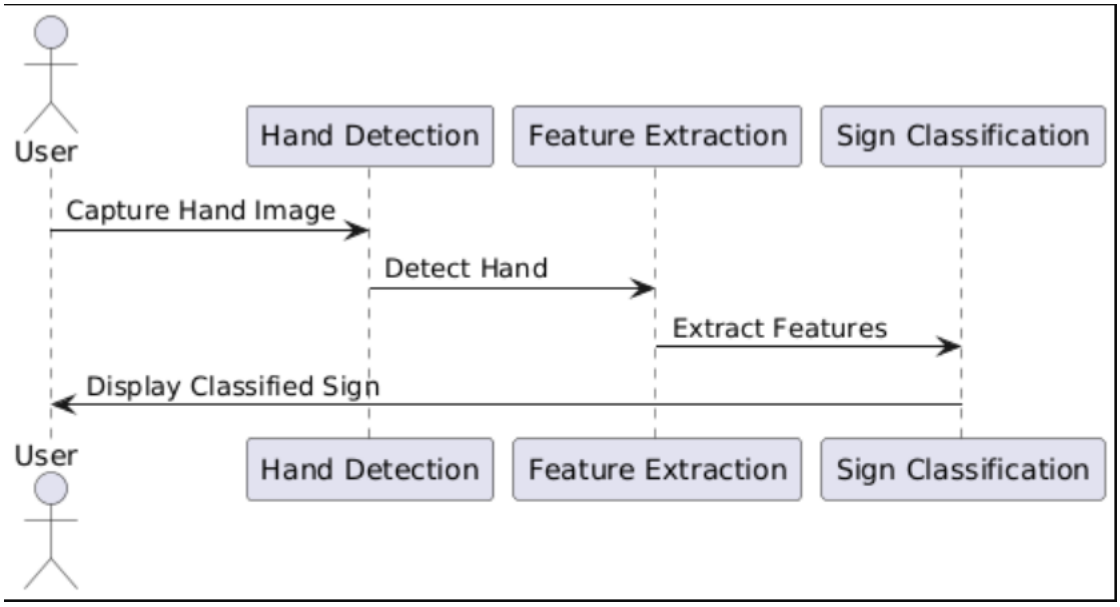


Figure 3.5: Sequence Diagram for SignScript

The state diagram for SignScript represents a high-level overview of the system's states and transitions. The system starts in the Idle state, where it waits for user input. When a hand is detected, the system transitions to the Hand Detected state. From there, it moves to the Features Extracted state once the hand landmarks are successfully extracted. Finally, the system transitions to the Sign Classified state, where the hand sign is classified and displayed to the user. After classification, the system returns to the Idle state, ready to process the next input. This state diagram is simple and does not account for error states or intermediate steps like preprocessing or validation.

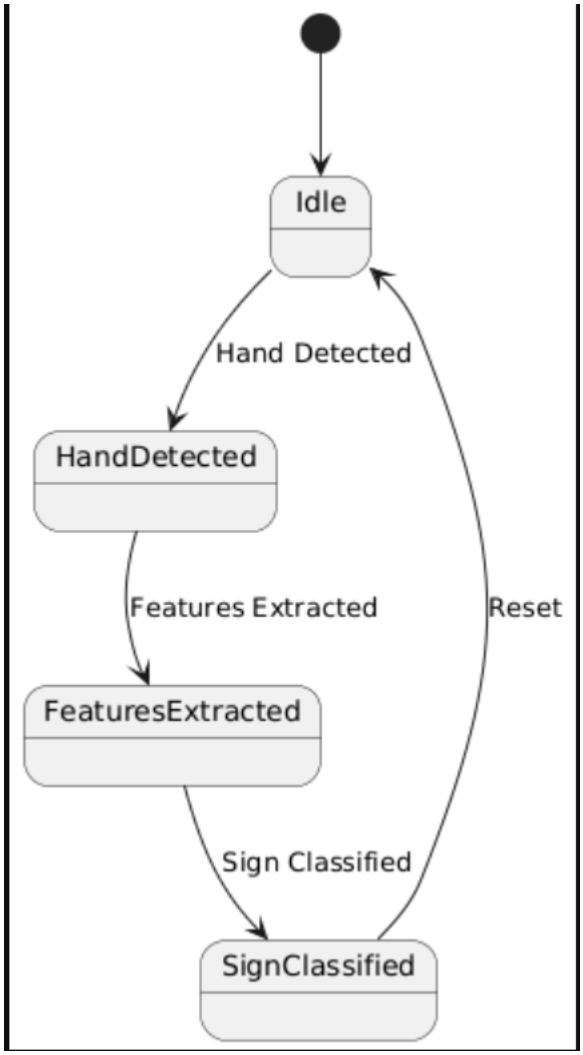


Figure 3.6: State Diagram for SignScript

3.1.3.3. Process Modelling Using Activity Diagram

The activity diagram for SignScript represents a basic flow of activities in the system. The process starts with the user capturing a hand image. The system then detects the hand using OpenCV, extracts the hand landmarks (features), and classifies the hand sign using a Random Forest model. Finally, the classified sign is displayed to the user. This activity diagram is simple and does not include error handling, preprocessing, or validation steps.

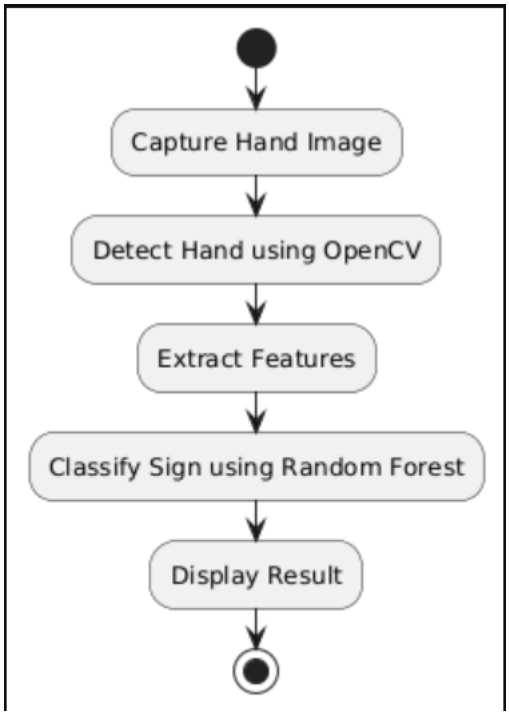


Figure 3.7: Activity Diagram for SignScript

3. 1.3.4. Data modelling Using ER Diagram

The Entity-Relationship (ER) diagram for SignScript represents the relationships between the key entities in the system. The main entities are: Hand: Represents the hand detected by OpenCV. Features: Represents the coordinates of hand landmarks extracted by the system.Sign: Represents the classified hand sign (label).

A Hand can have multiple set of features so the relation between them is 1:M. Features can be classified into a corresponding label so each set of features is mapped to a single label output, so the relation between them is 1:1.

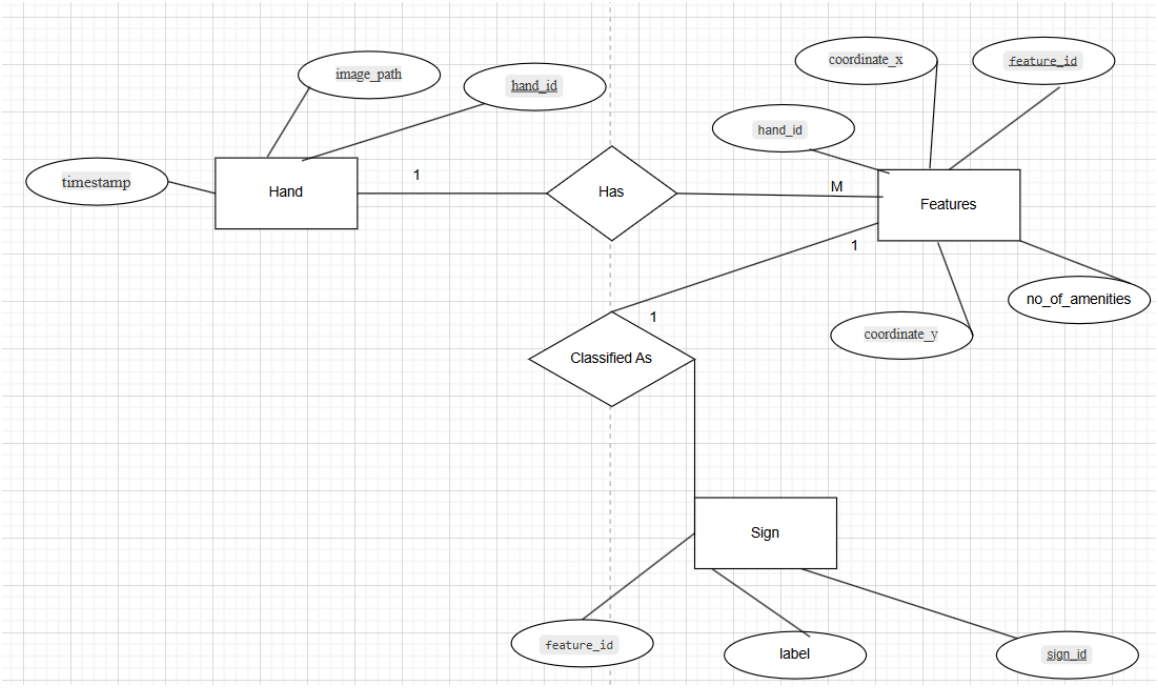


Figure 3.8: ER Diagram for SignScript

3. 1.3.5. Process modelling Using Data Flow Diagrams

In the following diagram, a rectangular box represents the User and the SignScript System, while a circular block represents the processes within the system. The user interacts with the system by capturing a hand image, which is then processed by the system. The system performs three main functions: Hand Detection, Feature Extraction, and Sign Classification.

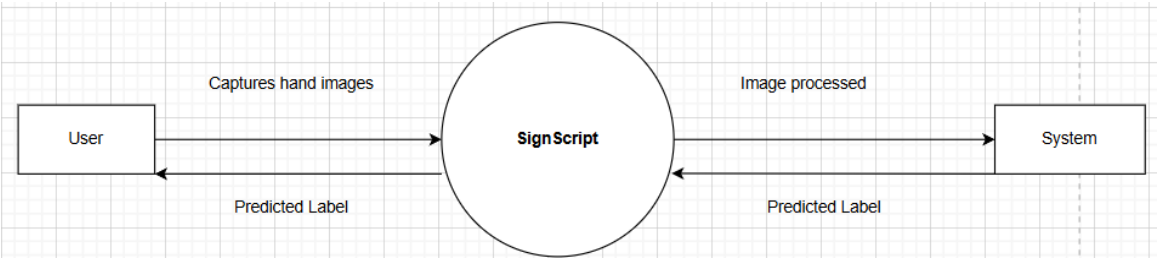


Figure 3.9: DFD Level 0 for SignScript

In DFD Level 1 of the SignScript system, the User captures a hand image, which flows into the SignScript Framework. Inside this framework, the Preprocessing Module prepares the image (e.g., resizing or normalization) before passing it to the Hand Detection Module, which uses OpenCV to detect the hand. The detected hand is then sent to the Feature

Extraction Module, where hand landmarks (features) are extracted. These features are passed to the Sign Classification Module, which uses a Random Forest model to classify the hand sign. The classified sign (label) is sent back to the User. Additionally, a Database stores the hand images, extracted features, and classification results for future reference or analysis.

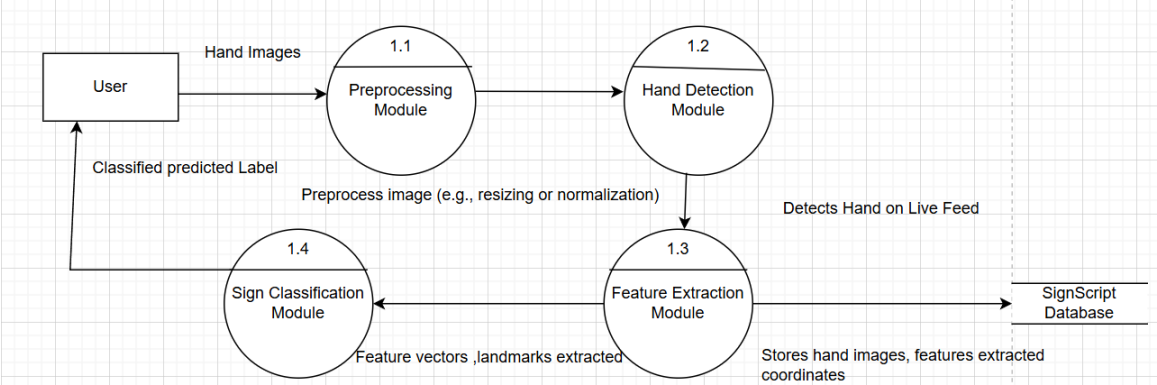


Figure 3.10: DFD Level 1 for SignScript

3.2. Software and hardware Requirement

3.2.1. Software Requirement

Following are the software requirements necessary for the project:

- Python Programming Language.
- PyCharm
- VisualStudio Code
- Jupyter or Google Colab
- Browser (Eg. Chrome, Microsoft Edge)
- Windows/macOS (Operating System)

3.2.2. Hardware Requirement

Followings are the hardware requirements necessary for this project.

- Computer Device (Desktop/Laptop)
- Mobile with Android OS,IOS
- Processor: Minimum Intel Core i3 / AMD Ryzen 5 (Recommended: Intel Core i5 or i7 / Ryzen 7 for faster processing)
- RAM: 4GB (Minimum), 8GB to 16GB or more (Recommended for Deep learning model training)

CHAPTER 4

SYSTEM DESIGN

4.1. UML Diagram

UML diagrams are used to visually represent the SignScript system, illustrating its main actors, roles, actions, artifacts, and classes. We have detailed the following types of UML diagrams:

- Class Diagram
- Activity Diagram
- Sequence Diagram
- Component Diagram
- Deployment Diagram

4.1.1. Class Diagram

The refined class diagram for SignScript outlines a structured process for hand sign recognition, comprising three main classes: Hand, Features, and Sign. The Hand class detects and stores hand data, the Features class extracts key characteristics (e.g., coordinates), and the Sign class classifies the features into specific hand signs. Relationships include a one-to-many link between Hand and Features (one hand can have multiple features) and a one-to-one link between Features and Sign (each feature set corresponds to one sign). This modular design ensures efficient detection, extraction, and classification of hand signs.

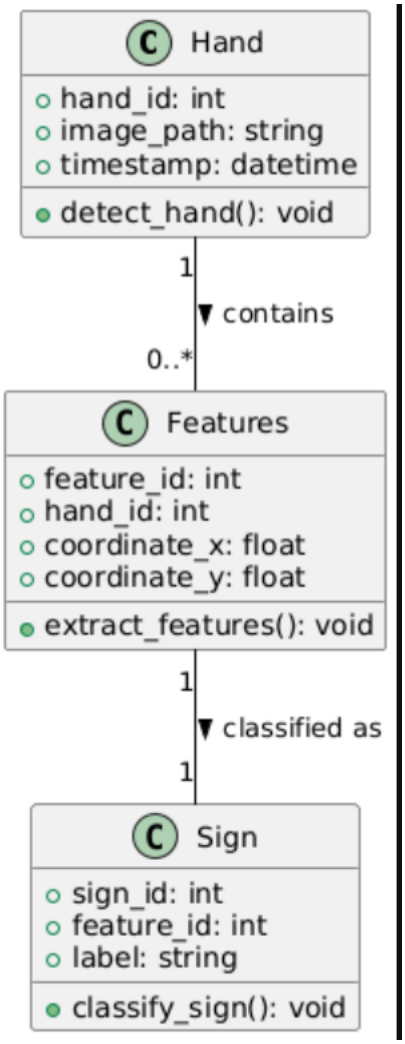


Figure 4.1: Refined Class Diagram for SignScript

4.1.2. Object Diagram

The refined object diagram clearly illustrates the relationships between the Hand1 object, its associated feature sets (Features1), and the corresponding classified signs (Sign1 and Sign2). This diagram provides a detailed snapshot of the SignScript system's state at a specific point in time, showing how data flows from hand detection to feature extraction and finally to sign classification. The Hand1 object is associated with two feature sets (Features1 and Features2) through a one-to-many relationship. This indicates that a single hand can have multiple sets of features extracted from it.

Features1 to Sign1 .Each feature set is classified into a corresponding sign through a one-to-one relationship.Features1 is classified as Sign1 with the label "Hudaina".

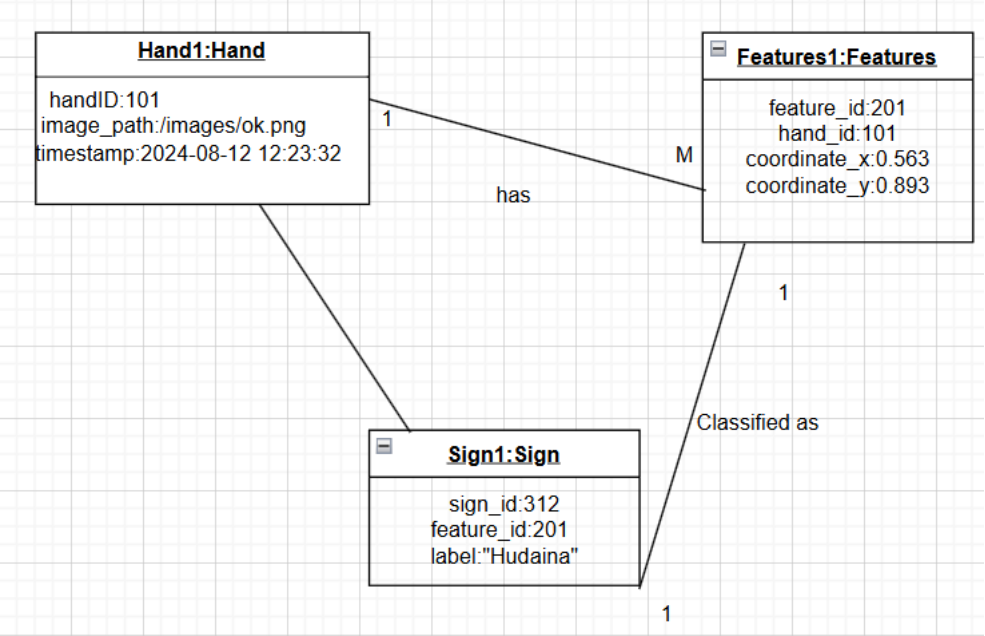


Figure 4.2: Refined Object Diagram for SignScript

4.1.3. Sequence Diagram

The refined sequence diagram for SignScript includes additional steps and error handling to make the system more robust and realistic. The user captures a hand image, which is first preprocessed (e.g., resized or normalized) by the Preprocessing module before being passed to the Hand Detection module. The Hand Detection module detects the hand and validates whether the detection was successful. If successful, the detected hand is sent to the Feature Extraction module, which extracts the hand landmarks and validates the features. If the features are valid, they are passed to the Sign Classification module, which classifies the hand sign and displays the result to the user. If any step fails (e.g., hand detection or feature extraction fails), an error message is displayed to the user..

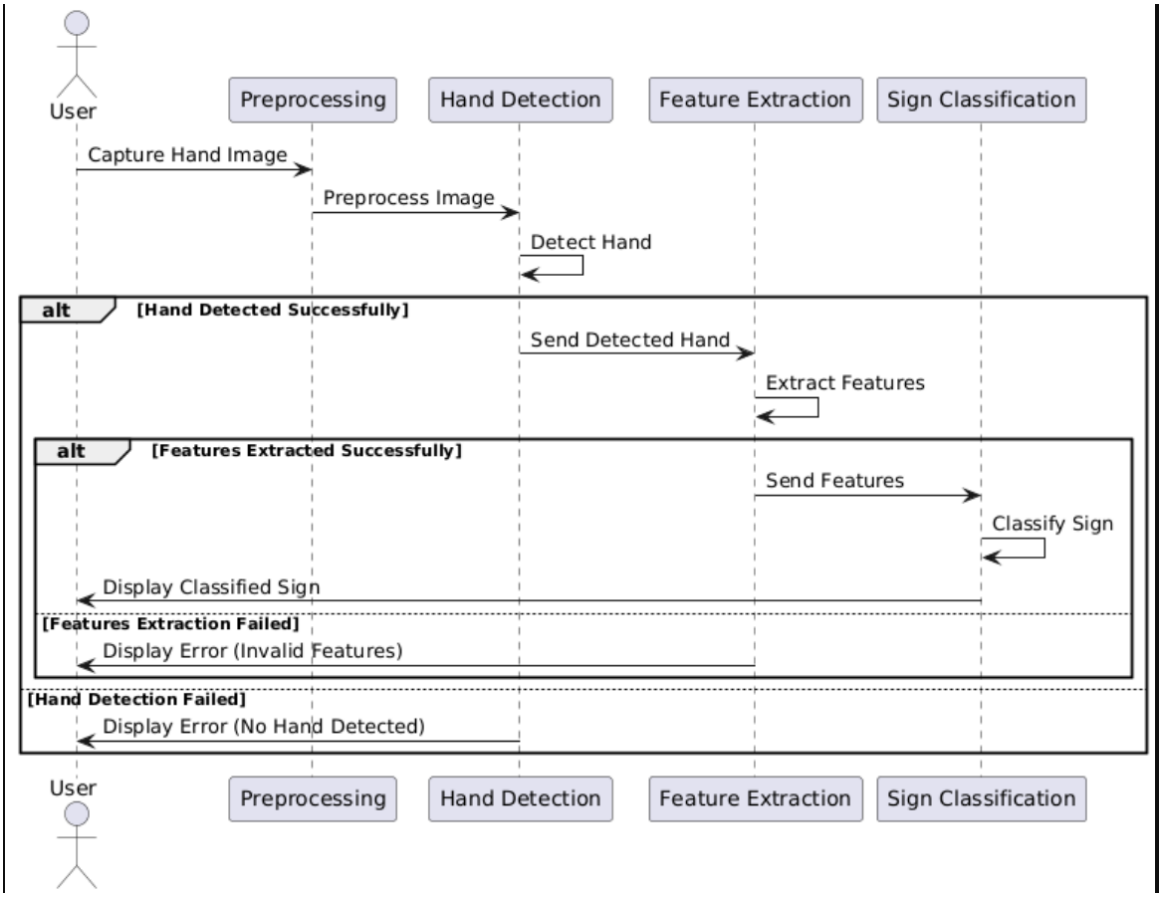


Figure 4.3: Refined Sequence Diagram for SignScript

4.1.4. State Diagram

The refined state diagram for SignScript includes additional states and transitions to handle errors and intermediate steps, making the system more robust and realistic. The system starts in the Idle state, waiting for user input. When a hand image is captured, it transitions to the Preprocessing state, where the image is prepared for detection. If preprocessing is successful, the system moves to the Hand Detected state. If hand detection fails, it transitions to the Error: No Hand Detected state and returns to Idle. If hand detection is successful, the system transitions to the Features Extracted state. If feature extraction fails, it moves to the Error: Invalid Features state and returns to Idle. If feature extraction is successful, the system transitions to the Sign Classified state, where the hand sign is classified and displayed. After classification, the system returns to the Idle state.

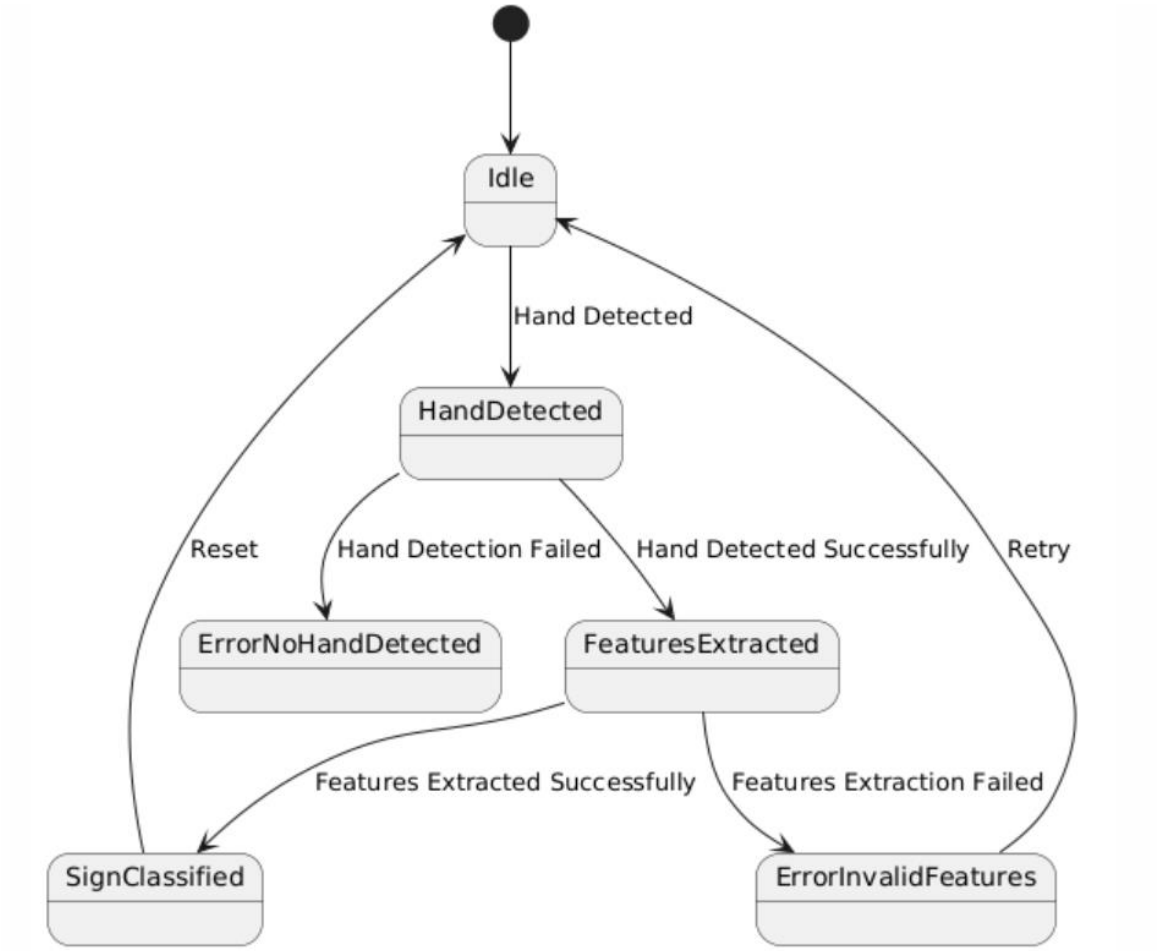


Figure 4.4: Refined State Diagram for SignScript

4.1.5. Activity Diagram

The refined activity diagram for SignScript includes additional activities and decision points to handle errors and intermediate steps, making the system more robust and realistic. The process starts with the user capturing a hand image, which is then preprocessed (e.g., resized or normalized). The system attempts to detect the hand using OpenCV. If hand detection fails, an error message is displayed, and the process ends. If successful, the system extracts the hand landmarks (features). If feature extraction fails, an error message is displayed, and the process ends. If feature extraction is successful, the system classifies the hand sign using a Random Forest model and displays the result to the user.

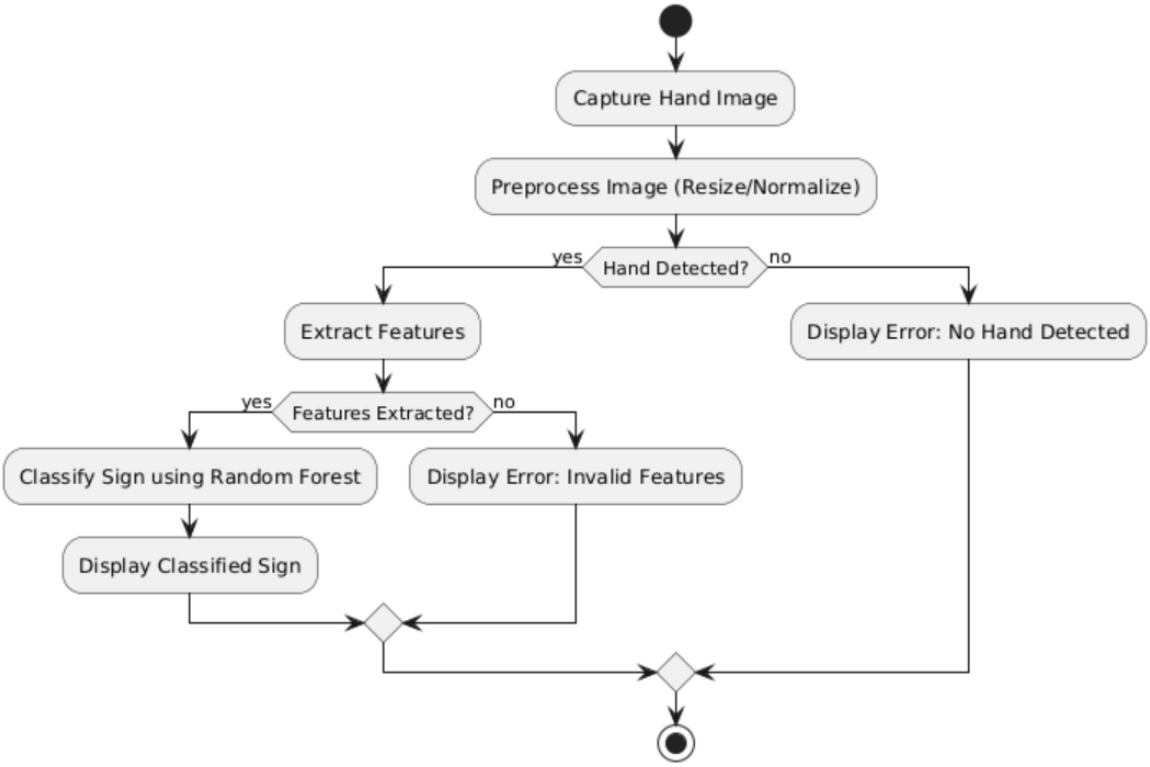


Figure 4.5: Refined Activity Diagram for SignScript

4.1.6. Component Diagram

The SignScript component diagram shows how the various components of the system, including users, sign language recognition, video processing, predictions, notifications, and database, interact with each other. It shows how the SignScript system is divided into independent components and how these components interact through defined interfaces. The diagram also highlights the external services that the system relies on, such as the video storage, the machine learning model for sign language recognition, and the database that stores user data and predictions. This helps to understand the overall structure and flow of data in the various parts of the system.

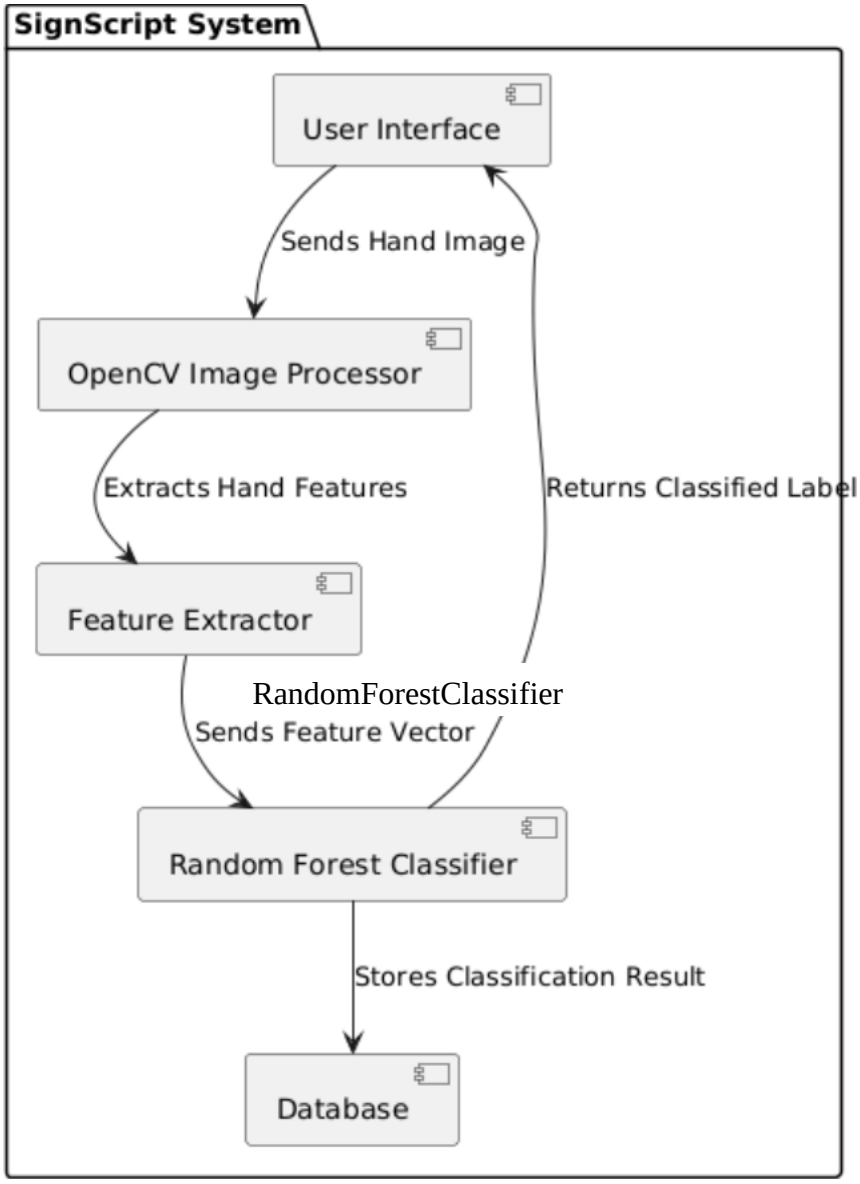


Figure 4.6: Component Diagram for SignScript

4.1.7. Deployment Diagram

The deployment diagram of the SignScript project represents the physical architecture of the system, focusing on the deployment and interaction of key components such as Camera, OpenCV, and RandomForestClassifier. It illustrates how these components, along with other services, function in a real-world environment. User Device → Camera: The user interacts with the system via a camera device, capturing sign language video. Camera → OpenCV: The video feed from the camera is sent to the OpenCV block, where it is processed. OpenCV → RandomForestClassifier: The processed frames are then forwarded

to the RandomForestClassifier block, which applies machine learning models for sign language recognition and prediction.

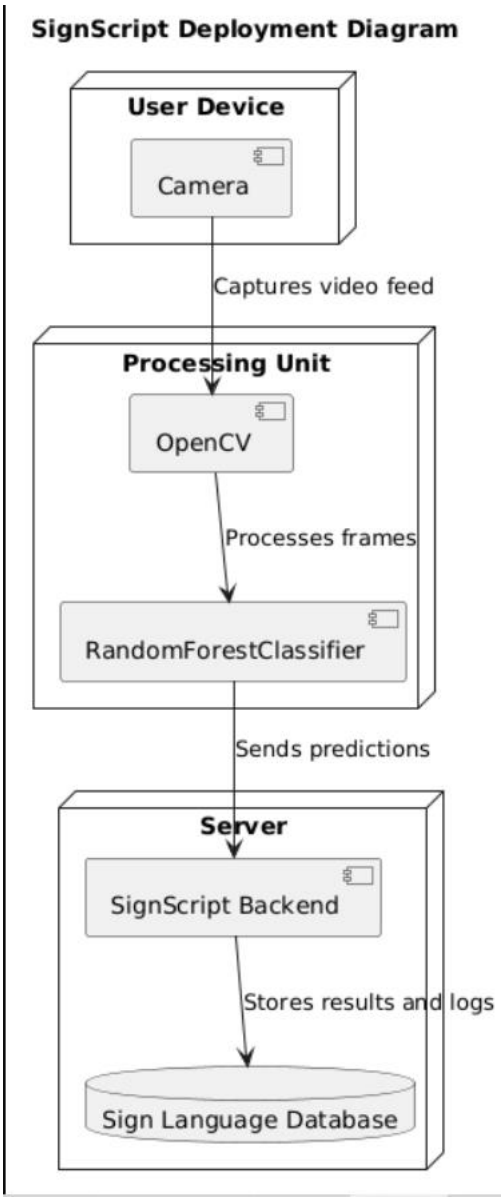


Figure 4.7: Deployment Diagram for SignScript

CHAPTER 5

IMPLEMENTATION AND TESTING

5.1. Implementation

5.1.1. Tools Used

Different tools and technologies have been used to implement this application. They are listed in the table below:

Table 5.1 Tools Used

Project Management Tools	ProjectLibre	ProjectLibre helped in our SignScript analysis project by enabling efficient task scheduling, resource allocation, and progress tracking through Gantt charts and work breakdown structures.
Programming Language	Python	Python played a key role in our SignScript analysis by providing powerful libraries like OpenCV and scikit-learn for deep learning, enabling efficient model training,and gesture recognition.
Diagramming Tools	diagrams.net/ draw.io	It helped visualize and design the flow and structure of the SignScript analysis project, aiding in clear planning and communication.
IDE	Visual Studio Code, Google Colab, Jupyter	The IDE provided a platform for writing, testing, and debugging the code efficiently, streamlining the development of the SignScript analysis project.

5.1.2. Algorithm Description

Random Forest is an ensemble learning technique that builds multiple decision trees and combines their outputs to improve accuracy and robustness. The algorithm is particularly effective for classification tasks as it reduces overfitting and enhances generalization.

5.1.2.1. Random Forest Classification

- **Step 1: Gathering and Preparing Data**

  The dataset used for training and testing the model contains labeled images of different sign language gestures. Each image is preprocessed and transformed into a feature vector to be used as input for classification. The dataset is split into training and testing sets to ensure that the model's performance can be properly evaluated on unseen data.

- **Step 2: Model Definition and Structuring**

  The Random Forest Classifier was defined as the base model. This algorithm constructs multiple decision trees and averages their predictions to provide a more stable and accurate output. Random Forests are particularly effective in handling high-dimensional datasets and noisy data [7]. We are seeing how the decision tree and random forest work mathematically:

  A decision tree is a tree-like model where each internal node represents a decision based on a feature, and each leaf node represents a predicted output (in the case of regression, this would be a continuous value). The tree makes predictions by recursively splitting the data based on the feature values that best reduce the variance at each node.

  Decision Tree for Classification

  A decision tree is a tree-like structure where each internal node represents a decision based on a feature, and each leaf node represents a class label. The tree makes predictions by recursively splitting the data based on the feature values that best separate the classes.

- **Splitting Criterion**: The decision tree uses information gain or Gini impurity as the criterion for splitting. The goal at each node is to maximize information gain or minimize impurity.

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

Where:

- C is the total number of classes,
- p_i is the proportion of samples belonging to class i.

A Random Forest is an ensemble method that combines multiple decision trees to improve classification accuracy and robustness. It uses the bagging (Bootstrap Aggregating) technique, where multiple decision trees are trained on different random subsets of the data. Each tree makes an independent classification, and the final output is the average of these individual classification.

- **Bootstrap Sampling**: Each tree in the random forest is trained on a bootstrap sample of the data, meaning that each tree uses a random sample with replacement from the original training data.

$$D_{bootstrap} = \{d_1, d_2, \ldots., d_m\} \ ,$$

d_i belongs D(data points are chosen with replacement)

In classification problem, the final classification from the random forest is the average of the classification made by all the individual trees:

$$y' = \frac{1}{N}\sum_{i=1}^{N} yi'$$

Where:

N is the number of decision trees in the forest

yi' the classification label from the i-th decision tree

y' is the final classification value

- **Step 3: Final Prediction in Classification**

  The final classification output is determined by majority voting across all decision trees.

$$y^* = \arg\max_c \sum_{i=1}^{N} I(y_i = c)$$

Where:

- N is the number of decision trees,
- y_i is the predicted class label from the i-th decision tree,
- $I(y_i = c)$ is an indicator function that equals 1 if the predicted label is c , otherwise 0.

5.2. Testing

Testing, within the setting of software improvement, is the method of assessing a framework or application to guarantee that it meets indicated necessities and capacities accurately. It is a basic portion of the computer program advancement lifecycle aimed at distinguishing surrenders, mistakes, or bugs within the program.

5.2.1. Unit Testing

Testing, within the setting of software improvement, is the method of assessing a framework or application to guarantee that it meets indicated necessities and capacities accurately. It is a basic portion of the computer program advancement lifecycle aimed at distinguishing surrenders, mistakes, or bugs within the program. Test case for the unit testing are:

Table 5.2 Unit Testing

Test Case	Test Data	Expected Result	Actual Result	Result
1	Image “ka.jpg” was taken.	Label = Ka	Label = Ka	Pass
2	Image “gha.jpg” was taken.	Label = Gha	Label = Gha	Pass
3	Image “nga.jpg” was taken.	Label = Nga	Label = Nga	Pass

5.2.2. System Testing

A comprehensive and integrated software system is assessed against its predetermined requirements at the system testing stage of software testing. It entails testing the entire system to make sure it meets all functional and non-functional requirements and operates as intended. System testing confirms that the program works as planned and is prepared for end users to access.

Test Cases for System Testing:

Computer Camera Vision Test

- Verify that the computer camera vision functions properly and is integrated to Scikit-learn.

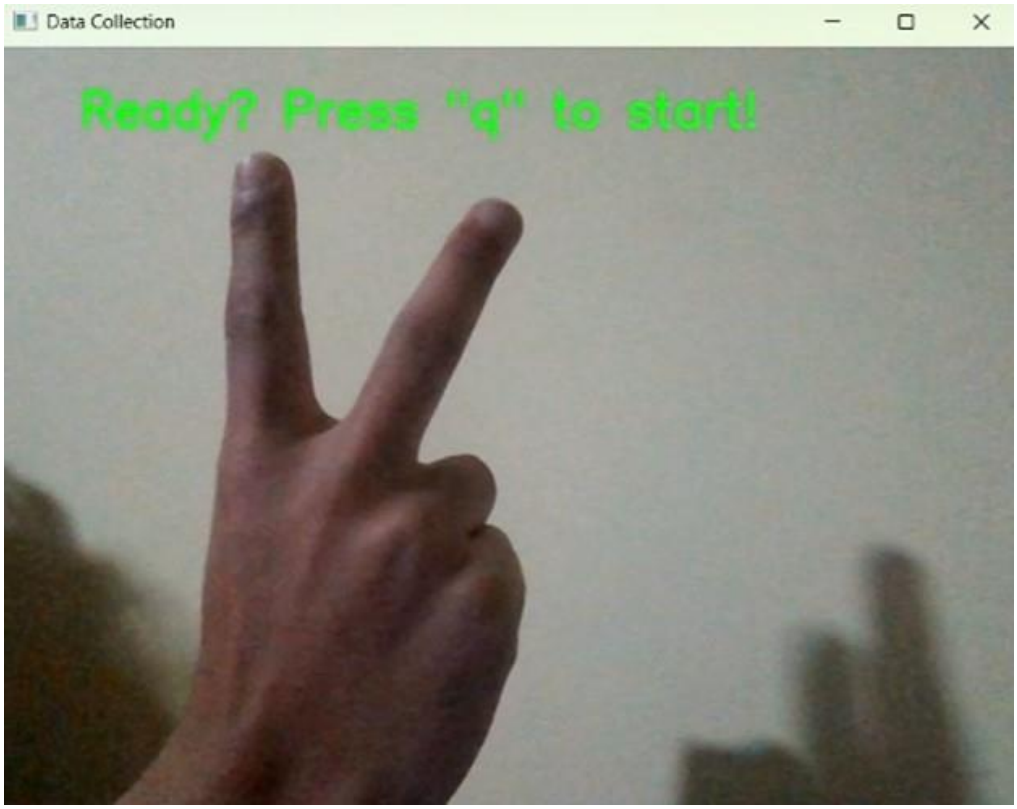


Figure 5.1: Camera Vision Test for SignScript

Caption Generation Test

- Verify that the system generates accurate captions in context to the recognized gesture.
- The following image predicts a label “Gha” which was the expected output, indicating the model is working fine.

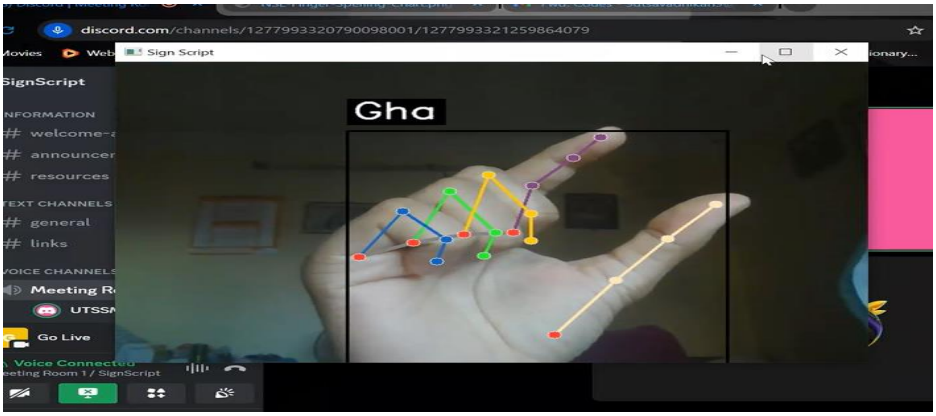


Figure 5.2: Caption Test 1 for SignScript

- The following image predicts a label “Sahayog”, which was the expected output, indicating the model is working fine.

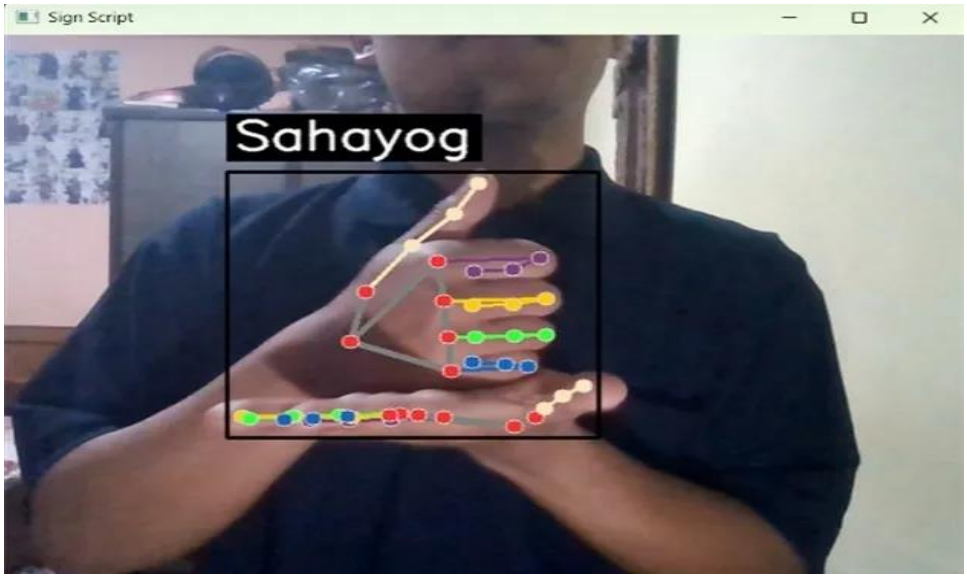


Figure 5.3: Caption Test 2 for SignScript

5.3. Result Analysis

The framework was tested through unit testing and demonstrated to be successful in executing its aiming functions. The results appeared that the project was able to meet its objectives, but there is still room for advancement in terms of growing the system's capabilities and expanding community association.

5.3.1. Evaluating Accuracy

In machine learning, accuracy may be a common metric utilized to assess the execution of a classifier show. Accuracy measures the extent of accurately classified instances among all occurrences within the dataset. To calculate precision, the primary step is to partition the dataset into two parts: a preparing set and a test set. The preparing set is utilized to train the demonstrate, while the test set is utilized to assess the model's execution.

In classifier demonstrate the foremost common degree to assess precision are:

- **Precision**: The exactness measures the extent of accurately distinguished sign dialect motions or developments among all the signals or developments that the demonstrate anticipated as having a place to a specific sign.

The equation for accuracy is:

Precision = True Positives / (True Positives + False Positives)

- **Recall**: The recall measures the extent of accurately distinguished sign dialect motions or developments among all the genuine sign dialect signals or developments display within the dataset.

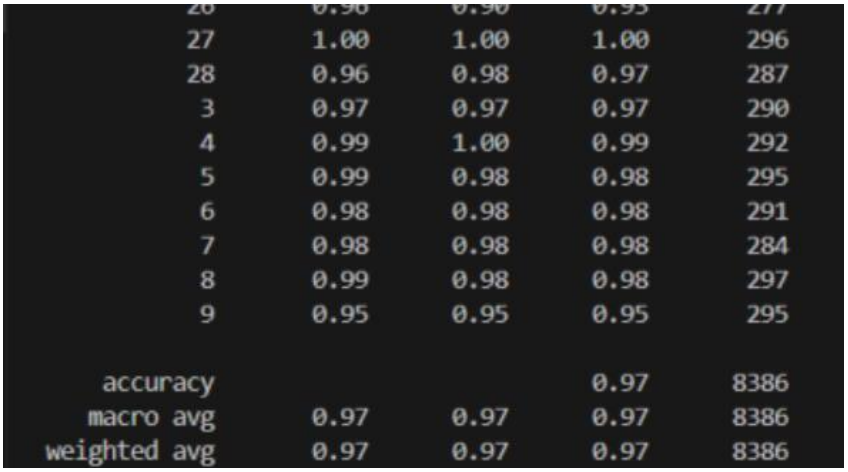
  The equation for recall is:

  Recall = True Positives / (True Positives + False Negatives)

- **F1 score**: The F1 score gives a single metric that equalizations the trade-off between exactness (precisely distinguishing sign dialect signals) and review (identifying most of the genuine sign dialect signals display).

  The equation for F1 score is:

  F1 score = 2 * (Precision * Recall) / (Precision + Recall)



27	1.00	1.00	1.00	296
28	0.96	0.98	0.97	287
3	0.97	0.97	0.97	290
4	0.99	1.00	0.99	292
5	0.99	0.98	0.98	295
6	0.98	0.98	0.98	291
7	0.98	0.98	0.98	284
8	0.99	0.98	0.98	297
9	0.95	0.95	0.95	295
accuracy			0.97	8386
macro avg	0.97	0.97	0.97	8386
weighted avg	0.97	0.97	0.97	8386

Figure 5.4: Classification Report for SignScript

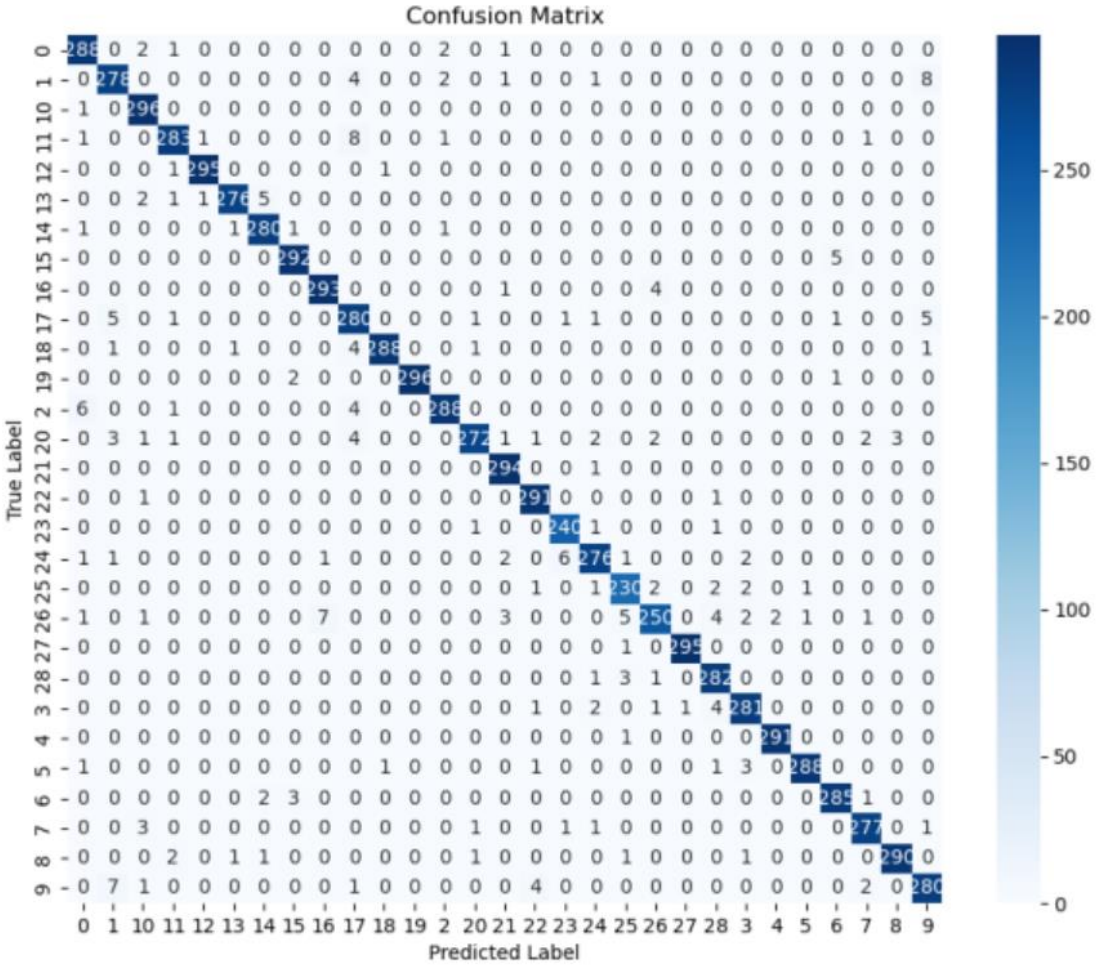


Figure 5.5: Confusion Matrix for SignScript

The matrix is typically structured as follows:

True Positives (TP): The number of instances where the model correctly predicted the class of a sign language gesture.

False Positives (FP): The number of instances where the model incorrectly predicted the class, classifying a gesture as belonging to a particular sign when it does not.

True Negatives (TN): The number of instances where the model correctly predicted that a gesture does not belong to a particular class.

False Negatives (FN): The number of instances where the model failed to identify a correct gesture and incorrectly classified it as belonging to a different class.

CHAPTER 6

CONCLUSION AND FUTURE RECOMMENDATIONS

6.1 Conclusion

To sum up, the "SignScript" Real-Time Sign Language Recognition project viably outlines the application of machine learning in sign language recognition. By utilizing Random Forest for classification and leveraging Scikit-Learn for information preprocessing and demonstrate assessment, the system effectively classifies hand signals. The dataset, which incorporates images with both plain and complex backgrounds, upgrades the model's strength and generalization.

SignScript demonstrates to be an important tool by guaranteeing adaptability and persistent execution change through optimized Random Forest models and dataset extension. This venture lays the establishment for future progressions, cultivating moved forward openness and communication for the hard of hearing and hard-of-hearing community.

Continuous investigate in this range proceeds to improve the capabilities of machine learning, and Random Forest models for sign language recognition are getting to be progressively modern.

As SignScript evolves with future updates, it is set to redefine the way sign language recognition is performed, offering versatility and accessibility for seamless communication.

6.2 Future Recommendations

To further improve the usefulness and precision of SignScript, the following advancements can be considered:

- **Expansion to Dynamic Gesture Recognition:**Implement recognition for dynamic hand signals, empowering the system to translate full words and sentences instead of just static signs.

- **Integration of Transformer-Based Models**:Join more advanced profound learning structures like Vision Transformers (ViTs) or Spatio-Temporal CNNs for improved feature extraction and precision.
- **Nepali Font Rendering in OpenCV**:Create Overcome OpenCV's limitation in rendering Nepali fonts by coordination PIL (Pad) or HarfBuzz for superior content visualization within the framework. System.
- **Mobile and Web Deployment**:Create a mobile or web-based interface to form the framework more available to clients, possibly employing a Flask/Django API for backend communication with React.js or another front-end system in the future when new versions of Scikit-Learn supports conversion of the code.
- **Dataset Expansion and Augmentation**:Increase dataset differing qualities by counting distinctive lighting conditions, hand introductions, and shifted foundations to progress demonstrate generalization.
- **Real-Time Performance Optimization**:Execute demonstrate quantization procedures such as TensorRT or ONNX to optimize deduction speed for real-time applications.
- **Sign-to-Speech Implementation:** Coordinated a Text-to-Speech (TTS) framework to change over recognized motions into talked words, improving availability for communication.
- **Support for Multiple Sign Languages:**Expand the system's capabilities past Nepali Sign Dialect to incorporate other sign dialects such as American Sign Language (ASL) or Indian Sign Language (ISL).

REFERENCES

[1] Jamie Berke, James Lacy, “Hearing loss/deafness| Sign Language," 2021

[2] T. Koller, O. Zargaran, and H. Ney, "Deep Sign: Enabling Robust Sign Language Recognition with Neural Machine Translation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 11, pp. 2891–2904, Nov. 2019.

[3] Z. Zhang, P. Cui, and W. Liu, "A CNN-LSTM Hybrid Model for Dynamic Sign Language Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 12345–12354.

[4] . Joze and T. Koller, "Self-Supervised Learning for Sign Language Recognition Using Large-Scale Video Datasets," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, Canada, Oct. 2021, pp. 4501–4510.

[5] L. Pigou, A. van den Oord, S. Dieleman, M. Van Herreweghe, and J. Dambre, "Beyond Hands: Using Facial Expressions and Body Language for Sign Language Recognition," in *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, Xi'an, China, May 2018, pp. 780–787.

[6] H. Cooper, B. Holt, and R. Bowden, "Sign Language Recognition," *Journal of Machine Learning Research*, vol. 13, pp. 1-28, 2012.

[7] S. M. Prasanna, G. S. Thangamalai, and A. Bharathi, "Hand Gesture Recognition for Sign Language: A Survey," *IEEE Access*, vol. 9, pp. 104066-104087, 2021.

[8] Vivek Thapa, Jhuma Sunuwar, and Ratika Pradhan, "Finger spelling recognition for Nepali sign language." "In Recent Developments in Machine Learning and Data Analytics," pages 219-227. Springer, 2019.

[9] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

Appendices A: Code

```
# Collect Images

import os
import cv2
import sys

DATA_DIR = './data'
NUMBER_OF_CLASSES = 3
DATASET_SIZE = 200
WINDOW_NAME = "Sign Script Data Collection"

def create_directory(path: str):
    try:
        os.makedirs(path, exist_ok=True)
    except OSError as e:
        print(f"Error: Unable to create directory {path}: {e}")
        sys.exit(1)

def initialize_data_directories():
    create_directory(DATA_DIR)
    for class_id in range(NUMBER_OF_CLASSES):
        class_dir = os.path.join(DATA_DIR, str(class_id))
        create_directory(class_dir)

def collect_images_for_class(cap: cv2.VideoCapture, class_id: int):
    class_dir = os.path.join(DATA_DIR, str(class_id))
    print(f"\nCollecting data for class {class_id}.")
    print("Press 'q' to start capturing images when ready.")

    while True:
        ret, frame = cap.read()
        if not ret:
            print("Failed to grab frame. Exiting...")
            break

        instruction = 'Ready? Press "q" to start!'
        cv2.putText(frame, instruction, (50, 50), cv2.FONT_HERSHEY_SIMPLEX,
                    1, (0, 255, 0), 2, cv2.LINE_AA)
        cv2.imshow(WINDOW_NAME, frame)

        if cv2.waitKey(25) & 0xFF == ord('q'):
            break

    counter = 0
    print(f"Capturing {DATASET_SIZE} images for class {class_id}...")
    while counter < DATASET_SIZE:
        ret, frame = cap.read()
        if not ret:
```

```
            print("Failed to grab frame. Skipping this frame...")
            continue

        cv2.imshow(WINDOW_NAME, frame)
        key = cv2.waitKey(25) & 0xFF

        if key == 27:
            print("Escape pressed. Exiting capture for this class.")
            break

        image_path = os.path.join(class_dir, f"{counter}.jpg")
        cv2.imwrite(image_path, frame)
        counter += 1

    print(f"Finished capturing images for class {class_id}.")

def main():
    initialize_data_directories()

    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open video capture device.")
        sys.exit(1)

    cv2.namedWindow(WINDOW_NAME)

    try:
        for class_id in range(NUMBER_OF_CLASSES):
            collect_images_for_class(cap, class_id)
    except KeyboardInterrupt:
        print("\nData collection interrupted by user.")
    finally:
        cap.release()
        cv2.destroyAllWindows()
        print("Released video capture and closed all windows.")

if __name__ == "__main__":
    main()
```

```
# Create Dataset
import os
import sys
import pickle
import cv2
import mediapipe as mp
import numpy as np

DATA_DIR = './data'
OUTPUT_PICKLE = 'data.pickle'
MIN_DETECTION_CONFIDENCE = 0.3

mp_hands = mp.solutions.hands
hands_detector = mp_hands.Hands(static_image_mode=True,
                        min_detection_confidence=MIN_DETECTION_CONFIDENCE)

def process_image(image_path: str) -> list:
    img = cv2.imread(image_path)
    if img is None:
        print(f"Warning: Could not read image '{image_path}'. Skipping.")
        return []

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    results = hands_detector.process(img_rgb)

    if not results.multi_hand_landmarks:
        print(f"Info: No hand landmarks detected in '{image_path}'. Skipping.")
        return []

    hands_list = results.multi_hand_landmarks
    features = []

    def extract_features(hand_landmarks):
        x_coords = [lm.x for lm in hand_landmarks.landmark]
        y_coords = [lm.y for lm in hand_landmarks.landmark]
        min_x, min_y = min(x_coords), min(y_coords)
        feat = []
        for x, y in zip(x_coords, y_coords):
            feat.append(x - min_x)
            feat.append(y - min_y)
        return feat

    if len(hands_list) >= 2:
        sorted_hands = sorted(hands_list, key=lambda hand: hand.landmark[0].x)
        features += extract_features(sorted_hands[0])
        features += extract_features(sorted_hands[1])
    elif len(hands_list) == 1:
        features += extract_features(hands_list[0])
        features += [0] * 42
    else:
```

```
        return []

    if len(features) != 84:
            print(f"Warning: Feature vector length {len(features)} != 84 for image
'{image_path}'. Skipping.")
        return []

    return features
def create_dataset(data_dir: str) -> (list, list):
    dataset = []
    labels = []
    if not os.path.isdir(data_dir):
        print(f"Error: Data directory '{data_dir}' does not exist.")
        sys.exit(1)
    # Process each class folder.
    for class_label in sorted(os.listdir(data_dir)):
        class_dir = os.path.join(data_dir, class_label)
        if not os.path.isdir(class_dir):
            print(f"Skipping non-directory item '{class_dir}'.")
            continue
        print(f"Processing class '{class_label}' in directory '{class_dir}'...")
        image_files = sorted(os.listdir(class_dir))
        for img_filename in image_files:
            img_path = os.path.join(class_dir, img_filename)
            feat_vector = process_image(img_path)
            if feat_vector:
                dataset.append(feat_vector)
                labels.append(class_label)

    return dataset, labels

def save_dataset(dataset: list, labels: list, output_path: str):
    try:
        with open(output_path, 'wb') as f:
            pickle.dump({'data': dataset, 'labels': labels}, f)
        print(f"Dataset saved successfully to '{output_path}'.")
    except Exception as e:
        print(f"Error saving dataset: {e}")

def main():
    print("Starting dataset creation...")
    data, labels = create_dataset(DATA_DIR)
    if not data:
        print("No valid data was processed. Exiting.")
        sys.exit(1)
    save_dataset(data, labels, OUTPUT_PICKLE)
    print("Dataset creation completed.")

if __name__ == '__main__':
    main()
```

```
# Training Classifier
# import pickle

# from sklearn.ensemble import RandomForestClassifier
# from sklearn.model_selection import train_test_split
# from sklearn.metrics import accuracy_score
# import numpy as np


# data_dict = pickle.load(open('./data.pickle', 'rb'))

# data = np.asarray(data_dict['data'])
# labels = np.asarray(data_dict['labels'])

# x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True, stratify=labels)

# model = RandomForestClassifier()

# model.fit(x_train, y_train)

# y_predict = model.predict(x_test)

# score = accuracy_score(y_predict, y_test)

# print('{}% of samples were classified correctly !'.format(score * 100))

# f = open('model.p', 'wb')
# pickle.dump({'model': model}, f)
# f.close()


# train_classifier.py
import os
import sys
import pickle
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Configuration
DATA_PICKLE_PATH = './data.pickle'
MODEL_OUTPUT_PATH = 'model.p'
TEST_SIZE = 0.2
RANDOM_STATE = 42

def load_data(data_path: str):
    """Load dataset from a pickle file."""
    if not os.path.exists(data_path):
```

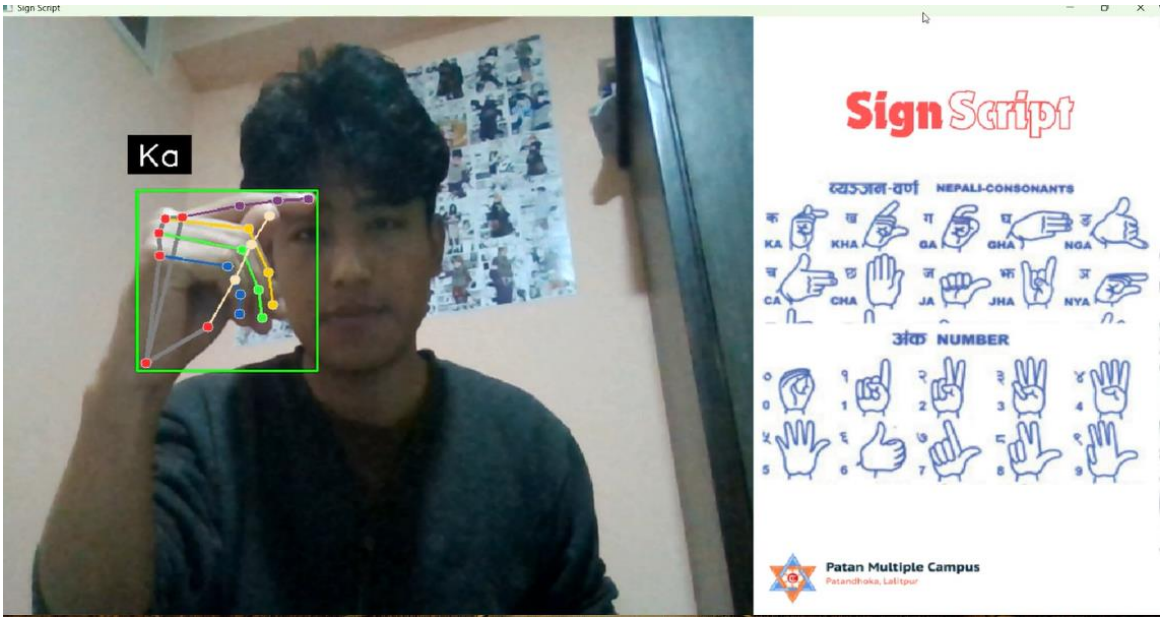
```
        print(f"Error: Data file '{data_path}' does not exist.")
        sys.exit(1)

    try:
        with open(data_path, 'rb') as f:
            data_dict = pickle.load(f)
        data = np.asarray(data_dict.get('data', []))
        labels = np.asarray(data_dict.get('labels', []))
        if data.size == 0 or labels.size == 0:
            print("Error: Loaded data or labels are empty.")
            sys.exit(1)
        print(f"Data loaded successfully from '{data_path}'.")
        return data, labels
    except Exception as e:
        print(f"Error loading data: {e}")
        sys.exit(1)

def train_classifier(data: np.ndarray, labels: np.ndarray) -> RandomForestClassifier:
    """Train a RandomForest classifier and print the accuracy."""
    try:
        x_train, x_test, y_train, y_test = train_test_split(
            data, labels, test_size=TEST_SIZE, shuffle=True, stratify=labels,
random_state=RANDOM_STATE
        )
    except Exception as e:
        print(f"Error during train/test split: {e}")
        sys.exit(1)

    model = RandomForestClassifier(random_state=RANDOM_STATE)
    try:
        model.fit(x_train, y_train)
    except Exception as e:
        print(f"Error training classifier: {e}")
        sys.exit(1)

    try:
        y_predict = model.predict(x_test)
        score = accuracy_score(y_test, y_predict)
        print(f"Classifier Accuracy: {score * 100:.2f}%")
    except Exception as e:
        print(f"Error during evaluation: {e}")

    return model

def save_model(model, output_path: str):
    """Save the trained model to a pickle file."""
    try:
        with open(output_path, 'wb') as f:
            pickle.dump({'model': model}, f)
        print(f"Model saved successfully to '{output_path}'.")
```

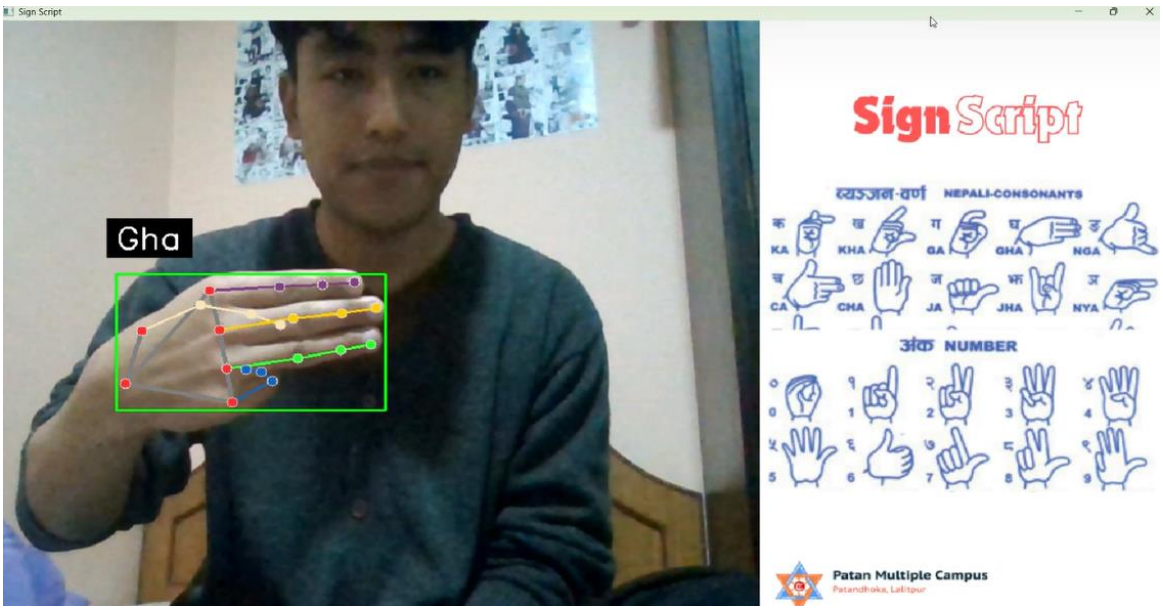
```
    except Exception as e:
        print(f"Error saving model: {e}")

def main():
    data, labels = load_data(DATA_PICKLE_PATH)
    model = train_classifier(data, labels)
    save_model(model, MODEL_OUTPUT_PATH)

if __name__ == '__main__':
    main()


 #Inference Classifier

import os
import sys
import pickle
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Configuration
DATA_PICKLE_PATH = './data.pickle'
MODEL_OUTPUT_PATH = 'model.p'
TEST_SIZE = 0.2
RANDOM_STATE = 42

def load_data(data_path: str):
    """Load dataset from a pickle file."""
    if not os.path.exists(data_path):
        print(f"Error: Data file '{data_path}' does not exist.")
        sys.exit(1)

    try:
        with open(data_path, 'rb') as f:
            data_dict = pickle.load(f)
        data = np.asarray(data_dict.get('data', []))
        labels = np.asarray(data_dict.get('labels', []))
        if data.size == 0 or labels.size == 0:
            print("Error: Loaded data or labels are empty.")
            sys.exit(1)
        print(f"Data loaded successfully from '{data_path}'.")
        return data, labels
    except Exception as e:
        print(f"Error loading data: {e}")
        sys.exit(1)

def train_classifier(data: np.ndarray, labels: np.ndarray) -> RandomForestClassifier:
```

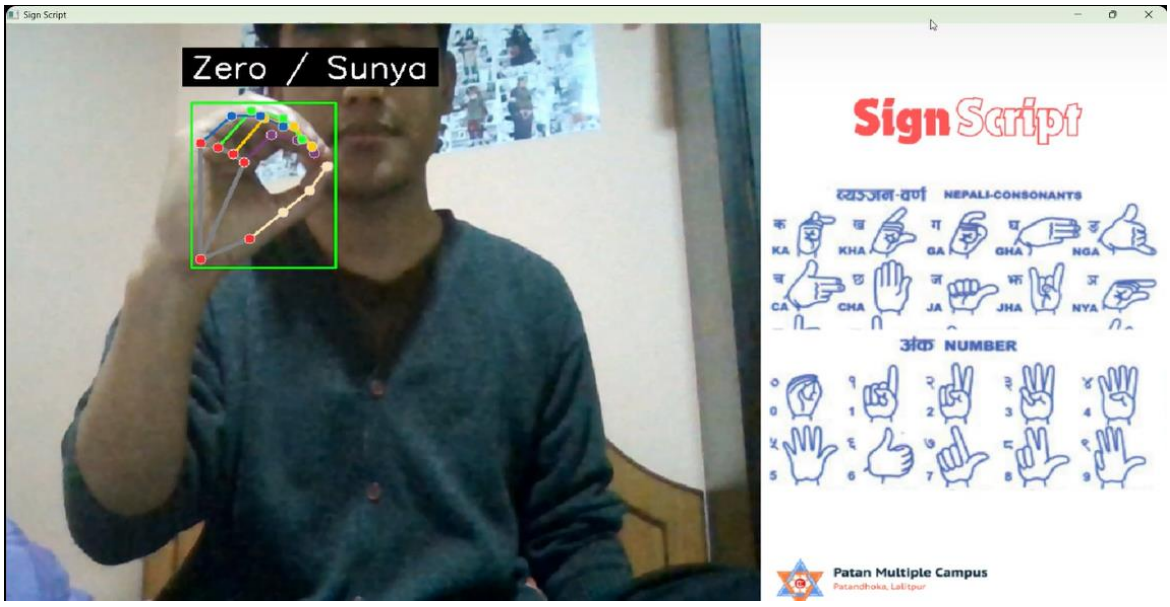
```
    """Train a RandomForest classifier and print the accuracy."""
    try:
        x_train, x_test, y_train, y_test = train_test_split(
            data, labels, test_size=TEST_SIZE, shuffle=True, stratify=labels,
random_state=RANDOM_STATE
        )
    except Exception as e:
        print(f"Error during train/test split: {e}")
        sys.exit(1)

    model = RandomForestClassifier(random_state=RANDOM_STATE)
    try:
        model.fit(x_train, y_train)
    except Exception as e:
        print(f"Error training classifier: {e}")
        sys.exit(1)

    try:
        y_predict = model.predict(x_test)
        score = accuracy_score(y_test, y_predict)
        print(f"Classifier Accuracy: {score * 100:.2f}%")

        # Generate Classification Report
        print("\nClassification Report:\n", classification_report(y_test, y_predict))

        # Plotting Accuracy Graph
        plot_accuracy(y_test, y_predict)

    except Exception as e:
        print(f"Error during evaluation: {e}")

    return model

def plot_accuracy(y_test, y_predict):
    """Plot the training and testing accuracy."""
    accuracy = accuracy_score(y_test, y_predict)
    plt.figure(figsize=(10, 6))
    plt.bar(['Test Accuracy'], [accuracy], color='blue', alpha=0.7)
    plt.title("Test Accuracy of RandomForest Classifier")
    plt.ylabel('Accuracy')
    plt.show()  # Ensure the plot is displayed

def save_model(model, output_path: str):
    """Save the trained model to a pickle file."""
    try:
        with open(output_path, 'wb') as f:
            pickle.dump({'model': model}, f)
        print(f"Model saved successfully to '{output_path}'.")
    except Exception as e:
        print(f"Error saving model: {e}")
```

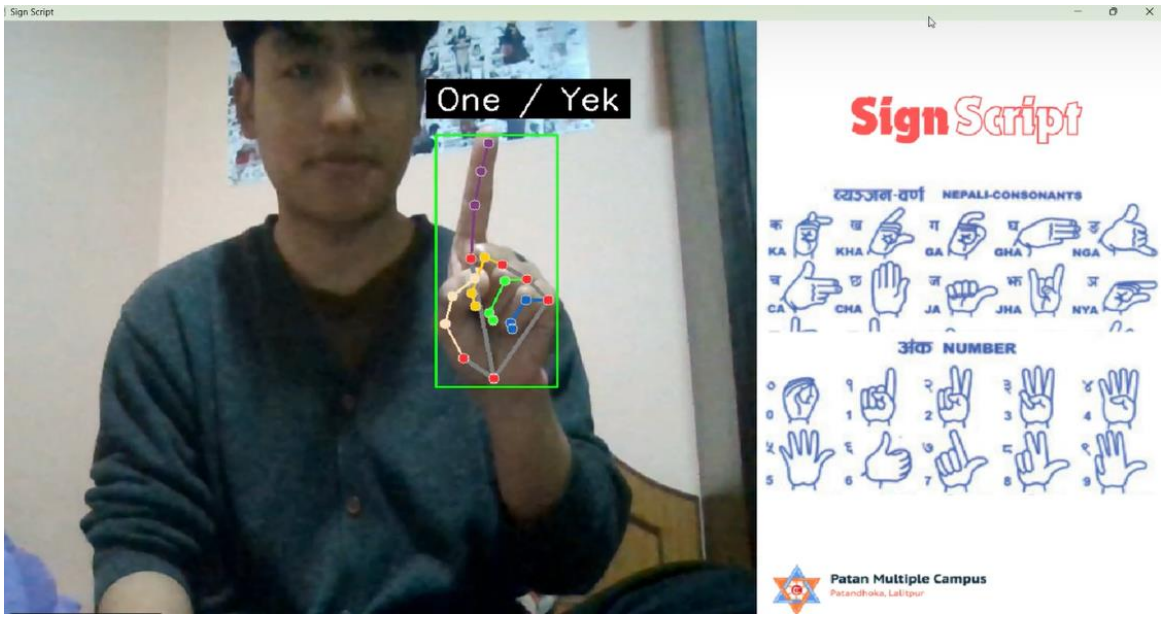
```
def main():
    data, labels = load_data(DATA_PICKLE_PATH)
    model = train_classifier(data, labels)
    save_model(model, MODEL_OUTPUT_PATH)

if __name__ == '__main__':
    main()
```

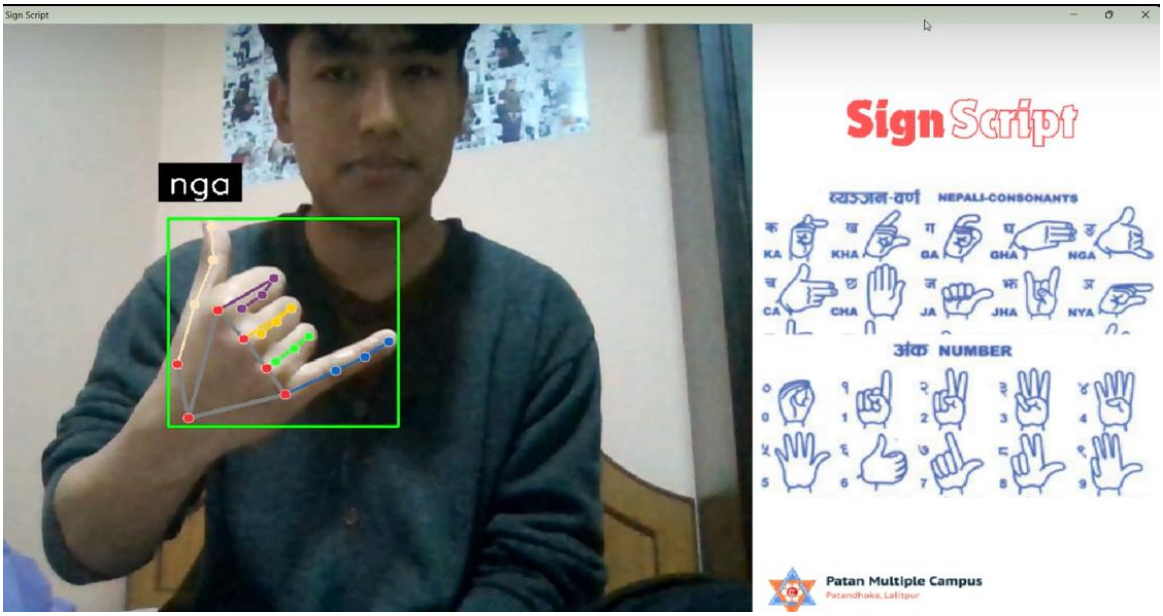
Appendices B: Output Screenshots

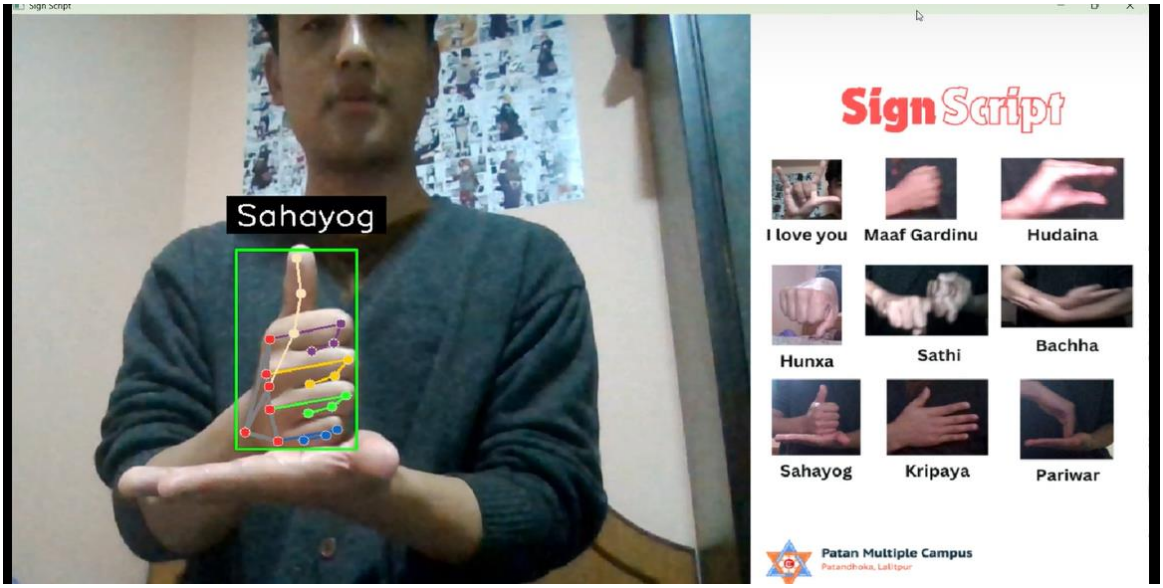


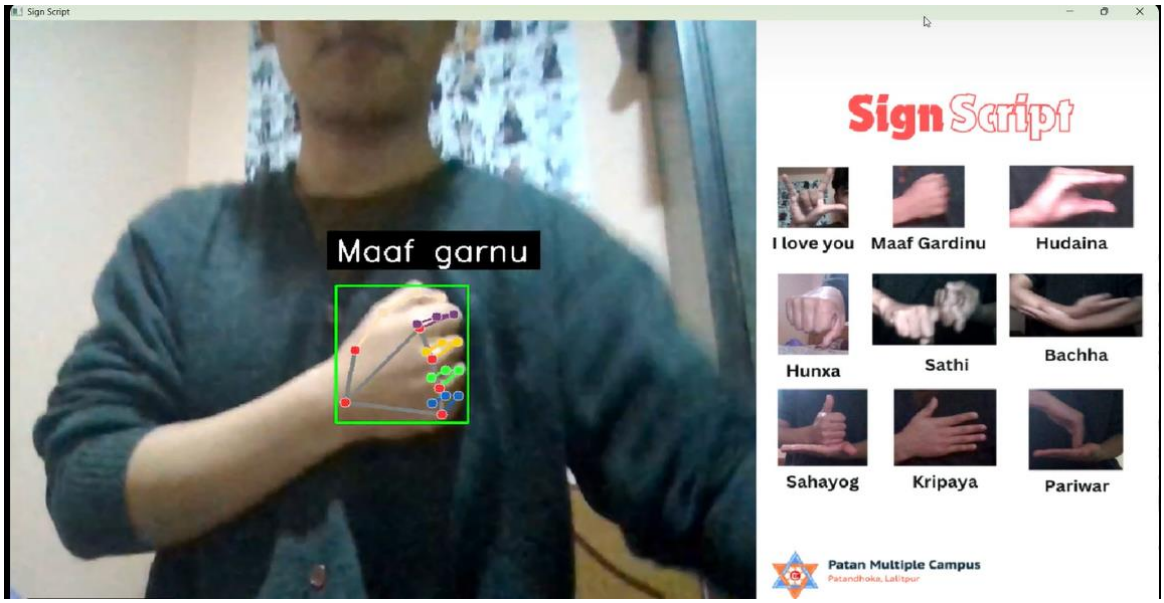


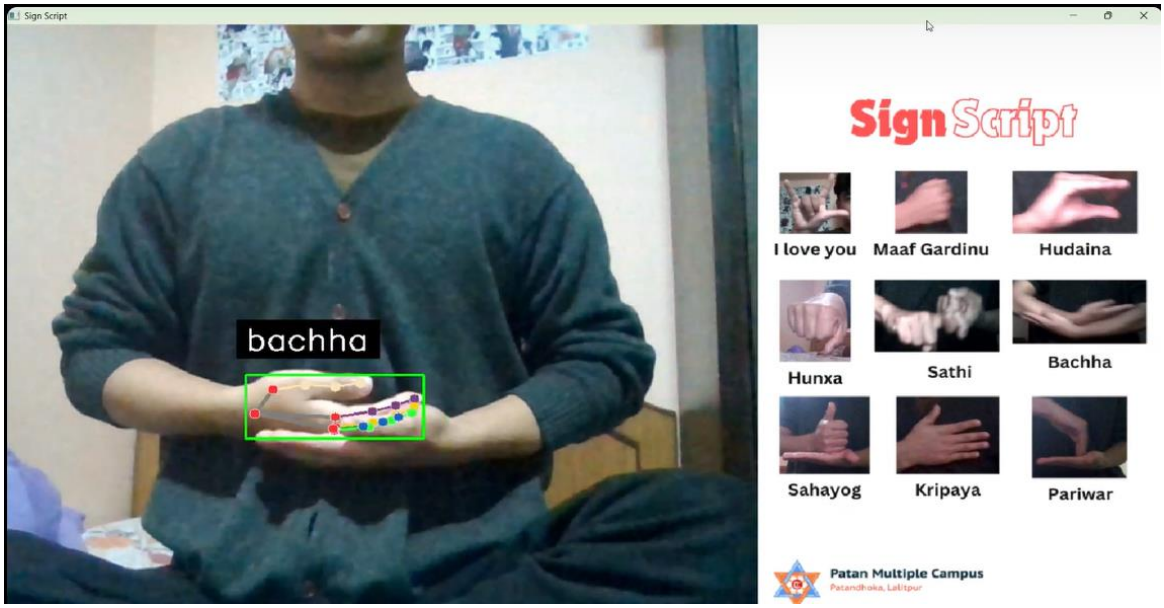

Sign Script
Zero / Sunya
Sign Script
व्यञ्जन-वर्ण NEPALI-CONSONANTS
KA KHA GA GHA NGA
CA CHA JA JHA NYA
अंक NUMBER
Patan Multiple Campus
Patandhoka, Lalitpur

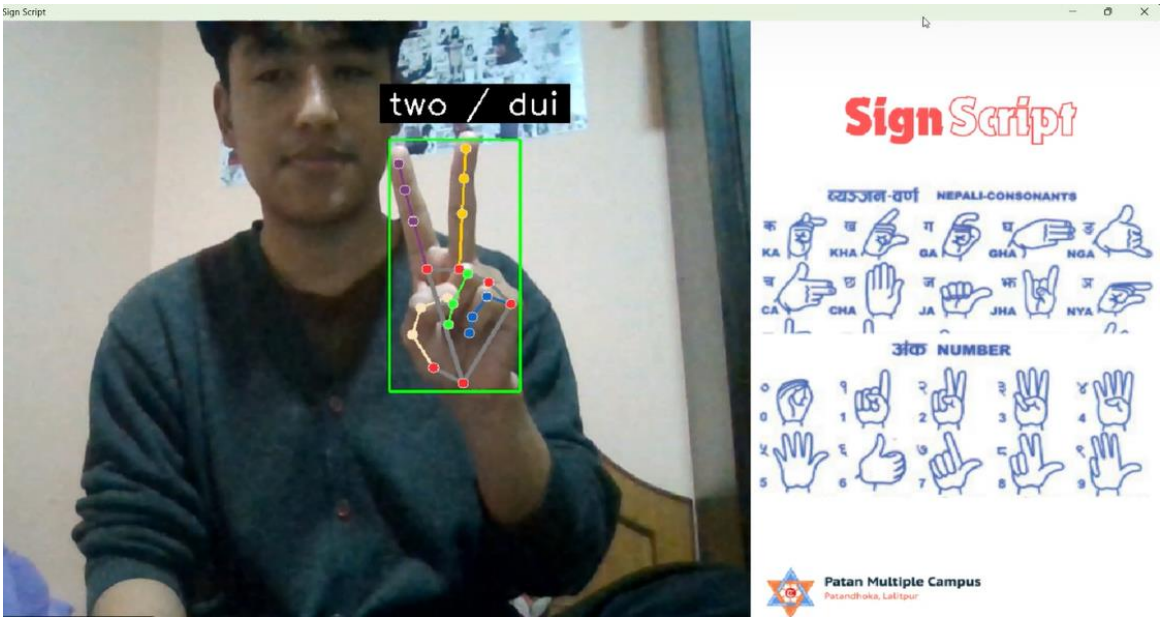

Sign Script
One / Yek
Sign Script
व्यञ्जन-वर्ण NEPALI-CONSONANTS
KA KHA GA GHA NGA
CA CHA JA JHA NYA
अंक NUMBER
Patan Multiple Campus
Patandhoka, Lalitpur

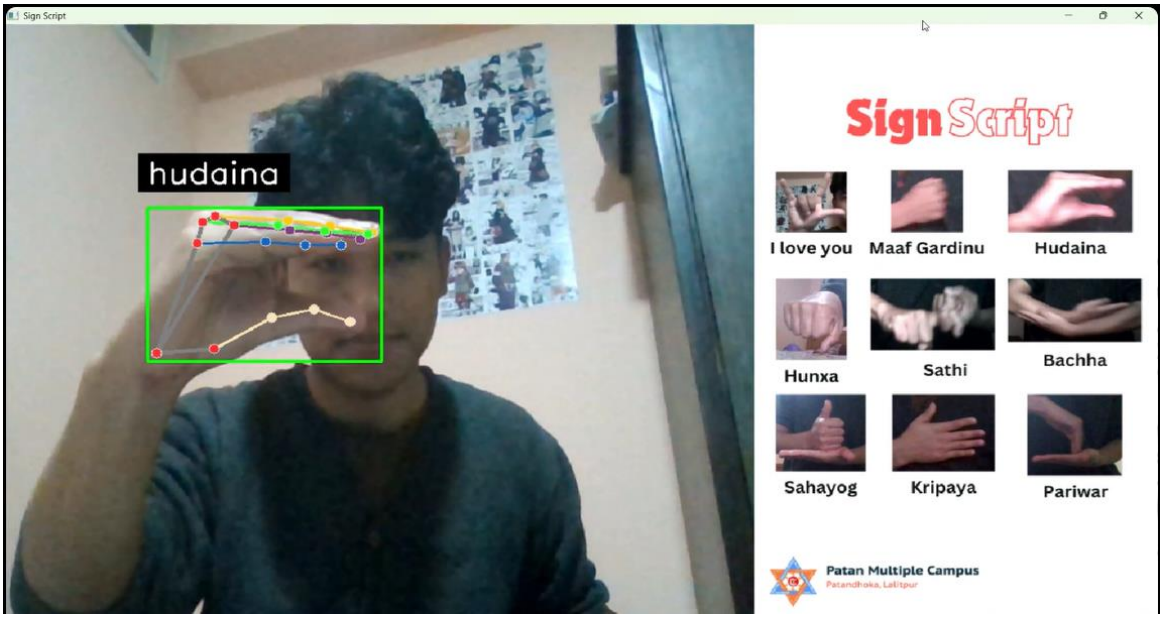

Sign Script
nga
Sign Script
व्यञ्जन-वर्ण NEPALI-CONSONANTS
KA
KHA
GA
GHA
NGA
CA
CHA
JA
JHA
NYA
अंक NUMBER
Patan Multiple Campus
Patandhoka, Lalitpur


Sahayog
Sign Script
I love you
Maaf Gardinu
Hudaina
Hunxa
Sathi
Bachha
Sahayog
Kripaya
Pariwar
Patan Multiple Campus
Patandhoka, Lalitpur


Sign Script
Maaf garnu
Sign Script
I love you
Maaf Gardinu
Hudaina
Hunxa
Sathi
Bachha
Sahayog
Kripaya
Pariwar
Patan Multiple Campus
Patandhoka, Lalitpur


Sign Script
bachha
Sign Script
I love you
Maaf Gardinu
Hudaina
Hunxa
Sathi
Bachha
Sahayog
Kripaya
Pariwar
Patan Multiple Campus
Patandhoka, Lalitpur


Sign Script
two / dui
Sign Script
व्यञ्जन-वर्ण NEPALI-CONSONANTS
KA KHA GA GHA NGA
CA CHA JA JHA NYA
अंक NUMBER
Patan Multiple Campus
Patandhoka, Lalitpur


Sign Script
hudaina
Sign Script
I love you
Maaf Gardinu
Hudaina
Hunxa
Sathi
Bachha
Sahayog
Kripaya
Pariwar
Patan Multiple Campus
Patandhoka, Lalitpur