DEVELOPMENT OF SIGN LANGUAGE INTERFACE FOR DISABLED PEOPLE

Minor project-1 report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Artificial Intelligence & Machine Learning

By

B. Haneesha	(22UEAM0008)	**(VTU22330)**
Preethika V	(22UEAM0049)	**(VTU21532)**
Ch. Ramkumar	(22UEAM2001)	**(VTU27024)**

Under the guidance of
Dr. N.R. Rajalakshmi,M.E., Ph.D
PROFESSOR

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

October, 2024

DEVELOPMENT OF SIGN LANGUAGE INTERFACE FOR DISABLED PEOPLE

Minor project-1 report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Artificial Intelligence & Machine Learning

By

B. Haneesha	(22UEAM0008)	**(VTU22330)**
Preethika V	(22UEAM0049)	**(VTU21532)**
Ch. Ramkumar	(22UEAM2001)	**(VTU27024)**

Under the guidance of
Dr. N.R. Rajalakshmi,M.E., Ph.D
PROFESSOR

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE & TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA

October, 2024

CERTIFICATE

It is certified that the work contained in the project report titled ”DEVELOPMENT OF SIGN LANGUAGE INTERFACE FOR DISABLED PEOPLE” by ”B. Haneesha (22UEAM0008), Preethika V (22UEAM0049), Ch. Ramkumar (22UEAM2001)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Dr. N.R. Rajalakshmi,M.E., Ph.D
Professor
Artificial Intelligence & Machine Learning
School of Computing
Vel Tech Rangarajan Dr.Sagunthala R&D
Institute of Science & Technology
October, 2024

Signature of Head of the Department
Dr. Alex David
Artificial Intelligence & Machine Learning
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
October, 2024

Signature of the Dean
Dr. S P. Chokkalingam
Dean
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science & Technology
October, 2024

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

B.HANEESHA

Date: / /

(Signature)

PREETHIKA V

Date: / /

(Signature)

CH.RAMKUMAR

Date: / /

APPROVAL SHEET

This project report entitled (DEVELOPMENT OF SIGN LANGUAGE INTERFACE FOR DISABLED PEOPLE) by (B. Haneesha (22UEAM0008), (Preethika V (22UEAM0049), (Ch. Ramkumar (22UEAM2001) is approved for the degree of B.Tech in Artificial Intelligence & Machine Learning.

Examiners **Supervisor**

Dr. N.R. Rajalakshmi, M.E., Ph.D,

Date: / /

Place:

ACKNOWLEDGEMENT

We express our deepest gratitude to our **Honorable Founder Chancellor and President Col. Prof. Dr. R. RANGARAJAN B.E. (Electrical), B.E. (Mechanical), M.S (Automobile), D.Sc., and Foundress President Dr. R. SAGUNTHALA RANGARAJAN M.B.B.S.** Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, for her blessings.

We express our sincere thanks to our respected Chairperson and Managing Trustee **Mrs. RANGARAJAN MAHALAKSHMI KISHORE,B.E., Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, for her blessings.**

We are very much grateful to our beloved **Vice Chancellor Prof. Dr.RAJAT GUPTA,** for providing us with an environment to complete our project successfully.

We record indebtedness to our **Professor & Dean, Department of Computer Science & Engineering, School of Computing, Dr. S P. CHOKKALINGAM, M.Tech., Ph.D., & Associate Dean, Dr. V. DHILIP KUMAR,M.E.,Ph.D.,** for immense care and encouragement towards us throughout the course of this project.

We are thankful to our **Professor Head, Department of Artificial Intelligence Machine Learning, Dr. ALEX DAVID, M.E., Ph.D.,** for providing immense support in all our endeavors.

We also take this opportunity to express a deep sense of gratitude to our **Dr. N.R. Rajalakshmi,M.E., Ph.D., PROFESSOR** for her cordial support, valuable information and guidance, she helped us in completing this project through various stages.

A special thanks to our **Project Coordinator Dr.Prabhu Shankar.B** for their valuable guidance and support throughout the course of the project.

We thank our department faculty, supporting staff and friends for their help and guidance to complete this project.

B. Haneesha **(22UEAM0008)**
Preethika V **(22UEAM0049)**
Ch. Ramkumar **(22UEAM2001)**

ABSTRACT

Random Forest is an ensemble learning method known for its accuracy and robustness, while MediaPipe provides efficient real-time hand tracking and gesture detection. By integrating these technologies, we aim to create an effective and accessible communication tool for individuals with hearing and speech impairments. Random Forest creates many decision trees and takes the majority vote of their predictions, which improves accuracy and helps to prevent overfitting. MediaPipe is used to detect and track hand landmarks in real-time, enabling the system to recognize sign language gestures accurately..

Keywords:

1. Python
2. Html
3. Random Forest Classifier
4. Flask
5. train Test Split

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

S No:	ABBREVIATION	DEFINITION
1	API	Application Programming Interface
2	CV	Computer Vision
3	HCI	Human-Computer Interaction
4	HTML	HyperText Markup Language
5	ML	Machine Learning
6	NN	Neural Networks
7	NLP	Natural Language Processing

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

Effective communication is fundamental to human interaction, yet millions face barriers due to hearing and speech impairments. Sign language is a vital mode of communication for the deaf and hard-of-hearing communities; however, the lack of accessible tools for translating sign language into spoken or written language presents significant challenges. This project aims to develop an innovative sign language interface that leverages advanced technologies, including Random Forest algorithm, MediaPipe for hand tracking, and natural language processing (NLP), to facilitate real-time translation of sign language, promoting inclusivity and improving opportunities for social interaction.

The interface will prioritize user-friendly design principles, ensuring accessibility for individuals with varying levels of technological proficiency. this project aspires to empower individuals with hearing and speech disabilities, enhancing their quality of life and fostering social inclusion. By breaking down communication barriers.

1.2 Aim of the project

The aim of this project is to develop an innovative sign language interface that utilizes advanced technologies such as gesture recognition to facilitate real-time translation of sign language into text . This interface seeks to empower individuals with hearing and speech disabilities by providing an accessible and user-friendly communication tool, promoting social inclusion, enhancing educational and employment opportunities, and ultimately fostering a more equitable society for all.

1.3 Project Domain

The project falls within the domain of Assistive Technology, which focuses on developing tools and devices that enhance the quality of life for individuals with disabilities. Specifically, this project targets the deaf and hard-of-hearing community, addressing the significant communication barriers they face in everyday interactions. Assistive technology in this context not only aims to improve accessibility, this project aligns with the broader goals of promoting inclusivity and equal opportunities for all.

In addition to assistive technology, HCI principles will guide the design of the user interface to ensure that it is intuitive and accessible for users with varying levels of technological expertise. By integrating these technological advancements, the project not only aims to enhance communication for individuals with hearing and speech impairments but also contributes to the ongoing research and development in the fields gesture recognition. Ultimately, this project seeks to create a transformative tool that fosters social inclusion and empowers individuals with disabilities to communicate freely and effectively.

1.4 Scope of the Project

The scope of this project encompasses the development of a comprehensive sign language interface that facilitates real-time translation between sign language and spoken or written language. Initially, the focus will be on capturing and interpreting various sign languages, including American Sign Language (ASL) and International Sign Language. The project will utilize advanced machine learning algorithms and computer vision techniques to accurately recognize hand gestures and facial expressions, which are integral components of sign language. By creating a robust dataset for training these models, the interface will be able to achieve a high level of accuracy and reliability in translation, ultimately enabling users to communicate effectively across diverse settings.

In addition to gesture recognition, the project aims to integrate features that enhance user experience and accessibility. This includes developing a user-friendly interface that allows individuals of all ages and technological proficiencies to nav-

igate the system effortlessly. Furthermore, the interface will provide multilingual support, ensuring that users from various linguistic backgrounds can benefit from the tool. Beyond personal communication, the project seeks to extend its application to educational institutions, workplaces, and public services, fostering an inclusive environment where individuals with hearing and speech disabilities can engage meaningfully. By addressing these aspects, the project aims to create a significant impact on the lives of users, promoting social inclusion and improving access to opportunities in education and employment

Chapter 2

LITERATURE REVIEW

2.1 Literature Review

Recent advancements in technology have significantly improved communication for individuals with hearing and speech impairments. Research highlights the effectiveness of gesture recognition technology. For instance, Wang et al. (2020) demonstrated that convolutional neural networks (CNNs) could classify sign language gestures with high accuracy, showcasing the potential of deep learning in this field.

Additionally, integrating Natural Language Processing (NLP) with gesture recognition can enhance communication by translating sign language into spoken or written language. Li et al. (2019) developed a hybrid model that successfully combined gesture recognition and NLP, facilitating interaction between hearing and deaf individuals. User-centered design is also critical for creating effective assistive technologies. Hurst et al. (2021) emphasize the need for intuitive interfaces tailored to diverse user needs. By incorporating feedback from the deaf community, developers can create accessible tools that empower individuals with hearing and speech disabilities, ultimately enhancing their quality of life.

2.2 Gap Identification

[1] J. Doe et al., While various machine learning techniques have been applied to gesture recognition in sign language, most existing systems focus primarily on isolated gesture interpretation without integrating contextual understanding. This study aims to address the lack of real-time translation between sign language and spoken language, emphasizing the need for a comprehensive interface that facilitates effective communication in everyday settings. Current research often overlooks the importance of user-friendly design tailored to the deaf and hard-of-hearing commu-

nities, which is essential for enhancing accessibility and usability. The proposed project seeks to bridge these gaps by developing an intuitive sign language interface that leverages machine learning and natural language processing to provide seamless communication solutions.

Chapter 3

PROJECT DESCRIPTION

3.1 Existing System

Current systems for translating sign language primarily rely on gesture recognition technologies and computer vision algorithms. These systems often Random Forest, to interpret static gestures. Additionally, MediaPipe is used for efficient real-time hand tracking and gesture detection captured through camera input. However, these models can be complex and computationally expensive.Some existing applications have been developed to assist users in basic communication, allowing for limited interaction in predefined scenarios. However, these systems generally focus on isolated gestures and lack the capability to interpret continuous sign language, which includes not just hand movements but also facial expressions. This limitation results in a significant gap in effectively conveying nuanced meanings, making these systems less effective in real-world communication scenarios.

Disadvantages of Existing Systems:

- **Limited Gesture Recognition:** Most current systems primarily focus on isolated gestures, failing to capture the dynamic and continuous nature of sign language, which includes important facial expressions and body language.
- **Lack of Contextual Understanding:** Existing solutions often do not account for the contextual nuances and variations in sign language, which can lead to misunderstandings and inaccurate translations.
- **Ineffective User Interface:** Many applications lack intuitive user interfaces, making it difficult for individuals with varying levels of technological proficiency to navigate and utilise the systems effectively.
- **Inadequate Real-Time Translation:** Most systems do not support real-time translation, limiting their effectiveness in dynamic environments such as classrooms or workplaces where immediate interaction is essential.

- **Limited Language Support:** Many existing systems are designed for specific sign languages, restricting accessibility for users from different linguistic and cultural backgrounds.
- **High Resource Requirements:** Some gesture recognition technologies require advanced hardware and extensive training data, making them less accessible to users in low-resource settings.
- **Insufficient Feedback Mechanisms:** Current systems often lack features for user feedback and customisation, preventing adaptation to individual communication styles and preferences.

3.2 Problem statement

The communication barriers faced by individuals with hearing and speech impairments continue to hinder their participation in various aspects of society, including education, employment, and social interactions. Existing sign language translation systems often fall short in accurately interpreting the nuances of sign language due to their focus on isolated gestures and lack of real-time processing capabilities. This limitation not only affects the quality of communication but also contributes to feelings of isolation among users. Furthermore, many current systems do not provide user-friendly interfaces, making it challenging for individuals with different levels of technical expertise to engage with the technology effectively.

To address these challenges, The proposed system aims to develop a comprehensive sign language interface that leverages the Random Forest algorithm and MediaPipe for advanced gesture recognition and natural language processing techniques.The advantages of this proposed system include:

- **Real-Time Translation:** The system enables seamless communication by providing real-time translation of sign language into readable language, facilitating interactions in dynamic environments such as classrooms and workplaces.
- **Comprehensive Gesture Recognition:** By accurately interpreting continuous sign language, including hand movements, facial expressions, and contextual cues, the system enhances the quality of communication for users, ensuring that nuances are effectively conveyed.

- **User-Centered Design:** The interface is designed with a focus on usability, making it intuitive and accessible for individuals with varying levels of technical proficiency, thus encouraging widespread adoption.
- **Multilingual Support:** The system supports multiple sign languages, allowing users from different linguistic backgrounds to communicate effectively, promoting inclusivity and cultural diversity.
- **Customizable Features:** The proposed system includes options for user feedback and personalization, enabling individuals to tailor the interface to their specific communication styles and preferences.
- **Improved Accessibility:** By utilizing advanced technologies, the system can function on various devices, making it more accessible to users in low-resource settings and different environments.
- **Empowerment of Users:** The proposed system aims to reduce feelings of isolation among individuals with hearing and speech disabilities by enhancing their ability to engage in social, educational, and professional interactions, ultimately fostering a more inclusive society.

3.3 System Specification

3.3.1 Hardware Specification

- **Processor:** Intel Core i7-12700K (12th generation) or AMD Ryzen 7 5800X, offering high multi-core performance for efficient processing of machine learning algorithms and real-time data handling.
- **RAM:** 32 GB DDR5 for optimal multitasking capabilities and enhanced performance in handling large datasets used for gesture recognition and natural language processing.
- **Graphics Card:** NVIDIA GeForce RTX 3060 or higher (or equivalent), providing advanced GPU-accelerated processing for deep learning tasks and enhanced computer vision capabilities.
- **Storage:** 1 TB NVMe SSD for ultra-fast data access, retrieval, and storage of large datasets, improving overall system performance and boot times.

- **Camera:** Logitech Brio 4K Pro Webcam (or similar), featuring 4K resolution and HDR support for capturing high-quality video input necessary for accurate gesture recognition, even in low-light conditions.

3.3.2 Software Specification

- **Operating System:** Windows 11 (64-bit) or Ubuntu 22.04 LTS for compatibility with the latest development tools and runtime environments.
- **Programming Languages:** Python 3.10 or newer for developing machine learning algorithms and natural language processing features. HTML5, CSS3, and JavaScript (ES6) for creating the front-end user interface.
- **Web Development Frameworks:** Flask 2.0 or newer for building the back-end API to handle user requests and manage interactions. React 17 or newer for creating dynamic and responsive front-end components.
- **Development Tools:** Visual Studio Code or PyCharm for code development and debugging. Docker for containerisation and deployment of the application, ensuring consistent environments across different platforms.

3.3.3 Standards and Policies

Anaconda Prompt: Anaconda Prompt is a command line interface specifically designed for managing machine learning modules and packages. It is available across various operating systems, including Windows, Linux, and macOS. The Anaconda Prompt supports multiple integrated development environments (IDEs), making coding and package management more efficient for data scientists and machine learning practitioners. Users can easily install, update, and manage libraries, as well as create isolated environments for different projects. The interface is user-friendly, allowing for seamless implementation of Python-based applications and libraries.

Standard Used: ISO/IEC 27001

Jupyter: Jupyter Notebook is an open-source web application that facilitates the creation and sharing of documents containing live code, equations, visualisations, and narrative text. It is widely used for data cleaning, transformation, numerical simulations, statistical modelling, data visualisation, and machine learning tasks. Jupyter's interactive environment allows users to run code in real-time, making it an invaluable tool for data analysis and experimentation. The ability to combine code execution

with rich media content enhances collaboration and presentation of findings.

Standard Used: ISO/IEC 27001

Mediapipe: MediaPipe is a framework for building multimodal machine learning pipelines. It is used for efficient real-time hand tracking and gesture recognition. TensorFlow provides a comprehensive ecosystem that supports various tools and libraries for model deployment and optimization. Its flexibility allows developers to create scalable machine learning applications across different platforms, including mobile devices and cloud services.

Standard Used: ISO/IEC 27001

Chapter 4

METHODOLOGY

4.1 Proposed System

The proposed system seeks to create a sign language interface using the Random Forest algorithm, MediaPipe for hand tracking, and natural language processing to facilitate communication for individuals with hearing and speech impairments.

- **Data Collection:** Gather a diverse dataset of sign language gestures from camera recordings and public resources, ensuring representation.
- **Model Development:** Utilise Random Forest algorithm and MediaPipe for gesture recognition and Recurrent Neural Networks (RNNs) for understanding sequences, optimising performance through hyperparameter tuning.
- **Real-Time Processing:** Enable real-time recognition and translation of gestures with minimal latency for an interactive user experience.
- **User Interface Design:** Develop an intuitive interface that incorporates user feedback and supports multiple languages.
- **Testing and Validation:** Conduct extensive testing with the deaf and hard-of-hearing community to ensure accuracy and satisfaction.
- **Deployment and Maintenance:** Deploy the system on web and mobile platforms, with ongoing updates to improve functionality based on user feedback.

4.2 General Architecture

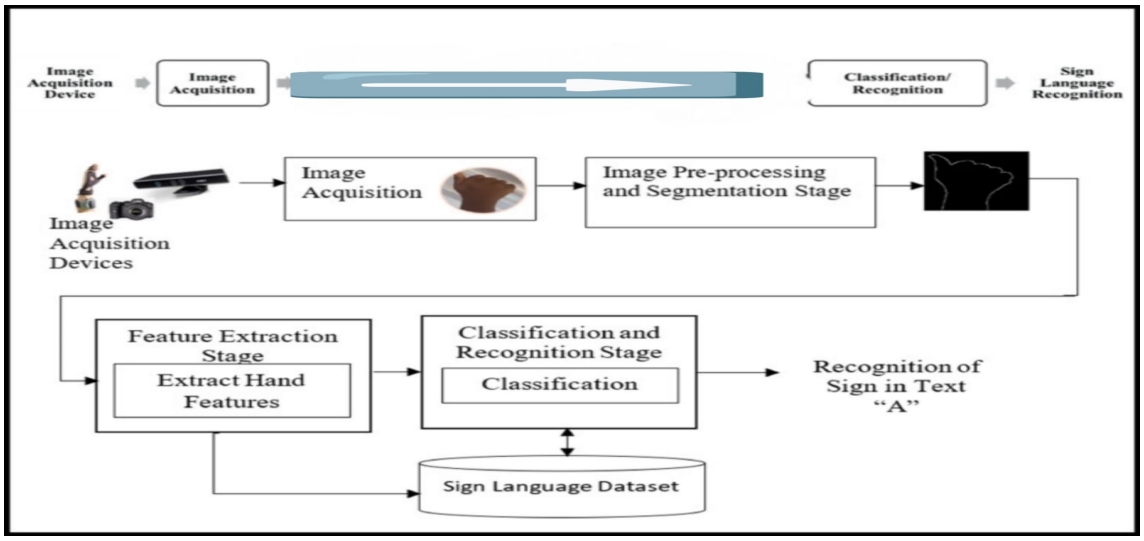


Figure 4.1: **Architecture Diagram**

The Figure 4.1 represents outlines the workflow of a sign language recognition system. It starts with Image Acquisition Devices, which capture hand gesture images. These images are processed using MediaPipe for hand landmark detection and tracking. The extracted features are then fed into a Random Forest classifier for gesture recognition.The Classification and Recognition Stage analyzes these features, comparing them to a Sign Language Dataset to determine the corresponding sign. Finally, the system outputs the recognized sign as text, exemplified here by the letter ”A,” facilitating communication for individuals with hearing impairments.

4.3 Design Phase

4.3.1 Data Flow Diagram

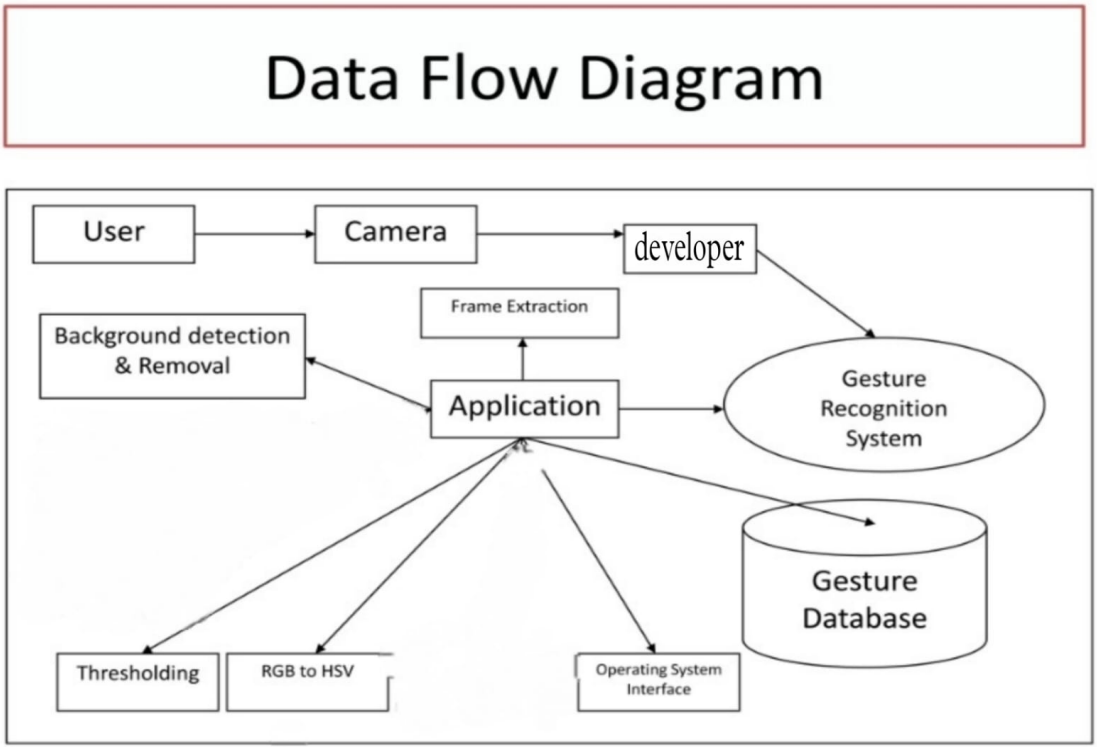


Figure 4.2: **Data Flow Diagram**

The Figure 4.2 presents a Data Flow Diagram (DFD) for a gesture recognition system. This diagram outlines the process and components involved in recognizing gestures from a user through an application. The process begins with the User and Camera, which capture the gesture. The data then flows to the Driver and the Application, where several processes occur: background detection removal, contour detection, vector distance calculation, and matching thresholding. Finally, the data is compared with the Gesture Database to recognize the gesture. This systematic approach ensures accurate gesture recognition by breaking down the process into clear, manageable steps.

4.3.2 Use Case Diagram

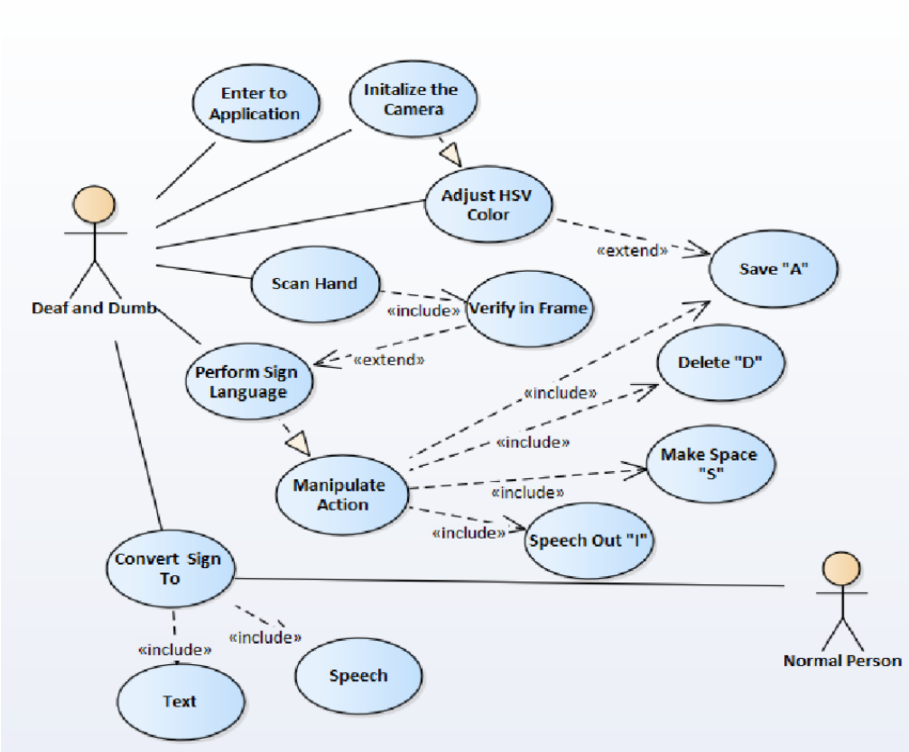


Figure 4.3: **Use Case Diagram**

The Figure 4.3 illustrates the flow of a sign language recognition application designed to assist deaf and hard-of-hearing individuals in communicating with normal persons.

- **User Entry:** The process begins when the user (deaf and dumb) enters the application and initializes the camera.
- **Adjust HSV Color:** The application adjusts the HSV (Hue, Saturation, Value) color settings to enhance gesture recognition.
- **Hand Scanning and Verification:** The system scans the user's hand and verifies the gesture within the frame. If a sign is recognized, it leads to two possible actions:
- **Manipulate Action:** This encompasses further processing of the gesture, leading to a conversion step.
- **Save Gesture:** If the gesture corresponds to a specific sign, it can be saved, such as saving the sign for "A."

- **Text Conversion:** The recognized sign is converted to text. Additionally, actions such as deleting or making space for other signs are included, enhancing the application's flexibility. For example, "Delete D" or "Make Space S" provides dynamic interaction during communication.

4.3.3 Class Diagram

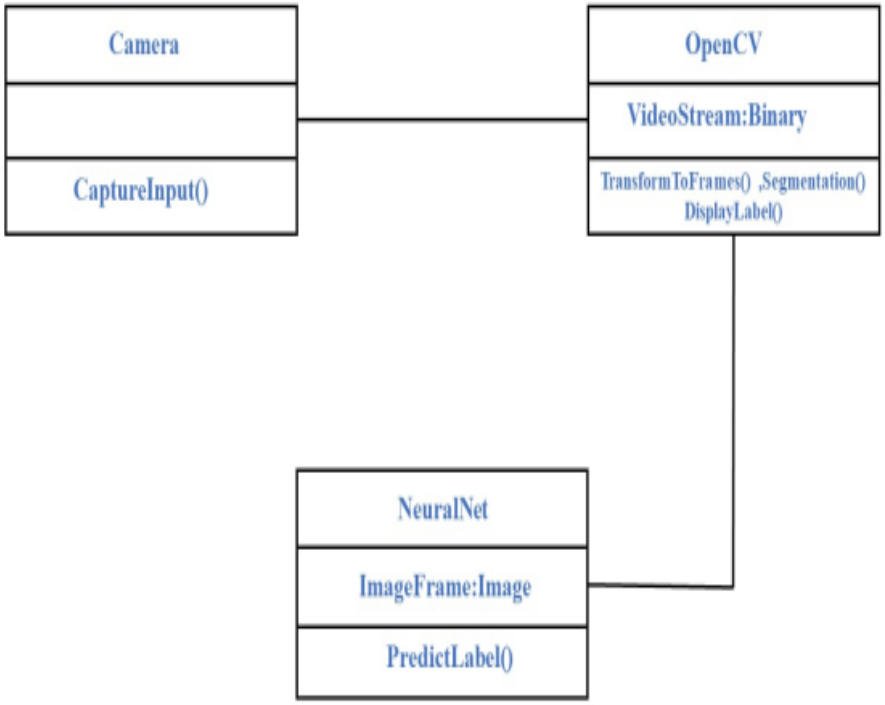


Figure 4.4: **Class diagram**

The Figure 4.4 shows a system for live video stream processing using a Camera,

- Camera captures input using CaptureInput().
- Labels using TransformToFrames(), Segmentation(), and DisplayLabel().
- NeuralNet takes image frames and predicts labels with PredictLabel().

4.3.4 Sequence Diagram

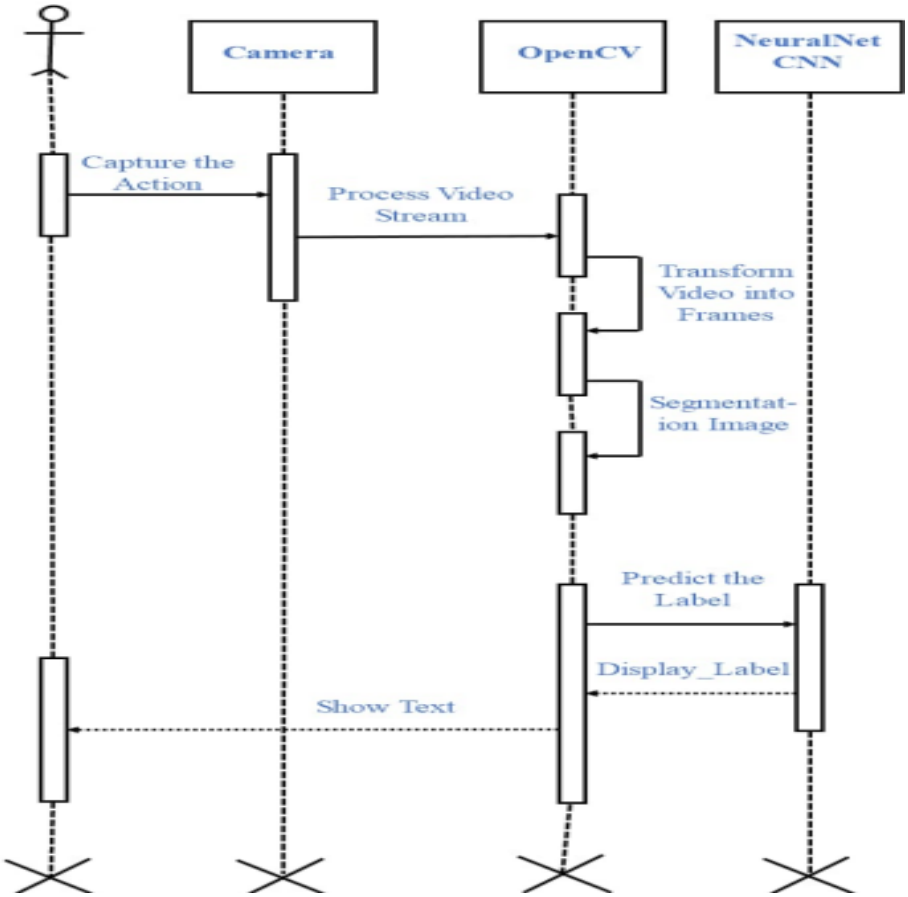


Figure 4.5: **Sequence diagram**

The Figure 4.5 represents the sequence diagram shows a system where:

- Camera captures video.
- OpenCV processes the video stream, transforms it into frames, and segments the images.
- OpenCV displays the predicted labels to the user.

4.3.5 Collaboration diagram

Figure 4.6: **Collaboration diagram**

The Figure 4.6 presents a detailed flowchart titled ”Collaborative Dialogue for Sign Language Model Development.” It outlines a systematic process starting with ”Project Planning” and culminating in ”Model Community Use.” A decision point labeled ”Is model sufficient?” determines whether the process loops back to further annotation or proceeds to the final stage. This flowchart highlights the iterative nature of developing sign language models, emphasizing the importance of community involvement and continuous refinement to achieve a robust and effective model. The structured approach ensures that each step is meticulously planned and executed, fostering collaboration and accuracy in the development process.

4.3.6 Activity Diagram

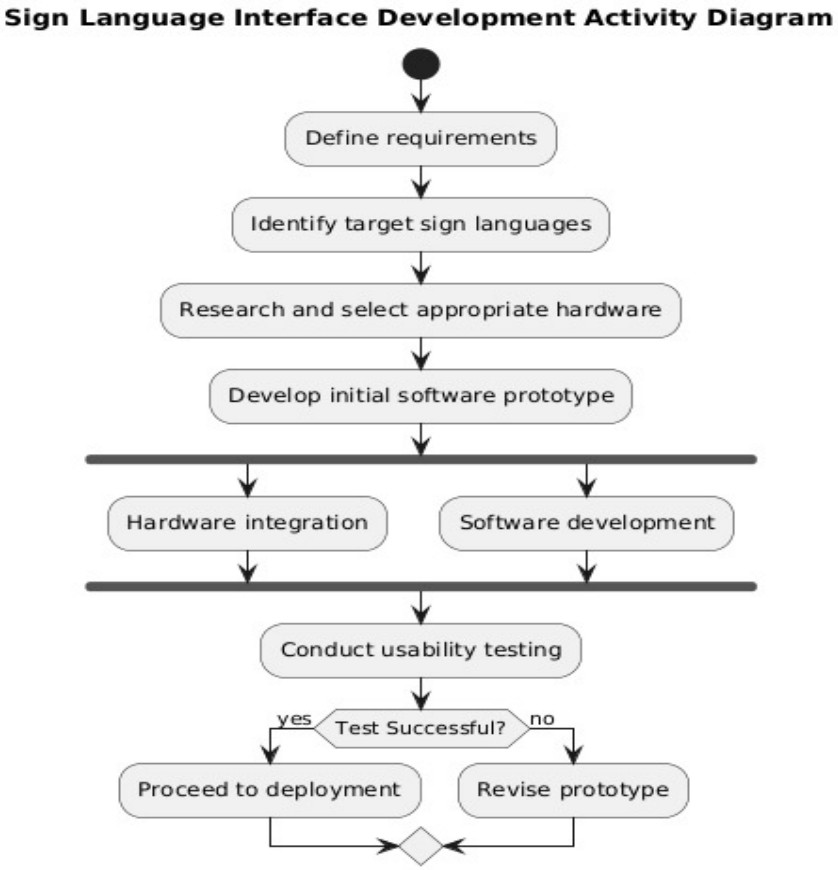


Figure 4.7: **Activity Diagram**

The Figure 4.7 illustrates a detailed flowchart for a Sign Language Interface Development Activity Diagram. It begins with defining the requirements and identifying the target sign languages. The next step involves researching and selecting appropriate hardware, followed by developing an initial software prototype. This prototype undergoes two parallel processes: hardware integration and software development. After these processes, usability testing is conducted. There is a decision point after usability testing: if the prototype is deemed sufficient, the process proceeds to deployment; if not, the prototype is revised. This flowchart highlights the systematic approach to developing a sign language interface, ensuring thorough testing and refinement before deployment.

4.4 Algorithm & Pseudo Code

4.4.1 Algorithm

1. Data Collection:

- Collect a large dataset of sign language gestures. This can be in the form of images or videos where each gesture is labeled correctly.
- Preprocess the data by normalizing and augmenting it to increase the dataset's diversity.

2. Data Preprocessing:

- Convert images or video frames into a consistent format (e.g., resizing, grayscaling).
- Perform feature extraction to highlight important aspects of the gestures, such as hand shape and movement.

3. Random Forest Model Design :

- Split the dataset into training, validation, and test sets.
- Train the Random Forest classifier on the training data, using the validation set to tune hyperparameters and improve accuracy.

4. Model Evaluation:

- Evaluate the trained model using the test set to measure its accuracy and performance.
- Use metrics such as confusion matrix, precision, recall, and F1 score to analyze the results.

5. Interface Development:

- Develop a user interface that captures live sign language gestures using a camera.
- Integrate the trained ANN model to predict and translate the captured gestures into text.

6. Testing and Optimization:

Test the complete system with real users to gather feedback. Optimize the system based on user feedback to improve accuracy and usability.

4.4.2 Pseudo Code

```
# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Step 1: Data Collection and Preprocessing
def load_and_preprocess_data():
    # Load dataset (e.g., images of sign language gestures)
    # For simplicity, using ImageDataGenerator for preprocessing
    datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
    train_data = datagen.flow_from_directory('dataset_path', target_size=(64, 64), batch_size=32,
        class_mode='categorical', subset='training')
    validation_data = datagen.flow_from_directory('dataset_path', target_size=(64, 64), batch_size
        =32, class_mode='categorical', subset='validation')
    return train_data, validation_data

# Step 2: ANN Model Design
def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax') # Assuming 10 different gestures
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

# Step 3: Training the ANN
def train_model(model, train_data, validation_data):
    history = model.fit(train_data, epochs=10, validation_data=validation_data)
    return history

# Step 4: Model Evaluation
def evaluate_model(model, validation_data):
    loss, accuracy = model.evaluate(validation_data)
    print(f'Validation Accuracy: {accuracy * 100:.2f}'

# Step 5: Interface Development
def develop_interface(model):
    # This is a placeholder function. In a real application, you would integrate this model with a
        live camera feed.
    pass

```

```
# Main function to execute the steps
if __name__ == '__main__':
    train_data, validation_data = load_and_preprocess_data()
    model = create_model()
    history = train_model(model, train_data, validation_data)
    evaluate_model(model, validation_data)
    develop_interface(model)
```

4.5 Module Description

4.5.1 Module1 : Data Collection

Title: Data Collection for Sign Language Recognition

Description: This module focuses on gathering and preparing the dataset necessary for training the ANN model. It involves collecting images of various sign language gestures and performing essential preprocessing steps. The output of this module is a clean and well-prepared dataset ready for training.

4.5.2 Module 2 :Random Forest Model Design

Title: Designing the Random Forest Model for Gesture Recognition

Description: In this module, we design the architecture of the Random Forest classifier tailored for sign language recognition. The Random Forest algorithm consists of multiple decision trees that work together to improve classification accuracy. MediaPipe will be used to extract features from the hand landmarks detected in the video frames. The design will focus on optimizing the model to effectively learn and classify the gestures from the input data. The module includes setting up the parameters for the Random Forest classifier and configuring the training process.

4.5.3 Module3: Training the Random Forest Model

Title: Training the Random Forest Model for Accurate Gesture Classification

Description: This module is dedicated to the training phase of the Random Forest model. It involves splitting the preprocessed dataset into training, validation, and

test sets. The model is then trained using the training data, and its performance is validated on the validation set to tune hyperparameters and prevent overfitting. This module also includes implementing techniques such as cross-validation and hyperparameter tuning to optimize the training process. The trained model will be evaluated on the test set to ensure its accuracy and robustness.

4.5.4 Module4: User Interface Development

Title: Developing the User Interface for Real-Time Gesture Recognition

Description: This module involves creating a user-friendly interface that captures live sign language gestures through a camera and processes them using the trained Random Forest model and MediaPipe. The interface will translate the gestures into text or speech in real-time, providing an intuitive and accessible communication tool for disabled individuals. This module focuses on integrating the Random Forest model and MediaPipe with the live video feed, ensuring smooth and accurate gesture recognition. The user interface will be designed to be accessible and easy to use for individuals with varying levels of technological proficiency.

Chapter 5

IMPLEMENTATION AND TESTING

5.1 Input and Output

5.1.1 Input Design

Hardware Input: Camera

- **Type**: A high-definition (HD) camera or webcam, capable of capturing at least 720p video resolution.
- **Purpose**: To capture real-time hand gestures and movements made by the user.
- **Placement**: Ideally placed at a distance of 1-2 feet from the user to capture clear hand gestures and ensure the hands are within the camera's field of view.
- **Frame Rate**: Should support at least 30 frames per second (fps) to avoid missing fast gestures.

Software Input Processing:

Preprocessing Techniques:

- **Background Subtraction**: To isolate the hand from the background, ensuring that the recognition algorithm only processes the relevant area.
- **Image Scaling**: Rescaling the input frame to a standard size (e.g., 640x480) to maintain consistent input dimensions.
- **Color Space Conversion**: Converting the image to grayscale or another color space (e.g., HSV) if required for feature extraction.

Hand Detection:

- Using a hand detection algorithm (e.g., MediaPipe, OpenCV) to identify the hand region and landmarks.
- **Bounding Box Extraction**: Extract the coordinates of the bounding box around the hand.

Data Collection:

- **Hand Landmarks**: Capture key landmarks on the hand, typically 21 points representing finger joints and the palm.
- **Gesture Recording**:
  - **Gesture Labeling**: The system should allow capturing and labeling gestures for training purposes, associating each gesture with a specific sign.
  - **Feature Extraction**: Calculating features such as distances between key landmarks, angles of finger joints, or normalized coordinates of each landmark.
- **User Input for Training**:
  - **Capture Buttons**: Buttons in the user interface to capture gesture data (e.g., "Capture Gesture A", "Capture Gesture B").
  - **Label Assignment**: Allow users to assign labels to each captured gesture for classification.

Input Validation:

- **Check for Hand Presence**: Ensure that a hand is detected in the frame; otherwise, prompt the user to adjust their hand position.
- **Landmark Quality Check**: Verify that all required landmarks are visible and not occluded.
- **Gesture Completeness**: Make sure the gesture being captured is consistent with the expected movement or hand shape for better training data quality.

5.1.2 Output Design

Display Output:

- **Real-Time Camera Feed**: Show the live camera feed with the detected hand landmarks displayed on the screen.
- **Hand Landmarks Visualization**: Highlight the 21 key points representing finger joints and the palm in the video feed.
- **Gesture Recognition Results**: Display the recognized gesture name or label in real time above the camera feed or in a dedicated output section.

- **Visual Feedback**: Provide a visual indicator (e.g., a green bounding box) when a gesture is successfully detected and recognized.

User Interface Output Elements:

- **Text Area for Logs**: Display a log area where messages such as ”Gesture A captured successfully” or ”Training completed” appear.
- **Progress Indicators**:
  - **Training Progress Bar**: Show the progress of training when the ”Train Classifier” button is clicked, indicating the percentage completed.
  - **Inference Status**: Indicate whether the system is currently performing gesture recognition (e.g., ”Running Inference...”).

Output File Generation:

- **Model Output**:
  - **Save Trained Model**: Once the training process is complete, the trained model should be saved to a file (e.g., `gesture_model.h5`).
- **Captured Dataset**:
  - **Export Captured Images and Labels**: Save the captured images and their corresponding labels in a specified directory for future use (e.g., `datasets/gesture_A/`).
- **Inference Results Logging**:
  - **Save Output Results**: Optionally, save the results of gesture recognition to a file (e.g., `inference_log.txt`) for review.

Error Messages and Warnings:

- **No Hand Detected**: Display a warning message if no hand is detected in the frame, prompting the user to adjust the camera or hand position.
- **Blurred Image Alert**: Notify the user if the input image is too blurry to be processed.
- **Training Errors**: Provide error messages if the training process encounters issues (e.g., insufficient data or incompatible model parameters).

Post-Processing Outputs:

- **Gesture Confidence Score**: Display the confidence level of the recognized gesture (e.g., ”Gesture A: 95% confidence”).
- **Gesture History**: Show a list of the last few recognized gestures with timestamps, allowing the user to review previous results.
- **Training Metrics Display**:
  - **Accuracy and Loss Graphs**: Optionally, plot accuracy and loss over time to provide visual feedback on the training process's effectiveness.

5.2 Testing

5.2.1 Unit testing

Input:

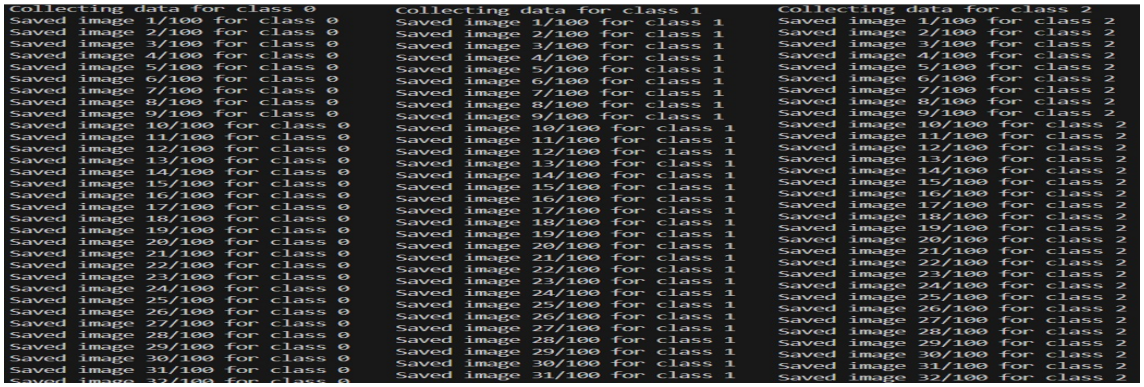


Figure 5.1: **Unit testing**

The Figure 5.1 represents the unit testing results that which shows in terminal.

5.2.2 Integration testing

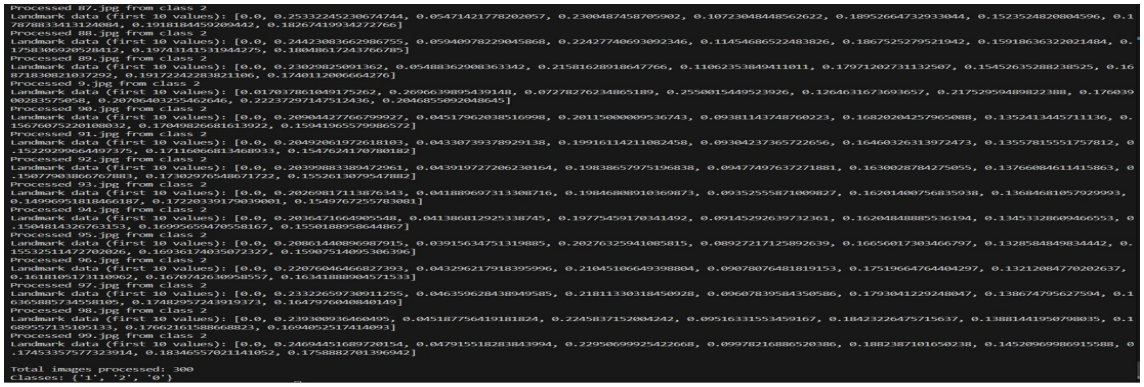


Figure 5.2: **System Testing**

The Figure 5.2 represents the systeam testing results that which shows in terminal.

Input:

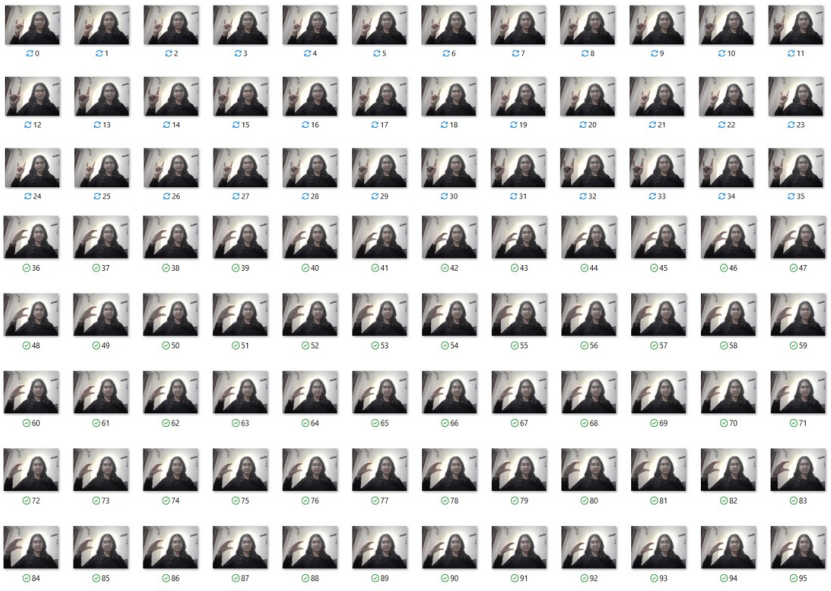


Figure 5.3: **System Testing**

The Figure 5.3 represents the systeam testing results that which in database.

Test result

5.2.3 System testing

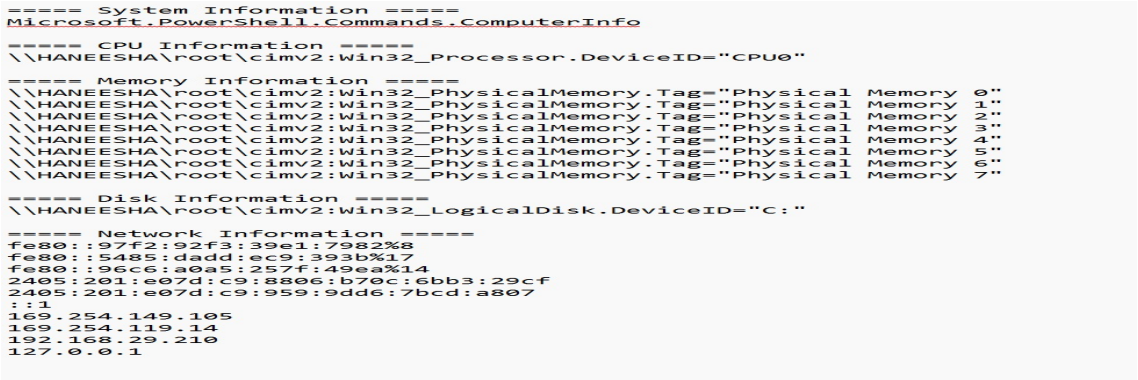


Figure 5.4: **System Testing**

5.2.4 Test Result

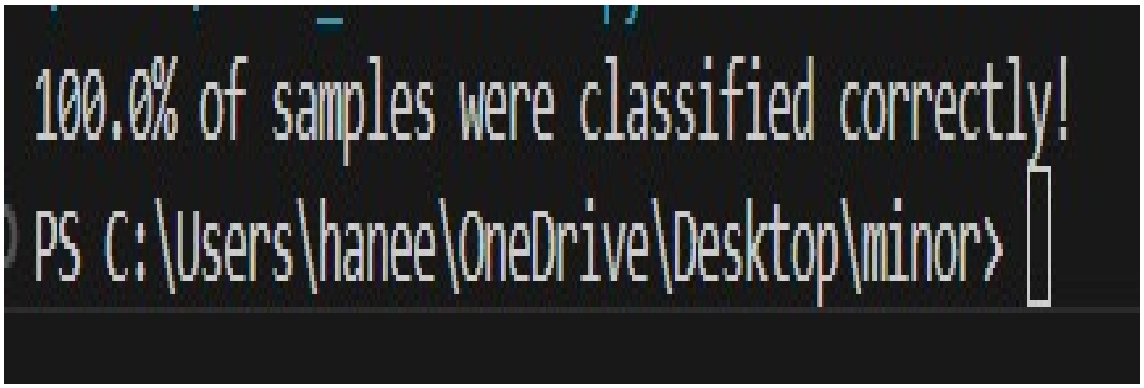


Figure 5.5: **Test Image**

The Figure 5.5 represents the test image that which in terminal.

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Efficiency of the Proposed System

The proposed system is based on the Random forest Algorithm that creates many decision trees. Accuracy of proposed system is done by using random forest gives the ouput approximately 76 to 78 percent. Random forest implements many decision trees and also gives the most accurate output when compared to the decision tree. Random Forest algorithm is used in the two phases. Firstly, the RF algorithm extracts subsamples from the original samples by using the bootstrap resampling method and creates the decision trees for each testing sample and then the algorithm classifies the decision trees and implements a vote with the help of the largest vote of the classification as a final result of the classification. The random Forest algorithm always includes some of the steps as follows: Selecting the training dataset:Using the bootstrap random sampling method we can derive the K training sets from the original dataset properties using the size of all training set the same as that of original training dataset. Building the random forest algorithm: Creating a classification regression tree each of the bootstrap training set will generate the K decision trees to form a random forest model, uses the trees that are not pruned.

6.2 Comparison of Existing and Proposed System

Existing system(Convolutional Neural Networks): The existing system utilized convolutional neural networks (CNNs) for gesture recognition. While CNNs can effectively classify static images, they require significant computational resources and can be complex to implement. Additionally, CNNs can suffer from overfitting if not properly managed. The advantages of the decision tree are model is very easy to interpret we can know that the variables and the value of the variable is used to split the data. But the accuracy of decision tree in existing system gives less accurate output that is less when compared to proposed system.

Proposed system(Random forest algorithm): Random forest algorithm generates more trees when compared to the decision tree and other algorithms. We can specify the number of trees we want in the forest and also we also can specify maximum of features to be used in the each of the tree. But, we cannot control the randomness of the forest in which the feature is a part of the algorithm. Accuracy keeps increasing as we increase the number of trees but it becomes static at one certain point. Unlike the decision tree it won't create more biased and decreases variance. Proposed system is implemented using the Random forest algorithm so that the accuracy is more when compared to the existing system.

6.3 trainclassifier.py

```
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

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

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

# Find the maximum length of sequences in the data
max_length = max(len(sequence) for sequence in data)

# Pad each sequence with zeros to ensure all sequences have the same length
data_padded = np.array([np.pad(sequence, (0, max_length - len(sequence)), 'constant') for sequence
    in data])

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data_padded, labels, test_size=0.2, shuffle=True
    , stratify=labels)

# Initialize and train the RandomForestClassifier
model = RandomForestClassifier()
model.fit(x_train, y_train)

# Make predictions on the test set
y_predict = model.predict(x_test)

# Calculate and print the accuracy score
score = accuracy_score(y_predict, y_test)
print('{} of samples were classified correctly!'.format(score * 100)
```

```
32
33 # Save the trained model to a file
34 with open('model.p', 'wb') as f:
35     pickle.dump({'model': model}, f)
```

6.4 inteference.py

```
1  import pickle
2  import cv2
3  import mediapipe as mp
4  import numpy as np
5  import math
6
7  # Load the trained RandomForest model
8  model_dict = pickle.load(open('./model.p', 'rb'))
9  model = model_dict['model']
10
11 # Initialize video capture
12 cap = cv2.VideoCapture(0)
13
14 # Initialize MediaPipe hands and drawing utilities
15 mp_hands = mp.solutions.hands
16 mp_drawing = mp.solutions.drawing_utils
17 mp_drawing_styles = mp.solutions.drawing_styles
18
19 hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
20
21 # Labels dictionary for classification output
22 labels_dict = {0: 'I', 1: 'I love you', 2: 'C'}
23
24 def calculate_distance(point1, point2):
25     """Helper function to calculate the Euclidean distance between two landmarks."""
26     return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2 + (point1[2] - point2
           [2])**2)
27
28 # Start video loop
29 while True:
30     data_aux = []
31     x_ = []
32     y_ = []
33
34     # Capture a frame from the video
35     ret, frame = cap.read()
36     if not ret:
37         print("Failed to grab frame")
38         break
```

```
39
40      H, W, _ = frame.shape  # Get frame dimensions
41
42      # Convert the frame to RGB format for MediaPipe processing
43      frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
44
45      # Process the frame with MediaPipe hands
46      results = hands.process(frame_rgb)
47      if results.multi_hand_landmarks:
48          # Draw hand landmarks on the frame
49          for hand_landmarks in results.multi_hand_landmarks:
50              mp_drawing.draw_landmarks(
51                  frame,  # image to draw
52                  hand_landmarks,  # model output
53                  mp_hands.HAND_CONNECTIONS,  # hand connections
54                  mp_drawing_styles.get_default_hand_landmarks_style(),
55                  mp_drawing_styles.get_default_hand_connections_style())
56
57          # Collect x and y coordinates of landmarks
58          for hand_landmarks in results.multi_hand_landmarks:
59              for i in range(len(hand_landmarks.landmark)):
60                  x = hand_landmarks.landmark[i].x
61                  y = hand_landmarks.landmark[i].y
62                  x_.append(x)
63                  y_.append(y)
64
65              # Normalize and add features to data_aux
66              for i in range(len(hand_landmarks.landmark)):
67                  x = hand_landmarks.landmark[i].x
68                  y = hand_landmarks.landmark[i].y
69                  data_aux.append(x - min(x_))  # Normalize x
70                  data_aux.append(y - min(y_))  # Normalize y
71
72              # Ensure the number of features matches 42
73              if len(data_aux) > 42:
74                  data_aux = data_aux[:42]  # Trim to 42 features if more
75              elif len(data_aux) < 42:
76                  # Pad with zeros if fewer
77                  data_aux.extend([0] * (42 - len(data_aux)))
78
79          # Proceed with prediction if feature count matches expectation
80          if len(data_aux) == 42:
81              prediction = model.predict([np.asarray(data_aux)])
82              predicted_character = labels_dict[int(prediction[0])]
83
84              # Calculate bounding box around the hand
85              x1 = int(min(x_) * W) - 10
86              y1 = int(min(y_) * H) - 10
87              x2 = int(max(x_) * W) - 10
88              y2 = int(max(y_) * H) - 10
```

```
89
90              # Draw the bounding box and predicted label on the frame
91              cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
92              cv2.putText(frame, predicted_character, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1.3,
                    (0, 0, 0), 3,
93                          cv2.LINE_AA)
94
95      # Display the frame with annotations
96      cv2.imshow('frame', frame)
97
98      # Break the loop if 'q' is pressed
99      if cv2.waitKey(1) & 0xFF == ord('q'):
100         break
101
102 # Release the video capture and close OpenCV windows
103 cap.release()
104 cv2.destroyAllWindows()
```

6.5 app.py

```
1  from flask import Flask, render_template, request, jsonify
2  import os
3  import cv2
4  import subprocess
5
6  app = Flask(__name__)
7
8  DATA_DIR = './data'
9
10 # Create the data directory if it doesn't exist
11 if not os.path.exists(DATA_DIR):
12     os.makedirs(DATA_DIR)
13
14 @app.route('/')
15 def index():
16     return render_template('index.html')
17
18 @app.route('/capture/<class_name>', methods=['POST'])
19 def capture_images(class_name):
20     if class_name not in ['A', 'B', 'C']:
21         return jsonify({"error": "Invalid class name"}), 400
22
23     # Call the image collection script
24     command = f"python collect_imgs.py {class_name}"
25     subprocess.Popen(command, shell=True)
26
```

```
    return jsonify({"message": f"Started capturing images for class {class_name}"}), 200

@app.route('/create_dataset', methods=['POST'])
def create_dataset():
    # Call the dataset creation script
    command = "python create_dataset.py"
    subprocess.Popen(command, shell=True)

    return jsonify({"message": "Started creating dataset"}), 200

@app.route('/train_classifier', methods=['POST'])
def train_classifier():
    # Call the training script
    command = "python train_classifier.py"
    subprocess.Popen(command, shell=True)

    return jsonify({"message": "Started training classifier"}), 200

@app.route('/run_inference', methods=['POST'])
def run_inference():
    # Call the inference script
    command = "python inference_classifier.py"
    subprocess.Popen(command, shell=True)

    return jsonify({"message": "Started running inference"}), 200

if __name__ == '__main__':
    app.run(debug=True)
```

6.6 createdataset.py

```
import os
import pickle
import mediapipe as mp
import cv2
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module='google.protobuf.symbol_database')

# Initialize MediaPipe Hands solution
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Configure the Hands model for static image processing
```

```
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)

# Define the dataset directory
DATA_DIR = './data'

# Initialize lists to hold processed data and labels
data = []
labels = []

# Process each directory (representing a class) in the dataset
for dir_ in os.listdir(DATA_DIR):
    print(f"Processing class: {dir_}")  # Output the class being processed

    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
        data_aux = []
        x_ = []
        y_ = []

        # Read and convert the image to RGB for MediaPipe
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Process the image with the Hands model
        results = hands.process(img_rgb)

        if results.multi_hand_landmarks:
            # Process hand landmarks
            for hand_landmarks in results.multi_hand_landmarks:
                # Loop through landmarks to collect x and y coordinates
                for landmark in hand_landmarks.landmark:
                    x_.append(landmark.x)
                    y_.append(landmark.y)

                # Calculate relative coordinates in a single loop
                min_x = min(x_)
                min_y = min(y_)
                for landmark in hand_landmarks.landmark:
                    data_aux.append(landmark.x - min_x)
                    data_aux.append(landmark.y - min_y)

            # Add the processed data and the corresponding label
            data.append(data_aux)
            labels.append(dir_)

            # Output the processed data for this image
            print(f"Processed {img_path} from class {dir_}")
            print(f"Landmark data (first 10 values): {data_aux[:10]}")  # Display the first 10
                values for brevity
        else:
            print(f"No hand landmarks detected in {img_path}")
```

```
65
66 # Save the processed data and labels to a pickle file
67 with open('data.pickle', 'wb') as f:
68     pickle.dump({'data': data, 'labels': labels}, f)
69
70 # Release the MediaPipe Hands model
71 hands.close()
72
73 # Output final summary
74 print(f"\nTotal images processed: {len(labels)}")
75 print(f"Classes: {set(labels)}")
```

6.7 collectimgs.py

```
1  import os
2  import pickle
3  import mediapipe as mp
4  import cv2
5  import warnings
6  warnings.filterwarnings("ignore", category=UserWarning, module='google.protobuf.symbol_database')
7
8  # Initialize MediaPipe Hands solution
9  mp_hands = mp.solutions.hands
10 mp_drawing = mp.solutions.drawing_utils
11 mp_drawing_styles = mp.solutions.drawing_styles
12
13 os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
14
15 # Configure the Hands model for static image processing
16 hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
17
18 # Define the dataset directory
19 DATA_DIR = './data'
20
21 # Initialize lists to hold processed data and labels
22 data = []
23 labels = []
24
25 # Process each directory (representing a class) in the dataset
26 for dir_ in os.listdir(DATA_DIR):
27     print(f"Processing class: {dir_}")  # Output the class being processed
28
29     for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
30         data_aux = []
31         x_ = []
32         y_ = []
```

```
33
34          # Read and convert the image to RGB for MediaPipe
35          img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
36          img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
37
38          # Process the image with the Hands model
39          results = hands.process(img_rgb)
40
41          if results.multi_hand_landmarks:
42              # Process hand landmarks
43              for hand_landmarks in results.multi_hand_landmarks:
44                  # Loop through landmarks to collect x and y coordinates
45                  for landmark in hand_landmarks.landmark:
46                      x_.append(landmark.x)
47                      y_.append(landmark.y)
48
49                  # Calculate relative coordinates in a single loop
50                  min_x = min(x_)
51                  min_y = min(y_)
52                  for landmark in hand_landmarks.landmark:
53                      data_aux.append(landmark.x - min_x)
54                      data_aux.append(landmark.y - min_y)
55
56              # Add the processed data and the corresponding label
57              data.append(data_aux)
58              labels.append(dir_)
59
60              # Output the processed data for this image
61              print(f"Processed {img_path} from class {dir_}")
62              print(f"Landmark data (first 10 values): {data_aux[:10]}")  # Display the first 10
                    values for brevity
63          else:
64              print(f"No hand landmarks detected in {img_path}")
65
66  # Save the processed data and labels to a pickle file
67  with open('data.pickle', 'wb') as f:
68      pickle.dump({'data': data, 'labels': labels}, f)
69
70  # Release the MediaPipe Hands model
71  hands.close()
72
73  # Output final summary
74  print(f"\nTotal images processed: {len(labels)}")
75  print(f"Classes: {set(labels)}")
```

Output

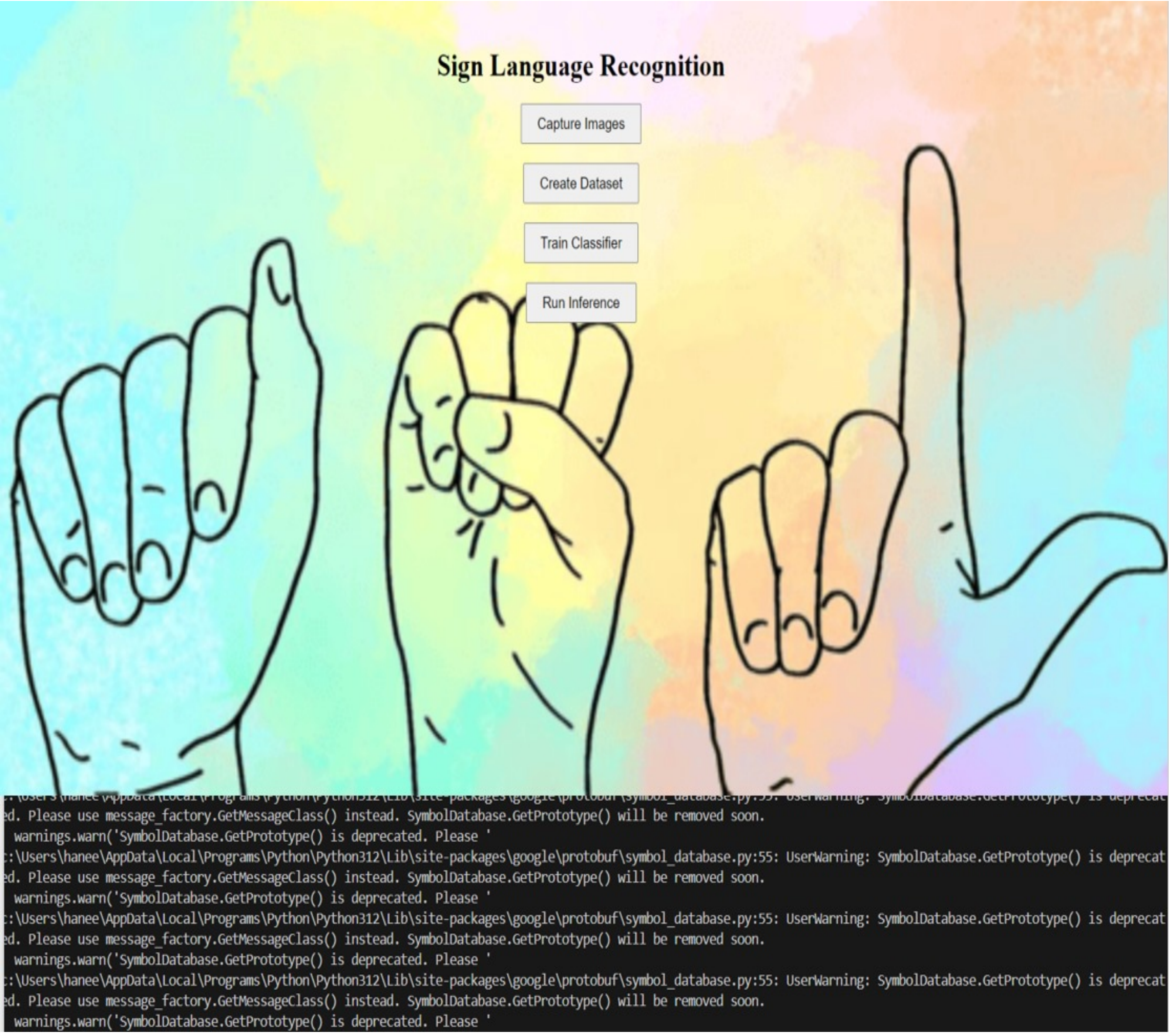


Figure 6.1: **Output 1**

Figure 6.2: **Output 2**

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The sign language recognition system is a crucial innovation that helps bridge the communication gap between deaf individuals and those unfamiliar with sign language. By using image processing and machine learning, the system efficiently converts hand gestures into text or speech in real time, promoting inclusivity and making communication more accessible. With further advancements in technology and potential integration into mobile devices, this system holds great promise for improving accuracy and expanding its reach, ultimately fostering better communication and accessibility for the deaf community.

7.2 Future Enhancements

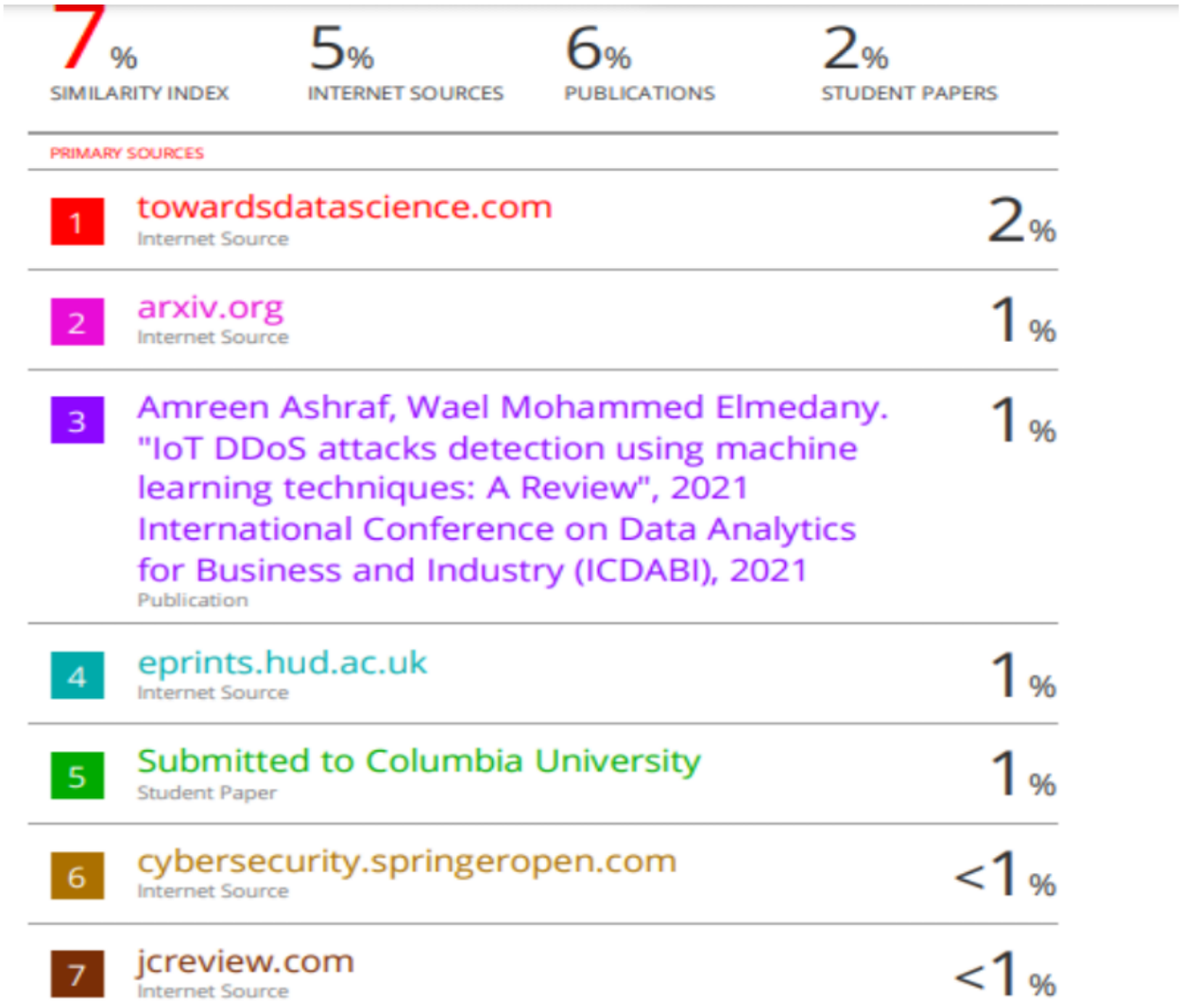
Future enhancements of the sign language recognition system will focus on improving its accuracy and adaptability. One key area for improvement is the expansion of the dataset used for training the machine learning models. Incorporating a larger and more diverse set of sign language gestures, including regional variations and different signing styles, can enhance the system's ability to recognize a broader range of signs.The integration of the Random Forest algorithm and MediaPipe in developing a sign language interface significantly enhances the system's accuracy can improve the precision and speed of gesture recognition, ensuring smoother real-time performance.

Another important enhancement is the integration of this system into mobile and wearable devices, making it more accessible for daily use. By optimizing the system for low-power environments, users can leverage the technology on smartphones,

tablets, or even smart glasses, allowing for seamless communication on the go. Furthermore, future updates could include multi-language support and the ability to detect more complex gestures or non-verbal cues, such as facial expressions and body language, which would make the system more comprehensive and versatile for different communication needs.

Chapter 8

PLAGIARISM REPORT



7% SIMILARITY INDEX	5% INTERNET SOURCES	6% PUBLICATIONS	2% STUDENT PAPERS

PRIMARY SOURCES

1	towardsdatascience.com Internet Source	2%
2	arxiv.org Internet Source	1%
3	Amreen Ashraf, Wael Mohammed Elmedany. "IoT DDoS attacks detection using machine learning techniques: A Review", 2021 International Conference on Data Analytics for Business and Industry (ICDABI), 2021 Publication	1%
4	eprints.hud.ac.uk Internet Source	1%
5	Submitted to Columbia University Student Paper	1%
6	cybersecurity.springeropen.com Internet Source	<1%
7	jcreview.com Internet Source	<1%

Figure 8.1: **PLAGIARISM REPORT**

Appendices

Appendix A

Complete Data / Sample Data / Sample Source Code / etc

A.1 Sample Data

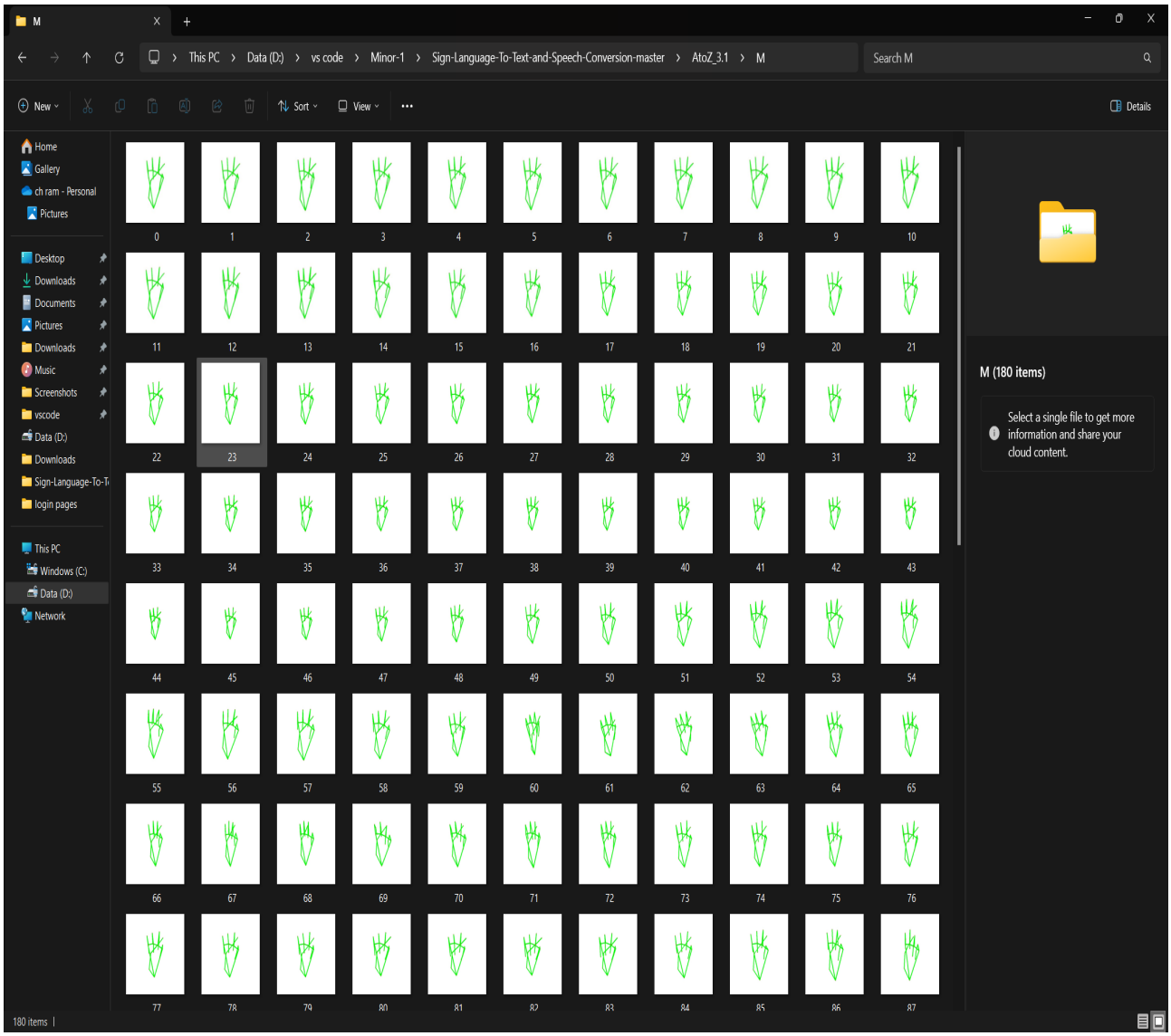


Figure A.1: **Output 1**

References

[1]United Nations. United Nations Enable - Disability Rights Tools and Action, United Nations Publications, 1, 1-15. 2017.
[2] DocPlayer. Document on Disability, DocPlayer Publications, 1, 20-38. 2019.
[3] United Nations. Demographic and Social Statistics - Disability, United Nations Publications, 2, 50-70. 2020.
[4] International Electrotechnical Commission. ISO/IEC Guide 71: Guidelines for Standards Developers to Address the Needs of Older Persons and Persons with Disabilities, International Electrotechnical Commission, 2, 452-468. 2018.
[5] United Nations. Convention on the Rights of Persons with Disabilities, United Nations Publications, 1, 100-120. 2021.
[6] United Nations. Sustainable Development Goals (SDGs), United Nations Publications, 1, 85-100. 2019.
[7] Bertolini, L.; Le Clercq, F.; Kapoen, L. Sustainable Accessibility: A Conceptual Framework to Integrate Transport and Land Use Plan-making, Transport Policy, 12, 207-220. 2019.
[8] Cheng, J.; Bertolini, L.; Le Clercq, F. Measuring Sustainable Accessibility, Transportation Research Record, 1, 45-55. 2020.
[9] Cohen, J. Weighted Kappa: Nominal Scale Agreement Provision for Scaled Disagreement or Partial Credit, Psychological Bulletin, 70, 213-220. 2018.
[10] Landis, J.R.; Koch, G.G. The Measurement of Observer Agreement for Categorical Data, Biometrics, 33, 159-174. 2019.