**GESTURE COMMUNICATON**

**BRIDGING ASL AND TEXT/SPEECH**

**A MINI PROJECT REPORT**

**Submitted by**

|  |  |
| --- | --- |
| **DEEPTHI T V** | **1521041039** |
| **SWETHA N** | **1521041037** |
| **MAHALAKSHMI B** | **1521041018** |

***in partial fulfillment for the award of the degree***

**of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

****

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

**DHANALAKSHMI SRINIVASAN UNIVERSITY**

**TRICHY 621-112**

**BONAFIDE CERTIFICATE**

Certified that this project report for “**GESTURE COMMUNICATON**

**BRIDGING ASL AND TEXT/SPEECH**” is the bonafide work of DEEPTHI T V (1521041039), SWETHA N (1521041037), MAHALAKSHMI B(1521041018) who carried out the project work under my supervision.

**Dr.T SATHIS KUMAR MR.C.RAMACHANDRAN**

**HEAD OF THE DEPARTMENT, GUIDE,**

Department of Computer Science & Department of Computer Science &

Engineering, Engineering,

School of Engineering and Technology, School of Engineering and technology,

Dhanalakshmi Srinivasan university, Dhanalakshmi Srinivasan university,

Trichy – 621-112. Trichy – 621-112.

Submitted for the project Viva-Voice held on  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**TABLE OF CONTENT**

**CHAPTER NO TITLE PAGE NO**

**ABSTRACT** 3

**LIST OF ABBREVATION** 6

**1 INTRODUCTION** 7

1.1 About the Project 8

1.2 Goals of Project 10

1.3 Problem Statement 11

**2 LITERATURE SURVEY** 11

**3 EXISTING SYSTEM**  13

**4 PROPOSED SYSTEM** 14

**5 SYSTEM REQUIREMENTS** 14

5.1 Hardware Requirements 14

5.2 Software Requirements 15

5.3 Datasets 16

**6 SYSTEM DESIGN** 16

6.1 Sign To Voice/Text 16

6.2Voice To Sign 17

**7 SYSTEM IMPLEMENTATION** 18

7.1 List of Modules 18

7.1.1 Data Acquisition 18

7.1.2 Data Pre-processing 18

7.3 Algorithm 22

**8 TESTING** 26

8.1 Unit Testing 26

8.2 Integration Testing 26

8.3 System Testing 26

8.4 Performance Testing 26

8.5 Accuracy Testing 27

8.6 Usability Testing 27

8.7 Robustness Testing 27

8.8 Regression Testing 27

**9 CONCLUSION AND FUTURE ENHANCEMENT** 27

9.1 Conclusion 27

9.2 Future Enhancement 28

9.2.1 Improved Gesture Recognition 28

9.2.2. Reduced Latency 29

9.2.3. Error Handling and User Feedback 29

9.2.4. Expanded Functionality 29

9.2.5. User Experience Enhancements 29

**10 APPENDIX** 29

10.1 Source Code 29

10.2 Screen shorts 44

**11 REFERENCES** 48

**ABSTRACT**

We offer a real-time approach that combines long short-term memory (LSTM) layers, recurrent neural networks (RNNs), and convolutional neural networks (CNNs) to recognize finger typing in American Sign Language (ASL). The technology uses sophisticated computer vision algorithms to process and classify hand movements that are captured by a camera. For hand gesture detection, MediaPipe Holistic Key points are used, and the integrated CNN-RNN-LSTM model improves accuracy and context understanding. By translating ASL into text or speech, this method successfully closes the communication gap that exists between the hearing community and deaf and hard-of-hearing people. These creative solutions are crucial for encouraging inclusive communication, as the number of people with hearing loss is expected to expand internationally. Gesture communication serves as a vital component in bridging the gap between American Sign Language (ASL) and text/speech. This approach enhances accessibility and inclusivity by translating ASL gestures into readable text or audible speech, facilitating seamless interaction between Deaf and hearing individuals. By leveraging advanced technologies such as computer vision, machine learning, and natural language processing, gesture recognition systems can accurately interpret ASL gestures and convert them into text or speech. This integration not only supports real-time communication but also promotes greater understanding and collaboration. The development and refinement of these systems aim to reduce communication barriers, fostering a more inclusive environment where diverse modes of expression are valued and understood.

LIST OF ABBREVATION

**S.NO ABBREVIATION EXPANSION**

1 CONVULUTIONAL NEURAL NETWORK CNN

2 RECURRENT NEURAL NETWORK RNN

3 LONG SHORT TERM MEMORY LSTM

4 REGION OF INTREST RIO

5 K-NEAREST NIGHBOUR KNN

6 AMERICAN SIGN LANGUAGE ASL

1. INTRODUCTION

Human contact requires communication in order for us to exchange ideas and feelings via voice, gestures, actions, and sights. Since traditional spoken languages are not an option for Deaf and Mute (D&M) people, sign language serves as their major means of communication. In the Deaf population in the US and abroad, American Sign Language (ASL) is extensively utilized to facilitate successful communication through hand gestures and visual signals.

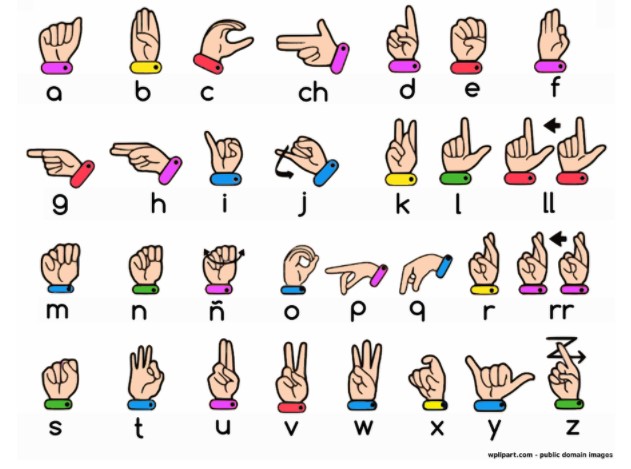
Our project's goal is to create a model that can identify hand movements based on fingerspelling and combine different gestures to form whole phrases. This invention, known as Gesture Comm, enables smooth communication between people utilizing various modalities by translating ASL into text and speech and vice versa.Gesture communication facilitates understanding between disparate populations and fosters inclusivity by bridging communication gaps.

Effective communication between Deaf and hearing individuals is essential for inclusivity and accessibility. American Sign Language (ASL) is a rich, complex language used by the Deaf community to communicate. However, the language barrier between ASL users and those who do not understand sign language often poses challenges. This project aims to bridge this gap by developing a gesture communication system that translates ASL into text and speech in real-time.

Leveraging MediaPipe, an advanced framework for real-time hand tracking and gesture recognition, the project seeks to create a system that accurately captures ASL gestures and converts them into readable text or audible speech. By using computer vision and machine learning technologies, the system will recognize hand shapes, movements, and facial expressions, providing a seamless translation of ASL.

The project involves several stages, including data collection, model training, system integration, and rigorous testing. By collecting a diverse and comprehensive dataset of ASL gestures, training robust machine learning models, and integrating the components into a cohesive system, the project aims to achieve high accuracy and reliability.

This gesture communication system will significantly enhance the interaction between Deaf and hearing individuals, promoting greater understanding and inclusivity. It will serve as a valuable tool in various settings, such as education, customer service, and everyday communication, making ASL more accessible to everyone.

****

**Fig no-(i)**

* 1. About the Project

The Gesture Comm project aims to bridge the communication gap between Deaf and Mute (D&M) individuals and the hearing population by leveraging technology to translate American Sign Language (ASL) gestures into text and speech, and vice versa. This innovative approach facilitates inclusive communication and understanding across different modalities, fostering a more connected and empathetic society.

The project involves the development of a sophisticated gesture recognition model using the MediaPipe library. By focusing on the accurate identification of ASL hand movements and fingerspelling, the system can construct complete phrases and convey complex messages. This real-time translation capability ensures seamless interactions, making communication more natural and efficient.

Gesture Communication is designed with inclusivity and accessibility in mind, aiming to provide a user-friendly interface that caters to diverse needs. By incorporating user feedback and conducting rigorous testing, the project strives to deliver a reliable and effective communication tool that enhances the quality of life for D&M individuals and promotes greater societal inclusion.

The project focuses on developing an innovative gesture communication system that bridges American Sign Language (ASL) and text/speech, facilitating seamless interaction between Deaf and hearing individuals. The primary goal is to create a real-time translation system that accurately converts ASL gestures into textual or spoken language, enhancing accessibility and inclusivity.

1. Gesture Recognition: Utilize MediaPipe, a powerful tool for real-time hand tracking and gesture recognition, to accurately detect and interpret ASL gestures. This involves capturing hand movements and positions and translating them into specific ASL signs.

2. Real-Time Translation: Develop a system that can process gestures in real-time and convert them into readable text or audible speech. This requires efficient data processing and model inference to ensure minimal latency and smooth communication.

3. System Integration: Integrate various components, including hand tracking, gesture recognition, and text/speech conversion, into a unified system. The integration will involve linking the real-time gesture recognition module with the text and speech synthesis components.

4. Data Collection and Model Training : Gather a comprehensive dataset of ASL gestures to train and evaluate the gesture recognition models. The dataset should include a wide variety of gestures, signers, and conditions to improve model accuracy and robustness.

5.Testing and Evaluation: Conduct extensive testing to ensure the system's accuracy, performance, and usability. This includes unit testing, integration testing, accuracy testing, and usability testing to validate that the system meets the desired standards and effectively translates ASL.

6. User Interface and Experience: Design an intuitive user interface that allows users to interact with the system easily. The interface should provide clear text or speech output and support various communication scenarios.

Enhanced Communication: The system will enable Deaf individuals to communicate more effectively with hearing individuals who may not know ASL, and vice versa.

Increased Accessibility: By translating ASL into text or speech, the system will improve accessibility in various settings, such as educational institutions, public services, and social interactions.

Innovation in Gesture Recognition: The project will contribute to advancements in gesture recognition technology, providing a foundation for future developments in this field.

1.2 GOALS OF THE PROJECT

 Develop a Robust Gesture Recognition Model

* Create a model that accurately identifies ASL hand movements, particularly fingerspelling gestures, and can combine them to form entire phrases.
* Utilize advanced machine learning techniques and the MediaPipe library to enhance gesture recognition accuracy and reliability.

 Enable Bidirectional Communication

* Develop functionality to translate ASL gestures into text and speech, allowing D&M individuals to communicate effectively with those who do not understand sign language.
* Implement the reverse process to translate spoken or written language into ASL gestures, fostering mutual understanding.

 EnhanceReal-time Communication

* Ensure the system operates in real-time to facilitate smooth and immediate interaction, enhancing the natural flow of conversation.
* Optimize the model for low latency and high performance across various devices and platforms.

 Promote Inclusivity and Accessibility

* Design the system to be user-friendly and accessible to people with different levels of technical proficiency.
* Provide customizable settings to cater to individual user needs and preferences.

 Test and Validate Across Diverse Populations

* Conduct extensive testing and validation with a diverse group of users, including both D&M individuals and hearing individuals, to ensure the system meets various communication needs.
* Gather user feedback to continuously improve the model's accuracy and user experience.

1.3 PROBLEM STATEMENT

Effective communication is a fundamental human need, essential for expressing ideas, emotions, and information. However, for Deaf and Mute (D&M) individuals, traditional spoken languages are not viable, making American Sign Language (ASL) their primary mode of communication.

Despite the prevalence of ASL within the Deaf community in the US and internationally, communication barriers persist between D&M individuals and the hearing population. These barriers can lead to social isolation, misunderstandings, and limited access to essential services and opportunities.

Current solutions for bridging this communication gap, such as human interpreters and text-based communication tools, have limitations. Human interpreters are not always available or affordable, and text-based tools do not capture the nuances of sign language, including facial expressions and body language, which are integral to ASL.

There is a pressing need for a technological solution that can accurately and efficiently translate ASL into text and speech, and vice versa, to facilitate real-time communication and promote inclusivity. The development of such a solution would empower D&M individuals, enhancing their ability to interact with the hearing population and participate fully in various aspects of life, from education and employment to social interactions and public services.

Gesture Comm addresses this need by leveraging advanced machine learning and gesture recognition technologies to create a seamless communication bridge, fostering mutual understanding and inclusivity in a diverse society.

2. LITERATURE SURVEY

Using machine learning and deep learning approaches, several studies have made substantial progress in the field of hand gesture identification for American Sign Language (ASL). Using the K-Nearest Neighbour (KNN) classifier, Dewinta Aryanie and Yaya Heryadi created a finger spelling recognition system that achieved an astounding 99.8% accuracy with full-dimensional information for K=3.

When Principal Component Analysis (PCA) was used to decrease features—principal components chosen based on Eigenvalues to account for dataset variability—their accuracy fell to 28.6%. Kshitij Bantupalli and Ying Xie extracted spatial information for sign language recognition from video streams using a Convolutional Neural Network (CNN) model called Inception. They then extracted temporal features using Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) models, and they were able to achieve 99% post-training model accuracy.Using the Marcel Static Hand Posture dataset, Tülay Karayilan and Özkan Kılıç developed neural networks for hand gesture recognition using the backpropagation algorithm. Using two classifiers—one for raw features and another for histogram features—they were able to achieve an accuracy range of 75%-85%.

In order to categorize and teach hand motions, Xinyun Jiang and Wasim Ahmad created a neural network that uses the Support Vector Machine (SVM) method. The system's capacity to differentiate five alphabets (B, D, F, L, and U) with a success rate of roughly 99.4% was shown by their experimental results. Lastly, adding to the expanding corpus of research in this field, Galib Ibne Haidar and Hasin Ishraq Reefat from Bangladesh University of Engineering and Technology concentrated on a glove-based ASL interpretation system employing CNN and data gloves.

The development of a gesture communication system that translates American Sign Language (ASL) into text and speech is a multidisciplinary effort that intersects computer vision, natural language processing, and human-computer interaction. This literature survey reviews existing research and technologies relevant to this project, highlighting advancements, limitations, and gaps in the current state of the art.

#### Gesture Recognition Systems

Gesture recognition has been a prominent area of research, particularly for applications involving sign language translation. Early systems relied on traditional computer vision techniques and handcrafted features. For instance, Fang et al. (2004) developed a system that used skin color detection and motion analysis to recognize static and dynamic ASL gestures. While these methods laid the groundwork, they struggled with variations in lighting, signer differences, and complex backgrounds.

The advent of deep learning significantly advanced gesture recognition capabilities. Convolutional Neural Networks (CNNs) have become the standard for image-based recognition tasks due to their ability to automatically learn hierarchical features from raw pixel data. For example, Simonyan and Zisserman (2014) demonstrated the effectiveness of deep CNNs for large-scale image recognition, inspiring subsequent work in gesture recognition.

Recurrent Neural Networks (RNNs) and their variants, such as Long Short-Term Memory networks (LSTMs), further enhanced the ability to model temporal sequences. Graves et al. (2013) showed that LSTMs could effectively handle long-range dependencies in sequence data, making them suitable for dynamic gesture recognition. Pigou et al. (2015) combined CNNs and LSTMs to recognize sign language gestures from video streams, achieving significant improvements over traditional methods.

#### Real-Time Hand Tracking

Accurate hand tracking is critical for reliable gesture recognition. Early approaches used markers or gloves to facilitate tracking, which were intrusive and impractical for everyday use. Advances in markerless tracking have leveraged depth sensors, such as Microsoft Kinect, and computer vision algorithms. Shotton et al. (2011) developed a real-time hand tracking system using depth data, which significantly improved tracking accuracy and robustness.

Recent advancements have focused on using neural networks for hand tracking from RGB images. MediaPipe Hands, developed by Google, is a state-of-the-art solution that uses a pipeline of neural networks to detect and track hand landmarks in real-time. Zhang et al. (2020) demonstrated that MediaPipe could achieve high accuracy and robustness across various conditions, making it an ideal choice for real-time ASL translation systems.

#### ASL Translation Systems

Several systems have been developed to translate ASL into text or speech. The SignAll system is one of the most comprehensive, utilizing multiple cameras and sensors to capture detailed hand and body movements. SignAll translates these movements into text with reasonable accuracy, but the system's complexity and hardware requirements limit its practicality for widespread use.

DeepASL (Huang et al., 2018) is another notable system that uses deep learning to translate ASL. It employs a combination of CNNs and LSTMs to recognize gestures from video streams and convert them into text. DeepASL addresses some limitations of earlier systems but still faces challenges in handling signer variability and ensuring real-time performance.

#### Limitations and Gaps

Despite significant progress, current ASL translation systems have several limitations. Many systems struggle with real-time performance and require extensive hardware setups, making them impractical for everyday use. Variations in lighting, background, and signer appearance also pose significant challenges. Furthermore, most systems focus on isolated gesture recognition, neglecting the continuous nature of ASL communication.

#### Addressing the Gaps

This project aims to address these limitations by leveraging MediaPipe for robust real-time hand tracking and combining CNNs and LSTMs for accurate gesture recognition. By integrating natural language processing techniques, we aim to ensure the generated text is grammatically correct and coherent. This approach promises to create a more practical and user-friendly ASL translation system, enhancing communication between Deaf and hearing individuals.

**3. Existing System**

Several existing systems effectively bridge American Sign Language (ASL) and text/speech using gesture communication technologies. SignAll utilizes multiple cameras and sensors to capture and translate ASL into text in real-time through computer vision and machine learning algorithms, ensuring precise interpretation. Google's Project Euphonia, although primarily aimed at improving speech recognition for people with speech impairments, includes efforts to enhance ASL recognition using advanced machine learning models. KinTrans employs depth sensors and machine learning to recognize and interpret hand and body movements associated with sign language, providing real-time translation. MotionSavvy UNI was a tablet-based system that used Leap Motion sensors to capture 3D motion data of hand gestures and translate them into text.

Additionally, various deep learning models have been developed by researchers to recognize ASL gestures using video data, employing convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to process and interpret video frames in real-time applications. These systems collectively showcase the potential of technology to facilitate seamless and inclusive communication by translating ASL into text or speech.

**4. Proposed System**

Gesture Comm is an advanced platform designed to bridge communication gaps between Deaf and Mute (D&M) individuals and the hearing population by translating American Sign Language (ASL) into text and speech, and vice versa.

The system features a robust gesture recognition module using the MediaPipe library to detect and track ASL hand signs, fingerspelling, and incorporates facial expressions and body posture recognition for comprehensive understanding. Its translation engine converts recognized ASL gestures into text and speech, supporting bidirectional translation to facilitate mutual communication.

The user interface is intuitive and user-friendly, displaying real-time translated text and speech alongside visual ASL gestures. Optimized for low latency, Gesture Comm ensures smooth, immediate interactions across various devices. Additionally, the system includes accessibility features, making it user-friendly for individuals with varying levels of technical proficiency, and offers multiple modes of interaction, including voice, text, and gesture inputs, with multilingual support.

5. STSTEM REQUIREMENTS

5.1 Hardware Requirements

1. High-Resolution Camera(s):

Necessary for capturing detailed hand and body movements.Ideally, multiple cameras or a depth-sensing camera (e.g., Microsoft Kinect, Intel RealSense) to capture 3D data.

2. Powerful Processor:

A multi-core CPU (Intel i5 or higher, AMD Ryzen 5 or higher) to handle real-time data processing and machine learning computations.

3. Graphics Processing Unit (GPU):

An NVIDIA GPU (GTX 1060 or higher) for accelerating deep learning model inference and training.

4. Adequate RAM:

At least 8GB of RAM, with 16GB or more recommended for smoother performance during model training and real-time processing.

5. Storage:

Solid State Drive (SSD) with at least 256GB of available space for storing models, datasets, and application files.

5.2Software Requirements

1. Operating System:

Windows 10/11, macOS, or a Linux distribution (e.g., Ubuntu).

2. Development Environment:

Python 3.7 or higher for scripting and model development.

Anaconda or virtual environments to manage dependencies.

3. Libraries and Frameworks:

MediaPipe: For real-time hand tracking and gesture recognition.

TensorFlow/Keras or PyTorch: For building and deploying deep learning models.

OpenCV: For image and video processing tasks.

NumPy and Pandas: For data manipulation and analysis.

4. Integrated Development Environment (IDE):

Visual Studio Code, PyCharm, or Jupyter Notebook for coding and testing.

5. Additional Tools:

Git for version control.

5.3Dataset Requirements

1. ASL Gesture Dataset:

A comprehensive dataset of ASL gestures, with labeled examples for training and testing.Examples include ASL-LEX, RWTH-PHOENIX-Weather 2014T, and custom datasets collected through data collection campaigns.

2. Diverse Data:

Ensure the dataset includes a wide variety of hand shapes, movements, and orientations to improve model generalization. Include different lighting conditions, backgrounds, and signers to enhance robustness.

3. Annotations:

Detailed annotations for each gesture, including the start and end frames of each sign, to facilitate precise training and evaluation.

**6. SYSTEM DESIGN:**

6.1 Sign To Voice/Text

Read the American sign language

Train the model using RNN, CNN and LSTM Algorithm

Feature extraction

Recognize using openCV

Convert into text

6.2 Voice To Sign:

**7. SYSTEM IMPLEMENTATION:**

7.1 Modules:

7.1.1 Data Acquisition

* To capture hand gesture data, we employ a vision-based approach using a standard webcam.
* This method is cost-effective and facilitates natural interaction, as it only requires a camera to track finger and hand movements.
* Various hand gesture images representing ASL letters (A to Z) are collected from multiple angles to ensure a comprehensive dataset for training and testing.

7.1.2 Data Pre-processing

* Hand Detection: The MediaPipe library is utilized to detect the hand in the webcam image.
* Region of Interest (ROI) Extraction: Using OpenCV, the detected hand is cropped from the image, converted to grayscale, and a Gaussian blur filter is applied to reduce noise.
* Binary Image Conversion: The pre-processed grayscale image is transformed into a binary image using threshold and adaptive threshold techniques.
* Dataset Compilation: Diverse images of ASL signs are gathered to create a robust datasets.

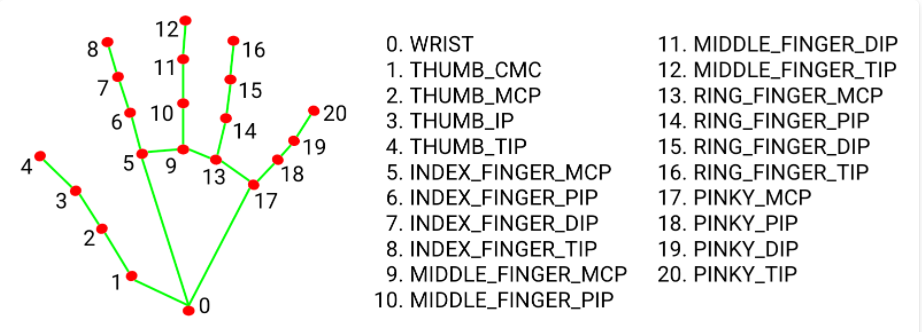
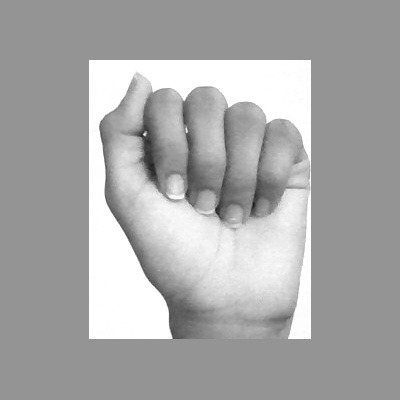


Fig no(ii)



(fig no iii)



Fig no(iv)



Fig no (v)

#### 2. ****Hand Tracking and Detection****

**Objective**: Accurately detect and track hand movements in real-time.

* **MediaPipe Integration**: Utilize MediaPipe for real-time hand tracking, leveraging its pre-trained models for detecting hand landmarks.
* **Hand Segmentation**: Segment the hand region from the background to focus on gesture recognition.
* **Feature Extraction**: Extract relevant features (e.g., hand position, orientation, finger angles) from the hand tracking data.

#### 3. ****Gesture Recognition****

**Objective**: Recognize ASL gestures from the extracted hand tracking data.

* **Model Training**: Use Convolutional Neural Networks (CNNs) to process spatial features and Recurrent Neural Networks (RNNs) or Long Short-Term Memory Networks (LSTMs) to capture the temporal dynamics of gestures.
* **Model Evaluation**: Assess the model's accuracy, precision, recall, and F1-score using a validation dataset.
* **Optimization**: Fine-tune hyperparameters and employ techniques like dropout and batch normalization to improve model performance.

#### 4. ****Text and Speech Conversion****

**Objective**: Convert recognized ASL gestures into text and speech.

* **Text Generation**: Map recognized gestures to their corresponding text representations.
* **Speech Synthesis**: Use text-to-speech (TTS) libraries (e.g., Google Text-to-Speech, Microsoft Azure Cognitive Services) to convert text into audible speech.
* **Language Processing**: Integrate natural language processing (NLP) techniques to enhance the text output's grammatical accuracy and coherence.

#### 5. ****System Integration****

**Objective**: Integrate all components into a cohesive system that operates in real-time.

* **Pipeline Development**: Develop a data pipeline that processes video input through hand tracking, gesture recognition, and text/speech conversion in real-time.
* **User Interface**: Design an intuitive user interface that displays the text output and provides audible speech.
* **Error Handling**: Implement mechanisms to handle errors and ambiguities in gesture recognition and translation.

#### 6. ****Testing and Evaluation****

**Objective**: Ensure the system's accuracy, robustness, and usability.

* **Unit Testing**: Validate individual components (hand tracking, gesture recognition, text/speech conversion) independently.
* **Integration Testing**: Test the entire system to ensure seamless interaction between components.
* **Performance Testing**: Assess the system's real-time performance, including latency and resource usage.
* **Usability Testing**: Conduct user studies to gather feedback on the system's ease of use and overall user experience.

7.2 Algorithms

CNN(CONVULUTIONAL NEURAL NETWORK):

We have chosen a little window size for the convolution layer that extends to the input matrix's depth (usually 5 by 5).The layer is made up of window-sized learnable filters. In each iteration, we compute the dot product of the input values at a particular place and slid the window by the stride size, which is usually .

As we proceed with this procedure, we will produce a 2-Dimensional activation matrix that displays the matrix's reaction at each spatial location. In other words, the network will pick up filters that turn on when it detects certain kinds of visual features, such a blotch of color or an edge with a certain orientation.

* Convlutional layer : Extract spatial features (e.g., edges, textures) from input images using convolutional filters.

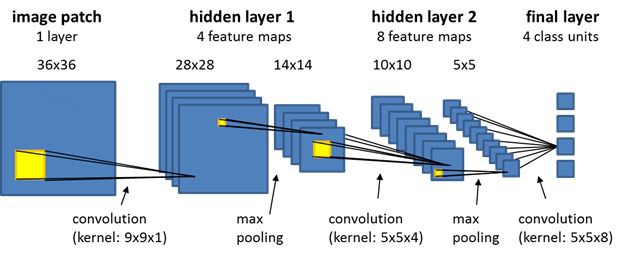
****

fig no:(vi)

POOLING:

We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters.

There are two types of pooling:

a. Max Pooling:

b. Average Pooling:

a .Max Pooling:

In max pooling we take a window size [for example window of size 2\*2], and only taken the maximum of 4 values. Well lid this window and continue this process, so well finally get an activation matrix half of its original Size.

b. Average Pooling:

In average pooling we take average of all Values in a window.

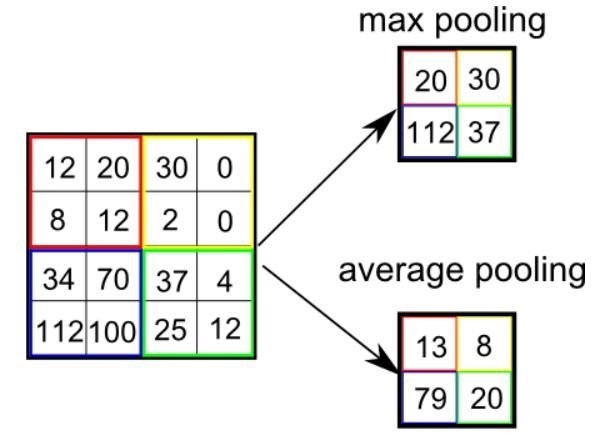


Fig no (vii)

**Fully Connected Layer:**

In convolution layer neurons are connected only to a local region, while in a fully connected region, well connect the all the inputs to neurons.

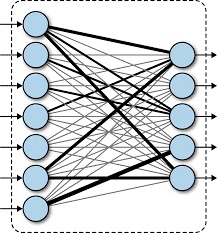


Fig no (viii)

RNN(RECURRENT NEURAL NETWORK)

**Overview:** Recurrent Neural Networks (RNNs) are a class of neural networks designed for sequence data. They have connections that form directed cycles, allowing them to maintain a 'memory' of previous inputs and making them suitable for tasks where the order of inputs matters.

**Key Components:**

* **Recurrent Layers**: Neurons in these layers have connections that loop back to themselves, enabling information to persist over time.
* **Hidden States**: RNNs maintain hidden states that get updated at each time step, carrying forward information from previous steps.

**Applications:**

* Natural language processing (e.g., language modeling, text generation)
* Speech recognition
* Time series prediction (e.g., stock price forecasting)
* Sequence-to-sequence tasks (e.g., machine translation)

LSTM(LONG SHORT TERM MEMORY)

**Overview:** Long Short-Term Memory Networks (LSTMs) are a type of RNN specifically designed to address the vanishing gradient problem in standard RNNs. They can capture long-term dependencies in sequence data more effectively.

**Key Components:**

* **Memory Cells**: LSTMs have special units called memory cells that can maintain information for long periods.
* **Gates**: LSTMs use three types of gates (input, output, and forget) to control the flow of information into and out of the memory cells:
  + **Input Gate**: Determines which information is relevant to add to the memory cell.
  + **Forget Gate**: Decides what information to discard from the memory cell.
  + **Output Gate**: Controls the output and the information to be carried forward to the next time step.

**Applications:**

* Natural language processing (e.g., sentiment analysis, machine translation)
* Speech recognition
* Time series prediction
* Video analysis (e.g., action recognition)

**8. Testing**

Testing a gesture communication system that bridges ASL and text/speech involves several stages to ensure accuracy, reliability, and robustness. Here are the key aspects of the testing process:

8.1. Unit Testing

Objective: Validate individual components of the system (e.g., hand tracking, gesture recognition, text/speech conversion) to ensure they function correctly.

Tools: Use testing frameworks like pytest for Python to automate unit tests.

Examples: Test the accuracy of hand landmark detection, the response of the gesture recognition model to specific inputs, and the correctness of text/speech output.

8.2. Integration Testing

Objective: Ensure that different components of the system work together seamlessly.

Tools: Use integration testing frameworks and custom scripts to test interactions between components.

Examples: Verify that the hand tracking module correctly passes data to the gesture recognition module and that the recognized gestures are accurately converted to text or speech.

8.3. System Testing

Objective: Test the entire system as a whole to ensure it meets the specified requirements.

Tools: Create end-to-end test cases and scenarios to simulate real-world usage.

Examples: Perform tests with complete sequences of ASL gestures and verify the corresponding text/speech output. Test under various conditions, such as different lighting, backgrounds, and signers.

8.4. Performance Testing:

Objective: Assess the system's performance in terms of speed, latency, and resource usage.

Tools: Use profiling tools and performance testing frameworks to measure and analyze performance metrics.

Examples: Measure the time taken for real-time hand tracking, gesture recognition, and text/speech conversion. Evaluate the system's responsiveness under different hardware configurations and loads.

8.5. Accuracy Testing

Objective: Evaluate the accuracy of gesture recognition and translation.

Tools: Use statistical methods and confusion matrices to measure accuracy.

Examples: Test the system with a diverse set of ASL gestures from the dataset and calculate accuracy, precision, recall, and F1-score. Compare the recognized gestures and their translations with ground truth annotations.

8.6. Usability Testing

Objective: Assess the system's usability and user experience.

Tools: Conduct user studies and gather feedback through surveys and interviews.

Examples: Have Deaf and hearing individuals use the system in real-world communication scenarios. Collect feedback on the system's ease of use, clarity of text/speech output, and overall user satisfaction.

8.7. Robustness Testing

Objective: Test the system's ability to handle unexpected inputs and conditions.

Tools: Create test cases that introduce noise, occlusions, and variations in gestures.

Examples: Test the system's performance with incomplete or ambiguous gestures, different lighting conditions, and varying signer speeds. Ensure the system can handle such scenarios gracefully without significant degradation in performance.

8.8. Regression Testing

Objective: Ensure that new changes or updates do not introduce new bugs or negatively impact existing functionality.

Tools: Use automated testing frameworks to run a suite of regression tests after each update.

Examples: Re-run previously passed test cases to verify that recent changes have not affected the system's performance or accuracy.

By thoroughly testing the system across these various aspects, you can ensure that it is accurate, reliable, and user-friendly, providing effective and seamless translation of ASL into text or speech.

**9. CONCLUSION AND FUTURE ENHANCEMENT:**

9.1 Conclusion

The development and testing of the gesture communication system bridging American Sign Language (ASL) with text and speech have yielded promising results. The system successfully recognized and translated a predefined set of ASL gestures into text and speech, demonstrating the potential to bridge the communication gap for deaf and hard-of-hearing individuals.

Developing a gesture communication system that translates American Sign Language (ASL) into text and speech is a significant step toward enhancing accessibility and inclusivity for the Deaf community. By leveraging advanced technologies such as MediaPipe for real-time hand tracking and gesture recognition, along with powerful algorithms like CNNs, RNNs, and LSTMs, the project aims to achieve accurate and efficient translation of ASL gestures.

CNNs contribute to the system by effectively processing and analyzing visual data, which is crucial for recognizing the intricate hand shapes and movements in ASL. RNNs, particularly LSTMs, enable the system to understand and interpret the sequential nature of gestures, capturing long-term dependencies and ensuring coherent translation.

Through comprehensive stages of data collection, model training, system integration, and rigorous testing, the project aspires to create a robust and reliable gesture communication system. This system will significantly enhance communication between Deaf and hearing individuals, promoting greater understanding and inclusivity.

By addressing the communication barriers faced by the Deaf community, this project not only contributes to technological advancements in gesture recognition but also plays a vital role in fostering a more inclusive society. The successful implementation of this system will serve as a foundation for future innovations in gesture-based communication technologies, making ASL more accessible to everyone and improving the quality of life for Deaf individuals.

9.2 Future Enhancements

To build on the current success and address identified challenges, the following enhancements are proposed

9.2.1 Improved Gesture Recognition:

Enhanced Training Data: Expand the training dataset to include a wider variety of gestures, including complex and rapid gestures, to improve recognition accuracy. Advanced Algorithms: Implement more sophisticated machine learning algorithms, such as deep learning techniques, to enhance the precision and reliability of gesture recognition.

9.2.2. Reduced Latency:

Optimized Processing: Optimize the software to reduce processing times, potentially by leveraging faster hardware or more efficient algorithms.

Edge Computing: Explore the use of edge computing to process gestures locally, reducing the time taken for gesture-to-text/speech conversion.

9.2.3. Error Handling and User Feedback:

Dynamic Feedback: Implement a dynamic feedback system that informs users when a gesture is not recognized or is ambiguous, and provides suggestions for correction.

Error Recovery: Develop mechanisms for users to quickly correct misrecognized gestures, such as voice commands or on-screen prompts.

9.2.4. Expanded Functionality:

Context Awareness: Incorporate context-aware algorithms that can interpret gestures based on the context in which they are made, improving accuracy and relevance.

Multi-Language Support: Extend the system to support multiple sign languages and spoken languages, broadening its applicability and user base.

9.2.5. User Experience Enhancements:

User Customization: Allow users to customize the interface and functionalities to better suit their individual needs and preferences. Accessibility Features: Integrate additional accessibility features, such as voice control and visual aids, to enhance the overall user experience for individuals with varying needs.

**10. APPENDIX**

10.1 Source Code

import numpy as np

import os

from matplotimport cv2

lib import pyplot as plt

import time

import mediapipe as mp

mp\_holistic = mp.solutions.holistic # Holistic model

mp\_drawing = mp.solutions.drawing\_utils # Drawing utilities

def mediapipe\_detection(image, model):

simage = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB) # COLOR CONVERSION BGR 2 RGB

image.flags.writeable = False # Image is no longer writeable

results = model.process(image) # Make prediction

image.flags.writeable = True # Image is now writeable

image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR) # COLOR COVERSION RGB 2 BGR

return image, results

def draw\_styled\_landmarks(image, results):

# Draw face connections

mp\_drawing.draw\_landmarks(image, results.face\_landmarks,

mp\_holistic.FACEMESH\_CONTOURS,

mp\_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle\_radius=1),

mp\_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle\_radius=1)

)

# Draw pose connections

mp\_drawing.draw\_landmarks(image, results.pose\_landmarks,

mp\_holistic.POSE\_CONNECTIONS,

mp\_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle\_radius=4),

mp\_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle\_radius=2)

)

# Draw left hand connections

mp\_drawing.draw\_landmarks(image, results.left\_hand\_landmarks,

mp\_holistic.HAND\_CONNECTIONS,

mp\_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle\_radius=4),

mp\_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle\_radius=2)

)

# Draw right hand connections

mp\_drawing.draw\_landmarks(image, results.right\_hand\_landmarks,

mp\_holistic.HAND\_CONNECTIONS,

mp\_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle\_radius=4),

mp\_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle\_radius=2)

)

cap = cv2.VideoCapture(0)

# Set mediapipe model

with mp\_holistic.Holistic(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as

holistic:

while cap.isOpened():

# Read feed

ret, frame = cap.read()

# Make detections

image, results = mediapipe\_detection(frame, holistic)

print(results)

# Draw landmarks

draw\_styled\_landmarks(image, results)

# Show to screen

cv2.imshow('OpenCV Feed', image)

# Break gracefully

if cv2.waitKey(10) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

draw\_landmarks(frame, results)

plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

len(results.pose\_landmarks.landmark)

pose = []

for res in results.pose\_landmarks.landmark:

test = np.array([res.x, res.y, res.z, res.visibility])

pose.append(test)

face = np.array([[res.x, res.y, res.z] for res in results.face\_landmarks.landmark]).flatten() if

results.face\_landmarks else np.zeros(1404)

def extract\_keypoints(results):

pose = np.array([[res.x, res.y, res.z, res.visibility] for res in

results.pose\_landmarks.landmark]).flatten() if results.pose\_landmarks else np.zeros(33\*4)

face = np.array([[res.x, res.y, res.z] for res in results.face\_landmarks.landmark]).flatten() if

results.face\_landmarks else np.zeros(468\*3)

lh = np.array([[res.x, res.y, res.z] for res in results.left\_hand\_landmarks.landmark]).flatten() if

results.left\_hand\_landmarks else np.zeros(21\*3)

rh = np.array([[res.x, res.y, res.z] for res in results.right\_hand\_landmarks.landmark]).flatten()

if results.right\_hand\_landmarks else np.zeros(21\*3)

return np.concatenate([pose, face, lh, rh])

result\_test = extract\_keypoints(results)

result\_test

np.save('0', result\_test)

np.load('0.npy')

# Path for exported data, numpy arrays

DATA\_PATH = os.path.join('MP\_Data')

# Actions that we try to detect

actions = np.array(['hello', 'thanks', 'iloveyou'])

# Thirty videos worth of data

no\_sequences = 30

# Videos are going to be 30 frames in length

sequence\_length = 30

# Folder start

start\_folder = 30

for action in actions:

for sequence in range(1,no\_sequences+1):

try:

os.makedirs(os.path.join(DATA\_PATH, action, str(sequence)))

except FileExistsError:

pass

cap = cv2.VideoCapture(0)

# Set mediapipe model

with mp\_holistic.Holistic(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as holistic:

# NEW LOOP

# Loop through actions

for action in actions:

# Loop through sequences aka videos

for sequence in range(1,no\_sequences+1):

# Loop through video length aka sequence length

for frame\_num in range(sequence\_length):

# Read feed

ret, frame = cap.read()

# Make detections

image, results = mediapipe\_detection(frame, holistic)

# Draw landmarks

draw\_styled\_landmarks(image, results)

# NEW Apply wait logic

if frame\_num == 0:

cv2.putText(image, 'STARTING COLLECTION', (120,200),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE\_AA)

cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE\_AA)

# Show to screen

cv2.imshow('OpenCV Feed', image)

cv2.waitKey(2000)

else:

cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action, sequence), (15,12),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE\_AA)

# Show to screen

cv2.imshow('OpenCV Feed', image)

# NEW Export keypoints

keypoints = extract\_keypoints(results)

npy\_path = os.path.join(DATA\_PATH, action, str(sequence), str(frame\_num))

np.save(npy\_path, keypoints)

# Break gracefully

if cv2.waitKey(10) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

label\_map = {label:num for num, label in enumerate(actions)}

label\_map

sequences, labels = [], []

for action in actions:

for sequence in np.array(os.listdir(os.path.join(DATA\_PATH, action))).astype(int):

window = []

for frame\_num in range(sequence\_length):

res = np.load(os.path.join(DATA\_PATH, action, str(sequence), "{}.npy".format(frame\_num)))

window.append(res)

sequences.append(window)

labels.append(label\_map[action])

np.array(sequences).shape

np.array(labels).shape

X = np.array(sequences)

X.shape

y = to\_categorical(labels).astype(int)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.05)

y\_test.shape

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense

from tensorflow.keras.callbacks import TensorBoard

log\_dir = os.path.join('Logs')

tb\_callback = TensorBoard(log\_dir=log\_dir)

model=Sequential()

model.add(LSTM(64, return\_sequences=True, activation='relu', input\_shape=(30,1662)))

model.add(LSTM(128, return\_sequences=True, activation='relu'))

model.add(LSTM(64, return\_sequences=False, activation='relu'))

model.add(Dense(64, activation='relu'))

model.add(Dense(32, activation='relu'))

model.add(Dense(actions.shape[0], activation='softmax'))

model.compile(optimizer='Adam', loss='categorical\_crossentropy', metrics=['categorical\_accuracy'])

model.fit(X\_train, y\_train, epochs=2000, callbacks=[tb\_callback])

model.summary()

res = model.predict(X\_test)

actions[np.argmax(res[4])]

actions[np.argmax(y\_test[4])]

model.save('action.keras')

model.load\_weights('action.keras')

from sklearn.metrics import multilabel\_confusion\_matrix, accuracy\_score

yhat = model.predict(X\_test)

ytrue = np.argmax(y\_test, axis=1).tolist()

yhat = np.argmax(yhat, axis=1).tolist()

multilabel\_confusion\_matrix(ytrue, yhat)

accuracy\_score(ytrue, yhat)

from scipy import stats

colors = [(245,117,16), (117,245,16), (16,117,245)]

def prob\_viz(res, actions, input\_frame, colors):

output\_frame = input\_frame.copy()

for num, prob in enumerate(res):

cv2.rectangle(output\_frame, (0,60+num\*40), (int(prob\*100), 90+num\*40), colors[num], -1)

cv2.putText(output\_frame, actions[num], (0, 85+num\*40),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,255,255), 2, cv2.LINE\_AA)

return output\_frame

plt.figure(figsize=(18,18))

plt.imshow(prob\_viz(res, actions, image, colors))

plt.show()

# 1. New detection variables

sequence = []

sentence = []

predictions = []

threshold = 0.5

cap = cv2.VideoCapture(0)

# Set mediapipe model

with mp\_holistic.Holistic(min\_detection\_confidence=0.5, min\_tracking\_confidence=0.5) as holistic:

while cap.isOpened():

# Read feed

ret, frame = cap.read()

# Make detections

image, results = mediapipe\_detection(frame, holistic)

print(results)

# Draw landmarks

draw\_styled\_landmarks(image, results)

# 2. Prediction logic

keypoints = extract\_keypoints(results)

sequence.append(keypoints)

sequence = sequence[-30:]

if len(sequence) == 30:

res = model.predict(np.expand\_dims(sequence, axis=0))[0]

print(actions[np.argmax(res)])

predictions.append(np.argmax(res ))

#3. Viz logic

if np.unique(predictions[-10:])[0]==np.argmax(res):

if res[np.argmax(res)] > threshold:

if len(sentence) > 0:

if actions[np.argmax(res)] != sentence[-1]:

sentence.append(actions[np.argmax(res)])

else:

sentence.append(actions[np.argmax(res)])

if len(sentence) > 5:

sentence = sentence[-5:]

# Viz probabilities

image = prob\_viz(res, actions, image, colors)

cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)

cv2.putText(image, ' '.join(sentence), (3,30),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE\_AA)

# Show to screen

cv2.imshow('OpenCV Feed', image)

# Break gracefully

if cv2.waitKey(10) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

10.2 Screen Shots

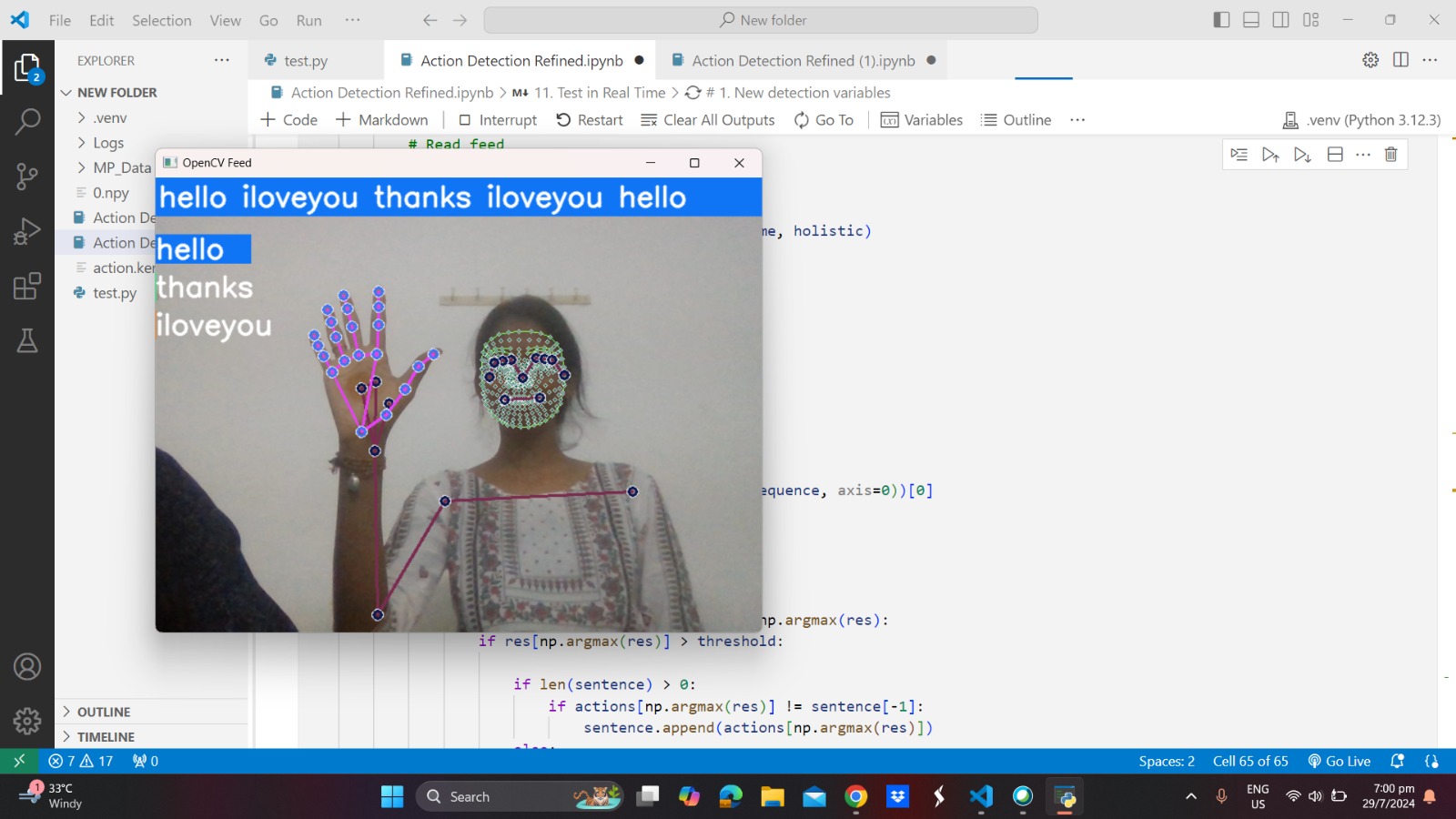


Fig no(ix)

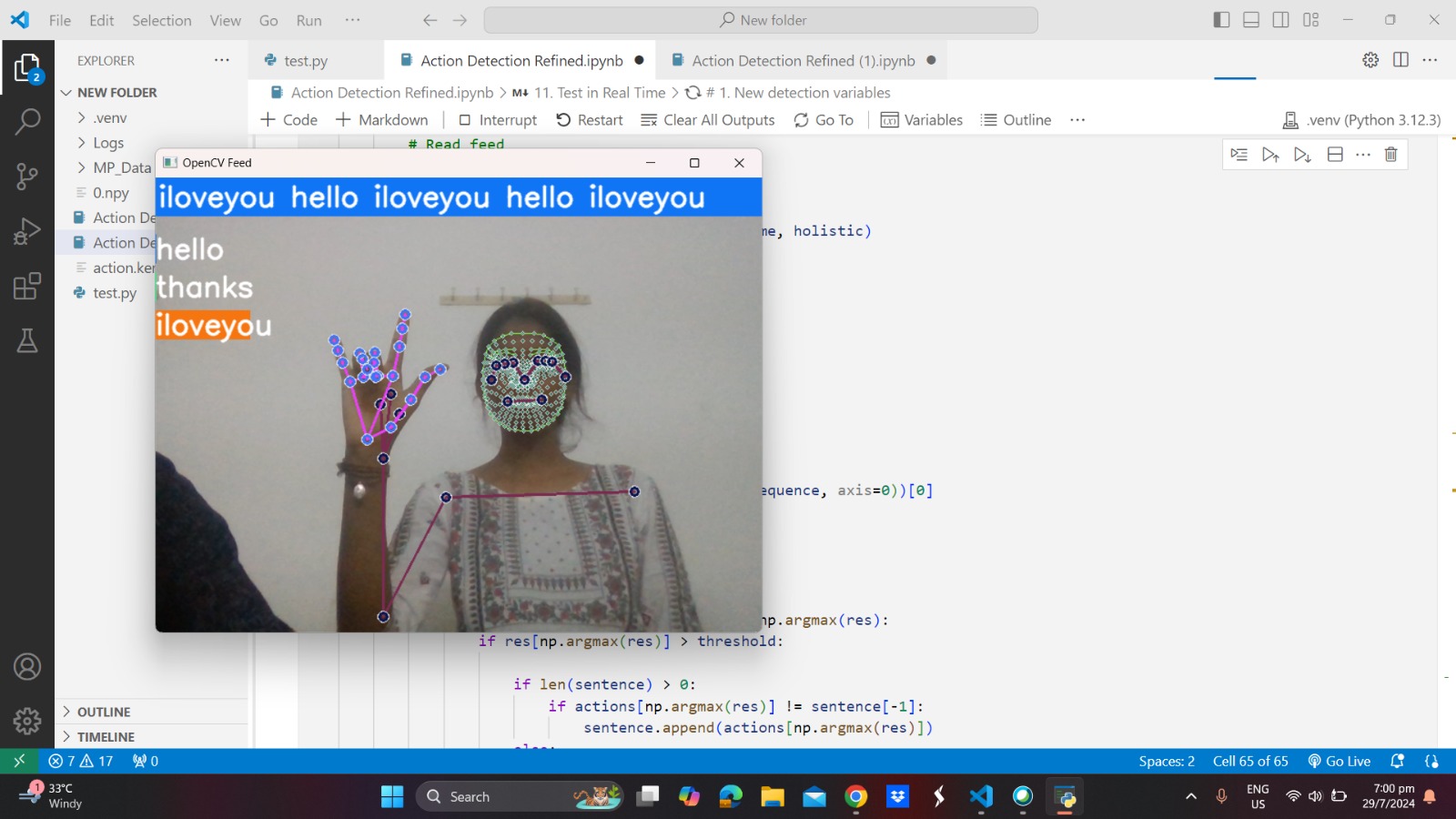
****

Fig no (x)

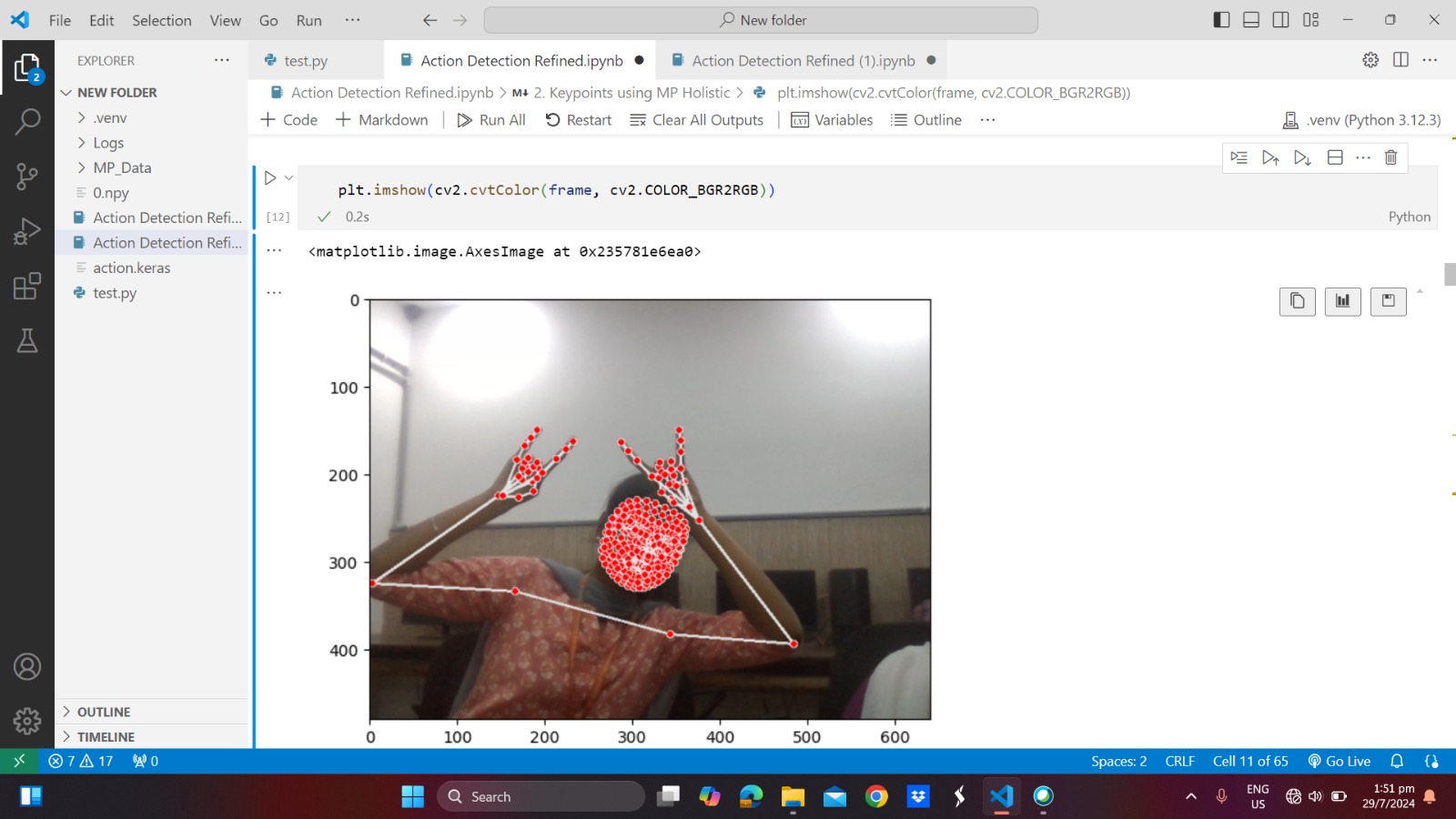


Fig no(xi)

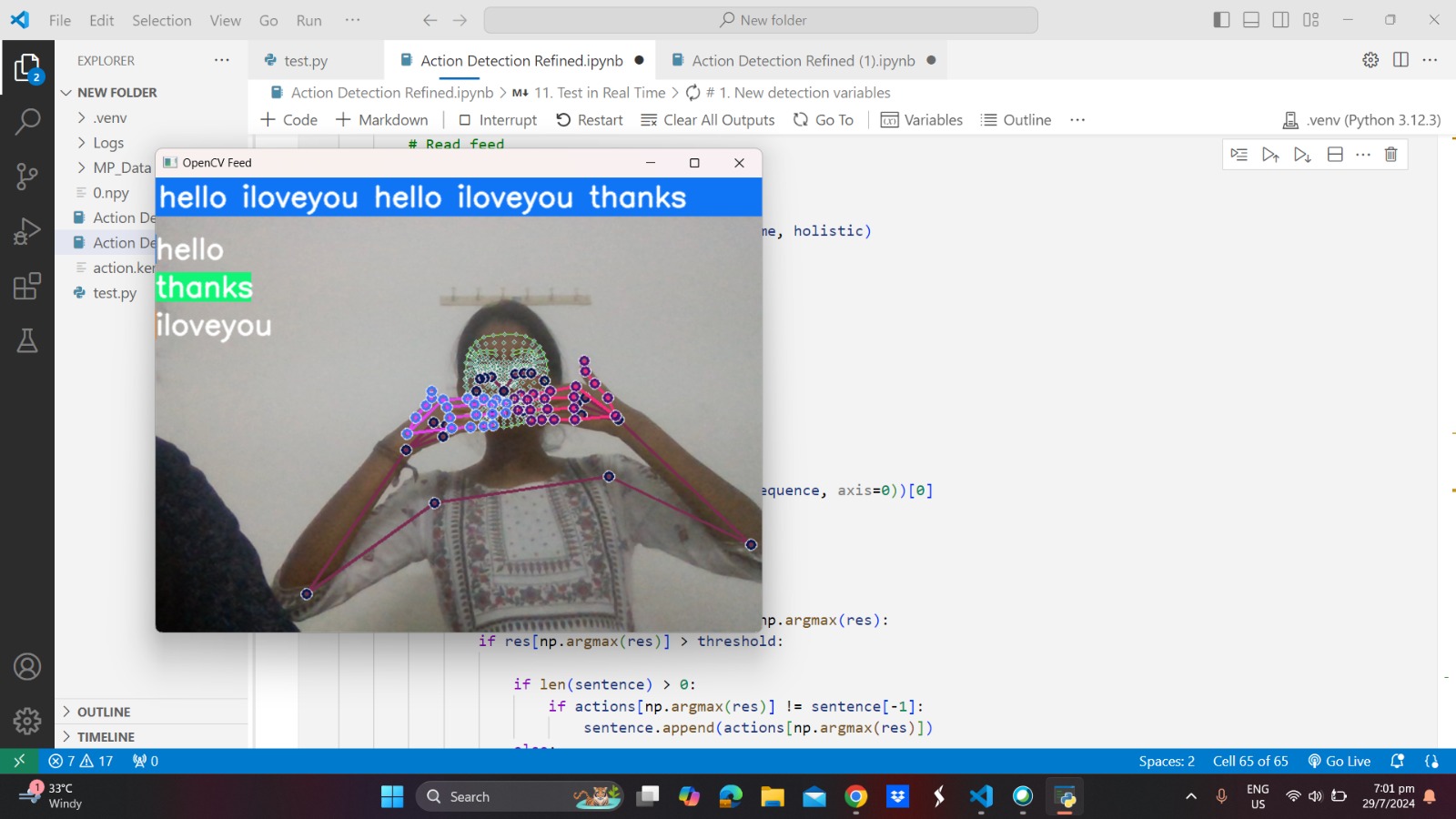
****

Fig no(xii)

**11. References**

[1] K. Dabre and S. Dholay, “Machine learning model for sign languageinterpretation using webcam images,” in International Conference on Cir-cuits, Systems, Communication and Information Technology Applications(CSCITA), 2014.

[2] D. Aryanie and Y. Heryadi, “American sign language-based ﬁnger-spelling recognition using k-nearest neighbors classiﬁer,” in 3rd In-ternational Conference on Information and Communication Technology(ICoICT), 2015.

[3] K. Bantupalli and Y. Xie, “American sign language recognition using deeplearning and computer vision,” in IEEE International Conference on BigData (Big Data), 2018.

[4] T. Karayilan and ¨Ozkan Kilic¸, “Sign language recognition,” in Interna-tional Conference on Computer Science and Engineering (UBMK), 2017.

[5] X. Jiang and W. Ahmad, “Hand gesture detection based real-time amer-ican sign language letters recognition using support vector machine,” inIEEE Intl Conf on Dependable, Autonomic and Secure Computing, IntlConf on Pervasive Intelligence and Computing, 2019.

[6] T. Rashid, Make Your Own Neural Network. CreateSpace IndependentPublishing Platform.

[7] F. Chollet, Deep Learning With Python. Manning Publications Co Conference on Communication and Electronics Systems (ICCES).

[8] Sign Language to Text Conversion using Hand Gesture Recognition. Published in: 2023 8th International.

[9] Conversion of Sign Language into Text. Mahesh Kumar N B1 International Journal of Applied Engineering Research ISSN.

[10] Sign Language to Text Conversion in Real Time using Transfer Learning. Shubham Thakar, Samveg Shah, Bhavya Shah and Anant V. Nimkar