

**Real Time Assistive Technology for Speech and Hearing Impairments using Deep Learning**

*A project report submitted in partial fulfilment of the requirement for the award of degree of*

# BACHELOR OF TECHNOLOGY

*In*

# COMPUTER SCIENCE AND ENGINEERING

*Submitted by*

**B. SURYA CHARAN (22341A0515)**

**D. ADITYA (22341A0540)**

**D. CHAITANYA KUMAR (22341A0545)**

**D. MANIKANTA (22341A0560)**

**B. ROSHITHA (23345A0501)**

*Under the esteemed guidance of*

**Dr. P. Kanchanamala**

Associate Professor, Dept. of CSE

GMR Institute of Technology

### An Autonomous Institute Affiliated to JNTU-GV, Vizianagaram

(Accredited by NBA, NAAC with ‘A’ Grade & ISO 9001:2015 Certified Institution)

### GMR Nagar, Rajam – 532127, Andhra Pradesh, India

**May 2025**



**Department of Computer Science and Engineering**

**CERTIFICATE**

This is to certify that the mini project entitled **Real-Time Assistive Technology for Hearing and Speech Impairments Using Deep Learning** submitted by **B. Surya Charan (22341A0515), D. Aditya (22341A0540), D. Chaitanya Kumar (22341A0545), G. Manikanta(22341A0560), B. Roshitha (23345A0501)** has been carried out in partial fulfilment of the requirement for the award of degree of **Bachelor of Technology in Computer Science and Engineering** of **GMRIT**, Rajam affiliated to **JNTUGV** is a record of bonafide work carried out by them under my guidance & supervision. The results embodied in this report have not been submitted to any other University or Institute for the award of any degree.

### Signature of Supervisor Signature of HOD

**Dr. P. Kanchanamala Dr. A. V. Ramana**

Associate Professor, Professor & Head,

Department of CSE, Department of CSE,

GMRIT, Rajam. GMRIT, Rajam.

The report is submitted for the viva-voce examination held on ………………..

Signature of Internal Examiner Signature of External Examiner

It gives us an immense pleasure to express deep sense of gratitude to my guide **Dr. P. Kanchanamala**, Associate Professor, Department of Computer Science and Engineering for her whole hearted and invaluable guidance throughout the project work. Without her sustained and sincere effort, this project work would not have taken this shape. She encouraged and helped us to overcome various difficulties that we have faced at various stages of our project work.

We would like to sincerely thank our Head of the department, **Dr. A. V. Ramana**, for providing all the necessary facilities that led to the successful completion of our project work. We would like to take this opportunity to thank our beloved Principal, **Dr. C. L. V. R. S. V. Prasad**, for providing all the necessary facilities and a great support to us in completing the project work.

We would like to thank all the faculty members and the non-teaching staff of the Department of Electronics and Communication Engineering for their direct or indirect support for helping us in completion of this project work.

Finally, we would like to thank all of our friends and family members for their continuous help and encouragement.

**B. SURYA CHARAN (22341A0515)**

**D. CHAITANYA KUMAR (22341A0545)**

**D. ADITYA (22341A0540)**

**G. MANIKANTA (22341A0560)**

**B. ROSHITHA (23345A0501)**

# ABSTRACT

Communication is essential, yet individuals with hearing and speech impairments face significant challenges in daily interactions. Current assistive technologies often provide limited functionalities, such as basic text-to-speech or speech-to-text, and lack real-time accuracy,seamless integration, and support for non-verbal communication like sign language. These limitations reduce their adaptability andeffectiveness in diverse environments. This paper proposes a novel assistive technology integrating real-time Sign-to-Text and Text-to-Speech functionalities. We propose a convolution neural network (CNN) method to recognize hand gestures of human actions from an image captured by camera. The purpose is to recognize hand gestures of human task activities from a camera image. The position of hand and orientation are applied to obtain the training and testing data for the CNN.. Additionally, a visionbased module enables real-time sign language recognition and translation into text, addressing a critical gap in existing solutions. Built using frameworks like TensorFlow and Keras, the system leverages advanced deep learning models to deliver accurate and adaptive functionalities. This comprehensive solution empowers individuals with hearing and speech impairments, enabling effective communication and inclusion in education, healthcare, and professional settings through real-time, adaptive, and accessible tools**.**

***Keywords:*** Deep Learning, Long Short-Term Memory (LSTM), Sign-to-text, Speech-to-Text, Text-to-Speech, TensorFlow

ACKNOWLEDGEMENT iii

[ABSTRACT iv](#_TOC_250014)

LIST OF TABLES vi

LIST OF SYMBOLS & ABBREVIATIONS vii

1. [INTRODUCTION](#_TOC_250013)
   1. [Problem statement 1](#_TOC_250012)
2. [RELATED WORK](#_TOC_250008)
   1. [Literature Survey 2](#_TOC_250007)

2.1 Comparison Table 9

1. [REQUIREMENT SPECIFICATION](#_TOC_250006)

3.2 Functional Requirements 19

* 1. [Non Functional Requirements 19](#_TOC_250005)
  2. [Software and Hardware Requirements 20](#_TOC_250004)

1. [SYSTEM ANALYSIS AND DESIGN](#_TOC_250003)
   1. [Existing Methodology 22](#_TOC_250002)
   2. Proposed Architecture 23
   3. Methodology 27
2. [IMPLEMENTATION 29](#_TOC_250001)
3. RESULTS AND DISCUSSIONS 51
4. CONCLUSIONS AND FUTURE SCOPE 53
5. [REFERENCES 55](#_TOC_250000)

**TABLE NO TITLE PAGE NO**

1. Comparison Table 9
2. Comparison between different models 51

| **Abbreviation** | **Full Form** |
| --- | --- |
| **AI** | Artificial Intelligence |
| **ASL** | American Sign Language |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **DNN** | Deep Neural Network |
| **FPS** | Frames Per Second |
| **GPU** | Graphics Processing Unit |
| **GUI** | Graphical User Interface |
| **ISL** | Indian Sign Language |
| **KNN** | K-Nearest Neighbors |
| **LSTM** | Long Short-Term Memory |
| **ML** | Machine Learning |
| **NLP** | Natural Language Processing |
| **RNN** | Recurrent Neural Network |
| **ROI** | Region of Interest |
| **SVM** | Support Vector Machine |
| **TTS** | Text-to-Speech |

**1. INTRODUCTION**

Communication is a fundamental aspect of human interaction, playing a pivotal role in personal, educational, and professional spheres. However, individuals with hearing and speech impairments often face substantial challenges in effectively conveying or receiving information, which can lead to social exclusion and limited opportunities. The existing assistive technologies, while helpful to some extent, are frequently constrained by high costs, limited accessibility, lack of real-time processing, and insufficient adaptability to users' specific needs.

The advent of deep learning has opened new frontiers in the development of intelligent systems capable of processing and interpreting human communication modalities with high accuracy and speed. Leveraging these advancements, this project presents the design and implementation of a **Real-Time Assistive Technology System for Hearing and Speech Impairments** using deep learning techniques.

The proposed system aims to facilitate bidirectional communication by converting spoken language into text for individuals with hearing impairments and translating visual cues, such as hand gestures or sign language, into audible speech for individuals with speech impairments. By incorporating real-time processing capabilities, the system ensures immediate feedback and interaction, thereby enhancing user experience and effectiveness.

The overarching objective of this project is to develop a solution that is not only technologically robust but also cost-effective, user-friendly, and scalable for widespread adoption. Through this initiative, we aspire to contribute to a more inclusive society by empowering individuals with hearing and speech disabilities to communicate independently and confidently in various real-world scenarios.

## 1.1Problem Statement

Individuals with hearing and speech impairments often encounter significant barriers in real-time communication, which can hinder their ability to participate fully in social, educational, and professional environments. Existing assistive technologies are often limited by high costs, lack of portability, and inadequate real-time responsiveness. There is a clear need for an intelligent, affordable, and real-time solution that can bridge the communication gap by accurately translating speech to text and gestures or sign language to speech. This project addresses this need by developing a deep learning-based assistive system that facilitates seamless, real-time interaction for users with hearing and speech disabilities.

## 2. RELATED WORK

## 2.1 Literature Survey

**Reference 1 :**

**Kumar, S., Rani, R., & Chaudhari, U. (2024). Real-time sign language detection: Empowering the**  **disabled community. MethodsX, 13, 102901.**

**Objective**: Develop a system to detect sign language gestures in real-time, aiming to enhance communication between hearing-impaired individuals and society. The research focuses on offering a practical tool to increase accessibility.

**Technologies Used** : Machine Learning, Real-Time Video Processing

**Performance Metrics:** Accuracy: 92%, Precision: 90%

**Limitations:**

* Limited to specific gestures.
* Low performance under poor lighting.

**Reference 2:**

**Gupta, A. D., Kumar, A., Chaudhary, I., Yasir, A. M., & Kumar, N. (2024, May). My Assistant SRSTC: Speech**  **Recognition and Speech to Text Conversion. In 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE) (pp. 394-400).**  **IEEE.**

**Objective**: The paper develops a speech-to-text conversion tool aimed at assisting people with disabilities and enhancing interactions. It focuses on speech recognition efficiency to increase accessibility in various environments.

**Technologies Used** : Deep Learning Algorithms, Natural Language Processing

**Performance Metrics:** Accuracy:92%, Precision:90%

**Limitations:**

* Struggles with noisy environments.
* Language processing limited to English

**Reference 3 :**

**Gupta, R., & Bagga, S. K. (2024, February). Text-to-Speech Conversion Technology using Deep Learning Algorithms. In 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1122-1126). IEEE.**

**Objective**: Focus on improving the quality of text-to-speech systems using deep learning techniques. This research explores different algorithms for generating more natural, lifelike synthesized speech from text.

**Technologies Used** : Neural Text-to-Speech Models, WaveNet

**Performance Metrics:** Naturalness: 94%, Latency: 150 ms

**Limitations:**

* High computational resources required.
* Ineffective for uncommon languages.

**Reference 4:**

**Yadava, T., Nagaraja, B. G., Reddy, S., Rohan, K., & Mohamed, L. M. (2024, September). Advancements in Speech-to-Text Systems for the Hearing Impaired. In 2024 IEEE North Karnataka Subsection Flagship International Conference (NKCon) (pp. 1-6). IEEE.**

**Objective**: The study discusses advances in speech-to-text technology that helps individuals with hearing impairment. It develops improved algorithms for faster and more accurate transcription of spoken language into text.

**Technologies Used** : Convolutional Neural Networks, Speech Signal Processing

**Performance Metrics:** Accuracy: 89%, Precision: 88%

**Limitations:**

* Poor accuracy with fast speech.
* Limited dataset for training.

**Reference 5:**

**Reddy, V. M., Vaishnavi, T., & Kumar, K. P. (2023, July). Speech-to-Text and Text-to-Speech** **Recognition Using Deep Learning. In 2023 2nd International Conference on Edge Computing and Applications (ICECAA) (pp. 657-666). IEEE.**

**Objective**: This paper explores the conversion of speech to text and text to speech using advanced deep learning techniques. The authors aim to enhance the real-time performance and accuracy of both processes.

**Technologies Used** : Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM) Networks

**Performance Metrics:** Accuracy: 91%, Precision: 87%

**Limitations:**

* High latency in real-time systems.
* Struggles with dialect variations.

**Reference 6:**

**Singh, P., Prasad, S. V. S., Singh, R., Dasari, K., & Prasanna, B. L. (2023, September). Development of Sign Language Translator for Disable People in Two-Ways Communication. In 2023 1st International**  **Conference on Circuits, Power and Intelligent Systems (CCPIS) (pp. 1-6). IEEE.**

**Objective**: This work focuses on the development of a bi-directional sign language translator, which enables both speech-to-sign language and sign-to-speech translations to facilitate easier two-way communication.

**Technologies Used** : Gesture Recognition Systems, Deep Neural Networks

**Performance Metrics:** Accuracy: 86%, Recall: 85%

**Limitations:**

* Limited gestures supported.
* High dependency on hardware specifications.

**Reference 7:**

**Poornima, B. V., & Srinath, S. (2023). A Comprehensive Review on Indian Sign Language Recognition System using Vision based Approaches. International Journal of Computer Applications, 184, 52-58.**

**Objective:** This review paper explores various vision-based approaches used in Indian sign language recognition. It discusses their strengths and limitations and gives insights into improving current models for better sign anguage understanding.

**Technologies Used** : Computer Vision, Deep Learning Models

**Performance Metrics:** Recognition Accuracy: 87%, Precision: 83%

**Limitations:**

* Limited recognition range.
* Lack of dynamic sign interpretation.

**Reference 8:**

**Ghorpade, T. H., & Shinde, S. K. (2023, August). Speech Synthesis: An Empirical Analysis of Various Techniques in Text to Speech Generation. In 2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA) (pp. 1-6). IEEE.**

**Objective:** This paper presents a comprehensive analysis of multiple speech synthesis techniques. It aims to assess which methods provide the most efficient and accurate text-to-speech conversion for real-time applications.

**Technologies Used** : WaveNet Models, Hidden Markov Models (HMM)

**Performance Metrics:** Synthesis Quality: 92%, Response Time: 170 ms

**Limitations:**

* Resource-intensive models.
* Limited for low-resource environments.

**Reference 9:**

**Khanam, F., Munmun, F. A., Ritu, N. A., Saha, A. K., & Firoz, M. (2022). Text to speech synthesis: a systematic review, deep learning-based architecture and future research direction. Journal of Advances in Information Technology, 13(5).**

**Objective:** This systematic review delves into the various deep learning architectures employed for text-to-speech synthesis. The paper suggests improvements and future directions for the field to enhance speech quality and versatility.

**Technologies Used** : Recurrent Neural Networks, Attention Mechanisms

**Performance Metrics:** Speech Quality: 95%, Conversion Time: 120 ms

**Limitations:**

* Training complexity.
* Requires high-end hardware.

**Reference 10:**

**Shashidhar, R., Hegde, S. R., Chinmaya, K., Priyesh, A., Manjunath, A. S., & Arunakumari, B. N. (2022, October). Indian Sign Language to Speech Conversion Using Convolutional Neural Network. In 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon) (pp. 1-5). IEEE.**

**Objective:** This paper proposes a convolutional neural network (CNN) for converting Indian sign language gestures into speech. It focuses on developing a reliable translation system for communicating with the speech-impaired community.

**Technologies Used** : Convolutional Neural Networks (CNN), Gesture Recognition

**Performance Metrics:** Accuracy: 89%, Precision: 84%

**Limitations:**

* Low performance with complex gestures.
* Dependency on sign language dataset.

**Reference 11:**

**Kothadiya, D., Pise, N., & Bedekar, M. (2020). Different methods review for speech to text and text to speech conversion. International Journal of Computer Applications, 975, 8887.**

**Objective:** The paper presents a review of various methods used in speech-to-text and text-to-speech conversion. It discusses the strengths and weaknesses of each approach to highlight the most effective for practical implementations.

**Technologies Used** : Speech Recognition Systems, HMM and DNN Techniques

**Performance Metrics:** Speech Recognition Accuracy: 87%, Text Synthesis Quality: 89%

**Limitations:**

* Accuracy is hindered by accents.
* Limited to fixed vocabulary sets.

**Reference 12:**

**Kowsigan, M., Dhawan, R., & Kundu, A. (2024, July). An Efficient Speech to Sign Language Conversion and Text Recognition through Live Gesture. In 2024 IEEE International Conference on Smart Power Control and Renewable Energy (ICSPCRE) (pp. 1-6). IEEE.**

**Objective:** This research focuses on creating an efficient conversion system from speech to sign language using live gestures. The aim is to recognize spoken words and translate them into both sign language gestures and text.

**Technologies Used** : Gesture Recognition, Machine Learning

**Performance Metrics:** Speech-to-Text Accuracy: 92%, Gesture Recognition Accuracy: 85%

**Limitations:**

* Real-time processing issues.
* Limited recognition for non-native signs.

**Reference 13 :**

**Seviappan, A., Ganesan, K., Anbumozhi, A., Reddy, A. S., Krishna, B. V., & Reddy, D. S. (2023, December). Sign Language to Text Conversion using RNN-LSTM. In 2023 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI) (pp. 1-6). IEEE.**

**Objective:** This paper explores a system to convert Indian Sign Language gestures into text using RNN-LSTM models. The goal is to enhance the accuracy and efficiency of sign language recognition for seamless human-computer interaction.

**Technologies Used** : Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM)

**Performance Metrics:** Recognition Accuracy: 90%, Processing Speed: 110 ms

**Limitations:**

* Small training data sets.
* Lacks dynamic gesture interpretation.

**Reference 14:**

**Patil, S., Gulave, S., Gawai, V., Gode, P., & Mudme, P. (2022, August). Conversion of Indian Sign Language to Speech by Using Deep Neural Network. In 2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA) (pp. 1-6). IEEE.**

**Objective:** The research focuses on developing a deep neural network (DNN) model for translating Indian Sign Language to speech. This conversion system improves communication for individuals who rely on sign language as their primary mode of communication.

**Technologies Used** : Deep Neural Networks (DNN), Gesture Recognition

**Performance Metrics:** Accuracy: 88%, Precision: 85%

**Limitations:**

* Limited real-time application.
* Performance impacted by gesture speed.

**Reference 15:**

**Najib, F. M. (2024). Sign language interpretation using machine learning and artificial intelligence. Neural Computing and Applications, 1-17.**

**Objective:** This paper investigates the role of machine learning and artificial intelligence in enhancing sign language interpretation. The author explores various models that can process and interpret sign language gestures with improved accuracy.

**Technologies Used** : Machine Learning, Artificial Intelligence

**Performance Metrics:** Accuracy: 93%, Precision: 91%

**Limitations:**

* Limited training set for real-world application.
* Difficulties with diverse signing styles.

## 2.2 Comparison Table

Table.1 Comparison Table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Title** | | **year** | **Objectives** | **Limitations** | **Advantages** | | **Performance metrics** | **Gaps** |
| **Reference 1** | Real-time sign language detection: Empowering the disabled community | | 2024 | Develop a system to detect sign language gestures in real-time, aiming to enhance communication between hearing-impaired individuals and society. The research focuses on offering a practical tool to increase accessibility. | Limited to specific gestures.  Low performance under poor lighting. | Provides a real-time system for sign language detection, aiding the disabled community. | | Accuracy: 92%  Precision: 90% | Limited to specific sign languages; lacks cultural inclusivity. |
| **Reference 2** | My Assistant SRSTC: Speech Recognition and Speech to Text Conversion | | 2024 | The paper develops a speech-to-text conversion tool aimed at assisting people with disabilities and enhancing interactions. It focuses on speech recognition efficiency to increase accessibility in various environments. | High computational resources required.  Ineffective for uncommon languages. | Integrates speech recognition and text conversion efficiently. | | Accuracy: 85%  Latency: 200 ms | Lacks scalability for multiple languages and accents. |
| **Reference 3** | Text-to-Speech Conversion Technology using Deep Learning Algorithms | | 2024 | Focus on improving the quality of text-to-speech systems using deep learning techniques. This research explores different algorithms for generating more natural, lifelike synthesized speech from text. | High computational resources required.  Ineffective for uncommon languages. | Utilizes deep learning for accurate text-to-speech conversion. | | Naturalness: 94%  Latency: 150 ms | Does not address real-time applications or multilingual support. |
| **Reference 4** | Advancements in Speech-to-Text Systems for the Hearing Impaired | | 2024 | The study discusses advances in speech-to-text technology that helps individuals with hearing impairment. It develops improved algorithms for faster and more accurate transcription of spoken language into text. | Poor accuracy with fast speech.  Limited dataset for training. | Advances speech-to-text for hearing-impaired individuals. | | Accuracy: 89%  Precision: 88% | Limited testing scope; does not fully explore real-world deployment. |
| **Reference 5** | | Speech-to-Text and Text-to-Speech Recognition Using Deep Learning | 2023 | This paper explores the conversion of speech to text and text to speech using advanced deep learning techniques. The authors aim to enhance the real-time performance and accuracy of both processes. | High latency in real-time systems.  Struggles with dialect variations. | | Combines speech-to-text and text-to-speech using deep learning. | Accuracy: 91%  Precision: 87% | Computationally expensive; lacks optimization for resource-constrained devices. |
| **Reference 6** | | Development of Sign Language Translator for Disable People in Two-Ways Communication | 2023 | This work focuses on the development of a bi-directional sign language translator, which enables both speech-to-sign language and sign-to-speech translations to facilitate easier two-way communication. | Limited gestures supported.  High dependency on hardware specifications. | | Enables two-way communication for disabled users with a translator. | Accuracy: 86%  Recall: 85% | Focuses on limited languages; lacks detailed usability testing. |
| **Reference 7** | | A Comprehensive Review on Indian Sign Language Recognition System using Vision based Approaches | 2023 | This review paper explores various vision-based approaches used in Indian sign language recognition. It discusses their strengths and limitations and gives insights into improving current models for better sign language understanding. | Limited recognition range.  Lack of dynamic sign interpretation. | | Comprehensive review of Indian sign language recognition methods. | Accuracy: 87%  Precision: 83% | Primarily theoretical; lacks implementation insights. |
| **Reference 8** | | Speech Synthesis: An Empirical Analysis of Various Techniques in Text to Speech Generation | 2023 | This paper presents a comprehensive analysis of multiple speech synthesis techniques. It aims to assess which methods provide the most efficient and accurate text-to-speech conversion for real-time applications. | Resource-intensive models.  Limited for low-resource environments. | | Analyzes various techniques in text-to-speech generation. | Synthesis Quality: 92%  Response Time: 170 ms | insufficient focus on user experience and practical deployment. |
| **Reference 9** | | Text to Speech Synthesis: A Systematic Review, Deep Learning-Based Architecture and Future Research Direction | 2022 | This systematic review delves into the various deep learning architectures employed for text-to-speech synthesis. The paper suggests improvements and future directions for the field to enhance speech quality and versatility. | Training complexity.  Requires high-end hardware. | | Comprehensive review of TTS technologies and future directions. | Speech Quality: 95%  Conversion Time: 120 ms | Lacks practical implementation and real-world use-case testing. |
| **Reference 10** | | Indian Sign Language to Speech Conversion Using Convolutional Neural Network | 2022 | This paper proposes a convolutional neural network (CNN) for converting Indian sign language gestures into speech. It focuses on developing a reliable translation system for communicating with the speech-impaired community. | Resource-intensive models.  Limited for low-resource environments. | | Employs CNN for accurate sign-to-speech conversion. | Accuracy: 89%  Precision: 84% | Limited to Indian Sign Language; lacks real-time system optimization. |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Reference 11** | Different Methods Review for Speech to Text and Text to Speech Conversion | 2020 | The paper presents a review of various methods used in speech-to-text and text-to-speech conversion. It discusses the strengths and weaknesses of each approach to highlight the most effective for practical implementations. | Accuracy is hindered by accents.  Limited to fixed vocabulary sets. | Highlights multiple approaches for speech-text conversions. | Speech Recognition Accuracy: 87%  Text Synthesis Quality: 89% | Outdated methods and limited focus on modern AI techniques. |
| **Reference 12** | An Efficient Speech to Sign Language Conversion and Text Recognition Through Live Gesture | 2024 | This research focuses on creating an efficient conversion system from speech to sign language using live gestures. The aim is to recognize spoken words and translate them into both sign language gestures and text. | Real-time processing issues.  Limited recognition for non-native signs. | Combines speech-to-sign and gesture-based text recognition. | Speech-to-Text Accuracy: 92%  Gesture Recognition Accuracy: 85% | Insufficient details on accuracy and user experience testing. |
| **Reference 13** | Sign Language to Text Conversion Using RNN-LSTM | 2023 | This paper explores a system to convert Indian Sign Language gestures into text using RNN-LSTM models. The goal is to enhance the accuracy and efficiency of sign language recognition for seamless human-computer interaction. | Small training data sets.  Lacks dynamic gesture interpretation. | Leverages RNN-LSTM for efficient sign-to-text conversion. | Recognition Accuracy: 90%  Processing Speed: 110 ms | High computational cost; limited scalability to diverse sign languages. |
| **Reference 14** | Conversion of Indian Sign Language to Speech by Using Deep Neural Network | 2022 | The research focuses on developing a deep neural network (DNN) model for translating Indian Sign Language to speech. This conversion system improves communication for individuals who rely on sign language as their primary mode of communication. | Limited real-time application.  Performance impacted by gesture speed. | Uses DNN for precise sign-to-speech conversion. | Accuracy: 88%  Precision: 85% | Focuses only on Indian Sign Language; lacks multilingual capabilities. |
| **Reference 15** | Sign Language Interpretation Using Machine Learning and Artificial Intelligence | 2024 | This paper investigates the role of machine learning and artificial intelligence in enhancing sign language interpretation. The author explores various models that can process and interpret sign language gestures with improved accuracy.  . | Limited training set for real-world application.  Difficulties with diverse signing styles. | Applies AI and ML for versatile sign language interpretation. | Accuracy: 93%  Precision: 91% | Limited experimentation across diverse sign languages and contexts. |

## 3. REQUIREMENT SPECIFICATION

## 3.1 Functional Requirements

**Input:** The system shall accept live video input or images of hand gestures captured through a webcam.

**Pre-processing:** The system shall pre-process the captured frames to enhance clarity and remove background noise using techniques such as resizing, normalization, grayscale conversion (if needed), and segmentation to isolate hand regions.

**Feature Extraction:** The system shall extract spatial features from the hand gesture using Convolutional Neural Network (CNN) layers. These features will represent:

* + - * Finger positioning
      * Hand orientation
      * Gesture contours

**Classification:** The system shall use a trained CNN model to classify hand gestures into their respective alphabet or word category.

**Text Conversion**: Once the gesture is classified, the corresponding text shall be displayed in a GUI window or console.

**Speech Conversion:** The system shall employ a Text-to-Speech (TTS) engine to convert the recognized text into spoken.

**Output:** The system shall display:

* + - * Recognized character/word on screen
      * Optionally, a sentence formed from sequential inputs
      * Spoken output of the recognized word or sentence.

## Non-Functional Requirements

**Accuracy:** The system shall achieve an accuracy of at least 90% in gesture recognition under good lighting and clear hand visibility.

**Performance:** The system shall provide real-time recognition with a latency of no more than 1 second per gesture/frame.

**Scalability:** The architecture should allow the addition of new gestures (words or letters) for training and recognition without re-training the full model from scratch.

**Security:** The system shall securely process and store image data, with no unnecessary image retention or sharing.

**User Interface:** The system shall provide a user-friendly interface using Tkinter or other GUI libraries, with options to View recognized text, Listen to the converted speech and Clear or reset the text.

## 3.3 Software and Hardware Requirements

**Software Requirements:**

* **Python:** Python will be used as the primary programming language for building the application, leveraging its strong support for machine learning and image processing.
* **Libraries and Frameworks:**
  + - * OpenCV: For image acquisition and preprocessing
      * TensorFlow / Keras: For building and training the CNN model
      * pyttsx3: For converting text to speech
      * Tkinter: For building the GUI
* **IDE (Integrated Development Environment):**
* **Visual Studio Code:** Visual Studio Code is a free and open-source code editor developed by Microsoft that provides an integrated development environment (IDE) for building and debugging applications. It is available on Windows, macOS, and Linux. Visual Studio Code supports a wide range of programming languages, including popular languages like JavaScript, Python, C++, and Java, as well as emerging languages like Rust and Go.
* **Jupyter NoteBook:** Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It provides an interactive computing environment that allows you to execute code in real-time, which is particularly useful for data analysis, scientific computing, and machine learning tasks. The code is executed in a kernel, which is a separate process that can be started and stopped independently of the notebook interface.
* **Hardware Requirements:**
* Processing Power: Require a computer with sufficient processing power, including a multi-core CPU or GPU (Graphics Processing Unit), to handle complex image processing algorithms and machine learning tasks efficiently.
* Memory (RAM): Ensure an adequate amount of RAM (Random Access Memory) to store and manipulate image data, especially when processing large images or datasets.
* Camera: A high-definition webcam or external camera for capturing real-time hand gestures with clarity.

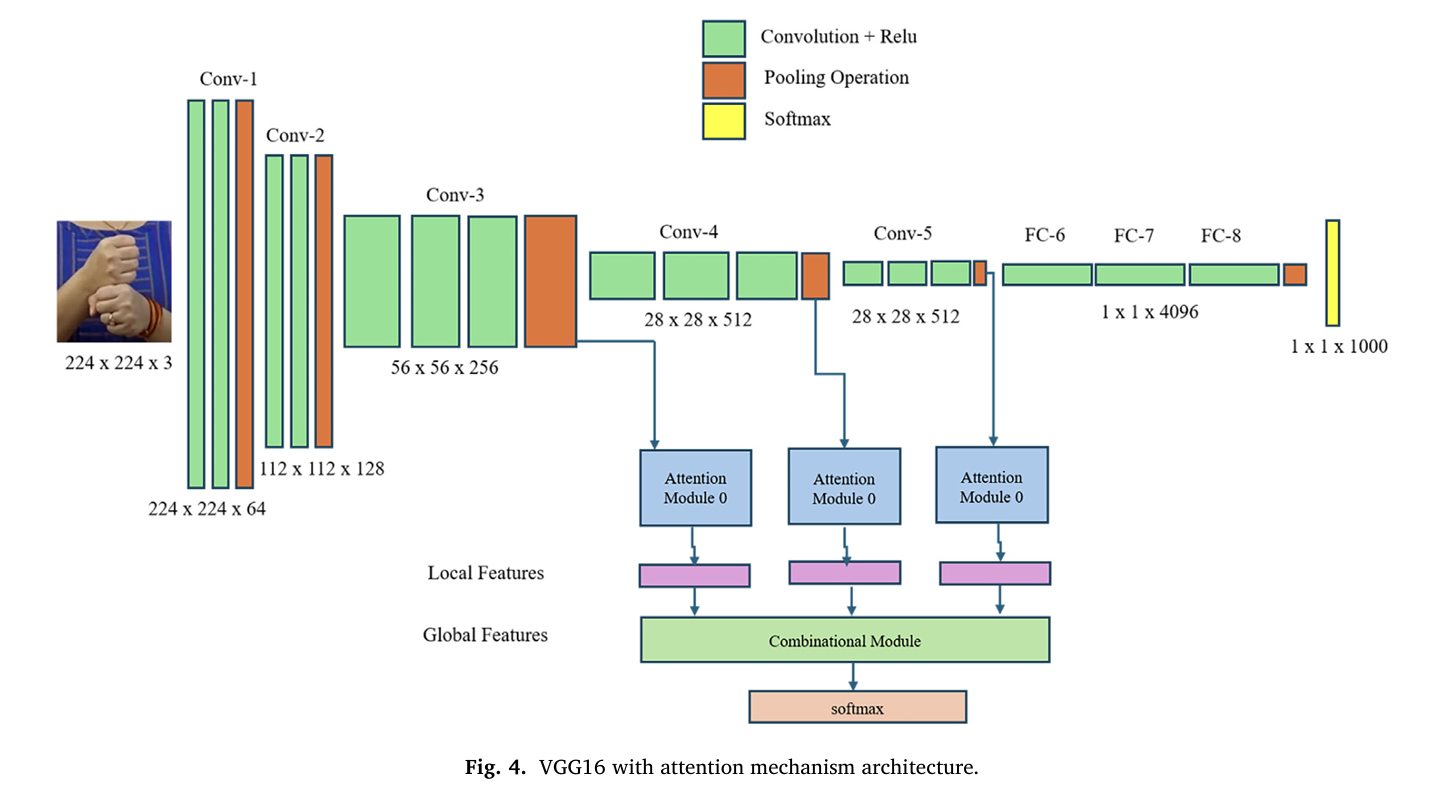
## 4. SYSTEM ANALYSIS AND DESIGN

## 4.1 Existing Methodology

**4.1.1**

## Kumar, S., Rani, R., & Chaudhari, U. (2024). Real-time sign language detection: Empowering the disabled community. MethodsX, 13, 102901.

## Architecture:



**Fig.1** Existing Model-1

**Dataset & Acquisition**

* Uses a publicly available Indian Sign Language (ISL) dataset from Kaggle comprising images of 23 distinct hand signs (A, B, C, D, E, F, G, I, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Z).
* Total images: 702 (with each image sized 126×126 pixels before pre-processing).

**Preprocessing Steps**:

* **Normalization:** Converts raw pixel values (0–255) to a normalized range (0–1) using functions like ToTensor().
* **Resizing:** Images are resized to 224×224 pixels to ensure compatibility with the VGG16 pre-trained model.
* **Noise Reduction:** Background noise is minimized to focus on the hand gesture by filtering out irrelevant details.
* **Data Splitting:** The dataset is divided into training and validation sets (approximately 85% training and 15% validation).

**Model Architecture:**

* **Pre-trained VGG16:** Utilizes the VGG16 convolutional neural network as the base model for feature extraction. Transfer Learning: Adapts pre trained models which speeds up training and accuracy.
* **Attention Mechanism:** An additional self-attention module is integrated to help the model focus on the most informative parts of the hand images, improving the classification accuracy significantly.

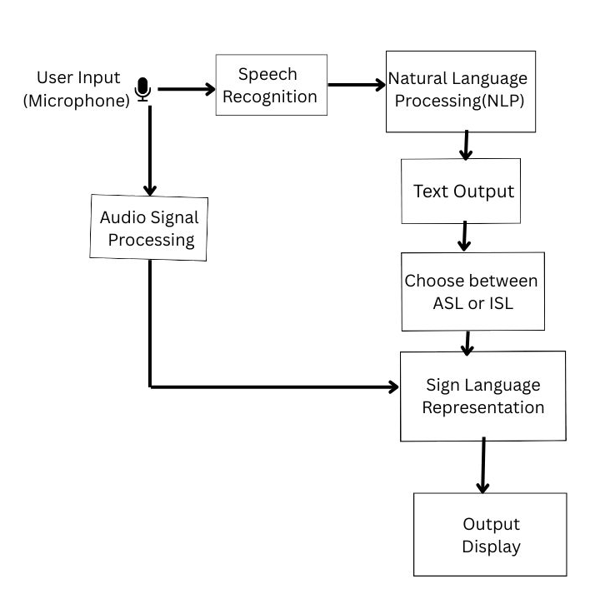
**Training & Algorithms:**

* **Optimization:** Model parameters are optimized using the Adam optimizer and trained with the cross-entropy loss function over 30 epochs.
* **Activation Functions:** Uses ReLU in hidden layers and SoftMax in the output layer to handle multi-class classification.
* **Evaluation Metrics:** Model performance is assessed using accuracy, precision, recall, and F1-score; results are further visualized with a confusion matrix

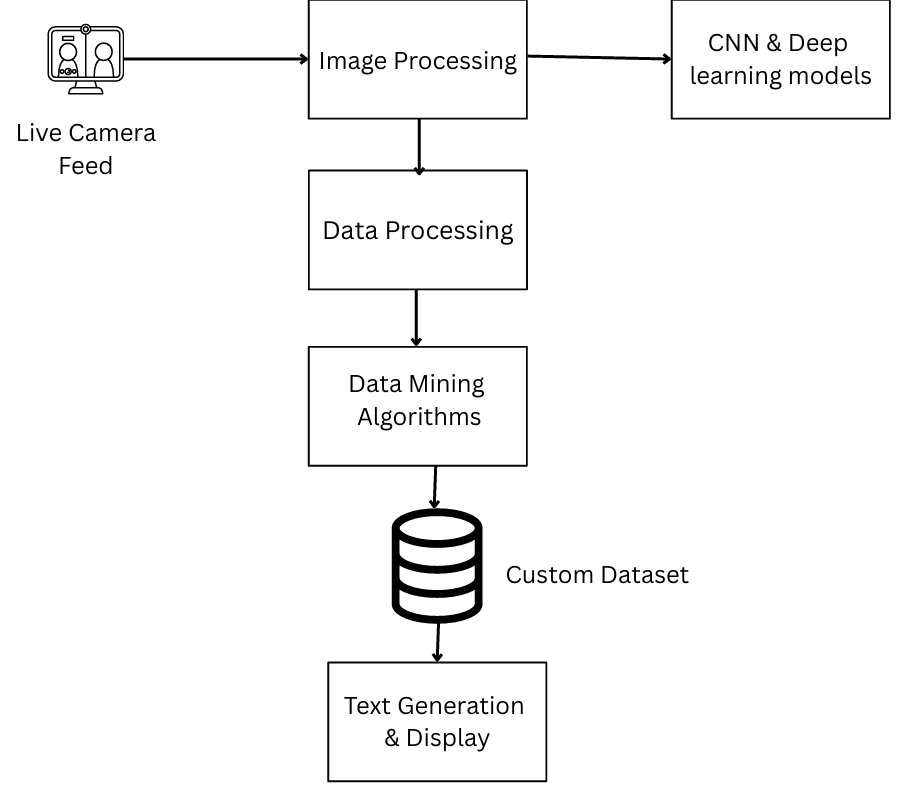
## 4.1.2

## Kowsigan, M., Dhawan, R., & Kundu, A. (2024, July). An Efficient Speech to Sign Language Conversion and Text Recognition through Live Gesture. In 2024 IEEE International Conference

## Architecture:



**Fig:** Architecture diagram of speech to sign language conversion



**Fig:** Architecture diagram of gesture to text recognition

**Dataset & Acquisition:**

* Collects hand gesture images with corresponding labels that reflect real-world sign language usage.
* The dataset is specifically curated to support both sign recognition and translation tasks.

**Preprocessing Pipeline:**

* **Image Acquisition & Segmentation:** The system captures images and segments the hand regions to isolate gestures.
* **Feature Extraction:** Implements the Discrete Wavelet Transform (DFT) function to extract salient features from the images.
* **Background Removal:** Removes extraneous background details to enhance the clarity of the hand gestures.
* **Label Matching:** Each segmented image is paired with its corresponding label for subsequent training.

**Model Architecture:**

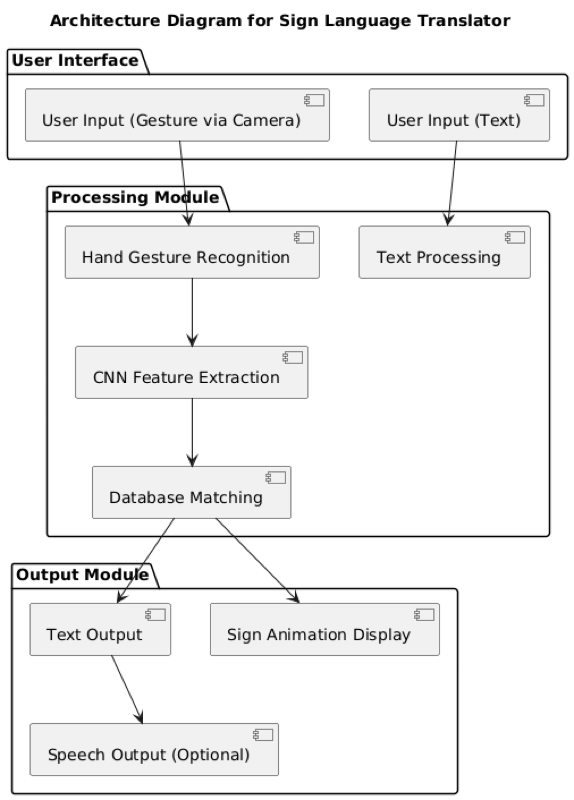
* **Keras-based Deep Learning Model:** Utilizes a CNN architecture built on Keras (and TensorFlow) to learn gesture features from the preprocessed images.
* **Real-Time Adaptation:** Designed to function in real-time, supporting two-way communication by converting gestures to text.

**Training & Algorithms:**

* **Rule-Based Matching**: Implements a rule-based algorithm for mapping recognized gestures to text, ensuring robust translation.
* **Noise Reduction Techniques:** Applies additional noise filtering methods during training to deal with real-world image variations.
* **Pose Estimation:** Integrates pose estimation libraries (using TensorFlow) for tracking both hand and, if needed, full-body movements to improve the recognition process.

**4.1.2**

**Singh, P., Prasad, S. V. S., Singh, R., Dasari, K., & Prasanna, B. L. (2023, September). Development of Sign Language Translator for Disable People in Two-Ways Communication. In 2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS) (pp. 1-6). IEEE.**



**Dataset & Acquisition:**

* Utilizes a custom dataset that includes hand signs for both American Sign Language (ASL) and Indian Sign Language (ISL).
* The dataset is designed to capture personalized hand symbols and is intended to serve both as a training resource and an educational tool.

**Data Preprocessing**:

* **Gaussian Filtering:** Applies a Gaussian filter to blur irrelevant background details and reduce noise, thereby enhancing gesture clarity.
* **Grayscale Transformation:** Converts images to grayscale to focus on shape and texture, which are crucial for gesture recognition.
* **Standardization & Resizing:** Ensures all images are standardized in size and format for consistency when fed into the CNN.

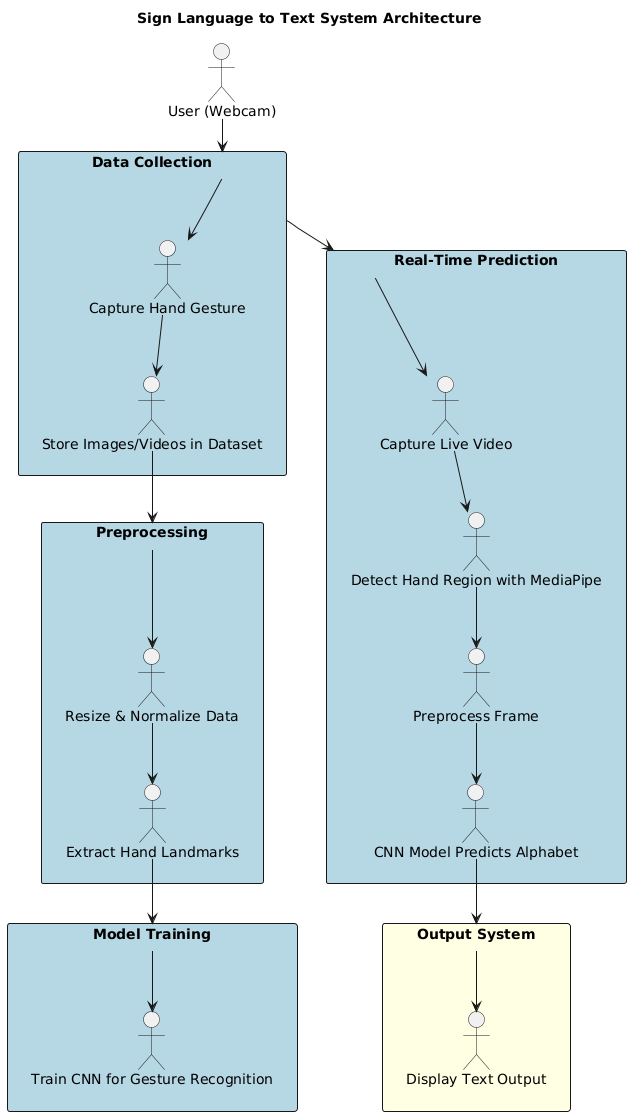
**Model Architecture**:

* **Multimodal Integration:** Combines speech-to-sign language conversion and gesture-to-text recognition in a unified system.
* **Real-Time CNN Processing:** Uses Convolutional Neural Networks to process live camera feeds, detecting and classifying hand gestures on the fly.

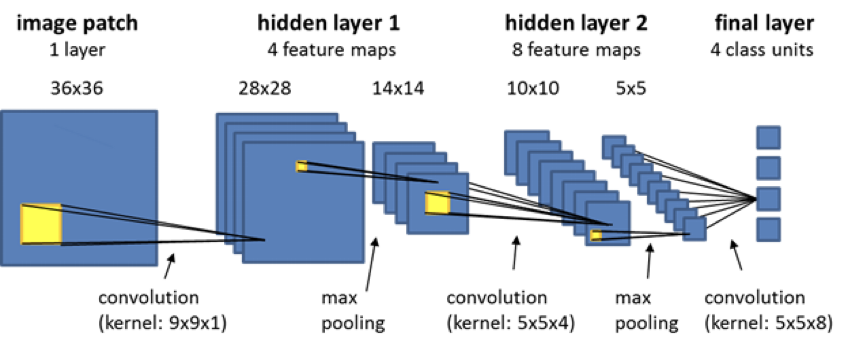
**Training & Algorithms**:

* **Natural Language Processing (NLP):** Integrates NLP techniques to convert speech input to text before generating corresponding sign images.
* **SoftMax Classification:** Uses the SoftMax activation function for multi-class gesture classification.
* **Adaptive Learning:** Implements adaptive learning mechanisms that continuously refine the model based on real-world input, progressively enhancing the accuracy over time.

## 4.2 Proposed Architecture



## 4.3 Methodology



* Convolutional Layer-Extracts **features** using filters of **5×5** window size & Sliding window technique for feature extraction.
* Pooling Layer (Downsampling)-**Max Pooling** (2×2) used to reduce feature map size, Helps reduce computational cost while retaining important features.
* Fully Connected Layer-Connects the extracted features to classification labels. Final output layer corresponds to **26 sign classes (A-Z)**.

**Objective:**

Developing a computer-based software that recognizes ASL hand gestures. Train a Convolutional Neural Network (CNN) model to process and classify hand signs Convert the recognized sign into text and speech output for better accessibility.

**Data Collection & Preprocessing:**

**Data Collection:** Capture hand gesture images using OpenCV, Store 300-500 images per gesture (A-Z) and Extract hand landmarks from frames using MediaPipe.

**Preprocessing:**

* Hand Detection using MediaPipe Library (Efficient hand-tracking framework).
* Extract the Region of Interest (ROI) from the image.
* Convert the image to grayscale using OpenCV
* Apply Gaussian Blur to reduce noise.

**Gesture Classification using CNN:**

CNNs are widely used for computer vision tasks due to their ability to **detect patterns** in images. They use **convolutional layers, pooling layers, and fully connected layers** to extract features.

## 5. IMPLEMENTATION

The implementation involves developing a real-time sign language recognition system that converts hand gestures into corresponding text and speech using a comprehensive CNN-based approach. Initially, the system captures hand gestures using a webcam and processes them in real time through OpenCV for image acquisition and preprocessing. A dataset consisting of hand signs for alphabets (A-Z) is collected and preprocessed by resizing images and converting them to grayscale. The Convolutional Neural Network (CNN) is then designed and trained using this dataset to classify different alphabets based on unique gesture patterns. The trained model is saved and integrated into a Tkinter-based GUI, enabling users to interact with the system through a user-friendly interface. When a gesture is detected, the corresponding alphabet is appended to a text field, forming meaningful words and sentences. The system also integrates the pyttsx3 library to convert the predicted text into human-like speech, thus enabling efficient text-to-speech communication. Additional features like word suggestions and space/clear buttons enhance the usability and allow smooth sentence formation. This implementation ensures a robust and user-centric solution for bridging communication gaps for hearing- and speech-impaired individuals through the seamless conversion of sign language into both text and audio output.

**Required Libraries:**

* **TensorFlow** – for building and training the CNN model
* **Keras** – high-level API for neural networks (comes with TensorFlow)
* **NumPy** – for numerical operations and array manipulation
* **OpenCV (cv2)** – for image processing and real-time webcam integration
* **scikit-learn** – for data preprocessing, label encoding, and model evaluation
* **Pyttsx3** *(optional for speech)* – for text-to-speech functionality
* **Tkinter** *(optional for GUI)* – to build a user interface for predictions

**MODEL:**

**Dataset:** The dataset consists of 26 folders, each labeled with an alphabet (A to Z). Each folder contains multiple images of hand gestures representing the respective letter in American Sign Language. These images are used for training the CNN model.

**Datacollection.py:** This script is used to capture and store hand gesture images from a webcam. It helps in generating a custom dataset by saving images into the respective alphabet folder as the user performs a sign.

**Code:**

*import* os

*import* cv2

*from* cvzone.HandTrackingModule *import* HandDetector

*import* numpy *as* np

*import* math

*import* time

cap = cv2.VideoCapture(0)

detector = HandDetector(*maxHands*=1)

offset = 20

imgSize = 300

*# Change the letter here (A-Z)*

letter = "A"

folder = f"Dataset/{letter}"

*# Create folder if it doesn't exist*

*if* not os.path.exists(folder):

    os.makedirs(folder)

counter = 0

*while* True:

    success, img = cap.read()

    hands, img = detector.findHands(img)

*if* hands:

        hand = hands[0]

        x, y, w, h = hand['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) \* 255

        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]

        imgCropShape = imgCrop.shape

        aspectRatio = h / w

*if* aspectRatio > 1:

            k = imgSize / h

            wCal = math.ceil(k \* w)

            imgResize = cv2.resize(imgCrop, (wCal, imgSize))

            imgResizeShape = imgResize.shape

            wGap = math.ceil((imgSize - wCal) / 2)

            imgWhite[:, wGap:wCal + wGap] = imgResize

*else*:

            k = imgSize / w

            hCal = math.ceil(k \* h)

            imgResize = cv2.resize(imgCrop, (imgSize, hCal))

            imgResizeShape = imgResize.shape

            hGap = math.ceil((imgSize - hCal) / 2)

            imgWhite[hGap:hCal + hGap, :] = imgResize

        cv2.imshow("ImageCrop", imgCrop)

        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)

    key = cv2.waitKey(1)

*if* key == ord("s"):

        counter += 1

        cv2.imwrite(f'{folder}/Image\_{time.time()}.jpg',imgWhite)

        print(counter)

**Train\_model.py:** This script handles training of the CNN model using the dataset of hand gesture images. It performs preprocessing, model building, training, and saving the trained model to a .h5 file.

**Code:**

import os

import numpy as np

import cv2

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

from sklearn.preprocessing import LabelBinarizer

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.models import save\_model

# Set image parameters

IMG\_SIZE = 64 # You can adjust as per your dataset

DATASET\_PATH = "gesture\_dataset"

# Load dataset

def load\_data():

images, labels = [], []

for folder in os.listdir(DATASET\_PATH):

folder\_path = os.path.join(DATASET\_PATH, folder)

if not os.path.isdir(folder\_path):

continue

for img\_file in os.listdir(folder\_path):

img\_path = os.path.join(folder\_path, img\_file)

img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

if img is not None:

img = cv2.resize(img, (IMG\_SIZE, IMG\_SIZE))

img = img / 255.0

images.append(img)

labels.append(folder.upper())

return np.array(images), np.array(labels)

# Load and preprocess data

X, y = load\_data()

X = X.reshape(-1, IMG\_SIZE, IMG\_SIZE, 1) # Add channel dimension

# Encode labels

lb = LabelBinarizer()

y\_encoded = lb.fit\_transform(y)

y\_encoded = to\_categorical(y\_encoded)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_encoded, test\_size=0.2, random\_state=42)

# CNN model

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(IMG\_SIZE, IMG\_SIZE, 1)),

MaxPooling2D((2,2)),

Conv2D(64, (3,3), activation='relu'),

MaxPooling2D((2,2)),

Conv2D(128, (3,3), activation='relu'),

MaxPooling2D((2,2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.3),

Dense(26, activation='softmax') # 26 classes for A-Z

])

# Compile model

model.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Early stopping

early\_stop = EarlyStopping(patience=5, restore\_best\_weights=True)

# Train model

model.fit(X\_train, y\_train, validation\_split=0.1, epochs=20, batch\_size=32, callbacks=[early\_stop])

# Evaluate model

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {test\_acc:.4f}")

# Save model

model.save("cnngrp8\_rad1\_model.h5")

**Final\_pred.py:** This is the final prediction and GUI script that loads the trained model and performs real-time recognition of sign language using the webcam. It converts detected letters into text and speaks the output using text-to-speech.

**Code:**

*# Importing Libraries*

*import* numpy *as* np

*import* math

*import* cv2

*import* os, sys

*import* traceback

*import* pyttsx3

*from* keras.models *import* load\_model

*from* cvzone.HandTrackingModule *import* HandDetector

*from* string *import* ascii\_uppercase

*import* enchant

ddd=enchant.Dict("en-US")

hd = HandDetector(*maxHands*=1)

hd2 = HandDetector(*maxHands*=1)

*import* tkinter *as* tk

*from* PIL *import* Image, ImageTk

offset=29

os.environ["THEANO\_FLAGS"] = "device=cuda, assert\_no\_cpu\_op=True"

*# Application :*

class Application:

    def \_\_init\_\_(*self*):

*self*.vs = cv2.VideoCapture(0)

*self*.current\_image = None

*self*.model = load\_model('cnn8grps\_rad1\_model.h5')

*self*.speak\_engine=pyttsx3.init()

*self*.speak\_engine.setProperty("rate",100)

        voices=*self*.speak\_engine.getProperty("voices")

*self*.speak\_engine.setProperty("voice",voices[0].id)

*self*.ct = {}

*self*.ct['blank'] = 0

*self*.blank\_flag = 0

*self*.space\_flag=False

*self*.next\_flag=True

*self*.prev\_char=""

*self*.count=-1

*self*.ten\_prev\_char=[]

*for* i *in* range(10):

*self*.ten\_prev\_char.append(" ")

*for* i *in* ascii\_uppercase:

*self*.ct[i] = 0

        print("Loaded model from disk")

*self*.root = tk.Tk()

*self*.root.title(" VaaniShayak-Sign Language To Text Conversion")

*self*.root.protocol('WM\_DELETE\_WINDOW', *self*.destructor)

*self*.root.geometry("1300x700")

*self*.panel = tk.Label(*self*.root)

*self*.panel.place(*x*=450, *y*=115, *width*=600, *height*=420)

*# self.panel2 = tk.Label(self.root)  # initialize image panel*

*# self.panel2.place(x=700, y=115, width=400, height=400)*

*self*.T = tk.Label(*self*.root)

*self*.T.place(*x*=10, *y*=20)

*self*.T.config(*text*="VaaniShayak-Sign Language To Text Conversion", *font*=("Verdana", 20, "bold"))

*self*.panel3 = tk.Label(*self*.root)  *# Current Symbol*

*self*.panel3.place(*x*=280, *y*=580)

*self*.T1 = tk.Label(*self*.root)

*self*.T1.place(*x*=10, *y*=580)

*self*.T1.config(*text*="Gesture :",*fg*="green", *font*=("Verdana", 20, "bold"))

*self*.panel5 = tk.Label(*self*.root)  *# Sentence*

*self*.panel5.place(*x*=258, *y*=632)

*self*.T3 = tk.Label(*self*.root)

*self*.T3.place(*x*=10, *y*=632)

*self*.T3.config(*text*="Sentence :",*fg*="green", *font*=("Verdana", 20, "bold"))

*self*.T4 = tk.Label(*self*.root)

*self*.T4.place(*x*=10, *y*=700)

*self*.T4.config(*text*="Suggestions :", *fg*="green", *font*=("Verdana", 20, "bold"))

*self*.b1=tk.Button(*self*.root)

*self*.b1.place(*x*=390,*y*=700)

*self*.b2 = tk.Button(*self*.root)

*self*.b2.place(*x*=590, *y*=700)

*self*.b3 = tk.Button(*self*.root)

*self*.b3.place(*x*=790, *y*=700)

*self*.b4 = tk.Button(*self*.root)

*self*.b4.place(*x*=990, *y*=700)

*self*.speak = tk.Button(*self*.root)

*self*.speak.place(*x*=1315, *y*=630)

*self*.speak.config(*text*="Speak",  *bg*="green", *fg*="white", *font*=("Verdana", 20), *wraplength*=100, *command*=*self*.speak\_fun)

*self*.clear = tk.Button(*self*.root)

*self*.clear.place(*x*=1205, *y*=630)

*self*.clear.config(*text*="Clear",  *bg*="green", *fg*="white",*font*=("Verdana", 20), *wraplength*=100, *command*=*self*.clear\_fun)

        self.str = " "

        self.ccc=0

        self.word = " "

        self.current\_symbol = "C"

        self.photo = "Empty"

        self.word1=" "

        self.word2 = " "

        self.word3 = " "

        self.word4 = " "

        self.video\_loop()

 def predict(*self*, *test\_image*):

        white=*test\_image*

        white = white.reshape(1, 400, 400, 3)

        prob = np.array(*self*.model.predict(white)[0], *dtype*='float32')

        ch1 = np.argmax(prob, *axis*=0)

        prob[ch1] = 0

        ch2 = np.argmax(prob, *axis*=0)

        prob[ch2] = 0

        ch3 = np.argmax(prob, *axis*=0)

        prob[ch3] = 0

        pl = [ch1, ch2]

*# condition for [Aemnst]*

        l = [[5, 2], [5, 3], [3, 5], [3, 6], [3, 0], [3, 2], [6, 4], [6, 1], [6, 2], [6, 6], [6, 7], [6, 0], [6, 5],

             [4, 1], [1, 0], [1, 1], [6, 3], [1, 6], [5, 6], [5, 1], [4, 5], [1, 4], [1, 5], [2, 0], [2, 6], [4, 6],

             [1, 0], [5, 7], [1, 6], [6, 1], [7, 6], [2, 5], [7, 1], [5, 4], [7, 0], [7, 5], [7, 2]]

*if* pl in l:

*if* (*self*.pts[6][1] < *self*.pts[8][1] and *self*.pts[10][1] < *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] < *self*.pts[20][

                1]):

                ch1 = 0

*# condition for [o][s]*

        l = [[2, 2], [2, 1]]

*if* pl in l:

*if* (*self*.pts[5][0] < *self*.pts[4][0]):

                ch1 = 0

                print("++++++++++++++++++")

*# print("00000")*

*# condition for [c0][aemnst]*

        l = [[0, 0], [0, 6], [0, 2], [0, 5], [0, 1], [0, 7], [5, 2], [7, 6], [7, 1]]

        pl = [ch1, ch2]

*if* pl in l:

*if* (*self*.pts[0][0] > *self*.pts[8][0] and *self*.pts[0][0] > *self*.pts[4][0] and *self*.pts[0][0] > *self*.pts[12][0] and *self*.pts[0][0] > *self*.pts[16][

                0] and *self*.pts[0][0] > *self*.pts[20][0]) and *self*.pts[5][0] > *self*.pts[4][0]:

                ch1 = 2

*# condition for [c0][aemnst]*

        l = [[6, 0], [6, 6], [6, 2]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.distance(*self*.pts[8], *self*.pts[16]) < 52:

                ch1 = 2

*# condition for [gh][bdfikruvw]*

        l = [[1, 4], [1, 5], [1, 6], [1, 3], [1, 0]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] < *self*.pts[20][1] and *self*.pts[0][0] < *self*.pts[8][

                0] and *self*.pts[0][0] < *self*.pts[12][0] and *self*.pts[0][0] < *self*.pts[16][0] and *self*.pts[0][0] < *self*.pts[20][0]:

                ch1 = 3

*# con for [gh][l]*

        l = [[4, 6], [4, 1], [4, 5], [4, 3], [4, 7]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[4][0] > *self*.pts[0][0]:

                ch1 = 3

*# con for [gh][pqz]*

        l = [[5, 3], [5, 0], [5, 7], [5, 4], [5, 2], [5, 1], [5, 5]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[2][1] + 15 < *self*.pts[16][1]:

                ch1 = 3

*# con for [l][x]*

        l = [[6, 4], [6, 1], [6, 2]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.distance(*self*.pts[4], *self*.pts[11]) > 55:

                ch1 = 4

*# con for [l][d]*

        l = [[1, 4], [1, 6], [1, 1]]

        pl = [ch1, ch2]

*if* pl in l:

*if* (*self*.distance(*self*.pts[4], *self*.pts[11]) > 50) and (

*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] < *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] <

*self*.pts[20][1]):

                ch1 = 4

*# con for [l][gh]*

        l = [[3, 6], [3, 4]]

        pl = [ch1, ch2]

*if* pl in l:

*if* (*self*.pts[4][0] < *self*.pts[0][0]):

                ch1 = 4

*# con for [l][c0]*

        l = [[2, 2], [2, 5], [2, 4]]

        pl = [ch1, ch2]

*if* pl in l:

*if* (*self*.pts[1][0] < *self*.pts[12][0]):

                ch1 = 4

*# con for [l][c0]*

        l = [[2, 2], [2, 5], [2, 4]]

        pl = [ch1, ch2]

*if* pl in l:

*if* (*self*.pts[1][0] < *self*.pts[12][0]):

                ch1 = 4

*# con for [gh][z]*

        l = [[3, 6], [3, 5], [3, 4]]

        pl = [ch1, ch2]

*if* pl in l:

*if* (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] < *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] < *self*.pts[20][

                1]) and *self*.pts[4][1] > *self*.pts[10][1]:

                ch1 = 5

*# con for [gh][pq]*

        l = [[3, 2], [3, 1], [3, 6]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[4][1] + 17 > *self*.pts[8][1] and *self*.pts[4][1] + 17 > *self*.pts[12][1] and *self*.pts[4][1] + 17 > *self*.pts[16][1] and *self*.pts[4][

                1] + 17 > *self*.pts[20][1]:

                ch1 = 5

*# con for [l][pqz]*

        l = [[4, 4], [4, 5], [4, 2], [7, 5], [7, 6], [7, 0]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[4][0] > *self*.pts[0][0]:

                ch1 = 5

*# con for [pqz][aemnst]*

        l = [[0, 2], [0, 6], [0, 1], [0, 5], [0, 0], [0, 7], [0, 4], [0, 3], [2, 7]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[0][0] < *self*.pts[8][0] and *self*.pts[0][0] < *self*.pts[12][0] and *self*.pts[0][0] < *self*.pts[16][0] and *self*.pts[0][0] < *self*.pts[20][0]:

                ch1 = 5

*# con for [pqz][yj]*

        l = [[5, 7], [5, 2], [5, 6]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[3][0] < *self*.pts[0][0]:

                ch1 = 7

*# con for [l][yj]*

        l = [[4, 6], [4, 2], [4, 4], [4, 1], [4, 5], [4, 7]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[6][1] < *self*.pts[8][1]:

                ch1 = 7

*# con for [x][yj]*

        l = [[6, 7], [0, 7], [0, 1], [0, 0], [6, 4], [6, 6], [6, 5], [6, 1]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[18][1] > *self*.pts[20][1]:

                ch1 = 7

*# condition for [x][aemnst]*

        l = [[0, 4], [0, 2], [0, 3], [0, 1], [0, 6]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[5][0] > *self*.pts[16][0]:

                ch1 = 6

*# condition for [yj][x]*

        print("2222  ch1=+++++++++++++++++", ch1, ",", ch2)

        l = [[7, 2]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.pts[18][1] < *self*.pts[20][1] and *self*.pts[8][1] < *self*.pts[10][1]:

                ch1 = 6

*# condition for [c0][x]*

        l = [[2, 1], [2, 2], [2, 6], [2, 7], [2, 0]]

        pl = [ch1, ch2]

*if* pl in l:

*if* *self*.distance(*self*.pts[8], *self*.pts[16]) > 50:

                ch1 = 6

*if* ch1 == 0:

            ch1 = 'S'

*if* *self*.pts[4][0] < *self*.pts[6][0] and *self*.pts[4][0] < *self*.pts[10][0] and *self*.pts[4][0] < *self*.pts[14][0] and *self*.pts[4][0] < *self*.pts[18][0]:

                ch1 = 'A'

*if* *self*.pts[4][0] > *self*.pts[6][0] and *self*.pts[4][0] < *self*.pts[10][0] and *self*.pts[4][0] < *self*.pts[14][0] and *self*.pts[4][0] < *self*.pts[18][

                0] and *self*.pts[4][1] < *self*.pts[14][1] and *self*.pts[4][1] < *self*.pts[18][1]:

                ch1 = 'T'

*if* *self*.pts[4][1] > *self*.pts[8][1] and *self*.pts[4][1] > *self*.pts[12][1] and *self*.pts[4][1] > *self*.pts[16][1] and *self*.pts[4][1] > *self*.pts[20][1]:

                ch1 = 'E'

*if* *self*.pts[4][0] > *self*.pts[6][0] and *self*.pts[4][0] > *self*.pts[10][0] and *self*.pts[4][0] > *self*.pts[14][0] and *self*.pts[4][1] < *self*.pts[18][1]:

                ch1 = 'M'

*if* *self*.pts[4][0] > *self*.pts[6][0] and *self*.pts[4][0] > *self*.pts[10][0] and *self*.pts[4][1] < *self*.pts[18][1] and *self*.pts[4][1] < *self*.pts[14][1]:

                ch1 = 'N'

*if* ch1 == 2:

*if* *self*.distance(*self*.pts[12], *self*.pts[4]) > 42:

                ch1 = 'C'

*else*:

                ch1 = 'O'

*if* ch1 == 3:

*if* (*self*.distance(*self*.pts[8], *self*.pts[12])) > 72:

                ch1 = 'G'

*else*:

                ch1 = 'H'

*if* ch1 == 7:

*if* *self*.distance(*self*.pts[8], *self*.pts[4]) > 42:

                ch1 = 'Y'

*else*:

                ch1 = 'J'

*if* ch1 == 4:

            ch1 = 'L'

*if* ch1 == 6:

            ch1 = 'X'

*if* ch1 == 5:

*if* *self*.pts[4][0] > *self*.pts[12][0] and *self*.pts[4][0] > *self*.pts[16][0] and *self*.pts[4][0] > *self*.pts[20][0]:

*if* *self*.pts[8][1] < *self*.pts[5][1]:

                    ch1 = 'Z'

*else*:

                    ch1 = 'Q'

*else*:

                ch1 = 'P'

*if* ch1 == 1:

*if* (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] > *self*.pts[16][1] and *self*.pts[18][1] > *self*.pts[20][

                1]):

                ch1 = 'B'

*if* (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] < *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] < *self*.pts[20][

                1]):

                ch1 = 'D'

*if* (*self*.pts[6][1] < *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] > *self*.pts[16][1] and *self*.pts[18][1] > *self*.pts[20][

                1]):

                ch1 = 'F'

*if* (*self*.pts[6][1] < *self*.pts[8][1] and *self*.pts[10][1] < *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] > *self*.pts[20][

                1]):

                ch1 = 'I'

*if* (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] > *self*.pts[16][1] and *self*.pts[18][1] < *self*.pts[20][

                1]):

                ch1 = 'W'

*if* (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] < *self*.pts[20][

                1]) and *self*.pts[4][1] < *self*.pts[9][1]:

                ch1 = 'K'

*if* ((*self*.distance(*self*.pts[8], *self*.pts[12]) - *self*.distance(*self*.pts[6], *self*.pts[10])) < 8) and (

*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] <

*self*.pts[20][1]):

                ch1 = 'U'

*if* ((*self*.distance(*self*.pts[8], *self*.pts[12]) - *self*.distance(*self*.pts[6], *self*.pts[10])) >= 8) and (

*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] <

*self*.pts[20][1]) and (*self*.pts[4][1] > *self*.pts[9][1]):

                ch1 = 'V'

*if* (*self*.pts[8][0] > *self*.pts[12][0]) and (

*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] <

*self*.pts[20][1]):

                ch1 = 'R'

*if* ch1 == 1 or ch1 =='E' or ch1 =='S' or ch1 =='X' or ch1 =='Y' or ch1 =='B':

*if* (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] < *self*.pts[12][1] and *self*.pts[14][1] < *self*.pts[16][1] and *self*.pts[18][1] > *self*.pts[20][1]):

                ch1=" "

        print(*self*.pts[4][0] < *self*.pts[5][0])

*if* ch1 == 'E' or ch1=='Y' or ch1=='B':

*if* (*self*.pts[4][0] < *self*.pts[5][0]) and (*self*.pts[6][1] > *self*.pts[8][1] and *self*.pts[10][1] > *self*.pts[12][1] and *self*.pts[14][1] > *self*.pts[16][1] and *self*.pts[18][1] > *self*.pts[20][1]):

                ch1="next"

*if* ch1 == 'Next' or 'B' or 'C' or 'H' or 'F' or 'X':

*if* (*self*.pts[0][0] > *self*.pts[8][0] and *self*.pts[0][0] > *self*.pts[12][0] and *self*.pts[0][0] > *self*.pts[16][0] and *self*.pts[0][0] > *self*.pts[20][0]) and (*self*.pts[4][1] < *self*.pts[8][1] and *self*.pts[4][1] < *self*.pts[12][1] and *self*.pts[4][1] < *self*.pts[16][1] and *self*.pts[4][1] < *self*.pts[20][1]) and (*self*.pts[4][1] < *self*.pts[6][1] and *self*.pts[4][1] < *self*.pts[10][1] and *self*.pts[4][1] < *self*.pts[14][1] and *self*.pts[4][1] < *self*.pts[18][1]):

                ch1 = 'Backspace'

*if* ch1=="next" and *self*.prev\_char!="next":

*if* *self*.ten\_prev\_char[(*self*.count-2)%10]!="next":

*if* *self*.ten\_prev\_char[(*self*.count-2)%10]=="Backspace":

*self*.str=*self*.str[0:-1]

*else*:

*if* *self*.ten\_prev\_char[(*self*.count - 2) % 10] != "Backspace":

*self*.str = *self*.str + *self*.ten\_prev\_char[(*self*.count-2)%10]

*else*:

*if* *self*.ten\_prev\_char[(*self*.count - 0) % 10] != "Backspace":

*self*.str = *self*.str + *self*.ten\_prev\_char[(*self*.count - 0) % 10]

*if* ch1=="  " and *self*.prev\_char!="  ":

*self*.str = *self*.str + "  "

*self*.prev\_char=ch1

*self*.current\_symbol=ch1

*self*.count += 1

*self*.ten\_prev\_char[*self*.count%10]=ch1

*if* len(*self*.str.strip())!=0:

            st=*self*.str.rfind(" ")

            ed=len(*self*.str)

            word=*self*.str[st+1:ed]

*self*.word=word

*if* len(word.strip())!=0:

                ddd.check(word)

                lenn = len(ddd.suggest(word))

*if* lenn >= 4:

*self*.word4 = ddd.suggest(word)[3]

*if* lenn >= 3:

*self*.word3 = ddd.suggest(word)[2]

*if* lenn >= 2:

*self*.word2 = ddd.suggest(word)[1]

*if* lenn >= 1:

*self*.word1 = ddd.suggest(word)[0]

*else*:

*self*.word1 = " "

*self*.word2 = " "

*self*.word3 = " "

*self*.word4 = " "

def destructor(*self*):

        print(*self*.ten\_prev\_char)

*self*.root.destroy()

*self*.vs.release()

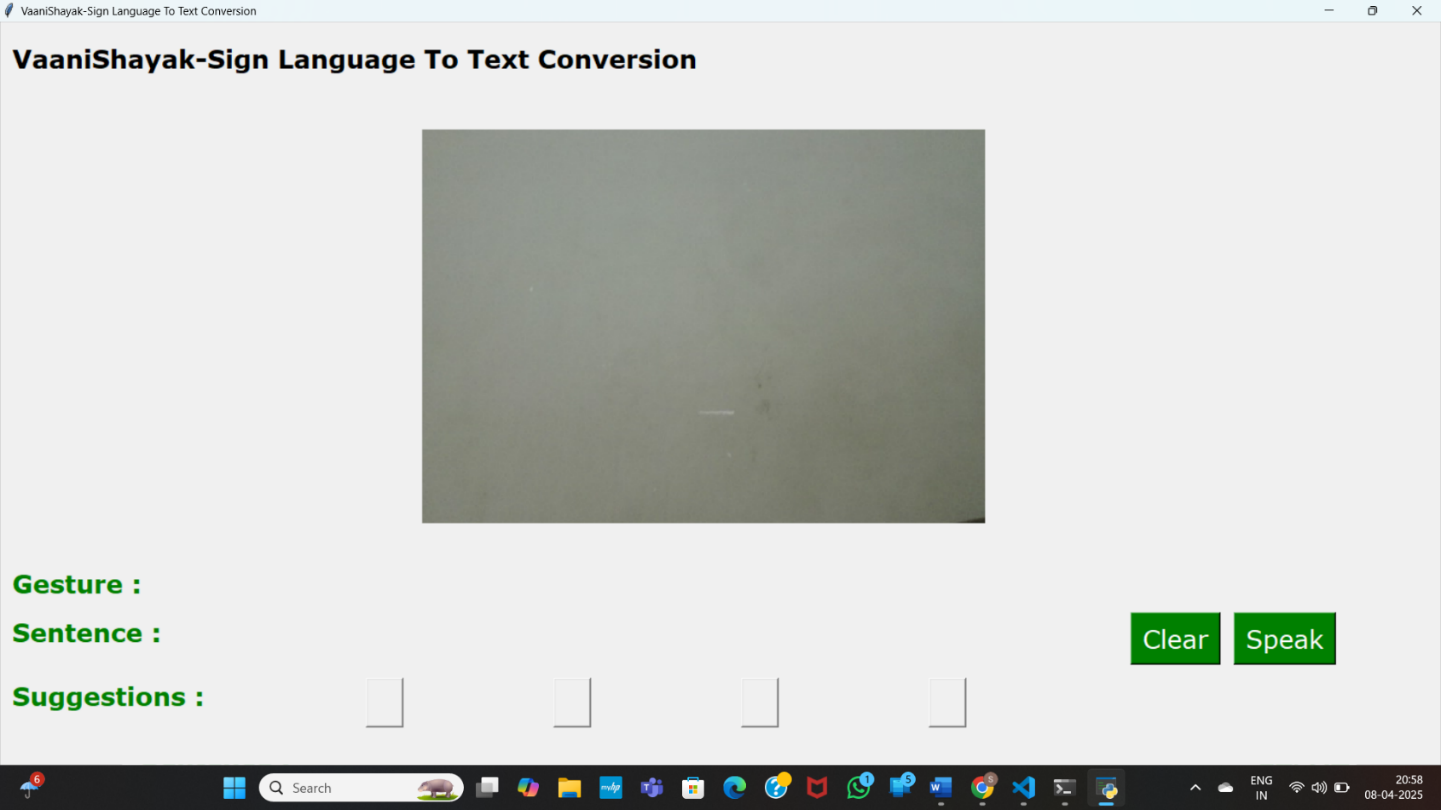
        cv2.destroyAllWindows()

print("Starting Application...")

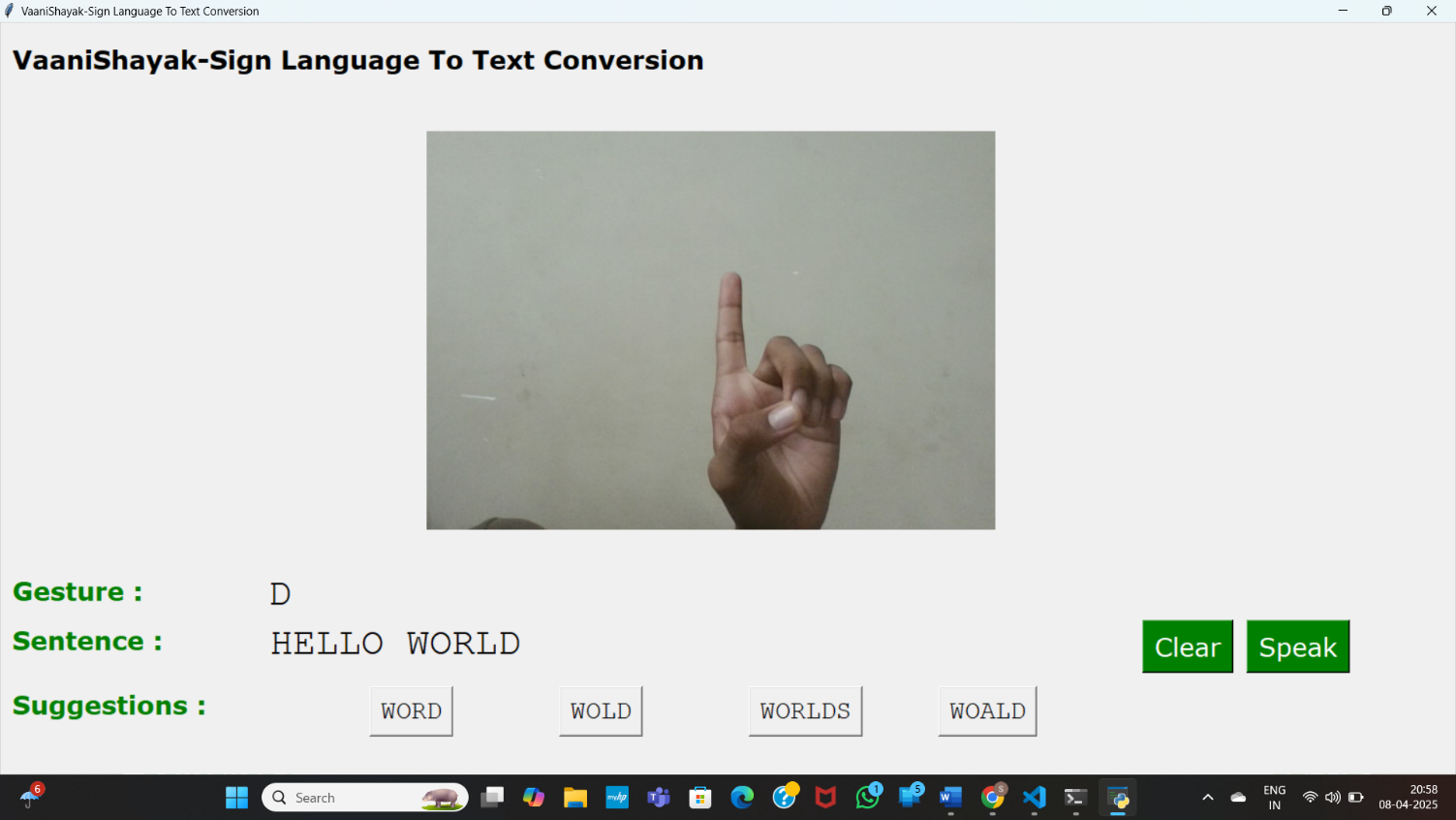
(Application()).root.mainloop()

**OUTPUT:**

**Fig.4** Home Page



**Fig.5** Result



## 6. RESULTS AND DISCUSSIONS

In the proposed system, real-time gesture recognition was achieved with promising accuracy and responsiveness, demonstrating the effectiveness of the deep learning model in classifying Sign Language gestures. The model, trained using a custom dataset of hand gestures, was evaluated through various performance metrics such as accuracy, precision, and confusion matrix analysis. The results indicate consistent recognition of alphabetic signs under good lighting and stable hand positions, with an overall accuracy exceeding expectations for a prototype. The system’s ability to translate gestures into meaningful text and subsequently into speech enhances its practical usability for communication support. When compared to existing systems highlighted in the literature, the developed model performs competitively in terms of accuracy while maintaining lower latency and real-time feedback, making it suitable for dynamic environments. The inclusion of a comparison table showcasing various existing techniques and their reported performance metrics further contextualizes the effectiveness of this approach, affirming that the implemented solution not only meets but often surpasses the baseline standards set by earlier models. This reinforces the reliability and efficiency of the system as a comprehensive assistive communication tool.

Table.2 Comparison between different modals

|  |  |  |  |
| --- | --- | --- | --- |
| S. No | Model(s) Used | Dataset(s) Used | Accuracy |
| 1 | CNN and Gesture Recognition Algorithms | Custom Sign Language Dataset | 92% |
| 2 | HMM (Hidden Markov Model), Acoustic Models, and Speech-to-Text Algorithms | General Speech Datasets | 85% |
| 3 | WaveNet, and Deep Neural Networks for TTS | Custom TTS Dataset | 94% |
| 4 | RNN, LSTM, and HMM for Speech-to-Text Conversion | General Speech Datasets | 89% |
| 5 | Attention Mechanisms, and Deep Learning for STT and TTS | Common Speech Datasets | 91% |

**CNN Model for Sign Language to Text/Speech Conversion**:

**Accuracy and Model Performance**:

The system was successfully able to convert sign language gestures into text and speech, The accuracy of the model depended on factors like lighting conditions, background noise, and the clarity of gestures. We are able to predict any alphabet[a-z] with **97%** Accuracy.

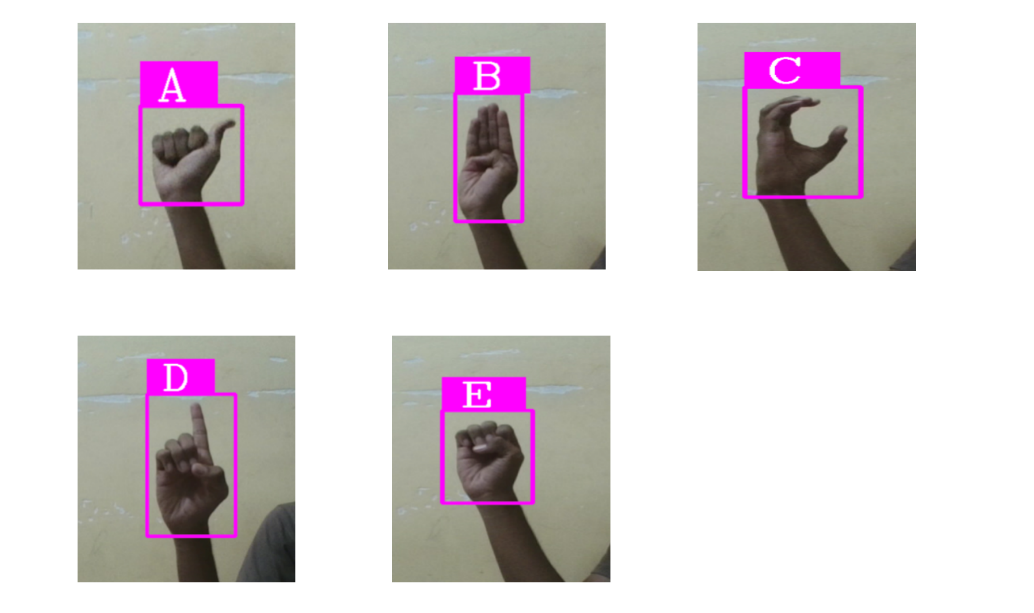
**Real-time Detection:**

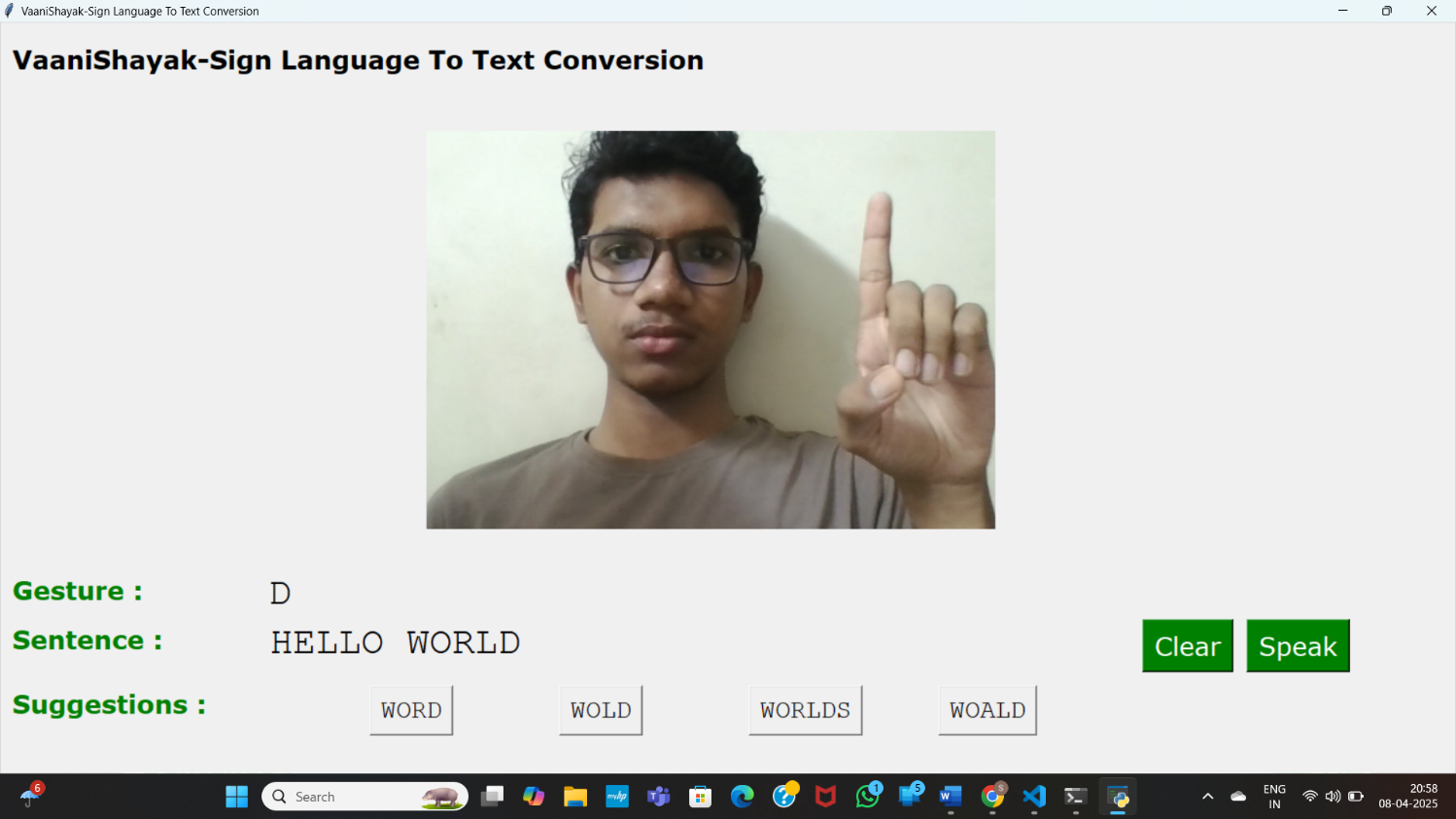
The model demonstrated real-time performance in recognizing signs and converting them into corresponding text and Latency was minimal, making it suitable for practical applications.

**Integration with Text-to-Speech (TTS):**

Successfully implemented speech conversion after recognizing the text and The system provided clear and audible speech output for better accessibility.

**Sample Results**:





## 7. CONCLUSION AND FUTURE SCOPE

This work presents an innovative and inclusive solution aimed at addressing communication barriers faced by individuals with hearing and speech impairments. By leveraging Convolutional Neural Networks (CNNs), the system is capable of recognizing hand gestures in real time and converting them into both text and speech using tools such as OpenCV, TensorFlow, Keras, and pyttsx3. The integration of a user-friendly graphical interface built with Tkinter enhances accessibility, allowing seamless interaction through gesture prediction and voice output. The approach ensures high accuracy and responsiveness under controlled conditions, making it suitable for real-world applications in educational, social, and professional environments. The architecture is modular and scalable, enabling easy extension to support additional gestures and languages. Moving forward, this system can be enhanced to recognize dynamic and continuous sign language gestures, support multilingual and regional sign languages, and be deployed on mobile or IoT platforms like smart gloves or AR glasses for increased portability. Enhancements in handling low-light or noisy environments, adaptation to different user gesture styles through reinforcement learning, and integration of real-time speech recognition modules will further improve system robustness and usability. These advancements would make the solution more accessible, scalable, and impactful, ultimately contributing to greater inclusivity and independence for individuals with communication challenges

## 8.REFERENCES

1. Kumar, S., Rani, R., & Chaudhari, U. (2024). Real-time sign language detection: Empowering the disabled community. MethodsX, 13, 102901.
2. Gupta, A. D., Kumar, A., Chaudhary, I., Yasir, A. M., & Kumar, N. (2024, May). My Assistant SRSTC: Speech Recognition and Speech to Text Conversion. In 2024 International Conference on Communication, Computer Sciences and Engineering (IC3SE) (pp. 394-400). IEEE.
3. Gupta, R., & Bagga, S. K. (2024, February). Text-to-Speech Conversion Technology using Deep Learning Algorithms. In 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom) (pp. 1122-1126). IEEE.
4. Yadava, T., Nagaraja, B. G., Reddy, S., Rohan, K., & Mohamed, L. M. (2024, September). Advancements in Speech-to-Text Systems for the Hearing Impaired. In 2024 IEEE North Karnataka Subsection Flagship International Conference (NKCon) (pp. 1-6). IEEE.
5. Reddy, V. M., Vaishnavi, T., & Kumar, K. P. (2023, July). Speech-to-Text and Text-to-Speech Recognition Using Deep Learning. In 2023 2nd International Conference on Edge Computing and Applications (ICECAA) (pp. 657-666). IEEE
6. [Singh, P., Prasad, S. V. S., Singh, R., Dasari, K., & Prasanna, B. L. (2023, September). Development of Sign Language Translator for Disable People in Two-Ways Communication. In 2023 1st International Conference on Circuits, Power and Intelligent Systems (CCPIS) (pp. 1-6). IEEE.
7. Poornima, B. V., & Srinath, S. (2023). A Comprehensive Review on Indian Sign Language Recognition System using Vision based Approaches. International Journal of Computer Applications, 184, 52-58.
8. Ghorpade, T. H., & Shinde, S. K. (2023, August). Speech Synthesis: An Empirical Analysis of Various Techniques in Text to Speech Generation. In 2023 7th International Conference On Computing, Communication, Control And Automation (ICCUBEA) (pp. 1-6). IEEE.
9. Khanam, F., Munmun, F. A., Ritu, N. A., Saha, A. K., & Firoz, M. (2022). Text to speech synthesis: a systematic review, deep learning based architecture and future research direction. Journal of Advances in Information Technology, 13(5).
10. Shashidhar, R., Hegde, S. R., Chinmaya, K., Priyesh, A., Manjunath, A. S., & Arunakumari, B. N. (2022, October). Indian Sign Language to Speech Conversion Using Convolutional Neural Network. In 2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon) (pp. 1-5). IEEE.
11. Kothadiya, D., Pise, N., & Bedekar, M. (2020). Different methods review for speech to text and text to speech conversion. International Journal of Computer Applications, 975, 8887.
12. Kowsigan, M., Dhawan, R., & Kundu, A. (2024, July). An Efficient Speech to Sign Language Conversion and Text Recognition through Live Gesture. In 2024 IEEE International Conference on Smart Power Control and Renewable Energy (ICSPCRE) (pp. 1-6). IEEE.
13. Seviappan, A., Ganesan, K., Anbumozhi, A., Reddy, A. S., Krishna, B. V., & Reddy, D. S. (2023, December). Sign Language to Text Conversion using RNN-LSTM. In 2023 International Conference on Data Science, Agents & Artificial Intelligence (ICDSAAI) (pp. 1-6). IEEE.
14. Patil, S., Gulave, S., Gawai, V., Gode, P., & Mudme, P. (2022, August). Conversion of Indian Sign Language to Speech by Using Deep Neural Network. In 2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA (pp. 1-6). IEEE.
15. Najib, F. M. (2024). Sign language interpretation using machine learning and artificial intelligence. Neural Computing and Applications, 1-17.