DEVELOPMENT OF A SIGN LANGUAGE DETECTION AND CONVERSION SYSTEM USING MEDIA PIPE FOR MUTE AND DEAF INDIVIDUALS

Project report submitted by

MOHAMMED SHAFEEHE (KNR21EC058), JASIR NUFAIZ V (KNR21EC046), NIDA SALAM K (KNR21EC068), FIDA FATHIMA RM (KNR21EC038).

Towards the partial fulfilment of the requirements for the award of B. Tech. degree in

Electronics and Communication Engineering

Department of Electronics and Communication Engineering
GOVERNMENT COLLEGE OF ENGINEERING
KANNUR - 670 563

March 2025

GOVERNMENT COLLEGE OF ENGINEERING KANNUR

DEPARTMENT OF
ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

Certified that this is a bonafide report of the project presented by **MOHAMMED SHAFEEHE (KNR21EC058), JASIR NUFAIZ V (KNR21EC046), NIDA SALAM K (KNR21EC068), FIDA FATHIMA RM (KNR21EC038).** on 25 March 2025 with the title **DEVELOPMENT OF A SIGN LANGUAGE DETECTION AND CONVERSION SYSTEM USING MEDIA PIPE FOR MUTE AND DEAF INDIVIDUALS** towards the partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Electronics and Communication Engineering from APJ Abdul Kalam Technological University during the academic year 2024-2025.

Prof. Indu C.	**Prof. Siraj M**	**Dr. A. Ranjith Ram**
Project Supervisor	Project Co-ordinator	Head of the Department
Department of ECE	Department of ECE	Department of ECE
GCE Kannur	GCE Kannur	GCE Kannur

DECLARATION

We, the undersigned, hereby solemnly declare that this project report titled **DEVELOPMENT OF A SIGN LANGUAGE DETECTION AND CONVERSION SYSTEM USING MEDIA PIPE FOR MUTE AND DEAF INDIVIDUALS**, submitted for the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering from APJ Abdul Kalam Technological University is a bonafide record of our own work carried out under the supervision of **Prof. Indu C.**.

Wherever We have used materials (data, theoretical analysis, and text) from other sources, We have adequately and accurately cited the original sources.

We also declare that this work has not been submitted to any other institution in this University or any other University.

ACKNOWLEDGMENTS

We would like to take this opportunity to extend our sincere thanks to people who helped us to make this project possible. This project will be incomplete without mentioning all the people who helped us to make it move forward. Firstly, We would like to thank GOD, almighty, our supreme guide, for bestowing his blessings upon us in our entire endeavour. We thank our respected principal **Dr. Rajesh K N** who gave us all the support during the project development. We also thank **Dr. A. Ranjith Ram** HOD, Dept. of ECE, GCE Kannur for providing all the facilities and resources. We would also like to express our gratitude to **Prof. Siraj M**. project Coordinator, for his continuous support, guidance without which the seminar wouldn't have been a reality.

Finally, We would like to express our deepest gratitude and respect to our guide **Prof. Indu C** for the excellent guidance, constructive criticism and encouragement for the successful completion of our project.

ABSTRACT

This project aims to develop an integrated system using MediaPipe for real-time detection and recognition of sign language to assist mute individuals and an audio-to-sign language conversion system to aid deaf individuals. The sign language detection system will leverage MediaPipe's advanced hand-tracking capabilities to interpret sign language gestures, providing instant audio announcements for effective communication. Additionally, the audio-to-sign language conversion system will translate spoken words into visual sign language, displayed on a screen to facilitate understanding for deaf users. This project focuses on creating an accurate, efficient, and user-friendly solution that bridges the communication gap for mute and deaf individuals, enhancing their ability to interact seamlessly in various social and professional settings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
ASL	American Sign Language
CNN	Convolutional Neural Network
CV	Computer Vision
DNN	Deep Neural Networks
DOI	Digital Object Identifier
GPU	Graphics Processing Unitfu
GUI	Graphical User Interface
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
ML	Machine Learning
NLP	Neural Language Processing
OS	Operating System/Open Source
TTS	Text to Speech
VSCODE	Microsoft Visual Studio Code

CHAPTER 1

INTRODUCTION

More than 70 million deaf people around the world use sign languages to communicate. Communication is a fundamental human need, yet mute and deaf individuals often face significant barriers in their daily interactions. For mute individuals, the primary mode of communication is sign language, which is not universally understood, creating challenges when interacting with people unfamiliar with it. Deaf individuals, meanwhile, experience difficulties in understanding spoken language, especially in situations where real-time interpretation is unavailable. These barriers contribute to social isolation, limited participation, and reduced access to opportunities in personal and professional environments.

While traditional solutions such as sign language interpreters offer some support, they are not always accessible or feasible in every setting. Advances in machine learning, computer vision, and hand-tracking technologies present new opportunities to bridge this communication gap. In particular, MediaPipe, an open-source framework developed by Google, provides robust hand-tracking capabilities crucial for interpreting sign language gestures. This project leverages MediaPipe's technology to create a real-time system that enables two-way communication: it translates sign language gestures into spoken language for mute individuals and converts audio into visual sign language for deaf individuals. By integrating these functionalities, the project aims to promote inclusivity, providing a practical solution that facilitates communication between hearing, mute, and deaf individuals. This system is designed to be accessible on various devices, making it scalable and adaptable to a wide range of environments.

CHAPTER 2

LITERATURE SURVEY

2.1 Background

Communication barriers significantly impact the social and professional lives of individuals who are mute or deaf. Mute individuals primarily use sign language to communicate, but this poses a challenge when interacting with people who are unfamiliar with it. Similarly, deaf individuals often struggle to understand spoken language, especially in environments lacking sign language interpretation. Traditional solutions like human interpreters are not always practical or accessible in all situations, which highlights the need for technology-driven solutions to bridge this communication gap.[1]

Recent advancements in machine learning, computer vision, and natural language processing (NLP) have enabled the development of systems capable of real-time gesture recognition and language translation. MediaPipe, an open-source framework developed by Google, has become popular for building multimedia processing pipelines, especially for real-time applications. Its hand-tracking capability is particularly useful in recognizing hand gestures associated with sign language.[2]

Utilizing such frameworks, this project aims to create a system that converts sign language to speech for mute individuals and translates audio into sign language visuals for deaf individuals, providing a unified solution for enhancing communication.

2.2 Literature Reviews

The development of sign language detection and translation systems has seen considerable progress, yet several challenges remain, particularly in accuracy, real-time responsiveness, and accessibility. Below is an overview of notable studies and technologies that have informed the development of this project.[3]

Sign Language to Speech Conversion: Adey Roh et. al[4] explored a real-time system for converting sign language to speech using artificial neural networks (ANN). This system aimed to predict gestures accurately, but it struggled with recognizing continuous or co-articulated gestures in live videos.

Independent Recognition of Signers: Athira. et al[3] addressed the challenge of recognizing sign language from live videos, particularly focusing on independent recognition of signers. By eliminating co-articulation, the study provided insights into building systems that can accurately identify gestures across different individuals, which is crucial for applications intended for diverse user groups.

Vision-Based Gesture Recognition: Sherma, Sin. et al[4]implemented deep learning for vision-based hand gesture recognition, targeting the interpretation of static hand gestures in sign language. Their research demonstrated the potential of convolutional neural networks (CNNs) for accurate gesture detection but highlighted limitations in dynamic or sequential gesture recognition.

Static Hand Gesture Recognition with Deep CNN: Adithya, Raj. et al[5] developed a deep CNN model specifically for static hand gesture recognition, showing promising results for individual sign language symbols. However, they noted the limitations of CNNs in handling transitions between gestures, which is a challenge for continuous sign language recognition systems.

Intelligent Systems for Audio-to-Visual Translation, Adeyanju et al. : investigated intelligent systems capable of converting audio information into visual formats for enhanced accessibility. This research provides foundational insights into the integration of NLP and visual processing for real-time audio-to-sign language conversion.

2.3 RESEARCH GAPS

While these studies have advanced the field of sign language detection and translation, several gaps remain. Existing systems often face challenges such as:

Inaccuracy in Real-Time Detection: Many systems lack sufficient accuracy for practical real-time communication, making them less viable for everyday interactions.

High System Complexity: Current solutions frequently require specialized hardware or intensive processing, limiting accessibility.

Lack of Dual-Purpose Systems: Few systems offer both sign-to-speech and audio-to-sign translation within a single platform.

Limited Audio-to-Sign Language Tools: Existing systems mainly focus on sign language detection, with fewer options available for translating spoken language into sign language for deaf users.

This project aims to address these gaps by utilizing MediaPipe's robust hand-tracking technology and integrating it with NLP to create a holistic, real-time solution for both mute and deaf individuals.

CHAPTER 3

PROPOSED SYSTEM

The proposed system is designed to bridge communication gaps for mute and deaf individuals by recognizing and translating sign language gestures and converting spoken language into sign language. This system employs MediaPipe's hand-tracking technology to detect sign language gestures in real-time, translating them into spoken audio for mute individuals, and similarly, it utilizes natural language processing (NLP) for converting audio into visual sign language to support deaf individuals.

This dual-function system aims to:

1. Recognize hand gestures and convert them into audible speech.

For mute individuals, the system will detect and interpret their sign language gestures in real-time and convert these gestures into spoken words. This allows mute individuals to communicate with hearing people who may not know sign language, as their gestures are "translated" into audible speech through the system.

2. Translate spoken language into visual sign language using a database of gestures.

For deaf individuals, the system will translate spoken language into visual sign language displays. Using a database of sign language gestures, the system converts spoken words into corresponding sign language symbols or animations on a screen, allowing deaf individuals to understand conversations without needing a human interpreter.

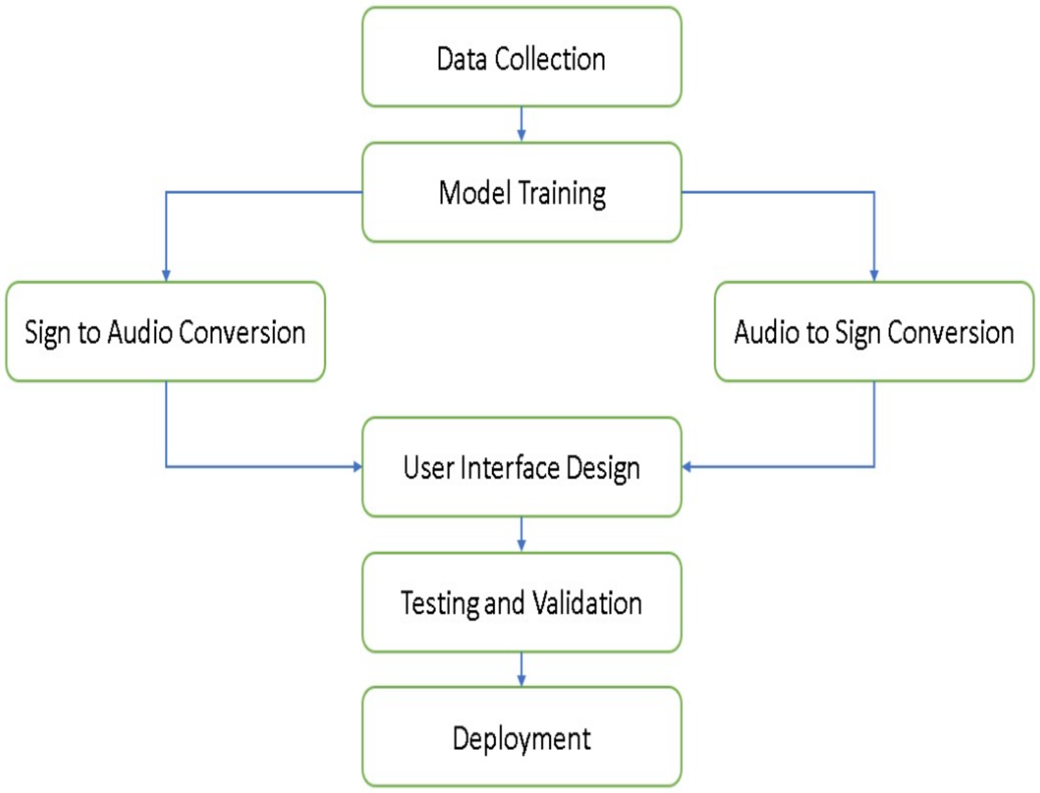


Fig. 3.1. Methodology Flowchart.

Using MediaPipe's lightweight and efficient framework, the system can be deployed across various platforms like smartphones and computers, enabling accessibility in diverse environments.

3.1 Methodologies and Algorithms

1. Data Collection: Sign language gesture data is gathered for model training. This data, consisting of images or video clips of hand gestures, forms the basis for accurate recognition and conversion processes.

2. Model Training: The collected data is used to train a machine learning model. The model learns to interpret the gestures and maps them to corresponding phrases or words, which can be converted into audio for sign-to-audio and into visual signs for audio-to-sign.

3. Sign-to-Audio Conversion:

Algorithm: The sign-to-audio conversion relies on MediaPipe's hand-tracking algorithm, which identifies key points in hand gestures. A machine learning model processes these gestures, mapping each gesture to its corresponding word, and then converting it to audio output.

Gesture Recognition: The algorithm detects hand landmarks, including finger and palm positions, to classify gestures accurately. These gestures are pre-mapped to phrases, which are then outputted as speech through a text-to-speech module.

4. Audio-to-Hand Sign Conversion:

Algorithm: Natural language processing (NLP) is a branch of artificial intelligence (AI) that allows computers to understand, process, and manipulate human language. NLP uses ML, statistical methods, neural networks, and text mining to interpret human language. It can be applied to both written text and speech. NLP techniques process audio input, converting it into corresponding gestures stored in a database.

Real-Time Display: A visual sign representation is generated for each recognized word, ensuring real-time feedback and facilitating comprehension. User Interface Design: The interface provides users with an intuitive platform to interact with both sign-to-audio and audio-to-sign functionalities, ensuring usability for individuals of varying technical abilities.

5. Testing and Validation: Comprehensive testing is conducted to ensure accurate detection, translation, and output across various scenarios, focusing on response time, gesture recognition accuracy, and audio clarity.

6. User Interface Design: The interface provides users with an intuitive platform to interact with both sign-to-audio and audio-to-sign functionalities, ensuring usability for individuals of varying technical abilities.

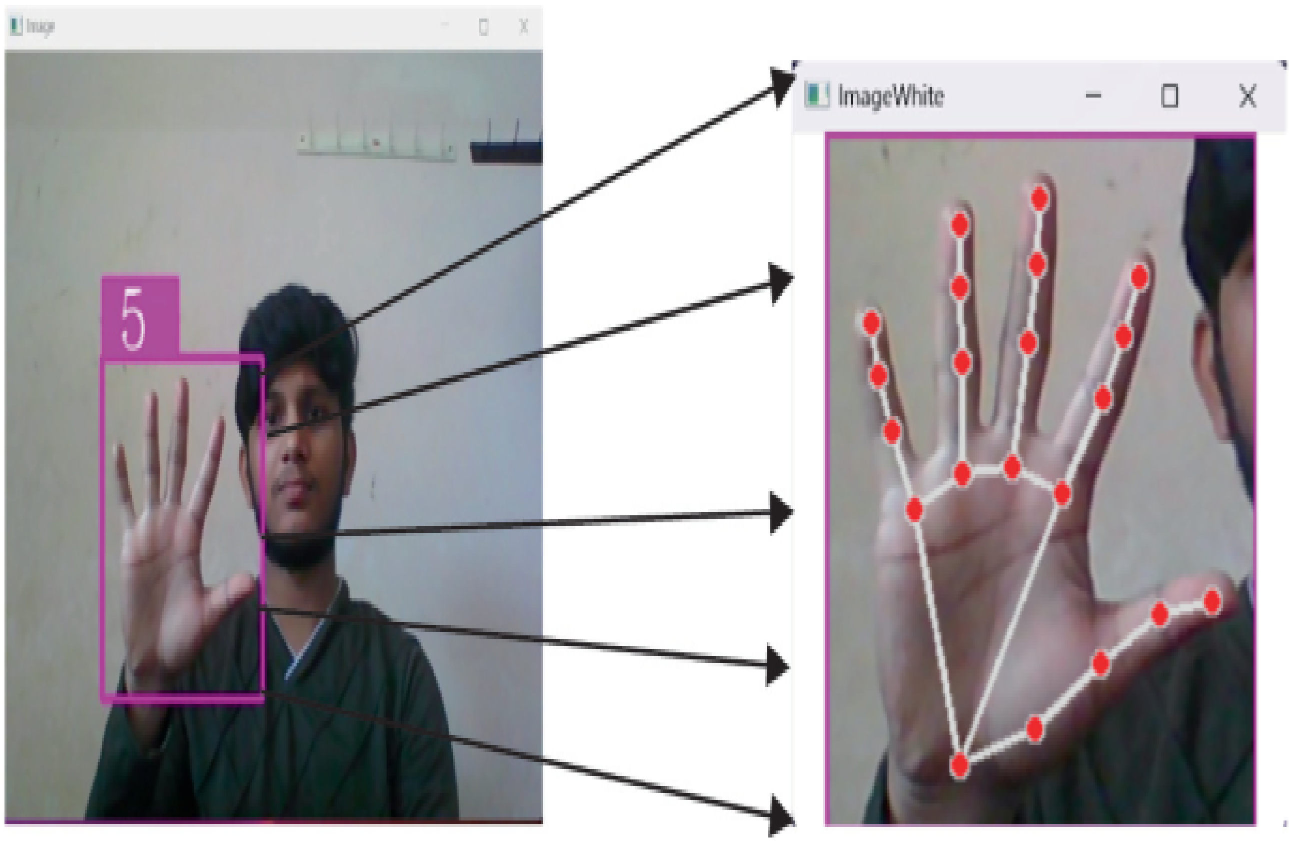


Fig. 3.2. Real-time handtracking implemented in MS Visual Code Studio

3.2 How Does The System Work?

Hand-Tracking Technology: The core technology used in this system is MediaPipe, an open-source framework developed by Google. MediaPipe's advanced hand-tracking capability allows the system to detect and track hand movements and shapes precisely. It captures hand gestures, maps them to specific meanings, and converts them into the desired output (speech or visual signs).

Real-Time Conversion: The system is designed for real-time usage, meaning it processes input (hand gestures or spoken language) instantly, ensuring smooth and natural communication without noticeable delay. This feature makes it useful in various settings, from casual social interactions to workplace communication.

Device Compatibility: Since MediaPipe is lightweight, the system can run on commonly available devices like smartphones, computers. This makes it a portable and easily accessible tool for users, allowing to carry the system wherever they go.

3.2.1 Data Acquisition

It uses electromechanical devices to provide exact hand configuration, and position. Different glove-based approaches can be used to extract information. But it is expensive and not user friendly.

In vision-based methods, the computer webcam is the input device for observing the information of hands and/or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices, thereby reducing costs. The main challenge of vision-based hand detection ranges from coping with the large variability of the human hand's appearance due to a huge number of hand movements, to different skin-color possibilities as well as to the variations in viewpoints, scales, and speed of the camera capturing the scene.

3.2.2 Data Pre-Processing and Feature Extraction

In this approach for hand detection, firstly we detect hand from image that is acquired by webcam and for detecting a hand we used media pipe library which is used for image processing. So, after finding the hand from image we get the region of interest (Roi) then we cropped that image and convert the image to gray image using OpenCV library after we applied the gaussian blur .The filter can be easily applied using open computer vision library also known as OpenCV. Then we converted the gray image to binary image using threshold and Adaptive threshold methods.

3.2.2.1 Hand landmark Model Bundle of MediaPipe

The hand landmark model bundle detects the keypoint localization of 21 hand-knuckle coordinates within the detected hand regions. The model can be trained on approximately 30K real-world images, as well as several rendered synthetic hand models imposed over various backgrounds.

The hand landmarker model bundle contains palm detection model and hand

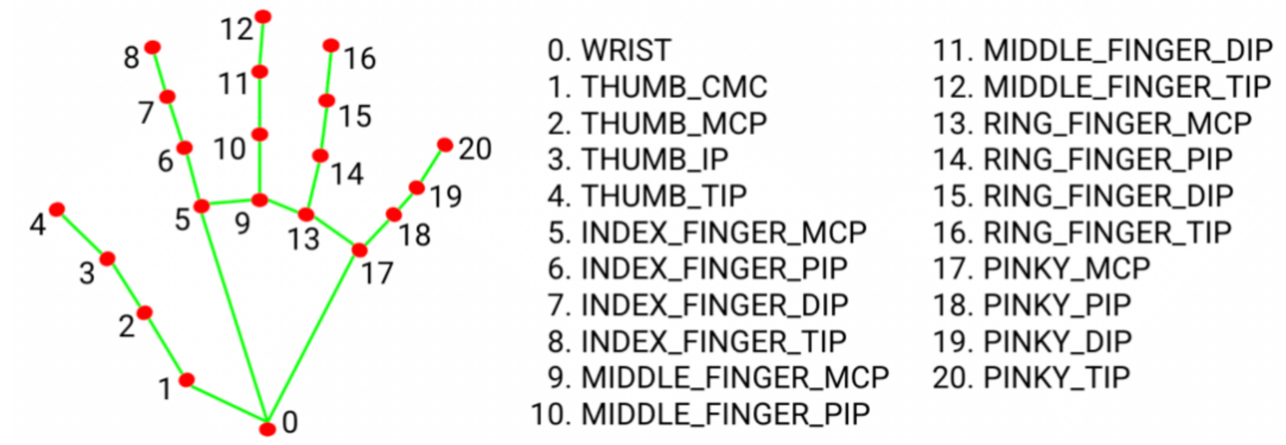


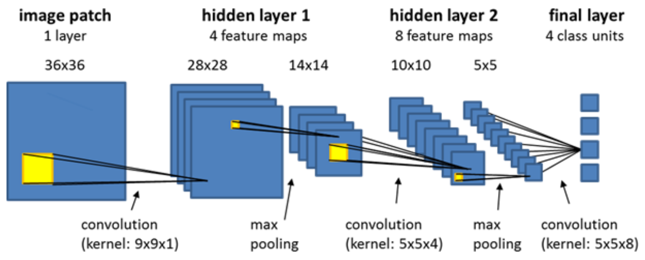
Fig. 3.3. Hand Landmark Model Bundle Detecting Key Points

landmarks detection model. Palm detection model localizes the region of hands from the whole input image, and the hand landmarks detection model finds the landmarks on the cropped hand image defined by the palm detection model.

Since palm detection model is much more time consuming, in Video mode or Live stream mode, Gesture Recognizer uses bounding box defined by the detected hand landmarks in the current frame to localize the region of hands in the next frame. This reduces the times Gesture Recognizer triggering palm detection model. Only when the hand landmarks model could no longer identify enough required number of hands presence, or the hand tracking fails, palm detection model is invoked to relocalize the hands.

3.2.3 Algorithm: Convolutional Neural Network(CNN)

A 3D Convolutional Neural Network (3D CNN) is a deep learning model that processes volumetric data by capturing spatial information across multiple slices simultaneously. It can be used for Medical Imaging, Human Activity Recognitions and in Engineering.



CNNs have shown remarkable performance in sign language recognition. Using CNNs for sign language recognition involves training the network to recognize patterns in a dataset of sign language gestures. This dataset typically comprises videos or images of people performing the signs, annotated with labels indicating the corresponding signs. During training, the CNN learns to identify the unique features of each sign and uses these features to recognize signs it has not seen before.

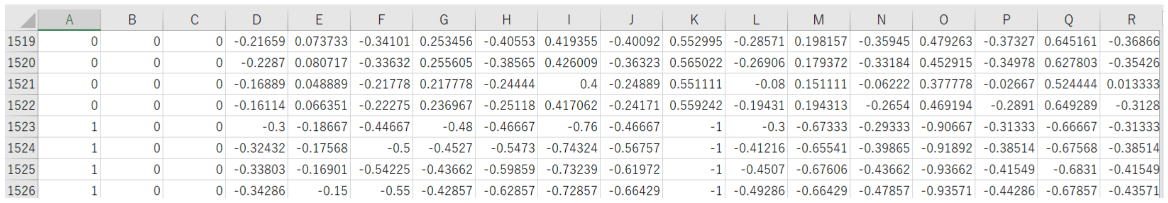
CNN is a class of neural networks that are highly useful in solving computer vision problems. They found inspiration from the actual perception of vision that takes place in the visual cortex of our brain. They make use of a filter/kernel to scan through the entire pixel values of the image and make computations by setting appropriate weights to enable detection of a specific feature. CNN is equipped with layers like convolution layer, max pooling layer, flatten layer, dense layer, dropout layer and a fully connected neural network layer. These layers together make a very powerful tool that can identify features in an image. The starting layers detect low level features that gradually begin to detect more complex higher-level features

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner.

Moreover, the final output layer would have dimensions(number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class IDs.

3.3 DATASET COLLECTION:

- **Gesture Recognition Training:**
  - The dataset was collected and preprocessed hand landmark coordinates. The extracted 21 keypoints per hand were converted into numerical features representing the spatial positioning of fingers. Then stored in CSV files, which were later used to train the classification model.

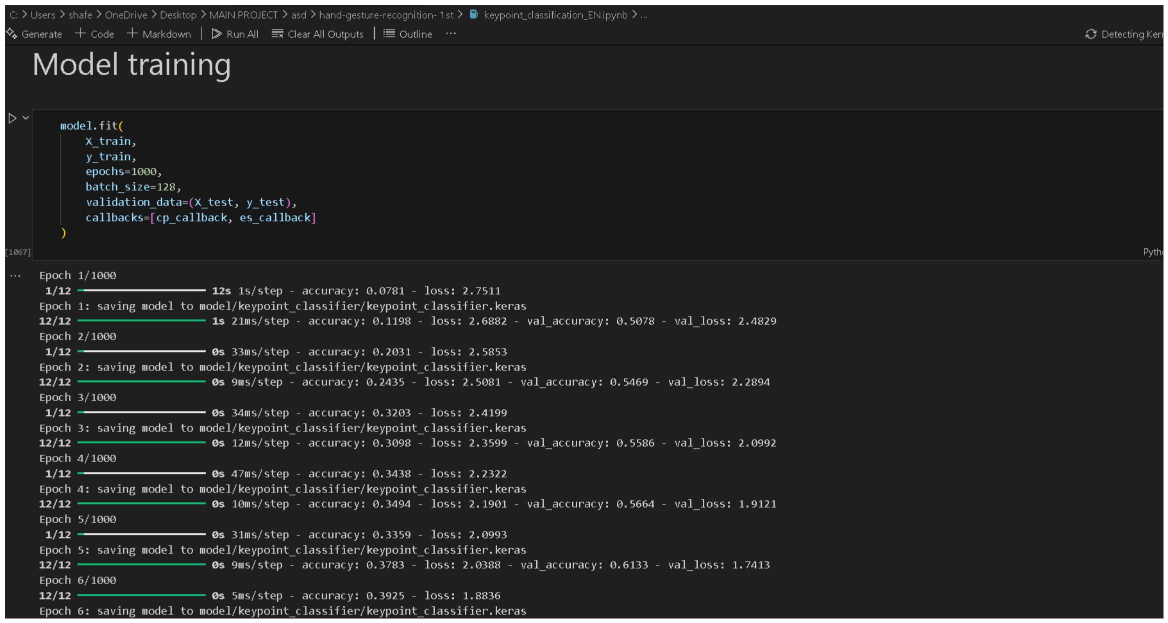


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1519	0	0	0	-0.21659	0.073733	-0.34101	0.253456	-0.40553	0.419355	-0.40092	0.552995	-0.28571	0.198157	-0.35945	0.479263	-0.37327	0.645161	-0.36866
1520	0	0	0	-0.2287	0.080717	-0.33632	0.255605	-0.38565	0.426009	-0.36323	0.565022	-0.26906	0.179372	-0.33184	0.452915	-0.34978	0.627803	-0.35426
1521	0	0	0	-0.16889	0.048889	-0.21778	0.217778	-0.24444	0.4	-0.24889	0.551111	-0.08	0.151111	-0.06222	0.377778	-0.02667	0.524444	0.013333
1522	0	0	0	-0.16114	0.066351	-0.22275	0.236967	-0.25118	0.417062	-0.24171	0.559242	-0.19431	0.194313	-0.2654	0.469194	-0.2891	0.649289	-0.3128
1523	1	0	0	-0.3	-0.18667	-0.44667	-0.48	-0.46667	-0.76	-0.46667	-1	-0.3	-0.67333	-0.29333	-0.90667	-0.31333	-0.66667	-0.31333
1524	1	0	0	-0.32432	-0.17568	-0.5	-0.4527	-0.5473	-0.74324	-0.56757	-1	-0.41216	-0.65541	-0.39865	-0.91892	-0.38514	-0.67568	-0.38514
1525	1	0	0	-0.33803	-0.16901	-0.54225	-0.43662	-0.59859	-0.73239	-0.61972	-1	-0.4507	-0.67606	-0.43662	-0.93662	-0.41549	-0.6831	-0.41549
1526	1	0	0	-0.34286	-0.15	-0.55	-0.42857	-0.62857	-0.72857	-0.66429	-1	-0.49286	-0.66429	-0.47857	-0.93571	-0.44286	-0.67857	-0.43571

Fig. 3.4. 1st column: Class ID's, subsequent columns: Key points.

3.4 Model Training:

The model training process in the Sign Language Detection and Conversion project was implemented using Keras and TensorFlow in a Jupyter Notebook environment. The training focused on recognizing hand gestures based on keypoint classification using MediaPipe hand tracking.


C: > Users > shafe > OneDrive > Desktop > MAIN PROJECT > asd > hand-gesture-recognition- 1st > keypoint_classification_EN.ipynb > ...
Generate
+ Code
+ Markdown
Run All
Clear All Outputs
Outline
Detecting Ker
Pyth

Model training

```
model.fit(
    X_train,
    y_train,
    epochs=1000,
    batch_size=128,
    validation_data=(X_test, y_test),
    callbacks=[cp_callback, es_callback]
)
```

[1067]

```
Epoch 1/1000
 1/12 ━━━━━━━━━━━━━━━━━━━━ 12s 1s/step - accuracy: 0.0781 - loss: 2.7511
Epoch 1: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 1s 21ms/step - accuracy: 0.1198 - loss: 2.6882 - val_accuracy: 0.5078 - val_loss: 2.4829
Epoch 2/1000
 1/12 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.2031 - loss: 2.5853
Epoch 2: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.2435 - loss: 2.5081 - val_accuracy: 0.5469 - val_loss: 2.2894
Epoch 3/1000
 1/12 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.3203 - loss: 2.4199
Epoch 3: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.3098 - loss: 2.3599 - val_accuracy: 0.5586 - val_loss: 2.0992
Epoch 4/1000
 1/12 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.3438 - loss: 2.2322
Epoch 4: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3494 - loss: 2.1901 - val_accuracy: 0.5664 - val_loss: 1.9121
Epoch 5/1000
 1/12 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.3359 - loss: 2.0993
Epoch 5: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.3783 - loss: 2.0388 - val_accuracy: 0.6133 - val_loss: 1.7413
Epoch 6/1000
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.3925 - loss: 1.8836
Epoch 6: saving model to model/keypoint_classifier/keypoint_classifier.keras
```

Fig. 3.5. Training datasets in Jupyter Notebook(in VSCODE)

3.4.1 Model Architecture and Training Process

The neural network used for classification was built using Keras, with a fully connected dense layer structure optimized for recognizing hand signs. The training process involved:

- Using Adam optimizer and categorical cross-entropy loss for multi-class classification.
- Applying batch normalization and dropout layers to prevent overfitting.
- Training over multiple epochs while tracking accuracy and loss.

The primary file for training datsets is a Jupyter Notebook that handled data preprocessing, model architecture, training, and evaluation. The trained model was then saved in the HDF5 format (model.hdf5) to store the learned weights and architecture for later inference.

3.4.2 Model Deployment in Main Script File

Once trained, the HDF5 model was loaded into main script file, where it was used for real-time gesture recognition. During runtime, live webcam frames were processed to extract keypoints, which were then fed into the trained model for classification. The model predicted the corresponding sign language gesture, displaying it on the UI and announcing the result via text-to-speech.

By training a custom sign language recognition model, the project successfully enabled real-time gesture detection and sign translation, making communication more accessible for users who rely on sign language.

CHAPTER 4

SOFTWARE AND HARDWARE REQUIREMENTS

4.1 Software Requirements:

1. Python Programming Language: Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

2. Microsoft Visual Code Studio: Visual Studio Code, also commonly referred to as VS Code, is an integrated development environment developed by Microsoft for Windows, Linux, macOS and web browsers. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded version control with Git. Users can change the theme, keyboard shortcuts, preferences, and install extensions that add functionality.

3. Machine Learning Libraries: ML libraries are collections of pre-built components, utilities, and conventions that help developers and data scientists build machine learning models. They can help speed up the development process by automating low-level tasks, allowing developers to focus on more complex aspects of the project. Frameworks like TensorFlow, PyTorch, Kernel, Matplotlib, NumPy are required to train and deploy gesture recognition models.

4. Natural Language Processing (NLP) Tools: Specificaly for the speech recognition purpose. NLP is a branch of artificial intelligence (AI) that helps computers understand human language. It can be used to process and analyze large amounts of data from various communication channels, such as emails, social media, and text messages. Libraries such as NLTK or spaCy and as well as python libraries are utilized to process audio-to-text conversion, necessary for translating audio into sign language.

5. Text-to-Speech (TTS) Library: The Text-to-Speech API enables developers to generate human-like speech. The API converts text into audio formats such as WAV, MP3, or Ogg Opus. Libraries like Google Text-to-Speech API or Pyttsx will be used to convert text output from recognized gestures into audible speech.

6. Software Platform Compatibility: The softwares and libraries used to compatible with each in a way that it meets the purpose of the system as well as it should run on Windows, macOS, and mobile operating systems like Android and iOS for broader accessibility.

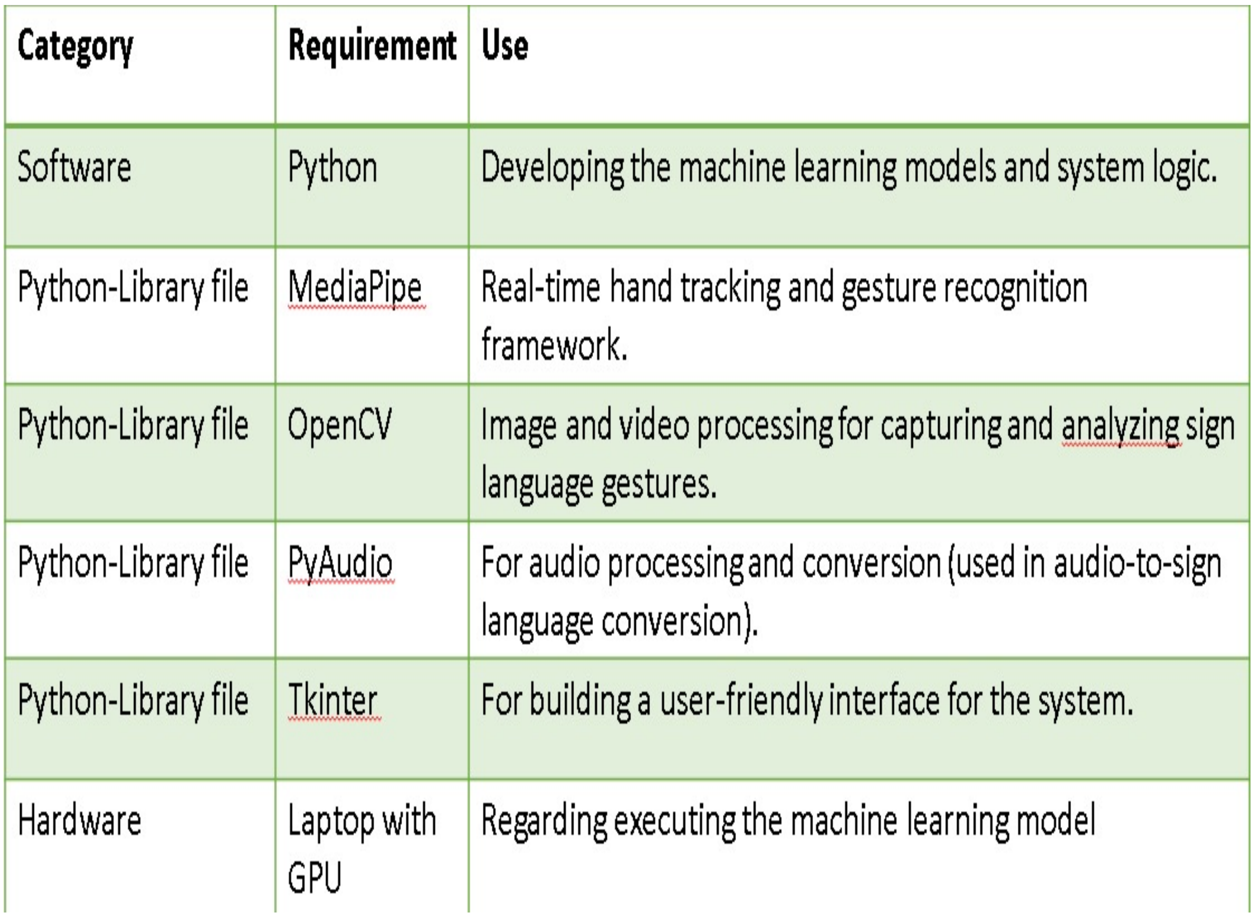
4.2 Hardware Requirements:

1. Webcam or Camera: A high-definition camera is required for capturing hand gestures accurately in real-time.

2. Microphone: A microphone is necessary to capture audio inputs for conversion to sign language.

3. Laptop Requirments: A modern processor (e.g., Intel i5/AMD Ryzen 5 or higher) is recommended for smooth real-time processing and for rendering purpose.

GPU (optional but recommended): A graphics processing unit, such as NVIDIA's GTX series, can significantly accelerate model training and inference



Category	Requirement	Use
Software	Python	Developing the machine learning models and system logic.
Python-Library file	MediaPipe	Real-time hand tracking and gesture recognition framework.
Python-Library file	OpenCV	Image and video processing for capturing and analyzing sign language gestures.
Python-Library file	PyAudio	For audio processing and conversion (used in audio-to-sign language conversion).
Python-Library file	Tkinter	For building a user-friendly interface for the system.
Hardware	Laptop with GPU	Regarding executing the machine learning model

Fig. 4.1. Software, Libraries and Hardwares Required

times.

Display Screen: For the audio-to-sign conversion output, a display is essential for showing visual sign language representations to the user.

4.3 Frameworks and Libraries Used:

Fig. 4.2. Open Source Computer Vision Library

1. OpenCV (Open Source Computer Vision Library): OpenCV, or Open Source Computer Vision Library, is a free and open-source software library that's considered the standard tool for computer vision and image processing. It is widely used for computer vision tasks, making it essential for processing video input and recognizing gestures in this project. Specifically, it helps with real-time video capture and processing, allowing the system to recognize hand gestures and other movements.

2. Python Library 'PyAudio' : In this project, PyAudio captures the spoken words using microphone that the system converts into sign language visuals. Additionally, it allows for playback of audio output, where sign language gestures are converted into speech for mute individuals.

3. Python Library 'Tkinter' : Tkinter is the standard GUI (Graphical User Interface) library for Python. It provides tools to create windows, buttons, text boxes, and other interactive elements. This library builds the user interface of the system, allowing users to interact with the sign language detection and conversion functions. Tkinter ensures that the system remains accessible and user-friendly on various devices.

4. MediaPipe Framework:

Fig. 4.3. Google MediaPipe for hand gesture recognition

This is central to hand-tracking and gesture recognition. MediaPipe enables high-precision tracking and image processing capabilities. These libraries and resources provide the core functionality for each MediaPipe solution.

MediaPipe Tasks: Cross-platform APIs and libraries for deploying solutions.

MediaPipe Models: Pre-trained, ready-to-run models for use with each solution. These tools allow you to customize and evaluate solutions.

MediaPipe Model Maker: Customize the models for solutions with your data.

MediaPipe Studio: Visualize, evaluate, and benchmark solutions in your browser.

The MediaPipe Gesture Recognizer task lets you recognize hand gestures in real time, and provides the recognized hand gesture results along with the landmarks of the detected hands. You can use this task to recognize specific hand gestures from a user, and invoke application features that correspond to those gestures. This task operates on image data with a machine learning (ML) model, and accepts either static data or a continuous stream. The task outputs hand landmarks in image coordinates, hand landmarks in world coordinates, handedness (left/right hand), and the hand gesture categories of multiple hands.

5.TensorFlow (version 18.2.0):

Fig. 4.4. Open Source TensorFlow framework

TensorFlow played a crucial role in the development of the Sign Language Detection and Conversion project by providing a robust machine learning framework for training and deploying the hand gesture recognition model. The key aspects of its usage in this project include:

4.3.1 Model Training for Hand Gesture Recognition:

TensorFlow was used to train a keypoint-based classification model to recognize different hand gestures. The model was trained using a dataset of hand landmarks extracted using MediaPipe Hands. The training data consisted of keypoint coordinates (x, y, z) for each detected hand landmark, which were labeled according to the corresponding gesture.

4.3.2 TensorFlow and Keras for Deep Learning:

Keras (TensorFlow's high-level API) was used to define a Sequential Neural Network for gesture classification. The model included multiple dense (fully connected) layers with activation functions such as ReLU and Softmax for classification.

4.3.3 Deployment and Real-Time Inference:

- The trained model was saved in .h5 format and loaded in app.py for real-time inference.
- TensorFlow's TFLite conversion allowed the model to be optimized for efficiency and speed during execution. The model performed classification by predicting the most likely hand gesture based on the detected keypoints.

4.3.4 Integration with OpenCV and MediaPipe:

TensorFlow worked alongside OpenCV and MediaPipe Hands to process live video input from a webcam. MediaPipe extracted hand keypoints, which were then fed into the TensorFlow model for classification. The recognized gesture was used to trigger corresponding text output and sign language translation.

6. Python Library 'Pyttsx3': The pyttsx3 library was used in the project to enable text-to-speech (TTS) functionality for announcing recognized hand signs in real-time. This allowed the system to provide auditory feedback alongside visual recognition, making it more accessible for communication between individuals who use sign language and those who do not.

7. Google Web Speech Recognition API:

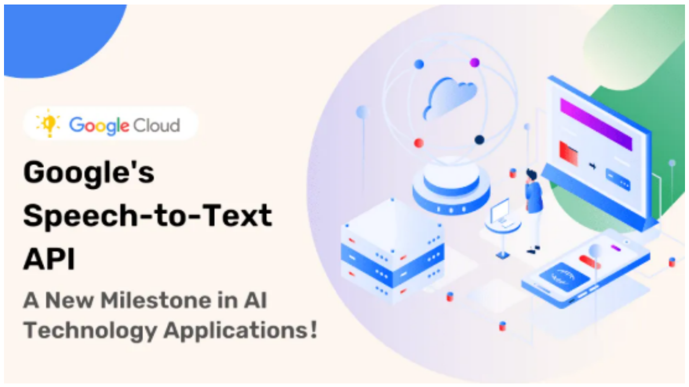

Google Cloud
Google's Speech-to-Text API
A New Milestone in AI Technology Applications!

Fig. 4.5. Google Web Speech Recognition API

This feature allowed the system to recognize spoken words and map them to corresponding sign language representations, facilitating audio-to-sign translation for better communication with individuals who are deaf or hard of hearing.

Although the Google Web Speech API requires an internet connection, it was chosen for its fast processing speed and high accuracy in recognizing natural speech. By integrating this API, the project successfully achieved bidirectional communication, where spoken words were converted into sign language gestures, enhancing accessibility and inclusivity for users who rely on sign language.

CHAPTER 5

RESULTS AND DISCUSSIONS

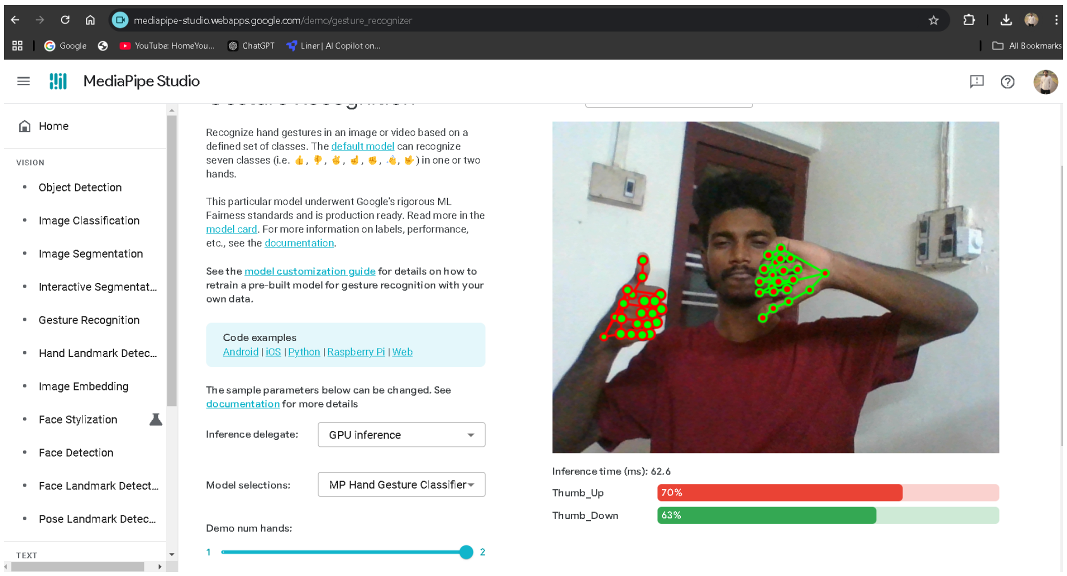
- ## PHASE 1 RESULTS:

Successfull implementation of recognition of basic hand gestures with FPS labelled in the o/p screen. The bounding box is also implemented to make the user interface user-friendly. Followed the Beginner Guide in MediaPipe to famaliarize with the Basics and programming algorithm of Hand Gesture Recognition through its Web Demo(fig[1]).

Built the Basic hand gesture recognition model using the Python programming language (python 3.12 as of now) leveraging the python Machine Learning Libraries(Tensorflow, Open-CV, Mediapipe). The compatibility issues arised are solved after proper research(fig[2]).

The data collection and model training is begun thereafter, demonstrated in the figures below. IDs are assigned to the respective hand gestures and their key-points logged onto the Directories created in the back-end.

5.1 Demonstration



Recognize hand gestures in an image or video based on a defined set of classes. The default model can recognize seven classes (i.e. 👍, 👎, ✌, ☝, ✊, 👋, 🤟) in one or two hands.

This particular model underwent Google's rigorous ML Fairness standards and is production ready. Read more in the model card. For more information on labels, performance, etc., see the documentation.

See the model customization guide for details on how to retrain a pre-built model for gesture recognition with your own data.

Code examples
Android | iOS | Python | Raspberry Pi | Web

The sample parameters below can be changed. See documentation for more details

Inference delegate: GPU inference

Model selections: MP Hand Gesture Classifier

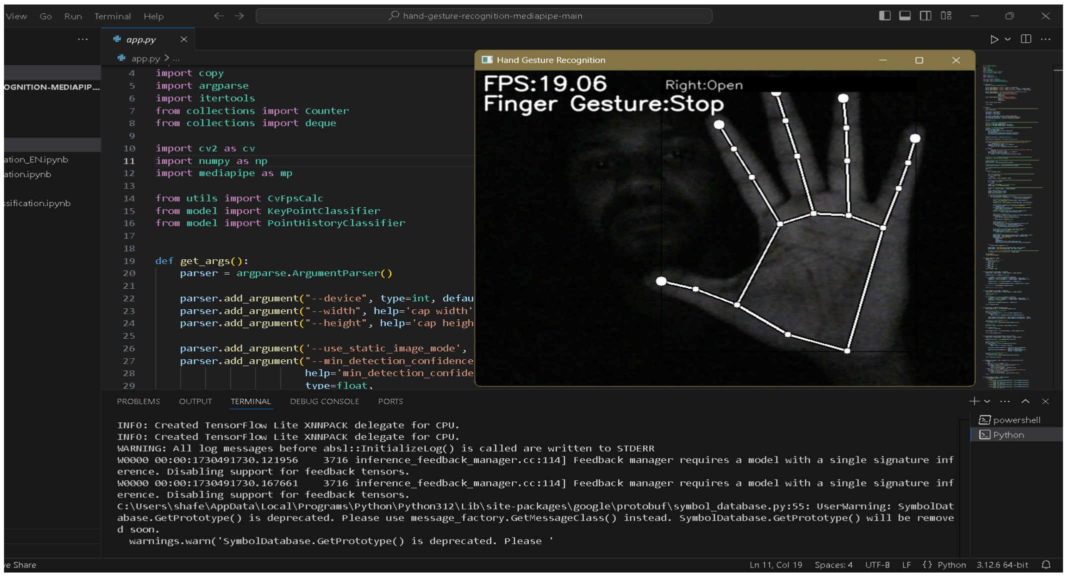
Demo num hands: 1 — 2

Inference time (ms): 62.6

Thumb_Up 70%

Thumb_Down 63%

Fig. 5.1. [1]Web Demo of Basic Hand Gesture Recognition in MediaPipe



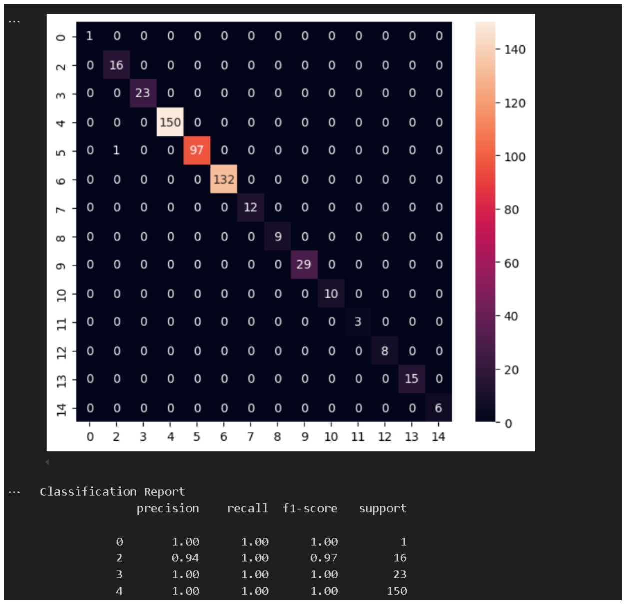
```python
import copy
import argparse
import itertools
from collections import Counter
from collections import deque

import cv2 as cv
import numpy as np
import mediapipe as mp

from utils import CvFpsCalc
from model import KeyPointClassifier
from model import PointHistoryClassifier


def get_args():
    parser = argparse.ArgumentParser()

    parser.add_argument("--device", type=int, defau
    parser.add_argument("--width", help='cap width'
    parser.add_argument("--height", help='cap heigh

    parser.add_argument('--use_static_image_mode',
    parser.add_argument("--min_detection_confidence
                        help='min_detection_confide
                        type=float,
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
W0000 00:00:1730491730.121956    3716 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signature inf
erence. Disabling support for feedback tensors.
W0000 00:00:1730491730.167661    3716 inference_feedback_manager.cc:114] Feedback manager requires a model with a single signature inf
erence. Disabling support for feedback tensors.
C:\Users\shafe\AppData\Local\Programs\Python\Python312\Lib\site-packages\google\protobuf\symbol_database.py:55: UserWarning: SymbolDat
abase.GetPrototype() is deprecated. Please use message_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will be remove
d soon.
  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
```

Fig. 5.2. [2]Basic Hand Gesture Recognition executed in MS Visual Studio Code

• END RESULT AND DISCUSSIONS (PHASE 2)

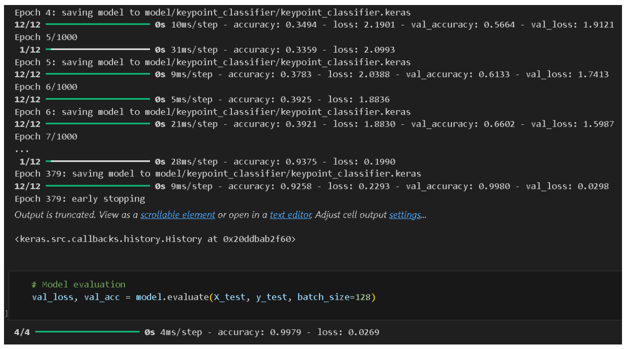
The advanced features of the project were successfully developed, tested, and validated as a result of extensive research and overcoming the challenges encountered throughout the development process.



	0	2	3	4	5	6	7	8	9	10	11	12	13	14
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	16	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	23	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	150	0	0	0	0	0	0	0	0	0	0
5	0	1	0	0	97	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	132	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	12	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	9	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	29	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	10	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	3	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	8	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	15	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	6

```
...  Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           2       0.94      1.00      0.97        16
           3       1.00      1.00      1.00        23
           4       1.00      1.00      1.00       150
```

Fig. 5.3. Confusion matrix in training process



```
Epoch 4: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.3494 - loss: 2.1901 - val_accuracy: 0.5664 - val_loss: 1.9121
Epoch 5/1000
 1/12 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.3359 - loss: 2.0993
Epoch 5: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.3783 - loss: 2.0388 - val_accuracy: 0.6133 - val_loss: 1.7413
Epoch 6/1000
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.3925 - loss: 1.8836
Epoch 6: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.3921 - loss: 1.8830 - val_accuracy: 0.6602 - val_loss: 1.5987
Epoch 7/1000
...
 1/12 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9375 - loss: 0.1990
Epoch 379: saving model to model/keypoint_classifier/keypoint_classifier.keras
12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9258 - loss: 0.2293 - val_accuracy: 0.9980 - val_loss: 0.0298
Epoch 379: early stopping
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

<keras.src.callbacks.history.History at 0x20ddbab2f60>

    # Model evaluation
    val_loss, val_acc = model.evaluate(X_test, y_test, batch_size=128)

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9979 - loss: 0.0269
```

Fig. 5.4. Running epochs and model evaluation

Fig. 5.5.

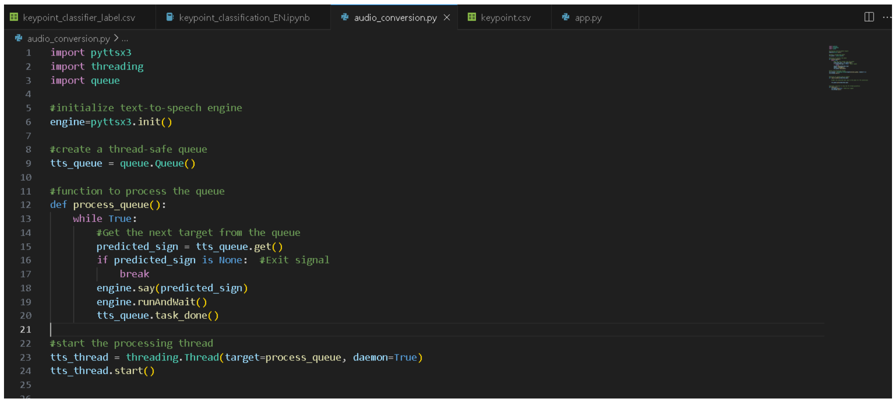
The project resulted in the development and implementation of the following features:

1. Hand Sign Recognition based on American Sign Language(ASL):

- Utilizes Mediapipe for real-time custom hand landmark detection. Some examples for custom trained ASL sign are:
  where is (ID=2), Ok (ID=3), I (ID=4), Want(ID=5), You(ID=6), To eat(Food)(ID=7), name(ID=10) I'm Sorry(ID=11) why(ID=12) How(ID=13) I need help(ID=14) and so on.

2. Hand Sign-to-Audio Conversion (audio conversion.py)

- The Hand Sign to Audio Conversion feature enables real-time recognition of hand gestures and converts them into spoken words using Mediapipe and a trained machine learning model (TensorFlow/Keras), the system classifies hand gestures into predefined categories. Once a gesture is recognized, the corresponding text is generated and converted into speech using pyttsx3, a text-to-speech (TTS) engine.



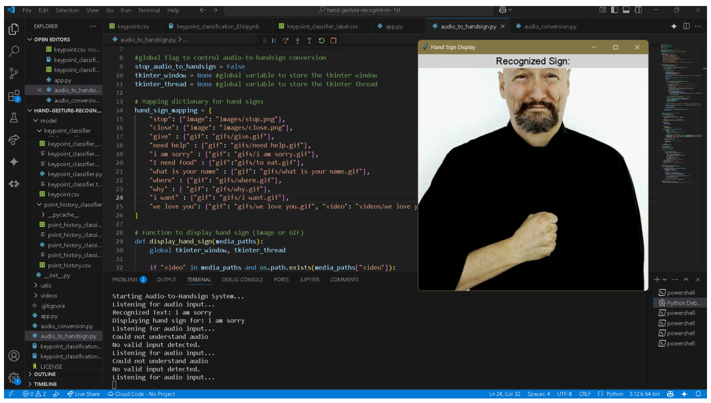
```python
import pyttsx3
import threading
import queue

#initialize text-to-speech engine
engine=pyttsx3.init()

#create a thread-safe queue
tts_queue = queue.Queue()

#function to process the queue
def process_queue():
    while True:
        #Get the next target from the queue
        predicted_sign = tts_queue.get()
        if predicted_sign is None:  #Exit signal
            break
        engine.say(predicted_sign)
        engine.runAndWait()
        tts_queue.task_done()

#start the processing thread
tts_thread = threading.Thread(target=process_queue, daemon=True)
tts_thread.start()
```

Fig. 5.6. script for audio announcement

3. Voice-to-Hand Sign Conversion (audio to handsign.py)

- This feature translates spoken words into sign language. It utilizes Google Web Speech Recognition API to transcribe voice input into text. This feature assists hearing-impaired individuals by providing a visual representation of spoken language.

- The recognized text is then matched with a predefined mapping of sign language(images, GIFs, or videos). If a match is found, the corresponding sign is displayed in a Tkinter window.



```python
#global flag to control audio-to-handsign conversion
stop_audio_to_handsign = False
tkinter_window = None #global variable to store the tkinter window
tkinter_thread = None #global variable to store the tkinter thread

# Mapping dictionary for hand signs
hand_sign_mapping = {
    "stop": {"image": "images/stop.png"},
    "close": {"image": "images/close.png"},
    "give" : {"gif": "gifs/give.gif"},
    "need help" : {"gif": "gifs/need help.gif"},
    "i am sorry" : {"gif": "gifs/i am sorry.gif"},
    "I need food" : {"gif":"gifs/to eat.gif"},
    "what is your name" : {"gif": "gifs/what is your name.gif"},
    "where" : {"gif": "gifs/where.gif"},
    "why" : { "gif": "gifs/why.gif"},
    "i want" : {"gif": "gifs/i want.gif"},
    "we love you": {"gif": "gifs/we love you.gif", "video": "videos/we love y
}

# Function to display hand sign (Image or GIF)
def display_hand_sign(media_paths):
    global tkinter_window, tkinter_thread

    if "video" in media_paths and os.path.exists(media_paths["video"]):
```

```
Starting Audio-to-Handsign System...
Listening for audio input...
Recognized Text: i am sorry
Displaying hand sign for: i am sorry
Listening for audio input...
Could not understand audio
No valid input detected.
Listening for audio input...
Could not understand audio
No valid input detected.
Listening for audio input...
```

Fig. 5.7. Running the Voice-to-Handsign Conversion script

5.2 Fine Tuning Techniques And Optimization Algorithm:

1. Confidence threshold based recognition algorithm :

- Cause: previously Opencv was detecting the hand signs based on which data it is nearest to, increasing false positives.
- Solution: Set a default value for min. confidence threshold. If it doesn't meet min.threshold, displays "UNKNOWN" but doesn't announce.

2. "Gesture Speed Adaption and Tracking" Algorithm:

- The system detect signing speed and dynamically adjust the confidence threshold. Faster gestures require a higher confidence threshold, reducing false positives.

5.2.1 How It Works:

- Track the position of key landmarks (e.g., wrist, fingers) over time.
- Calculate speed based on distance moved per frame.(Euclidean Distance formula)
- Adjust confidence thresholds or frame capture rate dynamically based on speed(slow,normal,

3. Sentence Formation using Buffer:

- Added sign sentence buffer at the beginning of main().
- Modified gesture recognition to store words in the buffer.
- Added logic to announce full sentences after a delay of 3 seconds.
- Added display of sentence buffer on the OpenCV window.

4. Added toggling ON/OFF Modes:

for TTS announcement, clear and reset operations and for mode selection

CHAPTER 6

FUTURE SCOPE

While this project demonstrates significant progress in making sign language communication accessible, several areas remain for future development and refinement. Potential advancements include:

1. Increased Language Coverage and Customization: Future versions of the system could incorporate additional sign languages and dialects, expanding usability globally. Customization options for different gestures or personalized signs could allow users to adapt the system to individual communication needs or specific regional sign variations.

2. Enhanced Gesture Recognition with Advanced AI Models: Incorporating more advanced machine learning models, such as deep learning architectures, could improve the system's ability to recognize complex gestures, including facial and emotional expressions and body posture. These improvements could facilitate more nuanced communication, as sign language often involves subtle variations in hand shape and movement that signify different meanings.

3. Device Optimization and Cross-Platform Compatibility:

Expanding compatibility with a wider range of devices, including wearable technologies like smart glasses, could further improve accessibility. Optimization efforts could ensure the system runs smoothly on low-powered devices like older smartphones, making it more accessible to individuals in different socio-economic settings.

4. Support for Conversational AI: By integrating conversational AI, the system could handle continuous or more complex interactions, including detecting multiple gestures and responding contextually in ongoing conversations. This feature would be particularly useful in professional and educational environments, where more dynamic and responsive systems are necessary.

5. Data Privacy and Security Enhancements:

With growing concerns around data privacy, future versions could implement stronger security measures to protect user data, especially in scenarios involving real-time gesture and audio capture. Developing secure offline functionality would enable users to operate the system in areas without internet access, preserving both privacy and accessibility.

6. Real-World Testing and Feedback Integration: Conducting extensive field testing with target user groups could provide valuable insights for refining the system's usability, accuracy, and effectiveness. Feedback-driven modifications based on the experiences of mute and deaf individuals would ensure that the system continues to meet real-world demands and remains user-centered in its design.

6.1 Final Remarks

The proposed sign language detection and conversion system represents a significant step toward enabling inclusive communication for mute and deaf individuals. The future developments outlined above could further enhance the system's performance, usability, and accessibility, making it an essential tool for communication in diverse environments. By continuously improving and expanding the system, we can contribute to a more inclusive world where everyone, regardless of hearing ability, can communicate and interact with ease and independence.

CHAPTER 7

CONCLUSION

This project aims to bridge the communication gap for mute and deaf individuals by developing a real-time sign language detection and conversion system using MediaPipe. The system provides dual functionality: converting sign language gestures into audible speech and translating spoken language into visual sign language.

The use of MediaPipe, a robust framework for hand-tracking, allows the system to detect and interpret hand gestures with high accuracy. By leveraging machine learning models for real-time gesture recognition, the system can produce quick responses suitable for real-world applications. The implementation of natural language processing (NLP) further allows for precise audio-to-sign conversion, supporting deaf individuals by visually representing spoken words. This integrated approach provides an innovative alternative to traditional methods, such as interpreters or text-based communication, and offers a scalable, efficient, and widely accessible solution.

CHAPTER 8

REFERENCES

1. P Sharmila, V Bri and R Jeg. (2023), 'IoT-Oriented Gesture Automation with Mesh Detection through OpenCV and Pyfirmata Protocol using ResNet Mediapipe', Innovations in Power and Advanced Computing Technologies (i-PACT), 2023 IEEE — DOI: 10.1109/I-PACT58649.2023.10434872

2. Athira, P.K., Sruthi, C.J., and Lijiya, A. (2022) 'A signer independent sign language recognition with co-articulation elimination from live videos: An Indian scenario', Journal of King Saud University – Computer and Information Sciences, 34, pp. 771–781. DOI: 10.1016/j.jksuci.2019.05.002.

3. "HAND GESTURE RECOGNITION AND VOICE CONVERSION SYSTEM FOR DUMB AND DEAF PEOPLE", International Journal of Emerging Technologies and Innovative Research (www.jetir.org), ISSN:2349-5162, Vol.6, Issue 4, page no.152-158, April-2019.

4. "AI based Two Way Sign Language System", Proceedings of the Second International Conference on Electronics and Renewable Systems (ICEARS-2023). IEEE Xplore Part Number: CFP23AV8-ART; ISBN: 979-8-3503-4664-0.

5. Sharma, S. and Singh, S. (2021) 'Vision-based hand gesture recognition using deep learning for the interpretation of sign language', Expert Systems with Applications, 162, pp. 113778. DOI: 10.1016/j.eswa.2020.113778 .

6. SakshiSharma,Sukhwinder Singh.,"Vision-based hand gesture recognition using deep learning for the interpretation of sign language", Elsevier,November2022.

7. Md Rashedul Islam, Ummey Kulsum Mitu, Rasel Ahmed Bhuiyan and Jungpil Shin, "Hand Gesture Feature Extraction Using Deep Convolutional Neural Network for Recognizing American Sign Language", 4th International Conference on Frontiers of Signal Processing. Publisher - IEEE, 2020.

8. A. Halder and A. Tayade, "Real-time vernacular sign language recognition using mediapipe and machine learning", Journal homepage, vol. 2582, pp. 7421, 2021, [online] Available: www.ijrpr.com.

9. Abraham, A. and Rohini, V. (2018) 'Real-time conversion of sign language to speech and prediction of gestures using artificial neural network', Procedia Computer Science, 143, pp. 587–594. DOI: 10.1016/j.procs.2018.10.435.