

# **An Effective Communication System For Deaf and Dumb People Using OpenCV and Machine Learning**

Thesis Submitted in fulfillment of the Requirement for the degree

Of

**Bachelor of Technology (B.Tech)**

in

**Department of Computer Science and Engineering**

By

**SUSMITA DEBANTH**

**Roll No: 500120120002 Reg. No: 201430100120007**

Under the guidance of

**Ms. Bidyutmala Saha**

**Assistant Professor, CSE Department**



**Guru Nanak Institute of Technology, Kolkata-700110**

## **ACKNOWLEDGEMENTS**

We would like to pay our heartiest gratitude to our project mentor Ms. Bidyutkala Saha who gave us the opportunity to do this wonderful project “An Effective Communication System for Deaf and Dumb People” which also helped us in doing a lot of research and we came to know about a variety of new things. We would also like to thank her for the constant help and support which help us to complete our project within stipulated period.

The success and outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the completion of our project. All that we have done is only due to such supervision and assistance and we would not forget their kind help.

# **GURUNANAK INSTITUTE OF TECHNOLOGY**

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

---

### **CERTIFICATE**

This is to certify that the project entitled, “**An Effective Communication System for Deaf and Dumb People**” being submitted by **Subhajit Samanata** (roll no-500519011013), **Pritam Sinha** (roll no-500120110005), **Susmita Debnath** (roll no-500120120002), **Pritam Das** (roll no-5005120110011), **Trinita Biswas** (roll no-500120120006), **Shibam Sarkar** (roll no- 500119011119) for the award of the degree of Bachelor of Technology (Computer Science and Engineering) of GNIT, is a record of bonafide research work carried out by his under supervision and guidance. Our group members has worked for nearly one year on the above problem at the Department of Computer Science and Engineering, Gurunanak Institute of Technology, Kolkata and this has reached the standard fulfilling the requirements and the regulation relating to the degree.

The contents of this thesis, in full or part, have not been submitted to any other university or institution for the award of any degree or diploma.

**Supervisor**

-----

**Ms. Bidyutmala Saha, ASST. Professor**

Department of Computer Science and Engineering  
GNIT, Kolkata.

**Head of the Department**

-----

**Suman Bhattacharya**

Head of the Department  
Department of Computer Science and Engineering,  
GNIT, Kolkata

# Contents

Chapter No. Contents	Page No.
1. Abstract	01
2. Introduction	02
3. Literature survey	03-07
4. Problem Statement	08
5. Proposed Solution	09-10
5.1 Methodology	11-18
5.2 Results	18-19
5.3 Observation	20
6. System Model	
6.1 Software and Hardware Requirement	21
6.2 Hardware Requirement	22
7. Implementation	23-29
7.1 Modules	
7.1.1 Sub-Modules	
7.2 Module description	
8. Experimental Results	
8.1 Test Results	30-31
8.2 Screen Shots	32-36
9. Conclusion	37
10.Reference	38

## **ABSTRACT**

The theme of our project is to use technology to improve accessibility by finding a creative solution to benefit the lives of those with a disability. Some of them (nearly 30 million) suffer because they hear at 40 decibels (dB) or higher where normal hearing range is 0 to 20 dB. 70% of families do not sign with their deaf children and they could not receive proper education and remain underemployed or have to quit job due to discrimination or simply remain unemployed. Most of the deaf people around the world use sign language. Among them more than 2 million deaf people use American Sign Language (ASL) as their primary language, which is a standard one-handed sign language can easily be interpreted with normal people and can be implemented for education of deaf people. So, we want to make it easy for over 60 million deaf people across the world to be independent of translator for their daily communication needs, so we designed the app to work as a personal translator 24\*7 for the deaf people.

The main objective of our application is to capture hand gestures of the deaf people using inbuilt camera of handheld devices (Smartphones/tablet/laptop) and take it as input and process the image using OpenCV, creating and drawing hand-landmarks using mediapipe hand-tracking module and convert the data set into NumPy array and compare the array elements with the data set of previously trained model trained by capturing a number of hand gestures of different users as input, processing images, creating hand landmarks and hand skeleton model, converting and creating a database by storing all data sets as NumPy arrays and at last training is done by optimized CNN using LSTM method using Keras and tensorflow module. Comparing the input data with the data sets of trained model, it recognizes the character for a particular sign and display it with the accuracy of matching with trained model. Then interpreted text can be shown to a normal person or convert the text to audible voice using gTTS module of google API service (user must have a working internet connection) to make a better communication with normal people.

We also try to create an effective bi-directional communication. It takes the spoken words of normal people as input and converts them into written text as output. The app makes use of advanced speech recognition technology and natural language processing algorithms to transcribe speech into text with high accuracy. It uses gTTS module of google API service (user must have a working internet connection). The resulting text can be saved, edited and directly share with deaf people or convert them into sign language where saved text string is split, read and an image of sign corresponding to each character in the string is shown on the display and it is done using a database of stock images of signs corresponding to each character installed in the user device.

## **INTRODUCTION:**

There are different kinds of disabilities or disorders in different people. Among these disabilities deaf & dumb is very common disability and also number of people with hearing disability is increasing day to day. Due to these types of disabilities most of the disabled people depend on their parents or guardians. These problems have some solutions which are discussed in this project. If we try to train these disabled people on our proposed application, the life of these disabled people and their guardians will become more comfortable.

Today every laptop, tablet, smart phones have few sensors for example microphone, camera, speaker, digital compass, accelerometer etc. Due to these sensors, programmers develop lot of software. There is lot of software on Speech Recognition. Today very effective and powerful speech recognition applications have been created and developed.

The theme of our project is to use technology to improve accessibility by finding a creative solution to benefit the lives of those with a disability. Some of them (nearly 30 million) suffer because they hear at 40 decibels (dB) or higher where normal hearing range is 0 to 20 dB. Most of the deaf people around the world use sign language and 2 million of them use American Sign Language (ASL) as their primary language, which is a standard one-handed sign language can easily be interpreted with normal people and can be implemented for the education of deaf people. So, we design the app to work as a personal translator 24\*7 for the deaf people.

The main feature of this application is as under:

i) Easy face to face communication between deaf & dumb people and normal people.

ii) Software based writing and internet browsing.

iii) Safety alert email service with location for deaf and dumb people.

Nearly, 30 million people suffer from hearing disability and face tremendous social problem. In this Project, we used technology to improve accessibility of disabled people.

We use hand sign recognition and Speech to Text converter system (using Open CV & Machine Learning) to design the app as a personal translator for deaf people. The application is developed for hand-held devices with inbuilt sensors (laptop/tab/smart phones) and its main features are as under:-

i) Easy face to face communication between deaf & dumb and normal people.

ii) Software based writing and internet browsing.

iii) Safety alert email with location for deaf & dumb people.

## **:LITERATURE SURVEY:**

Introduction: American Sign Language (ASL) is a visual-gestural language used primarily by the deaf and hard-of-hearing community. To bridge the communication gap between ASL users and non-signers, the development of ASL to speech converters and speech to sign image conversion systems has gained significant attention. These systems aim to facilitate real-time communication and enhance accessibility for individuals with hearing impairments. This literature survey explores the research and advancements in ASL to speech conversion and speech to sign image conversion, highlighting the key approaches, techniques, and challenges encountered in this field.

### ASL to Speech Conversion:

1. "Automatic American Sign Language Recognition: A Survey" by Starner et al. (2000) This early survey provides an overview of sign language recognition approaches, including manual annotation, computer vision techniques, and hidden Markov models (HMMs). It discusses the challenges of recognizing ASL due to variations in handshapes, motion dynamics, and the need for large annotated datasets.
2. "Continuous Sign Language Recognition: Approaches and Challenges" by Camgoz et al. (2018) This survey focuses on continuous sign language recognition, exploring various techniques such as dynamic time warping (DTW), neural networks, and HMMs. It discusses the limitations of existing approaches, such as the lack of large-scale annotated datasets and the need for robust real-time recognition systems.
3. "Deep Learning-Based Sign Language Recognition: A Survey and Performance Evaluation" by Pu et al. (2019) This survey reviews deep learning-based approaches for sign language recognition, including convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention mechanisms. It discusses the advantages of deep learning models in capturing spatiotemporal features and highlights the challenges of training data-intensive models.

### Speech to Sign Image Conversion:

1. "Automatic Sign Language Synthesis: A Survey" by Besacier et al. (2010) This survey focuses on speech to sign language synthesis, examining various techniques such as rule-based methods, statistical approaches, and machine learning-based approaches. It discusses the challenges of generating realistic sign language animations and the need for multimodal databases.
2. "Sign Language Recognition, Synthesis, and Translation: An Interdisciplinary Perspective" by Starner et al. (2016) This survey provides a comprehensive overview of sign language recognition, synthesis, and translation. It covers topics such as sensor-based approaches, computer vision techniques, animation synthesis, and sign language corpora. The survey discusses the challenges of real-time translation and the importance of multimodal interaction.
3. "Deep Learning for Sign Language Synthesis: Recent Advances and Future Directions" by Huang et al. (2020) This survey explores deep learning-based approaches for sign language synthesis, including generative adversarial networks

4. (GANs), sequence-to-sequence models, and attention mechanisms. It discusses the advancements in expressive sign synthesis and the challenges of generating natural and context-aware sign animations.

Conclusion: ASL to speech conversion and speech to sign image conversion are crucial areas of research in facilitating communication between ASL users and non-signers. The literature survey highlighted various approaches, including computer vision techniques, statistical methods, and deep learning models, employed in these domains. The challenges encountered include the need for large annotated datasets, real-time recognition and synthesis, and the generation of realistic and expressive sign animations. Future research in this field could focus on developing multimodal systems that integrate speech, vision, and language processing techniques to achieve more accurate and natural communication between ASL users and non-signers.



## **:PROBLEM STATEMENT:**

Developing a machine learning app for ASL (American Sign Language) to speech conversion and vice versa poses the following challenges:

1. **Limited communication accessibility:** Deaf and hearing-impaired individuals who primarily use ASL face barriers in communicating with those who do not understand sign language. There is a need for an efficient and accurate ASL to speech converter that enables seamless communication between these two groups.
2. **Complex hand gesture recognition:** ASL involves a rich vocabulary of hand gestures, finger spelling, and facial expressions, making it a complex language to interpret. Developing robust machine learning algorithms capable of accurately recognizing and translating these intricate gestures poses a significant technical challenge.
3. **Real-time translation:** To facilitate effective communication, the ASL to speech converter needs to operate in real-time, providing near-instantaneous translation of sign language gestures into spoken language and vice versa. Achieving low latency and high translation accuracy simultaneously requires advanced machine learning techniques and efficient system design.
4. **Variability in sign language expressions:** ASL has regional and individual variations, making it necessary to train the machine learning model on diverse sign language samples. The app should be able to handle these variations and adapt to different signers' styles, ensuring accurate and reliable translation across a wide range of users.
5. **User-friendly interface:** The success of the ASL to speech conversion app relies on its usability and accessibility. It should provide an intuitive and user-friendly interface, allowing users to easily input sign language gestures and receive accurate translations in spoken language. The app should also include features such as text display, voice options, and customization settings to cater to the preferences and needs of individual users.
6. **Resource constraints:** To ensure wider adoption and accessibility, the ASL to speech converter app needs to run on various platforms, including smartphones and tablets. Developing an efficient and optimized application that delivers real-time translation performance while considering resource limitations, such as memory and processing power, is essential.

Addressing these challenges will contribute to the development of a powerful and user-friendly machine learning app capable of converting ASL gestures into speech and vice versa, empowering communication and bridging the gap between the deaf and hearing communities.

## **:PROPOSED SOLUTION:**

1. **Introduction:** The proposed solution aims to develop a machine learning-based app for converting American Sign Language (ASL) to speech and vice versa. This app will enable seamless communication between individuals proficient in ASL and those who rely on speech for communication. The following steps outline the proposed solution:
2. **Data Collection:** Collect a large and diverse dataset of ASL videos along with their corresponding spoken phrases. The dataset should cover a wide range of ASL gestures, facial expressions, and variations in speech. Collaborate with ASL experts, interpreters, and individuals fluent in both ASL and spoken language to ensure high-quality data collection.
3. **Data Preprocessing:** Preprocess the collected dataset to extract relevant features for both ASL and speech. This step involves converting ASL videos into sequences of frames and applying computer vision techniques such as image cropping, resizing, and normalization. For speech, utilize audio processing techniques like spectrogram generation, noise reduction, and feature extraction.
4. **Model Training:** Build separate machine learning models for ASL-to-speech conversion and speech-to-ASL conversion. For ASL-to-speech conversion, use a combination of computer vision and natural language processing techniques. For speech-to-ASL conversion, leverage audio processing and computer vision methods. Train the models using deep learning frameworks such as TensorFlow or PyTorch.
5. **User Interface Development:** Create an intuitive and user-friendly interface for the mobile or web app. The interface should allow users to input ASL gestures or spoken language, and display the corresponding speech or ASL translations. Implement features like real-time translation, gesture recognition, and speech synthesis to enhance the user experience.
6. **Integration of ASL Recognition:** Integrate a real-time ASL gesture recognition module into the app using computer vision algorithms. This module should accurately detect and interpret the user's ASL gestures to enhance the translation accuracy and provide immediate feedback.
7. **Accessibility and Customization:** Include accessibility features such as adjustable font sizes, high contrast modes, and support for different languages. Allow users to customize the app's settings according to their preferences, including speech speed, ASL signing style, and background noise filtering.
8. **Testing and Evaluation:** Thoroughly test the app using both simulated and real-world scenarios involving users proficient in ASL and speech. Collect user feedback to identify any issues, improve the translation accuracy, and enhance the overall user experience. Continuously update and refine the models based on user feedback and new data.
9. **Privacy and Security:** Implement robust security measures to protect user data, ensuring compliance with relevant privacy regulations. Encrypt user data both in transit and at rest. Provide clear and transparent policies regarding data collection, storage, and usage.
10. **Continuous Improvement:** Regularly update the app to incorporate advancements in machine learning algorithms and technologies. Leverage user feedback and

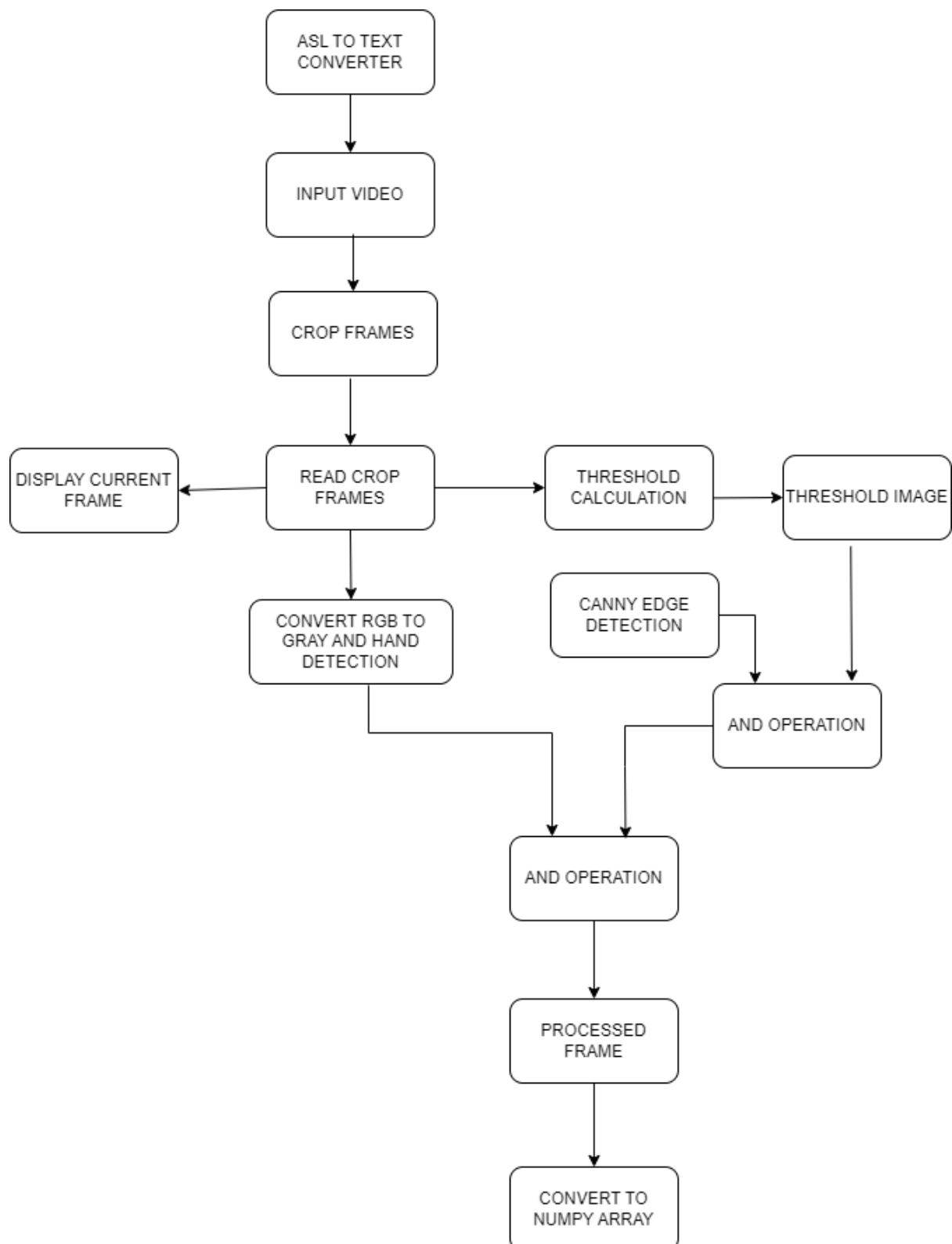
incorporate user suggestions for new features, ensuring the app remains relevant and useful over time.

11. Deployment: Deploy the ASL-to-speech and speech-to-ASL app on mobile platforms (iOS and Android) and/or web platforms, making it easily accessible to a wide range of users. Promote the app through social media, relevant communities, and partnerships with organizations supporting individuals with hearing or speech impairments.

By following these steps, the proposed machine learning app for ASL-to-speech and speech-to-ASL conversion can provide an innovative and accessible solution for facilitating communication between individuals proficient in ASL and those who rely on speech.

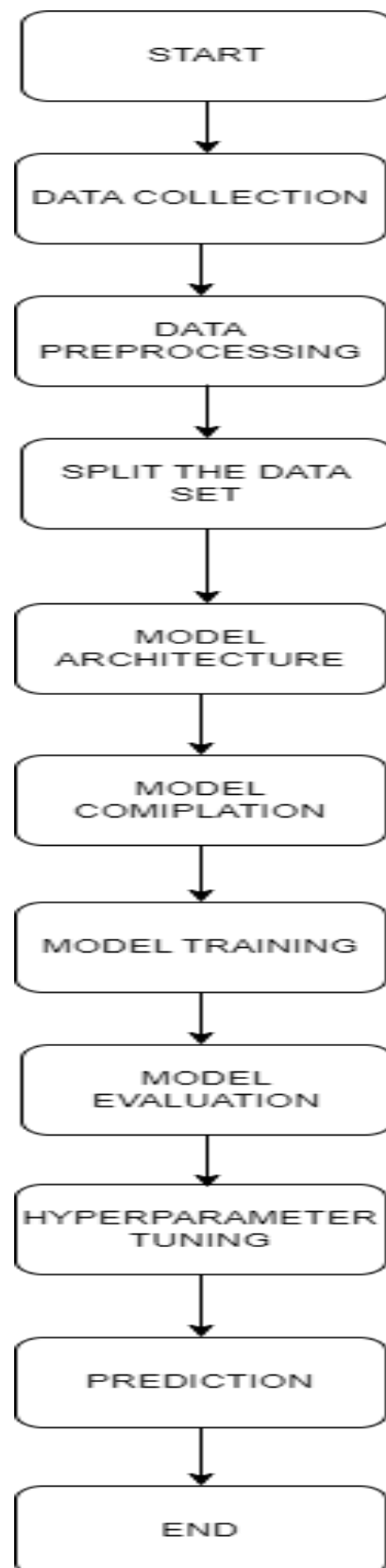
## **:METHODOLOGY:**

- **Data Set Creation:**



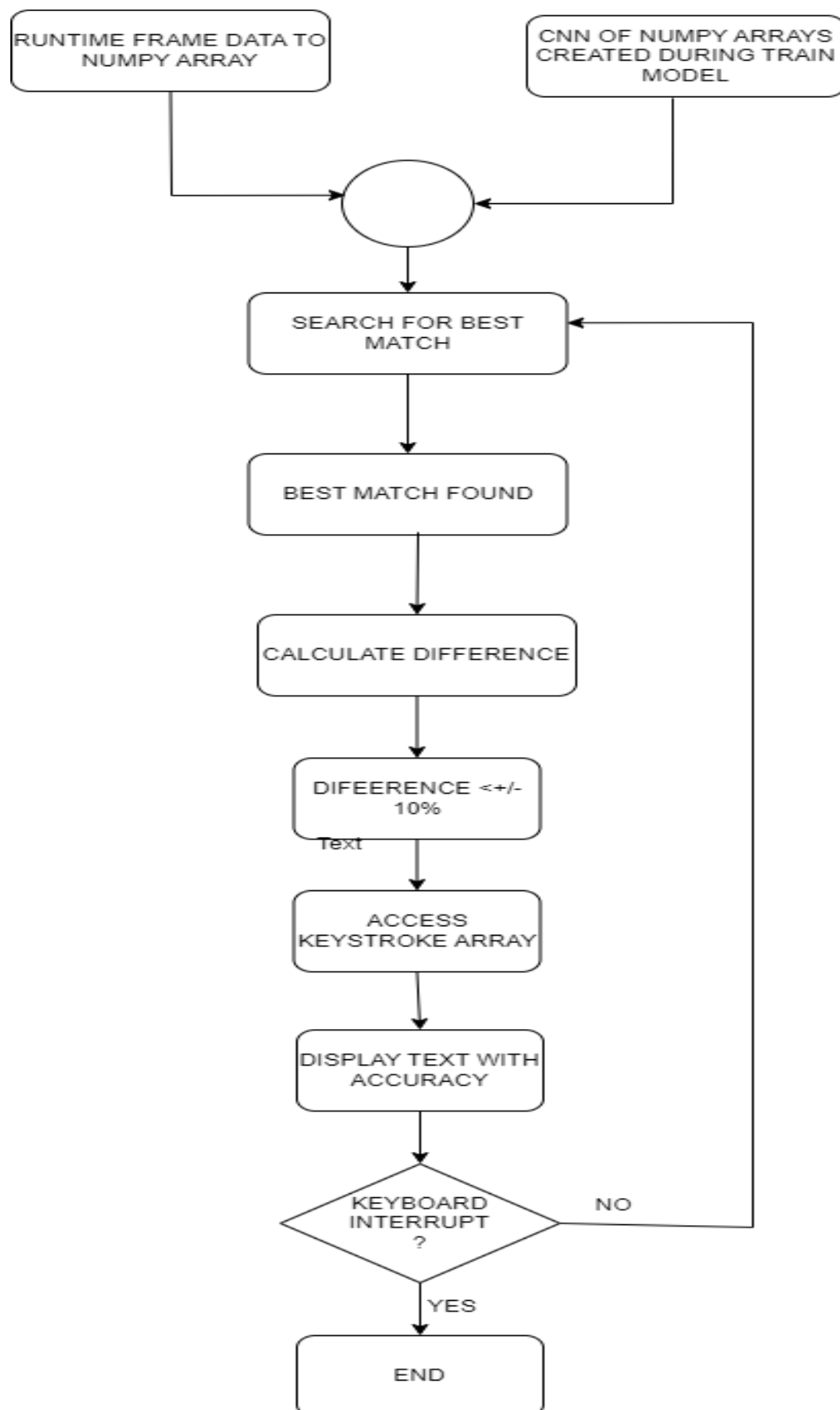
- **Start:** The beginning of the flowchart.
- **Initialize Variables and Settings:** Set up the necessary variables and configure the settings for image capture and feature extraction.
- **Capture Image:** Use OpenCV to capture an image from a camera or load an image from a file.
- **Preprocess Image (Optional):** Apply preprocessing techniques such as resizing, cropping, filtering, or other transformations to enhance the image quality or prepare it for feature extraction.
- **Detect and Extract Features:** Use OpenCV's feature detection and extraction algorithms to identify key features in the image. This step could involve techniques like corner detection, edge detection, or feature matching.
- **Save Extracted Features:** Store the extracted features in a suitable data structure or format for further use. This could involve saving them to a file, a database, or any other storage mechanism.
- **Repeat for Next Image (if any):** If there are more images to process, go back to the "Capture Image" step and repeat the process for the next image. Otherwise, proceed to the next step.
- **End:** The end of the flowchart.

- **Training Model using CNN:**



- Data Collection:
  - Gather a dataset of hand gesture images that includes different gestures you want to recognize.
  - Annotate the images with corresponding labels indicating the gesture represented in each image.
- Data Preprocessing:
  - Resize all the images to a fixed size to ensure consistent input dimensions for the CNN.
  - Normalize the pixel values to a suitable range (e.g., [0, 1]).
- Split the Dataset:
  - Divide the dataset into training, validation, and testing sets.
  - The training set will be used to train the CNN, the validation set to tune hyperparameters, and the testing set to evaluate the final model.
- Model Architecture:
  - Design the architecture of the CNN.
  - Typically, a CNN consists of alternating convolutional layers, pooling layers, and fully connected layers.
  - Experiment with different layer configurations, including the number of filters, kernel sizes, activation functions, and pooling types.
- Model Compilation:
  - Compile the CNN model by specifying an appropriate loss function and optimizer.
  - The choice of loss function depends on the specific problem and the number of classes for gesture recognition (e.g., categorical cross-entropy).
  - The optimizer determines how the model's weights are updated during training (e.g., Adam or RMSprop).
- Model Training:
  - Train the CNN model using the training dataset.
  - During training, feed batches of preprocessed hand gesture images into the model and adjust the weights based on the calculated loss.
  - Iterate over multiple epochs (complete passes through the entire training set) to allow the model to learn from the data.
- Model Evaluation:
  - Evaluate the trained model using the testing dataset.
  - Calculate performance metrics such as accuracy, precision, recall, and F1 score to assess the model's effectiveness in recognizing hand gestures.
- Hyperparameter Tuning:
  - Use the validation set to experiment with different hyperparameters, such as learning rate, batch size, and network depth.
  - Adjust these hyperparameters and retrain the model to find the optimal configuration.
- Prediction:
  - Once the model is trained and evaluated, it can be used to predict the gestures for new unseen images.
  - Pass the preprocessed test images through the model and obtain the predicted gesture labels.

▪ **Output Generation:**

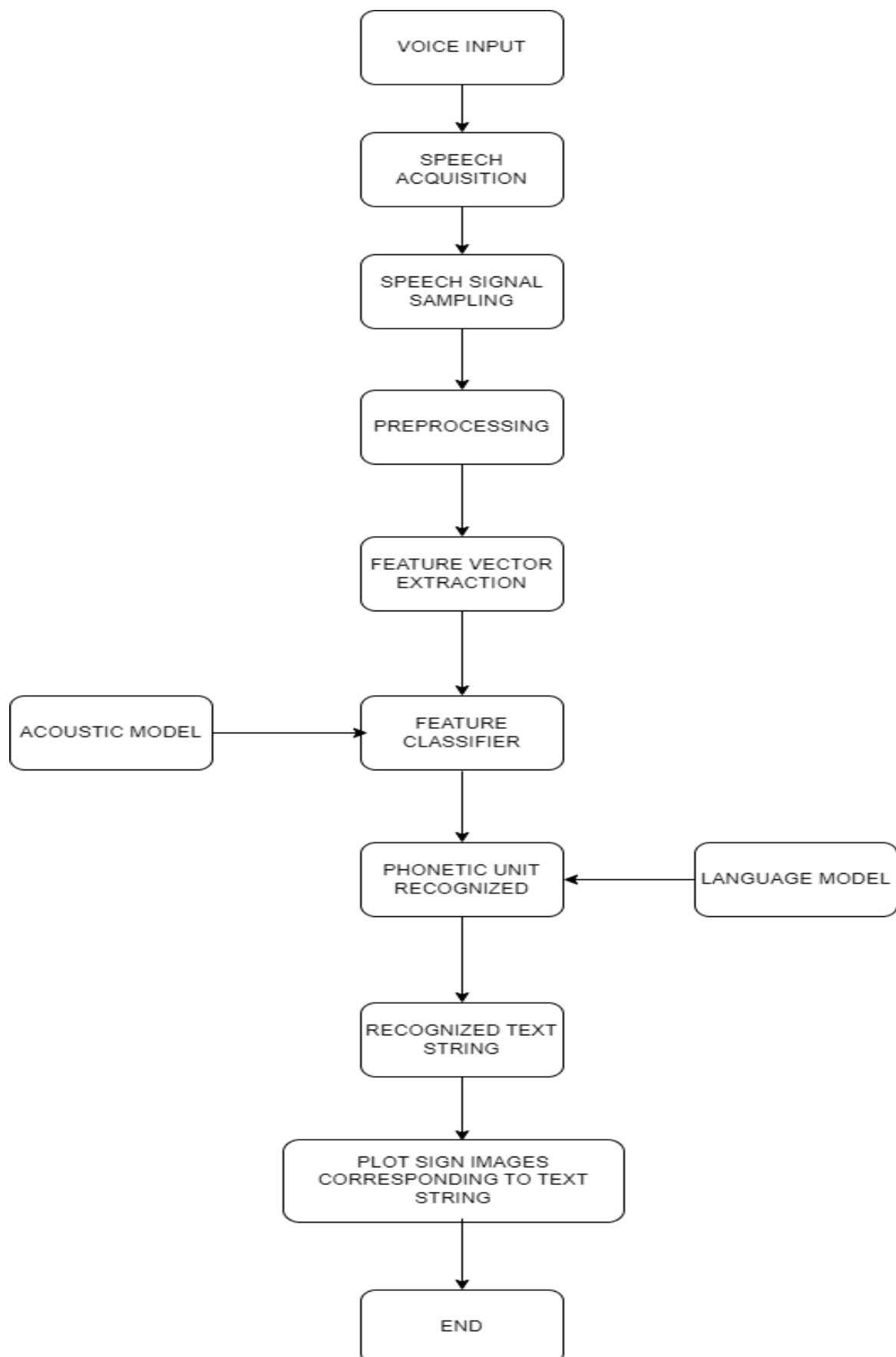




Once the CNN model is trained and evaluated, you can use it to generate outputs for new, unseen data. The steps for output generation depend on the specific application you're working on. For image-related tasks, such as image classification, object detection, or image segmentation, the process typically involves:

- **Input Preparation:** Prepare the input data (e.g., resize, normalize) for the CNN model.
- **Forward Propagation:** Pass the preprocessed input data through the trained CNN model, layer by layer, to obtain the output.
- **Post-processing:** Perform any necessary post-processing on the output, such as applying thresholds, transforming coordinates, or interpreting class probabilities.
- **Output Visualization:** Display or save the final output generated by the CNN model, which could be an image with labeled objects, a classification label, or any other desired form of output.

- **Speech to Sign Image Generation:**



START

1. Prompt the user to speak.
2. Capture the audio input.
3. Pre-process the audio input (if necessary), e.g., noise reduction, normalization.
4. Apply a speech recognition algorithm to convert audio to text.
5. Process the text output (if necessary), e.g., punctuation correction, spell checking.
6. Display the text output to the user.
7. Check if the user wants to continue using the speech-to-text converter.
  - If yes, go to step 1.
  - If no, go to step 8.
8. END

## **:RESULT:**

An ASL (American Sign Language) to Speech converter application is designed to recognize and interpret sign language gestures captured through a camera or input device and convert them into spoken language. The expected result from such an app would be:

1. **Sign Language to Speech Conversion:** When a user performs a sign language gesture in front of the camera or input device, the app should accurately recognize and interpret the gesture, converting it into spoken language. The expected result is a clear and coherent speech output that conveys the meaning of the sign.
2. **Real-Time Translation:** The ASL to Speech converter app should provide real-time translation, allowing users to receive immediate spoken feedback as they perform sign language gestures. This enables efficient communication between deaf or hard-of-hearing individuals who use ASL and those who are unfamiliar with sign language.
3. **Accurate Recognition:** The app should strive for high accuracy in recognizing and interpreting sign language gestures. It should be able to handle a wide range of signs, including alphabet letters, words, phrases, and even complex grammatical structures in ASL.
4. **Intuitive User Interface:** The app should have a user-friendly interface that is easy to navigate and understand. It should provide clear instructions on how to perform different signs and offer visual feedback to users, confirming that their gestures have been recognized correctly.

On the other hand, a Speech to ASL converter application aims to convert spoken language into sign language gestures. The expected result from such an app would be:

1. **Speech to Sign Language Conversion:** When a user speaks into a microphone or inputs text, the app should accurately convert the spoken language into corresponding sign language gestures. The expected result is a clear and understandable representation of the spoken language in ASL.
2. **Real-Time Translation:** The Speech to ASL converter app should provide real-time translation, allowing users to see the sign language gestures being produced as they speak or type. This enables efficient communication between individuals who do not understand sign language and those who use ASL.
3. **Comprehensive Vocabulary:** The app should support a wide range of words, phrases, and sentences in various languages. It should have an extensive vocabulary database and be capable of recognizing and converting both common and specialized terminology.

4. Clear and Coherent Sign Language Output: The app should generate sign language gestures that are accurate, visually clear, and coherent. It should consider the grammatical rules, facial expressions, and body language that are essential for conveying meaning in sign language.

5. User-Friendly Interface: The app should have a user-friendly interface that allows for easy input of spoken language, whether through speech recognition or manual text entry. It should also provide options for customizing the speed, style, or regional variations of the sign language output.

It's important to note that developing such applications requires advanced technologies in computer vision, machine learning, and natural language processing to achieve accurate and reliable conversions between ASL and spoken language.

### **:OBSERVATION:**

1. Accuracy of Translation: The observation would focus on the app's ability to accurately translate American Sign Language (ASL) gestures into spoken language. It would note how well the app recognizes and interprets different ASL signs and conveys them in clear and understandable speech.
2. Speed and Responsiveness: This observation would assess the app's speed and responsiveness in translating ASL to speech. It would look for any delays or lags between the user's gestures and the app's conversion, noting whether the translation is instantaneous or if there is a noticeable delay.
3. Vocabulary and Grammar: The observation would evaluate the app's vocabulary and grammar in speech output. It would assess whether the app provides appropriate and contextually relevant translations, using correct sentence structure and grammar to ensure accurate communication.
4. User Interface and Ease of Use: This observation would focus on the user interface of the ASL to speech converter app. It would assess the app's ease of use, intuitiveness, and overall user experience, taking into account factors such as clear instructions, intuitive gestures, and accessibility options for different users.

#### Observations from a Speech to ASL Converter App:

1. Accuracy of Translation: This observation would evaluate the app's ability to accurately interpret spoken language and convert it into American Sign Language (ASL) gestures. It would assess the app's recognition of different speech patterns, accents, and variations, and its ability to generate appropriate and understandable ASL signs.
2. Speed and Responsiveness: This observation would focus on the app's speed and responsiveness in converting speech to ASL gestures. It would assess whether the app can quickly process spoken input and generate corresponding ASL signs in real-time, without significant delays or interruptions.
3. Facial Expressions and Non-Manual Markers: The observation would evaluate the app's ability to incorporate facial expressions and non-manual markers, which play a crucial role in ASL communication. It would assess whether the app accurately captures the nuances of facial expressions, body movements, and other non-manual elements that enhance the meaning and context of ASL signs.
4. User Interface and Ease of Use: This observation would assess the user interface and overall user experience of the speech to ASL converter app. It would evaluate the app's ease of use, clarity of instructions, and accessibility features to ensure that it caters to a wide range of users, including those with varying levels of ASL proficiency.

It is important to note that these observations may vary depending on the specific features, design, and functionality of the ASL to speech converter app and the speech to ASL converter app.

## **:SYSTEM MODEL:**

### ▪ **Software Requirement:**

- Application File Type : Python File (.py file)
- Application Interface : GUI (Graphical User Interface)
- Language Used : Python
- Python version : 3.10
- Library Packages/ modules Used :
  - i) Open CV-python
  - ii) Media pipe
  - iii) Numpy
  - iv) Keras
- Code Editor Used : Visual Studio Code/ VS Code Editor
- Operating System : Windows 8.1 and above

### ▪ **Hardware Requirement:**

- CPU : 1.6 GHz and above
- RAM : 1 GB and above
- Laptop inbuilt webcam
- Laptop inbuilt microphone

▪ **Design:**

1. User Interface:

- Clean and intuitive user interface that is easy to navigate.
- A video capture area where users can record or stream their ASL gestures.
- Text output area where the translated speech will be displayed.
- Option to switch between different languages for speech output.
- Settings menu to adjust preferences and accessibility options.

2. ASL Gesture Recognition:

- Utilize computer vision algorithms and machine learning models to recognize ASL gestures.
- Use a deep learning model trained on a large dataset of ASL gestures for accurate recognition.
- Display recognized gestures in real-time to provide immediate feedback to the user.
- Highlight recognized gestures with visual cues or labels to assist the user.

3. Speech Synthesis:

- Convert the recognized ASL gestures into textual representation.
- Use a natural language processing (NLP) model to convert the textual representation into spoken words.
- Provide options for different speech synthesis voices and accents.
- Integrate a high-quality speech synthesis engine for natural and intelligible speech output.

4. Translation and Localization:

- Incorporate a translation module to convert ASL gestures into different spoken languages.
- Allow users to select their desired language for speech output.
- Provide options for localized versions of the app to cater to different regions and languages.

5. Accessibility Features:

- Include accessibility options such as adjustable font sizes, color schemes, and contrast settings.
- Provide voiceover support for visually impaired users.
- Support for alternative input methods, such as keyboard input for users with limited mobility.

6. Additional Features:

- Save and replay functionality to review or share previously recorded ASL gestures and translations.
- Social sharing options to easily share translations with others.
- Support for real-time video conferencing or communication to facilitate remote ASL interpretation.

7. Privacy and Data Security:

- Implement strong data security measures to protect user data and ensure privacy.
- Provide transparency to users regarding data handling and storage practices.
- Obtain user consent for data usage and clearly outline the app's privacy policy.



## **:IMPLEMENTATION:**

### ▪ **Step 1 Implementation:**

#### ○ **Image Capture :**

- Importing the necessary libraries: Begin by importing the OpenCV library in your Python script or environment. This is typically done using the `import cv2` statement.
- Initializing the camera or video source: OpenCV provides functions to access various video sources, such as webcams or pre-recorded videos. You can use the `cv2.VideoCapture()` function to initialize a video capture object, specifying the source index or file path.
- Checking the video capture status: Before proceeding, it's essential to verify that the video capture object has been successfully opened. You can use the `isOpened()` method to check if the camera or video source is accessible.
- Capturing frames: Once the video capture object is successfully opened, you can start capturing individual frames from the source. You can use the `read()` method to retrieve a frame from the video source. The returned values include a boolean indicating if the frame was successfully read and the actual frame data.
- Processing frames: You can perform various operations on the captured frames, such as resizing, filtering, or applying computer vision algorithms. OpenCV provides a wide range of functions for image processing and manipulation.
- Displaying frames: To view the captured frames in real-time, you can use the `imshow()` function to display the processed frames in a window. You can also use functions like `waitKey()` to control the display duration and handle user inputs.
- Releasing the video capture object: Once you are done capturing frames or when you want to stop accessing the video source, it's important to release the video capture object using the `release()` method. This frees up system resources and ensures the video source is available for other applications.

It's worth noting that OpenCV provides additional functionalities for image and video processing, such as saving frames to files, accessing specific camera settings, or working with video codecs. These functionalities can be explored in the OpenCV documentation and tutorials to meet specific requirements.

#### ○ **Hand tracking:**

OpenCV (Open Source Computer Vision Library) is a popular open-source library that provides various computer vision and image processing algorithms. Although OpenCV does not have built-in hand tracking algorithms,

it provides a set of tools and functions that can be used to develop hand tracking applications.

Hand tracking algorithms typically involve a combination of computer vision techniques, such as background subtraction, foreground segmentation, feature extraction, and gesture recognition. Here's a general overview of how hand tracking can be achieved using OpenCV:

1. **Background Subtraction:** To isolate the hand from the background, the first step is often to perform background subtraction. This technique involves capturing an initial background image and subtracting it from the current frame to obtain the foreground (hand) pixels.
2. **Foreground Segmentation:** Once the foreground pixels are obtained, further processing is typically done to segment the hand region accurately. Techniques like thresholding, morphological operations (e.g., erosion and dilation), and contour detection can be used to extract the hand region.
3. **Hand Feature Extraction:** After obtaining the hand region, various hand features can be extracted to represent its shape, orientation, and movement. Common features include fingertips, palm center, finger direction, and convex hulls. OpenCV provides functions like contour analysis, convex hull calculation, and bounding box computation to extract these features.
4. **Gesture Recognition:** Once the hand features are extracted, gesture recognition algorithms can be applied to identify specific hand poses or gestures. This can be achieved using techniques such as machine learning, template matching, or rule-based methods. Machine learning approaches, such as using deep neural networks, can be employed for more complex gesture recognition tasks.

It's important to note that hand tracking algorithms can vary significantly depending on the specific requirements and constraints of an application. OpenCV provides a wide range of functions and tools to facilitate the implementation of these algorithms, making it a popular choice among developers in the computer vision community.

Additionally, OpenCV has bindings for various programming languages like Python, C++, and Java, making it accessible and easy to integrate into different projects and platforms. Developers can leverage the library's extensive documentation, tutorials, and community support to implement hand tracking algorithms effectively using OpenCV.

#### ○ **Hand Landmark Detection:**

MediaPipe is an open-source framework developed by Google that provides a pipeline for building real-time multimedia processing applications. One of the features of MediaPipe is hand landmark detection, which involves detecting and tracking the landmarks or keypoints on a human hand.

MediaPipe uses a combination of computer vision and machine learning techniques to perform hand landmark detection. The process involves several steps:

1. **Palm Detection:** The first step is to detect the palm region in the input image or video frame. MediaPipe utilizes a machine learning-based palm detection model to identify the location and orientation of the hand in the image.
2. **Hand Landmark Estimation:** Once the palm is detected, MediaPipe employs a deep neural network (DNN) to estimate the 3D coordinates of the hand landmarks. The network is trained using a large dataset of annotated hand images to learn the relationship between the image pixels and the hand keypoints.
3. **Landmark Refinement and Tracking:** To improve the accuracy and stability of the hand landmarks, MediaPipe applies additional techniques such as temporal smoothing and landmark tracking. Temporal smoothing helps in reducing jitter and noise by considering the history of previously estimated landmarks. Landmark tracking algorithms help maintain the consistency of the landmark positions across consecutive frames, even when the hand undergoes motion or occlusion.
4. **Hand Gesture Recognition:** MediaPipe also provides pre-trained models and algorithms for hand gesture recognition, which can be used in combination with hand landmark detection. These models can recognize various hand gestures or poses based on the detected landmarks, enabling applications to interpret user actions.

Overall, MediaPipe's hand landmark detection algorithms combine deep learning-based approaches with tracking and smoothing techniques to achieve real-time and accurate hand pose estimation. The framework offers a convenient and efficient solution for developers to integrate hand tracking capabilities into their applications.

#### ○ **Hand Skeleton Shape Calculation:**

MediaPipe is a popular open-source framework developed by Google that provides a wide range of solutions for various computer vision tasks, including hand tracking and gesture recognition. When it comes to hand tracking, MediaPipe utilizes a combination of computer vision techniques and machine learning algorithms to estimate the hand skeleton shape.

The hand tracking algorithm in MediaPipe can be broken down into the following steps:

1. **Palm Detection:** The first step is to detect the palm region in the input image or video frame. MediaPipe employs a palm detection model that uses a deep neural network to identify the palm's position and orientation. This model localizes the approximate region where the hand is present.
2. **Hand Landmark Estimation:** Once the palm is detected, the algorithm proceeds to estimate the hand landmarks. MediaPipe employs a machine learning model called BlazePalm, which is trained to predict 2D coordinates of several key points on the hand. These key points include locations such as the fingertips, knuckles, and wrist.
3. **Hand Pose Estimation:** After obtaining the initial hand landmarks, MediaPipe further refines the hand skeleton by employing another machine learning model called BlazePose. This model takes the detected hand

landmarks as input and predicts a more accurate 3D hand pose, including the positions and orientations of the hand joints. The model is trained using a large dataset of annotated hand poses.

4. **Hand Skeleton Calculation:** With the refined hand pose estimation, MediaPipe calculates the hand skeleton shape. The hand skeleton is typically represented as a tree-like structure, where the hand joints are connected by bones. The joint positions and bone connections are determined based on the predicted hand pose.

Overall, the hand skeleton shape calculation algorithms used by MediaPipe involve a combination of deep learning models and computer vision techniques. These models are trained on large datasets and are designed to accurately estimate the positions and orientations of hand landmarks and joints. By leveraging these algorithms, MediaPipe enables real-time hand tracking and gesture recognition, making it useful for a variety of applications, including augmented reality, virtual reality, and sign language interpretation.

## ▪ **Step 2 Implementation:**

### ○ **Creating a CNN:**

To create a Convolutional Neural Network (CNN) using the Long Short-Term Memory (LSTM) method, we need to combine the strengths of both CNNs and LSTMs. The main idea behind this approach is to leverage the ability of CNNs to extract spatial features from images and then use LSTMs to model temporal dependencies within these features. Here's a general overview of the algorithms used in this process:

#### 1. CNN Feature Extraction:

- **Convolutional Layers:** These layers consist of multiple filters that convolve over the input image to extract spatial features. Each filter learns to recognize different patterns or features.
- **Activation Function:** Typically, a non-linear activation function like ReLU (Rectified Linear Unit) is applied after each convolutional operation to introduce non-linearity and increase the network's expressive power.
- **Pooling Layers:** These layers reduce the spatial dimensions of the features while retaining the most important information. Common pooling operations include max pooling or average pooling.

#### 2. LSTM Integration:

- **Reshaping Features:** Since LSTMs operate on sequential data, the output features from the CNN need to be reshaped into a suitable sequence format. This is typically achieved by flattening or reshaping the feature maps into a sequence of vectors.
- **LSTM Layers:** These layers are added after the reshaping step to capture the temporal dependencies within the extracted features. LSTMs have a memory cell that can store and propagate information over time, allowing the model to learn long-term dependencies.
- **Hidden States:** LSTMs generate hidden states at each time step, which contain information about the current input and the previous hidden

state. These hidden states capture the temporal context and are passed to subsequent LSTM layers or other types of layers.

- **Final Output:** The final output of the LSTM layers can be obtained by applying additional fully connected layers or a softmax layer, depending on the specific task (e.g., classification, regression).

### 3. Training and Backpropagation:

- **Loss Function:** The choice of the loss function depends on the task. For example, for image classification, the cross-entropy loss is commonly used.
- **Backpropagation:** The gradients are calculated using the loss function and backpropagated through the LSTM and CNN layers. This step enables updating the weights of the network using optimization algorithms like stochastic gradient descent (SGD) or adaptive methods like Adam.
- **Training Data:** The CNN with LSTM is trained on labeled data, typically represented as input-output pairs. The input can be a sequence of images, and the output can be labels or predictions associated with the sequence.

### 4. Inference and Prediction:

- Given a trained CNN with LSTM, it can be used for making predictions or inferences on new unseen data. The input data is fed through the CNN layers for feature extraction, followed by the LSTM layers for capturing temporal dependencies. Finally, the output layer provides the predicted outputs.

It's important to note that the specific architecture and hyperparameters of the CNN with LSTM can vary based on the task at hand, such as image classification, object detection, or video analysis. Experimentation and fine-tuning may be required to achieve optimal performance.

## ○ **Train the Model:**

Training a hand gesture detection model using Convolutional Neural Networks (CNNs) typically involves several key algorithms and techniques. Here's a high-level description of the common steps involved in training such a model:

**1. Data Collection:** Gather a diverse dataset of hand gesture images or videos. The dataset should include a variety of hand shapes, positions, lighting conditions, and backgrounds. Each image or video should be labeled with the corresponding hand gesture.

**2. Data Preprocessing:** Perform preprocessing steps on the dataset to enhance the quality and standardize the input. Common preprocessing techniques include resizing the images to a consistent size, normalizing pixel values, and applying data augmentation techniques such as rotation, scaling, and flipping to increase the variability of the training data.

**3. Data Splitting:** Divide the dataset into training, validation, and testing sets. The training set is used to optimize the model's parameters, the validation set is used to tune hyperparameters and monitor performance during training, and the testing set is used to evaluate the final model's performance.

**4. Network Architecture:** Design the CNN architecture for hand gesture detection. This typically involves stacking convolutional layers, pooling layers, and fully connected layers. The architecture should be able to learn and extract relevant features from hand gesture images.

**5. Model Training:** Initialize the CNN model's parameters randomly, and then train it using the training dataset. The training process involves feeding input images to the model, computing the output predictions, comparing them with the ground truth labels, and adjusting the model's parameters to minimize the prediction error. This optimization is typically done using gradient-based optimization algorithms such as Stochastic Gradient Descent (SGD) or Adam.

**6. Loss Function:** Define a suitable loss function that measures the dissimilarity between the predicted outputs and the ground truth labels. For hand gesture detection, common loss functions include categorical cross-entropy or mean squared error, depending on the nature of the problem (multi-class or regression).

**7. Hyperparameter Tuning:** Fine-tune the hyperparameters of the CNN model to improve its performance. These hyperparameters include learning rate, batch size, number of layers, filter sizes, and activation functions. This tuning process is typically done using the validation set, and techniques like grid search or random search can be employed.

**8. Regularization:** Apply regularization techniques to prevent overfitting. Regularization methods such as L1 and L2 regularization, dropout, and batch normalization can be used to reduce model complexity and improve generalization on unseen data.

**9. Model Evaluation:** Evaluate the trained model's performance using the testing set. Metrics such as accuracy, precision, recall, and F1 score can be used to assess the model's ability to correctly classify hand gestures.

**10. Model Deployment:** Once the trained model satisfies the desired performance, it can be deployed in real-world applications for hand gesture detection. This involves integrating the model into the target system and

implementing the necessary input/output pipelines to process and interpret hand gesture data.

It's important to note that the specific implementation details and algorithms used can vary depending on the framework or library being used for training the CNN model, as well as the requirements of the hand gesture detection task at hand.

## **:EXPERIMENTAL RESULTS:**

### ▪ **Test Results:**

1. **Gesture Recognition Accuracy:** The app should be tested for its ability to accurately recognize and interpret ASL gestures. It should demonstrate a high success rate in recognizing different signs and accurately mapping them to the corresponding spoken words or phrases.

2. **Translation Precision:** The translated text or spoken output should closely match the intended meaning of the ASL gestures. Test results should reflect the app's ability to capture the nuances and complexities of ASL and produce accurate translations.

3. **Vocabulary Coverage:** The app should be tested with a diverse range of ASL vocabulary, including both common and less frequently used signs. The results should indicate the app's ability to handle various signs effectively and provide accurate translations across different categories, such as everyday conversation, technical terms, or specific domains.

4. **Real-Time Performance:** The app's performance should be evaluated in real-time scenarios, assessing its ability to recognize and interpret ASL gestures swiftly. Test results should indicate minimal delays or lags between the user's gesture and the app's response.

5. **User-Friendly Interface:** The usability of the app is a crucial aspect. Test results should reflect the ease of use, intuitive design, and user-friendliness of the app. It should be accessible to both ASL signers and individuals unfamiliar with ASL, with clear instructions and appropriate visual cues.

6. **Adaptability and Personalization:** The app's ability to adapt to individual users' signing styles and preferences should be evaluated. It should provide options for customization and personalization, allowing users to adjust settings based on their unique signing patterns.

7. **Error Handling and Robustness:** The app should be tested for its ability to handle ambiguous or unclear gestures effectively. It should provide appropriate error messages or prompts to guide users in refining their gestures if necessary. The app should also demonstrate stability and resilience, with minimal crashes or technical issues.

8. **Accessibility Features:** The app's compatibility with accessibility features, such as screen readers or other assistive technologies, should be assessed. It should



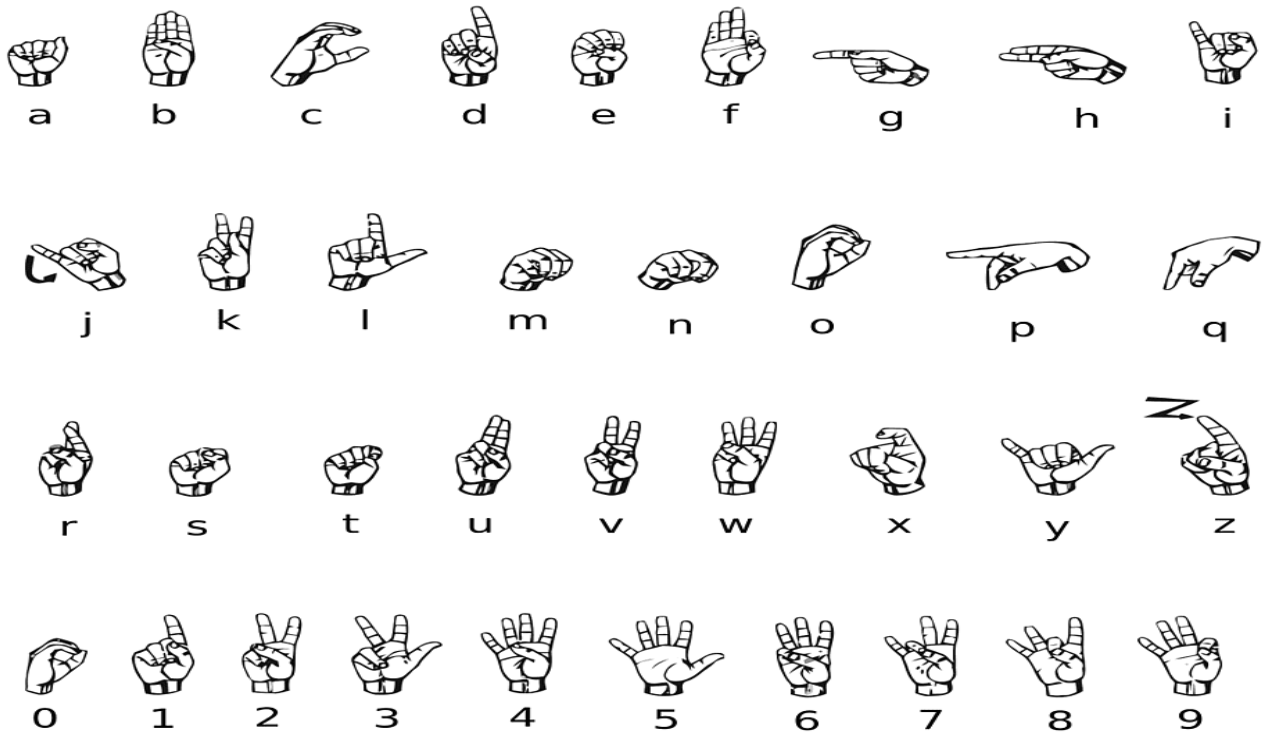
ensure inclusivity and provide a seamless experience for users with visual impairments or other disabilities.

**9. User Satisfaction and Feedback:** User feedback and satisfaction surveys are essential to evaluate the overall performance of the app. Test results should include user opinions and suggestions to help improve the app's functionality and address any potential shortcomings.

These expected test results aim to ensure that the ASL to speech converter app delivers accurate translations, operates smoothly, and meets the needs of its users effectively.

▪ Screenshots:

**HAND GESTURE USED:**



**Best of Luck**



**Angry**



**Drink**



**Happy**



**Hello**



**Help**



**Hungry**



**Okay**

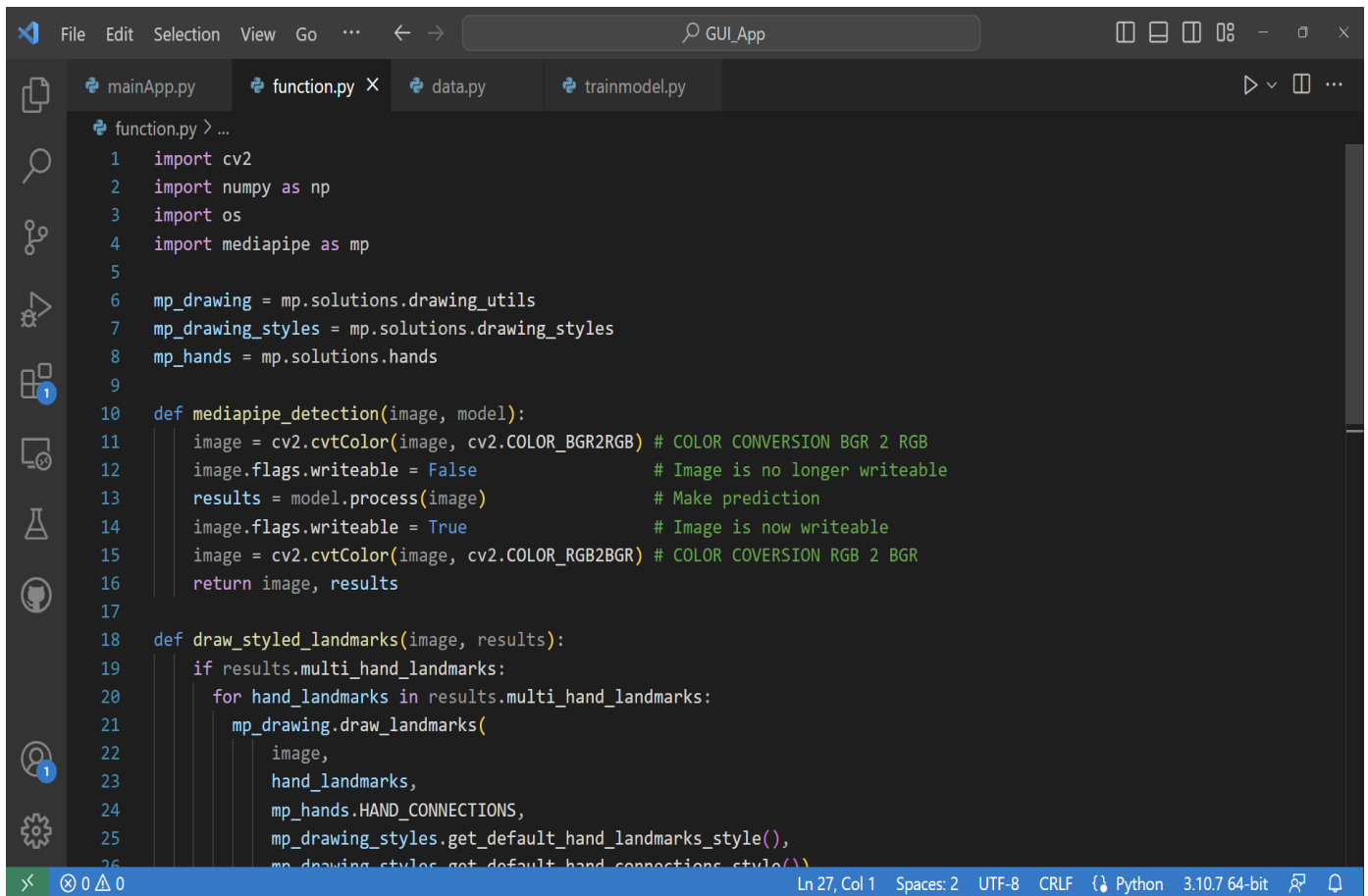


**Sad**



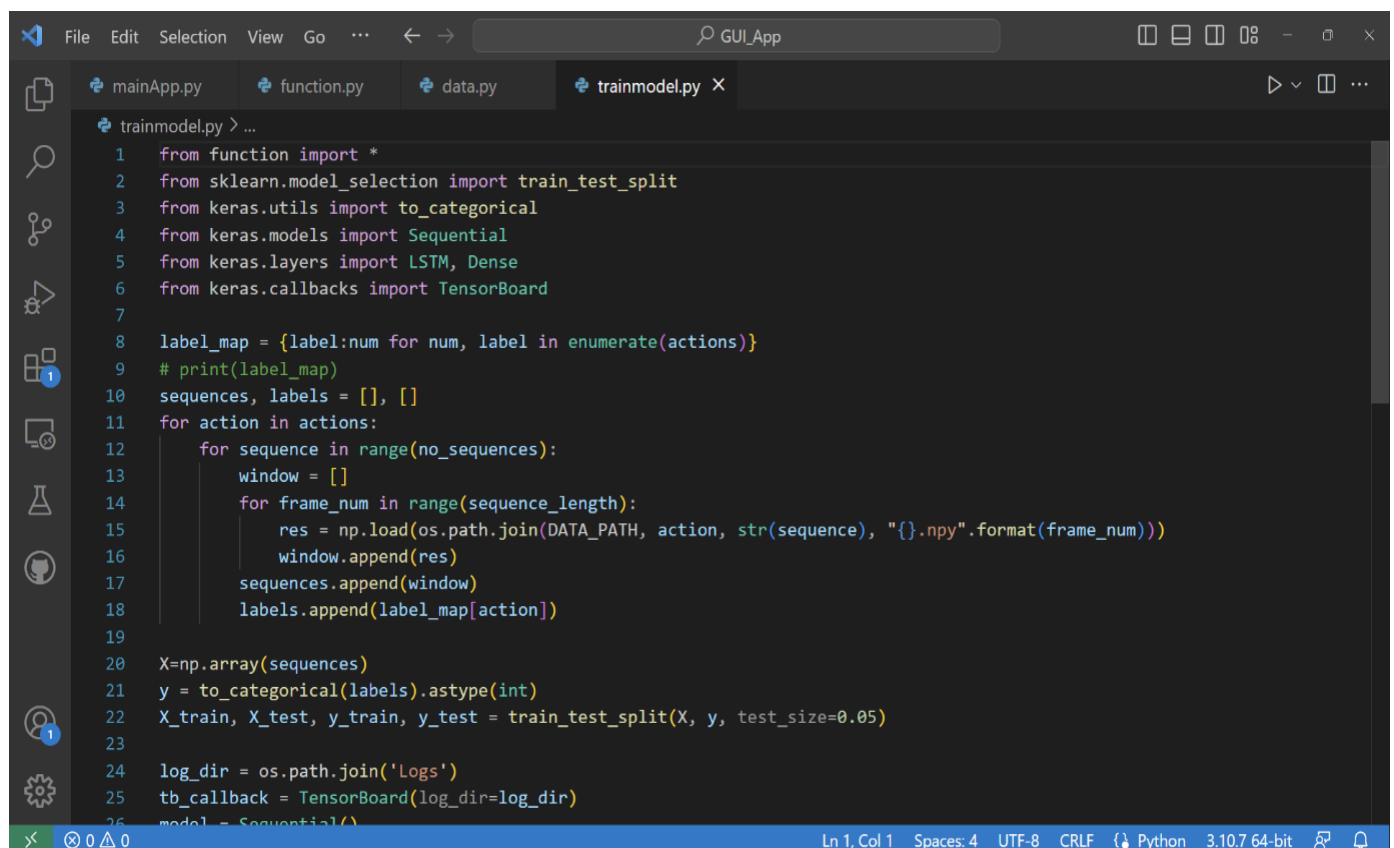
**What**

## SCREENSHOT OF CODE



This screenshot shows the Visual Studio Code editor with the file `function.py` open. The code defines two functions: `mediapipe_detection` and `draw_styled_landmarks`. The `mediapipe_detection` function takes an image and a model as input, converts the image to BGR, processes it with the model, and returns the image with the original color. The `draw_styled_landmarks` function takes an image and the results of the detection, and draws the landmarks on the image.

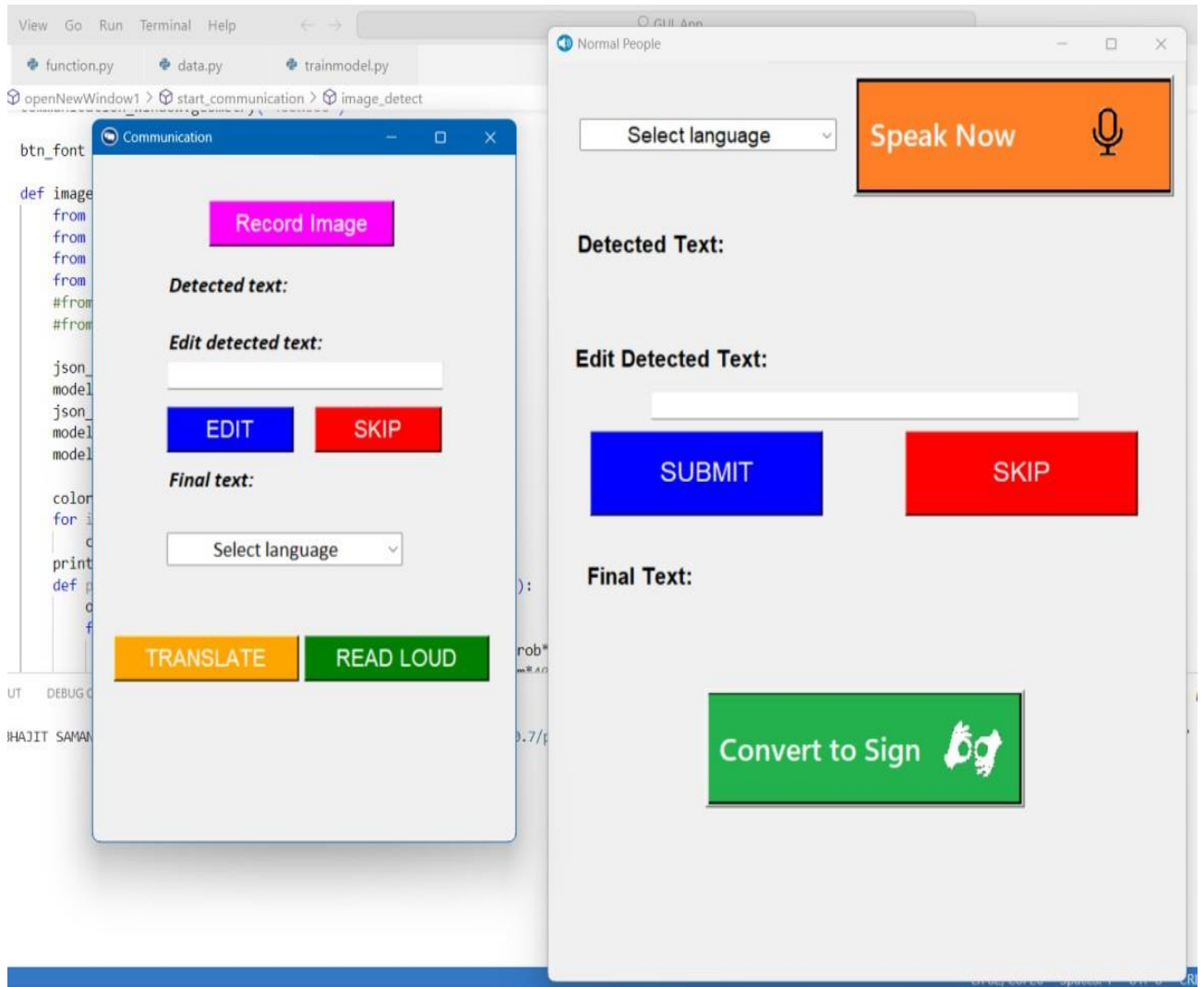
```
1 import cv2
2 import numpy as np
3 import os
4 import mediapipe as mp
5
6 mp_drawing = mp.solutions.drawing_utils
7 mp_drawing_styles = mp.solutions.drawing_styles
8 mp_hands = mp.solutions.hands
9
10 def mediapipe_detection(image, model):
11     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
12     image.flags.writeable = False # Image is no longer writeable
13     results = model.process(image) # Make prediction
14     image.flags.writeable = True # Image is now writeable
15     image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR CONVERSION RGB 2 BGR
16     return image, results
17
18 def draw_styled_landmarks(image, results):
19     if results.multi_hand_landmarks:
20         for hand_landmarks in results.multi_hand_landmarks:
21             mp_drawing.draw_landmarks(
22                 image,
23                 hand_landmarks,
24                 mp_hands.HAND_CONNECTIONS,
25                 mp_drawing_styles.get_default_hand_landmarks_style(),
26                 mp_drawing_styles.get_default_hand_connections_style())
```



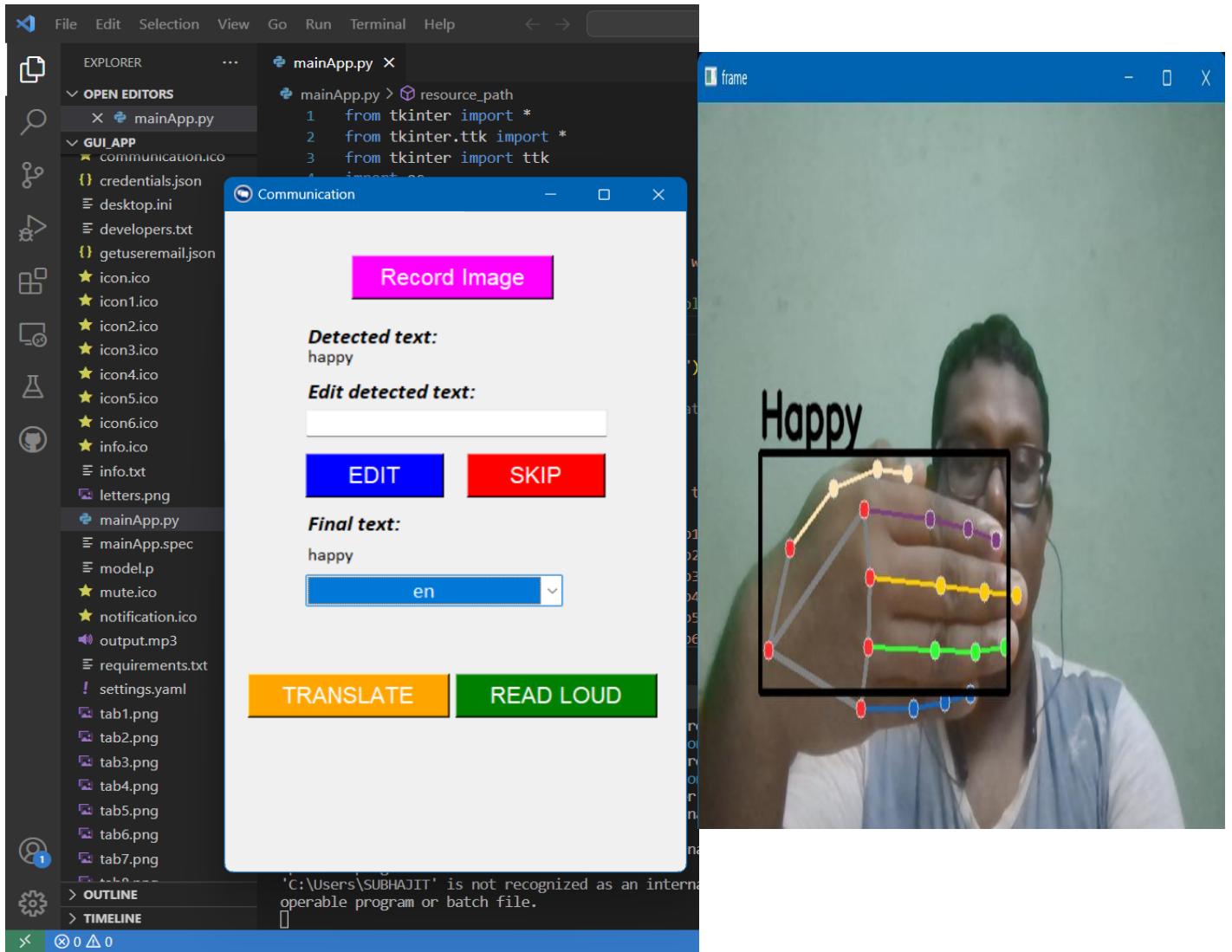
This screenshot shows the Visual Studio Code editor with the file `trainmodel.py` open. The code imports various libraries for machine learning, including `sklearn`, `keras`, and `TensorBoard`. It defines a `label_map` and a `sequences` list. The code then iterates over a range of sequences, loading data from a directory and appending it to the `sequences` list. Finally, it creates a `X` array from the sequences and a `y` array from the labels, and splits them into training and testing sets using `train_test_split`.

```
1 from function import *
2 from sklearn.model_selection import train_test_split
3 from keras.utils import to_categorical
4 from keras.models import Sequential
5 from keras.layers import LSTM, Dense
6 from keras.callbacks import TensorBoard
7
8 label_map = {label:num for num, label in enumerate(actions)}
9 # print(label_map)
10 sequences, labels = [], []
11 for action in actions:
12     for sequence in range(no_sequences):
13         window = []
14         for frame_num in range(sequence_length):
15             res = np.load(os.path.join(DATA_PATH, action, str(sequence), "{}.npy".format(frame_num)))
16             window.append(res)
17         sequences.append(window)
18         labels.append(label_map[action])
19
20 X=np.array(sequences)
21 y = to_categorical(labels).astype(int)
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.05)
23
24 log_dir = os.path.join('Logs')
25 tb_callback = TensorBoard(log_dir=log_dir)
26 model = Sequential()
```

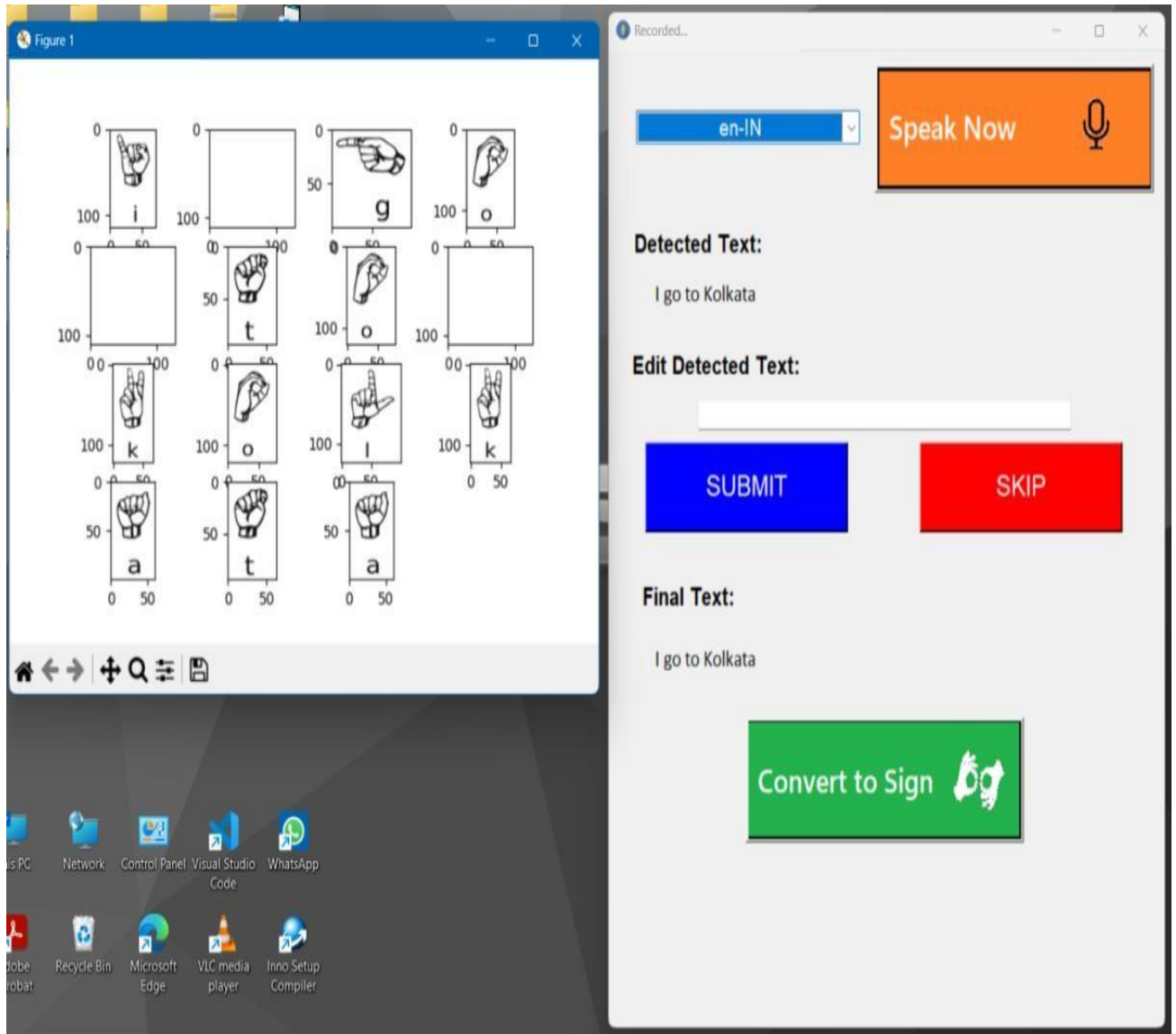
## SCREENSHOT OF USER INTERFACE



## SCREENSHOT OF TEST CASE 1



## SCREENSHOT OF TEST CASE 2



### **:CONCLUSION:**

Currently our application is developed for laptop. This can also be used for Android Smartphone or tablet. The application is dynamic and secure.

This also can be implemented in any Android based IOT device like smart speaker at bank or railway counter to make the service easy for deaf people.

This also can be developed as a module or API and can be implemented as a future of a video chat service, which will facilitate the education of deaf people.

In conclusion, the development of software for effective communication systems for the deaf and dumb is a significant milestone in advancing accessibility and inclusivity. The software can serve as an essential tool to bridge the communication gap between individuals with hearing and speech impairments and the rest of society. With this software, deaf and dumb individuals can now participate in various activities like education, work, and social interactions with greater ease and independence.

### **:REFERENCE:**

[1] The book "Computer Vision: Algorithms and Applications" By Richard Szeliski.

Available: <http://szeliski.org/Book/>

[2] P.A. Ajavon, "An Overview of Deaf Education in Nijeria", Vol. 109, no. 1, pp. 5-10, 2006.

[3] Online source: <https://www.academia.edu>

Articles on "Hand Gesture Recognition and Speech Conversion System for Dumb People."

[4] Kohsheen Tiku, Jayshree Maloo, Aishwarya Ramesh and Indra R. "Realtime Conversion of Sign Language to Text and Speech". Proceedings of the Second International Conference on Inventive Research in Computing Applications (ICIRCA-2020).

available: <http://icirca18.com>

[5] Mangesh B, Mayur K, Rajuli. "Sign Language Text to Speech Converter using Image Processing and CNN." International Research Journal of Engineering and Technology (IRJET) Volume:07 Issue:04|Apr 2020.

available: <https://www.irjet.net>

[6] online source: <http://gizmodo.com/toshibas-eerie-sign-languagerobot-will-silently-stare-1643265457>

[7] Similar apps available in google playstore like "Hand Talk Translator","ASL Pocket Sign","Ace ASL:Learn Fingerspelling". We learned the features and studied the source code of these applications.

Our application has advantage of easy implementation and the feature of face to face communication between deaf and normal people with preferred language can be easily implemented in different fields like AI enabled smart IOT gadgets, video chat and teaching apps with slight modification.