SIGN LANGUAGE TO TEXT AND SPEECH CONVERSION

A Project Report Submitted in partial fulfillment of the requirement for the Award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Y. DURGA NAVEEN 20T91A05A2

Under the Esteemed Guidance of

Mr. K. NagaRaju, M.Tech., (Ph.D.)

Associate Professor

Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GIET ENGINEERING COLLEGE

[Affiliated to JNTUK, Kakinada |Approved by AICTE| Accredited by NAAC A+]
NH-16, CHAITANYAKNOWLEDGE CITY, RAJAMAHENDRAVARAM– 533 296, ANDHRA PRADESH

2023-2024

GIET ENGINEERING COLLEGE

[Affiliated to JNTUK, Kakinada |Approved by AICTE| Accredited by NAAC A+]
NH-16, CHAITANYA KNOWLEDGE CITY, RAJAMAHENDRAVARAM – 533 296,
ANDHRA PRADESH

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that **Y. DURGA NAVEEN(20T91A05A2)** studying in IV B. Tech II Semester of Computer Science and Engineering have submitted their project “SIGN LANGUAGE TO TEXT AND SPEECH CONVERSION” during the academic year 2023-2024 in partial fulfillment of the requirements for the award of degree in Bachelor of Technology, JNTUK, Kakinada. The result embodied in this project has not been submitted to any other University or Institute for the award of degree.

Project Guide
Mr. K. NagaRaju, M.Tech., (Ph.D.)
Associate Professor
Dept of CSE

Head of the Department
Dr. SK. Meera Sharif, M.Tech., Ph.D.
Professor and HoD
Dept of CSE

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

It is a privilege for us to have undertaken the project **"SIGN LANGUAGE TO TEXT AND SPEECH CONVERSION"** using Machine Learning in **GIET ENGINEERING COLLEGE, RAJAMAHENDRAVARAM**.

We avail this opportunity to express our deep sense of gratitude and heart full thanks to **Sri K. SASI KIRAN VARMA**, Vice Chairman of **GIET ENGINEERING COLLEGE, RAJAMAHENDRAVARAM.**

We are thankful to our Principal **Dr. M. VIJAY SEKHAR BABU** and our HOD **Dr. SK. MEERA SHARIF** for encouraging us to do this project.

We are deeply indebted to our Internal Project Guide **Mr. K. NAGARAJU,** Associate Professor in Computer Science and Engineering, **GIET ENGINEERING COLLEGE**, whose motivation in the field of software development made us overcome all the hardships during the course of study.

We are heartily thankful to our Project Coordinator **Mr. J. BALA AMBEDKAR,** Assistant Professor, **GIET ENGINEERING COLLEGE**, for his moral support which was always there to comfort and solace during tough times.

Finally, we would like to thank our **TEACHING AND NON-TEACHING STAFF** whose blessings and encouragement were always there as a source of strength and inspiration. Although the title "Acknowledgement" cannot represent our true feelings for all these persons, we feel very much thankful to all of them and also to our **PARENTS** and **FRIENDS** for encouraging and giving us all the moral support required for making this endeavor a reality.

Y. DURGA NAVEEN **20T91A05A2**

DECLARATION

We hereby declare that this project entitled "**SIGN LANGUAGE TO TEXT AND SPEECH CONVERSION"** submitted to the Department of COMPUTER SCIENCE AND ENGINEERING, GIET ENGINEERING COLLEGE, affiliated to JNTUK, Kakinada, as partial fulfillment for the award of Bachelor of Technology degree is entirely the original work done by us and has not been submitted to any other organization.

Project Member	Pin number	Signature
Y. DURGA NAVEEN	**20T91A05A2**	

ABSTRACT

Sign language is one of the oldest and most natural form of language for communication, hence we have come up with a real time method using neural networks for finger spelling based American sign language. Automatic human gesture recognition from camera images is an interesting topic for developing vision. We propose a Convolution Neural Network (CNN) method to recognize hand gestures of human actions from an image captured by camera. The purpose is to recognize hand gestures of human task activities from a camera image. The position of hand and orientation are applied to obtain the training and testing data for the CNN. The hand is first passed through a filter and after the filter is applied where the hand is passed through a classifier which predicts the class of the hand gestures. Then the calibrated images are used to train CNN. Our project aims to create a computer application and train a model which when shown a real time video of hand gestures of American Sign Language shows the output for that particular sign in text format on the screen.

TABLE OF CONTENTS

LIST OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF KEY WORDS

1.	ASL	American Sign Language
2.	CNN	Convolutional Neural Networks
3.	CONV	Convolutional Layer
4.	D&M	Deaf and Dumb
5.	HCI	Human Computer Interface
6.	ML	Machine Learning
7.	OpenCV	Open Computer Vision
8.	PIL	Python Imaginary Library
9.	RNN	Recurrent Neural Network
10.	ROI	Region Of Interest
11.	SL	Sign Language
12.	SLRS	Sign Language Recognition System
13.	SLTTS	Sign Language To Text and Speech
14.	TTS	Text To Speech

CHAPTER - 1

INTRODUCTION

CHAPTER – 1
INTRODUCTION

1.1 Project Introduction

American sign language is a predominant sign language Since the only disability D&M people have been communication related and they cannot use spoken languages hence the only way for them to communicate is through sign language. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behavior and visuals. Deaf and dumb(D&M) people make use of their hands to express different gestures to express their ideas with other people. Gestures are the nonverbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and dumb people is called sign language.

In our project we basically focus on producing a model which can recognize Fingerspelling based hand gestures in order to form a complete word by combining each gesture.

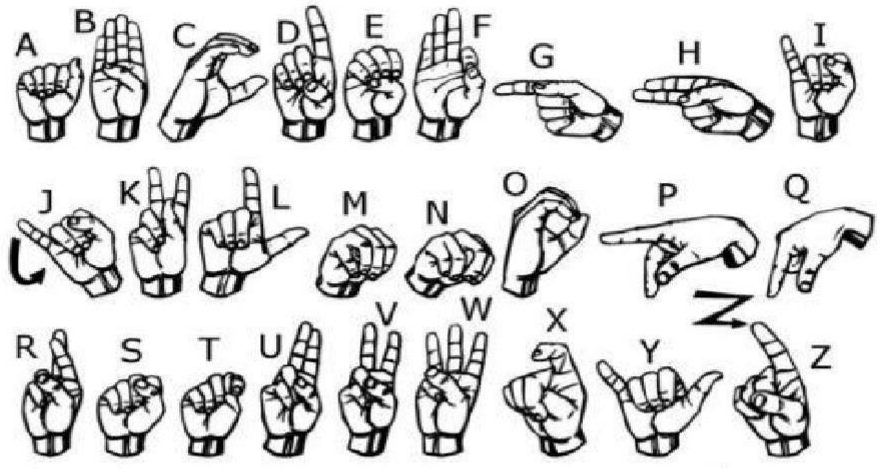


Fig 1.1 ASL

1.1.1 Motivation

For interaction between normal people and D&M people a language barrier is created as sign language structure which is different from normal text. So they depend on vision based communication for interaction. If there is a common interface that converts the sign language to text the gestures can be easily understood by the other people. So research has been made for a vision based interface system where D&M people can enjoy communication without really knowing each other's language. The aim is to develop a user friendly human computer interfaces (HCI) where the computer understands the human sign language. There are various sign languages all over the world, namely American Sign Language (ASL), French Sign Language, British Sign Language (BSL), Indian Sign language, Japanese Sign Language and work has been done on other languages all around the world.

1.2 Literature Survey

In the recent years there has been tremendous research done on the hand gesture recognition. With the help of literature survey done we realized the basic steps in hand gesture recognition are :-

- Data Acquisition
- Data Preprocessing
- Feature Extraction
- Gesture Classification

1.2.1 Data Acquisition:

The different approaches to acquire data about the hand gesture can be done in the following ways:

1.2.1.1 Use of Sensory Devices

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.

1.2.1.2 Vision Based Approach

In vision based methods computer camera is the input device for observing the information of hands 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. These systems tend to complement biological vision by describing artificial vision systems that are implemented in software and/or hardware. The main challenge of vision-based hand detection is to cope with the large variability of human hand's appearance due to a huge number of hand movements, to different skin-colour possibilities as well as to the variations in view points, scales, and speed of the camera capturing the scene.

1.2.2 Data Preprocessing and Feature Extraction for Vision Based Approach

- In the approach for hand detection combines threshold-based color detection with background subtraction. We can use Adaboost face detector to differentiate between faces and hands as both involve similar skin-color.

- We can also extract necessary image which is to be trained by applying a filter called Gaussian blur. The filter can be easily applied using open computer vision also known as OpenCV.

- For extracting necessary image which is to be trained we can use instrumented gloves as. This helps reduce computation time for preprocessing and can give us more concise and accurate data compared to applying filters on data received from video extraction.

- We tried doing the hand segmentation of an image using color segmentation techniques but as mentioned in the research paper skin color and tone is highly dependent on the lighting conditions due to which output we got for the segmentation we tried to do were no so great. Moreover we have a huge number of symbols to be trained for our project many of which look similar to each other like the gesture for symbol 'V' and digit '2', hence we decided that in order to produce better accuracies for our large number of symbols, rather than segmenting the hand out of a random background we keep background of hand a stable single color so that we don't need to segment it on the basis of skin color . This would help us to get better results.

1.2.3 Gesture Classification

- In Hidden Markov Models (HMM) is used for the classification of the gestures .This model deals with dynamic aspects of gestures. Gestures are extracted from a sequence of video images by tracking the skin-colour blobs corresponding to the hand into a body– face space centered on the face of the user. The goal is to recognize two classes of gestures: deictic and symbolic. The image is filtered using a fast look–up indexing table. After filtering, skin colour pixels are gathered into blobs. Blobs are statistical objects based on the location (x,y) and the colourimetry (Y,U,V) of the skin colour pixels in order to determine homogeneous areas.

- In Naïve Bayes Classifier is used which is an effective and fast method for static hand gesture recognition. It is based on classifying the different gestures according to geometric based invariants which are obtained from image data after segmentation. Thus, unlike many other recognition methods, this method is not dependent on skin colour. The gestures are extracted from each frame of the video, with a static background. The first step is to segment and label the objects of interest and to extract geometric invariants from them. Next step is the classification of gestures by using a K nearest neighbor algorithm aided with distance weighting algorithm (KNNDW) to provide suitable data for a locally weighted Naïve Bayes‟ classifier.

- According to paper on "Human Hand Gesture Recognition Using a Convolution Neural Network" by Hsien-I Lin , Ming-Hsiang Hsu, and Wei-Kai Chen graduates of Institute of Automation Technology National Taipei University of Technology Taipei, Taiwan, they construct a skin model to extract the hand out of an image and then apply binary threshold to the whole image. After obtaining the threshold image they calibrate it about the principal axis in order to center the image about it. They input this image to a convolutional neural network model in order to train and predict the outputs.

CHAPTER -2

SYSTEM ANALYSIS

CHAPTER – 2
SYSTEM ANALYSIS

2.1 Existing System

2.1.1 Sign Language Recognition System

Sign language popularity is a vital software of gesture popularity. Sign language popularity has distinct processes.

– Glove based approaches

– Vision based approaches.

2.1.2 Glove Based Approaches

This class calls for signers to put on a sensor glove or a coloured glove. The challenge could be simplified throughout the segmentation procedure via the means of wearing gloves. The disadvantage of this technique is that the signer has to put on the sensor hardware together with the glove throughout the operation of the system.

2.1.3 Vision Based Approaches

Image processing algorithms are utilized in Vision primarily based totally on the approach to locate and tune hand symptoms and symptoms and facial expressions of the signer. This approach is less complicated to the signer for the reason that there isn't any want to put on any more hardware. However, there are accuracy issues associated with photograph processing algorithms and those issues are but to be modified.

There are once more distinct processes in imaginative and prescient primarily based totally sign language popularity:

-3-D model based

-Appearance based

3-D version primarily based totally techniques employ 3-D statistics of key factors of the frame components. Using this statistics, several vital parameters, like palm position, joint angles etc., can be obtained. This technique makes use of volumetric or skeletal models, or an aggregate of the Volumetric technique is higher acceptable for pc animation enterprise and pc imaginative and prescient. This technique may be very computational extensive and also, structures for stay evaluation are nonetheless to be developed. Appearance-primarily based total structures use pix as inputs.

They are derived at once from the pix or movies the use of a template database. Some templates are the deformable 2D templates of the human components of the frame, mainly hands. The units of points at the define of an item referred to as deformable templates. It is used as interpolation nodes for the items defining approximation.

Existing system are struggle with complex signs, variations in signing styles, and real-time processing requirements. The existing system have been able to recognize gestures with high latency as it uses only image processing. Accuracy remains a challenge, especially in complex sentences with rapid signing and background noise.

2.1.4 Disadvantages

1. Limited Training Data: Training CNNs requires a large amount of labeled sign language data, which can be scarce for lesser-used sign languages.

2. Integration with Sign Language Grammar: Sign languages have their own grammar beyond just hand signs. A system that only translates individual signs might miss the nuances of the language.

3. Complexity of Sign Languages: Sign languages are rich with variations in hand shape, location, movement, facial expressions, and even body posture. CNNs can struggle to capture all these subtleties, leading to misinterpretations.

2.2 Proposed system

2.2.1 Image Capturing From Camera:

The net digital digicam is used to capture the motions. The complete signing length is captured using this OpenCV video flow. Frames from the flow are retrieved and transformed to grayscale images.

2.2.2 Hand Gesture Scan:

Hand motions are scanned withinside the accrued images. This is a step withinside the preprocessing process that happens earlier than the photograph is fed into the version for prediction. The passages which include gestures have been amplified. This multiplies the chance of an accurate forecast through an element of ten.

2.2.3 Hand Posture Recognition:

The Keras CNN version is fed the preprocessed pictures. The projected label is generated with the aid of using the version that has already been trained. All of the gesture labels have a chance related to them. The anticipated label is decided with the aid of using the label with the very best likelihood.

2.2.4 Gesture Classification:

To expect the very last image of the user quite a few symbols that offer comparable effects are detected, the classifiers designed specially for the ones units to categorize among them are used.

2.2.5 Text And Speech Translation:

The version translates recognised gestures into phrases the gtts library is used to convert the identified words into the precise speech the text-to-speech output is a easy workaround however it is a useful function as it simulates a real-life dialogue.

Our proposed system is sign language recognition system using conventional neural networks which recognizes various hand gestures by capturing video and converting it into frames. Then the hand pixels are segmented and the image it obtained sent for comparison to the trained model. Thus our system is more robust in getting exact text labels of letters. The purpose of the system is to improve the existing system in terms of response time and accuracy with the use of efficient algorithms, high quality data sets and better sensors.

2.2.6 Advantages

- **Accuracy:** The Accuracy rate of translation is high and is understood by common people which will help them to communicate without any interpreter.
- **Accessibility:** Sign language to text conversion enhances access to information for the deaf, promoting inclusivity in various aspects of life.
- **Communication Bridge:** It acts as a bridge, facilitating communication between those fluent in sign language and those who are not, fostering understanding.
- **Educational Support:** Enables real-time transcription, aiding deaf students in accessing educational content effectively.
- **Inclusive Technology:** Contributes to inclusive technologies, reducing communication barriers for individuals with hearing impairments.
- **Employment Opportunities:** Enhances job prospects for the deaf by facilitating effective communication in the workplace.
- **Enhanced Independence:** Provides independence by enabling access to information and participation in various situations.

- **Legal Compliance:** Meets legal requirements for accommodations, ensuring equal opportunities for individuals with disabilities.
- **Social Inclusion:** Promotes social inclusion by fostering effective communication and diversity in society.

2.3 System Study

2.3.1 Economical Feasibility

The economic feasibility of this project depends on several factors, and while there are significant cost-saving aspects, there are also ongoing expenses to consider. Here's a breakdown:

2.3.1.1 Cost-Saving Factors:

- **Open-Source Tools:** Development can utilize free and open-source libraries like TensorFlow eliminating licensing fees for commercial software.
- **Hardware Availability:** Powerful hardware with GPUs can accelerate training significantly, but the system can potentially run on moderately resourced devices with advancements in model optimization. This reduces reliance on expensive hardware for deployment.

2.3.1.2 Benefits:

- **Reduced reliance on sign language interpreters:** This can lead to significant cost savings in situations where interpreters are currently needed, such as education, healthcare, and legal settings.
- **Increased accessibility:** The technology can improve communication and participation for deaf and hard-of-hearing individuals in various aspects of life.

2.3.2 Technical Feasibility

2.3.2.1 Challenges:

- **Sign Language Complexity:** Sign languages are complex, with variations in hand shapes, facial expressions, and body movements. Capturing these nuances and differentiating between similar signs can be challenging for CNNs.
- **Limited Data Availability:** Large datasets of labeled sign language data are scarce, especially for less common sign languages. This can limit the accuracy and generalizability of the model.
- **Real-time Processing:** Achieving real-time sign language translation with high accuracy requires significant computational resources and optimization techniques.

2.3.2.2 Advantages of CNNs:

- **Image Recognition Expertise**: CNNs excel at image recognition tasks, making them well-suited for identifying hand shapes and poses in sign language videos.
- **Learning from Data:** CNNs can learn complex patterns from large datasets, potentially achieving high accuracy over time.

Technical feasibility is high, but ongoing research is needed to address challenges related to sign language complexity, limited data, and real-time processing demands.

2.3.3 Social Feasibility

2.3.3.1 Benefits:

- **Improved Communication:** The project can bridge communication gaps between deaf and hard-of-hearing individuals and the hearing population.
- **Social Inclusion:** Sign language translation can promote greater social inclusion for deaf and hard-of-hearing people in various social settings.
- **Language Preservation:** Technology can aid in documenting and preserving lesser-known sign languages.

2.3.3.2 Challenges:

- **Acceptance within Deaf Community:** Some members of the deaf community may prefer sign language over spoken language and might be hesitant to adopt new technology.
- **Accuracy Concerns:** Inaccurate translations can lead to misunderstandings and frustration.

Social feasibility is good. The project has the potential to significantly improve communication access for deaf and hard-of-hearing individuals. However, it's crucial to involve the deaf community in the development process to address concerns and ensure the technology is culturally appropriate.

CHAPTER - 3

REQUIREMENT SPECIFICATION

CHAPTER – 3
REQUIREMENT SPECIFICATION

3.1 Hardware Requirements:

System	:	Dual Core
Hard Disk	:	250 GB
Ram	:	8 GB
Camera	:	Webcam

3.2 Software Requirements:

Operating System	:	Windows 10
Coding Language	:	Python3.11
IDE	:	VS CODE
Libraries	:	OpenCV, NumPy, Keras, Mediapipe, TensorFlow, Tkinter, pyttsx3, PIL.

CHAPTER - 4

TECHNOLOGIES USED

CHAPTER – 4

TECHNOLOGIES USED

4.1 Python

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.

Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edittest- debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

4.2 Developing Environment

4.2.1 Hardware and Software:

- **Computer:** Any computer with decent processing power will suffice. While a GPU can accelerate training, a CPU can still handle the task, especially for smaller datasets.
- **Operating System:** Popular choices include Windows, macOS, or Linux.

4.2.2 Python Environment:

4.2.2.1 Python Programming Language: Python is widely used for machine learning projects due to its simplicity and the availability of numerous libraries. Download and install the latest stable version of Python. (https://www.python.org/downloads/)

1. Machine Learning Libraries: Libraries such as TensorFlow, Keras are essential for implementing neural networks and machine learning algorithms.

2. OpenCV: This library is crucial for image processing tasks, such as capturing and preprocessing sign language images.

3. Dataset: Gather a dataset of sign language gestures, Label the dataset, assigning appropriate class labels to each sign gesture to indicate the corresponding sign being performed

4. Data Preprocessing: Preprocessing techniques like image normalization, resizing, and data augmentation might be necessary to improve model performance.

5. Convolutional Neural Networks (CNN): CNNs are particularly effective for image-related tasks. we design and train a CNN model to recognize sign language gestures.

6. Text to Speech (TTS) Library: Library like pyttsx3 can convert text into speech.

4.2.3 Project Structure:

Organize your project with a clear directory structure to keep things well-managed:

```
Project name/
  |- dataset/  # Stores your sign language image dataset
  |- models/  # Saves trained CNN models
  |- main.py  # Script to handle real-time video capture and processing
  |- requirements.txt  # Lists project dependencies for easy installation
```

4.3 Libraries

4.3.1 TensorFlow :

Tensorflow is an open source software library for numerical computation. First we define the nodes of the computation graph, then inside a session, the actual computation takes place. TensorFlow is widely used in Machine Learning and Artificial Intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

4.3.2 Keras :

Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

4.3.3 OpenCV :

OpenCV(Open Source Computer Vision) is an open source library of programming functions used for real-time computer-vision. It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB/OCTAVE.

4.3.4 NumPy:

NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

4.3.5 MediaPipe:

MediaPipe is an open-source framework created by Google for building machine learning pipelines. It simplifies the process of creating on-device machine learning applications especially for computer vision tasks. Here's a breakdown of what MediaPipe offers:

- **Simplified Machine Learning on Devices:** MediaPipe handles the complexities of making machine learning models work on devices like phones and cameras. It takes care of optimizing the models for different hardware while maintaining accuracy.
- **Pre-built Solutions:** MediaPipe comes with a set of pre-built solutions for common tasks like hand tracking, pose estimation, face detection, and object detection. These solutions can be easily integrated into your applications.
- **Customization:** If the pre-built solutions aren't exactly what you need, MediaPipe allows you to customize them or build your own pipelines using its framework.
- **Cross-platform Deployment:** MediaPipe applications can be deployed on various platforms including mobile (Android and iOS), web, desktop, and even Internet of Things (IoT) devices.

4.3.6 Tkinter:

Tkinter is a graphical user interface (GUI) toolkit for Python. It's the standard library option for creating GUIs in Python applications.

- **Built-in:** Tkinter comes bundled with Python by default on most operating systems (Windows, macOS, Linux), making it readily available for use without needing external installations.
- **Simple and Easy to Learn:** Tkinter is considered beginner-friendly due to its relatively straightforward syntax and readily available documentation.
- **Cross-platform:** Applications built with Tkinter will run on different platforms without

needing major modifications, which is a plus for wider compatibility.

- **Widgets and Layouts:** Tkinter offers a variety of widgets like buttons, labels, text boxes, and more, to build the visual elements of your GUI. It also provides layout managers to organize these widgets within the application window.

4.3.7 Pyttsx3:

pyttsx3 is a powerful and versatile Python library that enables you to convert text into spoken audio output. It provides an interface to various text-to-speech (TTS) engines on your system, allowing you to leverage their capabilities within your Python applications.

4.3.8 Enchant:

Enchant is a library that allows you to check the spelling of words and suggest corrections for misspelled words. It can utilize popular spellchecking packages like ispell, aspell, and MySpell.

4.3.9 Python Imaging Library (PIL):

This is a free and open-source library used for image processing tasks in Python. It provides a user-friendly interface for various image manipulation operations, making it popular for image editing, format conversions, and other graphical applications.

CHAPTER - 5

SYSTEM DESIGN

CHAPTER – 5

SYSTEM DESIGN

5.1 Modules

5.1.1 Data Acquisition:

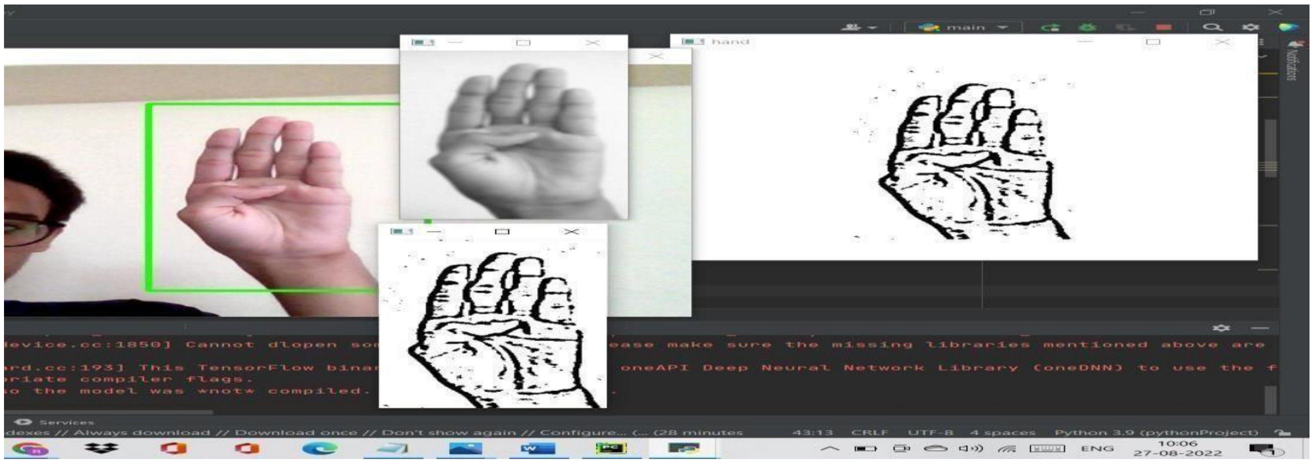
The different approaches to acquire data about the hand gesture can be done in the following ways:

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.

5.1.2 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. We have collected images of different signs of different angles for sign letter A to Z.



```
evice.cc:1850] Cannot dlopen so                ease make sure the missing libraries mentioned above are
rd.cc:193] This TensorFlow binar               oneAPI Deep Neural Network Library (oneDNN) to use the f
riate compiler flags.
o the model was *not* compiled.
```

Services
dexes // Always download // Download once // Don't show again // Configure... (... (28 minutes
43:13 CRLF UTF-8 4 spaces Python 3.9 (pythonProject)
ENG
10:06
27-08-2022

Fig 5.1.1 Hand gesture of ROI

In this method there are many loop holes like your hand must be ahead of clean soft background and that is in proper lightning condition then only this method will give good accurate results but in real world we don’t get good background everywhere and we don’t get good lightning conditions too.

So to overcome this situation we try different approaches then we reached at one interesting solution in which firstly we detect hand from frame using mediapipe and get the hand landmarks of hand present in that image then we draw and connect those landmarks in simple white image.

5.1.2.1 MediaPipe landmark System:

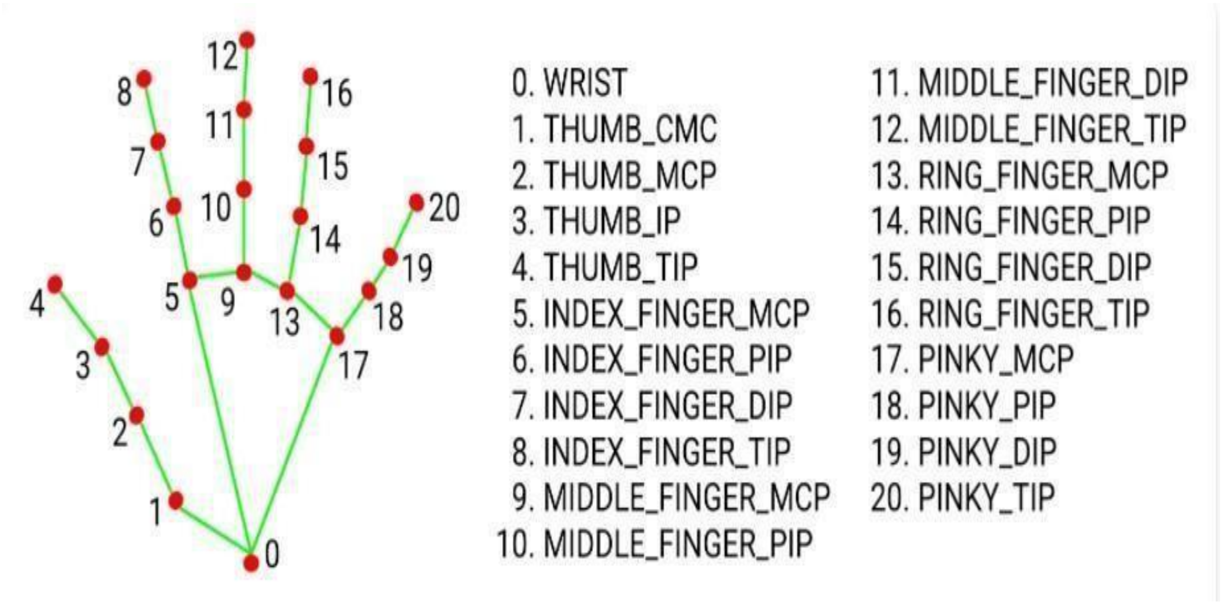


Fig 5.1.2 Mediapipe Landmark System

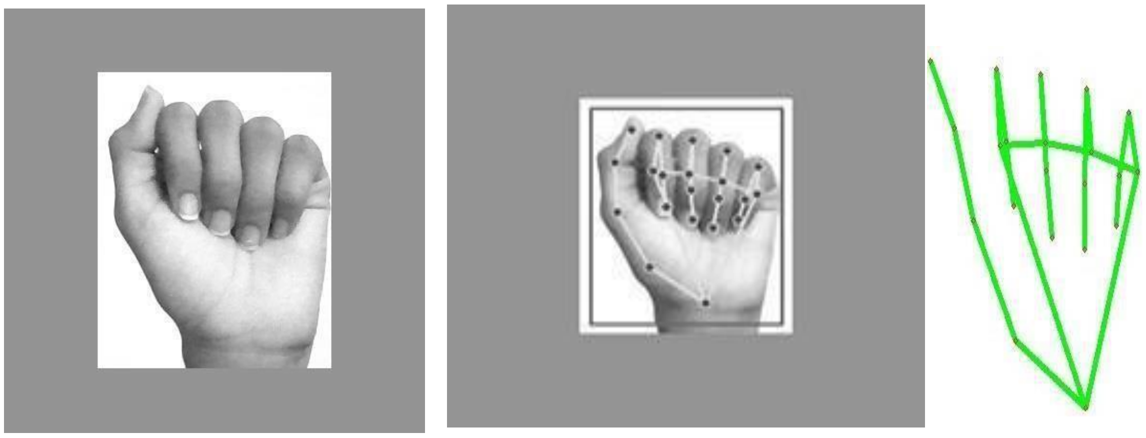
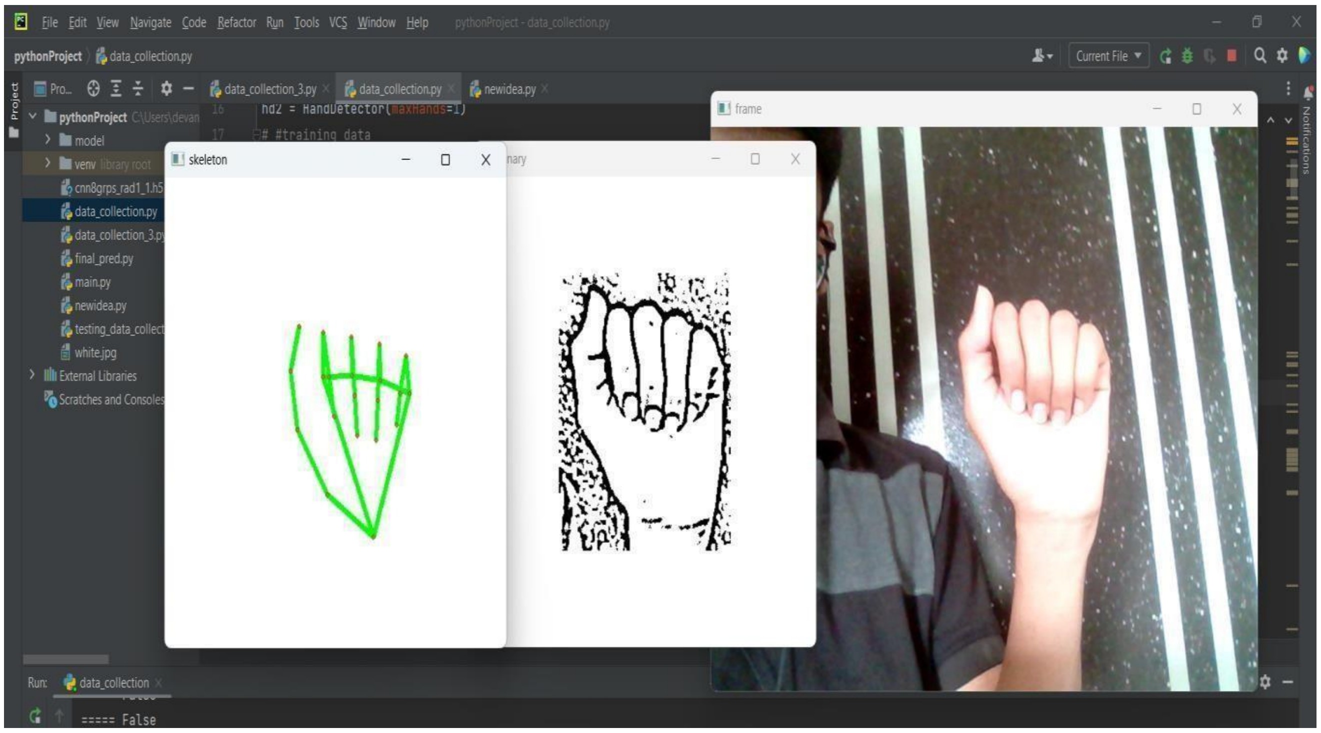


Fig 5.1.3 GrayScale Image

- Now we get this landmark points and draw it in plain white background using OpenCV library
- By doing this we tackle the situation of background and lightning conditions because the mediapipe Library will give us landmark points in any background and mostly in any lightning conditions.



```
hd2 = HandDetector(maxHands=1)
# #training data
```

```
===== False
```

Fig 5.1.4 Data Collection of Hand Gestures

we have collected 180 skeleton images of Alphabets from A to Z

5.1.3 Gesture Classification :

Our approach uses two layers of algorithm to predict the final symbol of the user.

Algorithm Layer 1:

1. Apply gaussian blur filter and threshold to the frame taken with OpenCV to get the processed image after feature extraction.
2. This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word.

Algorithm Layer 2:

1. We detect various sets of symbols which show similar results on getting detected.
2. We then classify between those sets using classifiers made for those sets only.

5.1.3.1 Convolutional Neural Network (CNN)

- 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 scores.

5.1.3.1.1 Convolutional Layer:

- In convolution layer I have taken a small window size [typically of length 5*5] that extends to the depth of the input matrix.

- The layer consists of learnable filters of window size. During every iteration I slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position.

- As I continue this process well create a 2-Dimensional activation matrix that gives the response of that matrix at every spatial position.

- That is, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour.

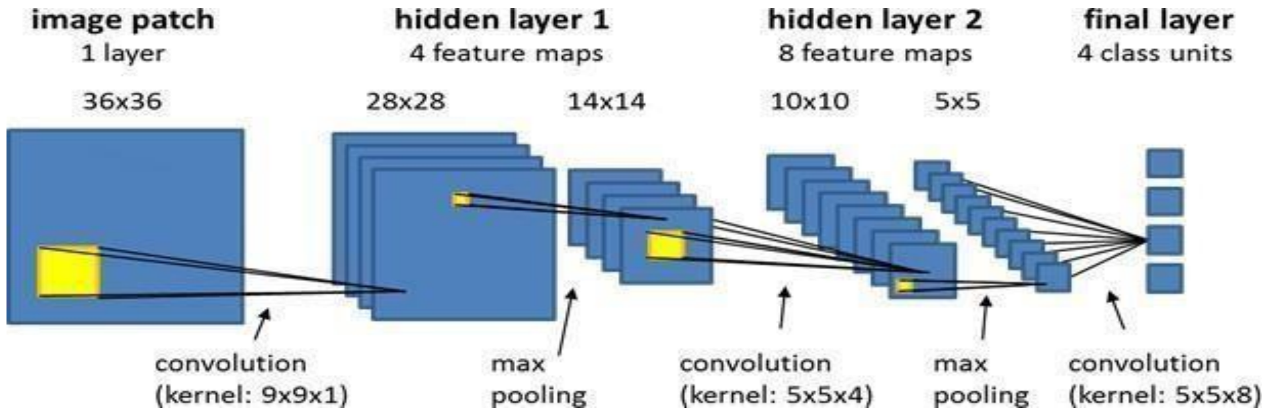


Fig 5.1.5 Convolutional Layer

5.1.3.1.2 Pooling Layer:

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

- There are two types of pooling:

a. Max Pooling:

- In max pooling we take a window size [for example window of size 2*2], and only taken the maximum of 4 values.

- Well lid this window and continue this process, so well finally get an activation matrix half of its original Size.

b. Average Pooling:

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

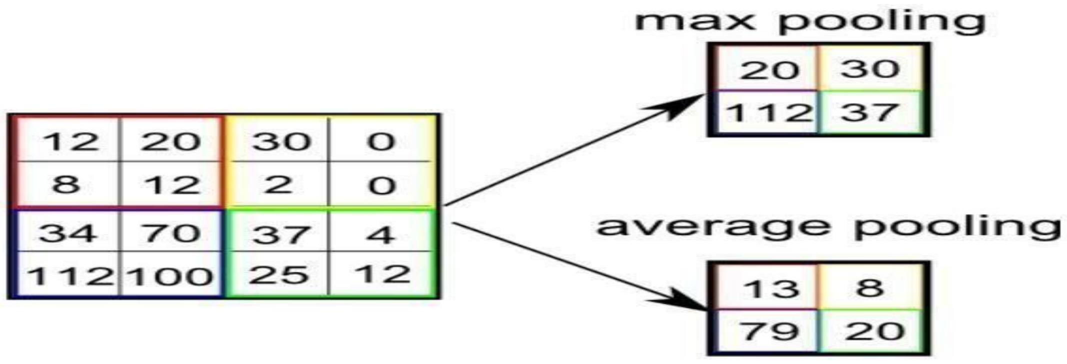


Fig 5.1.6 Pooling

5.1.3.1.3 Fully Connected Layer:

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

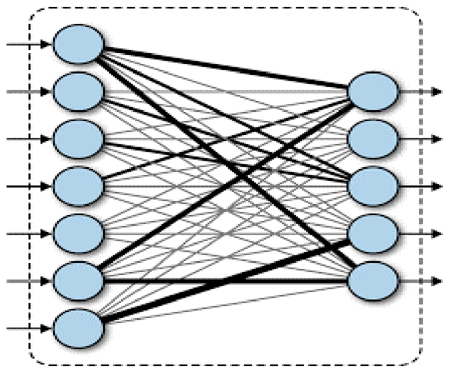


Fig 5.1.7 Fully Connected Layer

- The preprocessed 180 images/alphabet will feed the keras CNN model.
- Because we got bad accuracy in 26 different classes thus, We divided whole 26 different alphabets into 8 classes in which every class contains similar alphabets:
  [y,j]
  [c,o]
  [g,h]

[b,d,f,I,u,v,k,r,w]

[p,q,z]

[a,e,m,n,s,t]

- All the gesture labels will be assigned with a probability. The label with the highest probability will treated to be the predicted label.
- So when model will classify [aemnst] in one single class using mathematical operation on hand landmarks we will classify further into single alphabet a or e or m or n or s or t.
- -Finally, we got high Accuracy (with and without clean background and proper lightning conditions) through our method.

5.1.4 Text To Speech Translation:

The model translates known gestures into words. we have used pyttsx3 library to convert the recognized words into the appropriate speech. The text-to-speech output is a simple workaround, but it's a useful feature because it simulates a real-life dialogue.

5.2 UML Diagrams

UML stands for Unified Modeling Language. The UML is only language so is just one part of the software development method. The UML is process independent, although optimally it should be used in a process that should be driven, architecture-centric, iterative, and incremental. The UML is language for visualizing, specifying, constructing, documenting the articles in a software- intensive system. It is based on diagrammatic representations of software components.The UML is a graphical language, which consists of all interesting systems. There are also different structures that can transcend what can be represented in a programming language.

5.2.1 Class Diagram

Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagram describe the different perspective when designing a system conceptual, specification and implementation. Classes are composed of three things: name, attributes, and operations. Class diagram also display relationships such as containment, inheritance, association etc. The association relationship is most common relationship in a class diagram. The association shows the relationship between instances of classes.

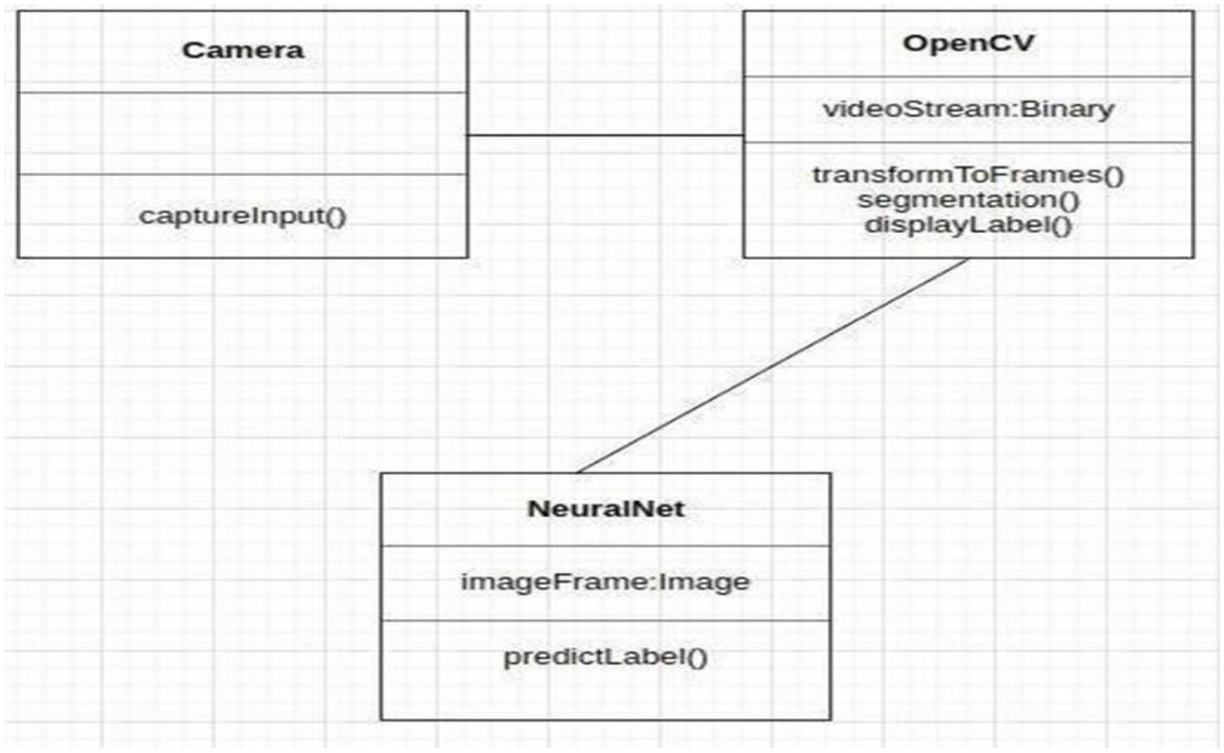


Fig 5.2.1 Class Diagram

5.2.2 Sequence Diagram

Sequence diagram displays the time sequence of the objects participating in the interaction. This consists of the vertical dimension(time) and horizontal dimension (different objects). A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams. This sequence illustrates the flow of information from gesture capture to text generation in real time. The model plays a crucial role in recognizing sign language gestures and converting them into meaningful text.

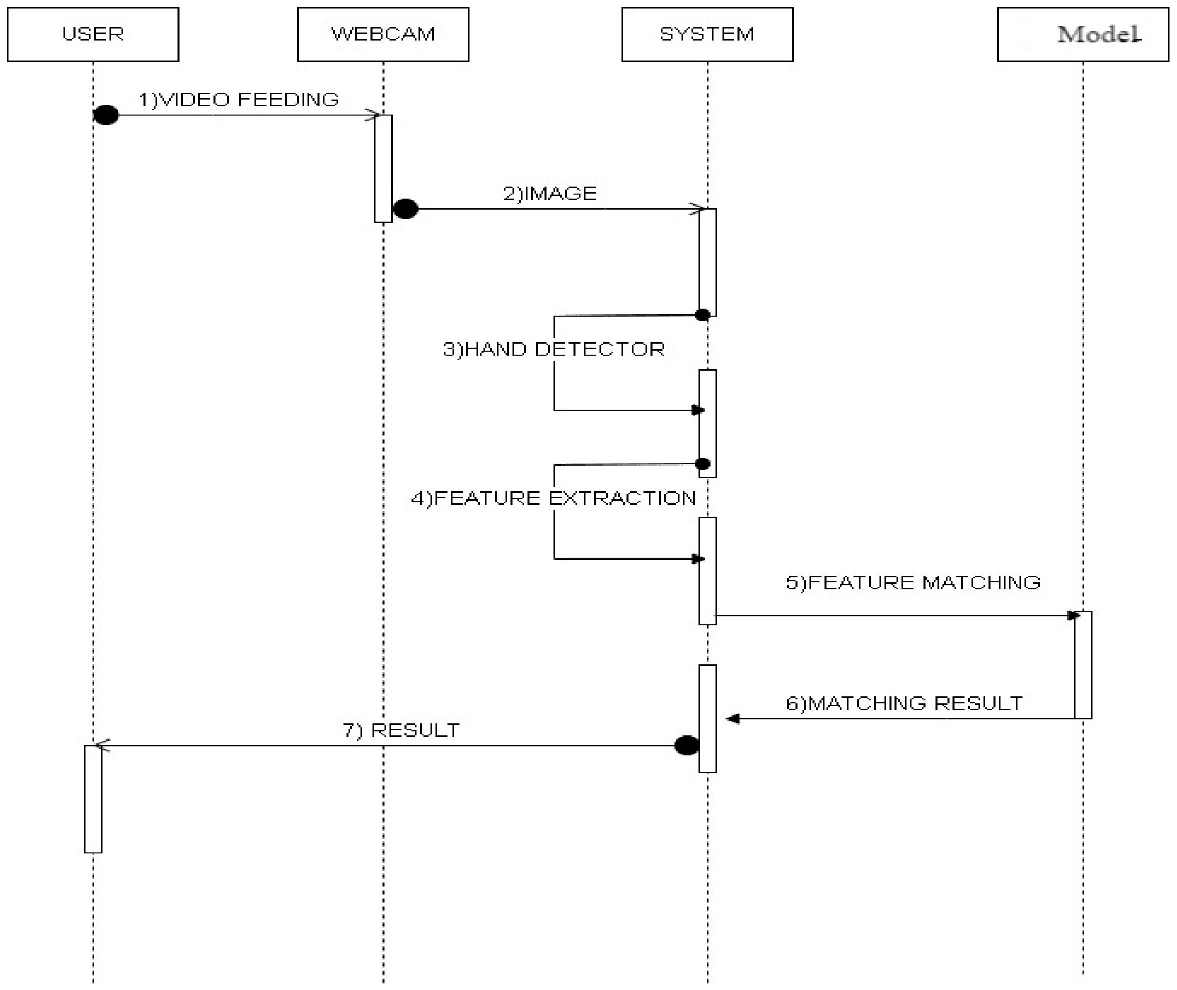


Fig 5.2.2 **Sequence Diagram**

5.2.3 Use Case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

In brief, the purposes of use case diagrams can be said to be as follows −

- Used to gather the requirements of a system.
- Used to get an outside view of a system.
- Identify the external and internal factors influencing the system.
- Show the interaction among the requirements are actors.

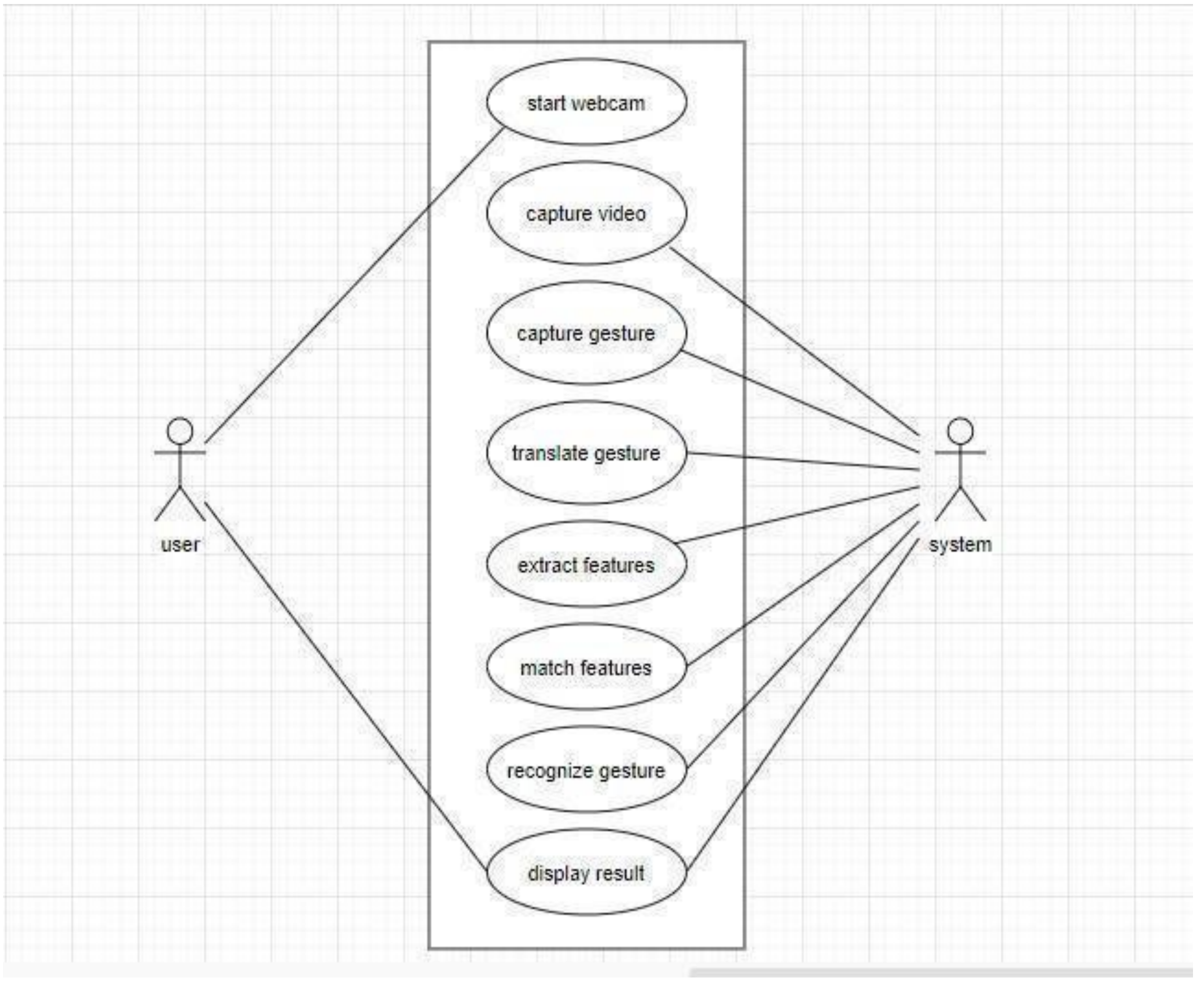


Fig 5.2.3 Use Case Diagram

5.2.4 Data Flow Diagram

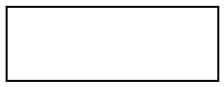
A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. The DFD is also known as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. It maps out the flow of information for any process or system, how data is processed in terms of inputs and outputs. It uses defined symbols like rectangles, circles and arrows to show data inputs, outputs, storage points and the routes between each destination. They can be used to analyse an existing system or model of a new one. A DFD can often visually "say" things that would be hard to explain in words and they work for both technical and non-technical. There are four components in DFD:

1. External Entity
2. Process
3. Data Flow
4. data Store

5.2.4.1 External Entity:

It is an outside system that sends or receives data, communicating with the system. They are the sources and destinations of information entering and leaving the system. They might be an outside organization or person, a computer system or a business system. They are known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram. These are sources and destinations of the system‘s input and output.

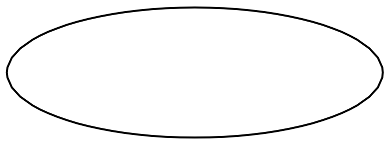
Representation:



5.2.4.2 Process:

It is just like a function that changes the data, producing an output. It might perform computations for sort data based on logic or direct the dataflowbased on business rules.

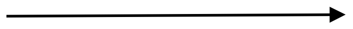
Representation:



5.2.4.3 Data Flow:

A dataflow represents a package of information flowing between two objects in the data-flow diagram, Data flows are used to model the flow of information into the system, out of the system and between the elements within the system.

Representation:



5.2.4.4 Data Store:

These are the files or repositories that hold information for later use, such as a database table or a membership form. Each data storereceives a simple label.

Representation:

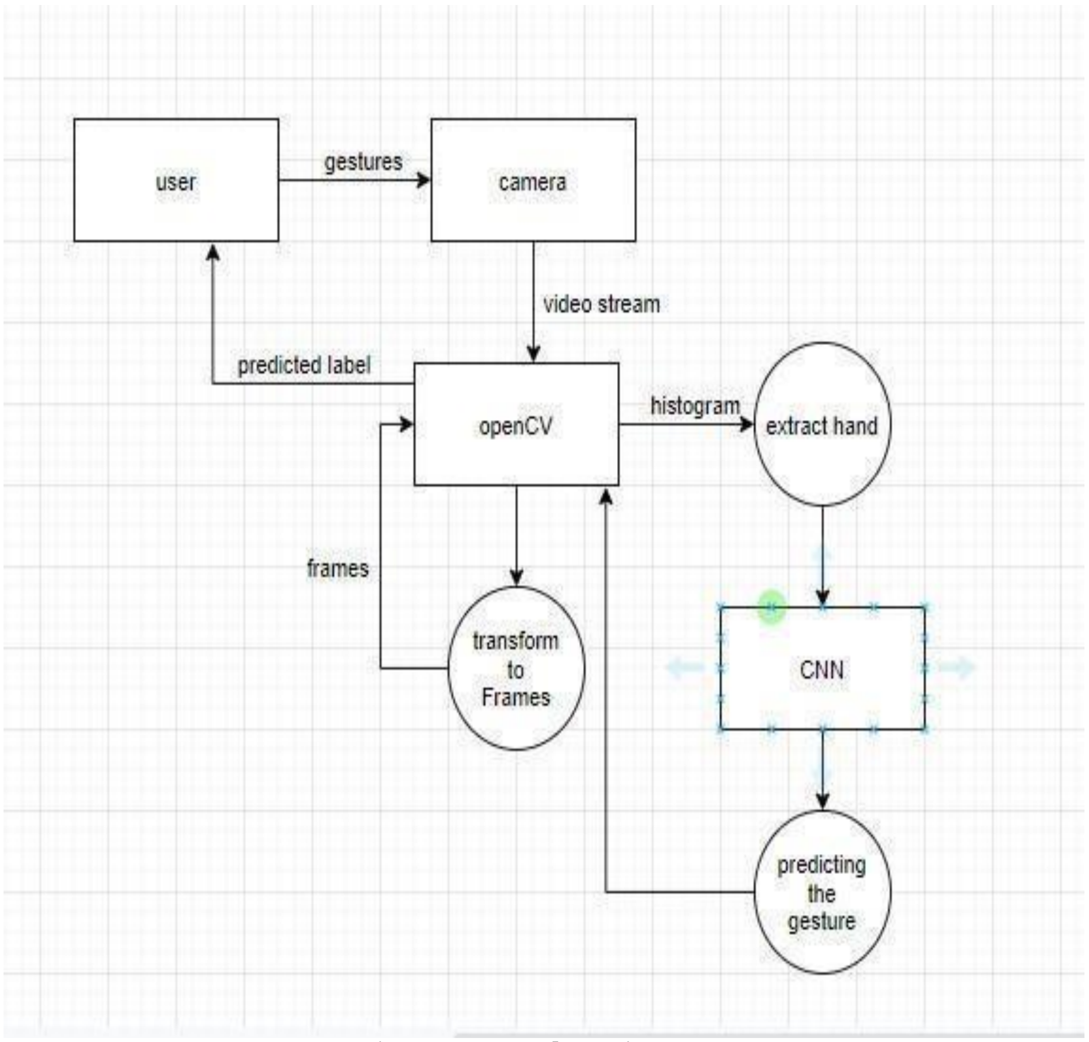


Fig 5.2.4 Data Flow Diagram

CHAPTER - 6

CODING AND IMPLEMENTATION

CHAPTER – 6

CODING AND IMPLEMENTATION

6.1 Source code:

Main.py.

This code utilizes a CNN to interpret sign language Gestures, converting them into text and then synthesizing speech output.it involves training the CNN on a dataset of sign language images paired with corresponding text labels, enabling real-time translation of sign language into spoken words.

```
# Importing Libraries
import numpy as np
import math
import cv2
import os, sys
import traceback
import pyttsx3
from keras.models import load_model
from cvzone.HandTrackingModule import HandDetector
from string import ascii_uppercase
import enchant
import tkinter as tk
from PIL import Image, ImageTk

offset=29

# os.environ["THEANO_FLAGS"] = "device=cuda, assert_no_cpu_op=True"

hs=enchant.Dict("en-US")
hd = HandDetector(maxHands=1)
hd2 = HandDetector(maxHands=1)

class Application:

    def __init__(self):
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.model = load_model('model.h5')
        self.speak_engine=pyttsx3.init()
        self.speak_engine.setProperty("rate",100)
        voices=self.speak_engine.getProperty("voices")
```

```
self.count=-1
self.ten_prev_char=[] for i
in range(10):
        self.ten_prev_char.append(" ")


    for i in ascii_uppercase:
        self.ct[i] = 0

    print("Loaded model from disk")

    self.root = tk.Tk()
    self.root.title("Sign Language To Text Conversion")
    self.root.protocol('WM_DELETE_WINDOW', self.destructor)
    self.root.geometry("1300x700")

    self.panel = tk.Label(self.root)
    self.panel.place(x=100, y=3, width=480, height=640)

    self.panel2 = tk.Label(self.root) # initialize image panel
    self.panel2.place(x=700, y=115, width=400, height=400)

    self.T = tk.Label(self.root)
    self.T.place(x=60, y=5)
    self.T.config(text="Sign Language To Text and Speech Conversion", font=("Courier", 30,
 "bold"))

    self.panel3 = tk.Label(self.root) # Current Symbol
    self.panel3.place(x=280, y=585)

    self.T1 = tk.Label(self.root)
    self.T1.place(x=10, y=580)
    self.T1.config(text="Character :", font=("Courier", 30, "bold"))

    self.panel5 = tk.Label(self.root) # Sentence
    self.panel5.place(x=260, y=632)

    self.T3 = tk.Label(self.root)
    self.T3.place(x=10, y=632)
    self.T3.config(text="Sentence :", font=("Courier", 30, "bold"))

    self.T4 = tk.Label(self.root)
    self.T4.place(x=10, y=700)
    self.T4.config(text="Suggestions :", fg="red", font=("Courier", 30, "bold"))
```

```
        self.b1=tk.Button(self.root)
        self.b1.place(x=390,y=700)

        self.b2 = tk.Button(self.root)
        self.b2.place(x=590, y=700)

        self.b3 = tk.Button(self.root)
        self.b3.place(x=790, y=700)

        self.b4 = tk.Button(self.root)
        self.b4.place(x=990, y=700)

        self.speak = tk.Button(self.root)
        self.speak.place(x=1305, y=630)
        self.speak.config(text="Speak", font=("Courier", 20), wraplength=100, command=self.speak_fun)

        self.clear = tk.Button(self.root)
        self.clear.place(x=1205, y=630)
        self.clear.config(text="Clear", font=("Courier", 20), wraplength=100, command=self.clear_fun)
        self.str = " "
        self.ccc=0
        self.word = " "
        self.current_symbol = "C"
        self.photo = "Empty"
        self.word1="  "
        self.word2 = " "
        self.word3 = " "
        self.word4 = " "

        self.video_loop()

    def video_loop(self):
        try:
            ok, frame = self.vs.read()
            cv2image = cv2.flip(frame, 1)
            hands = hd.findHands(cv2image, draw=False, flipType=True)
            cv2image_copy=np.array(cv2image)
self.pts[t + 1][1] + os1),
                                (0, 255, 0), 3)
                for t in range(17, 20, 1):
                    cv2.line(white, (self.pts[t][0] + os, self.pts[t][1] + os1), (self.pts[t + 1][0] + os, self.pts[t
+ 1][1] + os1),
                                (0, 255, 0), 3)
                cv2.line(white, (self.pts[5][0] + os, self.pts[5][1] + os1), (self.pts[9][0] + os, self.pts[9][1]
+ os1), (0, 255, 0),
                            3)
                cv2.line(white, (self.pts[9][0] + os, self.pts[9][1] + os1), (self.pts[13][0] + os,
self.pts[13][1] + os1), (0, 255, 0),
                         3)
                cv2.line(white, (self.pts[13][0] + os, self.pts[13][1] + os1), (self.pts[17][0] + os,
self.pts[17][1] + os1),
                         (0, 255, 0), 3)
                cv2.line(white, (self.pts[0][0] + os, self.pts[0][1] + os1), (self.pts[5][0] + os, self.pts[5][1]
+ os1), (0, 255, 0),
                         3)
                cv2.line(white, (self.pts[0][0] + os, self.pts[0][1] + os1), (self.pts[17][0] + os,
self.pts[17][1] + os1), (0, 255, 0),
                         3)

                for i in range(21):
                    cv2.circle(white, (self.pts[i][0] + os, self.pts[i][1] + os1), 2, (0, 0, 255), 1)

                res=white
                self.predict(res)

                self.current_image2 = Image.fromarray(res)

                imgtk = ImageTk.PhotoImage(image=self.current_image2)

                self.panel2.imgtk = imgtk
                self.panel2.config(image=imgtk)

                self.panel3.config(text=self.current_symbol, font=("Courier", 30))

                #self.panel4.config(text=self.word, font=("Courier", 30))

                self.b1.config(text=self.word1, font=("Courier", 20), wraplength=825,
command=self.action1)
                self.b2.config(text=self.word2, font=("Courier", 20), wraplength=825,
command=self.action2)
                self.b3.config(text=self.word3, font=("Courier", 20), wraplength=825,
command=self.action3)
                self.b4.config(text=self.word4, font=("Courier", 20), wraplength=825,
command=self.action4)

            self.panel5.config(text=self.str, font=("Courier", 30), wraplength=1025)
        except Exception:
            print("==", traceback.format_exc())
        finally:
            self.root.after(1, self.video_loop)

    def distance(self,x,y):
        return math.sqrt(((x[0] - y[0]) ** 2) + ((x[1] - y[1]) ** 2))

    def action1(self):
        idx_space = self.str.rfind(" ")
        idx_word = self.str.find(self.word, idx_space)
        last_idx = len(self.str)
        self.str = self.str[:idx_word]
        self.str = self.str + self.word1.upper()


    def action2(self):
        idx_space = self.str.rfind(" ")
        idx_word = self.str.find(self.word, idx_space)
        last_idx = len(self.str)
        self.str=self.str[:idx_word]
        self.str=self.str+self.word2.upper()
        #self.str[idx_word:last_idx] = self.word2


    def action3(self):
        idx_space = self.str.rfind(" ")
        idx_word = self.str.find(self.word, idx_space)
        last_idx = len(self.str)
        self.str = self.str[:idx_word]
        self.str = self.str + self.word3.upper()



    def action4(self):
        idx_space = self.str.rfind(" ")
        idx_word = self.str.find(self.word, idx_space)
        last_idx = len(self.str)
        self.str = self.str[:idx_word]
        self.str = self.str + self.word4.upper()


    def speak_fun(self):
        self.speak_engine.say(self.str)
        self.speak_engine.runAndWait()


    def clear_fun(self):
        self.str=" "
        self.word1 = " "
        self.word2 = " "
        self.word3 = " "
        self.word4 = " "

    def predict(self, test_image):
        white=test_image
        white = white.reshape(1, 400, 400, 3)
        prob = np.array(self.model.predict(white)[0], dtype='float32')
        ch1 = np.argmax(prob, axis=0)
        prob[ch1] = 0
        ch2 = np.argmax(prob, axis=0)
        prob[ch2] = 0
        ch3 = np.argmax(prob, axis=0)
        prob[ch3] = 0

        pl = [ch1, ch2]

        # condition for [Aemnst]
        l = [[5, 2], [5, 3], [3, 5], [3, 6], [3, 0], [3, 2], [6, 4], [6, 1], [6, 2], [6, 6], [6, 7], [6, 0], [6, 5],
             [4, 1], [1, 0], [1, 1], [6, 3], [1, 6], [5, 6], [5, 1], [4, 5], [1, 4], [1, 5], [2, 0], [2, 6], [4, 6],
             [1, 0], [5, 7], [1, 6], [6, 1], [7, 6], [2, 5], [7, 1], [5, 4], [7, 0], [7, 5], [7, 2]]
        if pl in l:
            if (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] <
self.pts[16][1] and self.pts[18][1] < self.pts[20][
                1]):
                ch1 = 0
                # print("00000")

        # condition for [o][s]
        l = [[2, 2], [2, 1]]
        if pl in l:
            if (self.pts[5][0] < self.pts[4][0]):
                ch1 = 0
                print("++++++++++++++++++")
                # print("00000")

        # condition for [c0][aemnst]
        pl = [ch1, ch2]
        if pl in l:
            if self.pts[6][1] < self.pts[8][1]:
                ch1 = 7
                # print("77777")

        # con for [x][yj]
        l = [[6, 7], [0, 7], [0, 1], [0, 0], [6, 4], [6, 6], [6, 5], [6, 1]]
        pl = [ch1, ch2]
        if pl in l:
            if self.pts[18][1] > self.pts[20][1]:
                ch1 = 7
                # print("77777")

        # condition for [x][aemnst]
        l = [[0, 4], [0, 2], [0, 3], [0, 1], [0, 6]]
        pl = [ch1, ch2]
        if pl in l:
            if self.pts[5][0] > self.pts[16][0]:
                ch1 = 6
                print("666661")


        # condition for [yj][x]
        print("2222 ch1=+++++++++++++++++", ch1, ",", ch2)
        l = [[7, 2]]
        pl = [ch1, ch2]
        if pl in l:
            if self.pts[18][1] < self.pts[20][1] and self.pts[8][1] < self.pts[10][1]:
                ch1 = 6
                print("666662")

        # condition for [c0][x]
        l = [[2, 1], [2, 2], [2, 6], [2, 7], [2, 0]]
        pl = [ch1, ch2]
        if pl in l:
            if self.distance(self.pts[8], self.pts[16]) > 50:
                ch1 = 6
                print("666663")

        # con for [l][x]

        l = [[4, 6], [4, 2], [4, 1], [4, 4]]
        pl = [ch1, ch2]
        if pl in l:
            if self.distance(self.pts[4], self.pts[11]) < 60:

        l = [[1, 5], [1, 7], [1, 1], [1, 6], [1, 3], [1, 0]]
        pl = [ch1, ch2]
        if pl in l:
            if (self.pts[4][0] < self.pts[5][0] + 15) and (
            (self.pts[6][1] < self.pts[8][1] and self.pts[10][1] < self.pts[12][1] and self.pts[14][1] <
self.pts[16][1] and
             self.pts[18][1] > self.pts[20][1])):
                ch1 = 7
                print("111114lll;;p")
        # con for [uvr]
        l = [[5, 5], [5, 0], [5, 4], [5, 1], [4, 6], [4, 1], [7, 6], [3, 0], [3, 5]]
        pl = [ch1, ch2]
        if pl in l:
            if ((self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] <
self.pts[16][1] and
                 self.pts[18][1] < self.pts[20][1])) and self.pts[4][1] > self.pts[14][1]:
                ch1 = 1
                print("111115")

        # con for [w]
        fg = 13
        l = [[3, 5], [3, 0], [3, 6], [5, 1], [4, 1], [2, 0], [5, 0], [5, 5]]
        pl = [ch1, ch2]
        if pl in l:
            if not (self.pts[0][0] + fg < self.pts[8][0] and self.pts[0][0] + fg < self.pts[12][0] and
self.pts[0][0] + fg < self.pts[16][0] and
                    self.pts[0][0] + fg < self.pts[20][0]) and not (
                    self.pts[0][0] > self.pts[8][0] and self.pts[0][0] > self.pts[12][0] and self.pts[0][0] >
self.pts[16][0] and self.pts[0][0] > self.pts[20][
                0]) and self.distance(self.pts[4], self.pts[11]) < 50:
                ch1 = 1
                print("111116")

        # con for [w]

        l = [[5, 0], [5, 5], [0, 1]]
        pl = [ch1, ch2]
        if pl in l:
            if self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] >
self.pts[16][1]:
                ch1 = 1
                print("1117")

        # -------------------------condn for 8 groups  end
            if self.pts[4][0] > self.pts[12][0] and self.pts[4][0] > self.pts[16][0] and self.pts[4][0] >
self.pts[20][0]:
                if self.pts[8][1] < self.pts[5][1]:
                    ch1 = 'Z'
                else:
                    ch1 = 'Q'
            else:
                ch1 = 'P'

        if ch1 == 1:
            if (self.pts[6][1] > self.pts[8][1] and self.pts[10][1] > self.pts[12][1] and self.pts[14][1] >
self.pts[16][1] and self.pts[18][1] > self.pts[20][
                1]):

            st=self.str.rfind(" ")
            ed=len(self.str)
            word=self.str[st+1:ed]
            self.word=word
            print(" ---------word = ",word)
            if len(word.strip())!=0:
                hs.check(word)
                lenn = len(hs.suggest(word))
                if lenn >= 4:
                    self.word4 = hs.suggest(word)[3]

                if lenn >= 3:
                    self.word3 = hs.suggest(word)[2]

                if lenn >= 2:
                    self.word2 = hs.suggest(word)[1]

                if lenn >= 1:
                    self.word1 = hs.suggest(word)[0]
            else:
                self.word1 = " "
                self.word2 = " "
                self.word3 = " "
                self.word4 = " "


    def destructor(self):

        print("Closing Application...")
        print(self.ten_prev_char)
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()


print("Starting Application...")

(Application()).root.mainloop()
```

CHAPTER - 7

TESTING AND VALIDATIONS

CHAPTER – 7

TESTING AND VALIDATIONS

7.1 Introduction

7.1.1 Testing:

- The purpose of testing is to discover errors. Testing is a process of trying to discover every conceivable fault or weakness in a work product.
- It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner.
- Software testing is an important element of the software quality assurance and represents the ultimate review of specification, design and coding. The increasing feasibility of software as a system and the cost associated with the software failures are motivated forces for well planned through testing.

7.1.2 Testing Objectives:

There are several rules that can serve as testing objectives they are:

- Testing is a process of executing program with the intent of finding an error.
- A good test case is the one that has a high probability of finding an undiscovered error.

7.2 Types of Testing(Strategies):

In order to make sure that the system does not have errors, the different levels of testing strategies that are applied at different phases of software development are :

1. Unit Testing
2. Black Box Testing.
3. White Box Testing.
4. Integration Testing.
5. System Testing.
6. Acceptance Testing.

7.2.1 Unit Testing:

Unit testing is done on individual models as they are completed and becomes executable. It is confined only to the designer's requirements. Unit testing is different from and should be preceded by other techniques, including:

- Inform Debugging
- Code Inspection

7.2.2 Black Box testing

In this strategy some test cases are generated as input conditions that fully execute all functional requirements for the program.

This testing has been used to find error in the following categories: Incorrect or missing functions

- Interface errors
- Errors in data structures are external database access
- Performance error
- Initialization and termination of errors
- In this testing only the output is checked for correctness
- The logical flow of data is not checked

7.2.3 White Box testing

In this the test cases are generated on the logic of each module by drawing flow graphs of that module and logical decisions are tested on all the cases.

It has been used to generate the test cases in the following cases:

- Guarantee that all independent paths have been executed
- Execute all loops at their boundaries and within their operational
- bounds.
- Execute internal data structures to ensure their validity.

7.2.4 Integration Testing

Integration testing ensures that software and subsystems work together a whole. It test the interface of all the modules to make sure that the modules behave properly when integrated together. It is typically performed by developers, especially at the lower, module to module level. Testers become involved in higher levels.

7.2.5 System Testing

Involves in house testing of the entire system before delivery to the user. The aim is to satisfy the user the system meets all requirements of the client's specifications. It is conducted by the testing organization if a company has one. Test data may range from and generated to production.

Requires test scheduling to plan and organize:

- Inclusion of changes/fixes.
- Test data to use

One common approach is graduated testing: as system testing progresses and (hopefully) fewer and fewer defects are found, the code is frozen for testing for increasingly longer time periods.

7.2.6 Acceptance Testing

It is a pre-delivery testing in which entire system is tested at client's site on real world data to find errors.

User Acceptance Test (UAT)

"Beta testing": Acceptance testing in the customer environment.

7.2.6.1 Requirements traceability:

- Match requirements to test cases.
- Every requirement has to be cleared by at least one test case.
- Display in a matrix of requirements vs. test cases

7.3 Test cases

Following table depicts the test cases and the results respectively.

Test No.	Input Description	Excepted Behavior	Observed Behavior	Status
1	Loading Model	Trained CNN model	Model loads without errors	Pass
2	Converting Video into Frames	Capturing video and convert into frames	System Successfully extracted all frames from the video.	Pass

3	Accuracy of Sign Language Recognition	High Accuracy	Achieved High Accuracy	Pass
4	Performance under Different Lighting Conditions	maintain accuracy in recognizing signs under various lighting conditions	The system maintained accuracy in recognizing signs under various lighting conditions	Pass
5	Speech Conversion Quality	Text Output from Sign language recognition	Natural sounding speech output corresponding to the converted text	Pass
6	Real-time performance	Live video stream of sign language gestures	Accurately identifies gestures performed by different signers with Low Latency	Pass
7	Recognize Hand Gestures	Identifies the sign language gesture	The system correctly identified the sign language gesture in the image frame.	Pass

Table 7.3: Test Cases

Test Results: All the Test Cases mentioned above passed successfully. No defects encountered

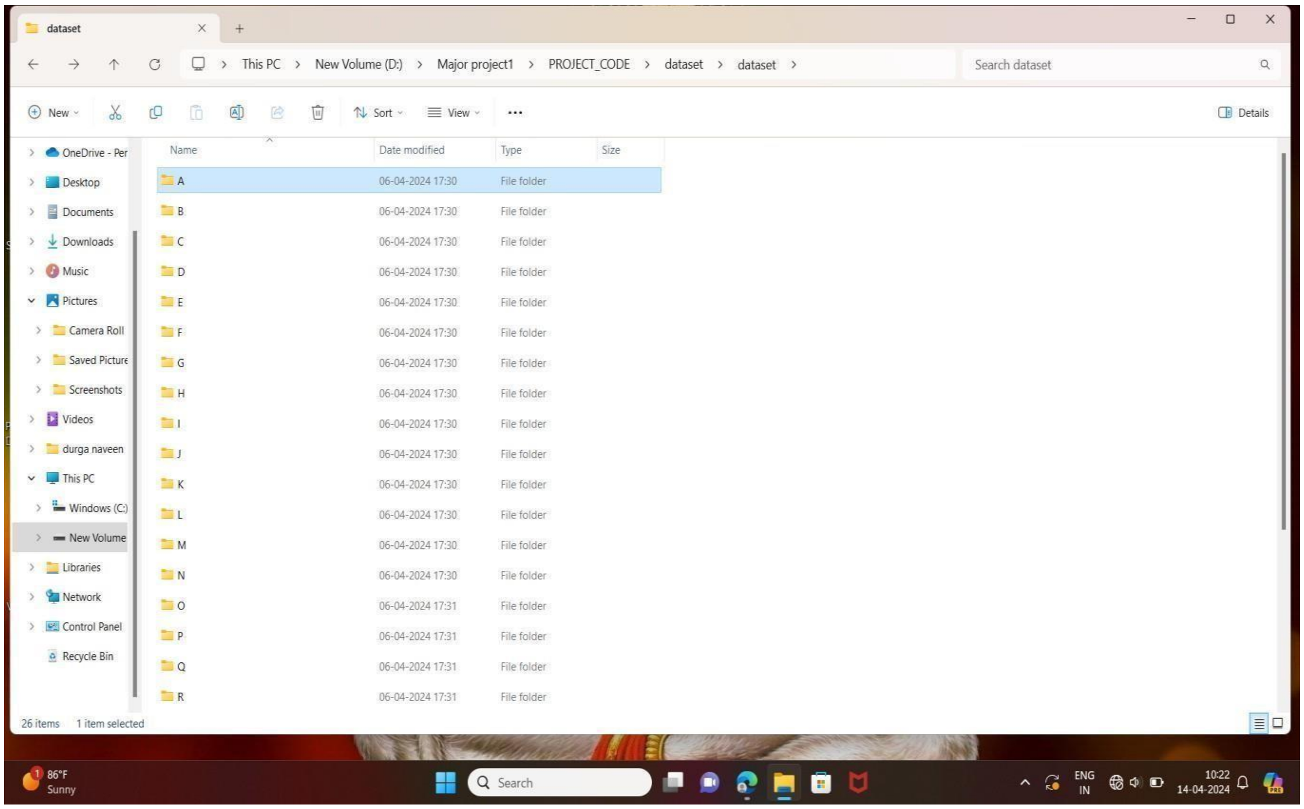
CHAPTER - 8

SCREENSHOTS

CHAPTER – 8

SCREENSHOTS

8.1 Folders contain ASL Gesture



Name	Date modified	Type	Size
A	06-04-2024 17:30	File folder	
B	06-04-2024 17:30	File folder	
C	06-04-2024 17:30	File folder	
D	06-04-2024 17:30	File folder	
E	06-04-2024 17:30	File folder	
F	06-04-2024 17:30	File folder	
G	06-04-2024 17:30	File folder	
H	06-04-2024 17:30	File folder	
I	06-04-2024 17:30	File folder	
J	06-04-2024 17:30	File folder	
K	06-04-2024 17:30	File folder	
L	06-04-2024 17:30	File folder	
M	06-04-2024 17:30	File folder	
N	06-04-2024 17:30	File folder	
O	06-04-2024 17:31	File folder	
P	06-04-2024 17:31	File folder	
Q	06-04-2024 17:31	File folder	
R	06-04-2024 17:31	File folder	

26 items 1 item selected

Fig 8.1 Folders contains ASL Gestures

8.2 Each Sign Gesture Contains 180 skeleton Images

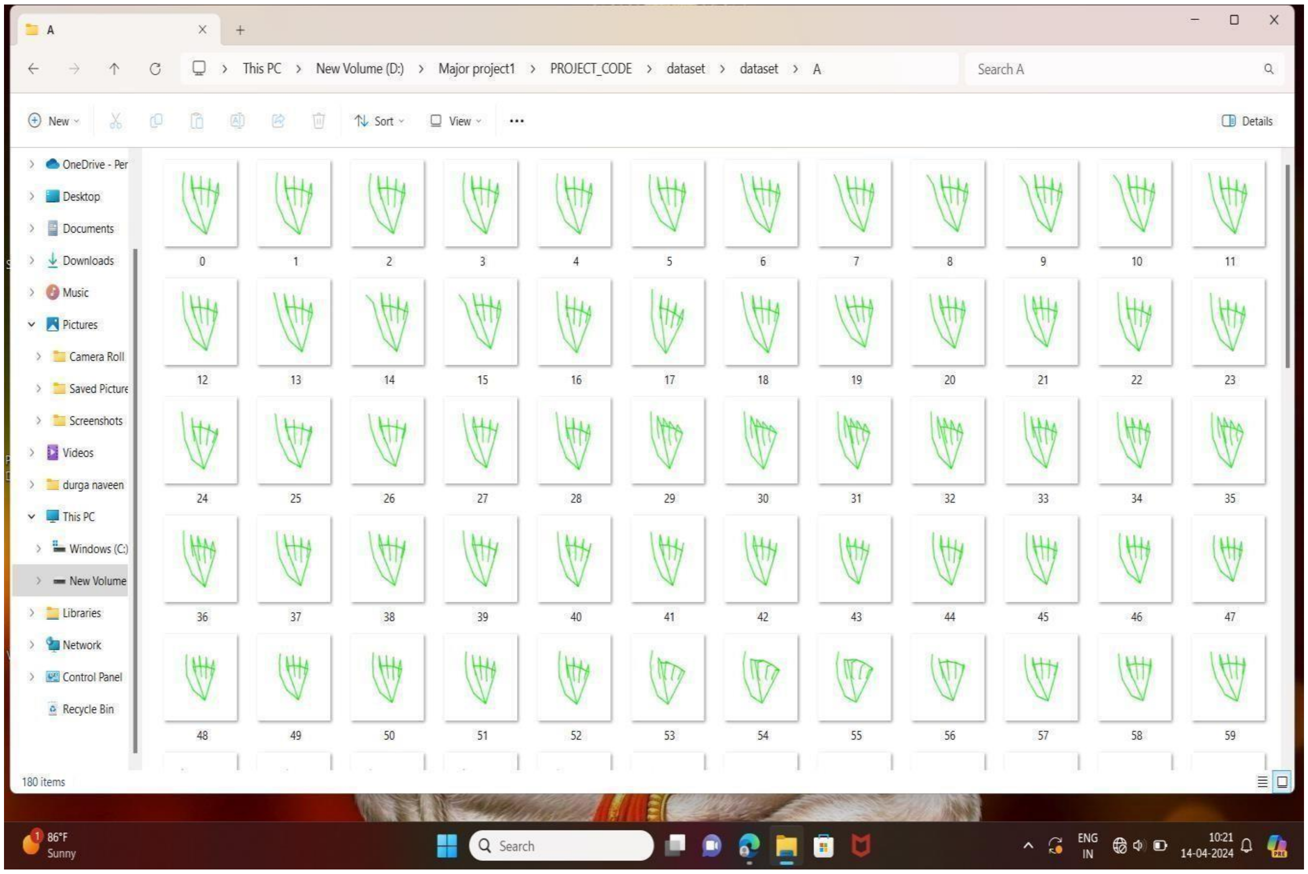
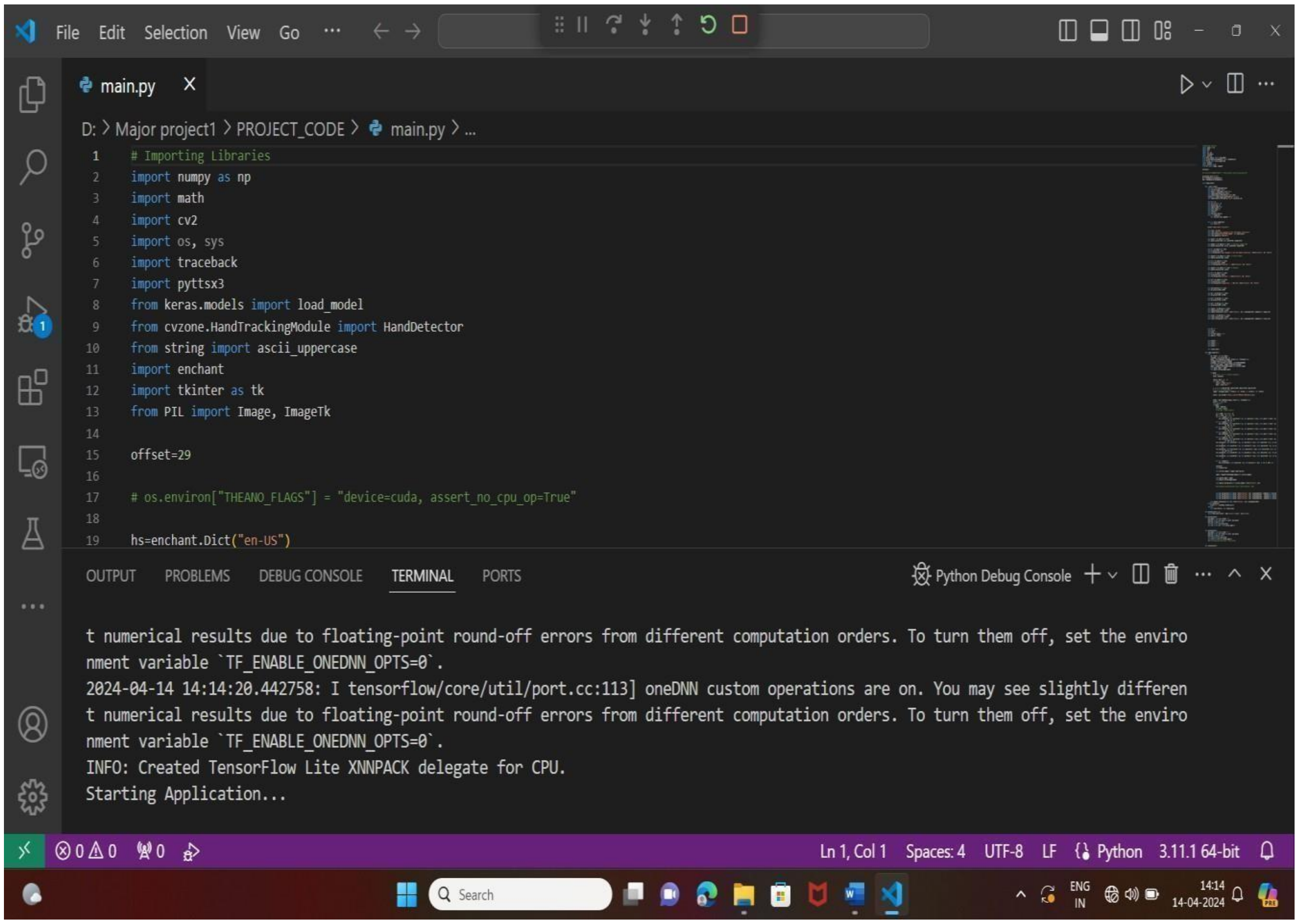


Fig 8.2 Each Sign Gesture Contains 180 skeleton Images

8.3 Starting Application



```python
# Importing Libraries
import numpy as np
import math
import cv2
import os, sys
import traceback
import pyttsx3
from keras.models import load_model
from cvzone.HandTrackingModule import HandDetector
from string import ascii_uppercase
import enchant
import tkinter as tk
from PIL import Image, ImageTk

offset=29

# os.environ["THEANO_FLAGS"] = "device=cuda, assert_no_cpu_op=True"

hs=enchant.Dict("en-US")
```

```
t numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the enviro
nment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-04-14 14:14:20.442758: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly differen
t numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the enviro
nment variable `TF_ENABLE_ONEDNN_OPTS=0`.
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
Starting Application...
```

Fig 8.3 Starting Application

8.4 Output in Windows

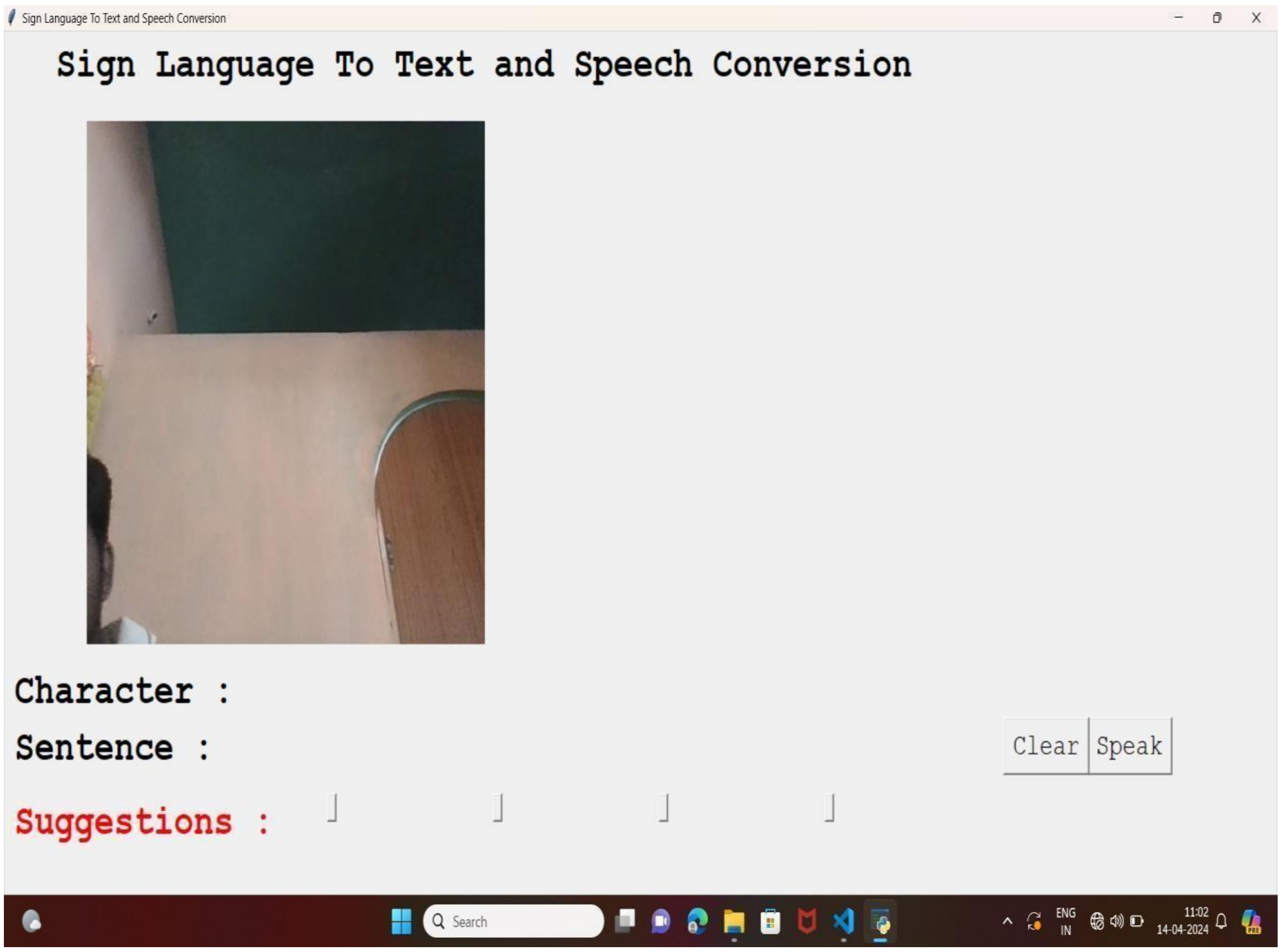

Sign Language To Text and Speech Conversion
Sign Language To Text and Speech Conversion
Character :
Sentence :
Suggestions :
Clear
Speak
Search
ENG IN
11:02
14-04-2024

Fig 8.4 Output in Windows

8.5 Output A

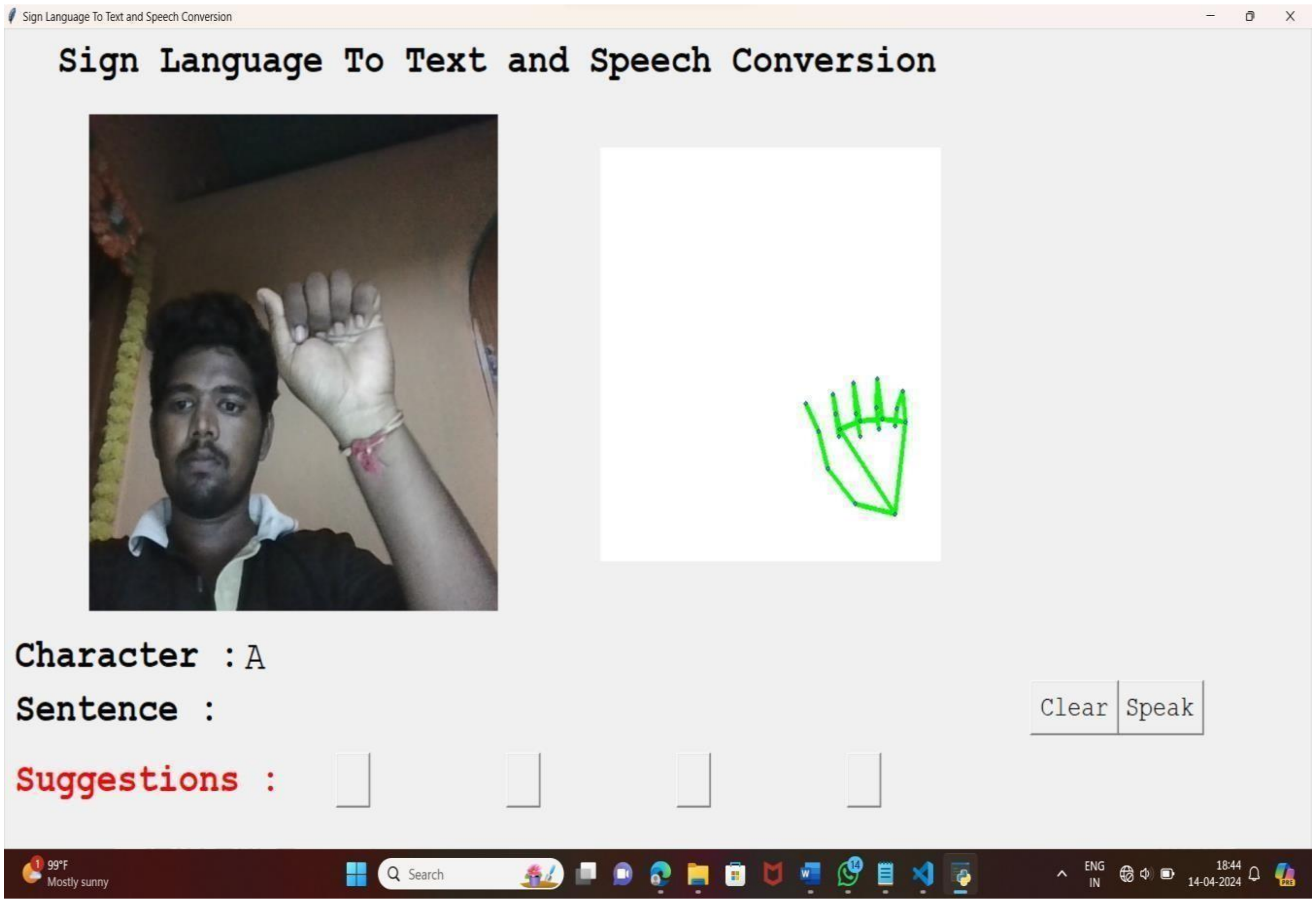


Fig 8.5 Output A

8.6 Final Output by using assigned Gestures

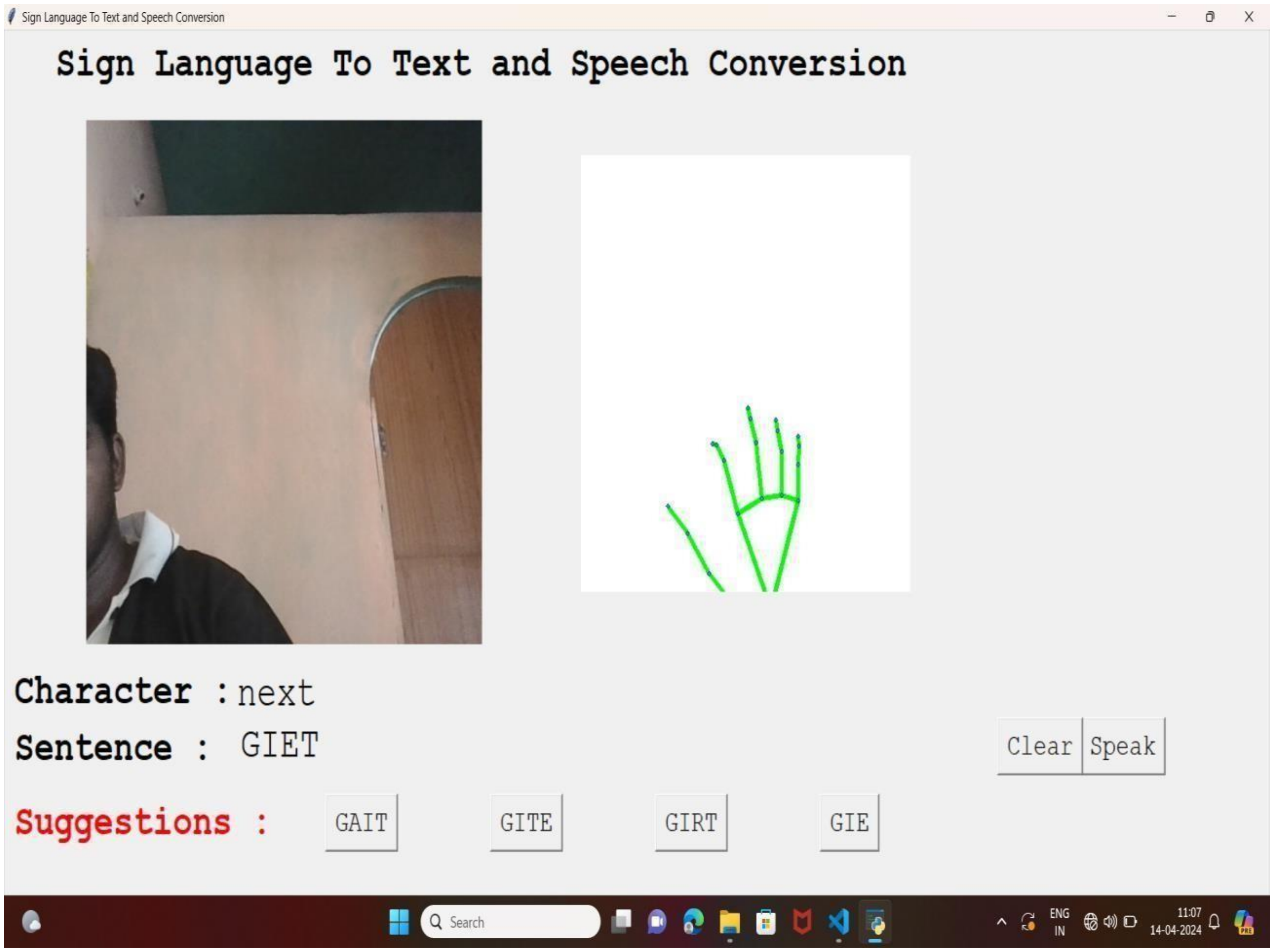


Fig 8.6 Final Output by using assigned Gestures

8.7 Suggestions given based on the current word

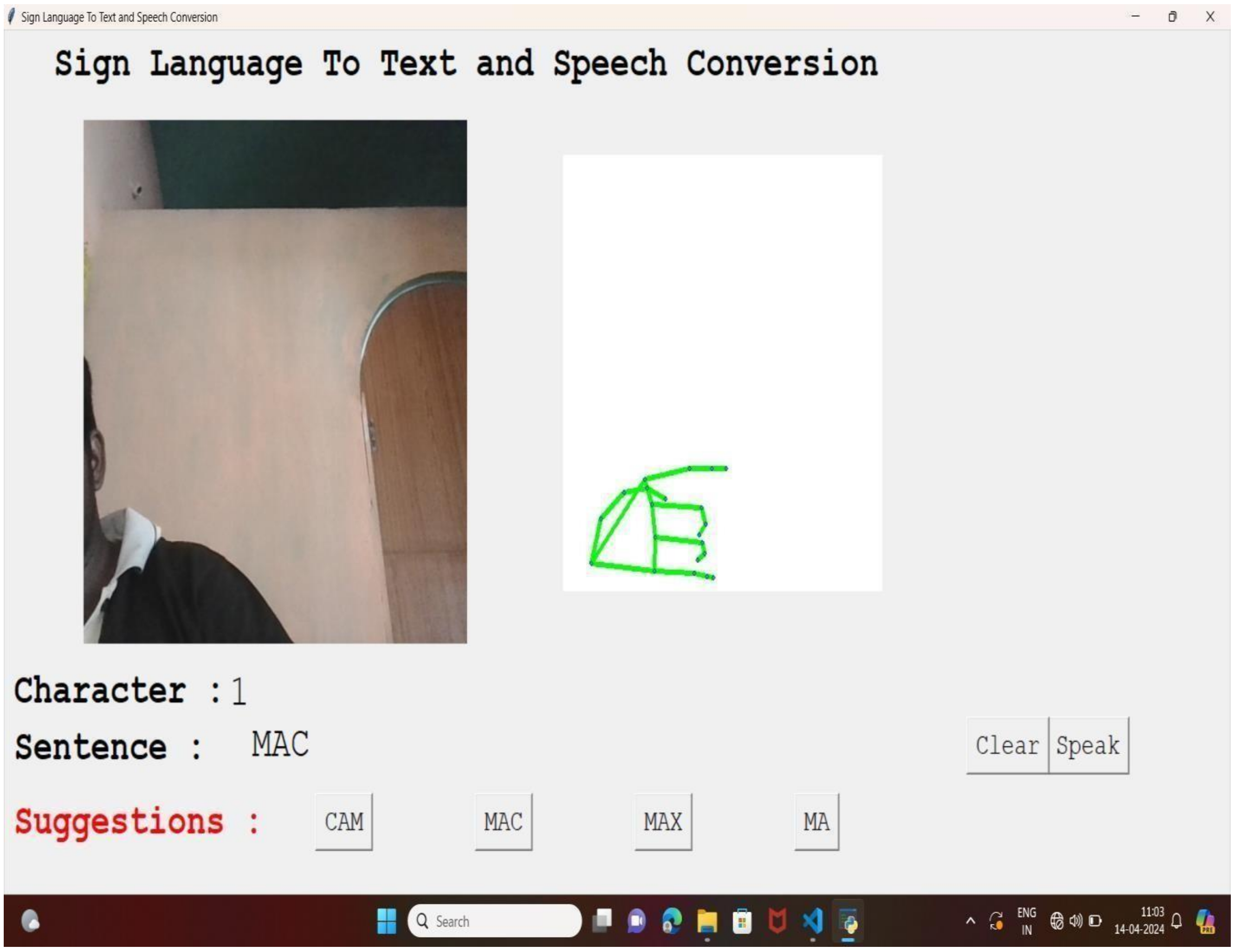

Sign Language To Text and Speech Conversion
Sign Language To Text and Speech Conversion
Character : 1
Sentence : MAC
Clear
Speak
Suggestions :
CAM
MAC
MAX
MA

Fig 8.7 Suggestions given based on the current word

CHAPTER - 9

CONCLUSION AND FUTURE SCOPE

CHAPTER – 9
CONCLUSION AND FUTURE SCOPE

9.1 Conclusion:

The proposed a prototype model can recognize and classify the American Sign Language using the deep structured learning techniques called CNN. We observe that the CNN model gives the highest accuracy due to the advanced techniques. The method takes hand gesture image as input using a webcam and gives text and speech as an output. The project demonstrated the feasibility and usefulness of such a system for facilitating communication between sign language users and non-sign language users, as well as for promoting accessibility and inclusivity for people with hearing and speech impairments. . The sentence creation algorithm significantly helped boost the accuracy of the model in real-time and improved the interface making it more user friendly. The main objective has been achieved, that is, the need for an interpreter has been eliminated. From the process, we can conclude that CNN is an efficient technique to categorize hand gestures with high degree of accuracy.

9.2 Future Scope:

This project can be enhanced in a few ways in the future: Currently, the project recognizes gestures for alphabets in American Sign Language (ASL). Expanding the vocabulary to include common words, phrases, and sentences would make the system more practical and useful. Fine-tune the CNN model to improve accuracy in gesture recognition. Explore transfer learning techniques or larger datasets for better performance. Extend the system to recognize sign languages from different regions (e.g., British Sign Language, Indian Sign Language).Create language-specific models for accurate translation. Implement dynamic gestures (e.g., signs for verbs, adjectives) beyond static alphabets. Recognize facial expressions and other non-manual components of sign language. Instead of displaying text, synthesize natural sign language gestures using avatars or animations. Combine text-to-speech with sign language synthesis for a complete communication experience. Develop a mobile app that uses the smartphone camera for on-the-go sign language translation .Integrate with existing communication apps for seamless interaction.

CHAPTER - 10

REFERENCES/WEB REFERENCES

CHAPTER - 10
REFERENCES/WEB REFERENCES

10.1 Research Papers and Text Books References:

[1] Parton, Becky Sue. "Sign language recognition and translation: A multidisciplined approach from the field of artificial intelligence." Journal of deaf studies and deaf education 11, no. 1 (2006): 94-101.

[2] Thad Starner, Mohammed J. Islam, proposed a Sign Language Recognition with Microsoft Kinect, IEEE 2013.

[3] Nikam, Ashish S., and Aarti G. Ambekar. "Sign language recognition using image-based hand gesture recognition techniques." In 2016 online international conference on green engineering and technologies (IC-GET), pp. 1-5. IEEE, 2016.

[4] Daniele Cippitelli, Davide Cipolla, proposed a DeepASL: Enabling Ubiquitous and Non Intrusive Mobile Sign Language Recognition, IEEE 2018.

[5] Alex Graves, Santiago Fernández, proposed a Sign Language Recognition Using a Convolutional Neural Network, IEEE 2018.

[6] Hrishikesh Kulkarni, Suchismita Saha, proposed a Deep Learning-Based American Sign Language (ASL) Recognition System, IEEE 2020.

[7] Deshpande, Aditi, Ansh Shriwas, Vaishnavi Deshmukh, and Shubhangi Kale. "Sign Language Recognition System using CNN." In 2023 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), pp. 906-911. IEEE, 2023.

10.2 URL References:

[1] https://en.wikipedia.org/wiki/Convolutional_neural_network

[2] https://www.geeksforgeeks.org/project-idea-sign-language-translator-speech-impaired/

[3] https://en.wikipedia.org/wiki/American_Sign_Language

[4] https://teachablemachine.withgoogle.com/

[5] https://docs.opencv.org/4.7.0/d6/d00/tutorial_py_root.html

[6] https://github.com/cvzone/cvzone

[7] https://numpy.org/learn/

[8] https://www.youtube.com/watch?v=wa2ARoUUdU8

[9] https://github.com/harshbg/Sign-Language-Interpreter-using-Deep-Learning