AI POWERED SIGN LANGUAGE INTERPRETER

AN INDUSTRIAL ORIENTED MINI PROJECT REPORT

Submitted

by

GANGONI VENKATESH (22841A0583)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Under the Guidance of

Dr.B.T.R. Naresh Reddy

AURORA'S TECHNOLOGICAL AND RESEARCH INSTITUTE

(Affiliated to JNTU, Hyderabad and approved by AICTE, New Delhi)

Parvathapur, Uppal, Hyderabad-500 039

JUNE 2025

AURORA'S TECHNOLOGICAL AND RESEARCH INSTITUTE

(Affiliated to JNTU, Hyderabad and approved by AICTE, New Delhi)

Parvathapur, Uppal, Hyderabad-500 039

DECLARATION

We hereby declare that the work described in this project, entitled **"AI POWERED SIGN LANGUAGE INTERPRETER"**, which is being submitted by us in partial fulfilment of the requirements for the award of the **Bachelor of Technology in Computer Science and Engineering** to **Aurora's Technological and Research Institute**, is the result of the investigation carried out by us under the guidance of **Dr.B.T.R. Naresh Reddy**, **Associate Professor, Department of Computer Science.**

We further declare that this work is original and has not been submitted for the award of any degree or diploma of this or any other university.

Place: Hyderabad

Date:

GANGONI VENKATESH (22841A0583)

AURORA'S TECHNOLOGICAL AND RESEARCH INSTITUTE

(Affiliated to JNTU, Hyderabad and approved by AICTE, New Delhi)

Parvathapur, Uppal, Hyderabad-500 039

CERTIFICATE

This is to certify that the Industrial Oriented Mini Project Report titled **"AI POWERED SIGN LANGUAGE INTERPRETER"** is the bonafide work of **"GANGONI VENKATESH (22841A0583)"**, who carried out the project work under my supervision.

Project Guide
Dr.B.T.R. Naresh Reddy
Associate Professor
Department of Computer Science and Engineering

Principal
(Signature with Seal)

External Examiner
(Signature)

ACKNOWLEDGEMENT

We sincerely thank our project guide and project coordinator,**Dr.B.T.R. Naresh Reddy**, **Associate Professor, Department of Computer Science**, for his constant support, encouragement, and valuable guidance, which greatly helped us in completing this project successfully.

We express our heartfelt thanks to the **Head of the Department and Vice Principal,Mr. Vinod Chavan**, for his support, kind attention, and encouragement throughout this course.

We also wish to express our deep gratitude to **Prof. Srikanth Jatla**, Principal, for providing us with a conducive academic environment and continuous motivation.

We are thankful to the **Project Review Committee members** for their constructive feedback and suggestions, which improved the quality of our work.

We also extend our thanks to the **teaching and non-teaching staff** of the department for their timely assistance and support throughout the project.

Special thanks to our **classmates and friends** for their cooperation, moral support, and teamwork during various stages of the project.

Lastly, we are deeply grateful to our **families** for their patience, understanding, and emotional support throughout this academic journey.

GANGONI VENKATESH (22841A0583)

ABSTRACT

The ability to communicate effectively is a fundamental human right, yet millions of individuals who are deaf or hard of hearing face significant barriers in accessing information and engaging with the hearing community. This project aims to develop an AI- powered sign language interpreter that translates spoken language into sign language in real-time, thereby facilitating seamless communication between hearing individuals and those who use sign language. The proposed system leverages advanced technologies in computer vision, natural language processing (NLP), and machine learning to interpret spoken words and convert them into corresponding sign language gestures .s

INDEX

LIST OF FIGURES

1. INTRODUCTION

AI-powered sign language interpreters use artificial intelligence to automatically translate spoken language into sign language and vice versa, bridging communication gaps between deaf and hearing individuals. These systems leverage technologies like computer vision, natural language processing, and machine learning to recognize, interpret, and generate sign language gestures. This technology aims to enhance accessibility and inclusion for the deaf and hard-of-hearing communities by providing real-time translation and interpretation in various settings, including educational, social, and professional environments.

This project presents an AI-powered Sign Language Interpreter — a real-time system that translates sign language gestures into spoken or written language using machine learning and computer vision technologies. The goal is to enable seamless communication between sign language users and non-signers, eliminating the need for a third-party interpreter and making everyday interactions more inclusive.

The interpreter uses video input to detect and interpret hand movements, facial expressions, and body posture, which are essential components of sign language. Deep learning models, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), are employed to analyze these visual cues and map them to corresponding words or phrases in the target language. The translated output is then delivered in a user-friendly format such as on-screen text or synthesized speech.

This documentation outlines the technical foundation, system architecture, datasets used, training process, implementation details, and performance evaluation of the AI-powered interpreter. It also discusses the challenges faced in recognizing nuanced gestures and maintaining accuracy across different sign language dialects. By addressing these challenges, the project aims to contribute to the development of scalable, reliable assistive technologies that promote equal access to communication for everyone.

2. LITERATURE SURVEY

The development of AI-powered sign language interpreters has become a focal point of research in human-computer interaction, accessibility technology, and computer vision. With the rise of open-source tools like MediaPipe by Google, real-time hand tracking and gesture recognition have become more accessible and efficient, making it feasible to build practical sign language recognition systems using Python.

TITLE: Vision-Based Sign Language Recognition System Using CNN

AUTHOR: Arslan et al. (2020)

ABSTRACT: This paper proposed a deep learning-based sign language interpreter using Convolutional Neural Networks (CNNs). It used real-time hand gesture recognition to convert signs into text with 93% accuracy.

TITLE: Real-Time American Sign Language Detection Using TensorFlow

AUTHOR: Patel, M., Sharma, K. (2019)

ABSTRACT: The authors developed a webcam-based ASL recognition system using TensorFlow and OpenCV. The system detected static gestures and classified them using a CNN with live feedback.

TITLE: Sign Language Recognition Using Deep Learning

AUTHOR: Gupta, R., Singh, M. (2021)

ABSTRACT: A dataset of Indian Sign Language (ISL) was created and trained using a deep neural network. The study achieved 95% accuracy for isolated word recognition using image sequences.

TITLE: Glove-Based Sign Language Translator Using Microcontrollers

AUTHOR: Joseph, P., Rajesh, P. (2018)

ABSTRACT: A hardware-based approach using sensors embedded in gloves to detect finger bending and translate sign gestures into speech using microcontrollers like Arduino.

1. **MediaPipeFramework:**

MediaPipe is a cross-platform, open-source framework that provides customizable ML solutions for live and streaming media. It offers robust and lightweight models for hand, face, and body landmark detection, which are crucial for interpreting sign language. Lugaresi et al. (2019) introduced the high-fidelity hand tracking pipeline in MediaPipe, capable of detecting 21 key landmarks per hand in real time, using a combination of palm detection and landmark regression models. This capability has proven useful for capturing the fine motor movements required for accurate sign recognition.

2. **Gesture and Sign Language Recognition Using MediaPipe and Python:**

Recent works have demonstrated the application of MediaPipe for gesture-based systems. For instance, Majumder et al. (2021) implemented a static ASL (American Sign Language) interpreter using MediaPipe's hand landmarks in combination with machine learning classifiers like SVM and KNN. These models showed high accuracy in classifying static gestures when trained on custom datasets.

3. **Real-Time Applications and Limitations:**

Several prototypes and open-source projects use MediaPipe in Python to create real-time sign language translation systems. These solutions often utilize hand landmark coordinates as input features for gesture classification. While effective for static signs, challenges remain in recognizing dynamic or continuous signs due to the need for temporal modeling, such as with LSTM or GRU networks.

4. **Dataset Limitations and Variability:**

Most current models rely on small, user-generated datasets which may not generalize well across different users, lighting conditions, or sign language dialects. Despite MediaPipe's robustness, additional preprocessing and normalization are often required to improve cross-user accuracy

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- **SignAll:** SignAll is a commercial system that translates American Sign Language (ASL) into English in real time using multiple cameras, depth sensors, and advanced natural language processing. It uses a complex setup with multiple vision inputs and is primarily designed for enterprises, making it accurate but not easily accessible for developers or consumers..
- **Sign2Text:** Sign2Text is an open-source initiative that leverages MediaPipe and TensorFlow for real-time hand gesture recognition. It focuses on converting sign language gestures into text, mainly targeting ASL alphabets. It’s a lightweight and Python-based solution, making it ideal for mobile and educational applications.
- **DeepASL:** DeepASL is a research project developed by the University of Rochester. It uses deep learning models with Leap Motion sensors to recognize both static and dynamic signs in ASL. The system supports continuous recognition and is designed to work without requiring the user to wear any special hardware.

DISADVANTAGES:

1. Limited Sign Language Support
2. Poor Recognition of Dynamic Gestures
3. High Computational Requirements

3.2 PROPOSED SYSTEM

An AI-powered sign language interpreter is a system that uses artificial intelligence (AI), particularly machine learning and computer vision, to recognize, translate, and interpret sign language gestures into text or speech, and vice versa. These systems are being developed to facilitate communication between deaf or hard-of-hearing individuals and hearing individuals who may not understand sign language.

ADVANTAGES:

1. Real-Time Communication
2. 24/7 Availability
3. Cost-Effectiveness
4. Scalability

3.3 SYSTEM REQUIREMENTS:

3.3.1 SOFTWARE REQUIREMENTS:

The development of an AI-powered Sign Language Interpreter involves various software tools, libraries, and frameworks to handle tasks such as computer vision, machine learning, data processing, and user interaction. The following software components are essential for building and running the system effectively.

- Operating system : Windows 10/11, Linux
- Coding language : python 3.9.0
- Environment: VS Code
- Python Libraries:Mediapipe,opencv

3.3.2 HARDWARE REQUIREMENTS:

To effectively run an AI-powered Sign Language Interpreter—particularly one that relies on real-time video processing and machine learning—the system must meet certain hardware specifications. These requirements vary depending on the complexity of the application (e.g., static vs dynamic sign recognition), whether model training is involved, and the deployment platform (desktop, embedded, or cloud-based).

- Processor (CPU) : Intel Core i5 or AMD Ryzen 5
- RAM : 8GB
- Webca : 720pHD m
- GPU : Integrated Graphics (Intel HD/UHD, etc.)
- Operating system: Windows 10/11

4. SYSTEM DESIGN

The AI-powered Sign Language Interpreter using Python and MediaPipe is designed as a modular, real-time system that captures hand gestures through a webcam and translates them into readable or spoken language using computer vision and machine learning. The system begins with a video capture module that uses OpenCV to continuously stream frames from the user's webcam, serving as the primary input for gesture detection. Each frame is preprocessed by resizing and converting it to the RGB color space, ensuring compatibility with MediaPipe's input requirements. The core of the system leverages MediaPipe's Hands module, which detects and tracks 21 landmarks on each hand in real time, providing x, y, and z coordinates for each joint, including fingertips, knuckles, and wrist points. These coordinates are then passed to a feature extraction layer where meaningful geometric relationships are calculated—such as the angles between fingers, distances between landmark pairs, and hand orientation. These features provide a more stable and generalized input for the classification model, helping to reduce sensitivity to differences in hand size, lighting, or positioning. The extracted features are then fed into a machine learning classifier—typically a K-Nearest Neighbors (KNN), Support Vector Machine (SVM), or a lightweight neural network—trained on a labeled dataset of hand gestures. In the initial version of the system, static gesture recognition is prioritized (such as ASL alphabets or common words), but the design can be extended to dynamic or continuous signs using sequential models like LSTM or GRU. Once a gesture is classified, the corresponding text is displayed on the screen, and optionally converted to speech using a text-to-speech (TTS) engine like pyttsx3 or Google's gTTS, enabling direct communication with non-signers. The entire system is built in Python for cross-platform compatibility and ease of development, while MediaPipe ensures low-latency processing suitable for real-time applications. Design considerations include accuracy, user independence, and extensibility, allowing for future improvements such as facial expression recognition, multi-hand detection, multilingual sign support.

4.1 SYSTEM ARCHITECTURE:

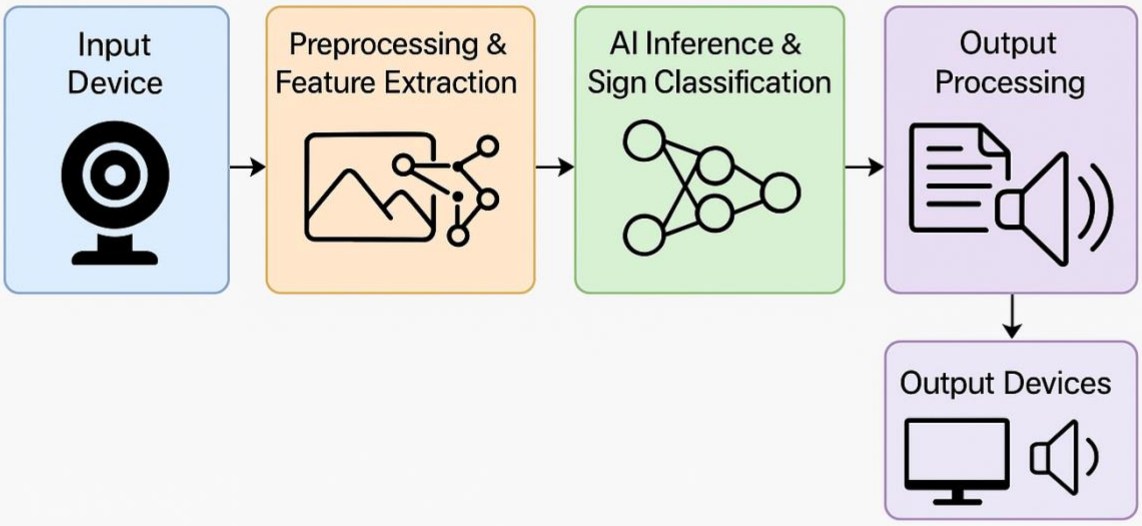

Input Device
Preprocessing & Feature Extraction
AI Inference & Sign Classification
Output Processing
Output Devices

Fig 4.1 Architecture Diagram

EXPLANATION OF ARCHITECTURE:

The system architecture depicted in the image outlines an AI-powered Sign Language Interpreter using Python with tools such as MediaPipe, OpenCV, cvzone, and TensorFlow. This setup allows real-time recognition of sign language gestures via webcam.

The process begins with the Web Cam, managed by OpenCV, which captures real-time video frames of the user's hand gestures. These frames are passed to the Hand Detection module. This module uses MediaPipe—a powerful framework for real-time hand tracking—alongside cvzone (which simplifies computer vision tasks), OpenCV, and optionally TensorFlow for hand detection. It identifies key points (landmarks) on the hand such as joints and fingertips. Once the hand is detected, the Feature Extraction module converts the hand landmarks into a numerical landmarks vector.

4.2 UML DIAGRAMS:

There are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation). The two broadest categories that encompass all other types are Behavioral UML diagram and Structural UML diagram. As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas others describe the behavior of the system, its actors, and its building components. The different types are broken down as follows.

- Use Case Diagram
- Class Diagram
- Sequence Diagram
- Activity Diagram

Just like any other thing in life, to get something done properly, you need the right tools. For documenting software, processes, or systems, you need the right tools that offer UML annotations and UML diagram templates. There are different software documentation tools that can help you draw a UML diagram. They are generally divided into these main categories: Paper and pen – this one is a no-brainer. Pick up a paper and a pen, open a UML syntax cheat sheet from the web and start drawing any diagram type you need. Online tools – there are several online applications that can be used to draw a UML diagram. Most of them offer free trials or a limited number of diagrams on the free tier. If you are looking for a long-term solution for drawing UML diagrams, it is generally more beneficial to buy a premium subscription for one of the applications. Free Online Tools – these do pretty much the same thing that the paid ones do. The main difference is that the paid ones also offer tutorials and ready-made templates for specific UML diagrams. A great free tool is draw.io.

DESIGNS:

4.2.1 USE CASE DIAGRAM:

The use case diagram provided illustrates the interaction between a user and the AI-powered Sign Language Interpreter System. This system is built using Python, MediaPipe, and associated technologies, enabling real-time sign language recognition.

The user, represented on the left, initiates the process by starting the webcam. This activates the Start WebCam function in the system, allowing it to begin capturing input. The system then proceeds to Capture Image/Video, continuously receiving video frames of the user's hand movements.

As the user performs sign language gestures, the system uses the Capture Gesture use case to isolate the relevant hand motion in each frame. Once the gesture is captured, the Translate Gesture and Extract Features use cases are triggered. These processes are powered by MediaPipe, which detects and maps hand landmarks—specific key points on the fingers and palm—and then converts them into structured data or feature vectors.

Next, the Match Features use case involves comparing these vectors against a pre-trained machine learning model, typically built with Teachable Machine or TensorFlow. The system identifies which known gesture the captured features most closely resemble.

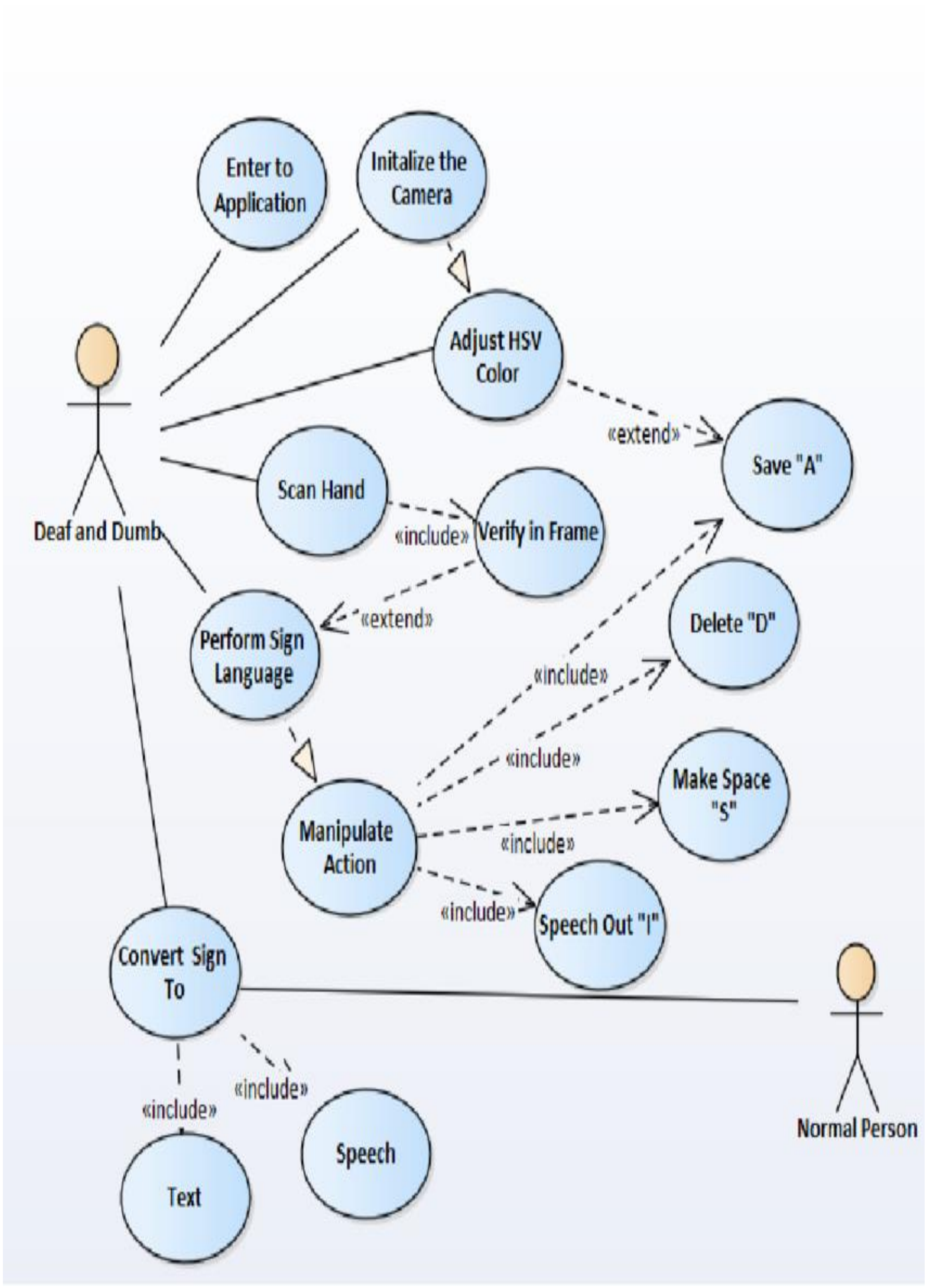


Figure No-4.2.1 Use case diagram

4.2.2 CLASS DIAGRAM:

The class diagram represents the object-oriented structure of an AI-powered Sign Language Interpreter system developed using Python and MediaPipe. The central class, SignLanguageInterpreter, manages the core functionality of the system. It holds two private attributes: images and words, both as lists, which store the training data and their corresponding textual representations. It offers two public methods: train(), used to train the model with image-word mappings, and translate(), which returns the translated sign language gesture as a string. This main class interacts with three subordinate classes: ImageDataset, Implementation, and HandTracker. The ImageDataset class maintains an image list and has a method load() responsible for loading the dataset used during training. The Implementation class contains an attribute labeled "oraich ve" and a method "traila", which seem to be placeholders or possibly typos, but likely refer to implementation-specific data and functionality. The HandTracker class includes a hand_position attribute of unspecified type (some_type) and a method track().

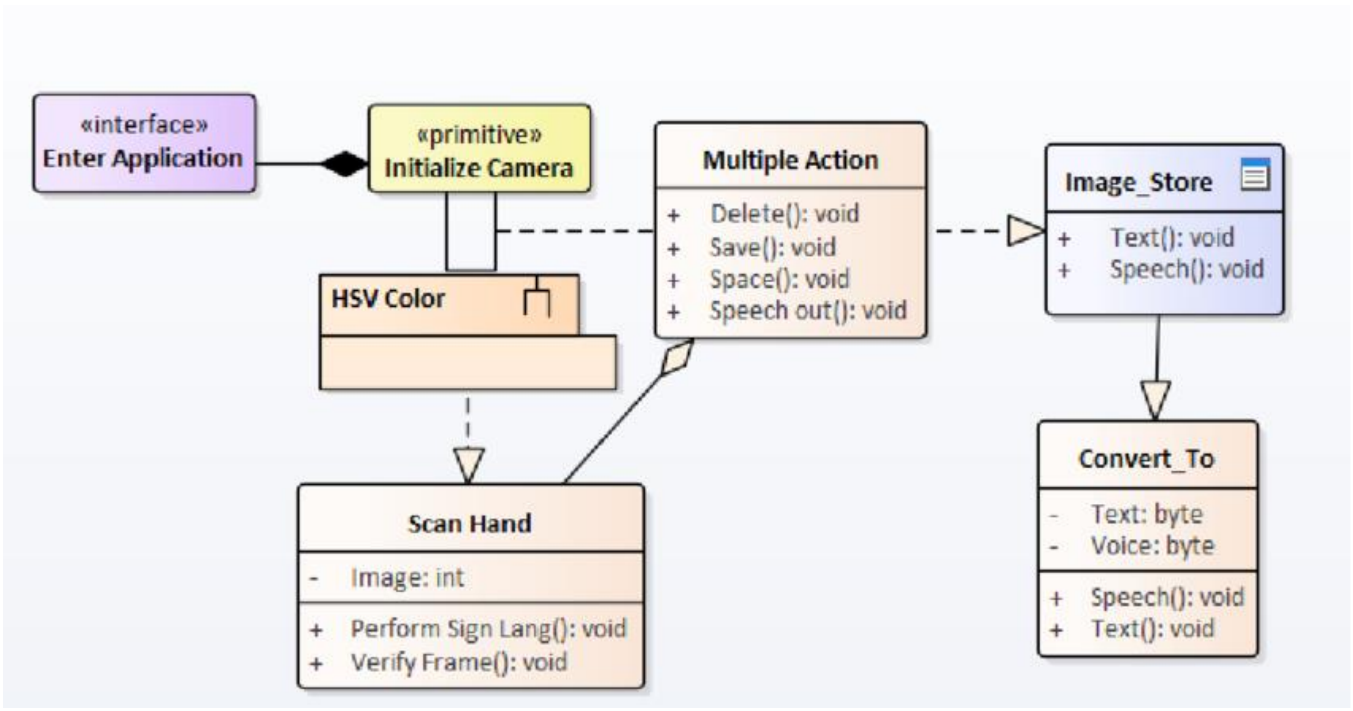


Figure No-4.2.2 Class Diagram

4.2.3 SEQUENCE DIAGRAM:

The sequence diagram illustrates the interaction flow of an AI-powered sign language interpreter using Python and MediaPipe. The user initiates the process through the GRE Form by setting domain, username, gesture type, and starting recognition. The system connects to the GloveDriver, which collects gesture data. The data is passed to MLPGenerator, which interacts with the MultiLayerPerceptron to create and train an artificial neural network (ANN) using ANNInfo. Once trained, the system recognizes gestures and retrieves corresponding symbols from the database via DBAccess.

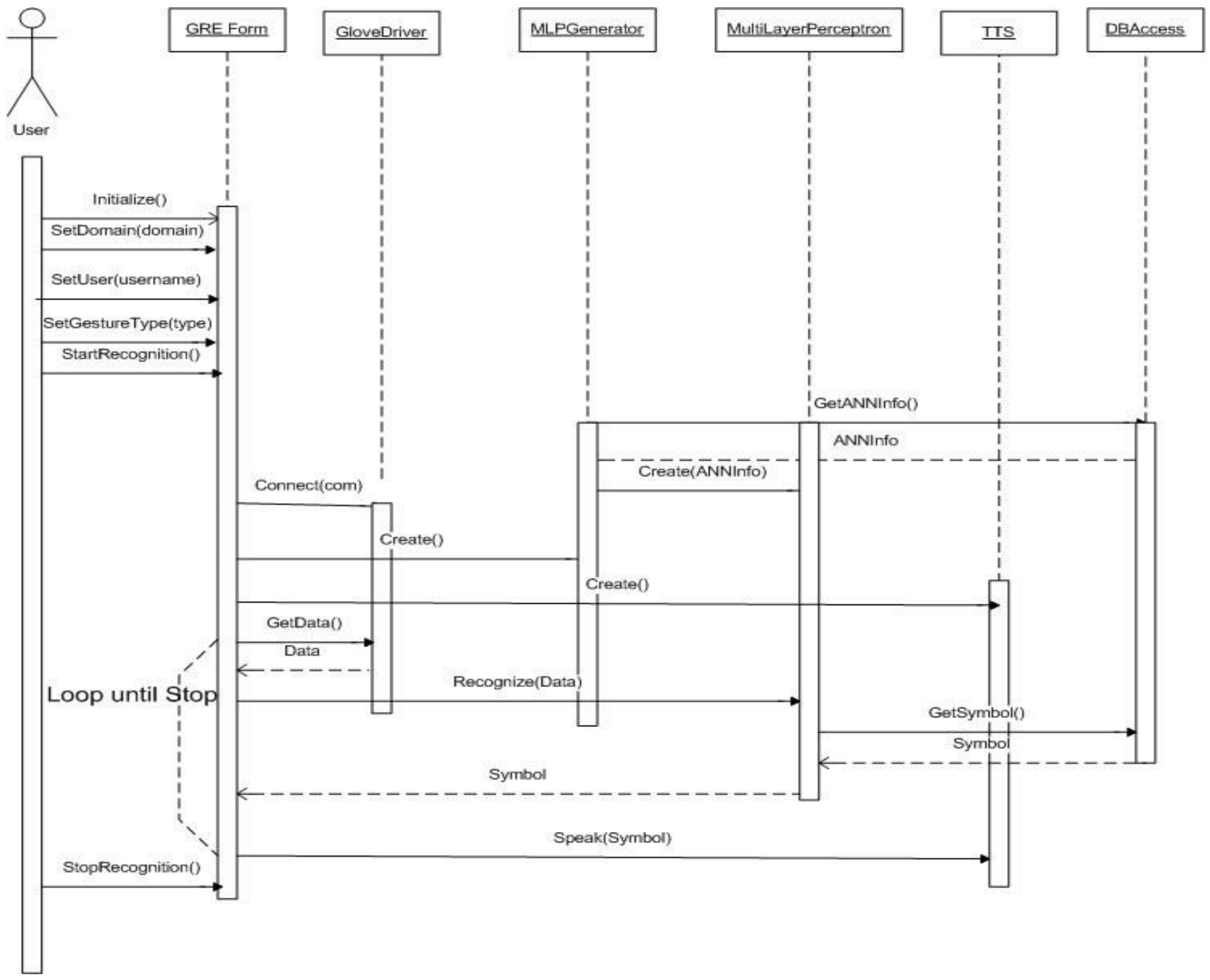


Figure No-4.2.3 Sequence diagram

4.2.4 ACTIVITY DIAGRAM:

The activity diagram outlines the process flow of an AI-powered sign language interpreter using Python and MediaPipe. The system begins by starting the webcam to capture live video input. It then detects the user's hand within the video frame using hand-tracking capabilities. Once the hand is detected, the image undergoes preprocessing to enhance recognition accuracy. The preprocessed image is then passed through a gesture prediction model to identify the sign being shown. Finally, the interpreted gesture is displayed as a result to the user. This structured workflow ensures real-time interpretation of sign language gestures into readable text.

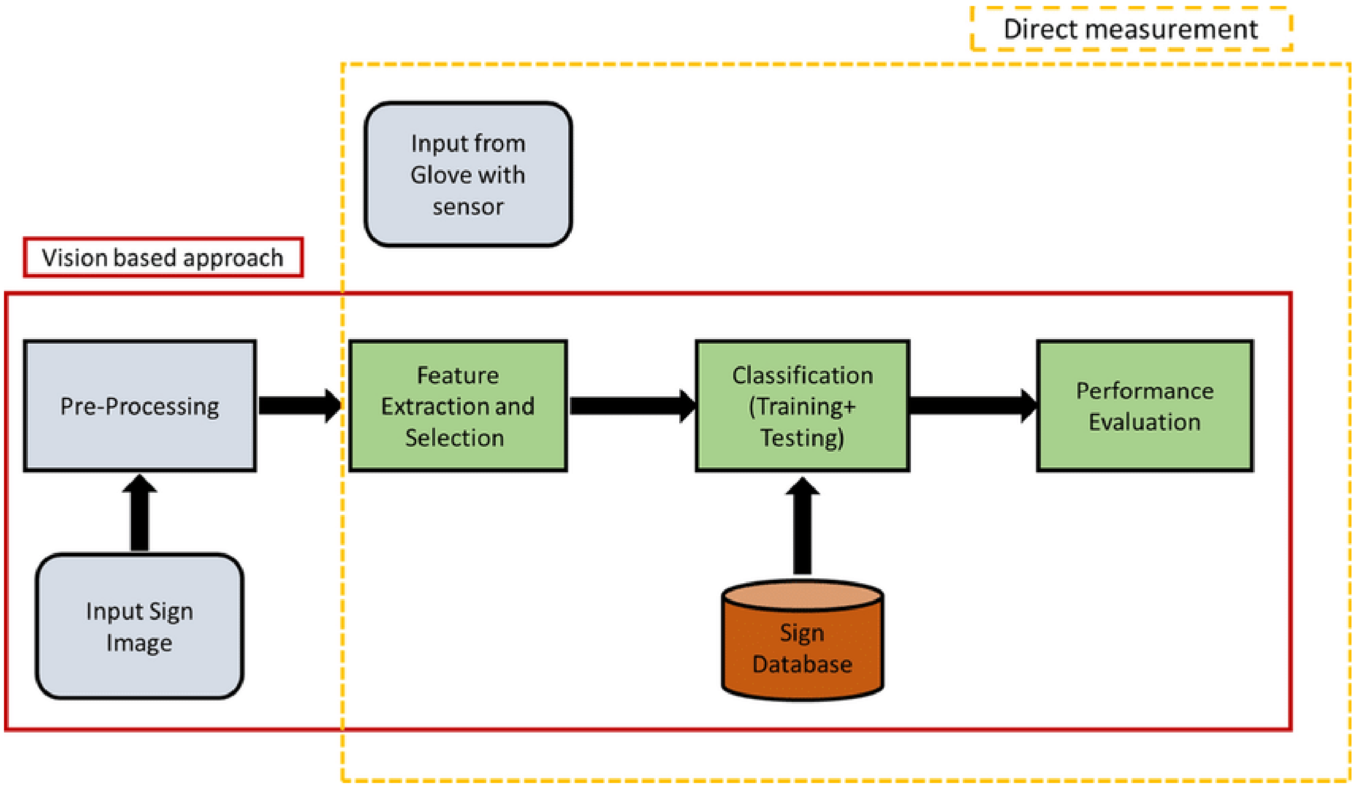
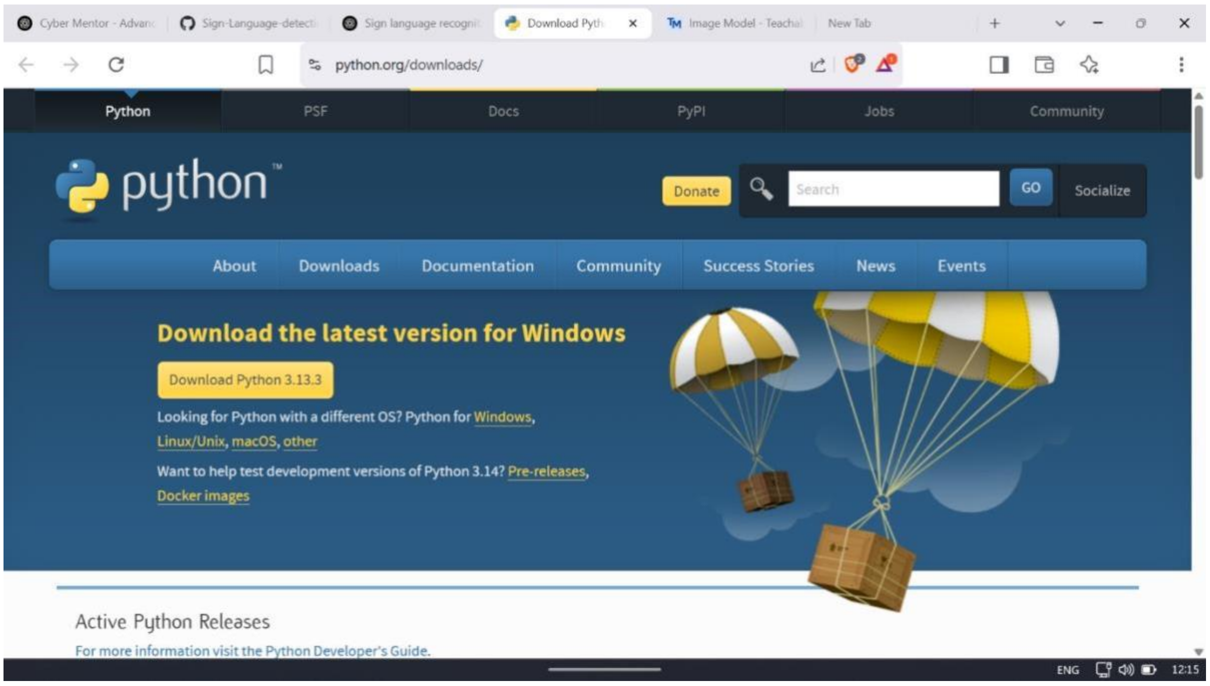


Figure No-4.2.3 Activity diagram

5. IMPLEMENTATION

The implementation of the project is done via the following steps by using Deep learning techniques, the process starts by downloading correct version of python into the system.

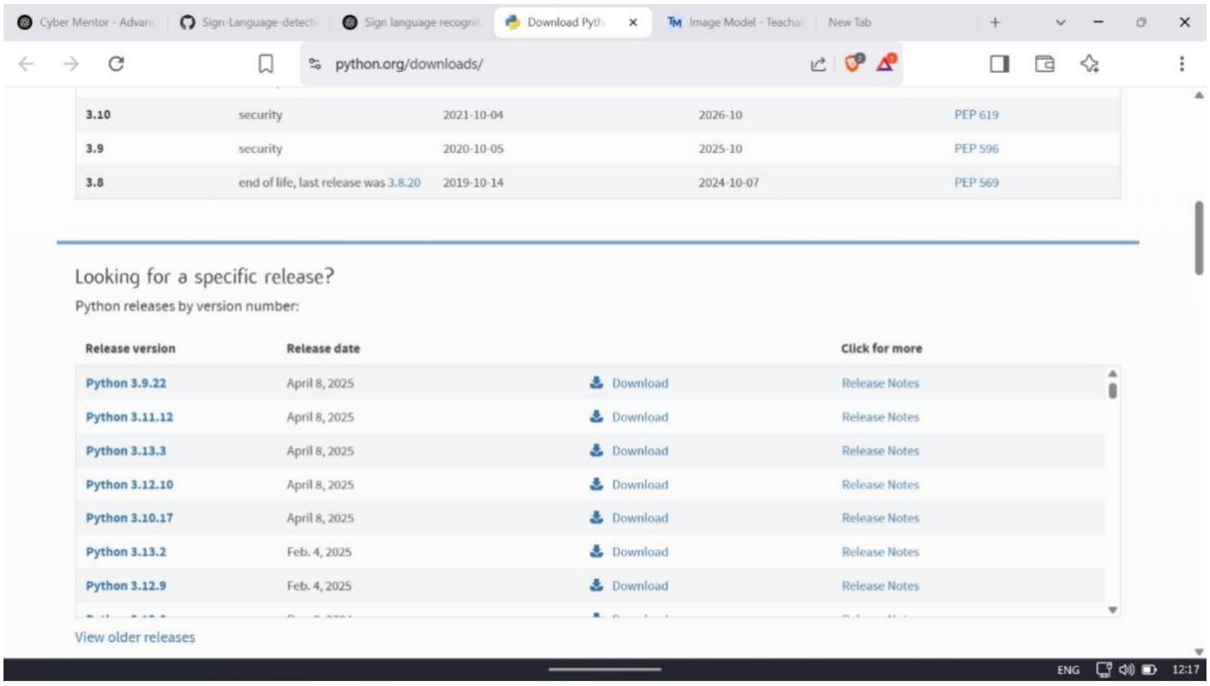
Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org



Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

Step 3: You can either select the Download Python for windows 3.7.0 button in Yellow Colour or you can scroll further down and click on download with respective to thei



3.10	security	2021-10-04	2026-10	PEP 619
3.9	security	2020-10-05	2025-10	PEP 596
3.8	end of life, last release was 3.8.20	2019-10-14	2024-10-07	PEP 569

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.9.22	April 8, 2025	Download	Release Notes
Python 3.11.12	April 8, 2025	Download	Release Notes
Python 3.13.3	April 8, 2025	Download	Release Notes
Python 3.12.10	April 8, 2025	Download	Release Notes
Python 3.10.17	April 8, 2025	Download	Release Notes
Python 3.13.2	Feb. 4, 2025	Download	Release Notes
Python 3.12.9	Feb. 4, 2025	Download	Release Notes

View older releases

version. Here, we are downloading the most recent python version for windows 3.7.0

- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 webbased installer.

- To download Windows 64-bit python, you can select any one from the three options:Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windowsx86-64 web-based installer**.**

- **Verify the Python Installation**

  Step 1: Click on Start

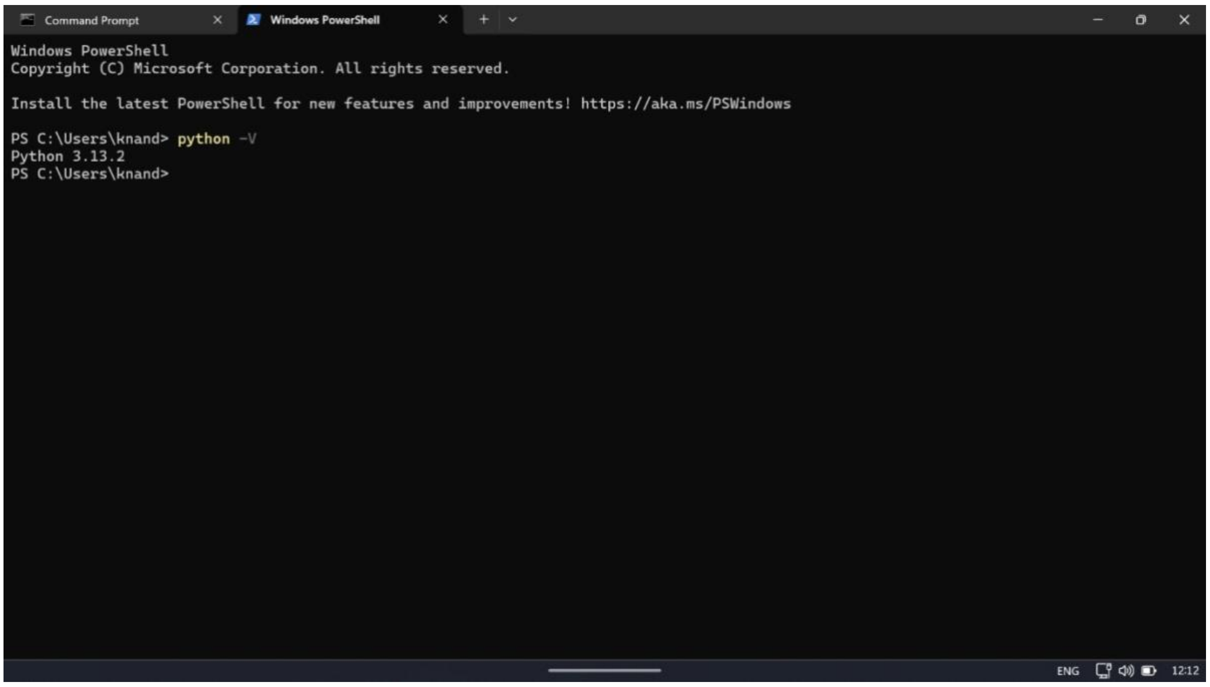
  Step 2: In the Windows Run Command, type —cmd‖.

  Step 3: Open the Command prompt option.

  Step 4: Let us test whether the python is correctly installed. Type python –version and press Enter.

  Step 5: You will get the answer as 3.7.0

  Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.



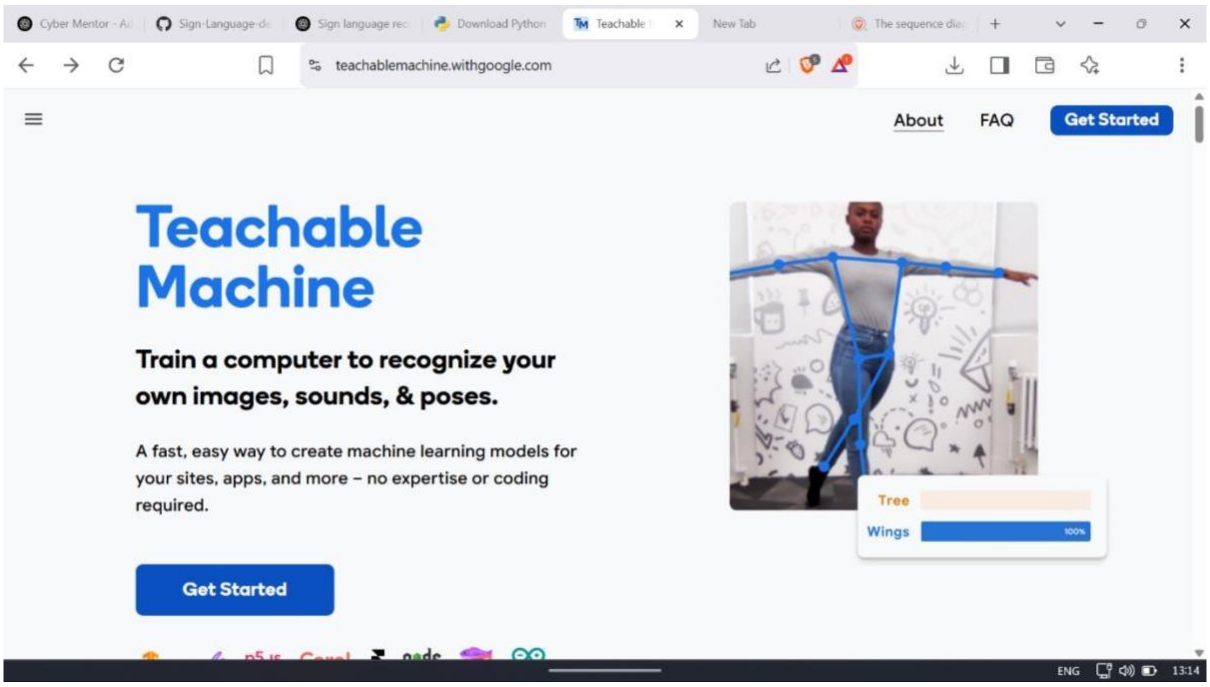
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\knand> python -V
Python 3.13.2
PS C:\Users\knand>
```

Next run the program.

6.IMPLEMENTATION OF TEACHABLE MACHINE:

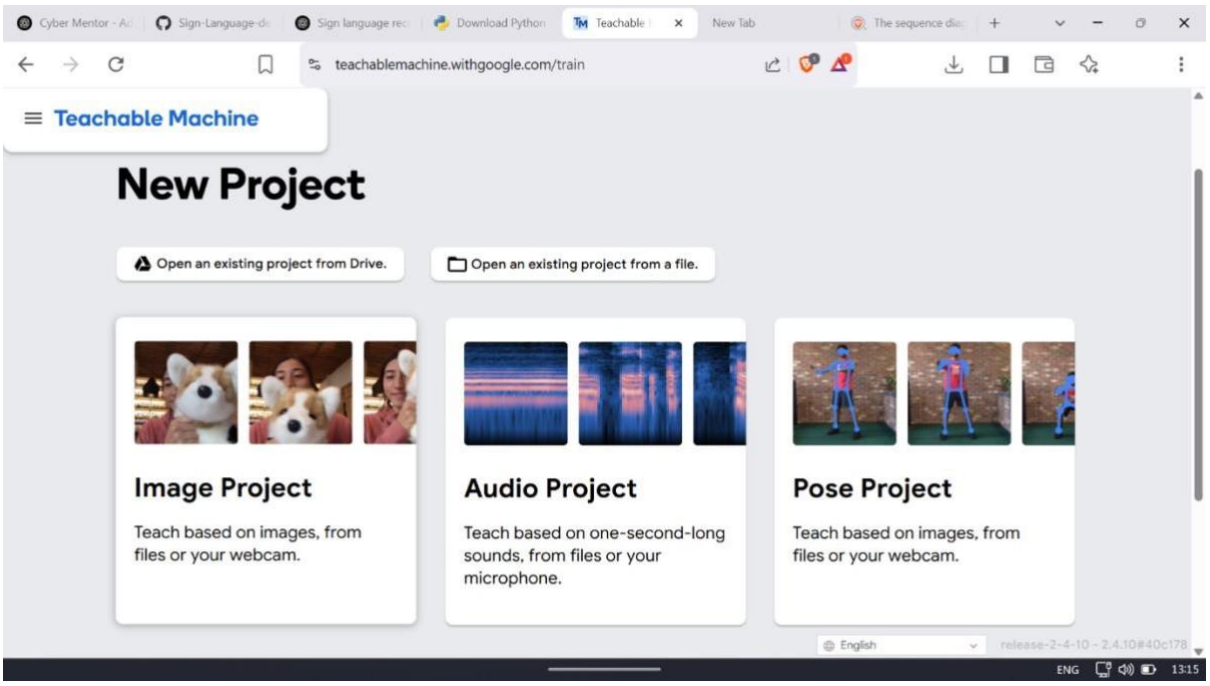
The implementation of the project is done via the following steps by using Teachable machine techniques, the process starts by downloading correct version of python into the system.

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://teachablemachine.withgoogle.com/

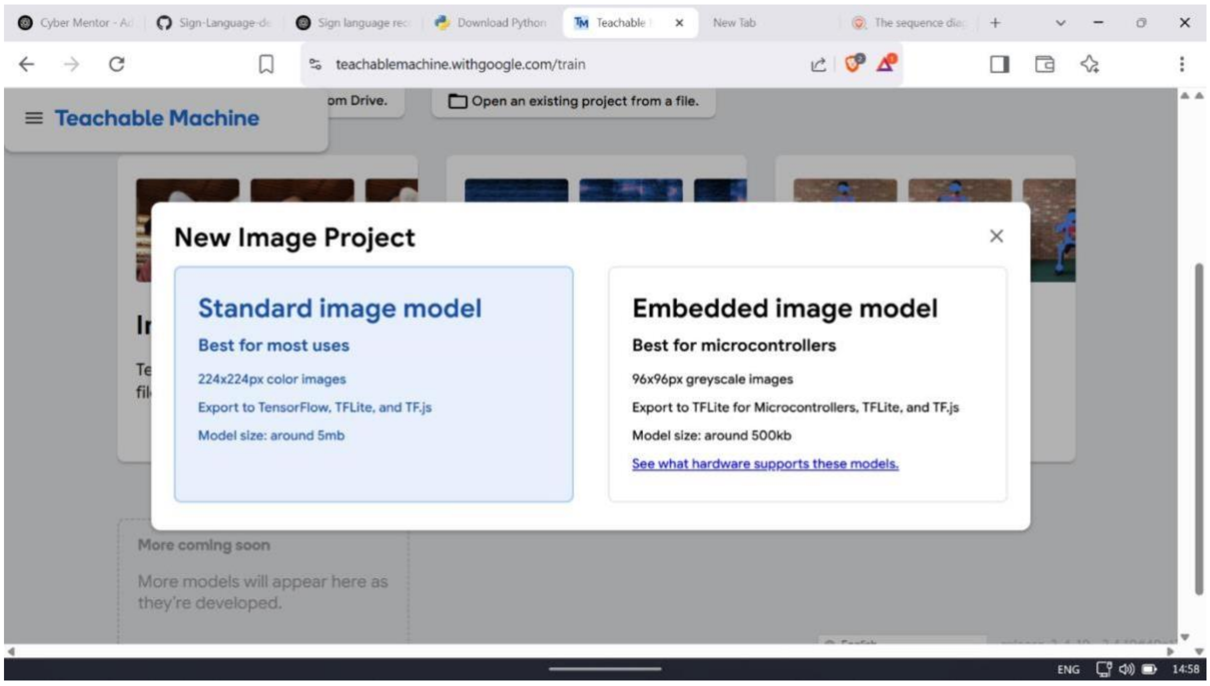


Cyber Mentor - Ad
Sign-Language-de
Sign language rec
Download Python
Teachable
New Tab
The sequence diag
teachablemachine.withgoogle.com
About
FAQ
Get Started
Teachable Machine
Train a computer to recognize your own images, sounds, & poses.
A fast, easy way to create machine learning models for your sites, apps, and more – no expertise or coding required.
Get Started
Tree
Wings
100%
ENG
13:14

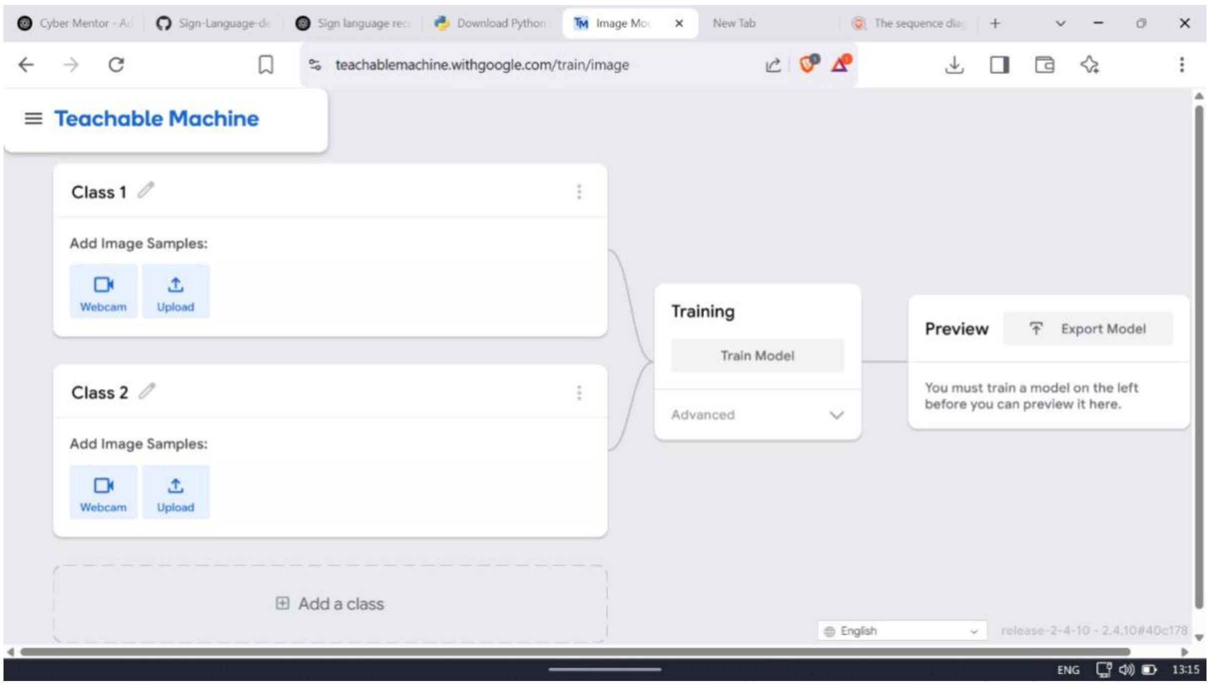
Step 2: Click on the Get Started.


teachablemachine.withgoogle.com/train
Teachable Machine
New Project
Open an existing project from Drive.
Open an existing project from a file.
Image Project
Teach based on images, from files or your webcam.
Audio Project
Teach based on one-second-long sounds, from files or your microphone.
Pose Project
Teach based on images, from files or your webcam.

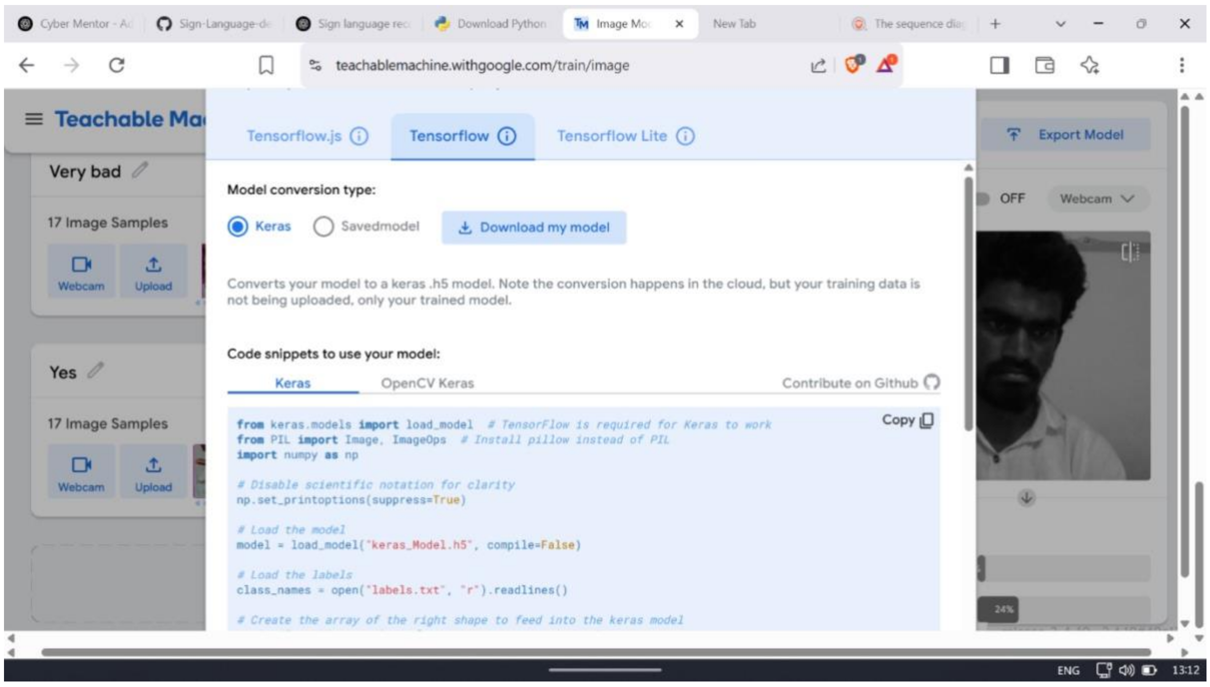
Step 3: Select on the Image Project Teach based on images,from files or your webcam.


teachablemachine.withgoogle.com/train
Teachable Machine
New Image Project
Standard image model
Best for most uses
224x224px color images
Export to TensorFlow, TFLite, and TF.js
Model size: around 5mb
Embedded image model
Best for microcontrollers
96x96px greyscale images
Export to TFLite for Microcontrollers, TFLite, and TF.js
Model size: around 500kb
See what hardware supports these models.
More coming soon
More models will appear here as they're developed.

Step 4: Select on the Standard image model option



Step 5: Write the class names and upload the those classes images,and click on the Training Model after the get the Explore model



Model conversion type:

[✓ Keras] Savedmodel Download my model

Converts your model to a keras .h5 model. Note the conversion happens in the cloud, but your training data is not being uploaded, only your trained model.

Code snippets to use your model:

Keras OpenCV Keras Contribute on Github

```
from keras.models import load_model  # TensorFlow is required for Keras to work
from PIL import Image, ImageOps  # Install pillow instead of PIL
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("keras_Model.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
```

Step 5: Exploring the Model click to the tensorflow,select on the Keras file and Download my Model

Step 6: After get in labels name and keras_model.h5 file are added in your desktop,those file path copy them past in source code

Labels.txt:

0 all the best

1 Hello

2 I love you

3 No

4 Okay

5 Please

6 Thank you

7 Very bad

8 Yes

Requide Libraries:

- opencv-python (for cv2)
- cvzone (for HandTrackingModule and ClassificationModule)
- numpy
- tensorflow (required by cvzone's ClassificationModule)
- pillow (sometimes needed for image handling)
- Other standard libraries (math, csv, datetime, time, os)

Install Commands:

Open your terminal (with your venv activated) and run:

- pip install opencv-python
- pip install cvzone
- pip install numpy
- pip install tensorflow
- pip install pillow

7. CODING

SORCE CODE:

Datacollection.py:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np import math import time
import os

cap = cv2.VideoCapture(0) detector =
HandDetector(maxHands=1)
offset = 20 imgSize
= 300
counter = 0

folder = "Data/Yes"
os.makedirs(folder, exist_ok=True)

# Set the main window to full screen (do this once, before the loop)
cv2.namedWindow('Image', cv2.WND_PROP_FULLSCREEN) cv2.setWindowProperty('Image',
cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)

while True:     success, img
= cap.read()     if not
success:
        print("Failed to grab frame")
        break

    hands, img = detector.findHands(img)     imgWhite =
np.ones((imgSize, imgSize, 3), np.uint8)*255     imgCrop =
None

    if hands:
hand = hands[0]
        x, y, w, h = hand['bbox']
        # --- Boundary check for cropping ---
y1 = max(0, y - offset)
        y2 = min(img.shape[0], y + h + offset)
x1 = max(0, x - offset)
        x2 = min(img.shape[1], x + w + offset)
imgCrop = img[y1:y2, x1:x2]
        # ----------------------------------

        if imgCrop.size != 0:
aspectRatio = h / w if w != 0 else 0

            if aspectRatio > 1:
k = imgSize / h                 wCal =
math.ceil(k * w)
                imgResize = cv2.resize(imgCrop, (wCal, imgSize))
wGap = math.ceil((imgSize-wCal)/2)                imgWhite[:,
wGap: wCal + wGap] = imgResize

else:
                k = imgSize / w if w != 0 else 0
hCal = math.ceil(k * h)
                imgResize = cv2.resize(imgCrop, (imgSize, hCal))
hGap = math.ceil((imgSize - hCal) / 2)
                imgWhite[hGap: hGap + hCal, :] = imgResize

            cv2.imshow('ImageCrop', imgCrop)
            cv2.imshow('ImageWhite', imgWhite)

    cv2.imshow('Image', img)

    key = cv2.waitKey(1)     if key == ord("s") and hands and imgCrop is not
None and imgCrop.size != 0:
        counter += 1
       cv2.imwrite(f'{folder}/Image_{time.time()}.jpg',   imgWhite)
print(counter)     elif key == ord("q"):
        break

cap.release() cv2.destroyAllWindows()
```

Test.py: import cv2 from cvzone.HandTrackingModule import HandDetector from cvzone.ClassificationModule import Classifier import numpy as np import math

```
cap = cv2.VideoCapture(0) detector =

HandDetector(maxHands=1)

classifier = Classifier(

    "C:/Users/knand/Desktop/model/converted_keras/keras_model.h5",

    "C:/Users/knand/Desktop/model/converted_keras/labels.txt"

)

offset = 20 imgSize

= 300

labels = [ "all the best","Hello","I love you","No", "Okay", "Please","Thank you","Very
bad","Yes"]

cv2.namedWindow('Image', cv2.WND_PROP_FULLSCREEN)

cv2.setWindowProperty('Image', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)


while True:

    success, img = cap.read()

if not success:

        print("Failed to grab frame")

        break
```

```
 imgOutput = img.copy()     hands, img =
detector.findHands(img)
    if hands:
        hand = hands[0]         x, y, w, h = hand['bbox']
imgWhite = np.ones((imgSize, imgSize, 3), np.uint8)*255

        # Safe boundary check for cropping
        y1 = max(0, y - offset)         y2 =
min(img.shape[0], y + h + offset)         x1 =
max(0, x - offset)         x2 =
min(img.shape[1], x + w + offset)
imgCrop = img[y1:y2, x1:x2]

        if imgCrop.size == 0:
            continue

        aspectRatio = h / w if w != 0 else 0         if aspectRatio > 1:              k =
imgSize / h             wCal = math.ceil(k * w)             imgResize =
cv2.resize(imgCrop, (wCal, imgSize))             wGap = math.ceil((imgSize-
wCal)/2)             imgWhite[:, wGap: wCal + wGap] = imgResize
prediction, index = classifier.getPrediction(imgWhite, draw=False)
            print(prediction, index)

        else:
            k = imgSize / w if w != 0 else 0             hCal = math.ceil(k * h)
imgResize = cv2.resize(imgCrop, (imgSize, hCal))             hGap =
math.ceil((imgSize - hCal) / 2)             imgWhite[hGap: hGap + hCal, :] =
imgResize            prediction, index = classifier.getPrediction(imgWhite,
draw=False)


        cv2.rectangle(imgOutput, (x-offset, y-offset-70), (x-offset+400, y-offset+60-50),
(0,255,0), cv2.FILLED)

        cv2.putText(imgOutput, labels[index], (x, y-30), cv2.FONT_HERSHEY_COMPLEX, 2,
(0,0,0), 2)        cv2.rectangle(imgOutput, (x-offset, y-offset), (x + w + offset, y + h + offset),
(0,255,0), 4)        cv2.imshow('ImageCrop', imgCrop)        cv2.imshow('ImageWhite',
imgWhite)     cv2.imshow('Image', imgOutput)      if cv2.waitKey(1) & 0xFF == ord('q'):
        break


cap.release() cv2.destroyAllWindows()
```

Test2.py:

```
import cv2

from cvzone.HandTrackingModule import
HandDetector

from cvzone.ClassificationModule import
Classifier

import numpy as np import

math  import csv from

datetime import datetime

import time import os


# Setup cap = cv2.VideoCapture(0)

detector = HandDetector(maxHands=1)

classifier = Classifier(


"C:/Users/knand/Desktop/model/conve
rted_keras/keras_model.h5",


"C:/Users/knand/Desktop/model/conve
rted_keras/labels.txt"

)

offset = 20 imgSize

= 300

labels = [ "all the best","Hello","I love
you","No", "Okay", "Please","Thank you","Very
bad","Yes"]



# Create directory to save test crops  if

not os.path.exists("SavedCrops"):

os.makedirs("SavedCrops")
```

```
# Open CSV file for logging

csv_file =
open('C:/Users/knand/Desktop/sign
language
detection/sign_language_predictions.cs
v', mode='w', newline='')  csv_writer =
csv.writer(csv_file)

csv_writer.writerow(["Timestamp", "Prediction",
"Confidence"])


# FPS tracking  pTime

= 0

cv2.namedWindow('Image', cv2.WND_PROP_FULLSCREEN)

cv2.setWindowProperty('Image',
cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)


while True:

   success, img = cap.read()      if

not success:         print("Failed to

grab frame")

       break


   imgOutput = img.copy()     hands, img

= detector.findHands(img)     if hands:

       hand = hands[0]         x,

y, w, h = hand['bbox']

       imgWhite = np.ones((imgSize,

imgSize, 3), np.uint8) * 255         #

Safe cropping         y1 = max(0, y -

offset)

       y2 = min(img.shape[0], y + h +

offset)        x1 = max(0, x - offset)
```

```
        x2 = min(img.shape[1], x + w +
 offset)        imgCrop = img[y1:y2,
 x1:x2]        aspectRatio = h / w if w !=
 0 else 0


        try:            if
 imgCrop.size == 0:
                raise Exception("Empty crop")
 if aspectRatio > 1:                k = imgSize
 / h                wCal = math.ceil(k * w)
                imgResize =
cv2.resize(imgCrop,  (wCal,  imgSize))
wGap = math.ceil((imgSize - wCal) / 2)
                imgWhite[:, wGap:wCal + wGap]
 = imgResize




            else:
                k = imgSize / w if w != 0 else 0
 hCal = math.ceil(k * h)
                imgResize =
cv2.resize(imgCrop,(imgSize,    hCal))
hGap = math.ceil((imgSize hCal)/2)
   imgWhite[hGap:hGap + hCal, :]=imgResize


 prediction,index=classifier.getPrediction
 (imgWhite, draw=False)
            # Robust confidence extraction for
 1D or 2D prediction
            if isinstance(prediction, (list,
 np.ndarray)) and len(prediction) > index:
```

```
                confidence =
round(float(prediction[index]) * 100, 2)

            elif hasattr(prediction, "_len") and
len(prediction) > 0 and
hasattr(prediction[0], "len_"):

                confidence =
round(float(prediction[0][index]) * 100,
2)

            else:

                confidence = 0.0

# Draw results

            cv2.rectangle(imgOutput, (x -
offset, y - offset - 70), (x - offset + 400, y
- offset + 60 - 50), (0, 255, 0), cv2.FILLED)




            cv2.putText(imgOutput,
f"{labels[index]} ({confidence}%)", (x, y - 30),
cv2.FONT_HERSHEY_COMPLEX, 1.5,
(0, 0, 0), 2)

            cv2.rectangle(imgOutput, (x - offset,
y - offset), (x + w + offset, y + h + offset),
(0, 255, 0), 4)


            # Log to CSV

            timestamp =
datetime.now().strftime('%Y-%m-%d %H:%M:%S')

            csv_writer.writerow([timestamp, labels[index],
confidence])

            csv_file.flush()


             # Show test images

            cv2.imshow('ImageCrop', imgCrop)
```

```
          cv2.imshow('ImageWhite', imgWhite)


          # Save crop automatically

          save_path = os.path.join(os.getcwd(),
"SavedCrops", f"Crop_{time.time()}.jpg")

          success = cv2.imwrite(save_path, imgCrop)

          print("Image saved:", success, "at",
save_path)




        except Exception as e:

          print("Error processing image:", e)


    # Show output and FPS

cTime = time.time()     fps = 1 /

(cTime - pTime + 0.001)      pTime

= cTime

    cv2.putText(imgOutput, f"FPS:
{int(fps)}", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
255, 255), 2)

cv2.imshow('Image', imgOutput)




    # Keyboard controls


    key = cv2.waitKey(1)     if

key == ord('s') and hands:
        print("imgCrop shape:", imgCrop.shape)

        save_path = os.path.join(os.getcwd(),
"C:/Users/knand/Desktop/sign language
detection/SavedCrops",
f"Crop_{time.time()}.jpg")

        success = cv2.imwrite(save_path, imgCrop)

        print("Image saved:", success, "at", save_path)



    elif key == ord('q'):

        break

# Cleanup

csv_file.close()  cap.release()

cv2.destroyAllWindows()
```

sign_language_predictions.csv

```
Timestamp,Prediction,Confidence

2025-06-01 13:37:15,all the best,67.87

2025-06-01 13:37:15,Hello,42.58

2025-06-01 13:37:16,Hello,83.66

2025-06-01 13:37:16,Hello,90.51

2025-06-01 13:37:16,Hello,96.05

2025-06-01 13:37:17,I love you,57.95
2025-06-01 13:37:17,Yes,40.02
2025-06-01 13:37:18,Yes,37.61
2025-06-01 13:37:18,Yes,45.59

2025-06-01 13:37:24,all the best,26.48
2025-06-01 13:37:24,No,78.83
2025-06-01 13:37:24,Very bad,72.86
2025-06-01 13:37:24,Very bad,69.55

2025-06-01 13:37:32,Thank you,57.44

2025-06-01 13:37:38,Okay,44.57
2025-06-01 13:37:39,Please,59.21
```

8. SYSTEM TEST

The purpose of testing is to discover errors. Testing is the 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. There are various types of tests. Each test type addresses a specific testing requirement.

TYPES OF TEST:

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals. Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted. Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration- o r i e n t e d system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a test in which the software tester has knowledge of the inner workings, structure, and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box. You cannot —see‖ into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach.

Field testing will be performed manually, and functional tests will be written in detail.

Test objectives.

All field entries must work properly.

Pages must be activated from the identified link.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined. Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration

9. OUT PUT SCREENS

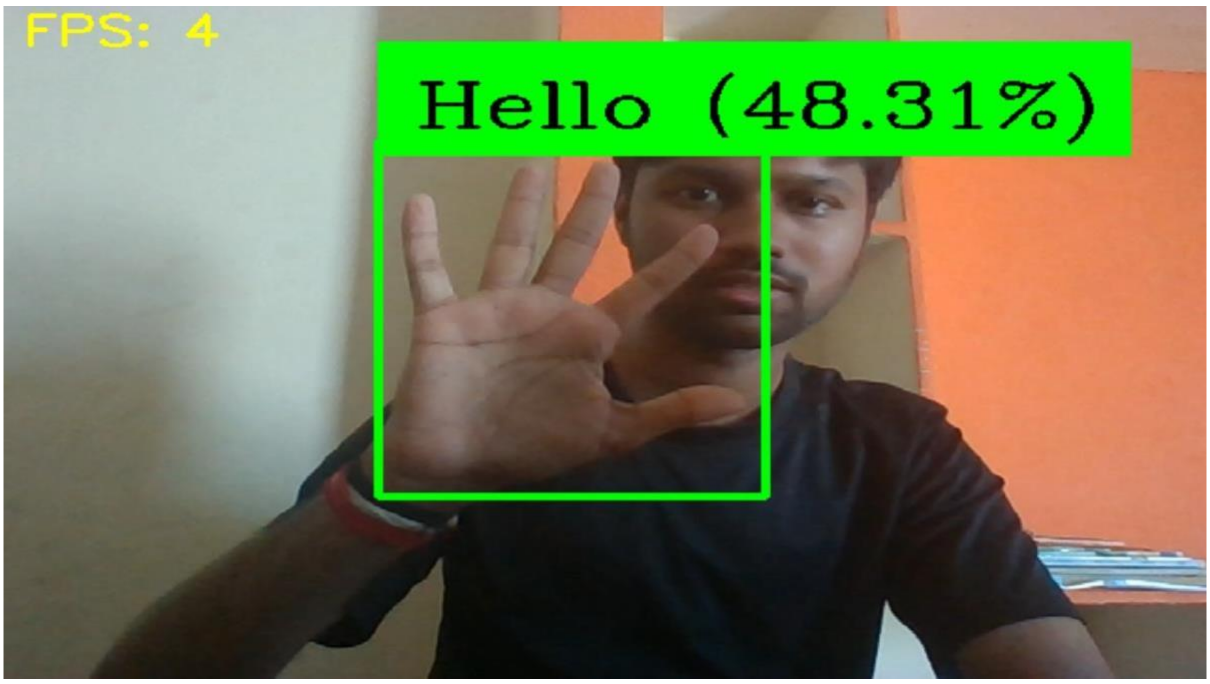


Figure No-8.1 Hello Sign

In above screen we can an ―Hello‖ message by showing the palm to the camera

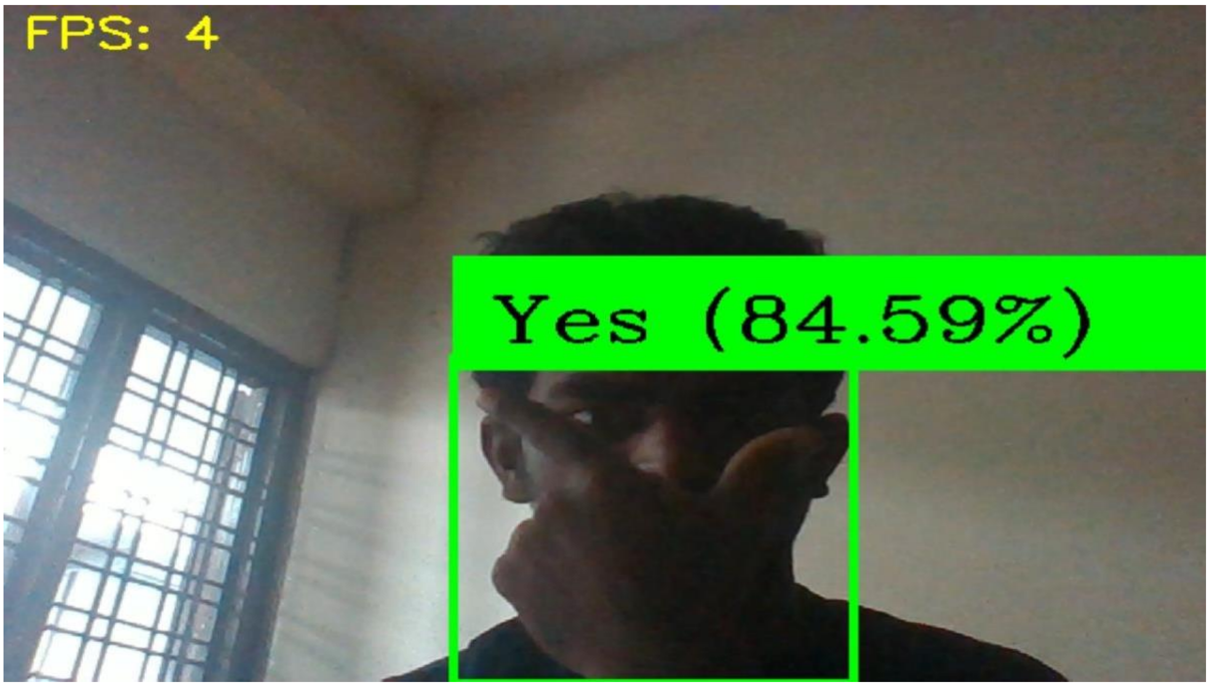


Figure No-8.2 Yes Sign

In above screen we can an —Yes‖ message by showing the Thumbs up to the camera

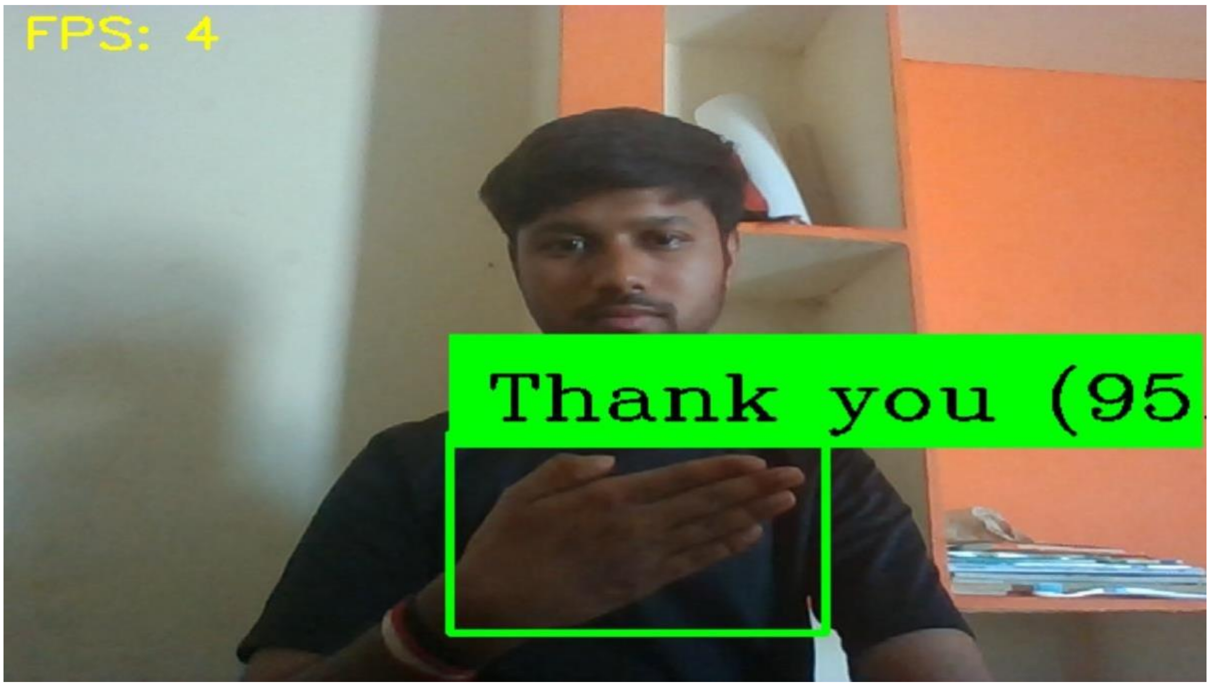


Figure No-8.3 Thankyou sign

In above screen we can an —Thankyou‖ message by showing the plam touching the lips to the camera

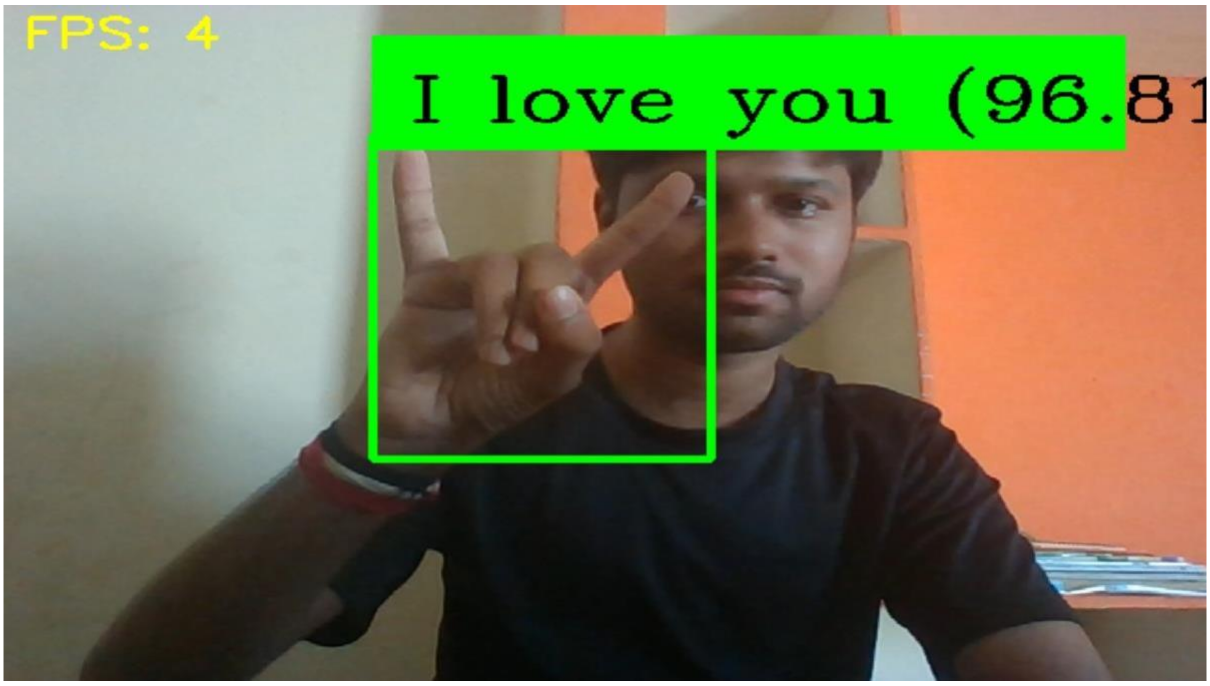


Figure No-8.4 Iloveyou sign

In above screen we can an ―Iloveyou‖ message by showing the swag to the camera

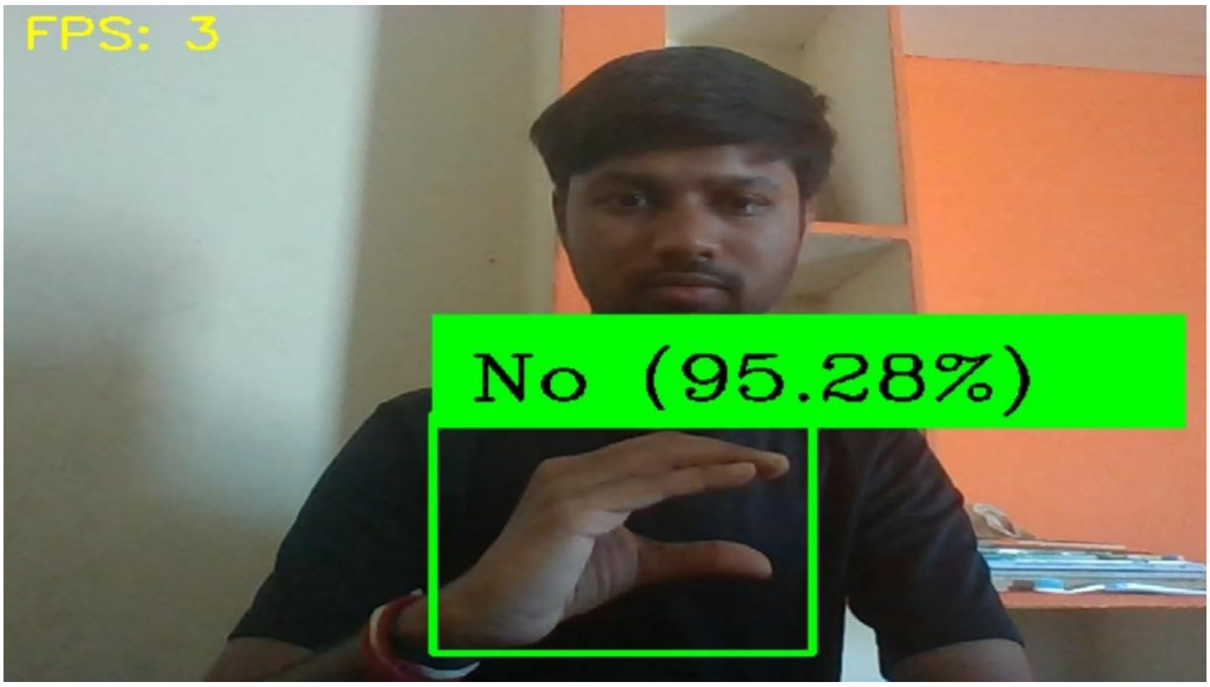


Figure No-8.5 NO sign

In above screen we can an ―NO‖ message by showing curved hand to the camera

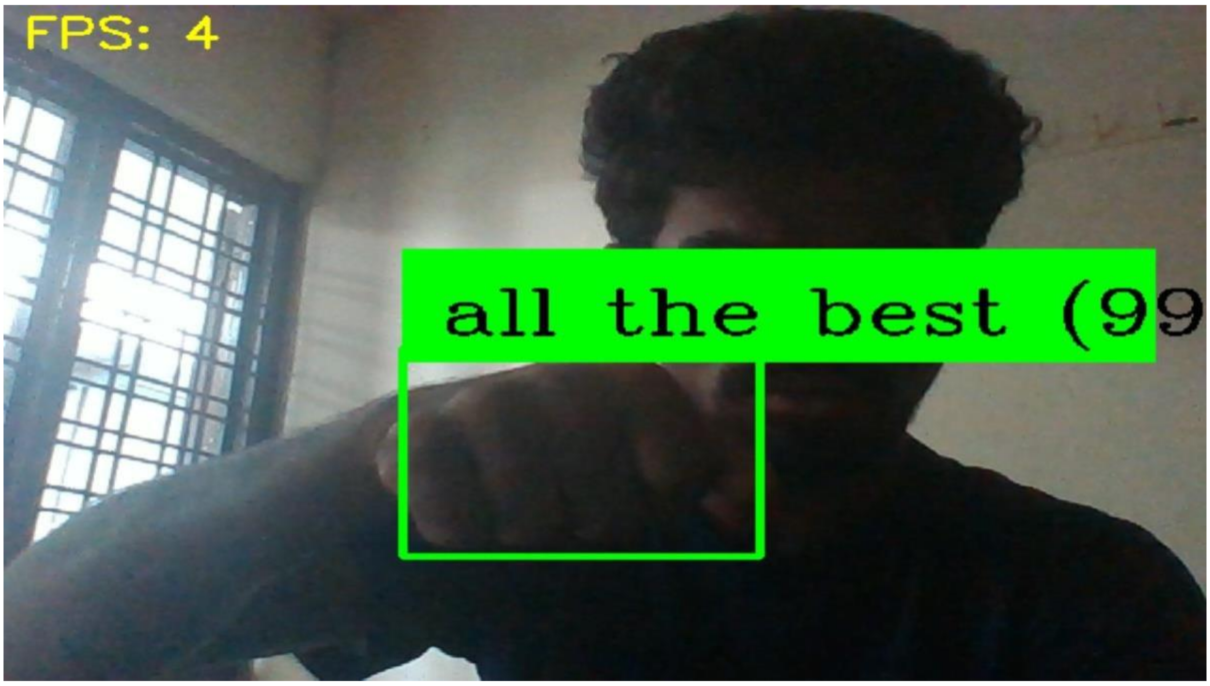


Figure No-8.6 Allthebest sign

In above screen we can an —Thankyou‖ message by showing the plam touching the lips to the camera

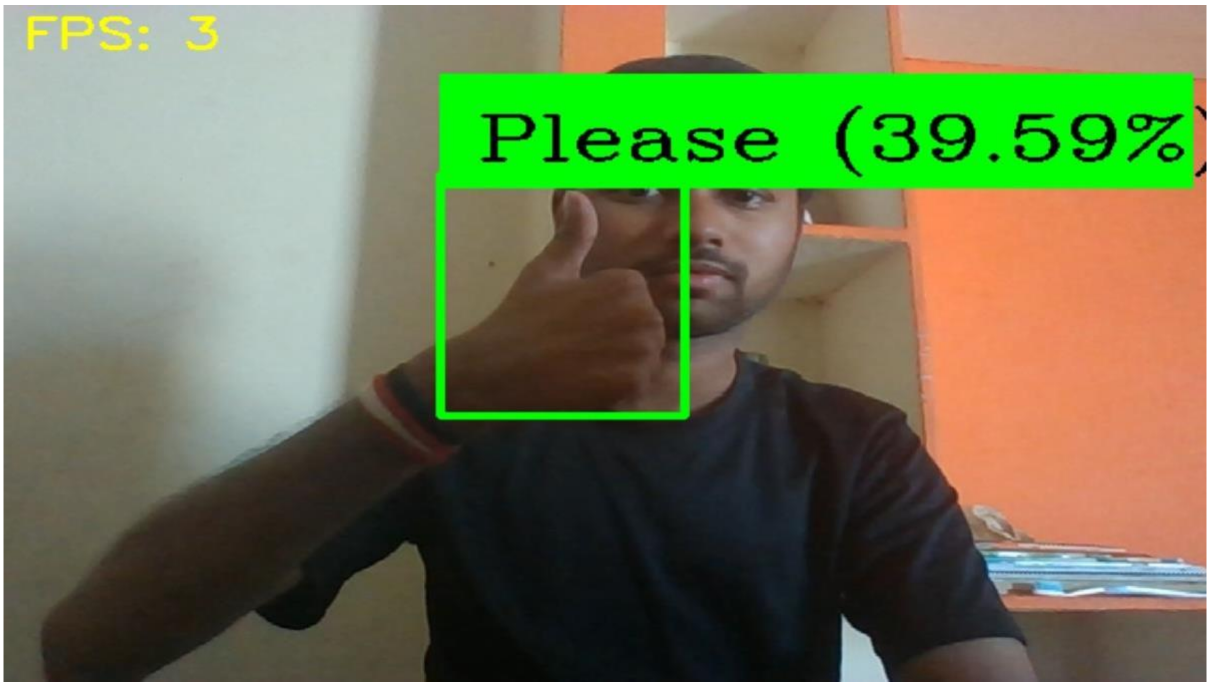


Figure No-8.7 Please

In above screen we can an —Please‖ message by showing fist to heart to the camera

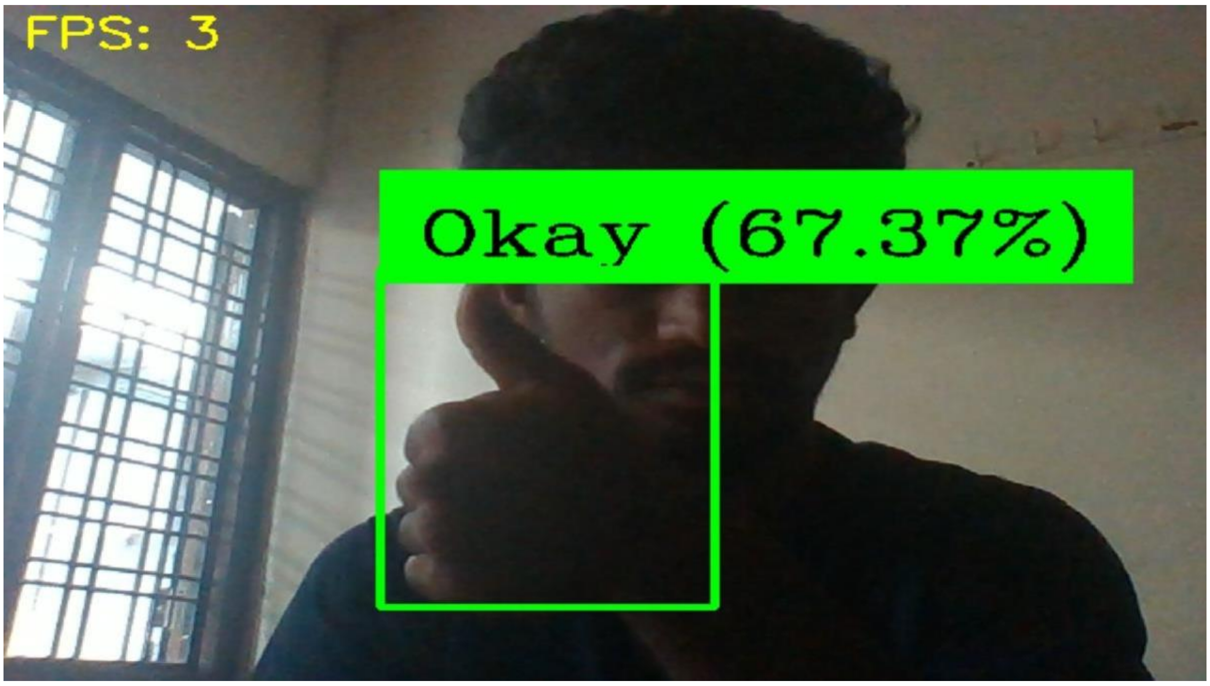


Figure No-8.8 Okay

In above screen we can an ―Okay‖ message by showing fist to upper to the camera

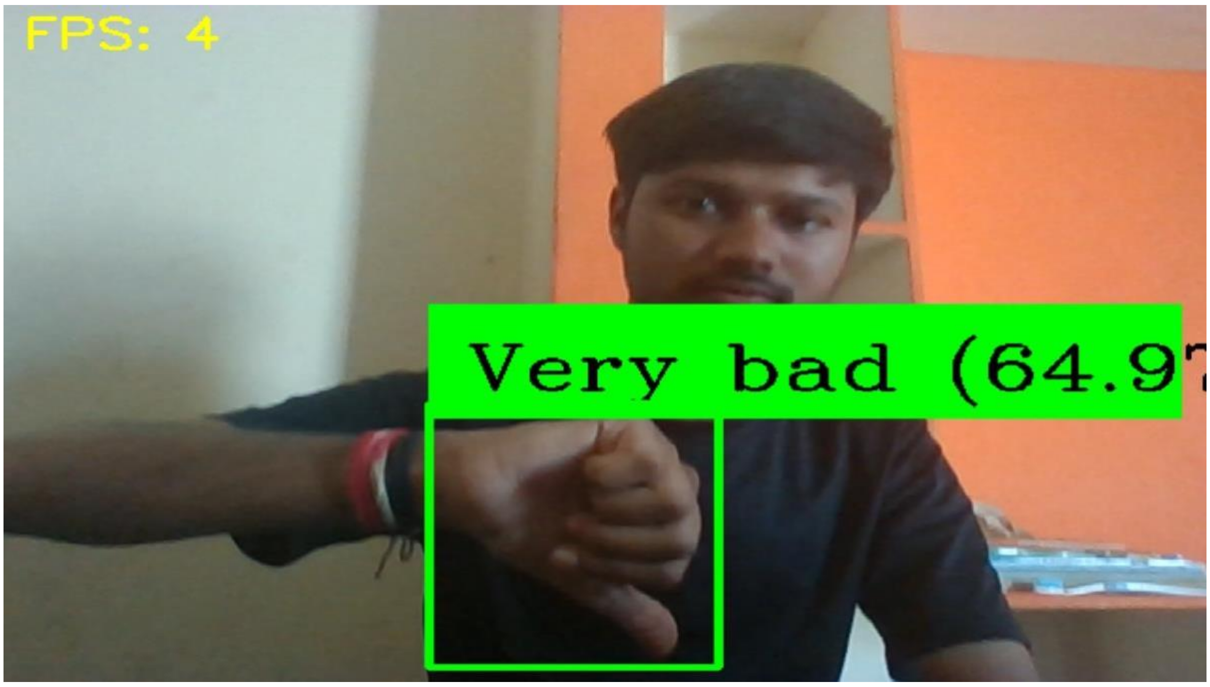


Figure No-8.9 Very bad

In above screen we can an ―Very bad‖ message by showing fist to down to the camera

10. CONCLUSION

In conclusion, the development of an AI-powered sign language interpreter using Python and MediaPipe represents a significant step toward bridging the communication gap between the hearing and speech-impaired communities and the rest of society. This innovative solution leverages computer vision and machine learning technologies to detect, process, and translate hand gestures into readable or audible language in real-time. By utilizing Python—a versatile and accessible programming language—and integrating it with MediaPipe's advanced hand tracking and landmark detection capabilities, developers are able to build systems that are not only efficient but also cost-effective and user-friendly. The system architecture typically involves starting a webcam to capture live hand movements, detecting hand positions using MediaPipe's built-in solutions, preprocessing captured images to enhance feature recognition, predicting gestures using trained models, and ultimately displaying or vocalizing the result to the end-user.

The adoption of this technology demonstrates how artificial intelligence can be tailored to serve socially impactful purposes. Unlike traditional systems that require expensive hardware such as sensor-based gloves or depth cameras, this AI-powered solution relies solely on standard webcams and software intelligence, making it highly scalable and accessible for a wider audience. Furthermore, the implementation of gesture recognition through a neural network or machine learning model ensures adaptability—wherein the system can be continuously trained and improved with more data, thereby increasing accuracy over time. This is particularly valuable in recognizing a vast vocabulary of signs, including letters, words, and potentially even dynamic phrases in sign language.

Another key strength of this interpreter system lies in its modular design, which allows for clear separation of functions such as hand detection, gesture prediction, and result display. This separation enhances maintainability and facilitates independent upgrades to different components of the system. MediaPipe, in particular, excels in this context by offering real-time performance and highly accurate landmark tracking without demanding extensive computational resources.

11. REFERENCES

- Aarons, D. (1994). Aspects of the Syntax of American Sign Language. PhD dissertation, Boston University.Google Scholar
- Aarons, D., Bahan, B., Kegl, J. & Neidle, C. (1992). Clausal structure and a tier for grammatical marking in American Sign Language. Nordic Journal of Linguistics, 15, 103–142.CrossRefGoogle Scholar
- Aarons, D., Bahan, B., Kegl, J. & Neidle, C. (1995). Lexical tense markers in American Sign Language. In Emmorey, K. & Reilly, J. S. (eds.), Language, Gesture and Space (pp. 225– 253). Hillsdale, NJ: Lawrence Erlbaum Associates.Google Scholar
- Abdel-Fattah, M. (2005). Arabic Sign Language: A perspective. Journal of Deaf Studies and Deaf Education, 10, 212–221.CrossRefGoogle ScholarPubMed
- Aboh, E. O., Pfau, R. & Zeshan, U. (2005). When a wh-word is not a wh-word: The case of Indian Sign Language. In Bhattacharya, T. (ed.), The Yearbook of South Asian Languages and Linguistics 2005 (pp. 11–43). Berlin: Mouton de Gruyter.Google Scholar