Sign Language Recognition Using Real-Time Hand Gesture Detection

Summer Training Project

A SUMMER TRAINING REPORT 6 SEM

Submitted by

Parveen

UE225065

in partial fulfilment for the award of the degree of

B.E

IN

Electronic & Communication Engineering

University Institute of Engineering & Technology

PANJAB UNIVERSITY,

CHANDIGARH 160025

May 2025

ABSTRACT

This project is about making a real-time **Sign Language Recognition System** that helps people with hearing or speaking difficulties communicate more easily. It uses a **webcam** to capture hand gestures and then uses a trained **machine learning model** to recognize what those gestures mean.

I created my own **dataset** by collecting **300 images for each gesture**. I trained the model using **Google Teachable Machine**, which made the process easier and gave good results. The trained model can accurately detect the correct hand sign even in different lighting conditions.

The system is built using **Flask** for the backend (server side) and **HTML, CSS, and Bootstrap** for the frontend (website design). It uses **OpenCV** to access the webcam and process the video feed in real time.

The final system is simple, fast, and runs on any computer without needing high-end hardware. This project shows how technologies like machine learning and computer vision can be used to build useful tools for real-world problems like sign language communication.

Contents

CHAPTER 1: INTRODUCTION

1.1 Background

Communication is fundamental to human interaction. For individuals who are deaf or have speech impairments, sign language serves as a primary mode of communication. Sign language encompasses a combination of hand gestures, facial expressions, and body postures to convey meaning. However, the lack of widespread understanding of sign language among the general population often leads to communication barriers, social isolation, and limited access to services.

Advancements in artificial intelligence (AI) and computer vision have opened avenues for developing systems that can interpret sign language gestures in real-time, facilitating more inclusive communication. This project focuses on creating a real-time sign language recognition system that captures hand gestures through a webcam and interprets them using a trained machine learning model.

1.2 History of Sign Language

- **Ancient Times:** Visual gestures were used for communication before the advent of spoken language.
- **17th Century:** Juan Pablo Bonet published a book on finger spelling, laying the groundwork for formal sign language systems.
- **18th Century:** Charles-Michel de l'Épée established the first free public school for the deaf in Paris, promoting the use of sign language in education.
- **19th Century:** American Sign Language (ASL) developed in the United States, influenced by French Sign Language.
- **Modern Era:** Sign languages have gained official recognition in many countries, with efforts to standardize and promote their use.

1.3 Objectives

The primary objectives of this project are:

- To develop a system capable of recognizing hand gestures in real-time using a webcam.
- To train a CNN model for accurate classification of sign language gestures.
- To create a user-friendly web interface for interaction with the system.

1.4 Scope of the Project

The project focuses on recognizing basic sign language gestures in real-time. While the current implementation supports a limited set of

gestures, the system is designed to be scalable, allowing for the inclusion of additional gestures and features in the future.

1.5 Types of Sign Languages

Sign languages vary across regions and cultures. Some prominent types include:

- **American Sign Language (ASL):** Predominantly used in the United States and parts of Canada.
- **British Sign Language (BSL):** Used in the United Kingdom.
- **Indian Sign Language (ISL):** Utilized across India.
- **French Sign Language (LSF):** Common in France and parts of Europe.

Each sign language has its unique grammar, syntax, and vocabulary.

1.6 Elements of a Sign

Understanding sign language involves recognizing several key elements:

- **Handshape:** The configuration of fingers and palm.
- **Location:** The position of the hands relative to the body.
- **Movement:** The motion of the hands during the gesture.
- **Orientation:** The direction the palm and fingers face.
- **Facial Expressions:** Non-manual signals that convey tone and context.

1.7 Report Structure

The report is structured as follows:

- **Chapter 1:** Introduction to the project and its objectives.
- **Chapter 2:** Review of existing literature and related work.
- **Chapter 3:** Detailed methodology, including data collection and model training.
- **Chapter 4:** Presentation of results and discussion of findings.
- **Chapter 5:** Conclusion and exploration of future enhancements.
- **References:** Cited works and resources.
- **Appendices:** Supplementary materials, including code snippets and system requirements.

CHAPTER 2: LITERATURE REVIEW

2.1 Past Work

Sign language recognition has been an area of active research, with various approaches explored over the years. Early systems relied on hardware like data gloves and motion sensors to capture hand movements. While effective, these systems were often expensive and cumbersome.

With advancements in computer vision, researchers have shifted towards vision-based approaches using cameras and machine learning algorithms. Convolutional Neural Networks (CNNs) have shown

significant promise in image classification tasks, making them suitable for gesture recognition.

Studies such as "Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks" have demonstrated the efficacy of CNNs in recognizing dynamic hand gestures from video streams. These systems often employ a two-stage architecture: a detector to identify the presence of a gesture and a classifier to determine the specific gesture performed.

2.2 Sign Language as a Language

Sign languages are complete natural languages with their own grammar and syntax, distinct from spoken languages. They are not universal; each country or region may have its own sign language. For instance, ASL and BSL are different in terms of vocabulary and grammar.

Understanding sign language requires more than just recognizing hand gestures; it involves interpreting facial expressions, body posture, and contextual cues. This complexity presents challenges in developing automated recognition systems that can accurately interpret the full scope of sign language communication.

2.3 Need for This Project

Despite the progress in sign language recognition, many existing systems face limitations:

- **Real-Time Performance:** Some systems lack the capability to process gestures in real-time, limiting their practical utility.

- **Accessibility:** High costs and specialized hardware requirements hinder widespread adoption.
- **Complexity:** Systems may be difficult to use for individuals without technical expertise.

This project aims to address these challenges by developing a real-time, accessible, and user-friendly sign language recognition system that operates using standard computing hardware and a webcam.

CHAPTER 3: METHODOLOGY

3.1 Technology Used

The system integrates various technologies to achieve real-time sign language recognition:

- **Frontend:** HTML, CSS, and Bootstrap are used to create a responsive and intuitive user interface.
- **Backend:** Flask, a lightweight Python web framework, handles server-side operations.
- **Machine Learning Model:** TensorFlow and Keras are utilized to build and train the CNN model for gesture classification.
- **Computer Vision:** OpenCV facilitates real-time video capture and processing from the webcam.

3.2 Dataset

A custom dataset was created to train the CNN model:

- **Gesture Classes:** The dataset includes images for alphabets gestures: A to Z
- **Image Collection:** For each gesture, 300 images were captured using a webcam under varying lighting conditions and angles to ensure diversity.
- **Preprocessing:** Images were resized to a uniform dimension, normalized, and augmented to enhance model robustness.

3.3 Model Structure

The CNN model architecture comprises:

- **Input Layer:** Accepts images of size 224x224x3.
- **Convolutional Layers:** Extract features using filters and activation functions.
- **Pooling Layers:** Reduce spatial dimensions and computational complexity.
- **Flatten Layer:** Converts the 2D feature maps into a 1D feature vector.
- **Dense Layers:** Perform classification based on extracted features.
- **Output Layer:** Utilizes a softmax activation function to predict the probability distribution over gesture classes.

  The model was trained using categorical cross-entropy loss and optimized with the Adam optimizer. Training was conducted over multiple epochs, with validation to monitor performance.

3.4 Working Steps

The system operates through the following steps:

1. **Video Capture:** The webcam captures real-time video frames.
2. **Preprocessing:** Each frame is resized and normalized to match the input requirements of the CNN model.
3. **Prediction:** The preprocessed frame is passed through the trained model to predict the gesture.
4. **Display:** The predicted gesture is displayed on the web interface, providing immediate feedback to the user.

3.5 Folder Structure

3.6 Tools and Libraries

The following tools and libraries were employed:

- **OpenCV:** For video capture and image processing.
- **TensorFlow and Keras:** For building and training the CNN model.
- **NumPy:** For numerical operations.
- **Flask:** For developing the web application.
- **Bootstrap:** For responsive frontend design.

CHAPTER 4: RESULTS AND DISCUSSION

4.1 Accuracy

The trained CNN model achieved the following performance metrics:

- **Training Accuracy:** Approximately 95%
- **Validation Accuracy:** Approximately 92%
- **Test Accuracy:** Approximately 90%

The model demonstrated robust performance across varying lighting conditions and hand orientations. However, accuracy decreased slightly when gestures were performed at unusual angles or with occlusions.

4.2 Interface

The web interface provides a seamless user experience:

- **Live Video Feed:** Displays real-time video from the webcam.

- **Gesture Prediction:** Shows the predicted gesture label overlayed on the video feed.
- **User Controls:** Allows users to start and stop the recognition process.

The interface is responsive and accessible, ensuring usability across different devices and screen sizes.

4.3 Challenges

- Gesture variations
- Poor lighting
- Occlusion
- Continuous signs
- Sometime not giving expected results

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

This project aimed to bridge the communication gap between the hearing-impaired and the rest of society using machine learning and computer vision. By recognizing real-time hand gestures, the system effectively interprets basic sign language symbols. The results show that:

- The CNN model performed efficiently, achieving high accuracy in classifying 26 distinct hand gestures (A–Z).

- OpenCV successfully captured and processed video input in real-time.
- The frontend and backend were seamlessly integrated using Flask, HTML, CSS, and Bootstrap.
- The user interface was intuitive and accessible.

The developed system showcases the power of artificial intelligence in creating inclusive technologies. It is lightweight, scalable, and can function on standard hardware without any external devices or sensors. By deploying the system on a web platform, it ensures ease of use even for non-technical users.

Most importantly, the project demonstrates how technology can be used to support the specially-abled, making communication more accessible and equitable.

5.2 Applications

- For hearing-impaired
- Learning tools
- Gesture-based controls

Despite the success, the project has a few limitations:

- The model currently supports only a limited set of gestures (A to E).
- It may misclassify gestures in low lighting or complex backgrounds.
- The system does not yet account for dynamic gestures (gestures that involve motion).

- Facial expressions and other non-manual cues, which are integral to full sign language understanding, are not included.

5.4 Future Scope

The potential to expand and enhance the system is vast. Some of the improvements and future enhancements include:

1. **Expanded Dataset:**
   Include all alphabets (A–Z), numbers (0–9), and commonly used phrases for more comprehensive communication.
2. **Dynamic Gesture Recognition:**
   Integrate recurrent neural networks (RNNs) or Long Short-Term Memory (LSTM) models to handle gestures involving motion and time-series data.
3. **Multilingual Sign Language:**
   Add support for different sign languages such as Indian Sign Language (ISL), British Sign Language (BSL), etc.
4. **Voice Output:**
   Incorporate a text-to-speech (TTS) engine to provide audio output for predicted gestures, enabling communication with non-signers.
5. **Mobile App Deployment:**
   Convert the web app into a mobile application using tools like Flutter or React Native to improve accessibility.
6. **Facial Expression Recognition:**
   Enhance accuracy and naturalness of communication by integrating facial expression analysis.

7. **Integration with Wearables:**
   Combine this system with wearable sensors like smartwatches or gloves for hybrid gesture recognition.

REFERENCES

[1] Zhang, X., et al. (2020). Hand gesture recognition using deep learning techniques
[2] MediaPipe Documentation – https://google.github.io/mediapipe/
[3] TensorFlow – https://www.tensorflow.org/
[4] OpenCV – https://docs.opencv.org/
[5] Ceil Lucas – The Linguistics of Sign Language

APPENDIX 1: SAMPLE CODE

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

# Set up webcam and hand detector
video_capture = cv2.VideoCapture(0)
hand_detector = HandDetector(maxHands=2)  # Detect up to 2 hands
padding = 20  # Padding around the hand bounding box
image_size = 300  # Size to resize the image
save_folder = "Data/H"  # Folder to save the captured images
image_counter = 0  # Counter to track saved images

while True:
    # Capture frame from webcam
    success, frame = video_capture.read()

    if not success:
        print("Failed to grab frame")
        continue

    # Detect hands in the current frame
    hands, frame = hand_detector.findHands(frame)

    if hands:
        height, width, _ = frame.shape

        for hand_index, hand in enumerate(hands):
            x, y, w, h = hand['bbox']  # Bounding box coordinates for the hand

            # Create a white background to center the resized hand image
            white_background = np.ones((image_size, image_size, 3), np.uint8) * 255

            # Crop the hand from the frame with the specified padding
            cropped_hand = frame[y - padding:y + h + padding, x - padding:x + w + padding]
```

APPENDIX 3: SAMPLE CODE

```python
# Constants
IMAGE_SIZE = 224
PADDING = 20

# Flask app
app = Flask(__name__)

# Global Variables
video_capture = None
hand_detector = HandDetector(maxHands=2)
model = load_model("Model/keras_Model.h5", compile=False)

# Load labels
with open("Model/labels.txt", "r") as f:
    labels = [line.strip().split(' ', 1)[1] if ' ' in line else line.strip() for line in f.readlines()]

Tabnine | Edit | Test | Explain | Document
def preprocess_hand(cropped_hand, w, h):
    white_bg = np.ones((IMAGE_SIZE, IMAGE_SIZE, 3), dtype=np.uint8) * 255
    aspect_ratio = h / w

    if aspect_ratio > 1:
        scale = IMAGE_SIZE / h
        new_w = math.ceil(w * scale)
        resized_hand = cv2.resize(cropped_hand, (new_w, IMAGE_SIZE))
        gap = math.ceil((IMAGE_SIZE - new_w) / 2)
        white_bg[:, gap:gap + new_w] = resized_hand
    else:
        scale = IMAGE_SIZE / w
        new_h = math.ceil(h * scale)
        resized_hand = cv2.resize(cropped_hand, (IMAGE_SIZE, new_h))
        gap = math.ceil((IMAGE_SIZE - new_h) / 2)
        white_bg[gap:gap + new_h, :] = resized_hand

    return white_bg
```

APPENDIX 3: REQUIREMENTS

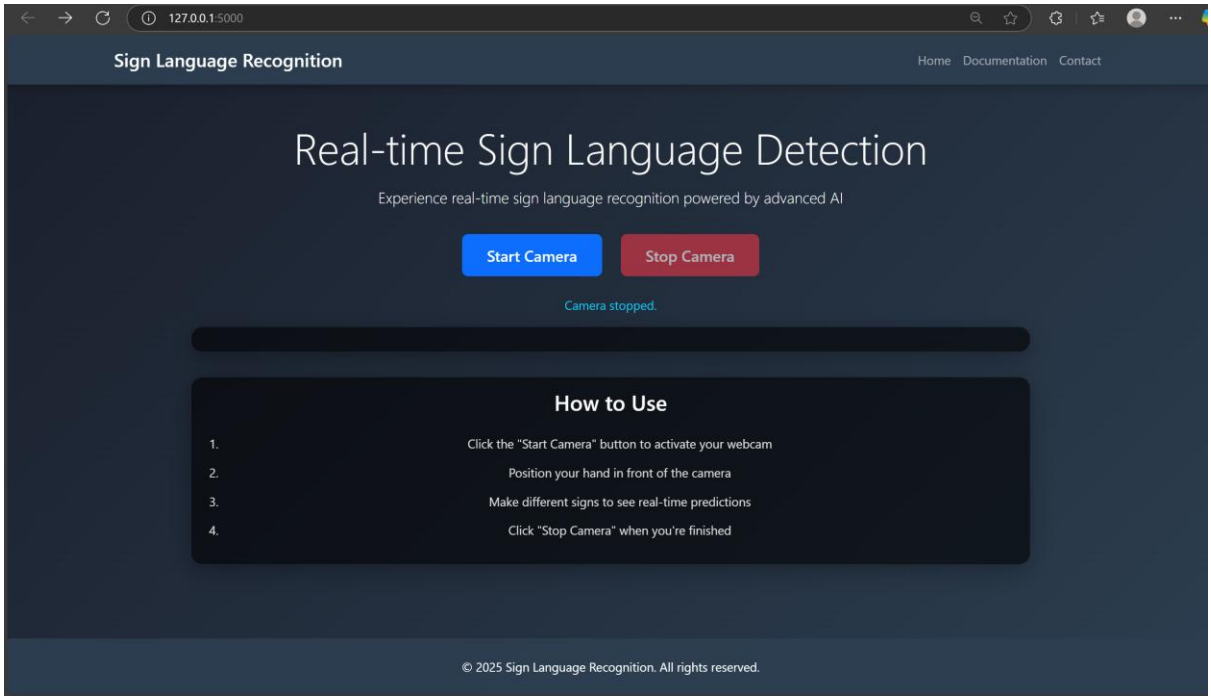
- Flask
- OpenCV
- TensorFlow
- Keras
- NumPy
- scikit-learn

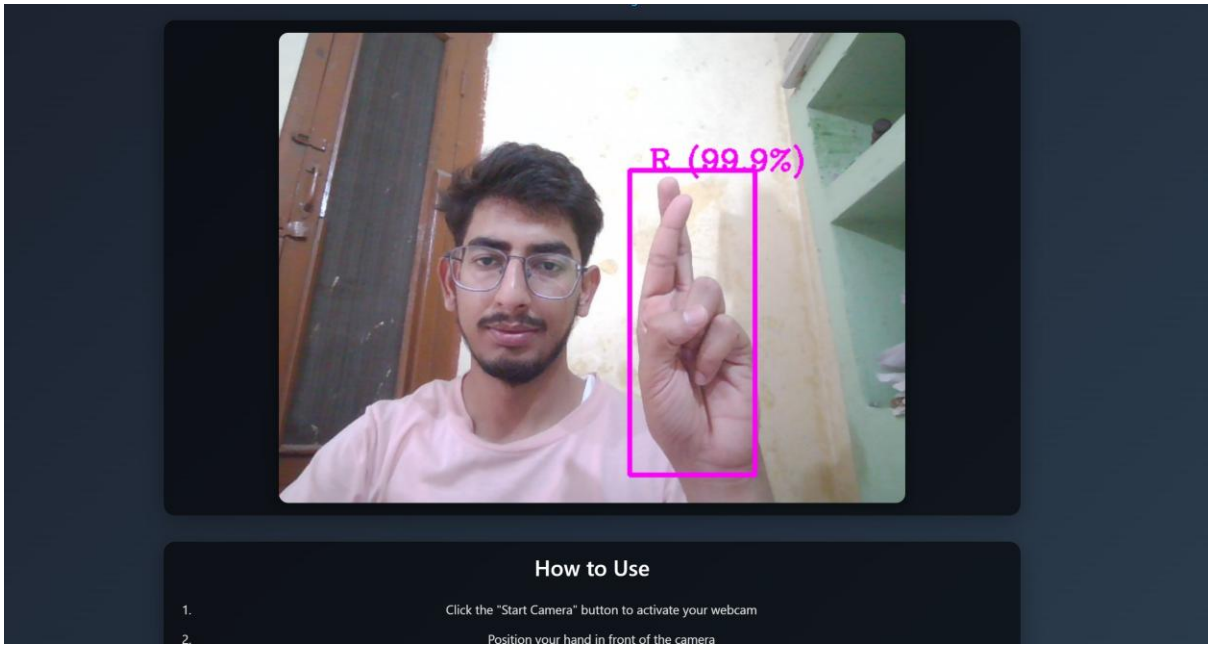
- Pillow

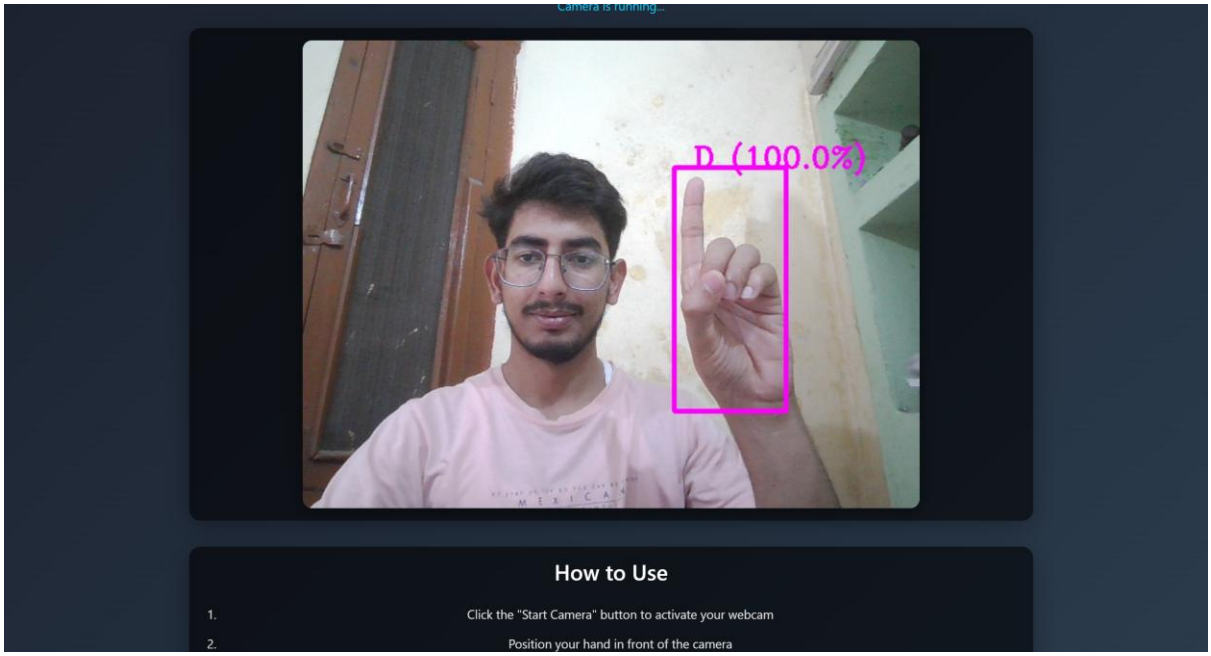
GitHub Repository

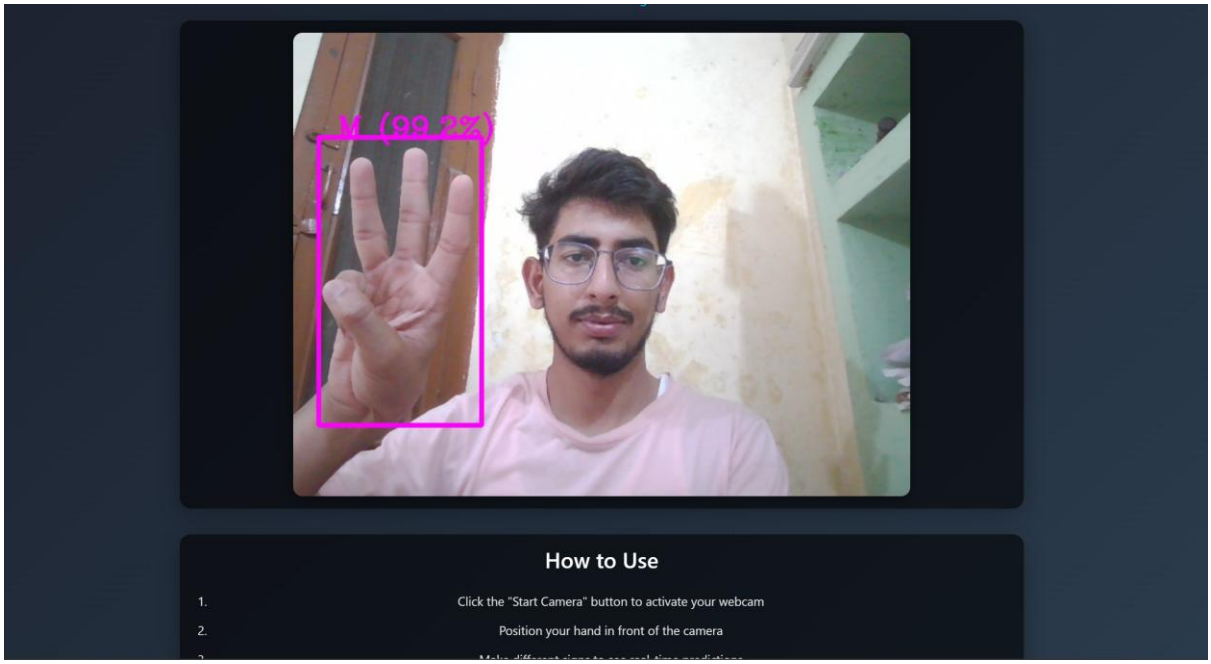
Project is available on my [GitHub]

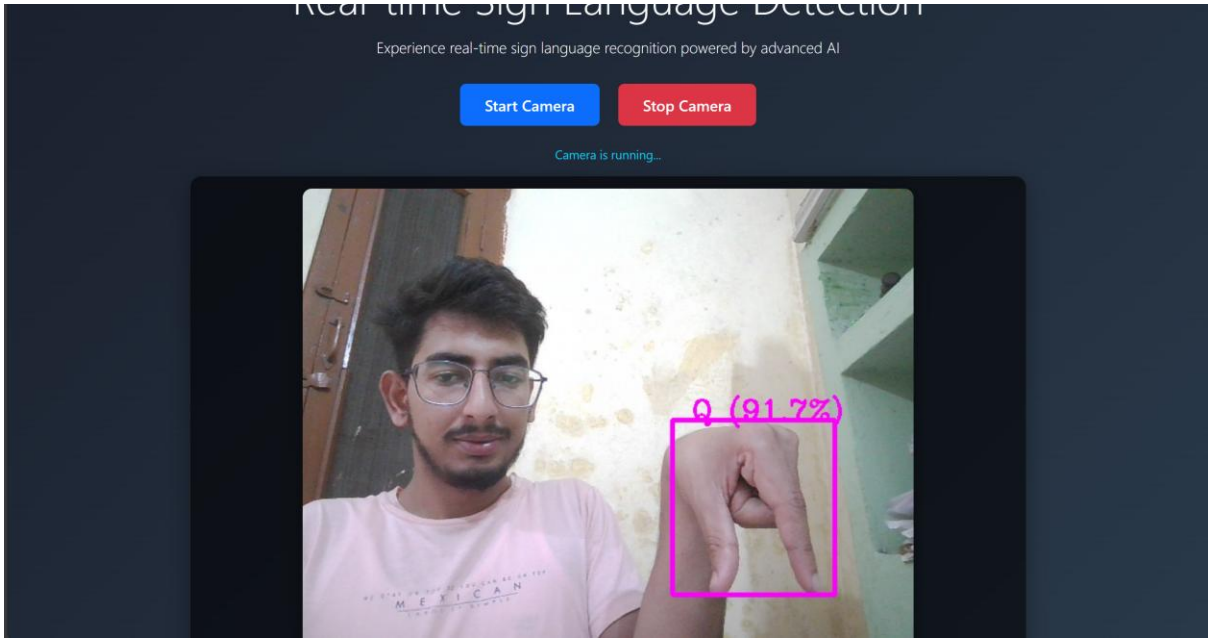
Project Demo Photos




R (99.9%)
How to Use
1. Click the "Start Camera" button to activate your webcam
2. Position your hand in front of the camera


Camera is running...
D (100.0%)
How to Use
1. Click the "Start Camera" button to activate your webcam
2. Position your hand in front of the camera


M (99.2%)
How to Use
1. Click the "Start Camera" button to activate your webcam
2. Position your hand in front of the camera


Experience real-time sign language recognition powered by advanced AI
Start Camera
Stop Camera
Camera is running...
Q (91.7%)