

**Development of AI/ML-Based Solution to Translate Sign Language Gestures into spoken or written text**

A project submitted in partial fulfillment of the requirement for award of the degree of

B.Sc. (Hons) Data Analytics and Artificial Intelligence

(2023-26)

|  |  |
| --- | --- |
| Submitted to:  Dr. Amita Sharma  Dr. Astha Pareek  Assistant Professor-Selection Grade  Department of CS&IT | Submitted by:  Ms. Neha Sharma  IISU/2023/ADM/35484  Roll no.:232021 |

**ACKNOWLEDGEMENT**

I would like to express my deepest appreciation to all individuals and organizations who have contributed to the successful completion of our project.

First and foremost, we extend our heartfelt thanks to our project supervisor**, Dr. Amita Sharma** and **Dr. Astha Pareek** (Assistant Professor (SG), for her unwavering support and mentorship.

Special gratitude to **Dr. Anubha Jain** (Director), **Dr. Ruchi Nanda** (Head of the Department), **Dr. Navneet Sharma** (Additional Head of Department), all faculty members, and lab assistants whose contribution to stimulating suggestions and encouragement, especially in writing this report.

Additionally, I appreciate the support and resources provided by our educational institution, IIS (deemed to be) University, throughout the duration of this project.

Furthermore, I would also like to express our gratitude to our friends and family for their unwavering support, understanding, and encouragement throughout this endeavour..

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| S.NO | Topic | Page no. |
|  | Abstract | 4 |
| 1. | Introduction  1.1 Problem statement  1.2 Significance of Problem  1.3 Task Assigned  1.4 Data Description | 5-6 |
| 2. | Tools and technologies | 7-8 |
| 3. | Task Description  3.1 Explanation of the Task  3.2 Frontend  3.3 Backend  3.4 Case study  3.5 Figures and screen-shorts  3.6 learning Outcomes | 9-12 |
| 4. | Conclusion | 13 |
| 5. | Reference | 14 |
| 6. | Annexure | 15-21 |

**ABSTRACT**

We’ve moved to virtual learning, which has created new opportunities but also highlighted critical barriers for students with hearing impairments. These students generally depend on sign language as their primary mode of communication, which can be limiting in virtual classrooms. However, there currently exists a crucial gap that is this project addresses through the creation of a real-time tool that transforms sign language gestures into spoken or written text to promote inclusivity and allow for true interaction in an online educational environment.

To achieve high accuracy and performance, it is important to integrate cutting-edge technologies such as image processing, computer vision, and natural language processing (NLP), all of which are implemented in the system that we are going to describe next. The project, developed with Python, leverages Mediapipe for sign language gesture detection and recognition, complemented by TensorFlow or PyTorch frameworks for training and deploying a resilient model. OpenCV processes real-time videos and captures hand & body moments accordingly.Particular focus is kept on improving metrics like real-time response, accuracy, and adapting to separate sign language dialects, providing tool practicality and reliability. The system’s performance will be critically evaluated with sign language experts, targeting recognition accuracy, speed, and satisfaction.

**1.Introduction**

**1.1 Problem Statement**

The communication gap between the hearing/speaking and the rest of the community is still very significant. Previous work Known sign langauges require additional skills by all participants in the conversation, which greatly constrains effective communication. This problem is solved by creating a real-time sign language translator which uses a simple webcam for capturing of hand gestures and create the corresponding output by translating it to sign language in the text and speech domain providing a transparent communication.

**1.2 Significance of the Problem**

According to the World Health Organization, more than 5% of the world’s population – approximately 400 million people in total – requires rehabilitation to address their severe hearing loss. In India, for example, many people need sign language to communicate with others, especially regarding health care and social services. With the aim of using technology to interpret sign language live, this project hopes to bridge the gap between sound and speech, with a view towards promoting accessibility and inclusivity in education, health care and public services.

**1.3 Task Assigned**

**1.3.1 Data Collection and Labeling**:Labeled dataset consisted of greyscale image containing gestures representing the characters "A-D" and the digits "0-3" Each gesture is stored in an independent folder with consistent naming conventionsThe dataset was examined for quality and invalid or unclear images were removed to ensure accuracy and integrity during training**.**

**1.3.2 Image Preprocessing and Normalization:**As one further step to avoid unidimensionalization of the classification data, we resized the images collected to 6464 pixels, converted the images to grayscale (in order to save computational costs) and normalized the image values between 0 and 1. The images were also encoded in numerical mode and then one-hot encoded for training.

**1.3.3 CNN Model Development and Training:**A convolutional neural network (CNN) was constructed using TensorFlow/Keras in which a number of convolutional and pooling layers were followed by dense layers whose optimization result was classified as eight gesture classes. A CNN model was trained with the Adam optimizer (with categorical cross-entropy loss) and tested with validation data.

**1.3.4 Real-Time Gesture Prediction Using Webcam**:The proposed live video capture module uses OpenCV wherein a region of interest (ROI) is defined and the user can choose an orientation of their hand to be recognized. The captured frames are preprocessed in real time and uploaded to the trained CNN model wherein the prediction result was displayed on screen as interactive feedback.

**1.3.5 Voice Feedback via Text-to-Speech Engine**:Based on user feedback and accessibility needs a text to speech (TTS) engine has been built, on which if you press a key (which it predicts) the expected character will be spoken out loud by the pyttsx3 library. This helps with accessibility as well as making interaction easier for people with additional accessibility needs.

**1.4 Data Description**

The dataset used in this project is an unannotated and preprocessed collection of American Sign Language (ASL) gesture images that contains eight distinct gesture classes of the alphabets A, B, C, D and digits (0, 1, 2, 3). Each gesture image is encoded in static hand signs in grayscale image format. It was organized into folder-wise format, where each folder name represents the label of the gesture and there are multiple sample images for the class.

All images were resized to 64x64 pixels to keep things uniform and make processing easier. We converted the images to black and white, which simplified the images and sped up the training of the model. In this task, color didn't matter for recognizing gestures.We then adjusted the pixel values to be between 0 and 1 to prepare for machine learning. The dataset was split into two parts: 80% for training the model and 20% for testing its performance. The image samples showed different hand positions and lighting to mimic real-world conditions. This helps the model predict gestures accurately even with new, unseen data.

**1.4.1 Features of the Dataset:**

* The dataset contains grayscale images showing hand signs from American Sign Language (ASL) for just a few characters: the letters A, B, C, D, and the numbers 0, 1, 2, and 3.
* Each type of sign has its own folder where you can find many pictures showing the same hand gesture. These pictures are taken against simple, plain backgrounds, without any distractions.
* All the pictures are made to be the same size, at 64 by 64 pixels. This makes sure that every picture follows the same standard size, no matter which sign it shows.
* The images are in grayscale, meaning they're in black and white with only one color channel. This choice helps make the process less complicated and faster for computers to handle and learn from the data.

**1.4.2 Challenges with the Dataset**

* **Class imbalance**: The number of images across gesture classes was not uniform, with some classes (like '0') having more samples than others (like 'C').
* **Lighting variation**: Images were captured under different lighting conditions, introducing variability in contrast and clarity.
* **Background noise**: Some samples had inconsistent backgrounds which could lead to misclassification.
* **Gesture similarity**: Certain gestures (e.g., ‘3’ and ‘B’) appear visually similar, increasing the model’s confusion during classification.

**2. Tools and technologies**

The Sign Language Translator system brings together several tools and technologies to recognize and translate gestures in real-time.

**2.1 Integrated Development Environment (IDE)**

**PyCharm:**We use PyCharm by JetBrains for this project. It's where we write and test our code. PyCharm is great for Python because it offers smart code suggestions, helps with debugging, and connects easily with systems that manage code versions.

The IDE works smoothly with libraries like TensorFlow, OpenCV, and pyttsx3, making our work more productive and streamlined.

**2.2 Programming Language**

**Python**:

Python is our main programming language because it's easy to use and has lots of helpful libraries.

This makes developing machine learning models, handling images, and creating speech functions faster and simpler.

**2.3 Libraries and Frameworks**

**TensorFlow:**

TensorFlow is an open-source library used to create and train the Convolutional Neural Network (CNN) model needed to recognize sign language.

Its flexibility and scalability help in managing complex neural networks.

**Keras:**

Keras simplifies the building and training of deep learning models. It's a high-level API that works with TensorFlow. It has easy-to-use interfaces and pre-built components, speeding up model creation.

**OpenCV (Open Source Computer Vision Library):**

OpenCV handles real-time computer vision tasks effectively.

In this project, it is used for:

- Capturing video from the webcam.

- Processing images by resizing, converting to grayscale, and normalizing.

- Identifying and extracting the Region of Interest (ROI) for gesture detection.

**NumPy:**

NumPy is crucial for scientific computing with Python.

We use it to manage numerical calculations and array manipulations during data preparation and when getting inputs ready for the model.

**pyttsx3:**

This Python library turns text into speech.

It allows the app to convert recognized text from gestures into spoken words, helping users understand better.

**2.4 Hardware Requirements**

**Webcam:**

A typical webcam is used to capture live video of hand gestures.

The video footage is processed to detect and interpret sign language gestures effectively.

**2.5 Operating System**

**Windows 10:**

The project is built and tested on Windows 10, ensuring a stable environment to combine various tools and libraries.

These tools and technologies come together to ensure the Sign Language Translator works efficiently, providing real-time gesture recognition and translation to help bridge communication gaps.

**3. Task Description**

**3.1 Task Explanation:**

The main aim of this project is to develop a real-time Sign Language Translator. This tool will recognize hand gestures and convert them into text and speech. This feature aims to assist people who use sign language, allowing them to communicate more effectively with those who don't know it.

The system works by using a webcam to capture hand movements. It processes these images to determine what each gesture means. Once a gesture is recognized, it displays the text on the screen and uses text-to-speech technology to say it out loud. The process includes several steps: capturing images, preparing them for recognition, identifying the gestures with a trained model, and generating the output.

**3.2 Frontend**

The user interface is designed to be simple and easy to navigate, suitable for people with different levels of technical understanding. Key features include:

* **-Webcam Feed Display\*\*:** Displays live video from the webcam, allowing users to see instant feedback.
* **Region of Interest (ROI) Highlighting:** Guides users to position their hands correctly for optimal gesture recognition.
* **Prediction Display:** Shows the identified gesture as text on the screen, allowing users to verify it.
* **Speech Output Option:** Users can hear the identified gesture spoken aloud, enhancing the communication process.

The interface is built using OpenCV's tools, ensuring smooth coordination with the backend.

**3.3 Backend:**

The backend processes the video input, recognizes gestures, and generates outputs. It consists of:

**Data Preprocessing:**

* **Grayscale Conversion:** Turns images into black and white to simplify data and speed up processing.
* **Resizing:** Changes image size to a standard like 64x64 pixels to meet model requirements.
* **Normalization:** Adjusts pixel values to a range between 0 and 1 to aid model training and prediction.

**Model Training:**

A Convolutional Neural Network (CNN) is trained on a set of hand gesture images. It learns the distinctive features of each gesture for accurate recognition**.**

* **Real-Time Prediction:**
* The trained model analyzes the webcam images immediately and predicts the gesture.
* **Text-to-Speech Conversion:**
* Uses the pyttsx3 library to convert recognized gestures into speech, providing audio feedback.

**3.4 Case Study**

In testing, the system successfully recognized and translated hand gestures for the alphabet ('A' to 'D') and numbers ('0' to '3'). For example:

- Gesture for 'c': When users showed the 'c' gesture, the system correctly displayed "A" on the screen and articulated it.

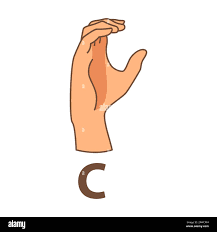
- \*Gesture for '2': Showing the '2' gesture resulted in the accurate display and spoken output.

These successful results confirm the system's potential effectiveness in real-world scenarios, supporting communication for sign language users.

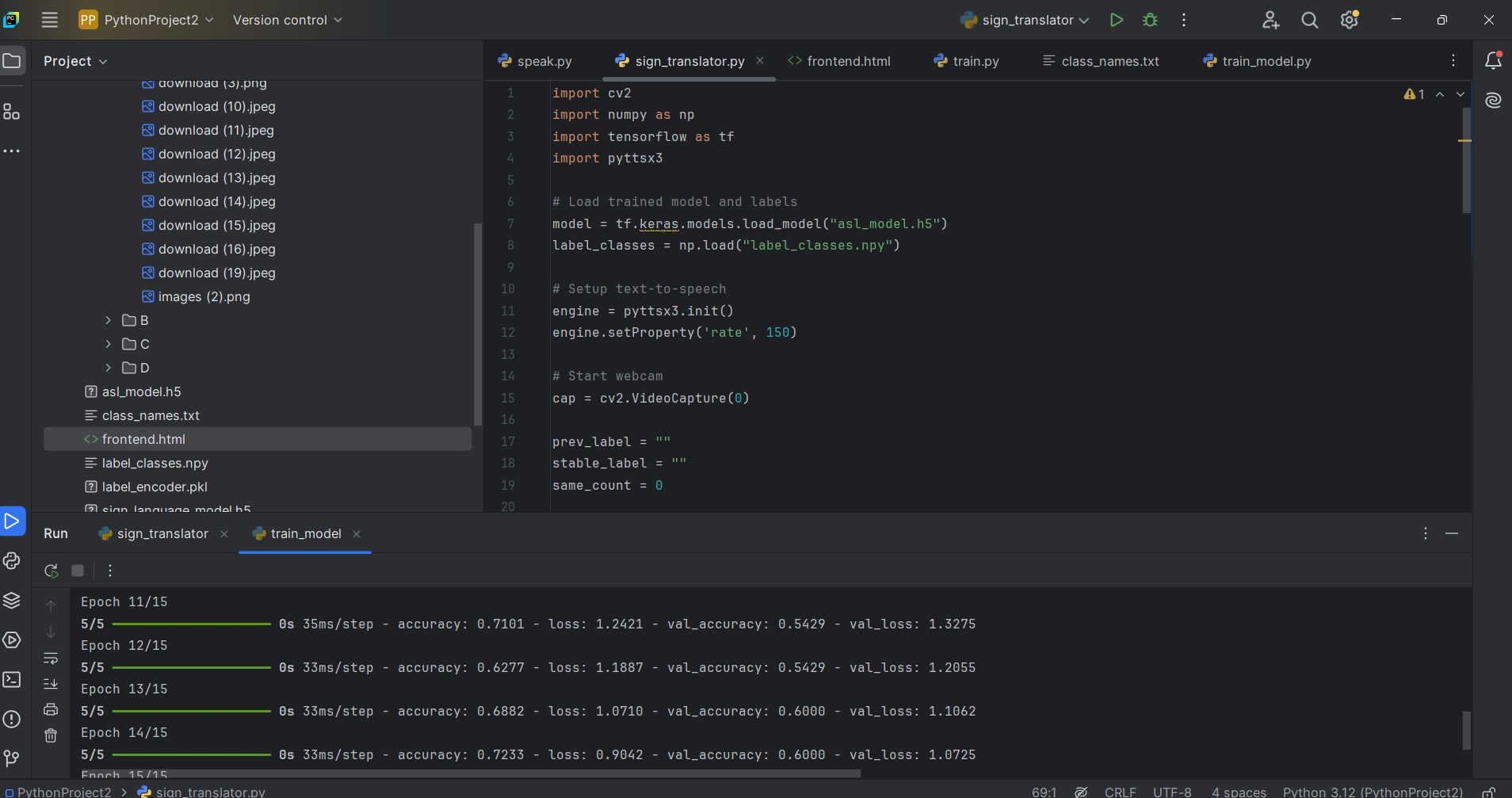
**3.5 Figures and Screenshots**

Inserting relevant images and diagrams here:

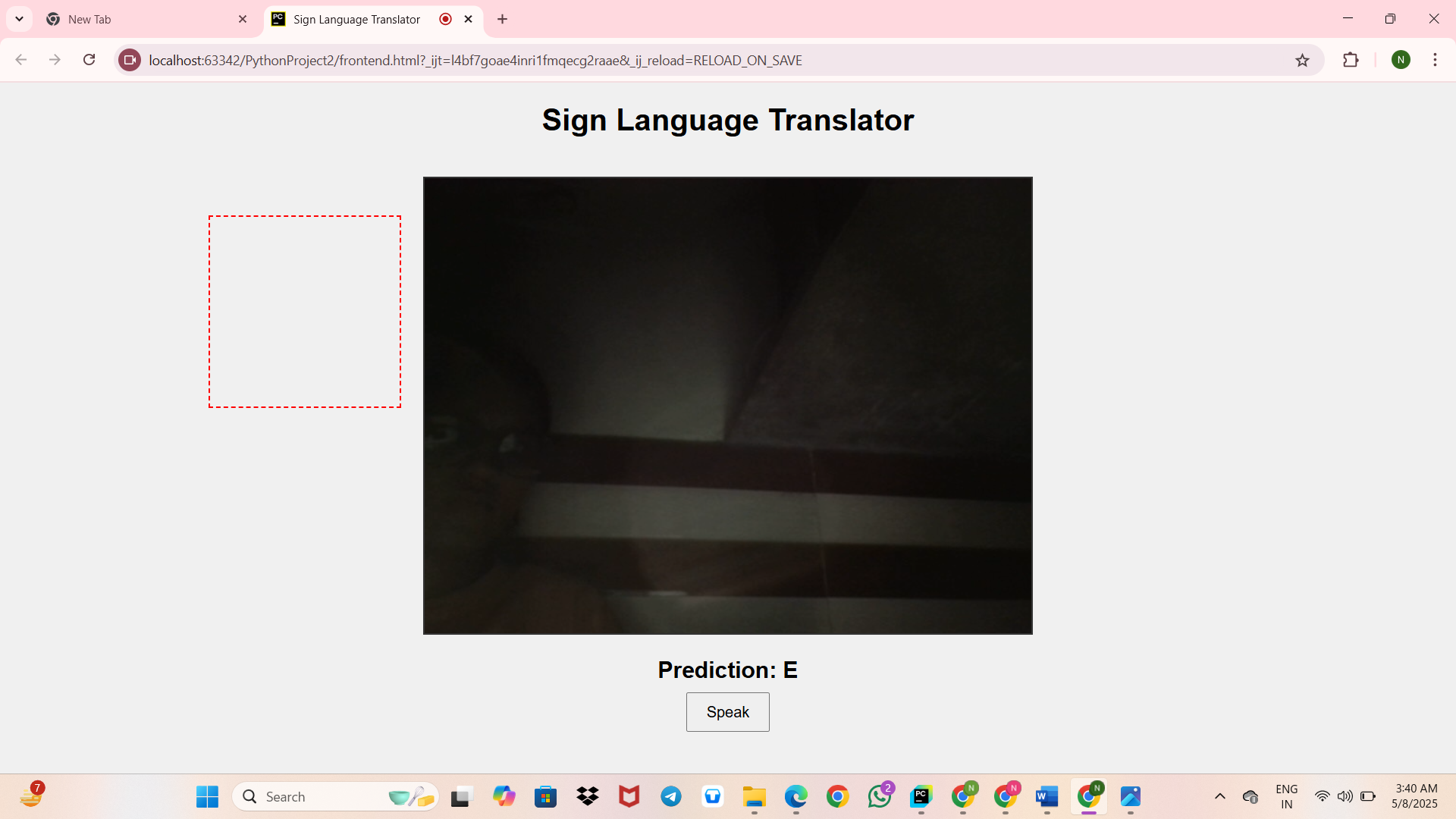
**- Sample Dataset Images:** Providing examples of hand gesture images used for model training.

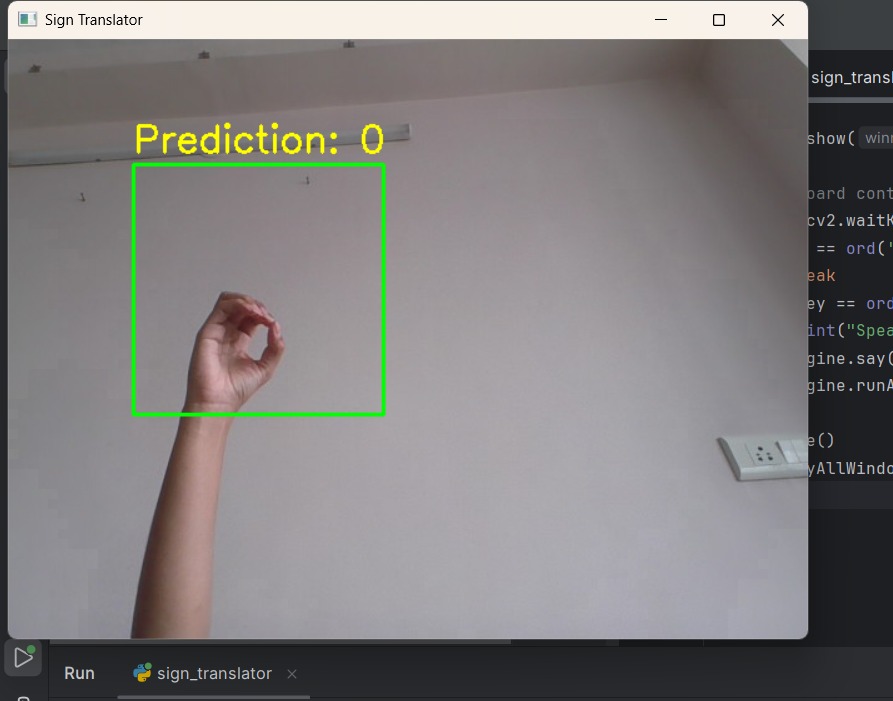
   

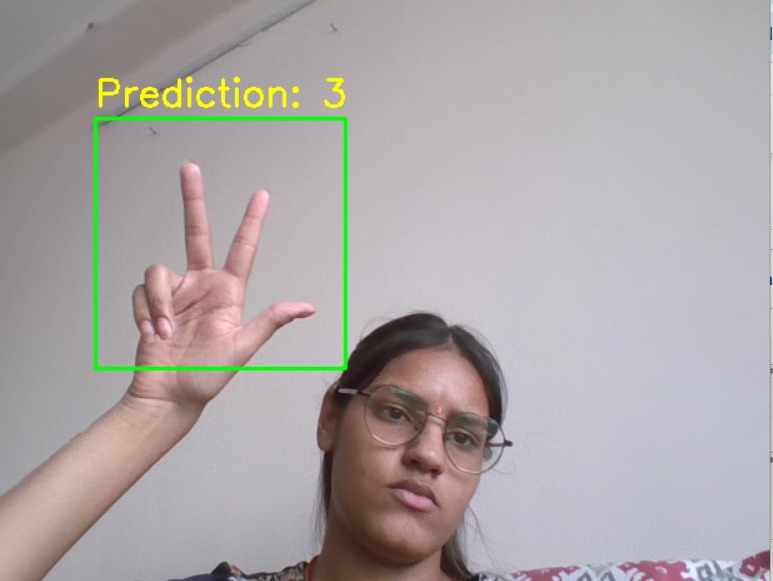
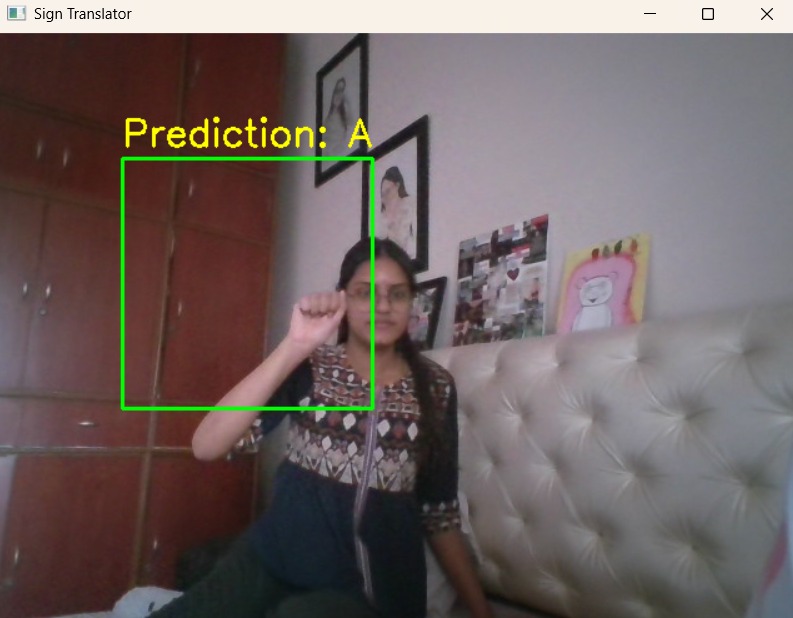
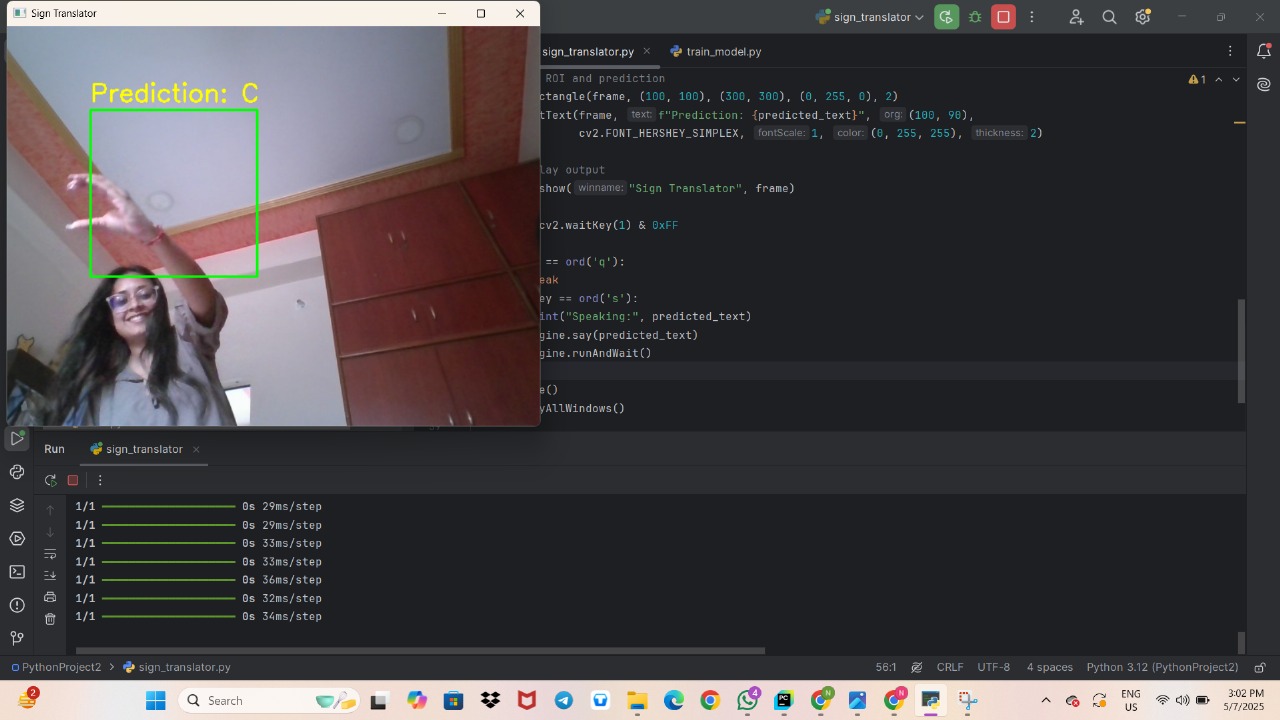
**- Model Architecture Diagram:** Show the structure of the CNN model, including its layers and connections.



**- User Interface Screenshots:** Capture the application's interface during live predictions, highlighting the webcam feed, ROI, and predictions**.**

****



**3.6 Learning Outcomes**

The development of this project provided several valuable lessons and skills:

**- Image Preprocessing Techniques:** Gained practical skills in preparing images for machine learning, including grayscale conversion, resizing, and normalization**.**

**- CNN Model Training:** Developed an understanding of designing and training Convolutional Neural Networks for image classification**.**

**- Real-Time Application Integration**: Learned how to incorporate machine learning models into applications that handle live data.

**- Text-to-Speech Implementation:** Learned to convert text data into speech using Python libraries, improving application accessibility.

**- Problem-Solving Skills**: Improved the ability to identify and solve challenges encountered during the development of the real-time gesture recognition system.

**4. Conclusion**

The Sign Language Translator is an essential tool designed to improve communication for people with hearing and speech impairments. It changes hand signs into text and spoken words quickly with the help of computer vision and machine learning.

To capture hand signs, the system uses a webcam. The images are processed by converting them to black and white, resizing, and getting them ready for analysis. A Convolutional Neural Network (CNN) model, trained to recognize different hand signs representing letters and numbers, does this analysis. The identified signs are converted into text that appears on the screen and is also read aloud using speech technology. This feature allows individuals using sign language to communicate smoothly with those who don't.

The system is user-friendly and can be operated by anyone, no matter their technical skills. It offers instant visual and sound feedback to ensure that users feel confident about the accuracy of the translations.

This project is a great demonstration of how assistive technologies can benefit from machine learning and computer vision by using inclusive designs that help close communication gaps. It lets people with hearing and speech challenges engage more in social, educational, and work environments.

There are plans to make the system even better. Ideas for improvement include adding more signs and phrases, making it adaptable to various sign languages, and developing a mobile version for easier access. By learning from user interactions, the system could further enhance its accuracy and response time.

In summary, the Sign Language Translator shows how technology can help make the world more inclusive for those with hearing and speech impairments by removing communication barriers.

**5. References:**

1. Scribd. (n.d.). \*Project File | Deep Learning | Sign Language\*. Retrieved from [https://www.scribd.com/document/739119772/project-file](https://www.scribd.com/document/739119772/project-file)

2. Scribd. (n.d.). \*Sign Language Translator Project Report | Artificial Neural Network\*. Retrieved from [https://www.scribd.com/document/743713081/Sign-Language-Translator-Project-Report](https://www.scribd.com/document/743713081/Sign-Language-Translator-Project-Report)

3. GitHub. (n.d.). \*sign-language-translator/sign-language-translator\*. Retrieved from [https://github.com/sign-language-translator/sign-language-translator](https://github.com/sign-language-translator/sign-language-translator)

4. Sathyabama Institute of Science and Technology. (n.d.). \*Hand Sign Language Interpreter\*. Retrieved from [https://sist.sathyabama.ac.in/sist\\_naac/documents/1.3.4/b.e-cse-batchno-75.pdf](https://sist.sathyabama.ac.in/sist\_naac/documents/1.3.4/b.e-cse-batchno-75.pdf)

**6. Annexure**

**Train.py**

# train\_model.py

import os

import cv2

import numpy as np

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def load\_data(folder):

images, labels = [], []

label\_map = {} # Map folder names to class indices

class\_names = sorted(os.listdir(folder))

for idx, label\_name in enumerate(class\_names):

label\_map[label\_name] = idx

label\_path = os.path.join(folder, label\_name)

for img\_name in os.listdir(label\_path):

img\_path = os.path.join(label\_path, img\_name)

img = cv2.imread(img\_path, cv2.IMREAD\_GRAYSCALE)

img = cv2.resize(img, (64, 64)) / 255.0

images.append(img)

labels.append(idx)

return (np.array(images).reshape(-1, 64, 64, 1),

to\_categorical(labels), class\_names)

def create\_model(num\_classes):

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(64, 64, 1)),

MaxPooling2D(2,2),

Conv2D(64, (3,3), activation='relu'),

MaxPooling2D(2,2),

Flatten(),

Dense(128, activation='relu'),

Dense(num\_classes, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

images, labels, class\_names = load\_data("dataset/asl\_data\_with\_images")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(images, labels, test\_size=0.2, random\_state=42)

model = create\_model(num\_classes=len(class\_names))

model.fit(X\_train, y\_train, epochs=15, validation\_data=(X\_test, y\_test))

model.save("asl\_model.h5")

np.save("label\_classes.npy", np.array(class\_names))

# Save class names

with open("class\_names.txt", "w") as f:

for name in class\_names:

f.write(name + "\n")

**sign\_translator.py**

import cv2

import numpy as np

import tensorflow as tf

import pyttsx3

# Load trained model and labels

model = tf.keras.models.load\_model("asl\_model.h5")

label\_classes = np.load("label\_classes.npy")

# Setup text-to-speech

engine = pyttsx3.init()

engine.setProperty('rate', 150)

# Start webcam

cap = cv2.VideoCapture(0)

prev\_label = ""

stable\_label = ""

same\_count = 0

print("Press 'q' to quit, 's' to speak the predicted label.")

while True:

ret, frame = cap.read()

if not ret:

break

# Flip and crop region of interest (ROI)

frame = cv2.flip(frame, 1)

roi = frame[100:300, 100:300]

gray = cv2.cvtColor(roi, cv2.COLOR\_BGR2GRAY)

resized = cv2.resize(gray, (64, 64))

normalized = resized / 255.0

input\_image = normalized.reshape(1, 64, 64, 1)

# Predict

prediction = model.predict(input\_image, verbose=0)

current\_label = label\_classes[np.argmax(prediction)]

# Stabilize predictions

if current\_label == prev\_label:

same\_count += 1

else:

same\_count = 0

prev\_label = current\_label

if same\_count >= 5:

stable\_label = current\_label

else:

stable\_label = ""

# Draw UI

cv2.rectangle(frame, (100, 100), (300, 300), (0, 255, 0), 2)

cv2.putText(frame, f"Detected: {stable\_label}", (100, 90),

cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255, 0, 255), 2)

cv2.imshow("Sign Language Translator", frame)

key = cv2.waitKey(1) & 0xFF

if key == ord('q'):

break

elif key == ord('s') and stable\_label != "":

print("Speaking:", stable\_label)

engine.say(stable\_label)

engine.runAndWait()

cap.release()

cv2.destroyAllWindows()

**frontend.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Sign Language Translator</title>

<style>

body {

font-family: Arial, sans-serif;

text-align: center;

background-color: #f0f0f0;

}

#video {

border: 2px solid #333;

margin-top: 20px;

}

#roi {

position: absolute;

border: 2px dashed red;

pointer-events: none;

}

#prediction {

margin-top: 20px;

font-size: 24px;

font-weight: bold;

}

#speakBtn {

margin-top: 10px;

padding: 10px 20px;

font-size: 16px;

}

</style>

</head>

<body>

<h1>Sign Language Translator</h1>

<video id="video" width="640" height="480" autoplay></video>

<div id="roi" style="width:200px; height:200px; top:140px; left:220px;"></div>

<div id="prediction">Prediction: <span id="predictedChar">None</span></div>

<button id="speakBtn">Speak</button>

<script>

const video = document.getElementById('video');

const predictedChar = document.getElementById('predictedChar');

const speakBtn = document.getElementById('speakBtn');

// Access the webcam

navigator.mediaDevices.getUserMedia({ video: true })

.then(stream => {

video.srcObject = stream;

})

.catch(err => {

console.error("Error accessing webcam: ", err);

});

// Placeholder for prediction logic

// In a real application, you'd capture the ROI, send it to your model, and get the prediction

// For demonstration, we'll simulate predictions every 2 seconds

const samplePredictions = ['A', 'B', 'C', 'D', 'E'];

setInterval(() => {

const randomIndex = Math.floor(Math.random() \* samplePredictions.length);

predictedChar.textContent = samplePredictions[randomIndex];

}, 2000);

// Text-to-Speech functionality

speakBtn.addEventListener('click', () => {

const utterance = new SpeechSynthesisUtterance(predictedChar.textContent);

speechSynthesis.speak(utterance);

});

</script>

</body>

</html>