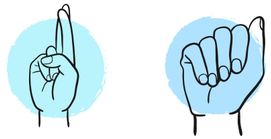
Formative Assessment I

BIT25PC01: Artificial Intelligence and Machine Learning



American Sign Language (ASL) Fingerspelling Recognition System

Arpita Rahul Patki[123B1F001] TY-IT-A, PCCOE college

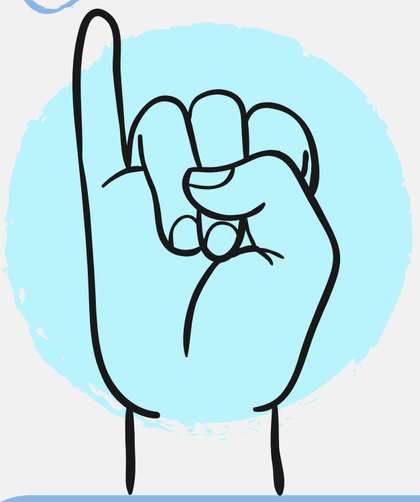
Varad Sushant Deshmukh[123B1F022] TY-IT-A, PCCOE college

American Sign Language(ASL)

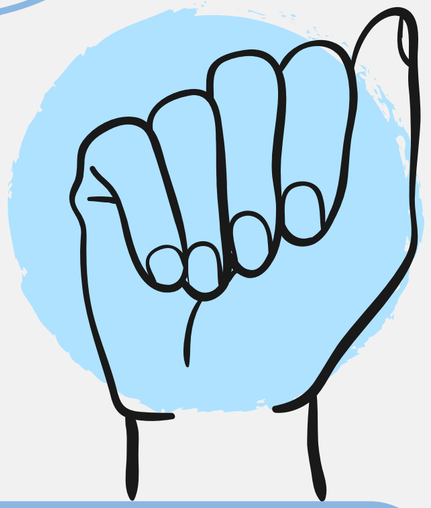
FingerSpelling Recognition System

Submitted by:
Arpita Rahul Patki
Varad Sushant Deshmukh

Guided by:
Dr.Gulbakshi Dharmale



Aim



To develop a real-time platform that captures American Sign Language (ASL) gestures via webcam, translates them into English text, and provides instant audio output—enabling seamless communication between deaf/hard-of-hearing individuals and those who do not understand sign language.

Objective

1. **Implement real-time ASL recognition:**

Build or integrate a reliable computer vision system to detect and classify ASL signs (starting with alphabets, then words) using a webcam

2. **Translate ASL to English text:**

Accurately convert recognized signs into English letters/words and display them clearly to the user.

3. **Provide English voice output:**

Integrate a text-to-speech module that reads out the translated text.

4. **Design a user-friendly interface:**

Ensure the platform is accessible, intuitive, and responsive for all users.

5. **Facilitate inclusive communication:**

Bridge the gap between sign language users and non-signers by providing immediate translation and audio feedback.

Motivation

More than 70 million deaf people around the world use sign languages to communicate. Sign language allows them to learn, work, access services, and be included in the communities.

It is hard to make everybody learn the use of sign language with the goal of ensuring that people with disabilities can enjoy their rights on an equal basis with others.

So, the aim is to develop a user-friendly human computer interface (HCI) where the computer understands the American sign language This Project will help the dumb and deaf people by making their life easy.

Proposed Solution & Architecture Diagram:

The project aims to develop a real-time system that recognizes ASL hand signs representing whole words and immediately converts them into spoken English. The system uses a webcam to capture hand gestures, processes the input using computer vision and deep learning, maps the recognized gesture to its corresponding English word, and employs a text-to-speech engine to generate audio output.

Key Components:

Camera Module: Captures real-time video frames of the user's hand.
Hand Detection & Landmark Extraction: Uses MediaPipe and OpenCV for stable tracking and feature extraction.
Gesture Recognition Model: CNN(Convolutional Neural Network)/LSTM(long short-term memory)-based deep learning model (built with Keras/TensorFlow) that identifies the ASL word directly from the extracted features.
Word Mapping: Dictionary mapping the recognized gesture/model output to its English word.
Text-to-Speech (TTS): Converts the final word into audible speech for the listener.

Project Requirements:

Hardware Requirement:

Webcam: A webcam is required to capture real-time video of hand gestures

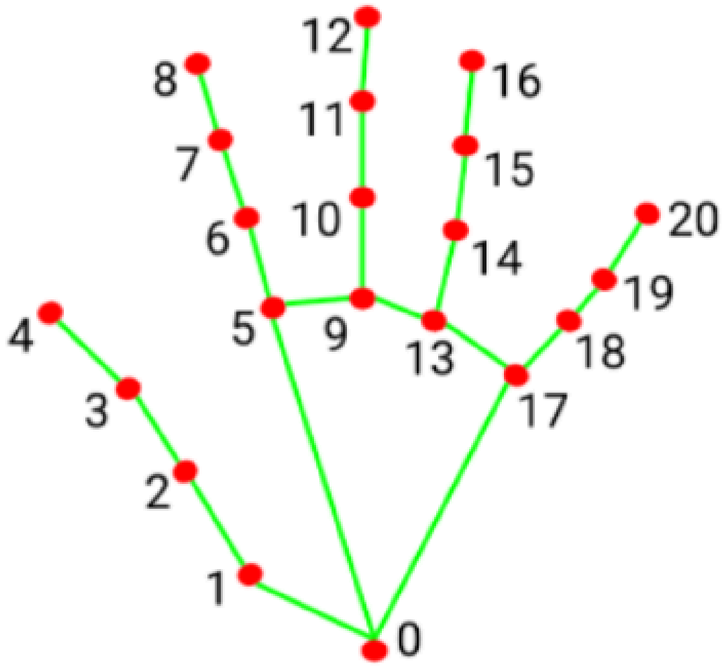
Software Requirement:

Operating System: Windows 8 and Above
IDE: PyCharm
Programming Language: Python 3.9 5

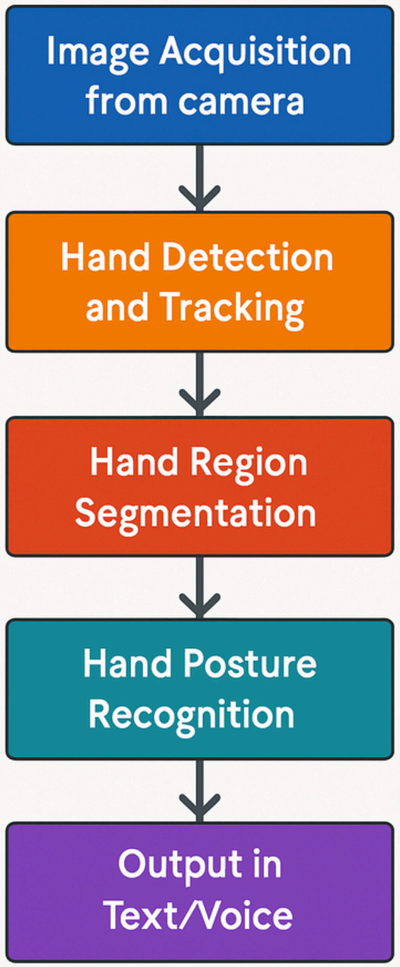
Python Libraries :

- **OpenCV (cv2):** Used for capturing video from the camera, preprocessing frames, drawing bounding boxes or hand landmarks, and general image processing tasks.
- **NumPy (np):** Handles numerical operations such as array manipulations, image array conversion, and mathematical computations required by models and image processing.
- **Mediapipe (mp):** Detects and tracks hand landmarks in real time, making it easier to extract features crucial for recognizing ASL signs.
- **TensorFlow (tf):** Provides the deep learning backend to build, train, and run neural networks for gesture classification.
- **Keras:** High-level API (usually from tensorflow import keras) for designing, compiling, fitting, and evaluating deep learning models in an accessible way.

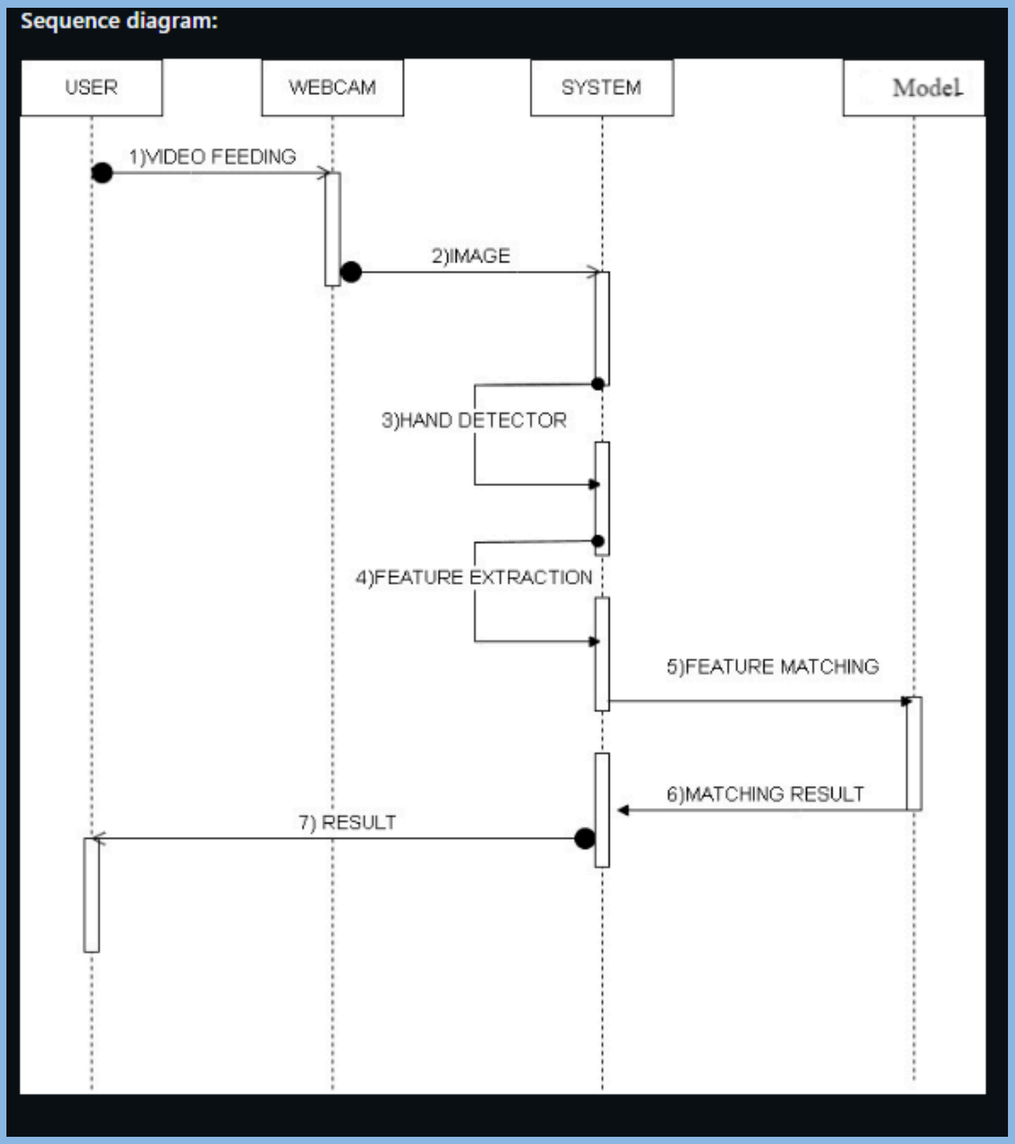
Mediapipe Landmark System:



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP



Sequence diagram:



Methodology

Data Collection Plan

Creating Our Own Dataset:

We will record hand gestures for the ASL alphabet with diverse volunteers, capturing multiple repetitions for each letter.

Using Existing Datasets:

We will start by using publicly available ASL datasets to build a strong initial model foundation.

Devices:

We will use webcams and smartphone cameras for easy and accessible recording.

Recording Conditions:

Our recordings will happen in well-lit, simple backgrounds initially, then extend to varied environments for real-world robustness.

Participant Diversity:

We will include people of different ages, skin tones, hand sizes, and signing styles to ensure inclusivity.

Data Labeling & Organization:

Each sample will be labeled by letter and signer, organized in folders for efficient access.

Data Enhancement:

We will extract hand landmarks using MediaPipe to create clear, noise-free “skeleton” images and apply augmentation techniques like flipping and brightness adjustments.

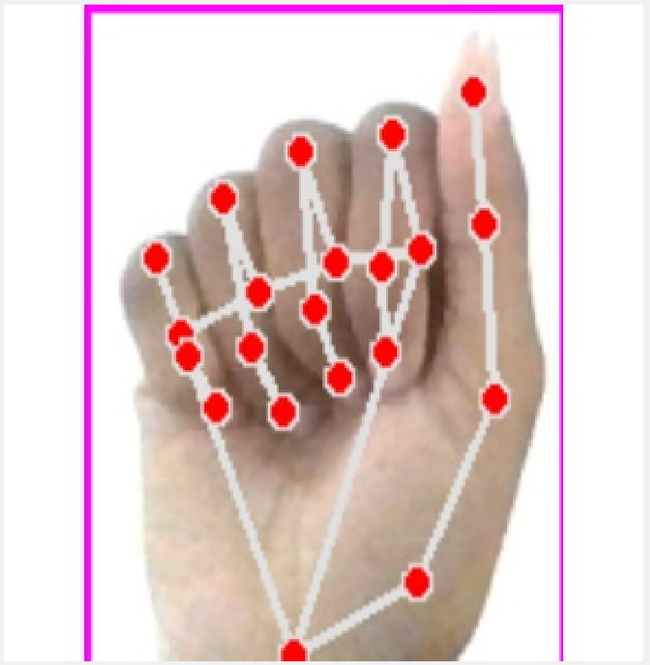
Quality Control & Ethics:

We will review all data for clarity and accuracy and obtain informed consent from all participants.
This hybrid approach of combining existing and custom data will make our recognition system accurate, reliable, and adaptable.

We have 3 Dataset

source: own dataset
unprocessed dataset
(neither normalised nor in
sekeleton.

source: kaggle
Normalised image
(i.e. 200px*200px).



source : kaggle
Normalised and and skeleton

Preprocessing:

Hand Detection & Segmentation:

We will detect the hand region in each video frame or image using MediaPipe. This ensures the model focuses only on the relevant hand gestures, ignoring background distractions.

Landmark Extraction:

MediaPipe will help us extract key points (landmarks) on the fingers and palm, creating a simplified “skeleton” representation that is robust to variations in lighting and background.

Skeleton Image Generation:

We will plot these landmarks on a plain white background, which removes noise from complex scenes and improves recognition accuracy.

Image Normalization and Resizing:

All images or skeleton representations will be resized to a uniform size (e.g., 64×64 pixels) and their pixel values normalized to help the model learn efficiently.

Preprocessing:

Data Augmentation:

To increase model robustness, we will augment the data by flipping, rotating, and adjusting brightness of images, simulating real-world variability.

Label Encoding:

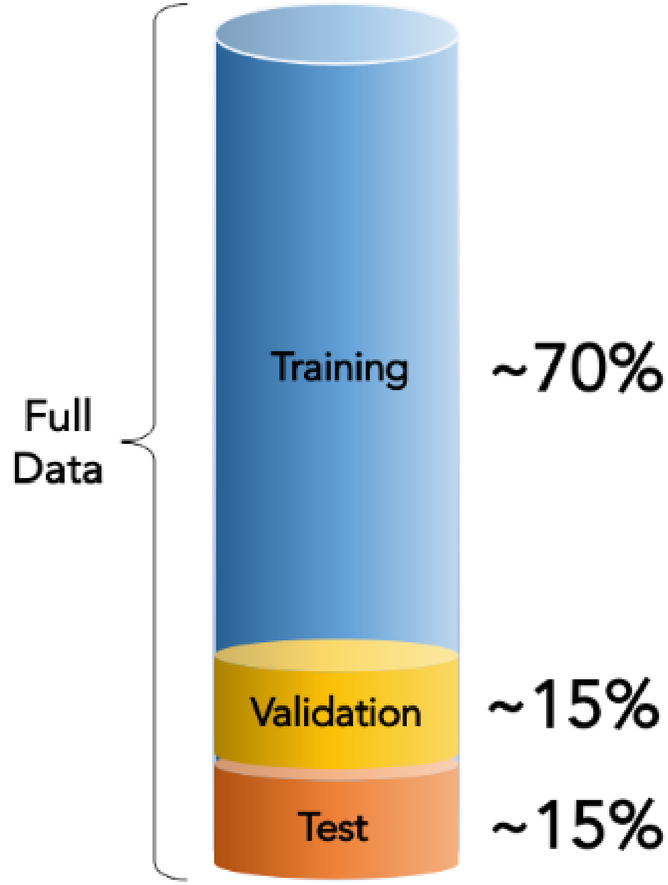
Each preprocessed image will be labeled with its corresponding letter or class to train the model effectively.

Data Splitting into Training &Testing:

Once preprocessing is complete, we will split our dataset into training and testing sets—typically allocating around 70-80% for training and 20-30% for testing. This split will be made carefully to ensure balanced representation of all letters, so the model can learn well and be fairly evaluated on unseen data.

Quality Control:

Throughout preprocessing, we will review samples to remove any blurry, mislabeled, or incomplete images to maintain high data quality.



MODEL TRAINING

1

We will use the preprocessed “skeleton” images of hand gestures as input for our model.

2

- Our model will be a Convolutional Neural Network (CNN), which is well-suited for recognizing patterns in images.
- During training, the CNN will learn to identify which hand skeleton corresponds to which letter.

3

- We’ll organize our training so the model sees thousands of labeled examples, allowing it to spot subtle differences between similar gestures.
- To help the model generalize, we’ll use images from various people, backgrounds, and lighting conditions.

4

Throughout training, we’ll monitor accuracy and loss to make sure the model is really learning—not just memorizing the data.

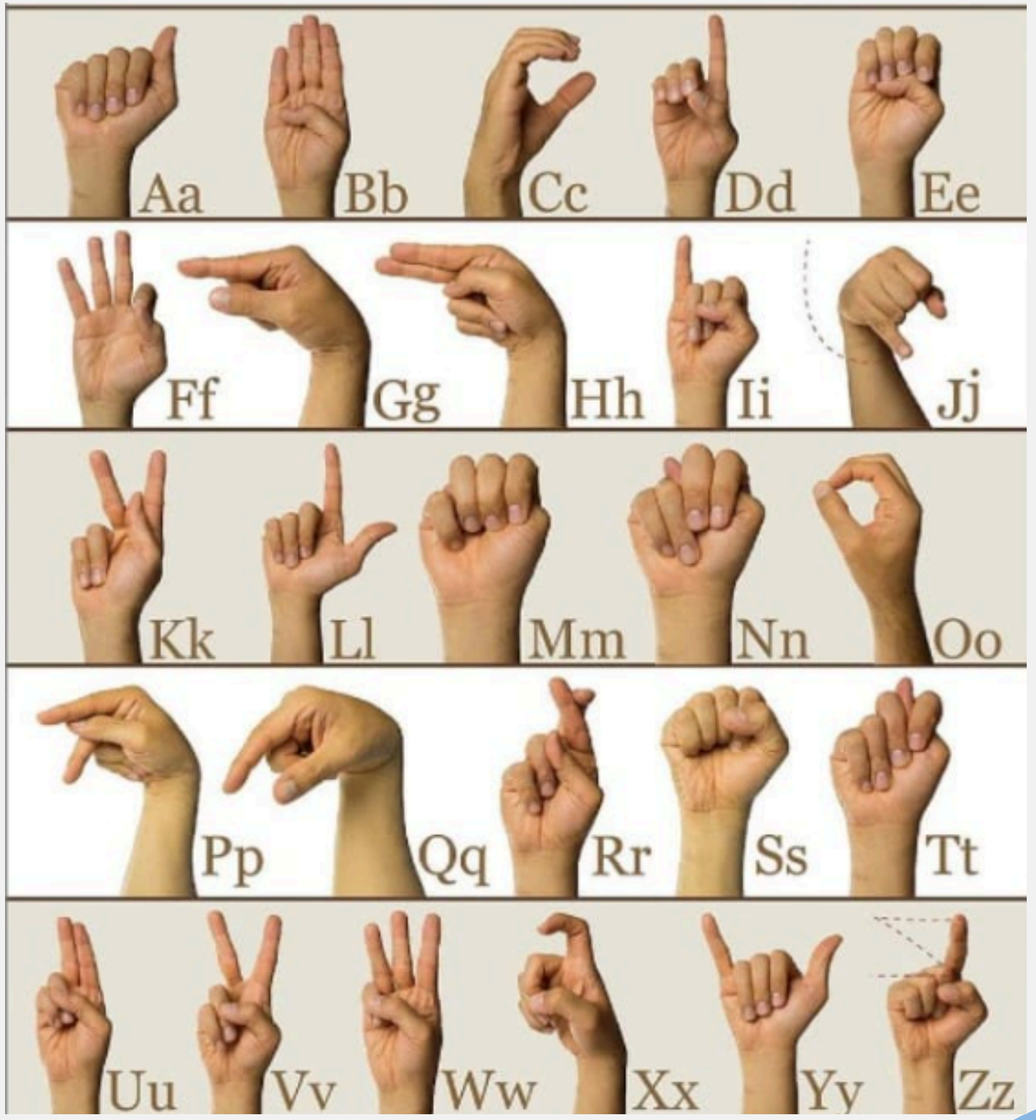
5

- After training, we’ll test the model with new, unseen skeleton images to confirm it can reliably predict the correct letter every time.
- Once a word is formed from predicted letters, we’ll use a text-to-speech library so our program can say the word aloud.

6

- For speech conversion, we use an existing text-to-speech (TTS) engine to convert the recognized word into audio without additional model training.
- pyttsx3 is a popular Python library that allows us to easily convert recognized words into natural-sounding speech.
  It works offline, is easy to integrate with our Python-based workflow, and supports different voices and speech rates.

Example of Grouping Similar Letters :



01.

- Some ASL letters look very similar and are hard to distinguish directly by the model—like ‘A’, ‘E’, ‘M’, ‘N’, ‘S’, and ‘T’.
- Instead of training the model to classify all 26 letters at once, we first group these similar letters into a single broad class (e.g., Group 1: [A, E, M, N, S, T]).

02.

- The model initially identifies the hand gesture as belonging to this group, reducing confusion between close-looking signs.
- Then, using detailed mathematical features from hand landmarks—like finger angles and distances—we further analyze the gesture within that group to predict the exact letter (e.g., distinguishing ‘A’ from ‘E’).

03.

- This two-step approach helps improve accuracy by simplifying the initial classification and refining predictions with subtle hand shape details.

What Comes After Model Training

01

Evaluate the Model:

We will test the trained CNN on the separate test dataset to measure its accuracy and performance on unseen hand gestures. This helps us check how well the model generalizes beyond training data.

02

Fine-tune and Optimize:

If needed, we will tweak hyperparameters (like learning rate, batch size) or add more data/augmentation to improve accuracy and reduce errors.

03

Integrate with Text-to-Speech:

Once the model reliably predicts letters, we'll combine the predicted letter sequence into words and use a text-to-speech engine to vocalize the recognized word in real time.

04

Real-time Testing:

We will deploy the model in a real-time system, taking live video input from a webcam, performing hand detection, gesture recognition, and immediately outputting te

Conclusion

Our system turns sign language letters into words and speech, making communication easier for the deaf community. By using smart hand detection and deep learning, we ensure accurate and reliable recognition. Combining existing and new data helps our model work well for everyone. Text-to-speech adds a natural voice, bridging gaps between signers and non-signers. In the future, we plan to add air-written word recognition to make the system even more user-friendly and versatile.

Thank You.