Name	College	Course	Mail ID
Naina Gupta	UPES, Dehradun	B.Tech. CSE	guptanaina021@gmail.com

Table of Contents

I. INTRODUCTION

Communication can be extremely difficult for those who are deaf or hard of hearing. Sign language serves as a critical means of interaction for individuals with hearing and speech impairments, allowing them to express themselves and engage with others. However, sign language remains less familiar to a larger portion of the population who primarily use spoken and written language for communication. Thus, there is a pressing need to develop technological solutions for interpreting sign language.

In this project, instead of relying on pre-defined dataset, I have created a dataset for training and testing, ensuring flexibility and adaptability to different regional variations within ISL.

Leveraging machine-learning techniques, the project focuses on identifying hand gestures from Indian Sign Language. The process involves capturing and categorizing two-dimensional and three-dimensional images of ISL gestures. Landmark detection in the images is facilitated through the MediaPipe framework.

The newly created database comprises a diverse set of gestures, including representations of vowels and consonants commonly used in Indian Sign Language. These gestures span various alphabets, such as A, B, C, D, E up to Z, allowing for comprehensive training and testing of the recognition model.

Key innovations:

- Focus on Indian Sign Language: Addressing the need for tools dedicated to regional sign languages.
- Landmarks Detection: Employing MediaPipe framework for accurate hand pose recognition.
- Text-to-Speech Integration: Translating recognized gestures into spoken language for seamless communication.

II. Problem Identification

Millions in India rely on Indian Sign Language (ISL) to communicate due to hearing or speech impairments. However, a significant communication barrier exists, as most people do not understand ISL. This creates challenges in daily life, hindering interaction and social inclusion.

Key issues:

- Limited understanding: The vast majority of the population does not understand ISL, creating a communication barrier with deaf and hard-of-hearing individuals.
- Restricted social interaction: This barrier limits opportunities for meaningful communication, social engagement, and participation in various aspects of life.
- Lack of real-time solutions: Existing solutions often focus on isolated signs or lack real-time translation, hindering natural conversation flow.

Impact:

- Social isolation and loneliness
- Limited access to information and education
- Difficulty engaging in daily activities and building relationships Barriers to employment and social participation

III. Proposed System Design

Develop a real-time system that recognizes Indian Sign Language (ISL) gestures and translates them into spoken words using text-to-speech technology, bridging the communication gap between deaf and hard-of-hearing individuals and the wider community.

System Components:

1. Data Acquisition:
    a. Collect a diverse dataset of hand images and videos representing various static and dynamic ISL signs.
    b. Include signs from different regional dialects to ensure wider applicability.
    c. Label data accurately with corresponding spoken words.
2. Pre-processing:
    a. Apply image and video processing techniques to normalize data, extract relevant features (hand shape, orientation, movement), and reduce noise.
    b. Segment individual signs from continuous signing sequences.
3. Sign Recognition:
    a. Train a deep learning model on the pre-processed data to recognize individual ISL signs.
4. Text-to-Speech Conversion:
    a. Integrate a high-quality text-to-speech engine that produces natural-sounding voices.
    b. Consider language variations and dialects for accurate pronunciation.
5. Real-time Translation:
    a. Design a user interface that captures hand gestures (camera, depth sensor) in real-time.
    b. Feed captured data into the trained model for sign recognition.
    c. Translate recognized signs into spoken words using the text-to-speech engine, providing near-instantaneous communication.

IV. Methodology

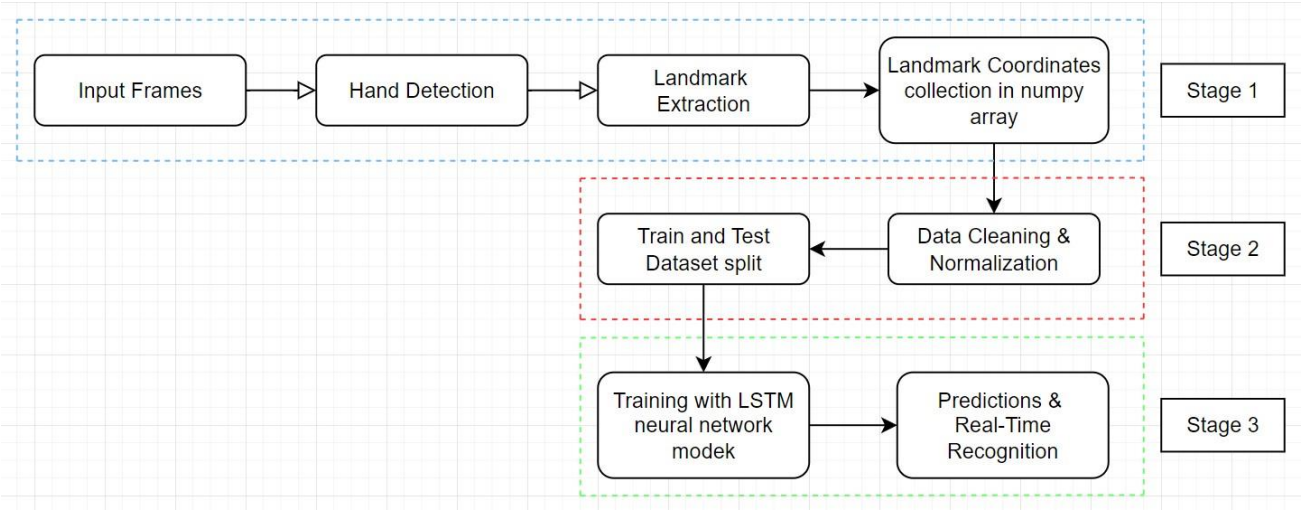

Fig. 1. Architecture Diagram of the System

System Architecture (Fig. 1)

The system can be divided into three main modules:

1. Data Acquisition: This module captures video frames from a webcam or loads image sequences for gesture recognition and performs hand detection and landmark extraction using MediaPipe's Holistic model (Fig. 2). The extracted key points (x, y, z coordinates) from landmarks are stored in a sequence buffer.
2. Pre-processing and Training: This module pre-processes data by creating sequences of frames and converting them into numerical representations. Then, it builds and trains an LSTM model to classify the actions within the sequences.
3. Gesture Recognition: This module utilizes a pre-trained LSTM model (Fig. 4) to analyse the sequence of hand key points. The model predicts the most likely gesture from the defined set of alphabets (Fig. 3).

Data Acquisition

1. Capture Video Frames: A loop continuously captures frames from the webcam using OpenCV's VideoCapture object.
2. Pre-process Frame: Frames are converted from BGR (OpenCV format) to RGB (MediaPipe format) for processing.
3. Hand Detection with MediaPipe: The MediaPipe Hands model processes each frame to detect hands and extract landmark coordinates (x, y, and z) for each key point.
4. Extract Key points: If hands are detected, the system iterates through each hand and stores the corresponding landmark data for further processing.
5. Data Storage: To create a training dataset, the extracted landmark data are saved along with corresponding gesture labels. Saving is done in NumPy arrays format to use in model training.

Fig. 2. Hand Landmarks representation used in MediaPipe

Pre-Processing and Training

Data Preprocessing:

1. Define data path and actions: Set data_path to the action data directory and create a list actions with characters representing the actions (e.g., 'A' to 'Z').
2. Label Mapping: Create a dictionary label_map to assign unique numerical labels (starting from 0) to each action in actions.
3. Sequence Creation:
    1. Define the number of sequences to be generated per action (sequences_no).
    2. Define the sequence length (number of frames per sequence) (sequences_length).
    3. Iterate through each action in actions:
        a. For each sequence in sequences_no:
            i. Create an empty list window to store frames for the current sequence.
            ii. Iterate through each frame in the sequence length (sequences_length):
                1. Construct the path to the data file for the current frame using os.path.join.
                2. Load the data from the file using np.load.

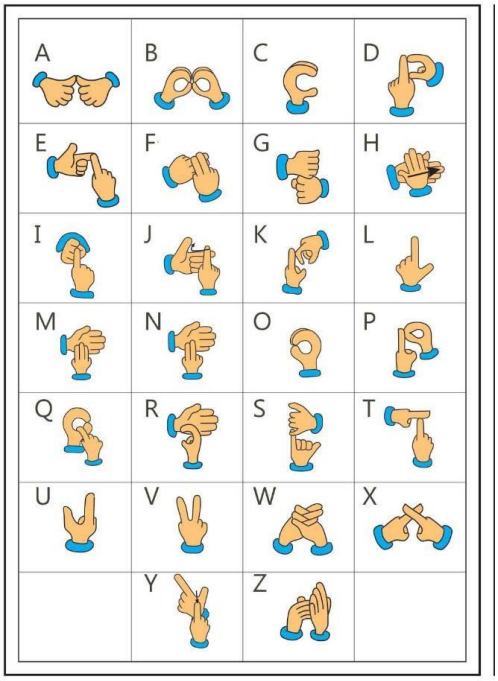

Fig. 3. Gestures for Each Alphabets from ISL [9]

3.Append the loaded data to the window list.

iii. Append the entire window (representing the sequence) to a list named sequences.

iv. Append the corresponding action label (using label_map) to a list named labels.

4. Data Conversion: Convert sequences to NumPy array X and use to_categorical to convert labels to one-hot encoded format (y).
5. Train-Test Split: Split X and y into training and testing sets using train_test_split.

Model Training:

1. Import Libraries:
   Import necessary layers from tensorflow.keras.models and tensorflow.keras.layers for building the LSTM model.
2. Define Model Architecture:
   - Create a sequential model using Sequential from tensorflow.keras.models.
   - Define the LSTM layers with desired units, return sequences behavior (to capture long-term dependencies), and activation functions (e.g., ReLU).
   - Include BatchNormalization layers after LSTM layers for regularization.
   - Add dense layers with decreasing units and appropriate activation functions (e.g., ReLU) for classification.
   - Use softmax activation in the final layer as there are multiple action classes. The number of units in this layer should match the number of actions.

Gesture Recognition Process

1. Data Acquisition:

   The system can operate in two modes:

   I. Real-time: Captures video frames from a webcam.

   II. Image-based: Loads a sequence of image frames representing a gesture.

2. Pre-processing:

   For each frame:

   - The frame is converted from BGR (OpenCV format) to RGB (MediaPipe format) for processing.
   - MediaPipe's Holistic model detects hands and extracts landmarks from the frame (Fig. 2).
   - Key point coordinates (x, y, and z) for detected hand landmarks are extracted.
   - The extracted key points are stored in a sequence buffer, maintaining the most recent 30 frames.

3. Gesture Recognition:

   Once the sequence buffer is full (30 frames):

   - The sequence of key points is transformed into a NumPy array.
   - The array is fed as input to the pre-trained LSTM model (Fig. 4).
   - The model predicts a probability score for each gesture class (A-Z).
   - The gesture class with the highest probability score (above a predefined threshold) is considered the predicted gesture.

4. Output and Feedback:

If a gesture is recognized (above the threshold): The recognized gesture is displayed as text overlaid on the current video frame.

Model Selection – LSTM (Long Short-term Memory)

Reason for Selecting this Model-

In the case of hand sign recognition, where the order of hand movements matters, LSTM's ability to retain important information over longer sequences makes it more effective. It can capture the hand movements better, helping the system recognize different signs more accurately.