Hand gesture mapping using Mediapipe and machine learning algorithm

I. Introduction

1. Motivation

The language of the deaf and hard-of-hearing communities differs from that of normal-hearing individuals. Those who are deaf, hard of hearing, or mute from birth often struggle with communication because they cannot articulate words like others. To address this challenge, sign language was created to help this community communicate and integrate into society more easily. However, sign language is not widely understood by the general public, leading to barriers in effective communication between the two sides. Learning sign language, like any other language, requires significant time and effort. While spoken languages emphasize pronunciation accuracy, sign language relies on making the correct hand signs. Additionally, learners may face difficulties as they do not always have someone to correct their mistakes, provide guidance, or practice with regularly.

2. Objective

This project aims to develop a system that detects hand signs representing the alphabet in sign language, translates them into text, and then converts the text into speech. One of its key objectives is to bridge the communication gap between the deaf and hard-of-hearing communities and the general public. Since sign language is not widely understood, communication barriers often arise. By recognizing hand signs, converting them into text, and even transforming them into speech, this system can facilitate more accessible and effective interactions. Additionally, the system serves as a valuable tool for learning sign language. It can help users identify hand gestures, detect errors, and improve their signing accuracy more efficiently. By providing instant feedback and a convenient learning environment, it supports both beginners and those looking to refine their skills, making sign language more approachable and easier to master.

3. Related work

Recognizing hand gestures in machine learning is considered a challenging task. There are two main approaches to classification: supervised and unsupervised methods. Based on these, sign language recognition (SLR) systems can identify either static or dynamic hand gestures. In 1991, Murakami and Taguchi [3] introduced the use of neural networks in sign language recognition for the first time. As computer vision advanced, new techniques were developed to support the disabled community. For instance, Wang and Popovic [4] created a real-time hand tracking system using colored gloves, with K-Nearest Neighbors (KNN) recognizing the glove patterns, though it required continuous hand movement input. Later studies by Rekha et al. [5], Kurdyumov et al. [6], Tharwat et al. [7], and Baranwal and Nandi [8] found that Support Vector Machines (SVM) outperformed KNN in recognizing hand gestures.

There are two main types of SLR: isolated sign recognition and continuous sentence recognition. Additionally, there are whole-sign level and subunit-sign level models. The subunit level is based on visual-descriptive or linguistic-oriented approaches. Elakkiya et al. [9] proposed a framework combining SVM and boosting algorithms to recognize alphabet subunits, achieving 97.6% accuracy, although it couldn't fully recognize all 26 letters. In Arabic sign language, Ahmed and Aly [10] used a combination of PCA and local binary patterns to extract features for 23 isolated signs, achieving

99.97% accuracy in signer-dependent mode, though issues arose in handling constant grayscale patterns.

Hand gesture recognition was initially tackled with conventional convolutional networks (CNNs), detecting gestures from image frames. R. Sharma et al. [11] trained their model with 80,000 numeric signs, using more than 500 images per sign. Their methodology involved pre-processing images by extracting features and converting them into grayscale for better contour detection, which helped improve the accuracy of the CNN model. W. Liu et al. [12] applied skin saliency techniques for 2D and 3D hand tracking, reaching 98% accuracy in classification.

These studies demonstrate that achieving high accuracy in hand gesture recognition demands large datasets and complex mathematical models. Image pre-processing is crucial for successful gesture tracking. Recently, a new method using Google's open-source MediaPipe framework has been introduced, showing improved accuracy in detecting human body parts. However, this approach has so far been limited to predicting individual letters. Our project aims to optimize the MediaPipe method to directly translate a full conversation into text and then convert it into voice output.

II. Methodology

1. Workflow

Train model step:

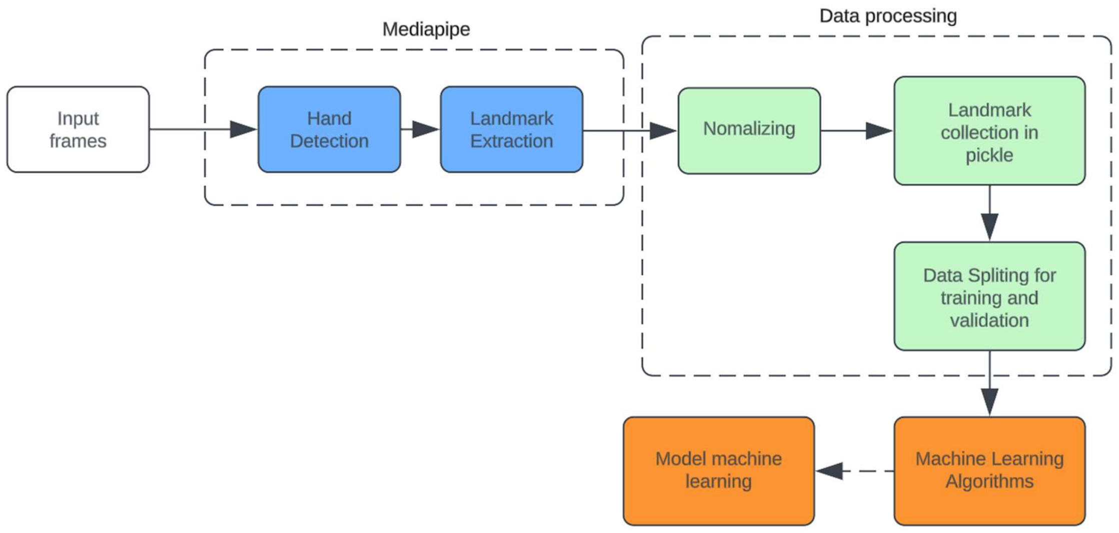


Figure 1: Train model

Real-Time Sign Language Recognition Pipeline:

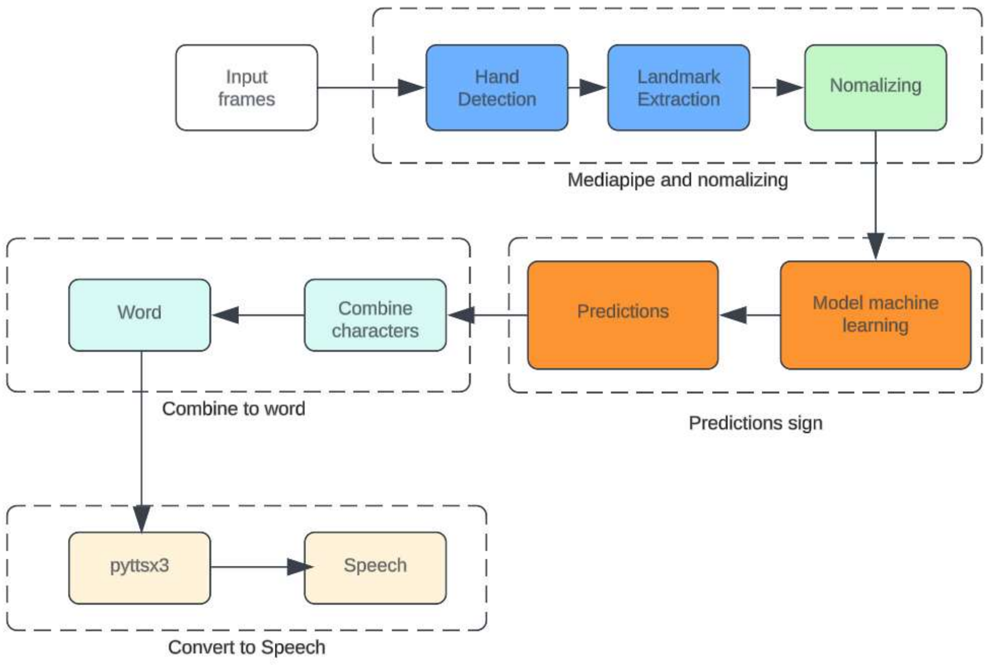


Figure 2: Real-Time Sign Language Recognition Pipeline

2. Method for each step

2.1 Stage 1: Pre-Processing of Images to get Multi-hand Landmarks using MediaPipe

MediaPipe is a framework that enables developers for building multi-modal(video, audio, any times series data) cross-platform applied ML pipelines. MediaPipe has a large collection of human body detection and tracking models which are trained on a massive and most diverse dataset of Google. As the skeleton of nodes and edges or landmarks, they track key points on different parts of the body. All co-ordinate points are three-dimension normalized. Models build by Google developers using Tensorflow lite facilitates the flow of information easily adaptable and modifiable via graphs. MediaPipe pipelines are composed of nodes on a graph which are generally specified in pbtxt file. These nodes are connected to C++ files. Expansion upon these files is the base calculator class in Mediapipe. Just like a video stream this class gets contracts of media streams from other nodes in the graph and ensures that it is connected. Once, rest of the pipelines nodes are connected, the class generates its own output processed data. Packet objects encapsulating many different types of information are used to send each stream of information to each calculator. Into a graph, side packets can also be imposed, where a calculator node can be introduced with auxiliary data like constants or static properties. This simplified structure in the pipeline of dataflow enables additions or modifications with ease and the flow of data becomes more precisely controllable. The Hand tracking solution has an ML pipeline at its backend consisting of two models working dependently with each other: a) Palm Detection Model b) Land Landmark Model. The Palm Detection Model provides an accurately cropped palm image and further is passed on to the landmark model. This process diminishes the use of data augmentation (i.e. Rotations, Flipping, Scaling) that is done in Deep Learning models and dedicates most of its power for landmark localization. The traditional way is to detect the hand from the frame and then do landmark localization over the current frame. But in this Palm Detector using ML pipeline challenges with a different strategy. Detecting hands is a complex procedure as you have to perform image processing and thresholding and work with a variety of hand sizes which leads to consumption of time. Instead of directly detecting hand from the current frame, first, the Palm detector is trained which estimates bounding boxes around the rigid objects like palm and fists which is simpler than detecting hands with coupled fingers. Secondly, an encoder-decoder is used as an extractor for bigger scene context.

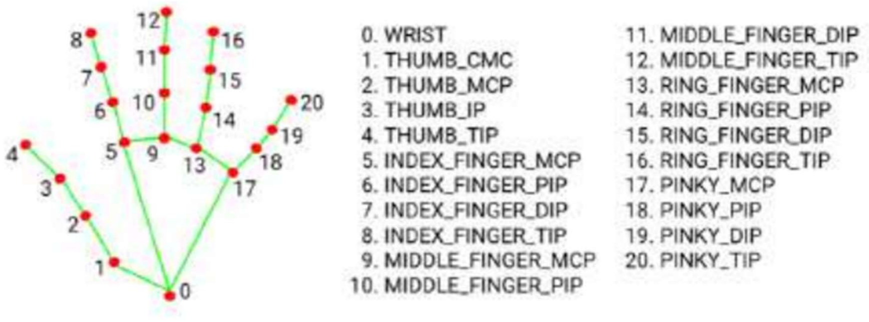


Figure 3: Landmark

After the palm detection is skimmed over the whole image frame, subsequent Hand Landmark models comes into the picture. This model precisely localize 21 3D hand-knuckle coordinates (i.e., x, y, z-axis) inside the detected hand regions. The model is so well trained and robust in hand detection that it even maps coordinates to partially visible hand. Figure 2 shows the 21 landmark points detection by the Hand Landmark model. Now that we have a functional Palm and Hand detection model running, this model is passed over our dataset of various language. Considering the American Sign Language dataset, we have a to z alphabets. So, we pass our detection model over every alphabet folder containing images and perform Hand detection which yields us the 21 landmark points as shown in Figure 2. The obtained landmark points are then stored in a file of CSV format. A simultaneous, elimination task is performed while extracting the landmark points. Here, only the x and y coordinates detected by the Hand Landmark model is considered for training the ML model. Depending upon the size of the dataset around 10-15 minutes is required for Landmark extraction.

2.2 Stage 2: Data cleaning and normalization

As in stage 1, we are only considering x and y coordinates from the detector, each image in the dataset is passed through stage 1 to collect all the data points under one file. This file is then scraped through the pandas' library function to check for any nulls entries. Sometimes due to blurry image, the detector cannot detect the hand which leads to null entry into the dataset. Hence, it is necessary to clean these points or will lead to biasness while making the predictive model. Rows containing these null entries are searched and using their indexes removed from the table. After the removal of unwanted points, we normalized x and y coordinates to fit into our system. The data file is then prepared for splitting into training and validation set. 80% of the data is retained for training our model with various optimization and loss function, whereas 20% of data is reserved for validating the model.

2.3 Stage 3: Prediction using Machine Learning Algorithm

The output from Mediapipe, which consists of the coordinates of 21 landmark points, will be passed through a machine learning model to predict the result.

2.4 Stage 4: Combine characters to text

Instead of processing each frame individually and directly producing the corresponding text, the first three stages will handle groups of 10 frames. Predictions will be made for each frame in the group, and the result with the highest frequency will be selected as the output, which will then be passed to stage 4.

Stage 4 will involve a two-tier data storage system. When a character is passed to stage 4, it will be compared with the previous data stored in the buffer. If the new character differs from the stored data, it will be added to the output text. If it matches the buffered data, it will be discarded.

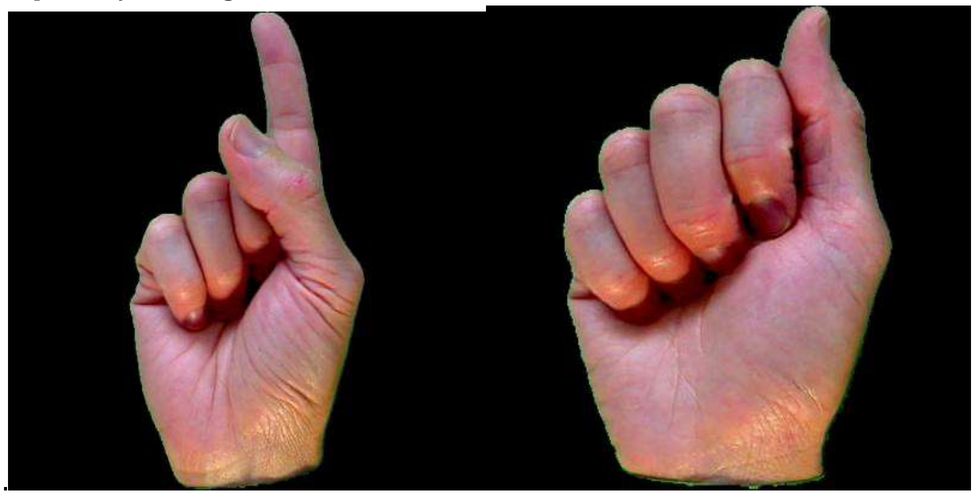
2.5 Stage 5: Convert text to speed

We use library pipeline to convert text to speed.

III. Dataset

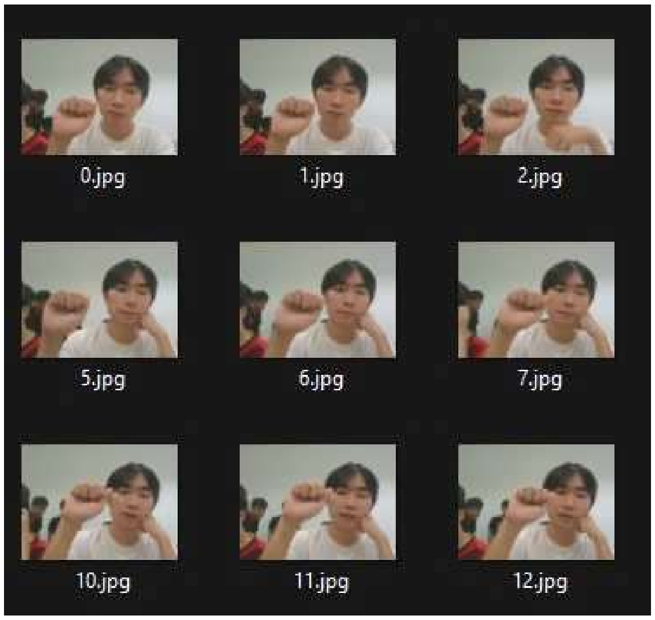
1. Data Source

- **External data collection:** We can turn to publicly available data sources, such as academic data repositories, and data-sharing communities like Kaggle, the UCl Machine Learning Repository, or Google Dataset Search.



From Kaggle

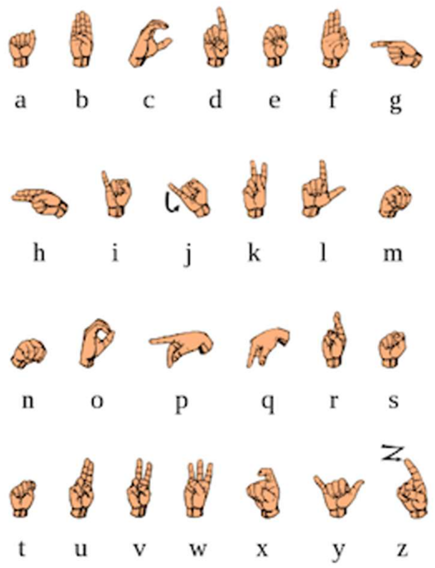
- **Web scraping:** This technique involves extracting data from websites using automated tools. It can be useful for gathering data from sources that cannot be accessed through other means, such as product reviews, articles, and social media.
- **Create data:** The data is generated by the members of the team themselves.



From our member

2. Data Description

Data includes 26 characters



- **Data Types: Image Data**
- **Annotations/Labels:** The data is usually labeled with the corresponding sign or meaning. For instance, a video frame showing a certain hand movement is annotated as representing the letter "A" in American Sign Language (ASL). These labels are crucial for training machine learning models.
- **Data Format:** The data is typically stored in formats picture: .jpg, .png, etc

IV. Model Development

1. Data Collection and Preprocessing:

- **Collection:** Images and videos of ASL gestures are gathered, often from publicly available datasets.
- **Preprocessing:**
    - **Hand Detection:** Using OpenCV and Mediapipe to detect and segment hand regions.
    - **Resizing and Normalizing:** Images are resized to a uniform size (e.g., 64x64 pixels) and pixel values are normalized.

2. Feature Extraction:

- **Landmark Detection:** Mediapipe extracts 21 key hand landmarks (x, y, z coordinates) for each frame.
- **Feature Vector:** These coordinates form a feature vector representing the hand's position and posture.

3. Model Training:

- **Data Splitting:** The dataset is split into training and testing sets.
- **Classifier Selection:** A machine learning model (e.g., Support Vector Machine (SVM) or neural network) is chosen.
- **Training:** The model is trained on the training set using the extracted features.
- **Hyperparameter Tuning:** Parameters are optimized to improve model performance.

4. Inference:

- **Real-time Prediction:** The trained model predicts ASL gestures from live video input, using the same preprocessing and feature extraction pipeline.

V. Evaluation Metrics

A. Evaluation Metrics: The performance of the ASL detection model is assessed using several key metrics, including accuracy, precision, recall, and the F1-score. These metrics provide insights into how well the model is performing and help identify areas for improvement.

1. Accuracy:

- **Definition**: The ratio of correctly predicted instances to the total instances.
- **Formula**: Accuracy = Number of Correct Predictions / Total Number of Predictions

$$acc = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

- **Interpretation**: Higher accuracy indicates better overall performance, but it might be misleading if the dataset is imbalanced.

2. **Precision**:

- **Definition**: The ratio of correctly predicted positive observations to the total predicted positives.
- **Formula**: Precision = True Positives / (True Positives+False Positives)
- **Interpretation**: Precision indicates the accuracy of the positive predictions made by the model.

3. **Recall (Sensitivity)**:

- **Definition**: The ratio of correctly predicted positive observations to all the observations in the actual class.
- **Formula**: Recall = True Positives / (True Positives+False Negatives)
- **Interpretation**: Recall measures the model's ability to detect all positive instances.

4. **F1-Score**:
   - **Definition**: The weighted average of Precision and Recall.
   - **Formula**: F1-Score = 2 × (Precision×Recall) / (Precision+Recall)
   - **Interpretation**: The F1-score is a balance between precision and recall, and it is especially useful when the class distribution is imbalanced.

B. Model Evaluation Process

1. **Confusion Matrix**:
   - A confusion matrix is created to visualize the performance of the classification model. It displays the true positive, true negative, false positive, and false negative predictions, helping to identify misclassified gestures.
2. **Accuracy**:
   - Calculate the ratio of correctly predicted gestures to the total number of predictions. This metric provides a general performance overview.
3. **Precision, Recall, and F1-Score**:
   - Precision: Measures the proportion of true positive predictions among all positive predictions.
   - Recall: Measures the proportion of true positives detected among all actual positives.
   - F1-Score: The harmonic mean of precision and recall, offering a balanced evaluation.
4. **Classification Report**:
   - A detailed report showing precision, recall, F1-score, and support for each class, providing insights into the model's performance on a per-class basis.

REFERENCES:

[1] Shukor AZ, Miskon MF, Jamaluddin MH, Bin Ali F, Asyraf MF, Bin Bahar MB. 2015. A new data glove approach for Malaysian sign language det ection. Procedia Comput Sci 76:60–67

[2] Almeida SG, Guimarães FG, Ramírez JA. 2014. Feature extraction in Brazilian sign language recognition based on phonological structure and using RGB-D sensors. Expert Syst Appl 41(16):7259–7271

[3] Murakami K, Taguchi H. 1991. Gesture recognition using recurrent neural networks. In: Proceedings of the ACM SIGCHI conference on Human factors in computing systems, pp 237–242. https ://dl.acm.org/doi/pdf/10.1145/10884 4.10890 0

[4] Wang RY, Popović J. 2009. Real-time hand-tracking with a color glove. ACM Trans Graph 28(3):63

[5] Rekha J, Bhattacharya J, Majumder S. 2011. Hand gesture recognition for sign language: a new hybrid approach. In: International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), pp 80–86

[6] Kurdyumov R, Ho P, Ng J. 2011. Sign language classification using webcam images, pp 1–4. http://cs229 .stanford.edu/proj2 011/ Kurdy umovH oNg-SignL angua geCla ssifi catio nUsin gWebc amIma ges.pdf

[7] Tharwat A, Gaber T, Hassanien AE, Shahin MK, Refaat B. 2015. Sift-based arabic sign language recognition system. In: Springer Afro-European conference for industrial advancement, pp 359–370. https ://doi.org/10.1007/978-3-319-13572-4_30

[8] Baranwal N, Nandi GC. 2017. An efficient gesture based humanoid learning using wavelet descriptor and MFCC techniques. Int J Mach Learn Cybern 8(4):1369–1388

[9] Elakkiya R, Selvamani K, Velumadhava Rao R, Kannan A. 2012. Fuzzy hand gesture recognition based human computer interface intelligent system. UACEE Int J Adv Comput Netw Secur 2(1):29–33 (ISSN 2250–3757)
[10] Ahmed AA, Aly S. 2014. Appearance-based arabic sign language recognition using hidden markov models. In: IEEE International Conference on Engineering and Technology (ICET), pp 1–6. https ://doi.org/10.1109/ICEng Techn ol.2014.70168 04
[11] R. Sharma, R. Khapra, N. Dahiya. June 2020. Sign Language Gesture Recognition, pp.14-19
[12] W. Liu, Y. Fan, Z. Li, Z. Zhang. Jan 2015 . Rgbd video based human hand trajectory tracking and gesture recognition system in Mathematical Problems in Engineering,