Sign Language Translation in WebRTC Application

Submitted in partial fulfilment of the

requirements for the award of

Bachelor of Engineering degree in Computer Science and Engineering

By

GANGADHAR CHAKALI (39110202)

CHINATHALACHERUVU GOVARDHAN REDDY (39110235)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHISALAI,

CHENNAI - 600119

NOVEMBER - 2023

SATHYABAMA

INSTITUTE OF SCIENCE ANDTECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with ¯A‖ grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai - 600119

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Gangadhar Chakali (39110202) and Chinthalacheruvu Govardhan Reddy (39110235),** who carried out the Project Phase-1 entitled **"Sign Language Translation in WebRTC Application"** under my supervision from June 2022 to November 2022.

Internal Guide

Dr B. BHARATHI M.E., PhD.

Head of the Department
Dr L. LAKSHMANAN,M.E.,PhD.

Submitted for Viva-voce Examination held on 24.03.23

Internal Examiner **External Examiner**

DECLARATION

I, **Gangadhar Chakali (Reg.No-39110202),** hereby declare that the Project Phase-1 Report entitled **"Sign Language Translation In WebRTC Application"** done by us under the guidance of **Dr. B. Bharathi, M.E., Ph.D.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 19-04-2023 Gangadhar Chakali

PLACE: Chennai **SIGNATURE OF THE CANDIDATE**

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr T. Sasikala M.E., Ph. D**, **Dean**, School of Computing, and **Dr L. Lakshmanan, M.E., Ph.D.** Head of the Department of Computer Science and Engineering for providing me with the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. B. Bharathi M.E., Ph.D.,** for her valuable guidance, suggestions, and constant encouragement that pave the way for the successful completion of my phase-1 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Communication has been an essential part of human life in which humans use spoken language as means of expression along with feelings. Due to the evolution of humans with human intelligence, different languages have evolved worldwide, but people lost their hearing ability and speaking ability by accidents, genetic birth, etc. Auditory-impaired people have found sign language helpful in communicating with others. In America, 28 and 32 million deaf and mute people communicate using American sign language (ASL). This method of communication has some limitations, such as the requirement that others must also learn sign language to comprehend what the deaf person is trying to say. Hearing people must always have a personal interpreter available for translations whenever they need to communicate. Hearing-impaired people must always have a personal interpreter available for translations whenever they need to communicate.

Sign language recognition (SLR) has drawn a lot of attention as a way to close the enormous communication gap. People with disabilities and the hearing impaired find it challenging to interact with others and form new relationships without open-source video-calling services for sign language translation. However, sign language is far more complex and unpredictable when compared to other activities, making it challenging for reliable recognition. Sign language is composed of both fine-grained finger motions and coarse-grained arm motions. The Speech-to-Text API enables mute and deaf people who can read to comprehend others. The Sign Language Translation Application (SLTA) allows them to communicate by translating their sign language into text that others can understand.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Human beings communicate with each other through different languages and expressions. The first language evolved around 50,000 to 150,000 years ago. Since then, humans have depended on voice communication to understand each other better. Not everyone in this world is gifted to speak and hear to communicate and understand others. Many people around the globe suffer from hearing and speaking problems and are referred to as deaf and mute people or hearing impaired. Approximately around 18 million Indians and 300 million around the world are suffering from auditory impairment. It is hard to communicate with these people because of their disabilities. With this disability, hearing-impaired people find it challenging to find new connections and make new relationships. These people communicate with each other through sign language. Sign language has wide varieties like American Sign Language (ASL), Chinese Sign Language (CSL), Arabic Sign Language, etc. American Sign Language (ASL) is the most recognized and widely used in different countries. With this sign language, there is a huge gap in communication with others. To communicate with people suffering from an auditory impairment, everyone should know sign language to interpret and understand each other.

Creating an interface that translates this sign language to the preferred language and vice versa will minimize the communication gap. This study attempts to develop a system that helps to communicate through the video-conferencing application. For developing the Sign Language Recognition (SLR) system, this study uses deep learning techniques to collect, train, and test data with open-source tools like Tensorflow, Keras, NumPy, Mediapipe framework, OpenCV, etc.

In the first part of the review, there is an explanation of the usage of sign language, how open source platforms use SLR services, the technology used, studies that help understand SLR model development in the part of the literature review, and inferences from it.

The second part of the review explains the definitions and analysis of the requirements

essential for developing the model and software integration tools used in this study and the description of Software requirements specifications (SRS).

The third part of the review explains the proposed model, describes software implementation and explains the development process. Explanation of the model's description, performance, test results, and proposal of a project management plan to fulfil project requirements.

1.1 Area of Research

Machine learning (ML) and Deep Learning (DL) investigate computer science algorithmic calculations that can work naturally through experience and data. It is part of artificial intelligence. Machine learning algorithms fabricate a model dependent on example data, known as "Training data," to make predictions or decisions without programming for every application. Machine learning algorithms are utilized in various applications, such as medication, email sorting, speech recognition, and computer vision. Developing standard algorithms to perform the required tasks is troublesome or unworkable. A subset of machine learning is rigidly identified with computational statistics, which cent ers around making computer predictions, yet not all machine learning is statistical learning. The investigation of mathematical optimization conveys strategies, hypotheses, and application areas to the field of machine learning. Data mining is a related field of study, focusing on exploring data investigation through unsupervised learning. Some executions of machine learning use data and neural networks that impersonate the working of an actual brain. Machine learning is likewise referred to as proactive analysis in its application across business issues.

CHAPTER 2

LITERATURE SURVEY

Researchers and computer scientists have researched this problem statement in great detail over the past few years to find a solution, and all of their answers range from examining numerous patterns of gesture recognition techniques to the analysis of different sign languages and different styles of data collection.

The study by author Yeresime Suresh [19] suggested the method employs Canny Edge detection, which provides a more accurate result in detecting edges to identify the edges of hand gestures in the frames. Compared to using embedded sensors in gloves to collect for the identification of gestures, as seen in [3], the suggested system also uses The prediction model of Convolutional Neural Networks (CNN) transcription of speech and hand motions. All CNN and Canny Edge Detection results are reliable and accurate when trained with a large dataset.

The study by Karly Kudrinko explained the feasibility of using wearable sensor-based devices to recognize hand movements in applications directly connected to sign language shown in her work. His review aims to identify trends and best approaches by examining earlier studies. The review also points out the difficulties and gaps the sensor-based SLR field is experiencing. Our analysis could help create better wearable sensor-based SLR systems that apply in real-world situations [13]. Due to diverse study methodologies, a standardized data collection protocol and evaluation processes have also been developed for his field.

Mathieu De Coster's [12] study tested a variety of neural network topologies, including Hidden Markov Models (HMM), Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNN), To improve the continuous SLR model. OpenPose is a framework used as a feature extractor to gather the skeleton motion of gestures. Since SLR relies not only on hand form, location, and orientation but also on non-manual components like mouth shape, OpenPose is the sole full-body pose estimation method and is thus an obvious choice. There are other pose estimate methods, but they only identify, for instance, body key points or hand key points (Fang et al., 2017). (Mueller et al., 2018). He retrieved data using the

OpenPose framework, trained the model, and developed the model.

In a study by Bhushan Bhokse, he created a program for gesture recognition that allows a user to demonstrate his hand making a particular motion in front of a video camera connected to a computer. The computer program must gather images of this move, examine it, and recognize the sign. To make the project more manageable, it was decided that identification would include counting the user's fingers and identifying the American Sign Language they use in the input image [11]. He performed experiments using static images on simple backgrounds, retrieved the pictures as grayscale images, and used the binary data from the image for gesture detection.

Sushmita Mitra described how some gestures, like those used in sign languages, include both static and dynamic components in a study she conducted [14]. Additionally, talk about how various body motions, including those made with the hands, arms, head, and torso, are used in sign languages. She conducted tests on different face, hand, and body movement-based algorithms for sign language recognition for the evaluation. She improved accuracy in those studies by fusing Hidden Markov Models (HMM) and finite state machines (FSM) in the form of hybridization.

Zhibo Wang's study covered past works and system trials on various SLRs with wearables that have embedded sensors, such as RF-based [5], PPG-based [8], Acoustic-based [6], Sensing Gloves[7], Vision-based [2][4], EMG-based [9][10][3], and SignSpeaker [1]. These tests are contrasted with his DeepSLR work, which is based on a multi-channel CNN architecture and produces findings that It takes less than 1.1 seconds to detect signals and recognize a sentence with four sign words, demonstrating DeepSLR's recognition effectiveness and real-time capability in practical situations. The average word error rate for continuous sentence recognition is 10.8%.

2.1 INFERENCES FROM LITERATURE SURVEY

AUTHOR	YEAR OF PUBLICATION	INFERENCE
Yeresime Suresh [19]	2020	The CNN algorithm can yield better accuracies for static images but fails to predict the motion
Mathieu De Coster [12]	2020	Multimodal Transformer Networks can accommodate large vocabulary with acceptable accuracy, but the open pose framework fails to differentiate the intersection of key points.
Karly Kudrinko [13]	2020	Using modified K-Nearest Neighbours and data collected from DG5-VHand Glove 180 can predict Arabic Sign Language of the vocabulary of 40 with an accuracy of 98.9%. Still, gloves are needed for this system to work.
Zhibo Wang [6]	2020	using the LSTM algorithm to detect motion gestures is possible but using data from the hand bands of the sensor is not cost-effective
Bhushan Bhokse [11]	2015	using a feed-forward network can be used to get better prediction but using grayscale with a plain background is a naïve approach.
Sushmita Mitra[14]	2007	The prediction of gestures is more accurate than any algorithms using a Hybrid of the Hidden Markov model (HMM) and finite State Machine (FSM). Still, it has been tested only on facial expressions.

2.2 OPEN PROBLEMS IN THE EXISTING SYSTEM

Most studies have been conducted based on gesture recognition, facial expression recognition, hand sign recognition over a static image, and gesture motion using wearables embedded with sensors. Open problems in the existing system are size vocabulary and motion gesture model trained with 2D camera image over wearables sensor gloves data.

This study will overcome the motion sign gesture recognition using camera 2D video and develop an LSTM model `to translate signs into a preferred language.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDY OF THE PROJECT

This study aims to develop a sign language translator who helps the hearing impaired communicate with others with the sign language barrier that others need to learn to understand and communicate with the speech impaired. For the development of this project, there is a need for a large vocabulary of words assigned with gesture video, which is an enormous task for this project. To build this corpus, it is essential to manually monitor and collect each word gesture video with different possible scenarios to make the model versatile to all conditions. The NLP model, which converts words to sentences, needs to be more accurate as possible because the primary purpose of this study is to reduce the communication gap. Still, if the sentences are not appropriately formed, it defeats the whole purpose. There is also a need to take care of the system integration issues, delays in detection and conversion, video transmission latency, and more. This requires continuous learning and integration of the vocabulary for a better SLR system.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

3.2.1 Introduction

3.2.1.1 Purpose of this document

The purpose of this document is to identify the essential requirements of the software and define the requirements to make sure the development is moving in the right path

3.2.1.2 Scope of this document

This project's main objective is to help the hearing-impaired (deaf and dumb) to communicate with others without the involvement of a third person to interpret the sign language. This software acts as an interpreter during video conferencing, where the conversion of signs to words and words to sentences happens in real-time.

3.2.1.3 Overview

Using static signs to predict words and model prediction dependence on sensor gloves will resolve some of the issues with the current methods in the proposed approach. By utilizing an intelligent framework that can extract essential details of the skeletal gesture structure using a digital camera, which is often included in any personal computer (PC) device, the issue of model dependence on sensor gloves can be lessened. This is achievable because of Google's MediaPipe framework, which can map the human body's skeleton and extract the coordinates of those critical point regions for gesture identification. This and other frames can build a motion sequence representing a sign language word.LSTM algorithmic Neural Network topologies can perform better for motion recognition, according to our literature analysis. This addresses the second issue: the limitation of predicting signs using only static images. As a result, the possibility broaden the model's language and increase its adaptability to various situations. For a seamless conversation, it is possible to turn these predicted words into sentences using the NLP model by showing sentences as captions in the interface. The NLP model will provide the sentences with additional meaning before sending them to the WebRTC transmission stream. The Spoken-to-Text paradigm can assist the hearing impaired in reading and understanding by turning speech recordings into text.

3.2.2 General description

There is a need for the hearing impaired to communicate with others in different scenarios but not every time the other person can understand the sign language. In most of the surveys conducted by the researchers, the need hearing impaired find it difficult to make new connections with the lack of communication. This WebRTC application helps translate sign language into live text and converts speech to text in live captions, which helps in communication during a video call.

3.2.3. Functional Requirements

The development of a deep learning model that can predict the motion gestures of human sign language users. They begin with collecting data in the form of video clips captured by the webcam of a PC. These video clips are then preprocessed with the MediaPipe framework to extract the skeleton structure key points to sequence data. After the development of the LSTM model, it gives words as output which are

transferred to the NLP model, which helps form meaningful sentences with words to sentences. Along with LSTM and NLP models, there is a need for the development of speech-to-text models which allow the deaf to understand the speaker's saying.

3.2.4. Interface Requirements

During the video call person with a disability communicate with sign language. These are recorded and converted to words by the LSTM model, where these words are converted to sentences by the NLP model. These sentences are sent through the network and displayed as captions to the other person. For speech-to-text, the speech is converted to a sentence by the NLP model and displayed as text for the hearing impaired, which came through the network.

3.2.5. Performance Requirements

For seamless communication and rich vocabulary prediction, Graphical Processing Units (GPUs) process the live feed data without latency and delay in communication needed. And also, to collect, train and test the large vocabulary, high-performance hardware is needed to develop the model. For small vocabulary, better accuracies can be achieved, but for better communication, the requirement to build a large dataset of sign language words

3.2.6. Design Constraints

With present hardware, testing and training of a small amount of the words can be done, and WebRTC Application development comes as a whole new project for adding new features, making it user-friendly, etc.

3.2.7. Non-Functional Attributes

Deployment of the WebRTC application should be done on the cloud with cloud computing capabilities with GPUs to process the real-time data for sign language translation.

3.2.8. Appendices

LSTM	Long Short-Term Memory
ASL	American Sign Language
WebRTC	Real-Time Communication using web
GPU	Graphical Processing Unit
SLR	Sign Language Recognition
CNN	Convolution Neural Network
HMM	Hidden Markov Model
SLTA	Sign Language Translation Application
NLP	Natural Language Processing

CHAPTER 4

DESCRIPTION OF THE PROPOSED SYSTEM

In this proposed system, the problems addressed in the existing systems will be solved, like model prediction dependence on sensor gloves and static signs to predict the words. The issue of model dependence on sensor gloves can be reduced by using an intelligent framework that can extract key points of gesture skeleton structure using a digital camera commonly embedded in any personal computing (PC) device. This is possible with the MediaPipe framework developed by Google, which can map the skeleton of the human body and extract coordinates of those key point areas for gesture recognition. Combined with multiple frames, this can create a motion sequence that can make a word in sign language. From the literature survey, LSTM algorithmic Neural Network architectures could perform better for motion recognition. This solves the second problem: the barrier of using only static images to predict signs. This means expanding the vocabulary and making the model more versatile and usable in different conditions.

Using the NLP model, these predicted words can be converted into sentences. The NLP model will add meaning and send it to the WebRTC transmission stream to make a seamless communication by displaying sentences as captions in the interface. The Speech-to-Text model can also help the deaf to read and understand by converting audio of speech and converting to text

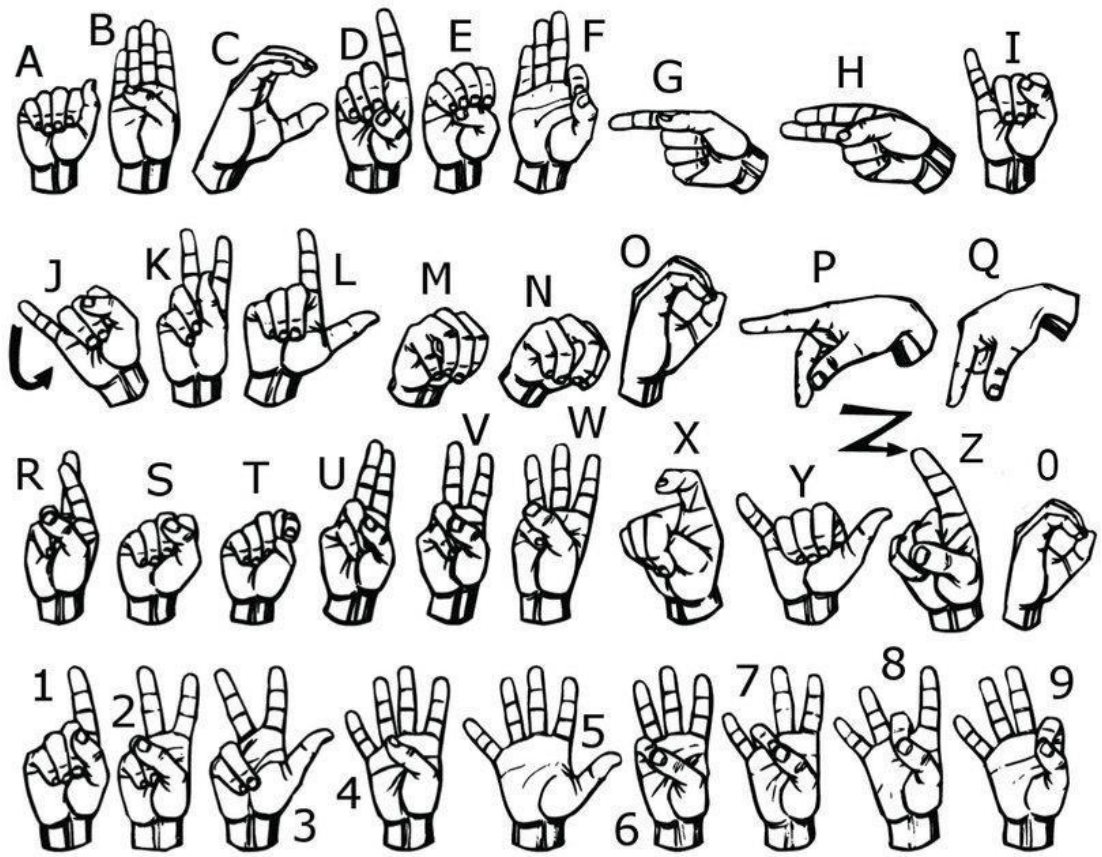


Fig 4.1: Alphabet and numbers of American Sign Language(ASL)

4.1 SELECTED PROCESS MODEL

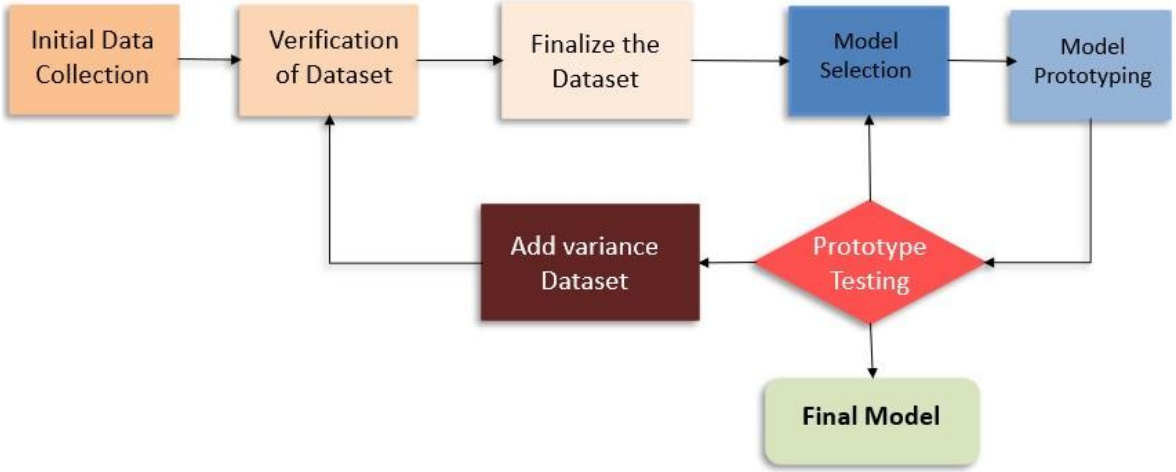


Fig 4.2: LSTM model development flow process

4.2 ARCHITECTURE OF THE PROPOSED SYSTEM

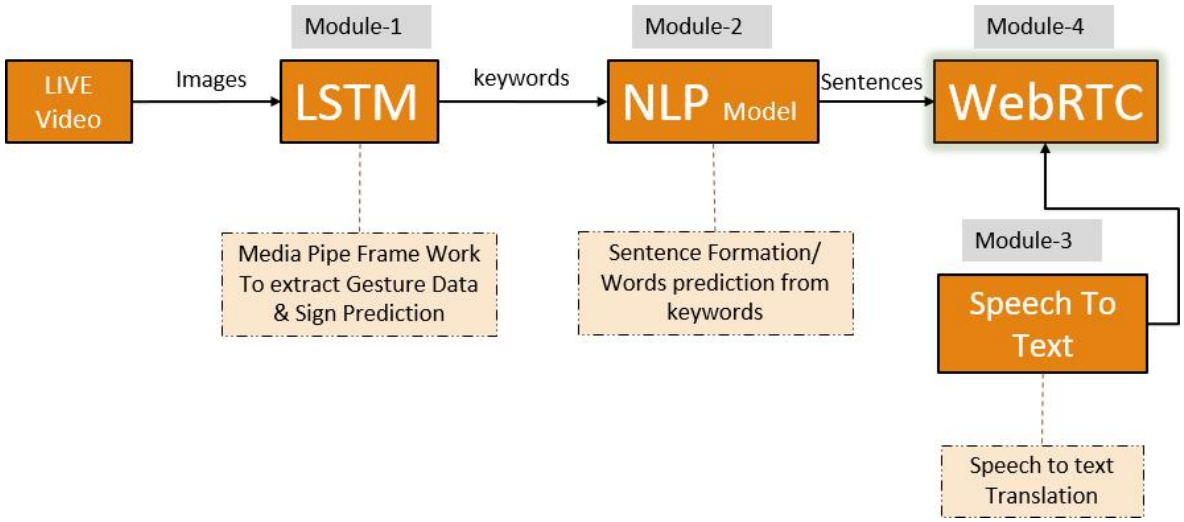


Fig 4.3: System Architecture for Sign Language Translation in WebRTC application

The proposed system's block diagram is the above. The camera captures continuous images of the signs during the web real-time communication that the person displays. The model extracted the key points of edges using MediaPipe Framework [18] from the continuous images and fed them to the SLR model that has been developed, which will convert signs to words. These words are then given to the Natural Language Processing (NLP) model, and these words are converted to sentences displayed in the WebRTC in the form of captions. Another model is developed where it converts speech to text and displays text as a caption in WebRTC, which helps the deaf to understand what the other person is saying.

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL

Initially, begin with collecting the data of the signs in the form of video clips captured by the webcam of a PC. Starting with three words to predict, experiments will be conducted up to 20 words with good model performance. These video clips are then preprocessed with the MediaPipe framework to extract the skeleton structure key points to sequence data which can be used to train the deep learning model. The opted algorithm's proposed model is Long Short Term Memory(LSTM) modeling, which helps predict the motion gesture. Begin with developing a basic LSTM architecture for initial data training and testing, and gradually improve the architecture by testing and adding the vocabulary.

After developing the model, the next step is to develop the NLP model, which helps form meaningful sentences with words to sentences. Along with LSTM and NLP models, there is a need to develop speech to text model which allows the deaf to understand the speaker's saying. With the development of all modules to integrate into the WebRTC software where sign-to-words, words-to-sentence, and speech-to-text conversion happens in real-time. The challenges faced during this process are latency in communication, delay in prediction, and all the problems +encountered in the soft real-time communication system.

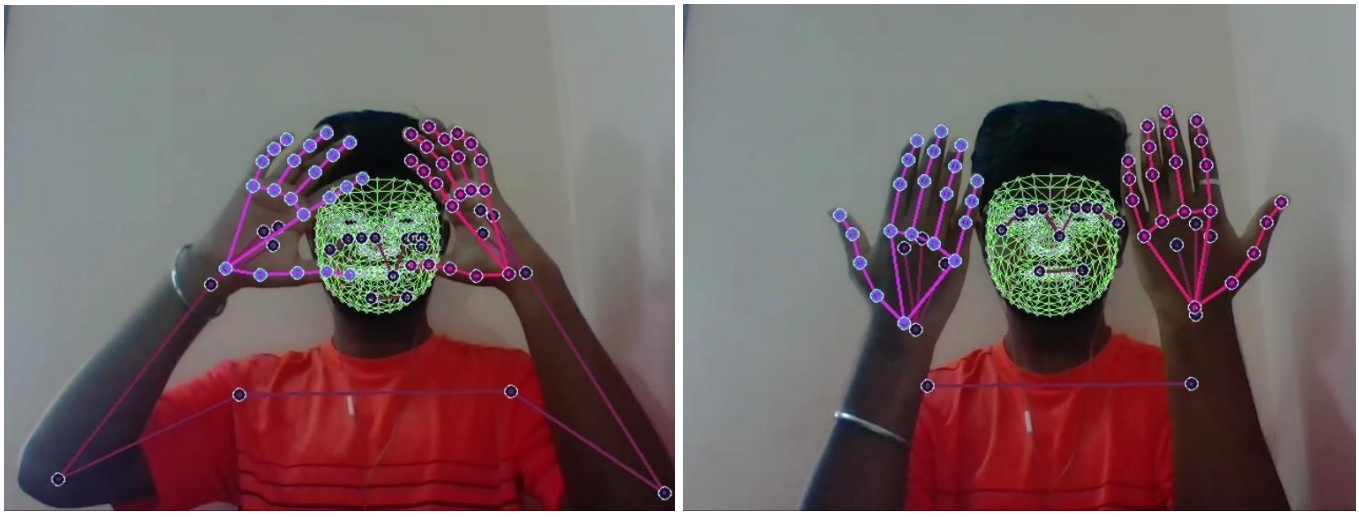


Fig 4.4: Keypoints mapping to live feed predicted from MediaPipe Framework

4.4 PROJECT MANAGEMENT PLAN

Weeks 1-2	• Data Collection of sign Language • Initial Model Training and Testing • Developing LSTM architecture
Weeks 2-4	• Testing model performance • Collection of data for Expanding Vocabulary • LSTM Model Evaluation
Weeks 4-8	• Finalizing LSTM Architrcture • model development and Traning with new data • LSTM model evaluation
Weeks 8-12	• Testing with Real Time data • Development of a speech-to-text model • Speech-to-text model evaluation
Weeks 12-15	• Testing Speech-to-text API • Application development for demonstration • Software deployment (if possible)
Weeks 15-18	• Finalizing Documentation • Research paper preparation • Presenting a paper at the conference • Publishing the research paper

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

Sign Language Translation Application (SLTA), helps people who face challenges with communication by translating American Sign Language (ASL) for communication, who aren't exposed to sign language communication. SLTA is a video-conferencing application that makes use of WebRTC protocols for real-time two-way video and audio communication. It is similar to Google Meet, Zoom, and Microsoft Teams, which are embedded with DGR and NLP models that help in SLR. Development of such a system comes with specific challenges like creating visual motion gestures, the vocabulary of a language, training a neural network to accommodate vast vocabulary for Prediction, and making a user-friendly interface system to use the ML translator model.

SLTA can capture the hand gestures with the help of python libraries of OpenCV and MediaPipe, via video frames of the user camera on PC that detects the positions of hand, palm, torso, and face for spatial positioning landmarks of the person, and 21 points of each palm which are coordinates of pixels that help to predict the sign more precisely as shown in figure 5.2.

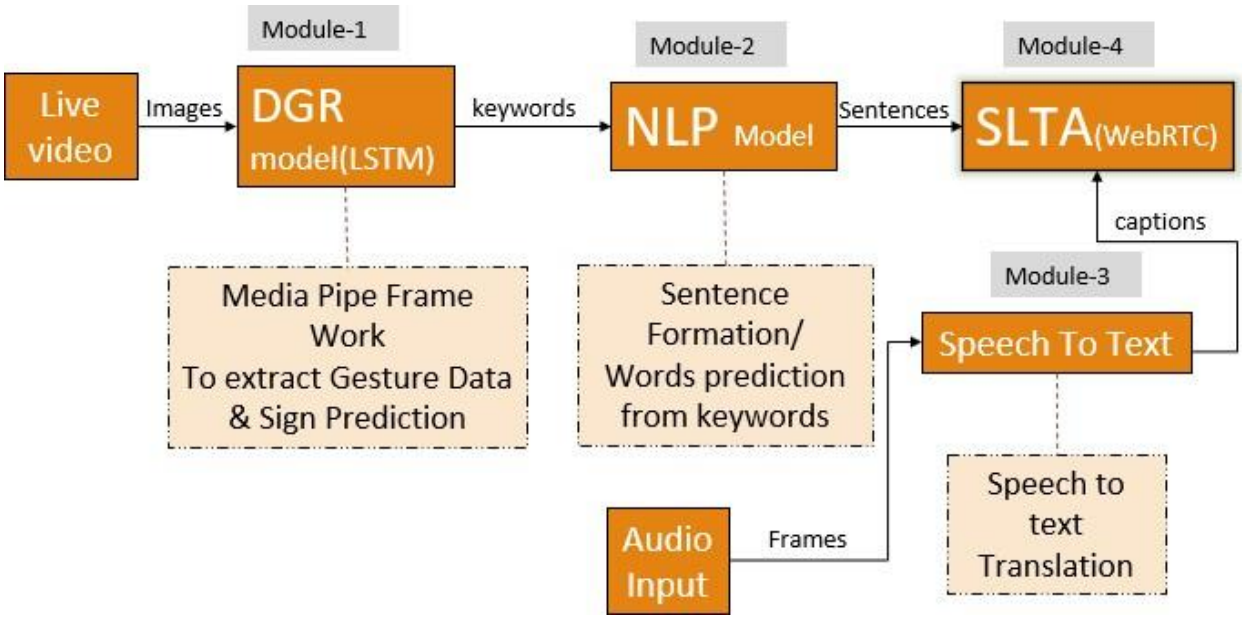


Fig. 5.1. System Architecture of Proposed System SLTA

Fig. 5.2. Hand Landmarks detected by Media-Pipe

The positions of the face, hand, and torso are generally involved with ASL, where movements of different combinations of fingers gesture separate each word in the vocabulary. In a gesture, the positions of the face, body, and torso are important as they create unique vocabulary relating to the wide sign gesture's actions. MediaPipe Framework can also detect the 33 points in pose holistic landmarks (coordinates) of the full body, as shown in figure 5.3. Most of the ASL vocabulary doesn't involves the legs and hip. Only the first 22 points mentioned in figure 5.3 are used to record the dataset.

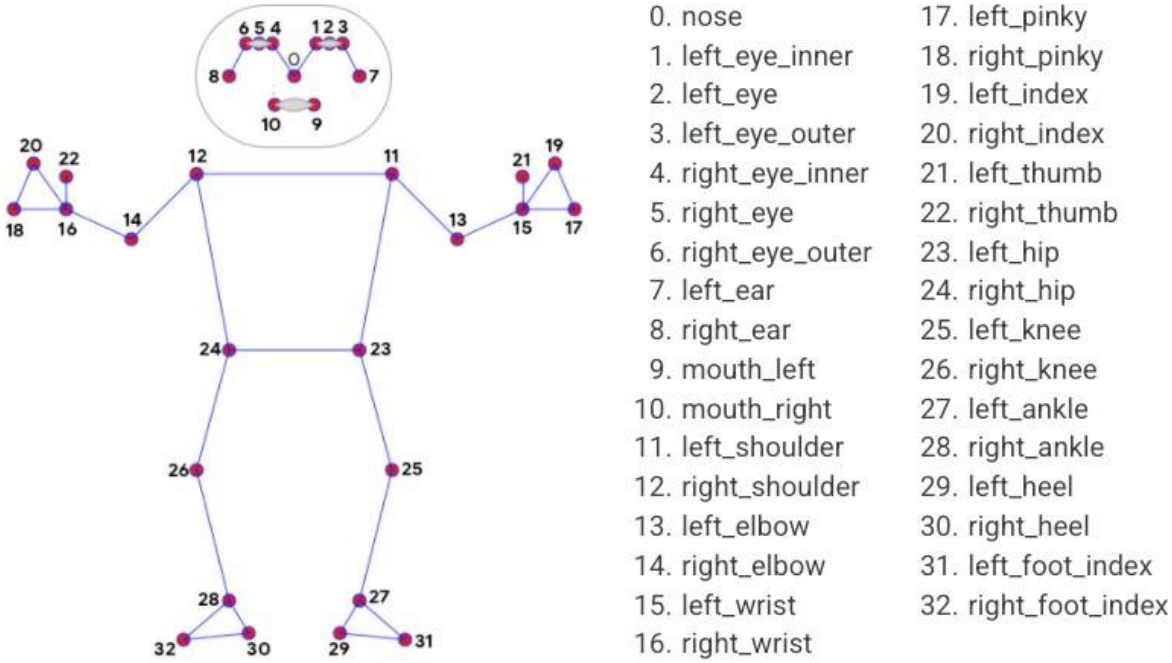


Fig. 5.3. Pose Full Body Landmarks detected By Media-Pipe

ASL follows the subject-verb-object (SVO), meaning sentences are communicated through sign

language without breaks. Phrases intended to communicate are understood with the help of the topic of discussion and verbs signed related to the topic by the cognitive perception of humans. Sign language gestures can be converted to words using the Deep Gesture Recognition model (DGR), developed using a neural network consisting of LSTM units, a module in SLTA. Previous work has shown the limitations of the Convolution Neural Network (CNN), which can only predict static sign gestures. Another study on motion gesture recognition employed wearables embedded with sensors that collected spatial 3D data of the hand motion of gestures with acceptable performance using LSTM neural networks. Still, it is challenging to employ economical gloves that require less maintenance. The DGR model uses and accepts the data from a digital front camera commonly found in personal devices as a primary video source. Data from a digital camera is pipelined to the SLTA via the OpenCV module, which helps to manage the digital camera data and manipulate it according to the requirement. MediaPipe Framework [18] makes gesture recognition using Machine Learning (ML) and Deep learning (DL) models for the detection of gesture holistic data of the person at every frame of the gesture video, and predicting the motion gesture with LSTM neural network is trained with ASL. DGR model can be able to detect sign language and predict the vocabulary in keywords form like "EAT," "DRINK," "HELP," "HELLO," and "THANK YOU," but can't predict the tense of the keywords like "EATING," "DRANK," "HELPED" and also it can't predict articles and prepositions in the English language. Natural Language Processing (NLP) model can analyze and predict sentences with grammar and reconstruct sentences with the meaning the user meant to communicate. Sentences that are predicted and reconstructed by the NLP model are displayed as closed captions (CC) in the SLTA application. Hearing-impaired people are subjected to understanding words and sentences of a language, which they learned with sign language as they progress the communication. The context of the commoner on the other side of the SLTA can be understood by the hearing impaired using the speech-to-text model, which converts speech to live captions of the speaker, making reading the context easy for the hearing impaired. Developing a user-friendly interface and features is possible for future work by using the Web Real-Time Communication (WebRTC) method and software engineering techniques.

5.2 ALGORITHMS

5.2.1 OpenCV

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today’s

systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features. The first OpenCV version was 1.0. OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. When OpenCV was designed the main focus was real-time applications for computational efficiency. All things are written in optimized C/C++ to take advantage of multi-core processing.

5.2.2 MediaPipe FrameWork

The MediaPipe Gesture Recognizer task lets you recognize hand gestures in real time, and provides the recognized hand gesture results along with the landmarks of the detected hands. You can use this task to recognize specific hand gestures from a user, and invoke application features that correspond to those gestures.

This task operates on image data with a machine learning (ML) model, and accepts either static data or a continuous stream. The task outputs hand landmarks in image coordinates, hand landmarks in world coordinates, handedness (left/right hand), and the hand gesture categories of multiple hands.

5.2.2.1 Holistic landmarks detection

The MediaPipe Holistic Landmarker task lets you combine components of the pose, face, and hand landmarkers to create a complete landmarker for the human body. You can use this task to analyze full-body gestures, poses, and actions. This task uses a machine learning (ML) model on a continuous stream of images. The task outputs a total of 543 landmarks (33 pose landmarks, 468 face landmarks, and 21 hand landmarks per hand) in real-time.

5.2.2.2 Face landmarks detection

The MediaPipe Face Landmarker task lets you detect face landmarks and facial expressions in selfie-like images and videos. You can use this task to apply facial filters and effects to create a virtual avatar that mimics human facial expressions. This task uses machine learning (ML) models that can work with single images or a continuous stream of images. The task outputs an estimate of 468 3-dimensional face landmarks and 52 blendshape scores (coefficients

representing facial expression) to infer detailed facial surfaces in real-time.

5.2.2.3 Pose landmarks detection

The MediaPipe Pose Landmarker task lets you detect the landmarks of human bodies in an image. You can use this task to identify key body locations and render visual effects on them. This task uses machine learning (ML) models that can work with single images or a continuous stream of images. The task outputs body pose landmarks in image coordinates and in 3-dimensional (x,y,z) world coordinates.

5.2.2.4 Hand landmark detection

The hand landmark model bundle detects the keypoint localization of 21 hand-knuckle coordinates within the detected hand regions. The model was trained on approximately 30K real-world images, as well as several rendered synthetic hand models imposed over various backgrounds.

The hand landmarker model bundle contains palm detection model and hand landmarks detection model. Palm detection model localizes the region of hands from the whole input image, and the hand landmarks detection model finds the landmarks on the cropped hand image defined by the palm detection model.

Since palm detection model is much more time consuming, in Video mode or Live stream mode, Gesture Recognizer uses bounding box defined by the detected hand landmarks in the current frame to localize the region of hands in the next frame. This reduces the times Gesture Recognizer triggering palm detection model. Only when the hand landmarks model could no longer identify enough required number of hands presence, or the hand tracking fails, palm detection model is invoked to relocalize the hands.

5.2.3 Long Short Term Memory

Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on

the basis of time-series data. Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for tasks such as language translation, speech recognition, and time seris forecasting.

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period of time. The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell. The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

LSTMs can be stacked to create deep LSTM networks, which can learn even more complex

patterns in sequential data. LSTMs can also be used in combination with other neural network architectures, such as Convolutional Neural Networks (CNNs) for image and video analysis.

Fig. 5.4. Structure Of LSTM

LSTM has a chain structure that contains four neural networks and different memory blocks called cells. Information is retained by the cells and the memory manipulations are done by the gates. There are three gates –

1. Forget Gate: The information that is no longer useful in the cell state is removed with the forget gate. Two inputs x_t (input at the particular time) and h_t-1 (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for output 1, the information is retained for future use.

Fig. 5.5. Forgot Gate in LSTM

2. Input gate: The addition of useful information to the cell state is done by the input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs h_t-1 and x_t. Then, a vector is created using tanh function

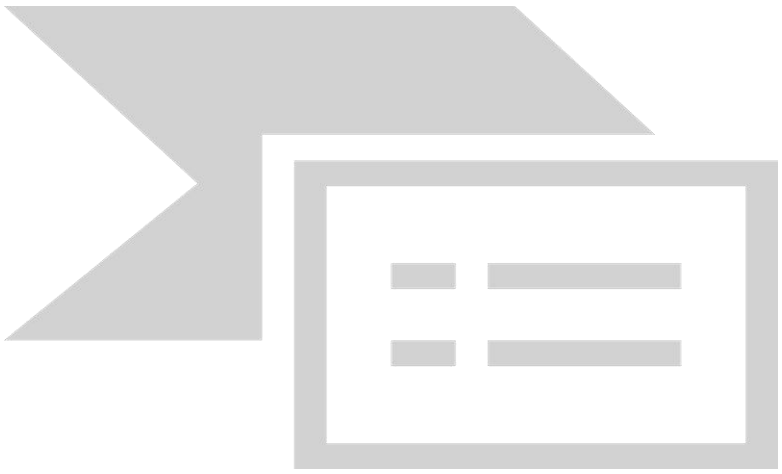


Fig. 5.6. Input Gate in LSTM

that gives an output from -1 to +1, which contains all the possible values from h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to obtain the useful information

3. Output gate: The task of extracting useful information from the current cell state to be presented as output is done by the output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter by the values to be remembered using inputs h_t-1 and x_t. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

Fig. 5.6. output Gate in LSTM

5.2.4 Natural language processing

Natural language processing (NLP) is an interdisciplinary subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

Challenges in natural language processing frequently involve speech recognition, natural-language understanding, and natural-language generation.

In the early days, many language-processing systems were designed by symbolic methods, i.e.,

the hand-coding of a set of rules, coupled with a dictionary lookup such as by writing grammars or devising heuristic rules for stemming.

More recent systems based on machine-learning algorithms have many advantages over hand-produced rules:

- The learning procedures used during machine learning automatically focus on the most common cases, whereas when writing rules by hand it is often not at all obvious where the effort should be directed.
- Automatic learning procedures can make use of statistical inference algorithms to produce models that are robust to unfamiliar input (e.g. containing words or structures that have not been seen before) and to erroneous input (e.g. with misspelled words or words accidentally omitted). Generally, handling such input gracefully with handwritten rules, or, more generally, creating systems of handwritten rules that make soft decisions, is extremely difficult, error-prone and time-consuming.
- Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on handwritten rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on handwritten rules, beyond which the systems become more and more unmanageable. However, creating more data to input to machine-learning systems simply requires a corresponding increase in the number of man-hours worked, generally without significant increases in the complexity of the annotation process.

Despite the popularity of machine learning in NLP research, symbolic methods are still (2020) commonly used:

- when the amount of training data is insufficient to successfully apply machine learning methods, e.g., for the machine translation of low-resource languages such as provided by the Apertium system,
- for preprocessing in NLP pipelines, e.g., tokenization, or
- for postprocessing and transforming the output of NLP pipelines, e.g., for knowledge extraction from syntactic parses.

5.2.4.1 Statistical methods

Since the so-called "statistical revolution" in the late 1980s and mid-1990s, much natural language processing research has relied heavily on machine learning. The machine-learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large *corpora* (the plural form of *corpus*, is a set of documents, possibly with human or computer annotations) of typical real-world examples.

Many different classes of machine-learning algorithms have been applied to natural-language-processing tasks. These algorithms take as input a large set of "features" that are generated from the input data. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to each input feature (complex-valued embeddings, and neural networks in general have also been proposed, for e.g. speech). Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.

Some of the earliest-used machine learning algorithms, such as decision trees, produced systems of hard if-then rules similar to existing hand-written rules. However, part-of-speech tagging introduced the use of hidden Markov models to natural language processing, and increasingly, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to the features making up the input data. The cache language models upon which many speech recognition systems now rely are examples of such statistical models. Such models are generally more robust when given unfamiliar input, especially input that contains errors (as is very common for real-world data), and produce more reliable results when integrated into a larger system comprising multiple subtasks.

Since the neural turn, statistical methods in NLP research have been largely replaced by neural networks. However, they continue to be relevant for contexts in which statistical interpretability and transparency is required.

5.2.4.2 Neural networks

A major drawback of statistical methods is that they require elaborate feature engineering. Since 2015, the field has thus largely abandoned statistical methods and shifted to neural networks for machine learning. Popular techniques include the use of word embeddings to capture semantic

properties of words, and an increase in end-to-end learning of a higher-level task (e.g., question answering) instead of relying on a pipeline of separate intermediate tasks (e.g., part-of-speech tagging and dependency parsing). In some areas, this shift has entailed substantial changes in how NLP systems are designed, such that deep neural network-based approaches may be viewed as a new paradigm distinct from statistical natural language processing. For instance, the term *neural machine translation* (NMT) emphasizes the fact that deep learning-based approaches to machine translation directly learn sequence-to-sequence transformations, obviating the need for intermediate steps such as word alignment and language modeling that was used in statistical machine The following is a list of some of the most commonly researched tasks in natural language processing. Some of these tasks have direct real-world applications, while others more commonly serve as subtasks that are used to aid in solving larger tasks.

5.2.4.3 Applications

Though natural language processing tasks are closely intertwined, they can be subdivided into categories for convenience. Optical character recognition (OCR) Given an image representing printed text, determine the corresponding text.

Speech recognition

Given a sound clip of a person or people speaking, determine the textual representation of the speech. This is the opposite of text to speech and is one of the extremely difficult problems colloquially termed "AI-complete" (see above). In natural speech there are hardly any pauses between successive words, and thus speech segmentation is a necessary subtask of speech recognition (see below). In most spoken languages, the sounds representing successive letters blend into each other in a process termed coarticulation, so the conversion of the analog signal to discrete characters can be a very difficult process. Also, given that words in the same language are spoken by people with different accents, the speech recognition software must be able to recognize the wide variety of input as being identical to each other in terms of its textual equivalent. Speech segmentation:Given a sound clip of a person or people speaking, separate it into words. A subtask of speech recognition and typically grouped with it. Text-to-speech:Given a text, transform those units and produce a spoken representation. Text-to-speech can be used to aid the visually impaired.

Word segmentation (Tokenization)

Separate a chunk of continuous text into separate words. For a language like English, this is fairly trivial, since words are usually separated by spaces. However, some written languages like Chinese, Japanese and Thai do not mark word boundaries in such a fashion, and in those languages text segmentation is a significant task requiring knowledge of the vocabulary and morphology of words in the language. Sometimes this process is also used in cases like bag of words (BOW) creation in data mining.

Morphological analysis

Lemmatization: The task of removing inflectional endings only and to return the base dictionary form of a word which is also known as a lemma. Lemmatization is another technique for reducing words to their normalized form. But in this case, the transformation actually uses a dictionary to map words to their actual form.

Morphological segmentation

Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology (*i.e.*, the structure of words) of the language being considered. English has fairly simple morphology, especially inflectional morphology, and thus it is often possible to ignore this task entirely and simply model all possible forms of a word (e.g., "open, opens, opened, opening") as separate words. In languages such as Turkish or Meitei, a highly agglutinated Indian language, however, such an approach is not possible, as each dictionary entry has thousands of possible word forms.

Part-of-speech tagging

Given a sentence, determine the part of speech (POS) for each word. Many words, especially common ones, can serve as multiple parts of speech. For example, "book" can be a noun ("the book on the table") or verb ("to book a flight"); "set" can be a noun, verb or adjective; and "out" can be any of at least five different parts of speech.

Stemming

The process of reducing inflected (or sometimes derived) words to a base form (e.g., "close" will be the root for "closed", "closing", "close", "closer" etc.). Stemming yields similar results as

lemmatization, but does so on grounds of rules, not a dictionary.

Syntactic analysis

Sentence breaking (also known as "sentence boundary disambiguation"): Given a chunk of text, find the sentence boundaries. Sentence boundaries are often marked by periods or other punctuation marks, but these same characters can serve other purposes (e.g., marking abbreviations).

Parsing

Determine the parse tree (grammatical analysis) of a given sentence. The grammar for natural languages is ambiguous and typical sentences have multiple possible analyses: perhaps surprisingly, for a typical sentence there may be thousands of potential parses (most of which will seem completely nonsensical to a human). There are two primary types of parsing: *dependency parsing* and *constituency parsing*. Dependency parsing focuses on the relationships between words in a sentence (marking things like primary objects and predicates), whereas constituency parsing focuses on building out the parse tree using a probabilistic context-free grammar (PCFG) (see also *stochastic grammar*).

Named entity recognition (NER)

Given a stream of text, determine which items in the text map to proper names, such as people or places, and what the type of each such name is (e.g. person, location, organization). Although capitalization can aid in recognizing named entities in languages such as English, this information cannot aid in determining the type of named entity, and in any case, is often inaccurate or insufficient. For example, the first letter of a sentence is also capitalized, and named entities often span several words, only some of which are capitalized. Furthermore, many other languages in non-Western scripts (e.g. Chinese or Arabic) do not have any capitalization at all, and even languages with capitalization may not consistently use it to distinguish names. For example, German capitalizes all nouns, regardless of whether they are names, and French and Spanish do not capitalize names that serve as adjectives.

Sentiment analysis (see also Multimodal sentiment analysis)

Extract subjective information usually from a set of documents, often using online reviews to determine "polarity" about specific objects. It is especially useful for identifying trends of public

opinion in social media, for marketing. Terminology extraction:The goal of terminology extraction is to automatically extract relevant terms from a given corpus.

Word-sense disambiguation (WSD)

Many words have more than one meaning; we have to select the meaning which makes the most sense in context. For this problem, we are typically given a list of words and associated word senses, e.g. from a dictionary or an online resource such as WordNet.

Entity linking

Many words—typically proper names—refer to named entities; here we have to select the entity (a famous individual, a location, a company, etc.) which is referred to in context.

5.2.5 Regular Expressions

A regular expression also known as regex or regexp sometimes referred to as rational expression) is a sequence of characters that specifies a match pattern in text. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. Regular expression techniques are developed in theoretical computer science and formal language theory. The concept of regular expressions began in the 1950s, when the American mathematician Stephen Cole Kleene formalized the concept of a regular language. They came into common use with Unix text-processing utilities. Different syntaxes for writing regular expressions have existed since the 1980s, one being the POSIX standard and another, widely used, being the Perl syntax. Regular expressions are used in search engines, in search and replace dialogs of word processors and text editors, in text processing utilities such as sed and AWK, and in lexical analysis. Regular expressions are supported in many programming languages.

5.2.5.1 Patterns

The phrase *regular expressions*, or *regexes*, is often used to mean the specific, standard textual syntax for representing patterns for matching text, as distinct from the mathematical notation described below. Each character in a regular expression (that is, each character in the string describing its pattern) is either a metacharacter, having a special meaning, or a regular character that has a literal meaning. For example, in the regex b., 'b' is a literal character that matches just 'b', while '.' is a metacharacter that matches every character except a newline. Therefore, this

regex matches, for example, 'b%', or 'bx', or 'b5'. Together, metacharacters and literal characters can be used to identify text of a given pattern or process a number of instances of it. Pattern matches may vary from a precise equality to a very general similarity, as controlled by the metacharacters. For example, . is a very general pattern, [a-z] (match all lower case letters from 'a' to 'z') is less general and b is a precise pattern (matches just 'b'). The metacharacter syntax is designed specifically to represent prescribed targets in a concise and flexible way to direct the automation of text processing of a variety of input data, in a form easy to type using a standard ASCII keyboard. A very simple case of a regular expression in this syntax is to locate a word spelled two different ways in a text editor, the regular expression seriali[sz]e matches both "serialise" and "serialize". Wildcard characters also achieve this, but are more limited in what they can pattern, as they have fewer metacharacters and a simple language-base. The usual context of wildcard characters is in globbing similar names in a list of files, whereas regexes are usually employed in applications that pattern-match text strings in general. For example, the regex ^[\t]+|[\t]+$ matches excess whitespace at the beginning or end of a line. An advanced regular expression that matches any numeral is [+-]?(\d+(\.\d*)?|\.\d+)([eE][+-]?\d+)?.

A regex processor translates a regular expression in the above syntax into an internal representation that can be executed and matched against a string representing the text being searched in. One possible approach is the Thompson's construction algorithm to construct a nondeterministic finite automaton (NFA), which is then made deterministic and the resulting deterministic finite automaton (DFA) is run on the target text string to recognize substrings that match the regular expression. The picture shows the NFA scheme $N(s^*)$ obtained from the regular expression s^*, where s denotes a simpler regular expression in turn, which has already been recursively translated to the NFA $N(s)$.

5.2.6 Web Real Time Communication (WebRTC)

WebRTC (Web Real-Time Communication) is a free and open-source project providing web browsers and mobile applications with real time communication (RTC) via application programming interfaces (APIs). It allows audio and video communication to work inside web pages by allowing direct peer-to-peer communication, eliminating the need to install plugins or download native apps.'Peer-to-peer' means, for example, two people having a video chat. Most network communication involves a client (for instance, you on a web browser on your computer) and a server (for instance, the website that you're viewing). With email, your computer's email

program (the client) contacts a SMTP (outgoing mail) server to send emails, and then an IMAP (incoming mail) server to retrieve emails that you received. In all of these cases, you and your computer are the client. But, if communication is peer-to-peer, that means that both ends work the same. It's almost as if each side is a client and a server to the other. Supported by Apple, Google, Microsoft, Mozilla, and Opera, WebRTC specifications have been published by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF). According to the webrtc.org website, the purpose of the project is to "enable rich, high-quality RTC applications to be developed for the browser, mobile platforms, and IoT devices, and allow them all to communicate via a common set of protocols".

5.2.6.1 Components

Major components of WebRTC include several JavaScript APIs:

- getUserMedia acquires the audio and video media (e.g., by accessing a device's camera and microphone).
- RTCPeerConnection enables audio and video communication between peers. It performs signal processing, codec handling, peer-to-peer communication, security, and bandwidth management.
- RTCDataChannel allows bidirectional communication of arbitrary data between peers. The data is transported using SCTP over DTLS. It uses the same API as WebSockets and has very low latency.

The WebRTC API also includes a statistics function:

- getStats allows the web application to retrieve a set of statistics about WebRTC sessions. These statistics data are being described in a separate W3C document.

The WebRTC API includes “no provisions for signaling,” that is discovering peers to connect to and determine how to establish connections among them. Applications use Interactive Connectivity Establishment for connections and are responsible for managing sessions, possibly relying on any of Session Initiation Protocol, Extensible Messaging and Presence Protocol, Message Queuing Telemetry Transport, Matrix, or another protocol. Signaling may depend on one or more servers. RFC 7478 requires implementations to provide PCMA/PCMU (RFC 3551), Telephone Event as DTMF (RFC 4733),

and Opus (RFC 6716) audio codecs as minimum capabilities. The PeerConnection, data channel and media capture browser APIs are detailed in the W3C specification. W3C is developing ORTC (Object Real-Time Communications) for WebRTC.

WebRTC allows browsers to stream files directly to one another, reducing or entirely removing the need for server-side file hosting. WebTorrent uses a WebRTC transport to enable peer-to-peer file sharing using the BitTorrent protocol in the browser. Some file-sharing websites use it to allow users to send files directly to one another in their browsers, although this requires the uploader to keep the tab open until the file has been downloaded. A few CDNs, such as the Microsoft-owned Peer5, use the client's bandwidth to upload media to other connected peers, enabling each peer to act as an edge server. Although initially developed for web browsers, WebRTC has applications for non-browser devices, including mobile platforms and IoT devices. Examples include browser-based VoIP telephony, also called cloud phones or web phones, which allow calls to be made and received from within a web browser, replacing the requirement to download and install a softphone.

5.2.7 Application Programming Interface(API)

An application programming interface **(**API**)** is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software. A document or standard that describes how to build or use such a connection or interface is called an *API specification*. A computer system that meets this standard is said to *implement* or *expose* an API. The term API may refer either to the specification or to the implementation.

In contrast to a user interface, which connects a computer to a person, an application programming interface connects computers or pieces of software to each other. It is not intended to be used directly by a person (the end user) other than a computer programmer who is incorporating it into the software. An API is often made up of different parts which act as tools or services that are available to the programmer. A program or a programmer that uses one of these parts is said to *call* that portion of the API. The calls that make up the API are also known as subroutines, methods, requests, or endpoints. An API specification *defines* these calls, meaning that it explains how to use or implement them.

One purpose of APIs is to hide the internal details of how a system works, exposing only those parts a programmer will find useful and keeping them consistent even if the internal details later

change. An API may be custom-built for a particular pair of systems, or it may be a shared standard allowing interoperability among many systems.

There are APIs for programming languages, software libraries, computer operating systems, and computer hardware. APIs originated in the 1940s, though the term did not emerge until the 1960s and 1970s. Contemporary usage of the term API often refers to web APIs, which allow communication between computers that are joined by the internet. Recent developments in APIs have led to the rise in popularity of microservices, which are loosely coupled services accessed through public APIs.

CHAPTER 6

RESULTS AND DISCUSSION

This section gives a brief discussion of the results of the experiments and analyses that are conducted on the proposed system. Sign language translation requires processing the video frames into sequence data of 30 frames in a video for each gesture captured by the system as one data point in the dataset. Thirty frames of each gesture data point have the gesture data in the form of positional coordinates used by MediaPipe, representing the gesture holistic of one frame. The sequence of frames creates motion of skeletal structure, making it a motion gesture detected by the MediaPipe Framework. These data points are ASL sign gestures that are mapped to words. Based on the vocabulary size, LSTM architecture is built to process the 30 frames of sequence gesture of live stream video data and detect the motion.

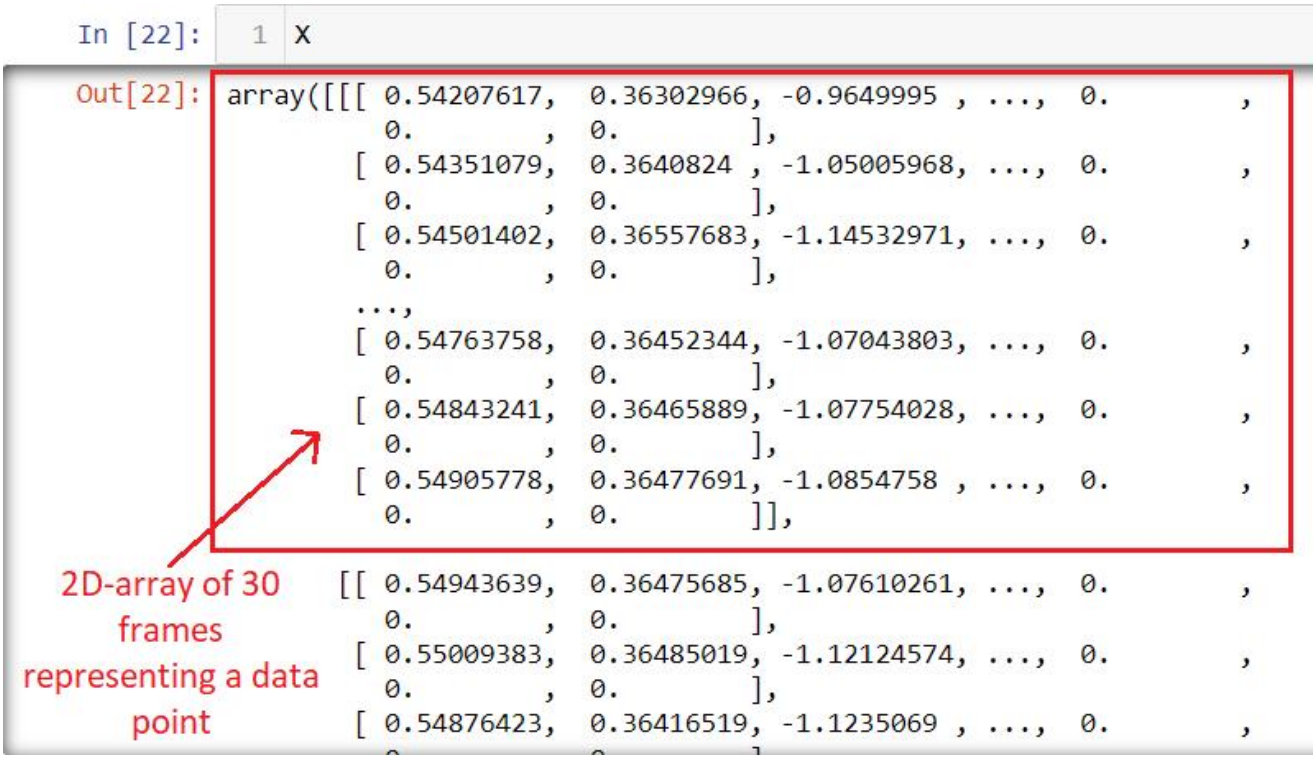
6.1 CONVERSION OF VIDEO FRAMES TO SEQUENCE DATA

Using OpenCV, the streamed data is pipelined to the application where gesture skeletal structure data is detected from video frames with the help of MediaPipe Framework. MediaPipe [18] uses ML and DL models that detect the skeletal structure in real-time and return collection arrays of each frame as output, considered one point in the ASL dataset of motion gestures. The ASL dataset for training ML models is built using the OpenCV and MediaPipe framework. Each word in the vocabulary is a gesture in ASL performed in front of a digital camera and pipelined to a data collection module that converts video frames to sequence data. The converted sequence data of each frame in 30 frames in motion gesture is stored in a folder where; the folder is considered as one data point representing the word. An independent executable file (.exe) is developed based on the python language and makes use of the 'auto-py-to-exe' library. It can run on any windows machine without installing any python libraries or frameworks. By making this system open-source, users can train their sign gestures with data collector executable files to work according to the ASL, allowing for a vast vocabulary. The ASL dataset is used to experiment with the LSTM architecture and parameters of Neural Networks to develop a reliable SLR system.

6.2 DATA PRE-PROCESSING

Data returned from the MediaPipe model will be in the form of an object variable. The pose, face, left hand, and right hand (PFLR)landmarks are extracted from MediaPipe objects as x, y, z, and

visibility variables. The visibility variable is for only pose landmarks. Extracted landmarks of PFLR have variables in a two-dimensional (2D) array that are flattened in each category and arranged as follows: pose, face, left hand, and right hand. So, every frame in the one motion gesture data



point consists of a PFLR sequence of the 1D array where 30 frames give us 30 arrays stored in one folder in the local drive, making it one data point.

Fig. 6.1. Array representation training dataset

ASL datasets stored in folders are retrieved and combined as a 2D array from the file manager to make a data structure to feed the DGR model for training, as shown in figure 6.1. Based on the label of the top folder, the training labels are generated for neural network training.

6.3 BUILDING THE LSTM ARCHITECTURAL NEURAL NETWORK

ASL dataset used to train the DGR model comprises six words, "HELLO," "EAT," "THANK YOU," "EXCUSE ME," "HELP," and "PLEASE" along with the word "NONE," also added to the vocabulary. The word "NONE" means the user does not perform a sign gesture. The dataset is developed with the MediaPipe, OpenCV, and other libraries. MediaPipe uses built-in libraries to extract the landmarks of the gesture data. DGR model architecture comprises six layers where the first three layers are LSTM layers, and the remaining three layers are fully connected layers. DGR model takes 30 frames of gesture data as input and predicts seven words as output,

including the word "None." The detailed structure of the DGR model is shown in figure 6.2.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            442112

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 7)                 231

=================================================================
Total params: 596,807
Trainable params: 596,807
Non-trainable params: 0
_________________________________________________________________
```

Fig. 6.2. DGR Model Architecture with LSTM layers

The presented training and testing results of LSTM architecture are based on the 360 data points of the dataset given for training the LSTM neural network, consisting of the four words mentioned above. As shown in figure 4.4, various hyperparameters of LSTM architecture are experimented with and tested to develop the model, which performs an accuracy of 98.81%.

```
optimizer=Adam(decay=1e-4)
# optimizer=SGD(momentum=0.1)
early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=20, restore_best_weights=True)#monitor=val_acc
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
```

```
checkpoints=ModelCheckpoint('Logs\\checkpoints\\' + 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
                            monitor='val_loss', save_weights_only=True, save_best_only=True)#, period=3)
```

```
model.fit(X, y, batch_size=30, epochs=1000, validation_data=(X_test, y_test), shuffle=True,callbacks=[early_stopping,checkpo
```

```
Epoch 22/1000
42/42 [==============================] - 4s 87ms/step - loss: 0.0575 - acc: 0.9841 - val_loss: 0.0548 - val_acc: 0.9841
Epoch 23/1000
42/42 [==============================] - 4s 85ms/step - loss: 0.0854 - acc: 0.9706 - val_loss: 0.0827 - val_acc: 0.9683
Epoch 24/1000
42/42 [==============================] - 4s 85ms/step - loss: 0.0791 - acc: 0.9722 - val_loss: 0.0798 - val_acc: 0.9603
Epoch 25/1000
42/42 [==============================] - 4s 85ms/step - loss: 0.0810 - acc: 0.9770 - val_loss: 0.0760 - val_acc: 0.9802
Epoch 26/1000
42/42 [==============================] - 4s 86ms/step - loss: 0.0673 - acc: 0.9738 - val_loss: 0.0554 - val_acc: 0.9881
Epoch 27/1000
42/42 [==============================] - 4s 87ms/step - loss: 0.0612 - acc: 0.9794 - val_loss: 0.0930 - val_acc: 0.9722
Epoch 28/1000
42/42 [==============================] - 4s 87ms/step - loss: 0.0644 - acc: 0.9825 - val_loss: 0.0448 - val_acc: 0.9921
Epoch 29/1000
42/42 [==============================] - 4s 92ms/step - loss: 0.0526 - acc: 0.9817 - val_loss: 0.0352 - val_acc: 0.9921
Epoch 30/1000
42/42 [==============================] - 4s 95ms/step - loss: 0.0433 - acc: 0.9889 - val_loss: 0.0381 - val_acc: 0.9921

<keras.callbacks.History at 0x17518829eb0>
```

Fig. 6.3. Training results of the DGR Model

6.4 SIGN GESTURE PREDICTION AND RESULTS

MediaPipe Framework extracts the gesture data from real-time video stream data from a digital camera pipelined with OpenCV to the application. It captures the pose, face, left-hand, and right-hand coordinates from each video frame and stores them as sequence data. This gesture data is continuously given to the DGR model, which considers 30 frames of data for the Prediction of a single gesture. Each frame sequence consists of pose landmarks: 132, face landmarks: 1404, left-hand landmarks: 63, and right-hand: 63 points, giving 1662 points of sequence data in the whole data point. The results DGR model with a confusion matrix, accuracy, and loss are shown in figure 6.4.

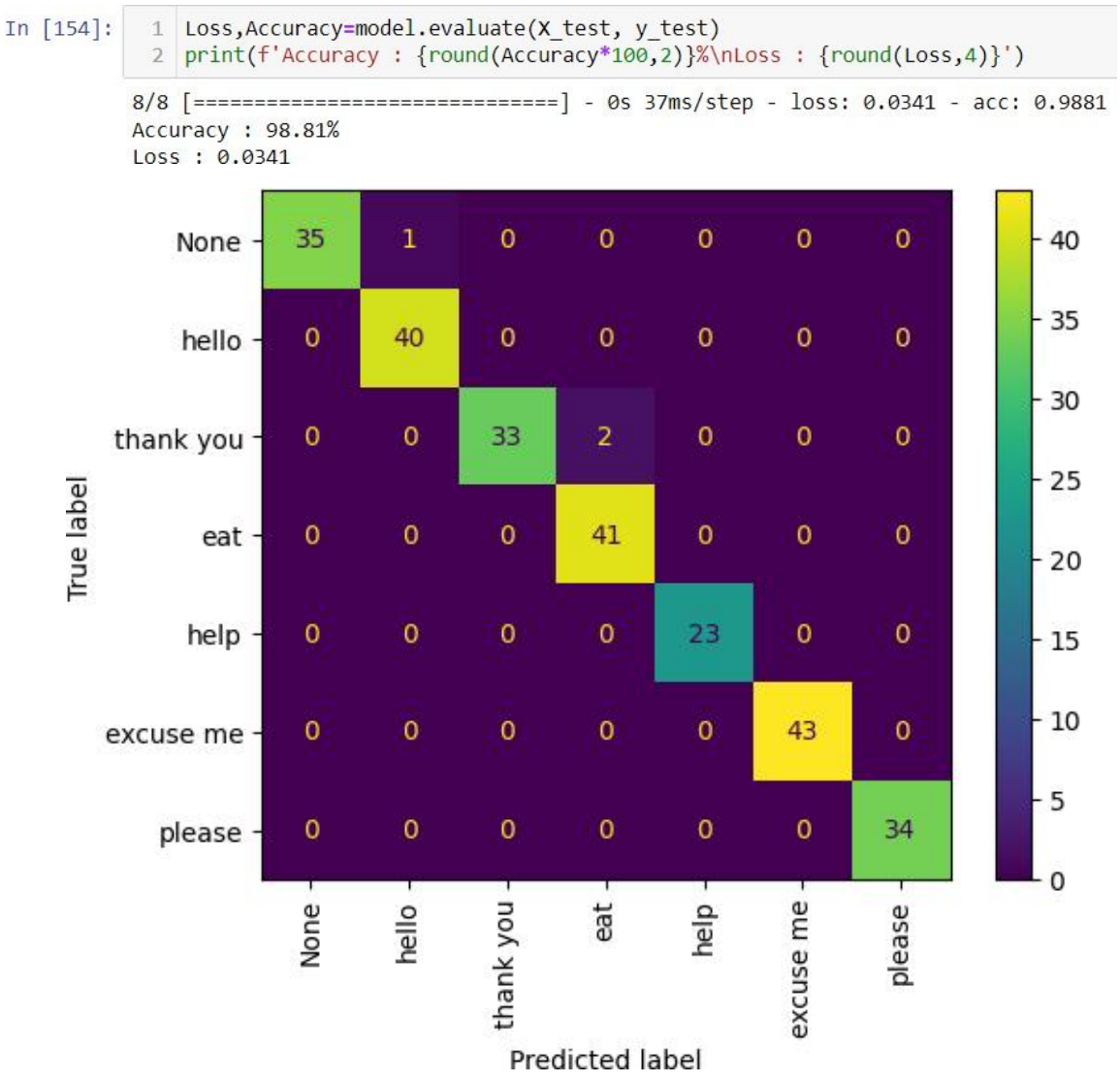


Fig. 6.4. Confusion Matrix, Accuracy and Loss of the DGR Model

DGR model with LSTM architecture predicts the gesture with the given data after pre-processing and conversion to data structure from live video data in the above-mentioned sequence. Each gesture is predicted based on the probability of the gesture performed at every sequence of 30 frames. The word is predicted based on the highest probability in vocabulary, and that word crosses the threshold probability of 0.9. Each predicted word is given to the display output, as shown in figure 6.5.

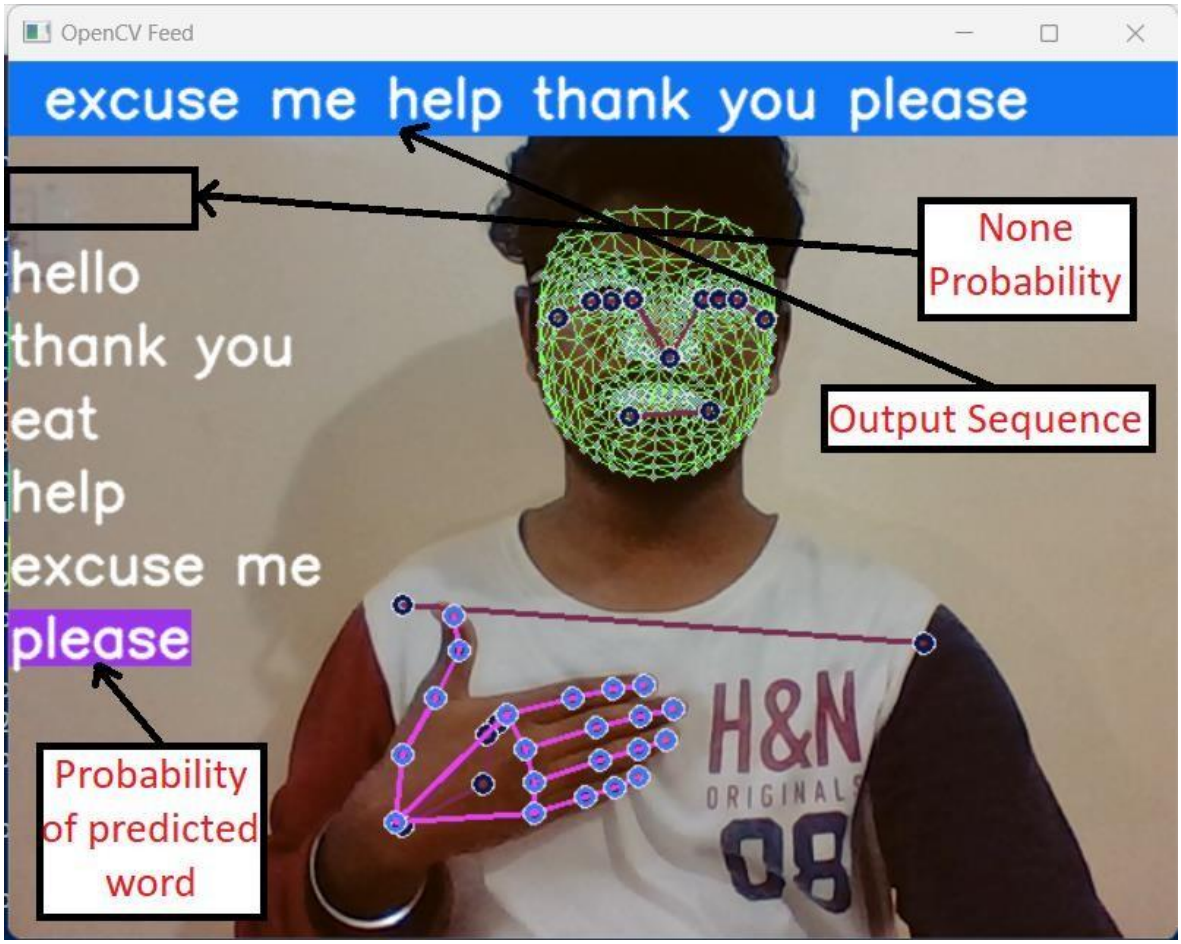


Fig. 6.5. Prediction of ASL with the DGR model

CHAPTER 7

CONCLUSION

Sign language has been the primary way of communication for hearing and speech-impaired people. Most people aren't aware of sign language, especially American sign language. Common people often find it difficult to understand disabled people. Sign Language Translation Application (SLTA) aims to bridge the communication barrier between disabled and abled persons using sign language recognition models. SLTA captures signed gestures with a digital camera and extracts the gesture data using the MediaPipe Framework. With the help of the DGR model developed with LSTM architecture, it can predict the dynamic motion gestures performed by the person in Infront of the camera in real-time. An Accuracy of 98.81% has been achieved with seven words of vocabulary and 252 test samples. The proposed system also overcomes the existing problems of static gesture recognition using LSTM architecture by predicting motion gestures and replacing the usage of sensor-based gloves and wearables for gesture motion data collection.

Using WebRTC protocols, this application can be implemented as a video conferencing for speech and hearing-impaired people. Further development and refinement can be done on this system by including the expansion of the size of ASL vocabulary, development of the NLP model for predicted words to sentences, and implementation of WebRTC protocol for video conferencing for communication can be considered for future work.

REFERENCES

[1] J. Hou, X.-Y. Li, P. Zhu, Z. Wang, Y. Wang, J. Qian, and P. Yang, "Signspeaker: A real-time, high-precision smartwatch-based sign language translator," in Proc. of ACM MobiCom, 2019.

[2] J. Huang, W. Zhou, Q. Zhang, H. Li, and W. Li, "Video-based sign language recognition without temporal segmentation," arXiv preprint arXiv:1801.10111, 2018.

[3] J. Wu, Z. Tian, L. Sun, L. Estevez, and R. Jafari, "Real-time American sign language recognition using wrist-worn motion and surface emg sensors," in Proc. of IEEE BSN, 2015, pp. 1–6.

[4] J. Zang, L. Wang, Z. Liu, Q. Zhang, G. Hua, and N. Zheng, "Attention-based temporal weighted convolutional neural network for action recognition," in Proc. of IFIP INTERACT, 2018, pp. 97–108.

[5] J. Zhang, J. Tao, and Z. Shi, "Doppler-radar based hand gesture recognition system using convolutional neural networks," in International Conference in Communications, Signal Processing, and Systems. Springer, 2017, pp. 1096–1113.

[6] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "Fingerio: Using active sonar for f ine-grained finger tracking," in Proc. of ACM CHI, 2016, pp. 1515–1525.

[7] Julian Menezes .R, Albert Mayan .J, M. Breezely George," Development of a Functionality Testing Tool for Windows Phones", Indian Journal of Science and Technology,Vol:8,Issue:22,pp: 1-7,September 2015.

[8] T. T. Swee, A. Ariff, S.-H. Salleh, S. K. Seng, and L. S. Huat, "Wireless data gloves malay sign language recognition system," in Information, Communications & Signal Processing, 2007 6th International Conference on. IEEE, 2007, pp. 1–4.

[9] T . Zhao, J. Liu, Y. Wang, H. Liu, and Y. Chen, "Ppg-based finger level gesture recognition leveraging wearables," in Proc. of IEEE INFOCOM, 2018, pp. 1457–1465.

[10] X. Zhang, X. Chen, Y. Li, V. Lantz, K. Wang, and J. Yang, "A framework for hand gesture recognition based on accelerometer and emg sensors," IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, vol. 41, no. 6, pp. 1064–1076, 2011.

[11] Z. Lu, X. Chen, Q. Li, X. Zhang, and P. Zhou, "A Hand Gesture Recognition Framework and Wearable Gesture-Based Interaction Prototype for Mobile Devices," in IEEE Transactions on Human-Machine Systems, vol. 44, no. 2, pp. 293-299, April 2014, doi: 10.1109/THMS.2014.2302794.

[12] Bhokse, B. (January 1, 2015). ISSN 2348 – 7968 hand gesture recognition using a neural network - IJISET. IJISET - International Journal of Innovative Science, Engineering & Technology. Retrieved October 24, 2022, from https://www.ijiset.com/vol2/v2s1/IJISET_V2_I1_01.pdf

[13] Coster(UGent), M. D., & Dambre(UGent), and J. (1970, January 1). Sign language recognition with Transformer Networks. Sign language recognition with transformer networks. Retrieved October 24, 2022, from http://hdl.handle.net/1854/LU-8660743

[14] K. Kudrinko, E. Flavin, X. Zhu, and Q. Li, "Wearable Sensor-Based Sign Language Recognition: A Comprehensive Review," in IEEE Reviews in Biomedical Engineering, vol. 14, pp. 82-97, 2021, doi: 10.1109/RBME.2020.3019769.

[15] Asha Pandian, Bharathi B, Albert Mayan J, Prem Jacob, Pravin "A Comprehensive View of Scheduling Algorithms for MapReduce Framework in Hadoop," Journal of Computational and Theoretical Nanoscience, Vol.16, No. 8, pp. 3582-3586, 2019

[16] Mitra, S. (2007, May 3). GESTURE RECOGNITION: A survey. IEEE Xplore. Retrieved October 24, 2022, from https://ieeexplore.ieee.org/document/4154947

[17] Razieh Rastgoo, Kourosh Kiani, Sergio Escalera, Sign Language Recognition: A Deep Survey, Expert Systems with Applications, Volume 164, 2021, 113794, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2020.113794. (https://www.sciencedirect.com/science/article/pii/S095741742030614X)

[18] Lugaresi, Camillo. "MediaPipe: A Framework for Building Perception Pipelines." arXiv.org, June 14, 2019. https://arxiv.org/abs/1906.08172.

[19] Wang, Z., Zhao, T., Ma, J., Chen, H., Liu, K., Shao, H., Wang, Q., & Ren, J. (2020). Hear sign language: A real-time end-to-end sign language recognition system. IEEE Transactions on Mobile Computing, 1-1. https://doi.org/10.1109/tmc.2020.3038303

[20] Wikimedia Foundation. (2022, October 23). American sign language. Wikipedia. Retrieved October 24, 2022, from https://en.wikipedia.org/wiki/American_Sign_Language

[21] Y. Suresh, J. Vaishnavi, M. Vindhya, M. S. A. Meeran and S. Vemala, "MUDRAKSHARA - A Voice for Deaf/Dumb People," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-8, doi: 10.1109/ICCCNT49239.2020.9225656.

[22] Bazarevsky, V., & Zhang, F. (2019, August 19). Hands. On-Device, Real-Time Hand Tracking with MediaPipe. Retrieved February 10, 2023, from https://google.github.io/mediapipe/solutions/hands.html

[23] Bazarevsky, V., & Grishchenko, I. (2020, August 13). Pose. On-device, Real-time Body Pose Tracking with MediaPipe BlazePose. Retrieved February 10, 2023, from https://google.github.io/mediapipe/solutions/pose.html

[24] Teak-Wei , C., & Boon Giin, L. (2018, October). The 26 letters and 10 digits of American Sign Language (ASL). American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach. Retrieved February 10, 2023, from https://www.researchgate.net/figure/The-26-letters-and-10-digits-of-American-Sign-Language-ASL_fig1_328396430

A. SOURCE CODE

LSTM_MODEL_TRAINING_.py

```
#!/usr/bin/env python
# coding: utf-8

# # Import and Install Dependencies
!pip install tensorflow==2.4.1 tensorflow-gpu==2.4.1 opencv-python mediapipe sklearn matplotlib
# In[1]:


import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
from math import exp
import mediapipe as mp
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score,ConfusionMatrixDisplay,log_loss


# In[2]:


from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop, SGD, Adam
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler, ModelCheckpoint
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,Dropout,Embedding, SpatialDropout1D
from tensorflow.keras.callbacks import TensorBoard
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical


# In[3]:


initial_learning_rate=0.1
batch_size=1
path="C:\\Users\\ganga\Jupyter Notebook\\FINAL YEAR PROJECT\\"


# # Keypoints using MP Holistic

# In[4]:


mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
```

```
# In[5]:


def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results


# In[6]:


def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION) # Draw
face connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # Draw pose
connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw
left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw
right hand connections

# In[7]:


def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION,
                              mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                              )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )


# # Extract Keypoint Values

# In[8]:


def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
    print(f'Pose: {pose}')
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if
results.face_landmarks else np.zeros(468*3)
    print(f'Face: {face}')
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
    print(f'Left Hand: {lh}')
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten() if
results.right_hand_landmarks else np.zeros(21*3)
    print(f'Right Hand: {rh}')
    return np.concatenate([pose, face, lh, rh])
```

```
# # Setup Folders

# In[9]:


# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data')

# Actions that we try to detect
actions = np.array(['','hello', 'thank you', 'eat','help', 'excuse me', 'please'])

# Thirty videos worth of data
no_sequences = 30

# Videos are going to be 30 frames in length
sequence_length = 30

# Folder start
start_folder = 30
label_map = {label:num for num, label in enumerate(actions)}
label_map

# # Preprocess Data and Create Labels and Features

# In[19]:


sequences, labels = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences.append(window)
        labels.append(label_map[action])


# In[20]:
np.array(os.listdir(os.path.join(DATA_PATH, action)))
np.array(sequences).shape
np.array(labels).shape
X = np.array(sequences)
X.shape
y = to_categorical(labels).astype(int)
y[:5]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,shuffle=True)
y_test.shape


# # Build and Train LSTM Neural Network

# In[47]:


log_dir = os.path.join('Logs')
tb_callback = TensorBoard(log_dir=log_dir)


# In[48]:


model = Sequential()
model.add(LSTM(64, return_sequences=True, activation='relu', input_shape=
(X.shape[1],X.shape[2])))#kernel_regularizer=regularizers.l2(0.005),
model.add(LSTM(128, return_sequences=True,
activation='relu'))#kernel_regularizer=regularizers.l2(0.005),
model.add(LSTM(64, return_sequences=False,
activation='relu'))#kernel_regularizer=regularizers.l2(0.005),
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(actions.shape[0], activation='softmax'))
model.summary()
```

```
# ## 1

# In[49]:


model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['categorical_accuracy'])


# In[63]:


model.fit(X_train, y_train, epochs=1, callbacks=[tb_callback],shuffle=True)


# ## 2
model.compile(optimizer=Adam(learning_rate=initial_learning_rate, clipvalue=2.0),
loss='categorical_crossentropy', metrics=['accuracy'])
early_stopping = EarlyStopping(monitor='val_accuracy', mode='max', patience=10,
restore_best_weights=True)#patient=10import math
steps_per_epoch = int(len(X_train/ batch_size))
print(steps_per_epoch)
def step_decay(epoch, learning_rate):
    drop = 0.5
    epochs_drop = 10.0
    lrate = learning_rate * math.pow(drop, math.floor((1+epoch)/epochs_drop))
    return lrate
# def step_decay(epoch):
#     initial_lrate = 0.1
#     drop = 0.5
#     epochs_drop = 10.0
#     lrate = initial_lrate * math.pow(drop,math.floor((1+epoch)/epochs_drop))
#     return lrate
def exp_decay(epoch):
    initial_lrate = 0.01
    k = 0.1
    lrate = initial_lrate * exp(-k*epoch)
    return lrate
# learning_rate_decay = LearningRateScheduler(step_decay, verbose=1)
learning_rate_decay = LearningRateScheduler(exp_decay, verbose=1)datagen = ImageDataGenerator(
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,shear_range = 0.2)# model.fit(X_train, y_train, epochs=500, callbacks=
[tb_callback],shuffle=True)
history =
model.fit(X_train,y_train,batch_size=batch_size,validation_split=0.1,epochs=100,use_multiprocessing=Fal
se,callbacks=[early_stopping, learning_rate_decay],steps_per_epoch=steps_per_epoch)
# ## 3

# In[128]:


optimizer=Adam(decay=1e-4)
# optimizer=SGD(momentum=0.1)
early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=20,
restore_best_weights=True)#monitor=val_acc
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])


# In[129]:


checkpoints=ModelCheckpoint('Logs\\checkpoints\\' + 'ep{epoch:03d}-loss{loss:.3f}-
val_loss{val_loss:.3f}.h5',
                            monitor='val_loss', save_weights_only=True, save_best_only=True)#,
period=3)


# In[130]:


model.fit(X, y, batch_size=30, epochs=1000, validation_data=(X_test, y_test), shuffle=True,callbacks=
[early_stopping,checkpoints])
```

```
# # Evaluation using Confusion Matrix and Accuracy

# In[131]:


ypred = model.predict(X_test)
ypred
ytrue = np.argmax(y_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
multilabel_confusion_matrix(ytrue, ypred)
[label_map.keys()]
cmp=ConfusionMatrixDisplay.from_predictions(ytrue, ypred,display_labels=['None','hello', 'thank you',
'eat','help', 'excuse me', 'please'],xticks_rotation='vertical')
Loss,Accuracy=model.evaluate(X_test, y_test)
print(f'Accuracy : {round(Accuracy*100,2)}%\nLoss : {round(Loss,4)}')


# # Make Predictions

# In[138]:

res = model.predict(X_test)
actions[np.argmax(res[4])]
actions[np.argmax(y_test[4])]

# # Save Weights

model.save('action.h5')

from tensorflow.keras.models import load_model
model = load_model("action.h5")
# In[44]:


test_DATA_PATH='MP_Data_Pl_HL_EM_GD'
sequences_test, labels_test = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(test_DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(test_DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences_test.append(window)
        labels_test.append(label_map[action])
sequences_test=np.array(sequences_test)
labels_test = to_categorical(labels_test).astype(int)
ypred = model.predict(sequences_test)
ytrue = np.argmax(labels_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
accuracy_score(ytrue, ypred)


# In[44]:


test_DATA_PATH='MP_Data_Pl_HL_EM_GD_2'
sequences_test, labels_test = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(test_DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(test_DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences_test.append(window)
        labels_test.append(label_map[action])
sequences_test=np.array(sequences_test)
labels_test = to_categorical(labels_test).astype(int)
ypred = model.predict(sequences_test)
ytrue = np.argmax(labels_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
accuracy_score(ytrue, ypred)
```

```
# In[45]:


test_DATA_PATH='MP_Data_Pl_HL_EM_UD'
sequences_test, labels_test = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(test_DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(test_DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences_test.append(window)
        labels_test.append(label_map[action])
sequences_test=np.array(sequences_test)
labels_test = to_categorical(labels_test).astype(int)
ypred = model.predict(sequences_test)
ytrue = np.argmax(labels_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
accuracy_score(ytrue, ypred)


# In[46]:


test_DATA_PATH='MP_Data_Pl_HL_EM_UD_2'
sequences_test, labels_test = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(test_DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(test_DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences_test.append(window)
        labels_test.append(label_map[action])
sequences_test=np.array(sequences_test)
labels_test = to_categorical(labels_test).astype(int)
ypred = model.predict(sequences_test)
ytrue = np.argmax(labels_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
accuracy_score(ytrue, ypred)


# In[84]:


test_DATA_PATH='MP_Data_Pl_HL_EM_KUN'
sequences_test, labels_test = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(test_DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(test_DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences_test.append(window)
        labels_test.append(label_map[action])
sequences_test=np.array(sequences_test)
labels_test = to_categorical(labels_test).astype(int)
ypred = model.predict(sequences_test)
ytrue = np.argmax(labels_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
accuracy_score(ytrue, ypred)
```

```
# In[85]:


test_DATA_PATH='MP_Data_Pl_HL_EM_KUN_2'
sequences_test, labels_test = [], []
for action in actions:
    for sequence in np.array(os.listdir(os.path.join(test_DATA_PATH, action if action else 'None'))):
        window = []
        for frame_num in range(sequence_length):
            res = np.load(os.path.join(test_DATA_PATH, action if action else 'None', str(sequence), "
{}.npy".format(frame_num)))
            window.append(res)
        sequences_test.append(window)
        labels_test.append(label_map[action])
sequences_test=np.array(sequences_test)
labels_test = to_categorical(labels_test).astype(int)
ypred = model.predict(sequences_test)
ytrue = np.argmax(labels_test, axis=1).tolist()
ypred = np.argmax(ypred, axis=1).tolist()
accuracy_score(ytrue, ypred)


# # Test in Real Time

# In[10]:


from tensorflow.keras.models import load_model
model = load_model("SltaModV3.h5")
model.summary()


# In[11]:


from scipy import stats


# In[12]:


colors = [(255, 0, 0),(245,117,16), (117,245,16), (16,117,245),(149, 235, 52),(52, 235, 162),(235, 52,
156)]
def prob_viz(res, actions, input_frame, colors):
    output_frame = input_frame.copy()
    for num, prob in enumerate(res):
        cv2.rectangle(output_frame, (0,60+num*40), (int(prob*100), 90+num*40), colors[num], -1)
        cv2.putText(output_frame, actions[num], (0, 85+num*40), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,255,255), 2, cv2.LINE_AA)

    return output_frame
```

```
# In[ ]:


# 1. New detection variables
sequence = []
sentence = []
predictions = []
threshold = 0.5

cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
#         print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)

        # 2. Prediction logic
        keypoints = extract_keypoints(results)
        sequence.append(keypoints)
        sequence = sequence[-30:]

        if len(sequence) == 30:
            res = model.predict(np.expand_dims(sequence, axis=0))[0]
            print(actions[np.argmax(res)])
            predictions.append(np.argmax(res))


        #3. Viz logic
            if np.unique(predictions[-10:])[0]==np.argmax(res):
                if res[np.argmax(res)] > threshold:

                    if len(sentence) > 0:
                        if actions[np.argmax(res)] != sentence[-1]:
                            sentence.append(actions[np.argmax(res)])
                    else:
                        sentence.append(actions[np.argmax(res)])

            if len(sentence) > 5:
                sentence = sentence[-5:]

            # Viz probabilities
            image = prob_viz(res, actions, image, colors)

        cv2.rectangle(image, (0,0), (640, 40), (245, 117, 16), -1)
        cv2.putText(image, ' '.join(sentence), (3,30),
                       cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)

        # Show to screen
        cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()


# In[48]:


plt.figure(figsize=(18,18))
plt.imshow(prob_viz(res, actions, image, colors))
```

Data Collector.py

```
print("Loading Packages and Libraries....")
import cv2
import numpy as np
import os
import time
import mediapipe as mp
import datetime
print("Model Initialization....")
mp_holistic = mp.solutions.holistic # Holistic model
mp_drawing = mp.solutions.drawing_utils # Drawing utilities
print("Loading MediaPipe Modules....")
def mediapipe_detection(image, model):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) # COLOR CONVERSION BGR 2 RGB
    image.flags.writeable = False                  # Image is no longer writeable
    results = model.process(image)                 # Make prediction
    image.flags.writeable = True                   # Image is now writeable
    image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR) # COLOR COVERSION RGB 2 BGR
    return image, results
def draw_landmarks(image, results):
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION) # Draw
face connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS) # Draw pose
connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw
left hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS) # Draw
right hand connections
def draw_styled_landmarks(image, results):
    # Draw face connections
    mp_drawing.draw_landmarks(image, results.face_landmarks, mp_holistic.FACEMESH_TESSELATION,
                              mp_drawing.DrawingSpec(color=(80,110,10), thickness=1, circle_radius=1),
                              mp_drawing.DrawingSpec(color=(80,256,121), thickness=1, circle_radius=1)
                              )
    # Draw pose connections
    mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_holistic.POSE_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(80,22,10), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(80,44,121), thickness=2, circle_radius=2)
                              )
    # Draw left hand connections
    mp_drawing.draw_landmarks(image, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(121,22,76), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(121,44,250), thickness=2, circle_radius=2)
                              )
    # Draw right hand connections
    mp_drawing.draw_landmarks(image, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                              mp_drawing.DrawingSpec(color=(245,117,66), thickness=2, circle_radius=4),
                              mp_drawing.DrawingSpec(color=(245,66,230), thickness=2, circle_radius=2)
                              )
def extract_keypoints(results):
    pose = np.array([[res.x, res.y, res.z, res.visibility] for res in
results.pose_landmarks.landmark]).flatten() if results.pose_landmarks else np.zeros(33*4)
    face = np.array([[res.x, res.y, res.z] for res in results.face_landmarks.landmark]).flatten() if
results.face_landmarks else np.zeros(468*3)
    lh = np.array([[res.x, res.y, res.z] for res in results.left_hand_landmarks.landmark]).flatten() if
results.left_hand_landmarks else np.zeros(21*3)
    rh = np.array([[res.x, res.y, res.z] for res in results.right_hand_landmarks.landmark]).flatten()
if results.right_hand_landmarks else np.zeros(21*3)
    return np.concatenate([pose, face, lh, rh])
```

```
print("<--------Creating Folders------->")
# Path for exported data, numpy arrays
DATA_PATH = os.path.join('MP_Data_new')

# Actions that we try to detect
actions = np.array(['hello', 'thank you', 'eat'])

# Thirty videos worth of data
no_sequences = 31

# Videos are going to be 30 frames in length
sequence_length = 30

# Folder start
for action in actions:
    print(f"\t> Creating Data for {action}--->")
    for sequence in range(0,no_sequences):
        print(f"\t\t> Generating Data Sequence {sequence}")
        try:
            current_time = str(datetime.datetime.now()).replace('-','').replace(':','').replace('
','').replace('.','')
            os.makedirs(os.path.join(DATA_PATH, action, current_time))
            time.sleep(0.1)
        except:
            pass

print("<---Adjust Postion--->")
cap = cv2.VideoCapture(0)
# Set mediapipe model
with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
    while cap.isOpened():

        # Read feed
        ret, frame = cap.read()

        # Make detections
        image, results = mediapipe_detection(frame, holistic)
#         print(results)

        # Draw landmarks
        draw_styled_landmarks(image, results)
        cv2.putText(image, 'Adjust Postion', (15,12),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
        # Show to screen
        cv2.imshow('OpenCV Feed', image)

        # Break gracefully
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()
```

```
print("<-|-|-|-|-|-|-|-|-|-|-|-|-|DATA COLLECTION INITIATION|-|-|-|-|-|-|-|-|-|-|-|-|-|->")
time.sleep(2)
cap = cv2.VideoCapture(0)
# Set mediapipe model

with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:

    # NEW LOOP
    # Loop through actions
    for action in actions:
        # Loop through sequences aka videos
        for sequence,folder in enumerate(os.listdir(os.path.join(DATA_PATH, action))):
            if(not(os.listdir(os.path.join(DATA_PATH, action,folder)))):
                for frame_num in range(sequence_length):

                    # Read feed
                    ret, frame = cap.read()

                    # Make detections
                    image, results = mediapipe_detection(frame, holistic)

                    # Draw landmarks
                    draw_styled_landmarks(image, results)

                    # NEW Apply wait logic
                    if frame_num == 0:
                        cv2.putText(image, 'STARTING COLLECTION', (120,200),
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255, 0), 4, cv2.LINE_AA)
                        cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action,
sequence), (15,12),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                        # Show to screen
                        cv2.imshow('OpenCV Feed', image)
                        cv2.waitKey(500)
                    else:
                        cv2.putText(image, 'Collecting frames for {} Video Number {}'.format(action,
sequence), (15,12),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
                        # Show to screen
                        cv2.imshow('OpenCV Feed', image)

                    # NEW Export keypoints
                    keypoints = extract_keypoints(results)
                    npy_path = os.path.join(DATA_PATH, action, str(folder), str(frame_num))
                    np.save(npy_path, keypoints)

                    # Break gracefully
                    if cv2.waitKey(10) & 0xFF == ord('q'):
                        break

    cap.release()
    cv2.destroyAllWindows()
```

B. RESEARCH PAPER

Sign Language Translation in WebRTC Application

Gangadhar Chakali [1]
Department of CSE
Sathyabama Institute Of Science And Technology
Chennai, India
gangadharganga90@gmail.com

Ch. Govardhan Reddy [2]
Department of CSE
Sathyabama Institute Of Science And Technology
Chennai, India
chinthalacheruvugovardhanreddy@gmail.com

Dr. B. Bharathi [3]
Department of CSE
Sathyabama Institute Of Science And Technology
Chennai, India
bharathi.cse@sathyabama.ac.in

***Abstract*—Communication has been an essential part of human life. Different languages are present around the world for communication. Still, people who have lost their hearing and speaking ability by accidents and genetic birth often face difficulty in communication. Auditory-impaired people have found sign language helpful in communicating with others. Hearing-impaired people must always have a personal interpreter available for translations whenever they need to communicate. People with disability of hearing impairment find it challenging to interact with others on social media and the internet to form new relationships on their own. An open-source video-conferencing application that can translate sign language is quite helpful for the hearing impaired. Sign language recognition (SLR) has drawn a lot of attention as a way to close the enormous communication gap. However, sign language is far more complex and unpredictable when compared to other activities, making it challenging for reliable recognition. The Speech-to-Text API enables speech-impaired people who can read to comprehend others. The Sign Language Translation Application (SLTA) allows them to communicate by translating their sign language into text that others can understand. The proposed method uses python, the MediaPipe Framework for gesture data extraction, and the Deep Gesture Recognition (DGR) Model to identify the sign motion in real-time. The proposed method achieves the highest accuracy of 98.81% using a neural network comprised of Long-Short Term Memory units for sequence identification.**

Keywords—American Sign Language (ASL), Neural Networks, Deep Learning, Web- Real-Time Communication (RTC), Long-Short Term Memory (LSTM), Gesture Recognition, Sign Language Translation.

I INTRODUCTION

Human beings communicate with each other through different languages and expressions. The first language evolved around 50,000 to 150,000 years ago. Since then, humans have depended on voice communication to understand each other better. Not everyone is gifted to speak and hear to communicate and understand others. Many people around the globe suffer from listening and speaking problems and are referred to as deaf and mute people or hearing impaired. Approximately around 18 million Indians and 300 million around the world are suffering from auditory impairment. It is hard to communicate with these people because of their disabilities. With this disability, hearing-impaired people find it challenging to find new connections and relationships. These people communicate with each other through sign language. Sign language has wide varieties like American Sign Language (ASL) [20], Chinese Sign Language (CSL), Arabic Sign Language, etc. American Sign Language (ASL) is the most recognized and widely used in different countries. With sign language, there is a huge gap in communication with others. Everyone should know sign language to interpret and understand each other to communicate with people suffering from an auditory impairment. Creating an interface that translates this sign language to the preferred language and vice versa will minimize the communication gap. This study attempts to develop a system that helps to communicate through the video-conferencing application [7][15]. For developing the Sign Language Recognition (SLR) system, this study uses deep learning techniques to collect, train, and test data with open-source tools like TensorFlow, Keras, NumPy, MediaPipe Framework, OpenCV, etc.

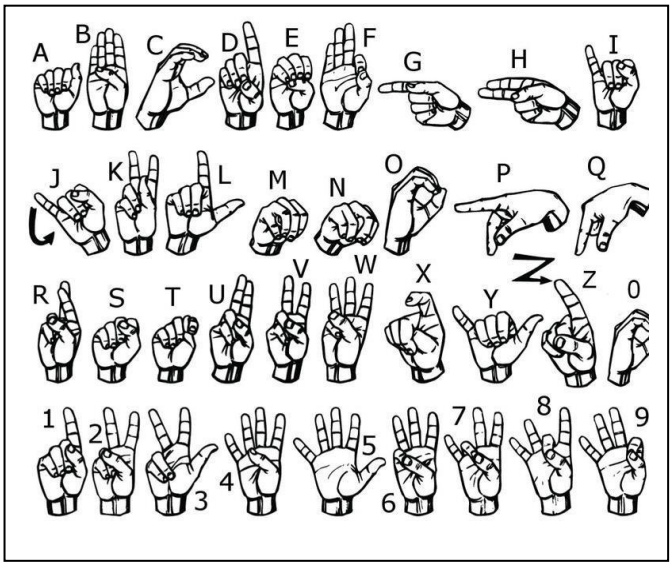


Fig. 1.1. Alpha-Numeric Hand Gestures of American Sign Language,Source: [24]

In the first part of the review, there is an explanation of the usage of sign language, how open-source platforms use SLR services, the technology used, studies that help understand SLR model development in the part of the literature review, and inferences from it.

The second part of the review explains the definitions and analysis of the requirements essential for developing the

model and software integration tools used in this study and the description of Software requirements specifications (SRS).

The third part of the review explains the proposed model and describes software implementation and the development process. Explanation of the model's description, performance, test results, and proposal of a system management plan to fulfil system requirements.

II Related Work

Researchers and computer scientists have researched this problem statement in great detail over the past few years to find a solution, and all of their answers range from examining numerous patterns of gesture recognition techniques to the analysis of different sign languages and different styles of data collection.

The study by author Yeresime Suresh [21] suggested the method employs Canny Edge detection, which provides a more accurate result in detecting edges to identify the edges of hand symbols in the frames. Compared to using embedded sensors in gloves to collect for the identification of gestures, as seen in [3], the suggested system also uses the prediction model of Convolutional Neural Networks (CNN) transcription of a speech and hand symbols shown in figure 1.1. The result of all CNN and Canny Edge Detection provides reliable and accurate results when trained with a large dataset.

The feasibility of using wearable sensor-based devices to recognize hand movements in an application directly connected to sign language was investigated by Karly Kudrinko [14] in her work. Her review aims to identify trends and best approaches by examining earlier studies. The review also points out the difficulties and gaps the sensor-based SLR field is experiencing. Our analysis could help create better SLR [17] systems that can be applied in real-world situations without the dependence on sensor-based wearables. A standardized data collection protocol and evaluation processes might also be developed for her field as a result of looking at diverse study methodologies.

Mathieu De Coster's [13] study tested a variety of neural network topologies, including Hidden Markov Models (HMM), Long Short-Term Memory (LSTM), and Convolutional Neural Networks (CNN), To improve the continuous SLR model. OpenPose is a framework used as a feature extractor to gather the skeleton motion of gestures. Since SLR [17] relies on hand form, location, orientation, and non-manual components like mouth shape, OpenPose is the sole full-body pose estimation methodology used to estimate gesture action. There are other pose estimate methods, but they only identify, for instance, body key points or hand key points (Fang et al., 2017). (Mueller et al., 2018). He retrieved data using the OpenPose Framework, trained the model, and developed the model.

In a study by Bhushan Bhokse, he created a program for gesture recognition that allows a user to demonstrate his hand making a particular motion in front of a video camera connected to a computer. The computer program must gather images of his moves, examine them, and recognize the sign. It was decided that identification would include counting the user's fingers and identifying the American Sign Language they use in the input image to make the system more manageable [12]. He performed experiments using static images on simple backgrounds, retrieved the pictures as grayscale images, and used the binary data from the image for gesture detection.

Sushmita Mitra described gestures, like those used in sign languages, including static and dynamic components, in a study conducted by her [16]. Additionally, talk about how various body motions, including those made with the hands, arms, head, and torso, are used in sign languages. She conducted tests on a different face, hand, and body movement-based algorithms for sign language recognition for the evaluation. She improved accuracy in those studies by fusing Hidden Markov Models (HMM) and finite state machines (FSM) in hybridization.

Zhibo Wang's study [19] covered past works and system trials on various SLRs with wearables that have embedded sensors, such as RF-based [5], PPG-based [9], Acoustic-based [6], Sensing Gloves [8], Vision-based [2][4], EMG-based [9][10][3], and SignSpeaker [1]. These tests are contrasted with his DeepSLR work, which is based on a multi-channel CNN architecture and produces findings that take less than 1.1 seconds to detect signals and recognize a sentence with four sign words, demonstrating DeepSLR's recognition effectiveness and real-time capability in practical situations. The average word error rate for continuous sentence recognition is 10.8%.

In this proposed system, the problems addressed in the existing systems will be solved, like model prediction dependence on sensor gloves and static signs to predict the words, as shown in figure 1.1. This system can reduce this system model dependence on sensor gloves by using an intelligent framework that can extract key points of gesture skeleton structure using a digital camera commonly embedded in any personal computing (PC) device. Extraction of gesture holistic (skeletal system) data is possible with the MediaPipe Framework developed by Google, which can map the skeleton of the human body and extract coordinates of those key point areas for gesture recognition. Combined with multiple frames, this can create a motion sequence to make a word in sign language. From the literature survey, LSTM algorithmic Neural Network architectures could perform better for motion recognition. LSTM algorithm is popularly known for the memory units that present the neural network nodes, which can remember the output of previous data prediction. The memory gate in the LSTM makes it a significant algorithm for SLR applications, where most of the ASL consists of motion gestures. Using the DGR model with LSTM architecture solves the second problem: the barrier of using only static images to predict signs. It means expanding the vocabulary and making the model more versatile and usable in different conditions.

Using the NLP model, these predicted words can be converted into sentences. The NLP model will add meaning and send it to the WebRTC transmission stream to communicate seamlessly by displaying sentences as captions in the interface. The Speech-to-Text model can also help the deaf to read and understand by converting audio of speech and converting to text.

III Proposed System

The proposed system, which is Sign Language Translation Application (SLTA), helps people who face challenges with communication by translating American Sign Language (ASL) for communication, who aren't exposed to sign language communication. SLTA is a video-conferencing

application that makes use of WebRTC protocols for real-time two-way video and audio communication. It is similar to Google Meet, Zoom, and Microsoft Teams, which are embedded with DGR and NLP models that help in SLR [17]. Development of such a system comes with specific challenges like creating visual motion gestures, the vocabulary of a language, training a neural network to accommodate vast vocabulary for Prediction, and making a user-friendly interface system to use the ML translator model.

SLTA can capture the hand gestures with the help of python libraries of OpenCV and MediaPipe, via video frames of the user camera on PC that detects the positions of hand, palm, torso, and face for spatial positioning landmarks of the person, and 21 points of each palm which are coordinates of pixels that help to predict the sign more precisely as shown in figure 3.2. [18]

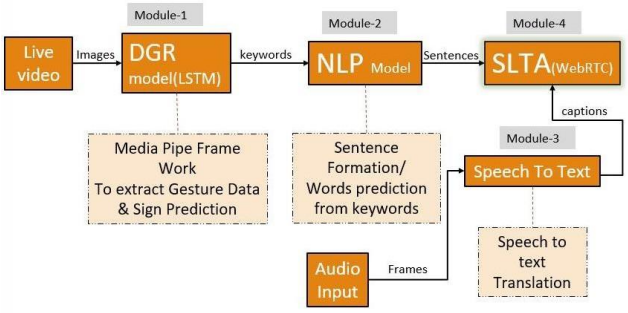


Fig. 3.1. System Architecture of Proposed System SLTA

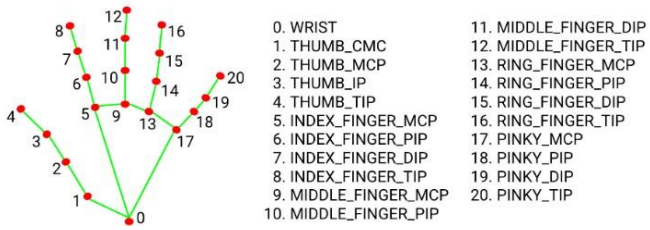


Fig. 3.2. Hand Landmarks detected by Media-Pipe, Source:[22]

The positions of the face, hand, and torso are generally involved with ASL, where movements of different combinations of fingers gesture separate each word in the vocabulary. In a gesture, the positions of the face, body, and torso are important as they create unique vocabulary relating to the wide sign gesture's actions. MediaPipe Framework can also detect the 33 points in pose holistic landmarks (coordinates) of the full body, as shown in figure 3.3. [18] Most of the ASL vocabulary doesn't involves the legs and hip. Only the first 22 points mentioned in figure 3.3 are used to record the dataset.

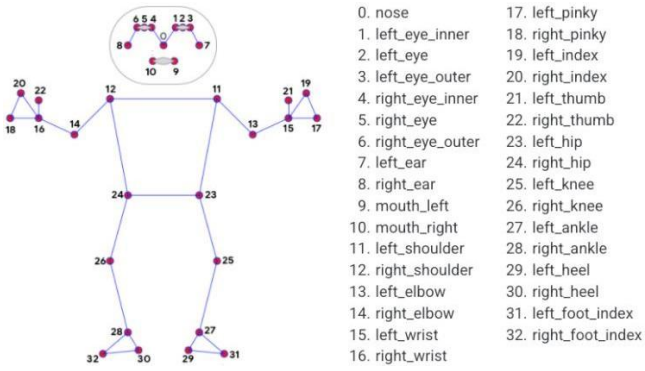


Fig. 3.3. Pose Full Body Landmarks detected By Media-Pipe,Source:[23]

ASL follows the subject-verb-object (SVO), meaning sentences are communicated through sign language without breaks. Phrases intended to communicate are understood with the help of the topic of discussion and verbs signed related to the topic by the cognitive perception of humans. Sign language gestures can be converted to words using the Deep Gesture Recognition model (DGR), developed using a neural network consisting of LSTM units, a module in SLTA. Previous work has shown the limitations of the Convolution Neural Network (CNN), which can only predict static sign gestures [20]. Another study on motion gesture recognition employed wearables [8][11] embedded with sensors that collected spatial 3D data of the hand motion of gestures with acceptable performance using LSTM neural networks. Still, it is challenging to employ economical gloves that require less maintenance. The DGR model uses and accepts the data from a digital front camera commonly found in personal devices as a primary video source. Data from a digital camera is pipelined to the SLTA via the OpenCV module, which helps to manage the digital camera data and manipulate it according to the requirement. MediaPipe Framework [18] makes gesture recognition using Machine Learning (ML) and Deep learning (DL) models for the detection of gesture holistic data of the person at every frame of the gesture video, and predicting the motion gesture with LSTM neural network is trained with ASL.

DGR model can be able to detect sign language and predict the vocabulary in keywords form like "EAT," "DRINK," "HELP," "HELLO," and "THANK YOU," but can't predict the tense of the keywords like "EATING," "DRANK," "HELPED" and also it can't predict articles and prepositions in the English language. Natural Language Processing (NLP) model can analyze and predict sentences with grammar and reconstruct sentences with the meaning the user meant to communicate. Sentences that are predicted and reconstructed by the NLP model are displayed as closed captions (CC) in the SLTA application.

Hearing-impaired people are subjected to understanding words and sentences of a language, which they learned with sign language as they progress the communication. The context of the commoner on the other side of the SLTA can be understood by the hearing impaired using the speech-to-text model, which converts speech to live captions of the speaker, making reading the context easy for the hearing impaired. Developing a user-friendly interface and features is possible for future work by using the Web Real-Time Communication (WebRTC) method and software engineering techniques.

IV RESULTS AND DISCUSSIONS

This section gives a brief discussion of the results of the experiments and analyses that are conducted on the proposed system. Sign language translation requires processing the video frames into sequence data of 30 frames in a video for each gesture captured by the system as one data point in the dataset. Thirty frames of each gesture data point have the gesture data in the form of positional coordinates used by MediaPipe, representing the gesture holistic of one frame. The sequence of frames creates motion of skeletal structure, making it a motion gesture detected by the MediaPipe Framework. These data points are ASL sign gestures that are

```
In [22]:  1 X

Out[22]: array([[[ 0.54207617,  0.36302966, -0.9649995 , ...,  0.        ,
                   0.        ,  0.        ],
                 [ 0.54351079,  0.3640824 , -1.05005968, ...,  0.        ,
                   0.        ,  0.        ],
                 [ 0.54501402,  0.36557683, -1.14532971, ...,  0.        ,
                   0.        ,  0.        ],
                 ...,
                 [ 0.54763758,  0.36452344, -1.07043803, ...,  0.        ,
                   0.        ,  0.        ],
                 [ 0.54843241,  0.36465889, -1.07754028, ...,  0.        ,
                   0.        ,  0.        ],
                 [ 0.54905778,  0.36477691, -1.0854758 , ...,  0.        ,
                   0.        ,  0.        ]],

                [[ 0.54943639,  0.36475685, -1.07610261, ...,  0.        ,
                   0.        ,  0.        ],
                 [ 0.55009383,  0.36485019, -1.12124574, ...,  0.        ,
                   0.        ,  0.        ],
                 [ 0.54876423,  0.36416519, -1.1235069 , ...,  0.        ,
```

2D-array of 30 frames representing a data point

mapped to words. Based on the vocabulary size, LSTM architecture is built to process the 30 frames of sequence gesture of live stream video data and detect the motion.

A. *Conversion of video frames to sequence data*

Using OpenCV, the streamed data is pipelined to the application where gesture skeletal structure data is detected from video frames with the help of MediaPipe Framework, as shown in figure 4.1. MediaPipe [18] uses ML and DL models that detect the skeletal structure in real-time and return collection arrays of each frame as output, considered one point in the ASL dataset of motion gestures.

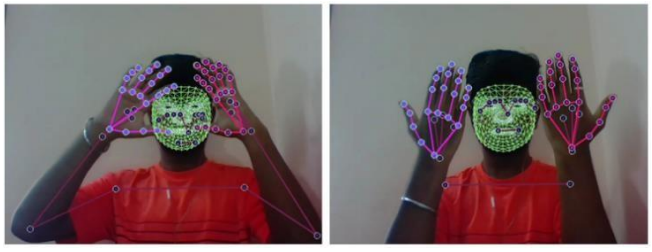


Fig. 4.1. Live Holistic Landmarks detected by MediaPipe

The ASL dataset for training ML models is built using the OpenCV and MediaPipe framework. Each word in the vocabulary is a gesture in ASL performed in front of a digital camera and pipelined to a data collection module that converts video frames to sequence data. The converted sequence data of each frame in 30 frames in motion gesture is stored in a folder where; the folder is considered as one data point representing the word. An independent executable file (.exe) is developed based on the python language and makes use of the 'auto-py-to-exe' library. It can run on any windows machine without installing any python libraries or frameworks. By making this system open-source, users can train their sign gestures with data collector executable files to

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 64)            442112

 lstm_1 (LSTM)               (None, 30, 128)           98816

 lstm_2 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 64)                4160

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 7)                 231

=================================================================
Total params: 596,807
Trainable params: 596,807
Non-trainable params: 0
_________________________________________________________________
```

work according to the ASL, allowing for a vast vocabulary. The ASL dataset is used to experiment with the LSTM architecture and parameters of Neural Networks to develop a reliable SLR [17] system.

B. *Data Pre-processing*

Data returned from the MediaPipe model will be in the form of an object variable. The pose, face, left hand, and right hand (PFLR)landmarks are extracted from MediaPipe objects as x, y, z, and visibility variables. The visibility variable is for only pose landmarks. Extracted landmarks of PFLR have variables in a two-dimensional (2D) array that are flattened in each category and arranged as follows: pose, face, left hand, and right hand. So, every frame in the one motion gesture data point consists of a PFLR sequence of the 1D array where 30 frames give us 30 arrays stored in one folder in the local drive, making it one data point.

Fig. 4.2. Array representation training dataset

ASL datasets stored in folders are retrieved and combined as a 2D array from the file manager to make a data structure to feed the DGR model for training, as shown in figure 4.2. Based on the label of the top folder, the training labels are generated for neural network training.

C. *Building the LSTM architectural Neural Network*

ASL dataset used to train the DGR model comprises six words, "HELLO," "EAT," "THANK YOU," "EXCUSE ME," "HELP," and "PLEASE" along with the word "NONE," also added to the vocabulary. The word "NONE" means the user does not perform a sign gesture. The dataset is developed with the MediaPipe Framework, OpenCV, and other libraries. MediaPipe uses built-in libraries to extract the landmarks of the gesture data.

DGR model architecture comprises six layers where the first three layers are LSTM layers, and the remaining three layers are fully connected layers. DGR model takes 30 frames of gesture data as input and predicts seven words as output, including the word "None." The detailed structure of the DGR model is shown in figure 4.3.

Fig. 4.3. DGR Model Architecture with LSTM layers

The presented training and testing results of LSTM architecture are based on the 360 data points of the dataset given for training the LSTM neural network, consisting of the four words mentioned above. As shown in figure 4.4, various

hyperparameters of LSTM architecture are experimented with and tested to develop the model, which performs an accuracy of 98.81%.

```
optimizer=Adam(decay=1e-4)
# optimizer=SGD(momentum=0.1)
early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=20, restore_best_weights=True)#monitor=val_
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['acc'])
```

```
checkpoints=ModelCheckpoint('Logs\\checkpoints\\' + 'ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5',
                            monitor='val_loss', save_weights_only=True, save_best_only=True)#, period=3)
```

```
model.fit(X, y, batch_size=30, epochs=1000, validation_data=(X_test, y_test), shuffle=True,callbacks=[early_stoppin
```

```
42/42 [==============================] - 4s 87ms/step - loss: 0.0575 - acc: 0.9841 - val_loss: 0.0548 - val_acc: 0.9841
Epoch 23/1000
42/42 [==============================] - 4s 85ms/step - loss: 0.0854 - acc: 0.9706 - val_loss: 0.0827 - val_acc: 0.9683
Epoch 24/1000
42/42 [==============================] - 4s 85ms/step - loss: 0.0791 - acc: 0.9722 - val_loss: 0.0798 - val_acc: 0.9603
Epoch 25/1000
42/42 [==============================] - 4s 85ms/step - loss: 0.0810 - acc: 0.9770 - val_loss: 0.0760 - val_acc: 0.9802
Epoch 26/1000
42/42 [==============================] - 4s 86ms/step - loss: 0.0673 - acc: 0.9738 - val_loss: 0.0554 - val_acc: 0.9881
Epoch 27/1000
42/42 [==============================] - 4s 87ms/step - loss: 0.0612 - acc: 0.9794 - val_loss: 0.0930 - val_acc: 0.9722
Epoch 28/1000
42/42 [==============================] - 4s 87ms/step - loss: 0.0644 - acc: 0.9825 - val_loss: 0.0448 - val_acc: 0.9921
Epoch 29/1000
42/42 [==============================] - 4s 92ms/step - loss: 0.0526 - acc: 0.9817 - val_loss: 0.0352 - val_acc: 0.9921
Epoch 30/1000
42/42 [==============================] - 4s 95ms/step - loss: 0.0433 - acc: 0.9889 - val_loss: 0.0381 - val_acc: 0.9921

<keras.callbacks.History at 0x17518829eb0>
```

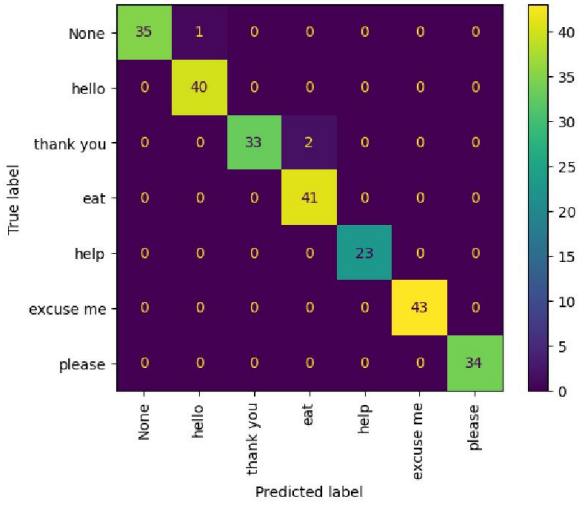
Fig. 4.4. Training results of the DGR Model

D. Sign Gesture Prediction and results

MediaPipe Framework extracts the gesture data from real-time video stream data from a digital camera pipelined with OpenCV to

```
Loss,Accuracy=model.evaluate(X_test, y_test)
print(f'Accuracy : {round(Accuracy*100,2)}%\nLoss : {round(Loss,4)}')
```

```
8/8 [==============================] - 0s 37ms/step - loss: 0.0341 - acc: 0.9881
Accuracy : 98.81%
Loss : 0.0341
```



the application. It captures the pose, face, left-hand, and right-hand coordinates from each video frame and stores them as sequence data. This gesture data is continuously given to the DGR model, which considers 30 frames of data for the Prediction of a single gesture. Each frame sequence consists of pose landmarks: 132, face landmarks: 1404, left-hand landmarks: 63, and right-hand: 63 points, giving 1662 points of sequence data in the whole data point. The results DGR model with a confusion matrix, accuracy, and loss are shown in figure 4.5.

Fig. 4.5. Confusion Matrix, Accuracy and Loss of the DGR Model

DGR model with LSTM architecture predicts the gesture with the given data after pre-processing and conversion to data structure from live video data in the above-mentioned sequence. Each gesture is predicted based on the probability of the gesture performed at every sequence of 30 frames. The word is predicted based on the highest probability in vocabulary, and that word crosses the threshold probability of 0.9. Each predicted word is given to the display output, as shown in figure 4.6.

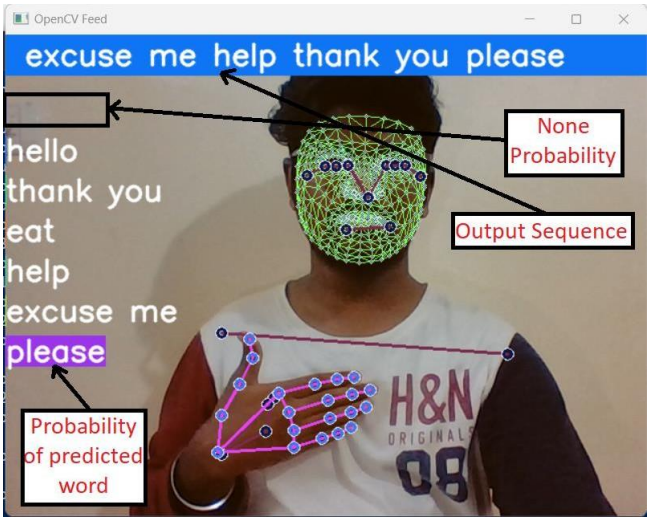


Fig. 4.6. Prediction of ASL with the DGR model

V CONCLUSION

Sign language has been the primary way of communication for hearing and speech-impaired people. Most people aren't aware of sign language, especially American sign language. Common people often find it difficult to understand disabled people. Sign Language Translation Application (SLTA) aims to bridge the communication barrier between disabled and abled persons using sign language recognition models. SLTA captures signed gestures with a digital camera and extracts the gesture data using the MediaPipe Framework. With the help of the DGR model developed with LSTM architecture, it can predict the dynamic motion gestures performed by the person in Infront of the camera in real-time. An Accuracy of 98.81% has been achieved with seven words of vocabulary and 252 test samples. The proposed system also overcomes the existing problems of static gesture recognition using LSTM architecture by predicting motion gestures and replacing the usage of sensor-based gloves and wearables for gesture motion data collection.

Using WebRTC protocols, this application can be implemented as a video conferencing for speech and hearing-impaired people. Further development and refinement can be done on this system by including the expansion of the size of ASL vocabulary, development of the NLP model for predicted words to sentences, and implementation of WebRTC protocol for video conferencing for communication can be considered for future work.

REFERENCES

[1] J. Hou, X.-Y. Li, P. Zhu, Z. Wang, Y. Wang, J. Qian, and P. Yang, "Signspeaker: A real-time, high-precision smartwatch-based sign language translator," in Proc. of ACM MobiCom, 2019.

[2] J. Huang, W. Zhou, Q. Zhang, H. Li, and W. Li, "Video-based sign language recognition without temporal segmentation," arXiv preprint arXiv:1801.10111, 2018.

[3] J. Wu, Z. Tian, L. Sun, L. Estevez, and R. Jafari, "Real-time American sign language recognition using wrist-worn motion and surface emg sensors," in Proc. of IEEE BSN, 2015, pp. 1–6.

[4] J. Zang, L. Wang, Z. Liu, Q. Zhang, G. Hua, and N. Zheng, "Attention-based temporal weighted convolutional neural network for action recognition," in Proc. of IFIP INTERACT, 2018, pp. 97–108.

[5] J. Zhang, J. Tao, and Z. Shi, "Doppler-radar based hand gesture recognition system using convolutional neural networks," in International Conference in Communications, Signal Processing, and Systems. Springer, 2017, pp. 1096–1113.

[6] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "Fingerio: Using active sonar for fine-grained finger tracking," in Proc. of ACM CHI, 2016, pp. 1515–1525.

[7] Julian Menezes .R, Albert Mayan .J, M. Breezely George," Development of a Functionality Testing Tool for Windows Phones", Indian Journal of Science and Technology,Vol:8,Issue:22,pp: 1-7,September 2015.

[8] T. T. Swee, A. Ariff, S.-H. Salleh, S. K. Seng, and L. S. Huat, "Wireless data gloves malay sign language recognition system," in Information, Communications & Signal Processing, 2007 6th International Conference on. IEEE, 2007, pp. 1–4.

[9] T . Zhao, J. Liu, Y. Wang, H. Liu, and Y. Chen, "Ppg-based finger level gesture recognition leveraging wearables," in Proc. of IEEE INFOCOM, 2018, pp. 1457–1465.

[10] X. Zhang, X. Chen, Y. Li, V. Lantz, K. Wang, and J. Yang, "A framework for hand gesture recognition based on accelerometer and emg sensors," IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans, vol. 41, no. 6, pp. 1064–1076, 2011.

[11] Z. Lu, X. Chen, Q. Li, X. Zhang, and P. Zhou, "A Hand Gesture Recognition Framework and Wearable Gesture-Based Interaction Prototype for Mobile Devices," in IEEE Transactions on Human-Machine Systems, vol. 44, no. 2, pp. 293-299, April 2014, doi: 10.1109/THMS.2014.2302794.

[12] Bhokse, B. (January 1, 2015). ISSN 2348 – 7968 hand gesture recognition using a neural network - IJISET. IJISET - International Journal of Innovative Science, Engineering & Technology. Retrieved October 24, 2022, from https://www.ijiset.com/vol2/v2s1/IJISET_V2_I1_01.pdf

[13] Coster(UGent), M. D., & Dambre(UGent), and J. (1970, January 1). Sign language recognition with Transformer Networks. Sign language recognition with transformer networks. Retrieved October 24, 2022, from http://hdl.handle.net/1854/LU-8660743

[14] K. Kudrinko, E. Flavin, X. Zhu, and Q. Li, "Wearable Sensor-Based Sign Language Recognition: A Comprehensive Review," in IEEE Reviews in Biomedical Engineering, vol. 14, pp. 82-97, 2021, doi: 10.1109/RBME.2020.3019769.

[15] Asha Pandian, Bharathi B, Albert Mayan J, Prem Jacob, Pravin "A Comprehensive View of Scheduling Algorithms for MapReduce Framework in Hadoop," Journal of Computational and Theoretical Nanoscience, Vol.16, No. 8, pp. 3582-3586, 2019

[16] Mitra, S. (2007, May 3). GESTURE RECOGNITION: A survey. IEEE Xplore. Retrieved October 24, 2022, from https://ieeexplore.ieee.org/document/4154947

[17] Razieh Rastgoo, Kourosh Kiani, Sergio Escalera, Sign Language Recognition: A Deep Survey, Expert Systems with Applications, Volume 164, 2021, 113794, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2020.113794. (https://www.sciencedirect.com/science/article/pii/S0957417420306 14X)

[18] Lugaresi, Camillo. "MediaPipe: A Framework for Building Perception Pipelines." arXiv.org, June 14, 2019. https://arxiv.org/abs/1906.08172.

[19] Wang, Z., Zhao, T., Ma, J., Chen, H., Liu, K., Shao, H., Wang, Q., & Ren, J. (2020). Hear sign language: A real-time end-to-end sign language recognition system. IEEE Transactions on Mobile Computing, 1–1. https://doi.org/10.1109/tmc.2020.3038303

[20] Wikimedia Foundation. (2022, October 23). American sign language. Wikipedia. Retrieved October 24, 2022, from https://en.wikipedia.org/wiki/American_Sign_Language

[21] Y. Suresh, J. Vaishnavi, M. Vindhya, M. S. A. Meeran and S. Vemala, "MUDRAKSHARA - A Voice for Deaf/Dumb People," 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 2020, pp. 1-8, doi: 10.1109/ICCCNT49239.2020.9225656.

[22] Bazarevsky, V., & Zhang, F. (2019, August 19). Hands. On-Device, Real-Time Hand Tracking with MediaPipe. Retrieved February 10, 2023, from https://google.github.io/mediapipe/solutions/hands.html

[23] Bazarevsky, V., & Grishchenko, I. (2020, August 13). Pose. On-device, Real-time Body Pose Tracking with MediaPipe BlazePose. Retrieved February 10, 2023, from https://google.github.io/mediapipe/solutions/pose.html

[24] Teak-Wei , C., & Boon Giin, L. (2018, October). The 26 letters and 10 digits of American Sign Language (ASL). American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach. Retrieved February 10, 2023, from https://www.researchgate.net/figure/The-26-letters-and-10-digits-of-American-Sign-Language-ASL_fig1_328396