AMERICAN SIGN LANGUAGE RECOGNITION USING DEEP LEARNING

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

by

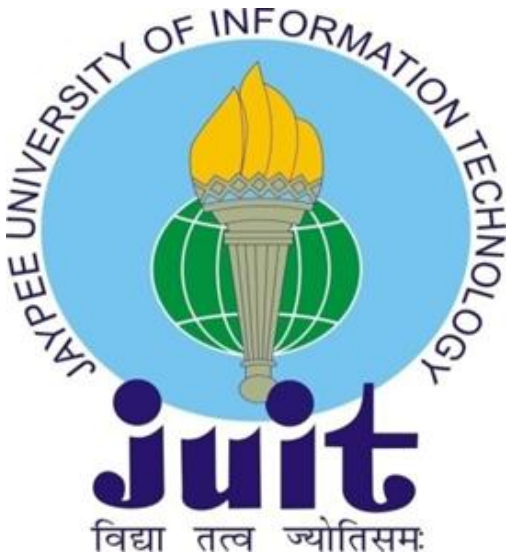
Manya Malhotra (191427)

Under the supervision of

Dr. Diksha Hooda

to

Department of Computer Science & Engineering and Information Technology



Jaypee University of Information Technology,
Waknaghat, Solan-173234, Himachal Pradesh

CERTIFICATE

I hereby declare that the work presented in this report entitled **American Sign Language Recognition using Deep Learning** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over the period from July 2022 to May 2023 under the supervision of **Dr. Diksha Hooda** (Assistant Professor, Department of CSE).

I also authenticate that I have carried out the above mentioned project work under the proficiency stream **Artificial Intelligence**.

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

Manya Malhotra
191427

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

Dr. Diksha Hooda
Assistant Professor
Department of Computer Science and Engineering
Dated:

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: .....

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: _____ **Department:** _____ **Enrolment No** _____

Contact No. _____ **E-mail.** _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at .....(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><td>Copy Received on</td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td>Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2"></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

ACKNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for his divine blessing making it possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to my supervisor **Dr. Diksha Hooda, Assistant Professor (Grade-II)**, Department of CSE, Jaypee University of Information Technology, Waknaghat. Her deep knowledge & keen interest in the field of **"Artificial Intelligence"** helped me to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Diksha Hooda**, Department of CSE, for her kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Manya Malhotra
191427
Project Group No.: 90

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

ASL	American Sign Language
SLR	Sign Language Recognition
ML	Machine Learning
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
MP	MediaPipe
CV	Computer Vision
BGR	Blue → Green → Red
RGB	Red → Green → Blue
ROI	Region Of Interest

LIST OF FIGURES

LIST OF GRAPHS

LIST OF TABLES

ABSTRACT

To interact with one another, we humans require a means of communication. "Specially abled" persons, those with speech or hearing disorders, "Mute" and "Deaf" people, are always reliant on some form of visual communication. People who may not have visual or hearing impairments may have difficulty communicating with those who do.

In order to achieve two-way communication between people with disabilities and the general public, a system that can translate hand gestures into text and speech needs to be developed. Sign language is one of the oldest and most natural forms of language for communication, but because the majority of people do not know sign language, it is very difficult to find interpreters. In light of this, we have developed a real-time approach for fingerspelling based on American sign language utilising neural networks.

Deep learning approaches can aid in the reduction of communication barriers. The following are the major steps in system design: tracking, segmentation, gesture acquisition, feature extraction, gesture recognition and text to speech conversion. The Sign Language recognition algorithm is trained on a dynamic dataset of ordinary motions created by the author. The trained model correctly recognises the gesture, displays it on the screen in the form of text and converts it into speech.

1. INTRODUCTION

A gesture is any movement of a body part, such as the face or the hand. Image processing and computer vision are used here for sign language recognition and python text to speech is used for speech conversion. Sign recognition allows computers to understand human actions and serves as a translator between computers and humans. This could allow humans to engage naturally with computers without coming into direct contact with the mechanical equipment. Deaf and dumb people use hand and pose gestures to communicate in sign language. When transmitting voice is impossible or typing and writing is problematic, but there is the possibility of seeing, this community uses sign language to communicate. The sole means of communication between people at the moment is sign language.

Normally, everyone uses sign language when they don't want to speak, but for the deaf and dumb community, this is their sole means of communication. The same meaning is conveyed through sign language as it is through spoken language. All around the world, the deaf and dumb community uses this, though in localized forms like ISL and ASL. One or two hands can be used to make hand gestures when utilizing sign language. There are two types of it: continuous sign language and isolated sign language. Continuous ISL, also known as Continuous Sign language, is a series of movements that produce a coherent sentence as opposed to isolated sign language, which consists of a single motion with a single word. We used an independent ASL gesture recognition algorithm in this work.

1.1 Introduction

Humans can communicate with one another in a variety of ways. This include behaviour such as physical gestures, facial expressions, spoken words, etc. However, those who have hearing loss are restricted to using hand gestures to communicate. People with hearing loss and/or speech impairments

communicate using a standard sign language that is incomprehensible to non-users.

Sign language is the communication system for those who are hard of hearing and deaf. It ranks as the sixth most utilized language worldwide. It is a type of communication that uses hand movements to communicate ideas. Each region has its specific sign language like normal language. In 2005 there were an estimated 62 million deaf people worldwide and about 200 different sign languages in use around the world, many of which have distinctive features.

ASL is the primary language of many deaf citizens in North America. Hard-of-hearing and hearing people also use it. Hand gestures and facial expressions are used to convey this language. The deaf community has access to ASL as a means of communication with the outside world and inside the community. But not everyone is familiar with the signs and motions used in sign language. Understanding sign language and being familiar with its motions takes a lot of practice. Since there are no reliable, portable tools for identifying sign language, learning sign language takes a lot of time. However, since the development of neural networks and deep learning, it is now possible to create a system that can identify things, or even objects of different categories.

In this project, our primary focus is on creating a model that can recognize hand movements and combine each motion to form a whole word & then convert that predicted text into speech. Few gestures that are practiced are displayed in the image below.

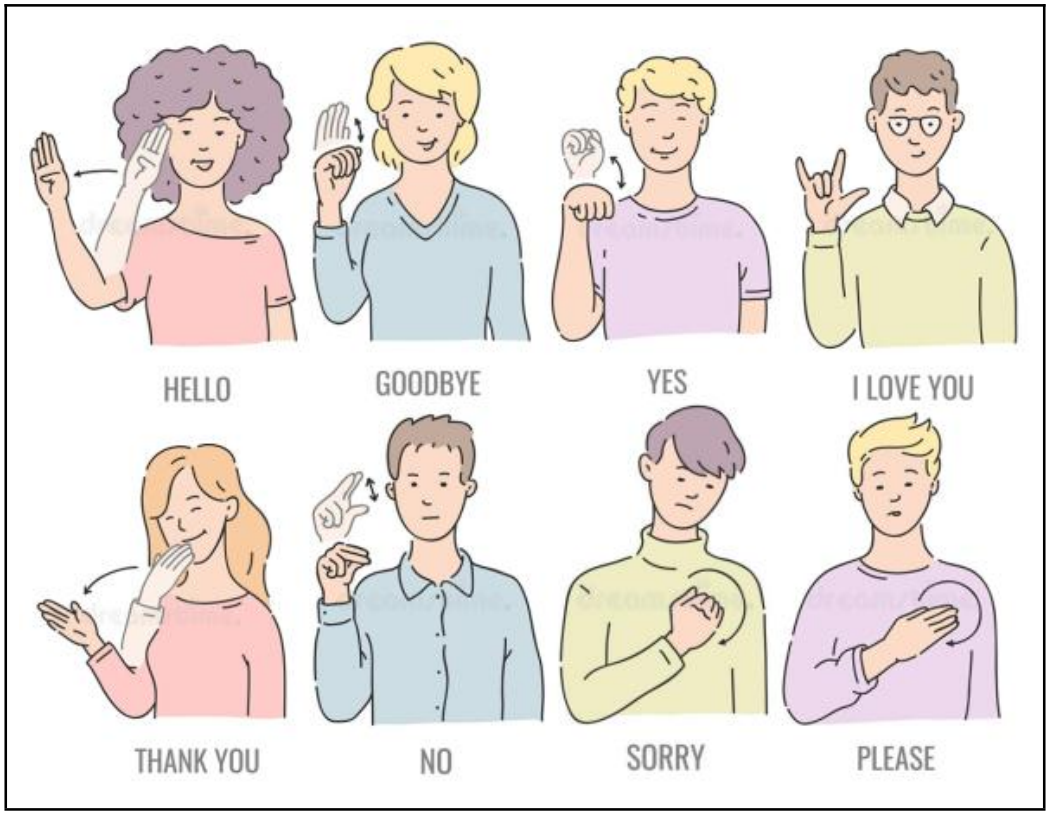


Fig. 1 - American Sign Languages.

1.2 Problem Statement

It's important to interact with everyone in our modern culture, whether it's for fun or for work. Communication has always had a great impact in every domain and how it is considered the meaning of the thoughts and expressions that attract the researchers to bridge this gap for every living being.

Speech impairment is a disability which affects an individuals ability to communicate using speech and hearing. People who are affected by this use other media of communication such as sign language. However, learning and understanding sign language requires a lot of practice, and not everyone will comprehend what the gestures in sign language indicate. It takes time to learn sign language because there is no reliable, portable tool for doing so.

Hearing or speech-impaired individuals who are proficient in sign language need a translator who is equally proficient in sign language to

effectively communicate their ideas to others. This technique assists people with hearing loss or speech impairments in learning and translating their sign language in order to help them overcome these issues.

1.3 Objectives

- According to statistics, over 80% of specially abled individuals are illiterate, and the system tries to bridge the gap between a normal, a hearing-impaired and a visually impaired person by turning a majority of sign language to text and speech.

- People who are deaf or hard of hearing can communicate their message using gestures that can be read.

- People who are not visually challenged can use the software to comprehend sign language and communicate effectively with those who are. Also, people who are visually impaired can also communicate when the sign language predicted text is converted to speech.

- This project will bridge the gap of difficulty in understanding sign language that existed previously.

- To train a model, it will employ cutting-edge deep learning algorithms. The model will collect frames for gestures using the camera, train the model, and evaluate the precision for each gesture. The gesture will then be predicted in real time. The gesture is then translated to text and speech.

- The objective of this project is to identify the symbolic expression through images so that the communication gap between a normal and hearing impaired person can be easily bridged by:

i. Creating data with respect to American sign language & pre-process it.
ii. Training the pre-processed data with Deep Learning based models to perform sign language recognition & speech conversion in real time.
iii. Testing the model in the real world scenario.

1.4 Methodology

The system uses a vision-based method. Since all of the signs are portrayed with bare hands, there is no need for any artificial gadgets for interaction.

We looked for pre-made datasets for the project but couldn't find any in the form of raw images that met the software specifications. As a result, we decided to develop our own data set. To create the dataset, Open Computer Vision (OpenCV) library of python is used.

To accurately recognize the sign gestures and translate them into text & then convert the text to speech, our proposed method comprises three stages: data preprocessing and feature extraction, data cleaning and labelling and gesture recognition & speech translation. Data preprocessing and feature extraction are carried over by the MediaPipe framework. Here, features from the face, hands, and body are extracted as keypoints and landmarks using built-in data augmentation techniques from sequence of input frames taken from a web camera. In stage 2, the extracted keypoints from stage 1 are saved in a file to identify and remove the null entries from the data, after which data labelling follows. In stage 3, the cleaned and labelled gestures are trained and classified by our LSTM model for sign language recognition in the form of text on the screen. The displayed text is then converted to speech. The three stages of the proposed methodology are elaborately discussed below.

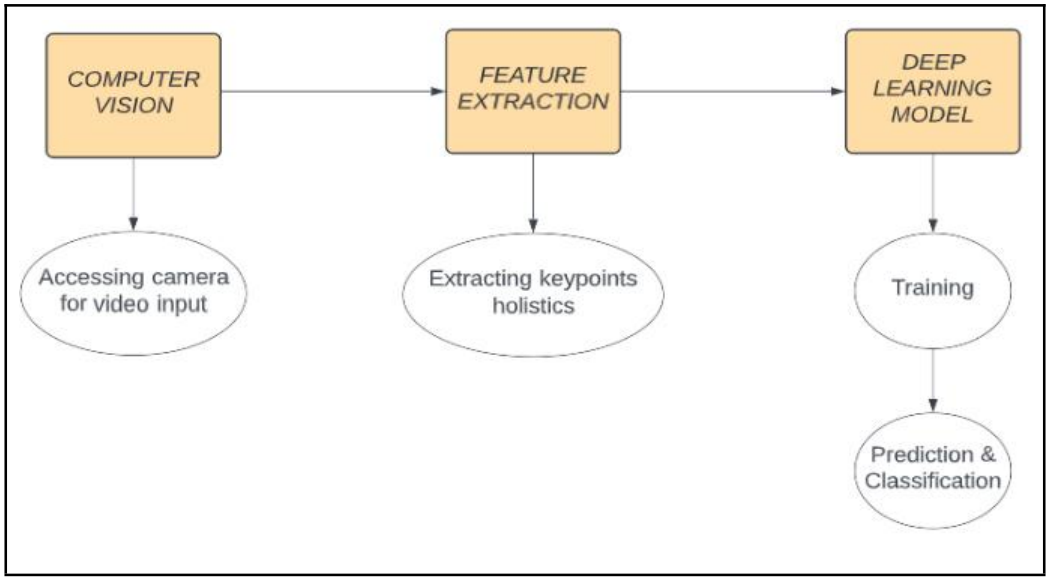


Fig. 2 - Methodology

Stage 1: Data preprocessing and feature extraction. For data preprocessing and feature extraction from the image, we applied a multistage pipeline from MediaPipe, called MediaPipe Holistic. For each input frame from the web camera, the MediaPipe Holistic handled individual models for the hands, face, and pose components using a region- appropriate image resolution. The workflow of stage 1 is briefly discussed below:

• The human pose and subsequent landmark model were estimated using BlazePose's pose detector. Then, three ROI crops for the face and hands (2×) were derived from the inferred pose landmarks, and a re-crop was employed to improve the ROI.

• Next, the corresponding landmarks were estimated. To achieve this, the full-resolution input coordinates were cropped to the ROIs for task-specific hand and face models.

• Finally, all landmarks were combined to yield the full 540+ landmarks.

Stage 2: Data cleaning and labelling. After step 1, the landmark points (21 3 +213 + 33 4 + 468 3 = 1662) are fattened, concatenated, and put in a file to be

checked for and any null entries from the data are removed. Data cleaning is essential because it avoids failed feature detection, which happens when a blurry image is submitted to the detector and results in a null entry in the dataset. However, when using this noisy data for training, the prediction accuracy may suffer and bias may develop. Labels are constructed for each class and their associated frame sequences are saved in order to suit the received data for the subsequent stages of training, testing, and validation.

Stage 3: Gesture recognition & Speech translation. Now that we have an LSTM model, we can detect action with a limited amount of frames by training it with the data we have already stored.

The number of epochs for the model is decided upon; as the number of epochs rises, so does the amount of time needed to run the model, and overfitting of the model for gesture recognition is a possibility.

As soon as the model is trained, we can use it to recognise sign language in real time utilising OpenCV module.

The recognised text from the sign language is then converted into speech using python-text-to speech module. The converted audio file is then played simultaneously using the default audio output device present in the system.

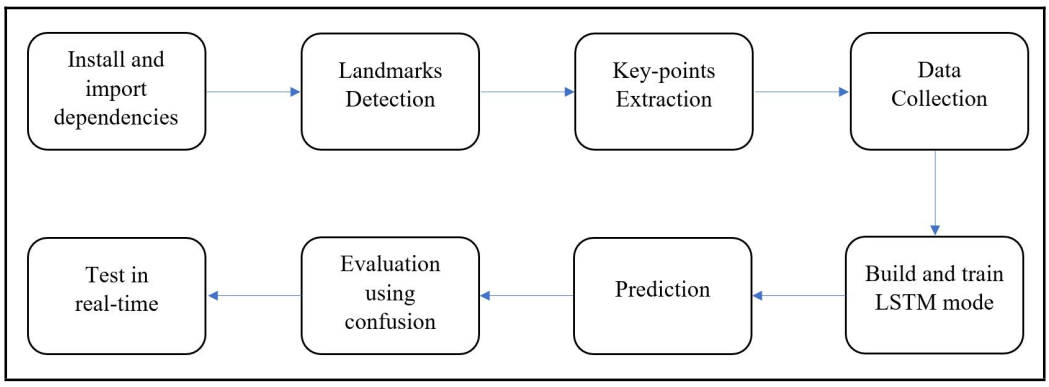


Fig. 3 - Detailed Flow of Project.

```
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, 3)                 99

=================================================================
Total params: 596,675
Trainable params: 596,675
Non-trainable params: 0
```

Fig. 4 - Model Summary.

1.5 Organization

The rest of this report is organized as follows:

Chapter 2 gives an overview of the literature study performed.

Chapter 3 discusses the system development and workflow.

Chapter 4 shows the performance analysis..

Chapter 5 highlights the conclusion, future scope and application contribution.

2. LITERATURE SURVEY

2.1 Related Work

Previous researchers have emphasised their work on the prediction of sign language gestures to support people with hearing impairments using advanced technologies with artificial intelligence algorithms. Although much research has been conducted for SLR, there are still limitations and improvements that need to be addressed to improve the hard-of-hearing community. This section presents a brief literature review of recent studies on SLR using sensor and vision-based deep learning techniques.

Literature review of the problem shows that there have been several approaches to address the issue of gesture recognition in video using several different methods. In [1] the authors used Hidden Markov Models (HMM) to recognize facial expressions from video sequences combined with Bayesian Network Classifiers and Gaussian Tree Augmented Naive Bayes Classifiers.

Francois et al. [2] also published a paper on Human Posture Recognition in a Video Sequence using methods based on 2D and 3D appearance. The work mentions using PCA to recognize silhouettes from a static camera and then using 3D to model posture for recognition. This approach has the drawback of having intermediary gestures which may lead to ambiguity in training and therefore a lower accuracy in prediction.

Let's approach the analysis of video segments using Neural Networks which involves extracting visual information in the form of feature vectors. Neural Networks do face issues such as tracking of hands, segmentation of subjects from the background and environment, illumination of variation, occlusion, movements and position. The paper by Nandy et al. [3] splits the dataset into segments, extracts features and classifies using Euclidean Distance and K-Nearest Neighbors.

Similar work by Kumud et al. [4] defines how to do Continuous Indian Sign Language Recognition. The paper proposes frame extraction from video data, pre-processing the data, extracting key frames from the data followed by extracting other features, recognition and finally optimization. Pre-processing is done by converting the video to a sequence of RGB frames. Each frame having the same dimensions. Skin colour segmentation used to extract skin regions, with the help of HSV. The images obtained were converted to binary form. The key frames were extracted by calculating a gradient between the frames. And the features were extracted from the key frames using oriental histogram. Classification was done by Euclidean Distance, Manhattan Distance, Chess Board Distance and Mahalanobis Distance.

Table 1. List of References

AUTHOR	METHODOLOGY	LIMITATIONS
Kshitij Bantupalli and Ying Xie International Conference, 10-13 December 2018 [11]	Video sequences are fed into the model, which then extracts temporal and spatial information from them.	The model ends up learning inaccurate features from the films while testing with varying skin tones and including diverse faces, which causes accuracy to decline over time.
Aju Dennisan Journal, 2019 [7]	Making a model to recognize ASL alphabet from RGB images.	They have done the recognition only on the alphabets with 83.29% accuracy.
K Amrutha and P Prabu	Developing a system that can read and interpret a sign using a dataset and	The model showed an accuracy of only 65% due to less data set.

International Conference, 11-12 February 2021 [5]	the best algorithm.	
A. Mittal, P. Kumar, P. P. Roy, R. Balasubramanian, and B.B. Chaudhuri [2]	A Modified LSTM Model for Continuous Sign Language Recognition Using Leap Motion	The model showed an accuracy of only 72.3% and 89.5% due training dataset.
M. Wurangian [6]	American sign language alphabet recognition	They have done the recognition only on the alphabets.
P. Likhar, N. K. Bhagat and R. G N [9]	Deep Learning Methods for Indian Sign Language Recognition	This model was developed for Indian Sign Language, our model is concerned with American Sign Language.
Shivashankara, Ss, and S. Srinath [12]	American sign language recognition system: an optimal approach	Model is restricted to alphabet recognition only.

3. SYSTEM DEVELOPMENT

3.1 Background

Gesture recognition is a hot topic in Human-Computer Interaction research. It has a wide range of applications, including virtual environment control, sign language translation, robot control, and music creation. We will create a real-time Sign Language Recognizer using the MediaPipe framework and Tensorflow in OpenCV and Python in this machine learning project on American Sign Language Recognition.

OpenCV is a real-time computer vision and image processing framework built on C/C++. However, we will use it in Python via the OpenCV-python package.

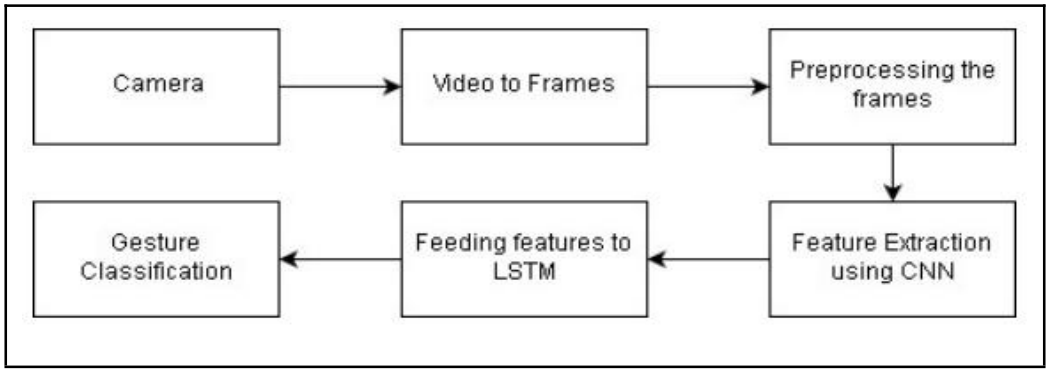


Fig. 5 - High Level System Architecture.

What is MediaPipe?

Creating pipelines for processing perceptual data including photos, movies, and audio requires making the system compliant with the MediaPipe, a hybrid open-source architecture. Real-time hand tracking and gesture detection are accomplished using a thorough approach that makes advantage of ML. By precisely identifying sign gestures, it provides more hand and

finger tracking solutions. Using a MediaPipe Holistic pipeline, we were able to extract the landmarks from the position of the torso, hands, and face.

MediaPipe holistic pose landmarks:

The MediaPipe Holistic body pose model locates the person/position areas of interest (ROI) inside the frame using its BlazePose detector to infer around 33 3D landmarks on the body consisting of x, y, and z coordinates from the input image or video. The ROI-cropped frame is used as input by the pose landmark and division masks to progressively detect postures. Thus, it more precisely localizes critical locations and suits SLR.

MediaPipe holistic hand landmarks:

In a single frame, MediaPipe Holistic hands infers approximately 21 3D hand landmarks consisting of x, y, and z coordinates and produces the desired output by combining two models:

i. the palm detection model

ii. the hand keypoint localization model.

Initially, the model was used with a single-shot detector known as the Blaze Palm. Given a large dataset of hand sizes in the input image, this detector supports the MediaPipe to reduce the time complexity of palm detection. Instead of focusing on unnecessary objects, this model works on the entire image and returns a focused bounding box that highlights the rigid parts, such as palm and fist, for palm detection. The palm detection output is then used by the model to perform hand keypoint localization.

This produced three possible outputs as follows:

- 21 hand knuckle points in a 2D or 3D space.
- Hand flag showing the probability of hand presence in the input image.
- Binary classification of left and right hand.

MediaPipe holistic face landmarks:

The MediaPipe face mesh is a face geometry solution that computes 468 3D face landmarks in real time using a single input camera rather than a depth sensor. It operates on the basis of two deep neural network models: a detector that computes and operates face locations on a full image, and a 3D face landmark model that operates on the computed locations to predict approximate surface geometry using regression. Cropping the face accurately reduces data augmentation processes like rotation, scaling, and translation, allowing the network to focus more on coordinate prediction accuracy.

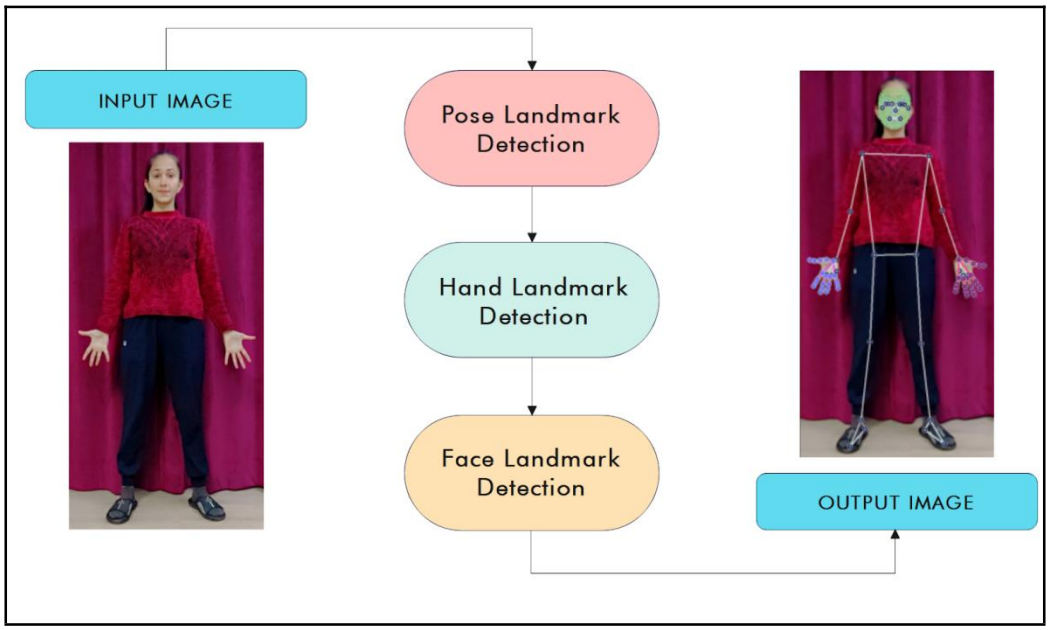


Fig. 6 - Overview of Mediapipe Holistic.

3.2 Proposed Work

The proposed work considers the issues faced by prior models and works to reduce them. We created a system that does not compromise on efficiency or performance. The non-uniform background and segmentation of hands from the background was one of the key issues that researchers confronted. We solved that problem by creating a dataset using Google's Mediapipe solution and the openCV library to detect landmarks from the hand, which we also used during the real-time detection part of the hand gesture recognition system,

so that regardless of whether one is in their car, at home,or on the street, the system will detect accurate landmarks from their hands. As illustrated in fig. 6, the suggested system consists of three primary modules that are linked in series.

The Dataset module is where the landmarks are extracted and the dataset construction process is completed, followed by the Preprocessing module, where the data is processed to input into the final LSTM module, where the training for detecting gestures takes place. Finally, once the model is delivering appropriate results after modifying the hyper parameters, a camera renders the live real-time feed and sends it through the model, and the name of the gesture is shown as text on the screen. Figure 7 depicts the architecture of the proposed Hand Gesture Recognition System.

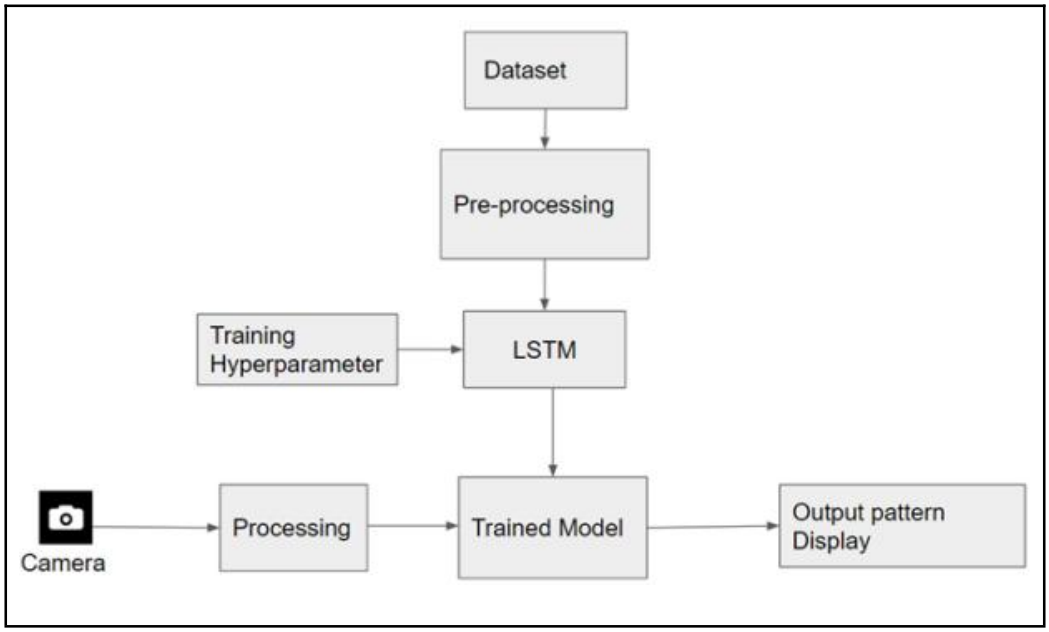


Fig. 7 - Architecture of the Hand Gesture Recognition System. [16]

3.2.1. Comparison of proposed vs existing approach based on technology used, gesture recognized and accuracy

A couple of metrics were imported from sci-kit learn to evaluate the performance of the designed model. Using multi_label_confusion_matrix to

get the confusion matrix for each one of the different labels. This allowed us to evaluate what's being detected as a True Positive and a True Negative & what's being detected as a False Positive and a False Negative. Later by using accuracy_score the model accuracy was found out to be 80%. Table 2 shows the comparison between the existing approaches and our proposed method.

Reference	**Technology Used**	**Gestures Recognized**	**Static / Dynamic videos/images**	**Accuracy**
[14]	Image Processing, Squeezenet model	All 26 English alphabet	Only static images	83.29%
[3]	Hidden Markov model and keyframe Extraction	The number of persons is 5, and the sentences are 10	Static/Dynamic images and videos	94%
[4]	Sensors: Leap Motion & Microsoft Kinect	Dataset consists 35 isolated sign words	Videos	72.3% (sign sentences), 89.5% (sign words)
[9]	U-net with ResNet 101, Microsoft Kinect	The dataset was gathered from 5 subjects having 10 distinct dynamic gestures commonly exercised in ISL (Indian Sign Language)	Static and dynamic based images and videos	78.3%
Proposed Work	Video processing, keypoint extraction using	The number of persons are 2 and the the number of phrases are 10	Static and dynamic images and videos	80%

	mediapipe holistic			

3.3 Experimental Setup

3.3.1. Tools / technologies used

The proposed model is deployed on an Intel Core processor with 8 GB RAM and Windows 10 operating system. The proposed model is developed using the Anaconda Navigator platform to set up the environment for Jupyter Notebook.Python programming language is used to develop the model in combination with Python libraries (OpenCV, MediaPipe) and Deep learning model LSTM.

3.4 Software Design

The proposed system follows a vision-driven methodology to perform gesture-based sign recognition from frames extracted from video inputs. The sign recognition process consists of three phases, namely data collection, data pre-processing and feature extraction and gesture recognition. After the data collected is pre-processed and augmented , the feature extraction process is initiated. In this, facial features, landmarks of both hands, and bodily postures are extracted as keypoints from a sequence of input frames captured via a web camera. Then the extracted essential data points are considered as crucial features to be fed to the implemented classifier that recognizes the gestures performed by the user . These recognized gestures and moments are further converted to the textual form and displayed on the screen in real time. Figure 8 shows the architectural diagram of the proposed system. The three phases of the system are discussed in detail below:

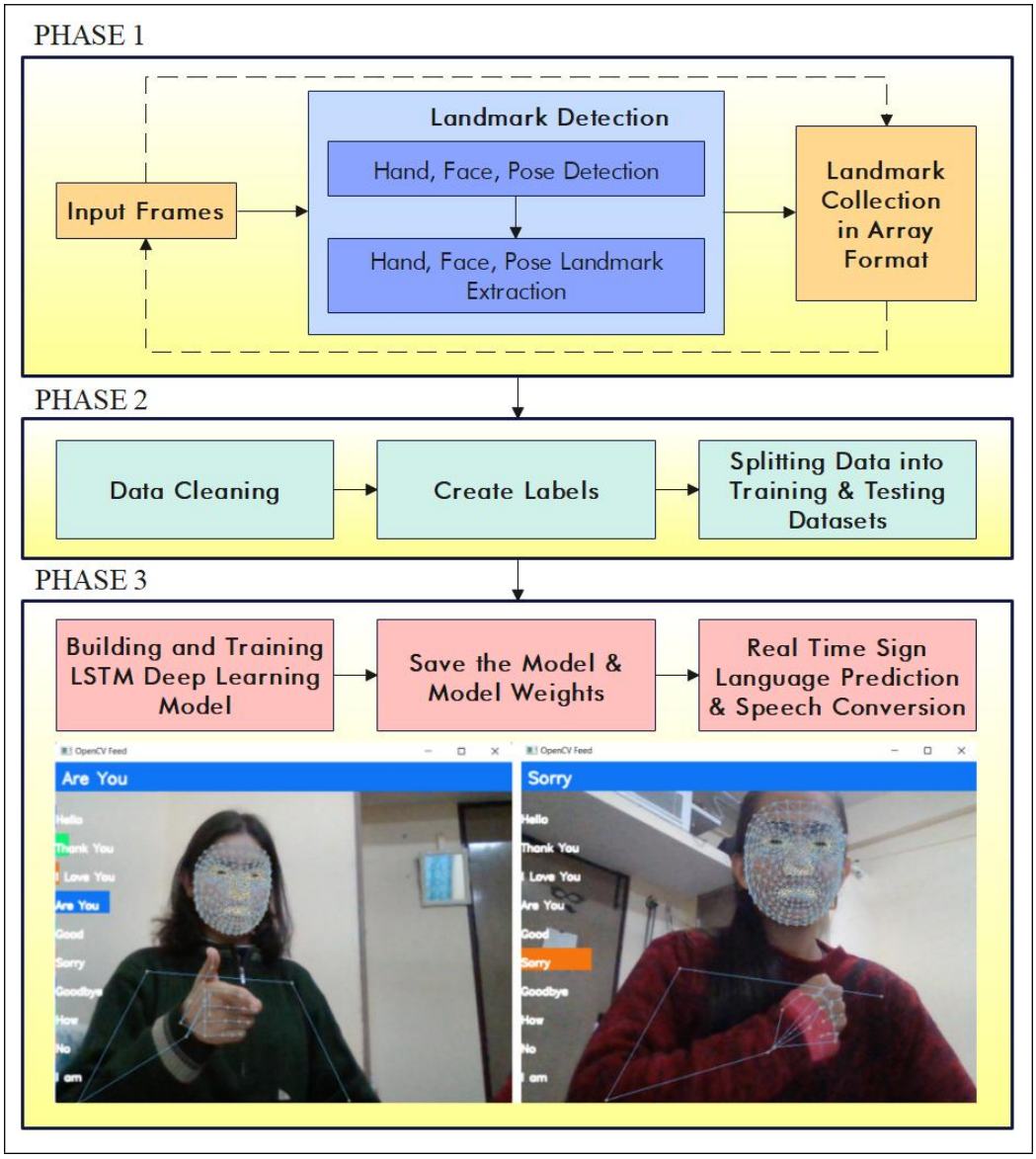


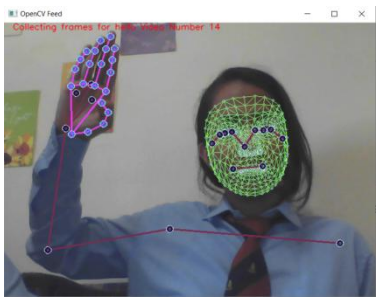
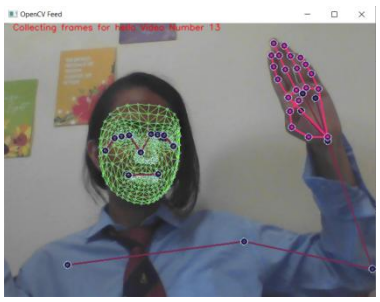
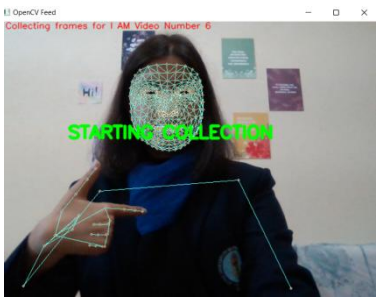
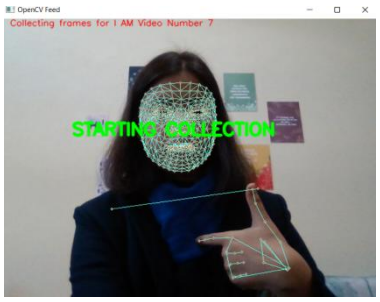
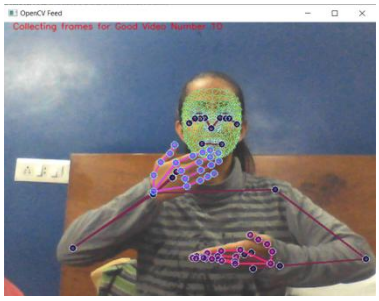
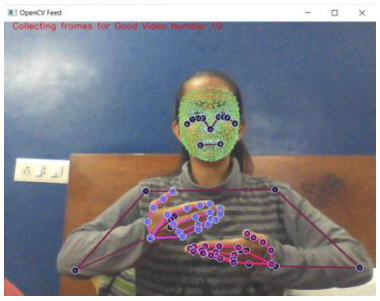
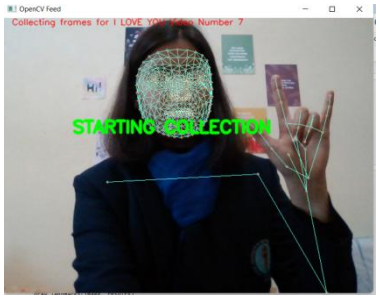
Fig. 8 - Architectural diagram of the system.

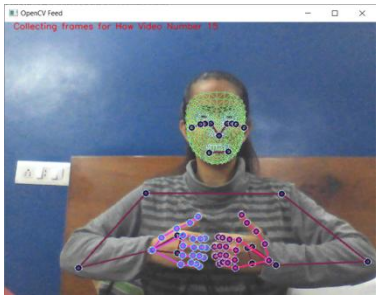
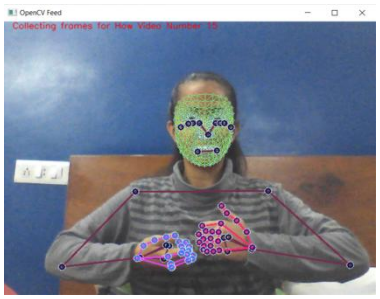
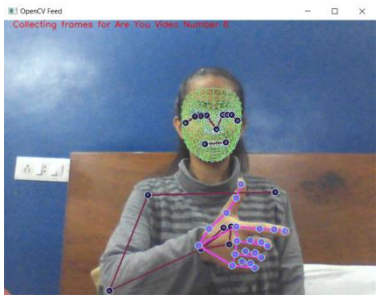
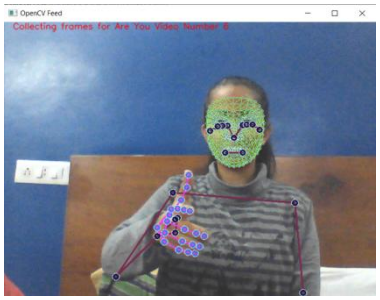
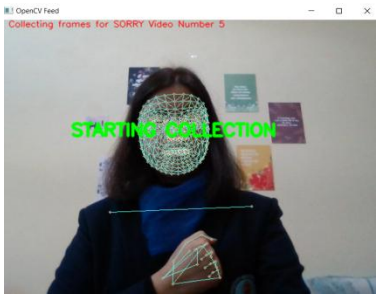
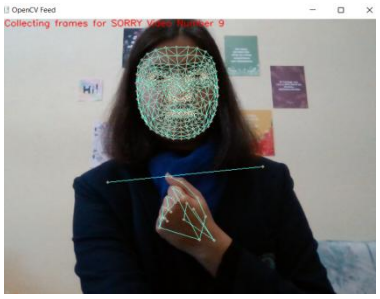
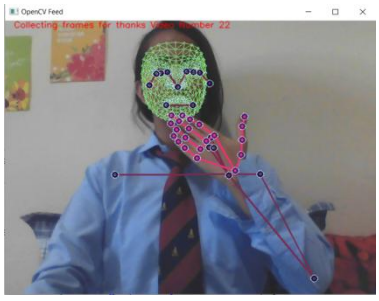
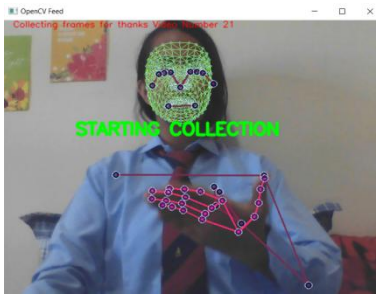
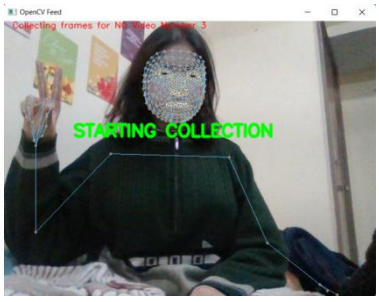
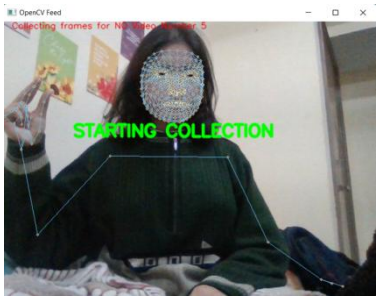
Phase 1. Data Collection:

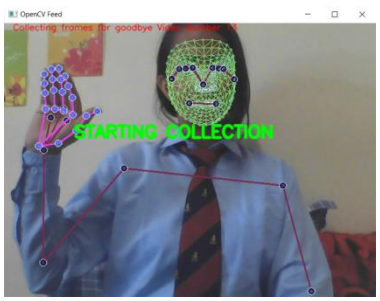
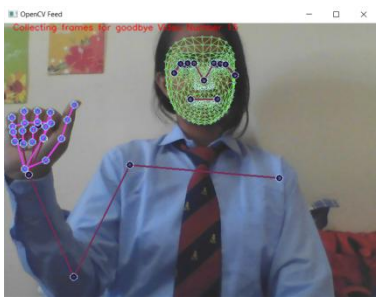
The task of data collection is an important step to initiate the recognition process. The collected data is segregated into the vocabulary considered. Subsequently, the data is pre-processed to extract crucial features which are fed to the deep learning based classifier for sign recognition. The existing data was not enough in its relevance and size, this provided the driving force behind creation of new data instances in the form of video sequences. In order to create such video input, it is necessary to decide on the

set of gestures needed for the training and testing algorithm. The gestures are chosen based on the deaf community jargon and culture. These sets of gestures formed the fundamental vocabulary for the proposed methodology. Table 3 shows the vocabulary of gestures/words chosen.

Table 3. The vocabulary of gestures / words chosen.

ACTIONS	SAMPLE 1	SAMPLE 2
HELLO		
I AM		
GOOD		
I LOVE YOU		

HOW		
ARE YOU		
SORRY		
THANK YOU		
NO		

GOODBYE		

The next step is to create input data portfolios to segregate the data to be collected according to the gestures considered. Then, the video-based data is collected from the system's in-built camera using the OpenCV [8] library. A total of 30 video sequences are captured for each gesture out of which 30 frames are extracted from each of the videos. The environment for further pre-processing and analysis of image data. Table 4 shows the list of libraries necessary to provide an environment for implementation of the sign recognition process.

Table 4. List of Libraries.

Tensorflow and Tensorflow-gpu	to run mathematical operations on CPU and GPU.
Opencv-python	to access webcam keypoints
Mediapipe	to extract hand, face, pose landmarks
Scikit-learn	for evaluation matrix as well as to leverage a training and testing split
Matplotlib	to visualize images easily
numpy	to work with different arrays & structure different datasets
os	to work with file handling to store dataset

pyplot	to visualize images easily
time	to take a sleep between each frame that we collect, so as to get enough time to get into position.
pyttsx3	It provides a cross-platform Text-to-Speech (TTS) engine. It allows developers to synthesize natural-sounding speech from text using different voices and languages.

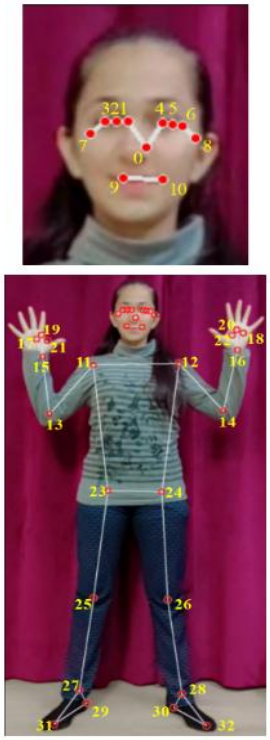
Phase 2. Feature Extraction:

i. *Keypoint Extraction:*

After the video data is collected for each gesture, by deployment of Mediapipe holistic pipeline, keypoints are extracted with respect to face, hands and pose, detailed as below:

1. *Pose landmarks:*

In this step, around 33 3D landmarks of the body consisting of coordinates x, y, and z from the input frame or image are extracted as a result of these areas of interest corresponding to a person or position are identified. The pose landmark and division masks within the ROI detect poses sequentially using the ROI-cropped frame as input. This accurately localizes more keypoints to make the gesture recognition process more efficient. Figure 9 shows the pose landmarks detected in one instance of the input frame.



0. NOSE
1. RIGHT EYE INNER
2. RIGHT EYE
3. RIGHT EYE OUTER
4. LEFT EYE INNER
5. LEFT EYE
6. LEFT EYE OUTER
7. RIGHT EAR
8. LEFT EAR
9. MOUTH RIGHT
10. MOUTH LEFT
11. RIGHT SHOULDER
12. LEFT SHOULDER
13. RIGHT ELBOW
14. LEFT ELBOW
15. RIGHT WRIST
16. LEFT WRIST
17. RIGHT PINKY KNUCKLE #1
18. LEFT PINKY KNUCKLE #1
19. RIGHT INDEX KNUCKLE #1
20. LEFT INDEX KNUCKLE #1
21. KNUCKLE #1
22. LEFT THUMB KNUCKLE #1
23. RIGHT HIP
24. LEFT HIP
25. RIGHT KNEE
26. RIGHT THUMB
27. LEFT KNEE
28. RIGHT ANKLE
29. LEFT ANKLE
30. RIGHT HEEL
31. LEFT HEEL
32. RIGHT FOOT INDEX
33. LEFT FOOT INDEX

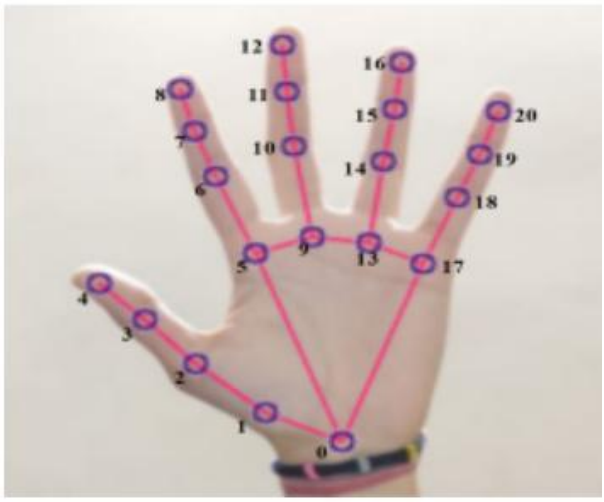
Fig. 9 - The pose_landmarks model of MediaPipe.

2. *Hand landmarks:*

Here around 21 3-dimensional hand landmarks in the form of (x,y,z) coordinates are extracted.Hand landmarks consist of two types, namely the palm keypoints and finger based keypoints as shown in Figure 10.This task of hand landmark extraction is achieved by combined implementation of two supervised based learning models. One focuses on the palm landmarks and the other localizes keypoints with respect to hand fingers. This is achieved by the Blaze Palm detector which is a part of the Mediapipe model. This environmental setup reduces the time complexity of detecting the palm by taking the focus off on unnecessary objects in the image. The result of this is a bounding box that focuses and segments the rigid parts such as palm, fist relevant for detection of palm from the other lesser important areas. Then hand keypoint localization is performed through the palm detection output.

As a result of this, following three possible outputs are produced:

- 21 hand-knuckle-points in a 2-dimensional or 3-dimensional space.
- Hand flag signifying the chances of hand presence in the input frame or image.
- Binary classification of the left hand and the right hand.



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

Fig. 10 - The hand_landmarks model of MediaPipe.

3. *Face landmarks:*

Further approximately 468 3D face landmarks are extracted instantaneously to assist in the gesture or sign recognition process as shown in Figure 11.

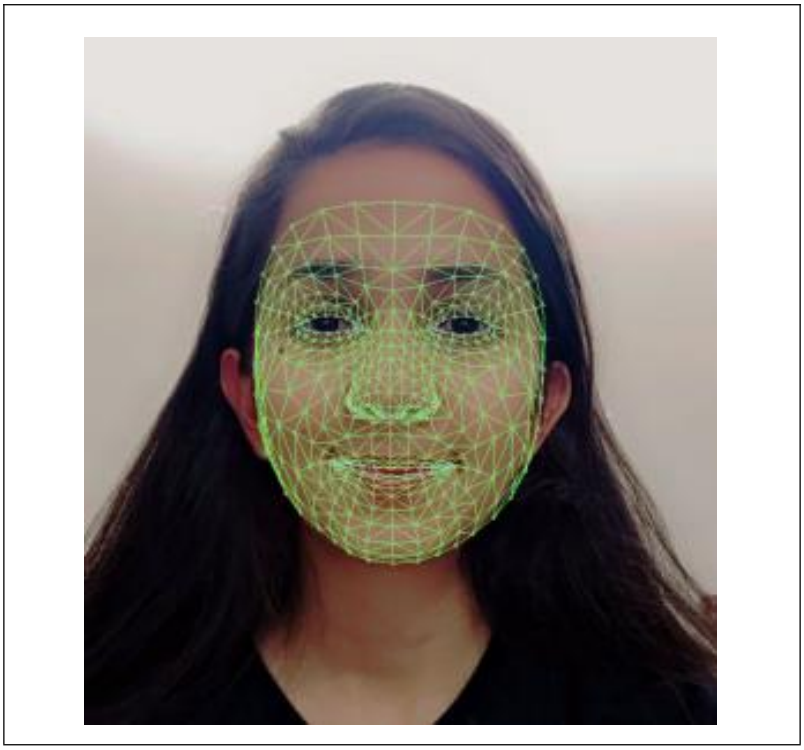


Fig. 11 - The face_landmarks model of MediaPipe.

Finally, all of the landmarks are combined to give the full 540+ landmarks. The result of the keypoint extraction task is in the form of arrays for each input frame. Each frame will contain 1662 landmark values i.e. 10*30 sequences, 30 frames, 1662 landmarks.

Number of sequences = 30 (number of videos)

Sequence length = 30 (30 frames per video)

The aggregated landmark points [21*3 (left hand) + 21*3 (right hand) + 33*4 (pose) + 468*3 (face) = 1662) are flattened and then concatenated, and put in a file to be checked for any null value entries from the data if any.

ii. Data pre-processing:

As part of data pre-processing, captured images or frames are transformed from BGR (Blue-Green-Red) to RGB (Red-Green-Blue). Then the converted images or frames are set to un-writeable file mode to save memory and reduce complexity. Data cleaning is an essential step of data pre-processing in the proposed methodology because it avoids failed feature

detection in case of blurry or hazy image frames. In this step, the null entries in the arrays are removed for amplifying the model performance and ruling out the possibility of bias. Further, the labels are constructed for every gesture or class and their associated sequences of frames are saved in order to suit the received data for subsequent stages of training phase, testing phase, and validation phase.

Phase 3. Gesture Recognition:

Real-time gesture recognition requires a learned model. To achieve this, in this phase the video data samples are taken as input to train the model. So, the data collection performed in the previous phase commences the model training. The recognition process is featured by the positional keypoints of the palm and fingers and the relative angles. The featured keypoints with respect to the palm and finger data are considered to be vital for this task. In total, 543 features(33 pose, 468 face and 21 hand landmarks) are extracted for each input word. This gives us a total of 9000 data instances, which includes 900 samples for every word in the input vocabulary. Each sample constitutes a total of 30 videos and each video consists of 30 frames. This extracted data is integrated into a npy format file. Additionally, a set of labels are created which correspond to the classes of the data samples. For the training of the collected data, deep learning based LSTM model is used.

Figure 12 shows how the data is collected and stored in different folders.

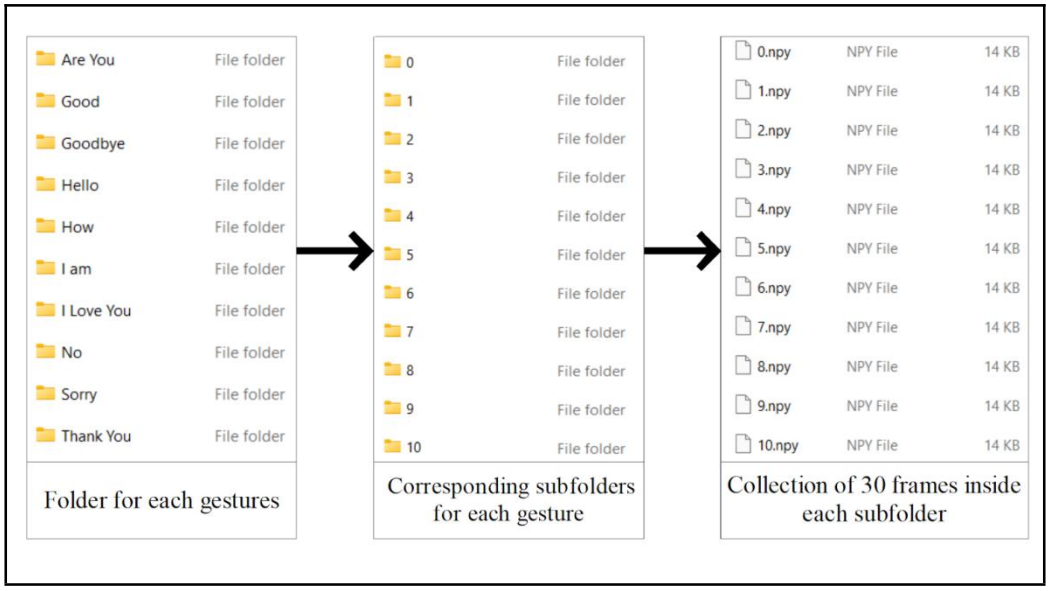


Fig. 12 - Data collection in folders.

First, The LSTM layer is chosen because of its ability to handle data over an extended period of time. LSTM units, number of epochs, and Batch size are the three variables that need to be estimated.

- Batch Size: The quantity of input data used for training each time is referred to as the batch size. Larger batch size appears to result in a machine learning model with lesser accuracy, whereas smaller batch size needs considerably more amount of training time, which is inefficient.

- Number of epochs: It denotes the total number of runs through the dataset. An evaluation is performed after each epoch and the neural network's weights are adjusted. The model should become more accurate as additional epochs are trained. Models with too many epochs trained, on the other hand, appear to be overfitting. Overfitting occurs when the model predicts data in an overly complicated manner.

- For LSTM units: It refers to the dimensionality of the LSTM output space. It also represents the number of neurons present in the layer.

To maximise performance of the model, the loss function should be chosen once the aforementioned parameters are selected. Categorical cross-entropy, a multiclass logarithmic loss, is used. The training data was used to create the suggested model. When building the model, another parameter that must be selected is the optimiser. Adam, the selected optimizer, uses gradient-based stochastic objective function optimization. Lower order moment estimates are used during operation. It differs from conventional ones in that it keeps a constant learning rate throughout the training session for all weight adjustments. However, the approach adapts different learning rates for various parameter selections by calculating the first and second moments of gradient. Adam combines the advantages of the Root Mean and Adaptive Gradient Algorithm. While Root Mean Square Propagation excels at handling non-stationary conditions, the Adaptive Gradient Algorithm excels at solving sparse gradient problems. Adam provides the benefits of both.

Adam is the best optimiser choice for the suggested model for the reasons mentioned below:

- It is computationally efficient and so requires less memory.
- It is well-suited to dealing with challenges involving massive amounts of data.
- Finally, it can handle dynamic targets as well as situations with a lot of noise.

The sequential notion is used to create the LSTM network. The first five layers are activated using the "Sigmoid" activation function. A Dense layer, which is a regular completely linked layer, would be the final layer before the output of the result. As the network's output function, a Softmax function, which is logistic regression, is frequently utilised. In multiclass classification, the log odd ratios calculated would be the probabilities of each class. As the last layer, the Dense layer is chosen to turn group predictions into class probabilities for output. After each epoch of model training, the accuracy

and loss for both training and validation are recorded. Table 5 displays the dependencies imported.

Table 5. Imported Dependencies.

Sequential Module	allow us to build a sequential neural network
LSTM Layer	provides with a temporal component to build the neural network & allows to perform action detection
Dense Layer	a normal fully connected layer
Tensorboard	allows to perform some logging inside of tensorboard if we want to go and trace and monitor our model as its training

The next step will be to train and fit the model, make predictions based on the model, save the weights, and then compare the model in real-time with the ground truth labels.

The proposed model, as shown in Figure 13, consists of 6 layers after the input layer.

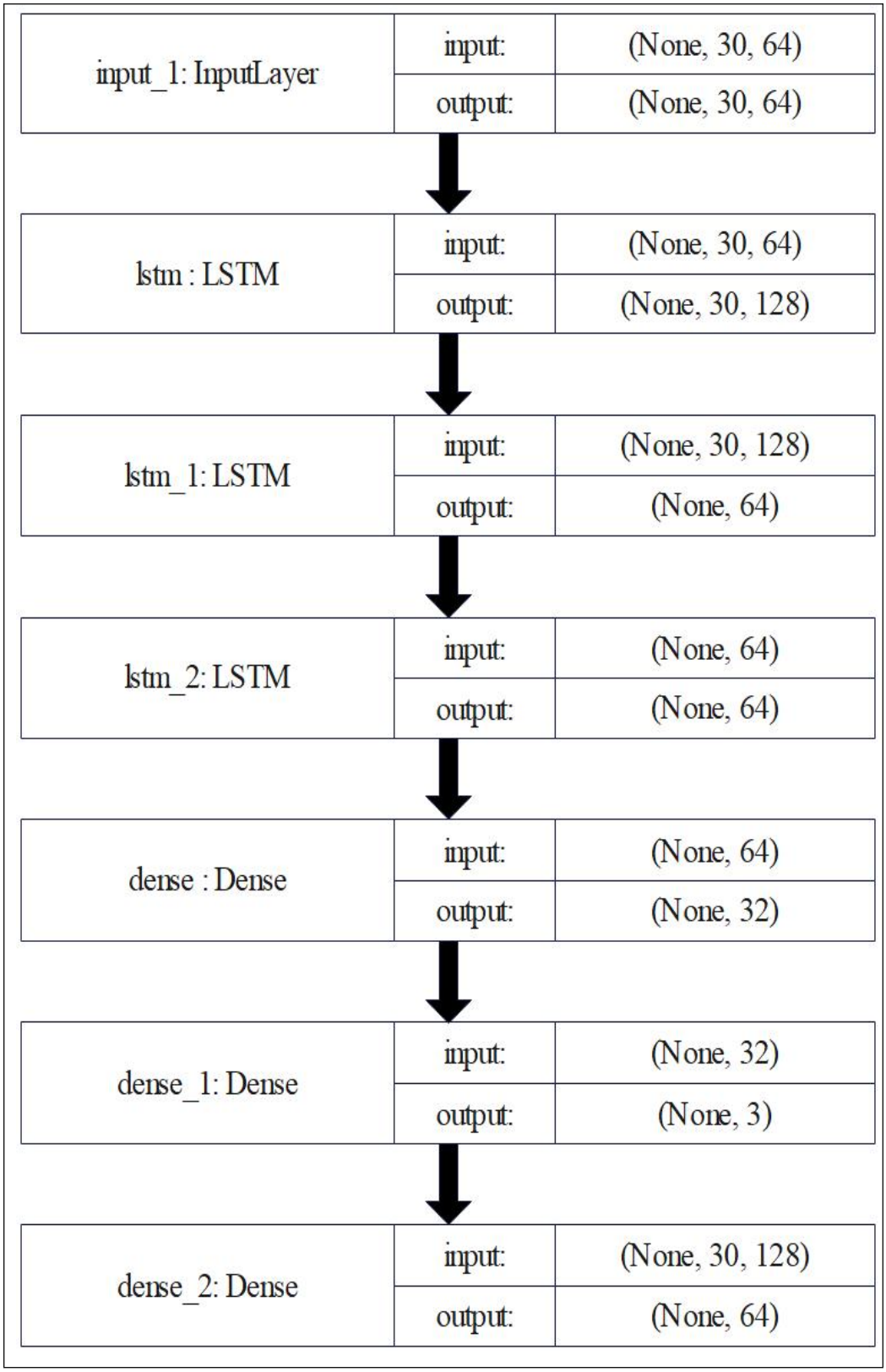


Fig. 13 - Proposed Classifier Model.

3.5 Data Preparation and Label Creation

So after the completion of the data collection process, the key points extracted from the data are then structured using data preprocessing. The data is structured in such a way that all the arrays of key points of each gesture are saved as one numpy array (X), which is then mapped to another numpy array of labels (Y). We then use the to_categorical function to convert Y into a binary class matrix, such as [1,0,0....] for hello, [0,1,0,0....] for thanks, and [0,0,1,0,0,.....] for "I love you," and so on. Following the completion of the preprocessing, the data is split into training and testing data using the train test split function.

Implementation of LSTM Long short term memory

LSTM is an RNN variant that is primarily used for data sequences or data in time-series. The LSTM unit is more complex than the RNN unit. It has gates that control the flow of data from a single unit. The loop is an essential component of the LSTM architecture because it can store previous history of information in the internal state memory. This is done to extract both spatial and temporal information from data.

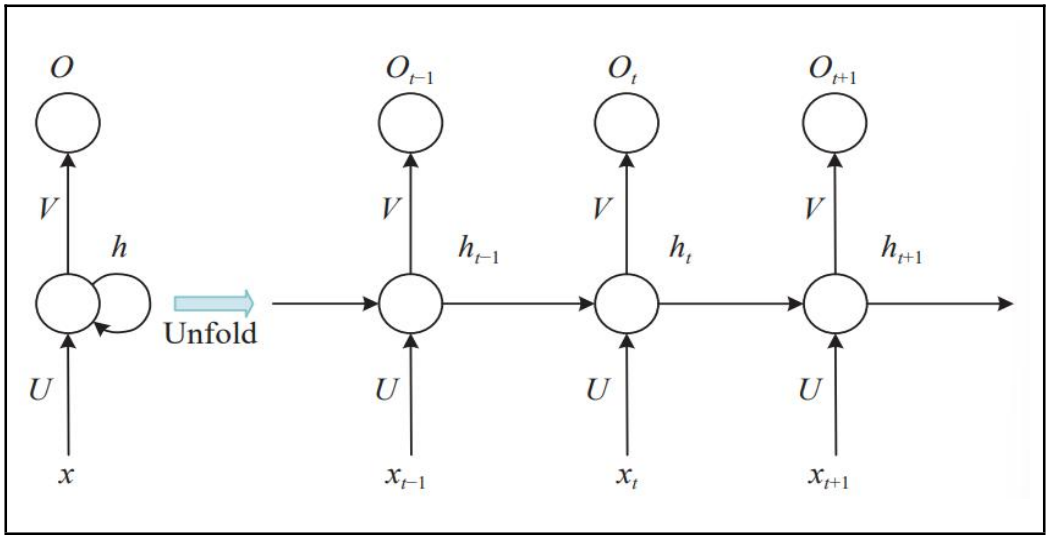


Fig. 14 - The Architecture of Recurrent Neural Network.

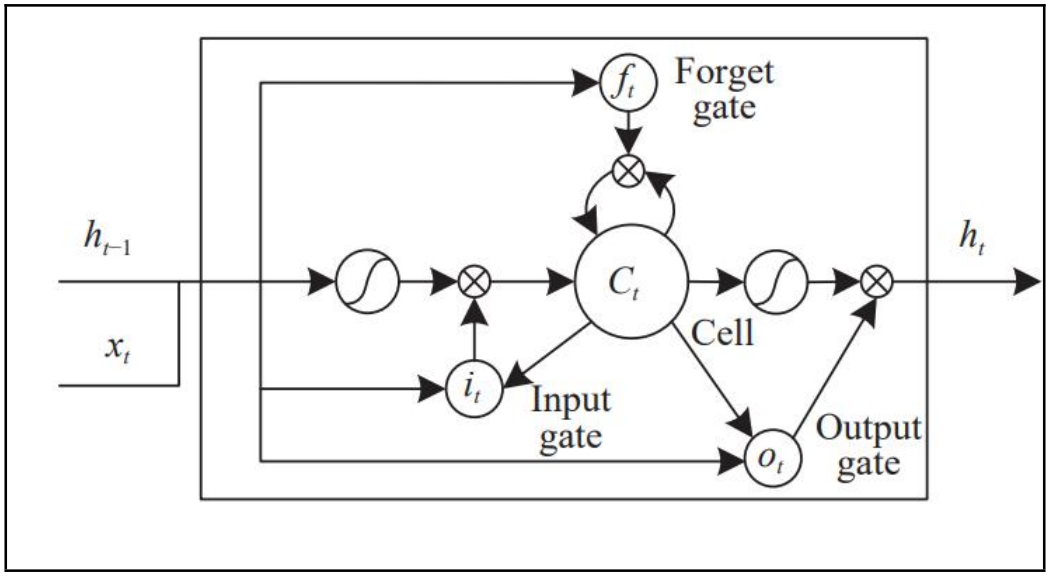


Fig. 15 - The Architecture of the LSTM block. [16]

There are 3 gates and a module called as a memory cell in a single LSTM model's architecture (see Figure 15 & Figure 16).

i. Forget Gate - Information that is no longer helpful is removed from the cell state by the forget gate. When knowledge is no longer pertinent to the context, it can be forgotten according to it.

ii. Input Gate - The purpose of the input gate is to enrich the cell state with crucial information. It is concerned with what new details can be added to or updated in our working storage state.

iii. Output Gate - It is in charge of extracting useful information from the current cell state and presenting it as output.

What section of the total data contained in the current state should be provided as the output in a particular instance? Long-term memory, which is a self-state, and short-term memory are the two states that memory cells can be in. A value between 0 and 1 is applied to each gate's operation. Any value between 0 and 1, as well as the values in between, results in no information being transmitted.

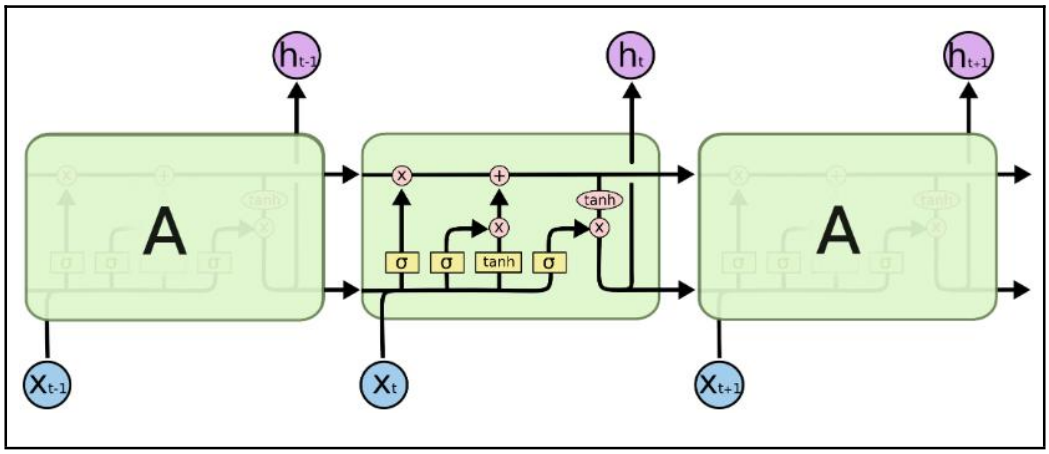


Fig. 16 - The Repeating module in an LSTM. [16]

Our information is organized as a collection of frames, each of which contains details on the locations of different landmarks that were collected. Every one of these frames is recorded as a NumPy array. With the sequential notion, the LSTM network is created. Three thick layers are the first of four layers in the LSTM. "sigmoid" activation function is applied to the first six levels. For the last dense layer, "softmax" is employed. For stochastic gradient descent, an optimizer called ADAM is employed. Throughout model training, accuracy and loss for both training and validation are recorded after each epoch.

3.6 Workflow

I. Import and Install Dependencies

Firstly, we needed to install and import some dependencies.

We have installed 6 different dependencies:

i. tensorflow *(which includes keras)*

ii. tensorflow-gpu

iii. OpenCV-Python *(to access our webcam keypoints)*

iv. mediapipe *(to extract hand, face, pose landmarks)*

v. scikit-learn *(for our evaluation matrix as well as to leverage a training and testing split)*

vi. matplotlib *(to visualize images easily)*

Then we have imported certain libraries:

i. cv2 for OpenCV
ii. NumPy *(to work with different arrays & how we structure our different datasets)*
iii. os *(to work with file handling to store our dataset)*
iv. pyplot *(to visualize images easily)*
v. time *(to take a sleep between each frame that we collect, so that we get enough time to get into position)*
vi. mediapipe

```
!pip install tensorflow==2.4.1 tensorflow-gpu==2.4.1 opencv-python mediapipe sklearn matplotlib
```

```
import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
import time
import mediapipe as mp
```

Fig. 17 - Import & Install Dependencies.

II. Detect Face, Hand and Pose Landmarks (using MP Holistic)

First we are going to make sure if we can access our webcam using OpenCV properly. All we are doing is basically setting up a video capture, then we are going to loop through every single frame and render that to the screen. So, even though we are looping through a frame, it's going to look like a video because we are just effectively stacking multiple frames together.

- cap = cv2.VideoCapture(0)
  Using this function of the OpenCV module we can access our webcam with device value = 0.
- cap.isOpen()
  Check if we are accessing our webcam to initiate a loop.

- cap.read()

  To get the current frame from the webcam. It returns 2 values: return value & the current frame.
- cv2.imshow()

  To show the output frame on to the screen
- cv2.waitKey()

  Wait for the key to be pressed from the keyboard.
- cap.release()

  To release the webcam.
- cv2.destroyAllWindows() This is going to close down our frame.

Next, we will start setting up mediapipe holistic.
In order to make detection with mediapipe, first we need to grab the image we converted from BGR to RGB. We then set it to un-writeable so that it saves a little bit of memory, then we make our detection and then set it back to writable, then convert it from RGB to BGR. This is done by defining the function:

def mediapipe_detection(image, model)

By default when we get a feed from OpenCV, it reads that feed in the channel format of BGR. But when we actually want to make a detection using mediapipe, we need it to be in the format of RGB. So, we are going to make that transformation using OpenCV.

- cv2.cvtColor()

  Converts an image from one color space to another.

  The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR).

We are now able to make our detection using mediapipe.

Next we will render these detected keypoints onto the image. As of now we are not actually visualizing the landmarks to the frame. We will do so by defining the function:

def draw_landmarks(image, results)

- mp.drawing.draw_landmarks()

III. Extract Keypoint Values

Till now we have obtained left hand landmarks, right hand landmarks, face landmarks and body-pose landmarks.

What we need to do is to extract these in a way that will be a little bit more resilient particularly if we don't get any value. So, we will concatenate these into a NumPy array and if we don't have values at a point in time we are just going to create a NumPy zeros array so that we get an array with the same shape but with zero value.

The input data used for this action detection model is a series of 30 arrays each of which contains 1662 values (30, 1662).

Each of the 30 arrays represents the landmark values (1662 values) from a single frame.

We will create a placeholder array each for left hand, right hand, face and pose. We will extract each one of different landmarks for each keypoint and update it in one flattened array, using the function:

def extract_keypoints(results)

IV. Setup Folders for Data Collection

The next thing that we are going to do is to start setting up folders for our array collection. We will be using the extracted keypoints to decode our sign language.

We have used 30 different frames of data, that is 30*1662 keypoints to be able to detect a particular action.

The key difference between action detection and other computer vision tasks is that a sequence of data rather than a single frame is used for detection.

Effectively, we have collected data for 10 different actions. We are going to collect 30 videos per action i.e. hello, thanks, I love you, etc.
Then each one of those video sequences are going to contain 30 frames of data. Each frame will contain 1662 landmark values i.e. 10*30 sequences, 30 frames, 1662 landmarks.

Number of sequences = 30 (number of videos)
Sequence length = 30 (30 frames per video)

First we created our data path, where the data is stored. Then we created our actions variable which represents each one of the different actions that we will be trying to detect.

We created a loop through the number of actions and number of sequences, and then created the folders respectively.

V. Collect Keypoint Sequences

We are going to take a break between each video that is collected.

Having breaks between each sequence collection allows us to reset and reposition our-self to collect the action from start to finish.

After taking the data input, the frames are stored as NumPy arrays at the data path location.

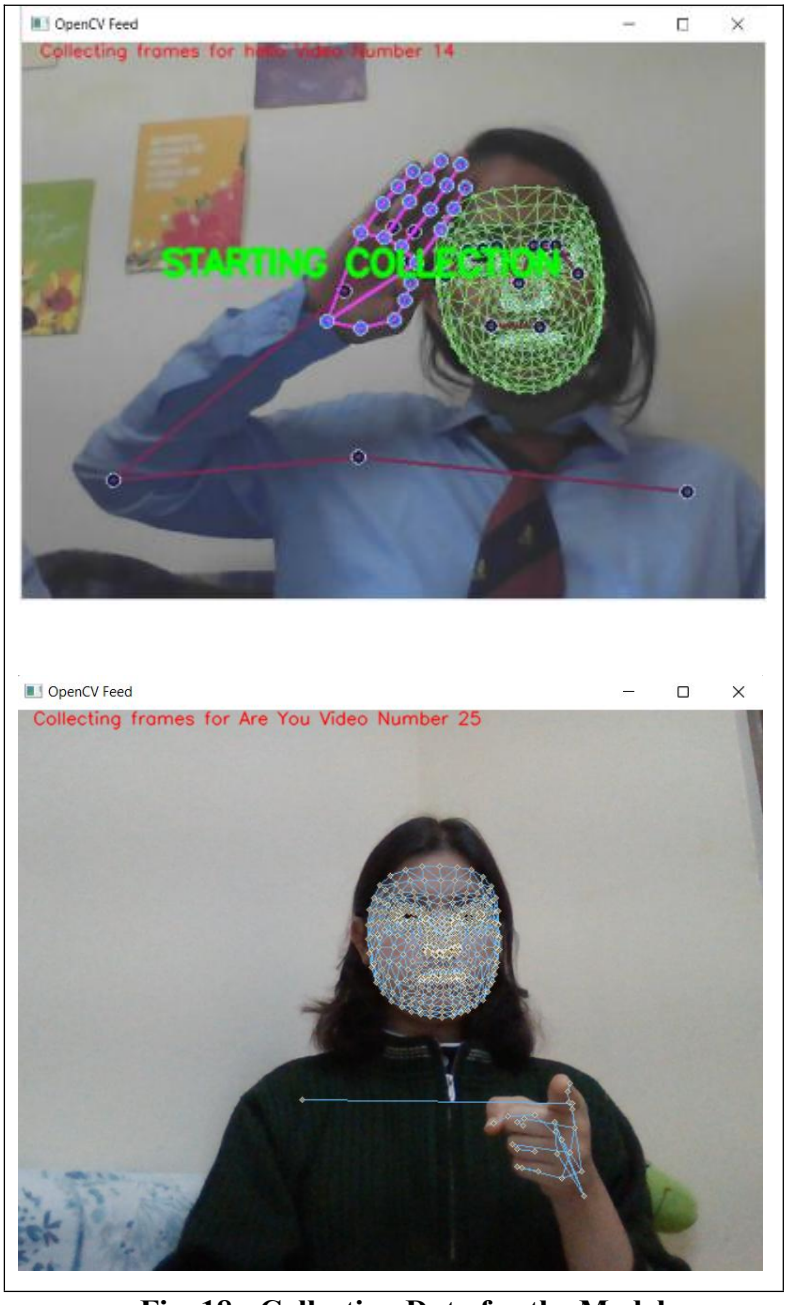


Fig. 18 - Collecting Data for the Model.

VI. Pre-process Data and Create Labels

To pre-process our data and create labels, first we are going to import 2 more dependencies:

i. train_test_split from sklearn *(allows us to partition out data into a training and a testing partition so as to train on one partition and test it out on a different segment of out data)*
ii. to_categorical functions from keras utilities *(to create labels, useful when we have to convert our data to one hot encoded data)*

Next we will create a label map to represent each one of our different actions.

Now we will start bringing all of our data together and structure it. That is, creating one big array that will contain all of our data.

So, effectively we are going to end up having 10*30 arrays with 30 frames in each one of those arrays with 1662 values which represent our keypoints.
First we will create 2 blank arrays:

i. sequences - represents our feature data or our X data
ii. labels - represent our labels or Y data

So, we are going to use the features to train a model to represent the relationship between the labels.

We are looping through each of our actions, then each of sequences and then we are creating a blank array called window which represents all of the different frames that we have got for that particular sequence.

Then we loop through each one of our frames, using numpy.load() to load up that frame.

The *sequences* array is going to have 10*30 different videos represented in it each one of those is going to be 30 frames each.

Now we start pre-processing the collected data.
We will store our sequences inside of a numpy array because that will make it easier to work with.
Next we will perform training and testing partitions using train_test_split() with test_size=0.05 i.e. 5% of our data.

VII. Build and Train LSTM Deep Learning Model

For training our LSTM neural network, we will use tensorflow and specifically keras.
First import some dependencies:

i. sequential module *(allow us to build a sequential neural network)*
ii. lstm layer *(provides us with a temporal component to build our neural network & allow us to perform action detection)*
iii. dense layer *(a normal fully connected layer)*
iv. tensorboard *(allow us to perform some logging inside of tensorboard if we want to go and trace and monitor our model as its training)*

Next we will create a log directory and set up our tensorboard callbacks.
Build the neural network architecture, compile the model and then fit it. We have used *adam* optimizer, categorical_crossentropy loss function as we have multi class classification model to compile the model.

Next we will fit and train our model.

VIII. Make Model Predictions on Training & Testing data

Using function:

model.predict(X_test)

XI. Save Model Weights

Using function:

model.save()

X. Evaluation using a Confusion Matrix and Accuracy

For this we will import a couple of metrics from sci-kit learn to evaluate the performance of our model:

- Multi_label_confusion_matrix *(gives us a confusion matrix for each one of our different labels. This allows us to evaluate what's being detected as a True Positive and a True Negative & what's being detected as a False Positive and a False Negative)*
- accuracy_score *(to evaluate accuracy)*

XI. Test in Real Time

The saved model is loaded and executed in real time.

The input video is taken in the form of frames using the OpenCV python module & then the sign language prediction is performed with the help of the trained model after keypoint detection & extraction using mediaPipe holistic.

Next we will perform text to speech conversion on the the text that is predicted from the sign language input in the last step using python pyttsx3 module, using function:

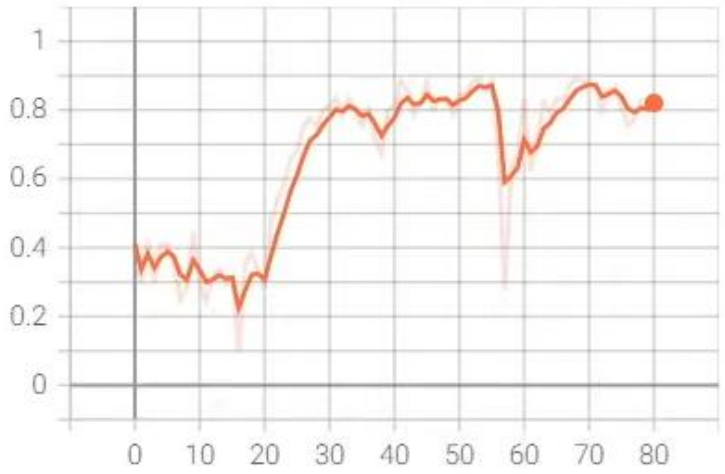
engine.say(text)

4.PERFORMANCE ANALYSIS

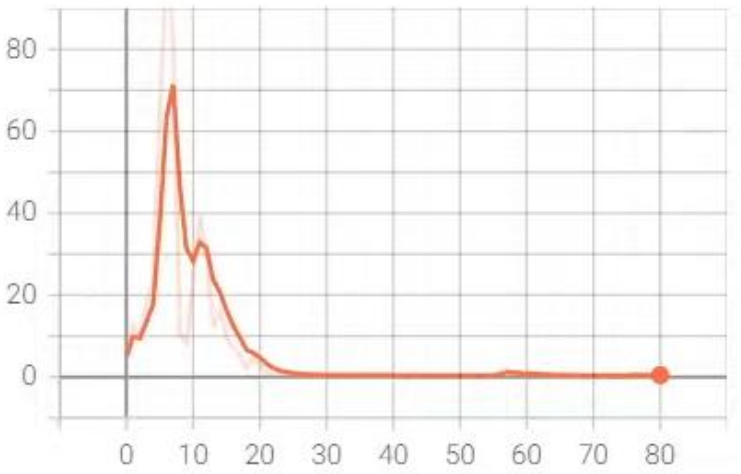
4.1 Testing on validation data

The dataset used for testing was taken from the original dataset where it was divided between training and testing data. The evaluation metrics we use is Accuracy. The LSTM model is working successfully(see figure).

Accuracy of 80% was achieved with the test dataset.



Graph 1. Epoch Categorical accuracy



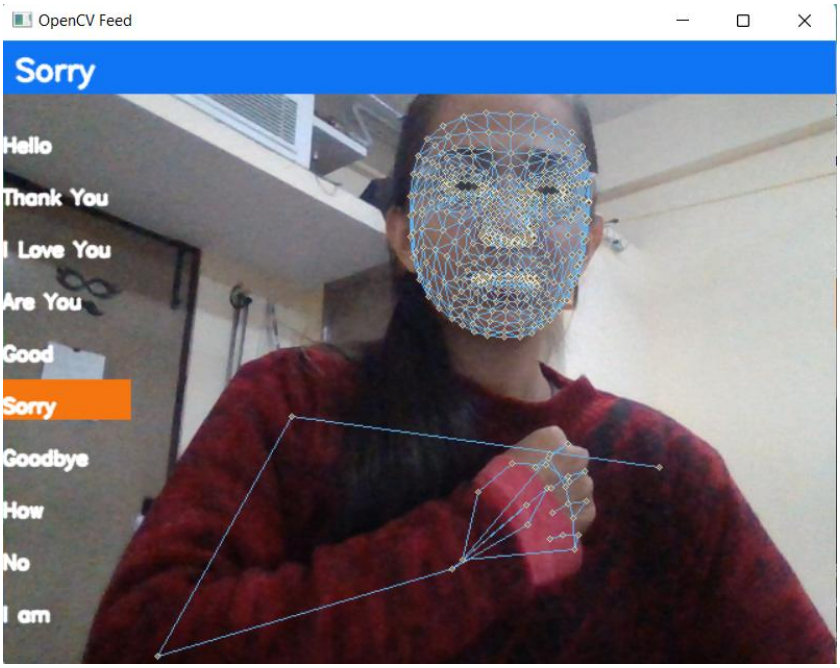
Graph 2. Epoch Loss

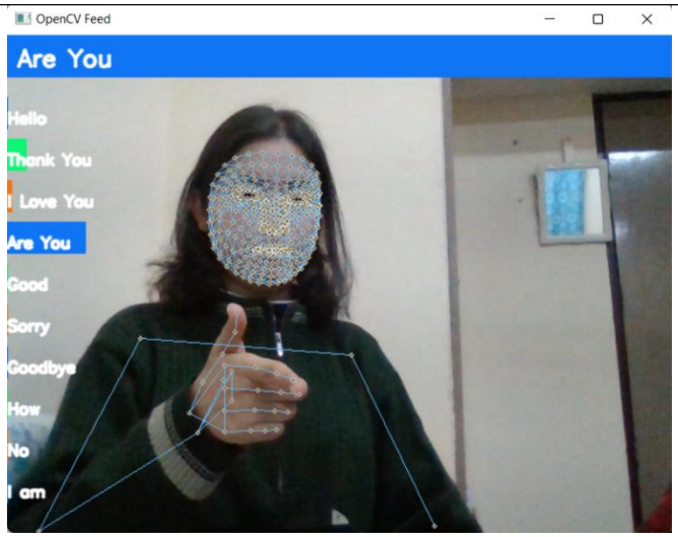
4.2 Testing in Real Time

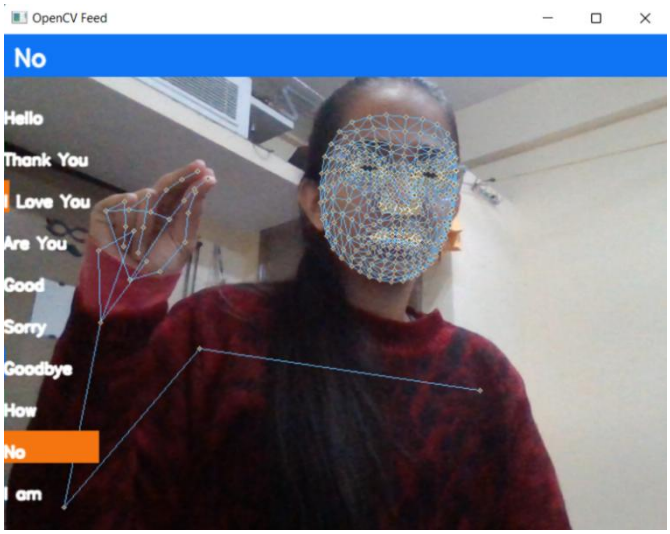
Using live feed, the recognition in the form of text was shown for gestures. Real time testing was done on each gesture with promising results. As our model is trained to perform the prediction on 30 frames worth of key points, thus in order to enable it in real-time the video feed is processed in the following manner: An empty array: sequence = [] is defined. The real-time feed is captured using open CV, and various methods that have been explained earlier in this paper such as: mediapipe_detection and extract_keypoints are called upon to perform the detection and collect the key points in real time. The collected key points are appended to the empty array we defined in the beginning. The array is then assigned as: sequence = [-30:].

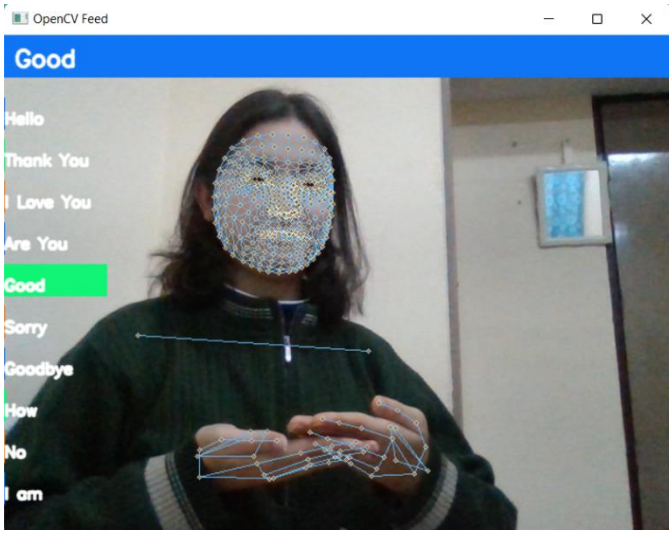
Thus, it saves the last 30 frames worth of key points appended, When the length of the sequence array is determined as 30, then the model is called upon to predict the output.

Figure 19 shows the output screens after testing in real time.




OpenCV Feed
Are You
Hello
Thank You
Love You
Are You
Good
Sorry
Goodbye
How
No
I am


OpenCV Feed
No
Hello
Thank You
Love You
Are You
Good
Sorry
Goodbye
How
No
I am


OpenCV Feed
Good
Hello
Thank You
Love You
Are You
Good
Sorry
Goodbye
How
No
I am

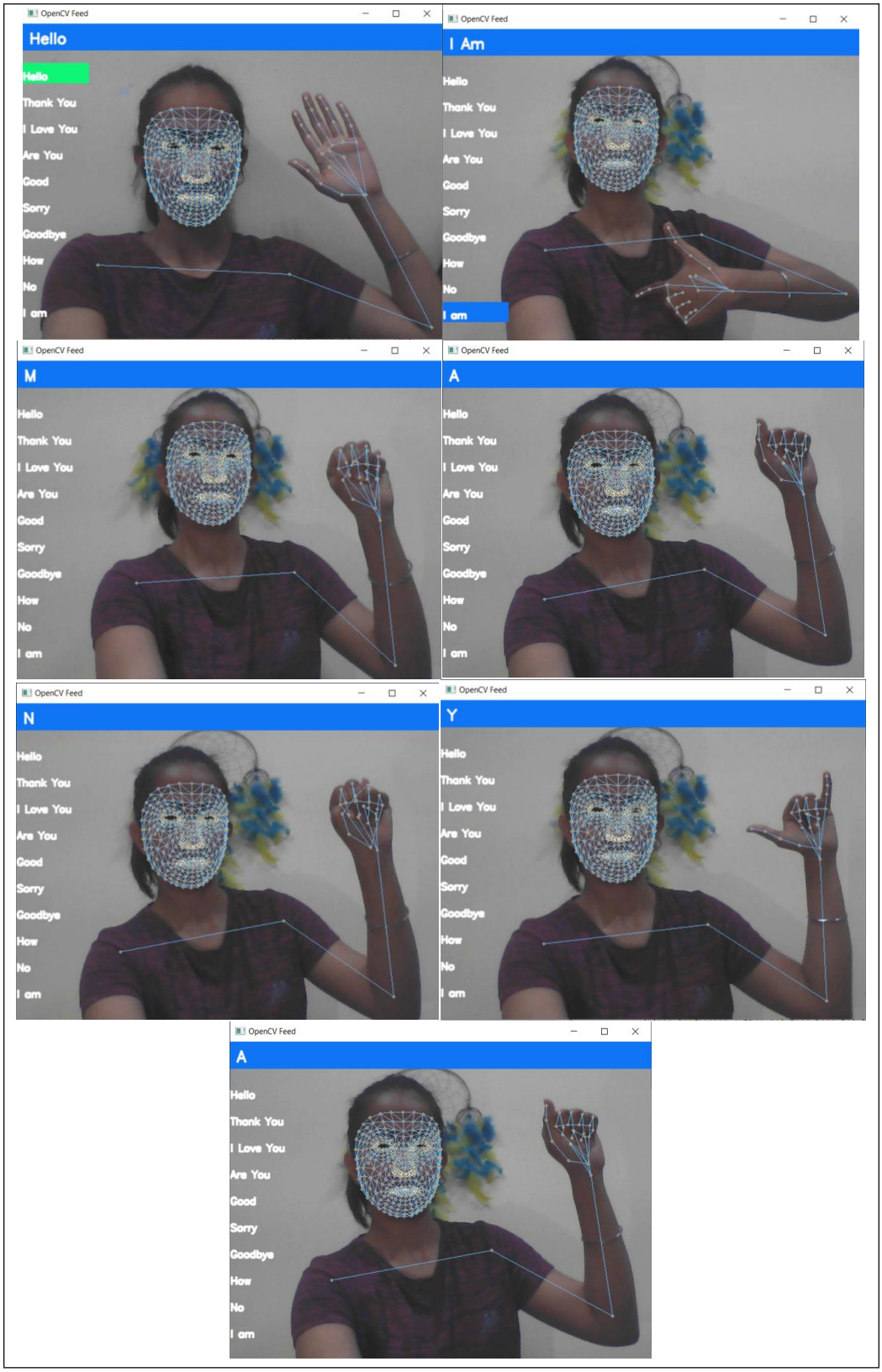


Fig 19. Testing in Real Time.

CONCLUSION

5.1 Conclusion

Using the popular MediaPipe and LSTM, this comprehensive study proposed and developed a system for American Sign Language recognition. Initially, a folder was created for gestures, and for each gesture, 30 subfolders were created; these subfolders can be thought of as video folders, and each subfolder contains 30 frames, each of which is in the form of a numpy array containing landmark values detected and extracted using Mediapipe Holistic Solution. The data was used to train the LSTM network, which yielded an accuracy of 80% on testing data. Finally, the system was tested using real-time data that was directly fed into the model, and the results for each gesture were displayed on the screen and the text is translated into speech. There was some lag when recognising gestures in real time. We learned about how sometimes basic approaches work better than complicated approaches. We also realised the time constraints and difficulties of creating a dataset from scratch.

5.2 Future Scope

- We can develop a complete product that will help speech and hearing-impaired people, and thereby reducing the communication gap. By putting the entire system online, allowing any user to utilize their camera to quickly recognize the gesture.

- Add a lot more dynamic gestures that are used frequently, as well as more data for each of those motions. The model is currently trained on a small number of words, but it may be expanded to train on full sentences, alphabets, and integers. Training with various skin tones, hand postures, lighting, and environmental factors for best results.

- Integrating the system with the mobile device. Building an application for mobile which will make it possible to use in real-time. Allowing the user to choose the language to be recognized at any time whenever needed.

5.3 Applications Contributions

ASL is a language that is used primarily by people with disabilities. This language uses hand gestures and facial expressions to convey information. Because of this, it is sometimes difficult for people who are not deaf to understand ASL. In the paper “ASL Recognition Using Deep Learning” by B. Li et al, the authors discuss the work of using deep learning to recognize and analyse ASL data. The paper also presents an implementation of this technique in a real application scenario using Google Glass.

They developed a system that can recognize signs from videos recorded by the Glass and translate the signs into English text in real-time. This allows the user to identify the sign and then obtain a translation via a voice message.

REFERENCES

[1] K. Cheng, “Top 10 & 25 American sign language signs for beginners – the most know top 10 & 25 ASL signs to learn first: Start ASL,” Start ASL Learn American Sign Language with our Complete 3-Level Course!, 29-Sep-2021. [Online]. Available:
Top 10 & 25 American Sign Language Signs for Beginners – The Most Know Top 10 & 25 ASL Signs to Learn First.

[2] A. Mittal, P. Kumar, P. P. Roy, R. Balasubramanian, and B.B. Chaudhuri, "A Modified LSTM Model for Continuous Sign Language Recognition Using Leap Motion," in IEEE Sensors Journal, vol. 19, no. 16, pp. 7056-7063, 15 Aug.15, 2019.

[3] “Real time sign language detection with tensorflow object detection and Python | Deep Learning SSD,” YouTube, 05-Nov-2020. [Online]. Available: https://www.youtube.com/watch?v=pDXdlXlaCco&t=1035s.

[4] V. Sharma, M. Jaiswal, A. Sharma, S. Saini and R. Tomar, "Dynamic Two Hand Gesture Recognition using CNN-LSTM based networks," 2021 IEEE International Symposium on Smart Electronic Systems (iSES), 2021,pp. 224-229, doi: 10.11.09.

[5] K. Amrutha and P. Prabu, "ML Based Sign Language Recognition System," 2021 International Conference on Innovative Trends in Information Technology (ICITIIT), 2021, pp. 1-6, doi: 10.1109/ICITIIT51526.2021.9399594. Available:
ML Based Sign Language Recognition System | IEEE Conference Publication

[6] M. Wurangian, “American sign language alphabet recognition,” Medium, 15-Mar-2021. [Online]. Available:
American Sign Language Alphabet Recognition | by Marshall Wurangian | MLearning.ai | Medium.

[7] A. Dennisan, “American sign language alphabet recognition using Deep Learning,” ArXiv, 10-Feb-2022. [Online]. Available:
American Sign Language Alphabet Recognition using Deep Learning.

[8] OpenCV (2022) Wikipedia. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/OpenCV.

[9] P. Likhar, N. K. Bhagat and R. G N, "Deep Learning Methods for Indian Sign Language Recognition," 2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin), 2020, pp. 1-6.

[10] Scikit Learn - Documentation
Scikit-learn

[11] K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep Learning and Computer Vision," 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 4896-4899, doi: 10.1109/BigData.2018.8622141. Available:
American Sign Language Recognition using Deep Learning and Computer Vision.

[12] Shivashankara, Ss, and S. Srinath. “American sign language recognition system: an optimal approach.” International Journal of Image, Graphics and Signal Processing 11, no. 8 (2018): 18.

[13] N. K. Bhagat, Y. Vishnusai and G. N. Rathna, ”Indian Sign Language Gesture Recognition using Image Processing and Deep Learning,” 2019 Digital Image Computing: Techniques and Applications (DICTA), Perth, Australia, 2019, pp. 1-8, doi: 10.1109/DICTA47822.2019.8945850.

[14] Q. Wu, Y. Liu, Q. Li, S. Jin and F. Li, “The application of deep learning in computer vision,” 2017 Chinese Automation Congress (CAC),Jinan, 2017, pp. 6522-6527

[15] D. G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision, vol. 13, no. 2, pp. 111122, 1981.

[16] S. Agrawal, A. Chakraborty, and C.M. Rajalakshmi, “ Real-Time Hand Gesture Recognition System Using MediaPipe and LSTM”, https://ijrpr.com/uploads/V3ISSUE4/IJRPR3693.pdf.

APPENDICES

1. OpenCV

OpenCV (Open-Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

2. TensorFlow

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google brain team for internal Google use. It was released under the Apache 2.0 open source library on November 9, 2015. TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.