Sign Language Recognition using Landmark Detection, GRU and LSTM

Project report in partial fulfillment of the requirement for the award of the degree
of
Bachelor of Technology
in
COMPUTER SCIENCE & ENGINEERING

Submitted By

Sukanta Dinda University Roll No. 12018009019372

Under the guidance of

Prof. (Dr). Subhalaxmi Chakraborty

&

Prof. (Dr). Panchali Dutta Chowdhury

Department of Computer Science & Engineering

UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA

University Area Plot No.III - B/5, New Town Action Area - III, Kolkata - 700160

CERTIFICATE

This is to certify that the project titled **Sign Language Recognition using Landmark Detection, LSTM and GRU** submitted by **SUKANTA DINDA (University Roll No. 12018009019372)** student of **UNIVERSITY OF ENGINEERING & MANAGEMENT, KOLKATA**, in partial fulfillment of requirement for the degree of Bachelor of Computer Science & Engineering is a bona fide work carried out by them under the supervision and guidance of **Prof. (Dr.) Subhalaxmi Chakraborty & Prof. (Dr.) Panchali Dutta Chowdhury** during 8th Semester of academic session of 2018-2019. The content of this report has not been submitted to any other university or institute. I am glad to inform you that the work is entirely original and its performance is found to be quite satisfactory.

Signature of Guide

Signature of Guide

Signature of Head of the Department

ACKNOWLEDGEMENT

We would like to take this opportunity to thank everyone whose cooperation and encouragement throughout the ongoing course of this project remains invaluable to us.

We are sincerely grateful to our guide Prof. (Dr.) Subhalaxmi Chakraborty and Prof. (Dr.) Panchali Dutta Chowdhury of the Department of Computer Science & Engineering, UEM, Kolkata, for her wisdom, guidance and inspiration that helped us to go through with this project and take it to where it stands now.

Last but not the least, we would like to extend our warm regards to our families and peers who have kept supporting us and always had faith in our work.

SUKANTA DINDA

TABLE OF CONTENTS

SIGN LANGUAGE RECOGNITION USING LANDMARK DETECTION, GRU and LSTM

UNIVERSITY OF ENGINEERING AND MANAGEMENT KOLKATA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

---------- ***----------

ABSTRACT:

Speech impairment is a disability which affects an individual's ability to communicate using speech and hearing. People who are having this hearing problem use sign language to communicate. Although sign language is quite common in recent times, there is always a communication gap between non-sign language speakers and signers. To overcome this complexity researchers are trying to develop systems using deep learning. The main objective of this project is to create a vision-based application which offers sign language translation to text and voice message to bridge the communication gap between signers and non-signers. The proposed model takes video sequences and extracts temporal and spatial features from them. For extracting spatial features, we have used MediaPipe Holistic, which consists of several solutions for the detecting hand, face and pose landmarks. We have used **GRU (Gated Recurrent Unit) and LSTM (Long Short Term Memory)** both of which are variations of **RNN (Recurrent Neural Network)** to train on temporal features. By using both of these methods, we achieved an **accuracy of 99% on our test dataset**. The sign language convention used here is American Sign Language. The experimental results show that the recognition method with **MediaPipe Holistic** followed by GRU or LSTM can achieve a high recognition rate which will meet the needs of a real-time Sign Language Recognition system. Overall, it is expected that this study will facilitate knowledge accumulation and creation of intelligent-based Sign Language Recognition systems and provide a roadmap to guide future direction.

Key Words: Sign Language Recognition, MediaPipe, landmark detection, Recurrent Neural Network, Long Short-Term Memory, Gated Recurrent Unit

---------- *** ----------

CHAPTER 1
INTRODUCTION

Sign language is a necessary communication tool for deaf-mute people. It is composed of complicated hand shape, movement, location of hand, facial expression, etc. People with impaired speech and hearing use Sign language as a tool of non-verbal communication to express their own emotions and thoughts to other common people. But these common people find it difficult to understand their expression, thus trained sign language experts are needed during medical and legal appointments, educational and training sessions. Over the past few years, there has been an increase in demand for these services. Other forms of services such as video remote human interpretation using the high-speed Internet connection, has been introduced, thus these services provide an easy-to-use sign language interpreter service, which can be used and benefited, yet have major limitations such as accessibility to the high-speed Internet and an appropriate device.

To address this, we use a landmark recognition pipeline named MediaPipe Holistic followed by GRU and LSTM to recognize gestures in sign language. We use a custom recorded American Sign Language dataset based on an existing list of videos available on the Internet [1] for training the model to recognize gestures. The dataset has 10 different gestures performed multiple times giving us variation in context and video conditions. For simplicity, the videos are recorded at a common frame rate and have the same number of frames. We propose to use MediaPipe Holistic to extract spatial features and detect landmarks from the video stream for Sign Language Recognition (SLR). Then by using GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory), two variations of RNN (Recurrent Neural Network), we can extract temporal features from the video sequences. We have compared the performance of GRU with that of LSTM.

We have used OpenCV for capturing and displaying video frames and performing sign language recognition in real-time. OpenCV is a library of programming functions mainly aimed at real-time computer vision.

The entire project has been done using Python programming language.

CHAPTER 2
LITERATURE REVIEW

Tanuj Bohra et al. [2] proposed a real-time two-way sign language communication system built using image processing, deep learning and computer vision. Techniques such as hand detection, skin color segmentation, median blur and contour detection are performed on images in the dataset for better results. CNN model trained with a large dataset for 40 classes and was able to predict 17600 test images in 14 seconds with an accuracy of 99%.

Joyeeta Singha and Karen Das [3] proposed a system for indian sign language recognition from a live video. The system comprises three stages. Preprocessing stage includes skin filtering and histogram matching. Eigenvalues and eigenvectors are being considered for feature extraction stage and Eigen value weighted euclidean distance for classification. Dataset consisted of 480 images of 24 signs of ISL signed by 20 people. System was tested on 20 videos and achieved an accuracy of 96.25%.

Muthu Mariappan H. and Dr. Gomathi V. [4] designed a real time sign language recognition system as a portable unit using contour detection and fuzzy c-means algorithm. Contours are used for detecting face, left and right hand. While a fuzzy c-means algorithm is used to partition the input data into a specified number of clusters. System was implemented on a dataset that contained videos recorded from 10 signers for several words and sentences. It was able to achieve an accuracy of 75%.

Salma Hayani et al. [5] proposed an arab sign language recognition system based on CNN, inspired from LeNet-5. Dataset contained 7869 images of arab signs of numbers and letters. Various experiments were conducted by varying the number of training sets from 50% to 80%. 90% accuracy was obtained with 80% training dataset. The author has also compared the results obtained with machine learning algorithms like KNN (k-nearest neighbor) and SVM (support vector machine) to show performance of the system. This model was purely image based and it can be extended to video based recognition.

Kshitij Bantupalli and Ying Xie [6] worked on an American sign language recognition system which works on video sequences based on CNN, LSTM and RNN. A CNN model named Inception was used to extract spatial features from frames, LSTM for longer time dependencies and RNN to extract temporal features. Various experiments

were conducted with varying sample sizes and the dataset consists of 100 different signs performed by 5 signers and maximum accuracy of 91% was obtained. Sequence is then fed to a LSTM for longer time dependencies. Outputs of softmax layer and max pooling layer are fed to RNN architecture to extract temporal features from softmax layer.

G. Anantha Rao et al. [9] proposes an indian sign language gesture recognition using convolutional neural network. This system works on videos captured from a mobile's front camera. Dataset is created manually for 200 ISL signs. CNN training is performed with 3 different datasets. In the first batch, dataset of only one set is given as input. Second batch has 2 sets of training data and third batch respectively has 3 sets of training data. Average recognition rate of this CNN model is 92.88%.

Real Time Detection And Recognition Of Indian And American Sign Language Using Sift In [10]: Author proposed a real time vision based system for hand gesture recognition for human computer interaction in many applications. The system can recognize 35 different hand gestures given by Indian and American Sign Language or ISL and ASL at faster rate with virtuous accuracy. RGB-to-GRAY segmentation technique was used to minimize the chances of false detection. Authors proposed a method of improvised Scale Invariant Feature Transform (SIFT) and same was used to extract features. The system is model using MATLAB. To design and efficient user friendly hand gesture recognition system, a GUI model has been implemented.

A Review on Feature Extraction for Indian and American Sign Language in [11]: Paper presented the recent research and development of sign language based on manual communication and body language. Sign language recognition system typically elaborate three steps pre processing, feature extraction and classification. Classification methods used for recognition are Neural Network (NN), Support Vector Machine (SVM), Hidden Markov Models (HMM), Scale Invariant Feature Transform (SIFT), etc.

The literature review shows that there have been different approaches to this problem within neural networks itself. The input feed to the neural networks plays a big role in how the architecture of the network is shaped, such is a 3DCNN model would take RGB input along with the depth field. Lu et al. [12] used a general CNN network to extract spatial features and used an LSTM to extract sequence features. Vivek et al. [13] used CNN models with RGB inputs for their architecture. The authors of [13] worked on American Sign Language with a custom dataset of their own making. The

architecture in [12] was a pretrained CNN called ResNet along with a custom LSTM of their 8 design whereas [13] used a CNN for stationary hand gestures.

CHAPTER 3
PROBLEM STATEMENT AND DATASET DESCRIPTION

The **objective** is to apply Deep Learning algorithms to detect, in real time, what is being conveyed using American Sign Language by a signer. The program should be able to detect static signs as well as signs which require motion. The input will be obtained from a camera focused directly on the signer providing a front view. The sequence of frames will be run processed and the resultant output will be displayed and played.

The **dataset** used here was obtained by first recording videos of ourselves using 10 common words and phrases in American Sign language, they are "hello", "I love you", "thank you", "food", "friend", "forget", "again", "me", "want" and "please". There is also an additional category "no action", in which the signer shows no signs or the signer is not present in the frame at all. Each word was individually recorded for a minimum of 90 times with subtle variations in distance from the camera and angle of the camera, speed of signing, etc. These variations were introduced to imitate practical real-time scenarios. The videos were recorded using a HP-eq0500au laptop webcam with a resolution of 640 X 480 at 30 frames per second (fps). For simplicity, the length of each video was capped at 30 frames.

Each frame of a video was fed into the MediaPipe holistic pipeline to get face, pose and hand landmarks. Each of the face and hand landmarks consists of the following:

- x and y: Landmark coordinates normalized to [0.0, 1.0] by the image width and height respectively.
- z: Represents the landmark depth with the depth at the midpoint of hips being the origin, and the smaller the value the closer the landmark is to the camera. The magnitude of z uses roughly the same scale as x.

In addition to these, each pose landmark consists of a visibility attribute, which is a value in [0.0, 1.0] indicating the likelihood of the landmark being visible (present and not occluded) in the image.

For each frame, 468 face landmarks and 33 pose landmarks were produced. For each hand 21 landmarks were produced, provided that the particular hand was in the frame. If a particular hand was not in the frame, then MediaPipe holistic returns no landmarks for that hand. We designed the programming logic such that if no hand is present in the frame, we get zeros for the coordinates of each hand landmark. To feed these

landmarks into the LSTM model, we wrote a function extract_keypoints() which first flattens and concatenates each category (face, pose, left hand, right hand) of landmarks into 4 numpy arrays. It then concatenates the four numpy arrays into a **single numpy of length 1662** (i.e., 4✕33 + 3✕468 + 3✕21✕2). All the 30 frames of a video were processed like this and the resultant numpy arrays were appended. Hence **from each video, we got a numpy array of shape (30, 1662)**. We recorded a total of 1600 videos which includes 90 videos for the "no action" category (meaning no signs are being shown), giving us a numpy **array of shape (1600, 30, 1662) as our dataset**. For some signs we had to record more videos, in order to reduce overfitting and improve the performance of the model.

CHAPTER 4
PROPOSED SYSTEM

In this section, we present our LSTM based neural network architecture for continuous SLR using landmark detection with MediaPipe Holistic.

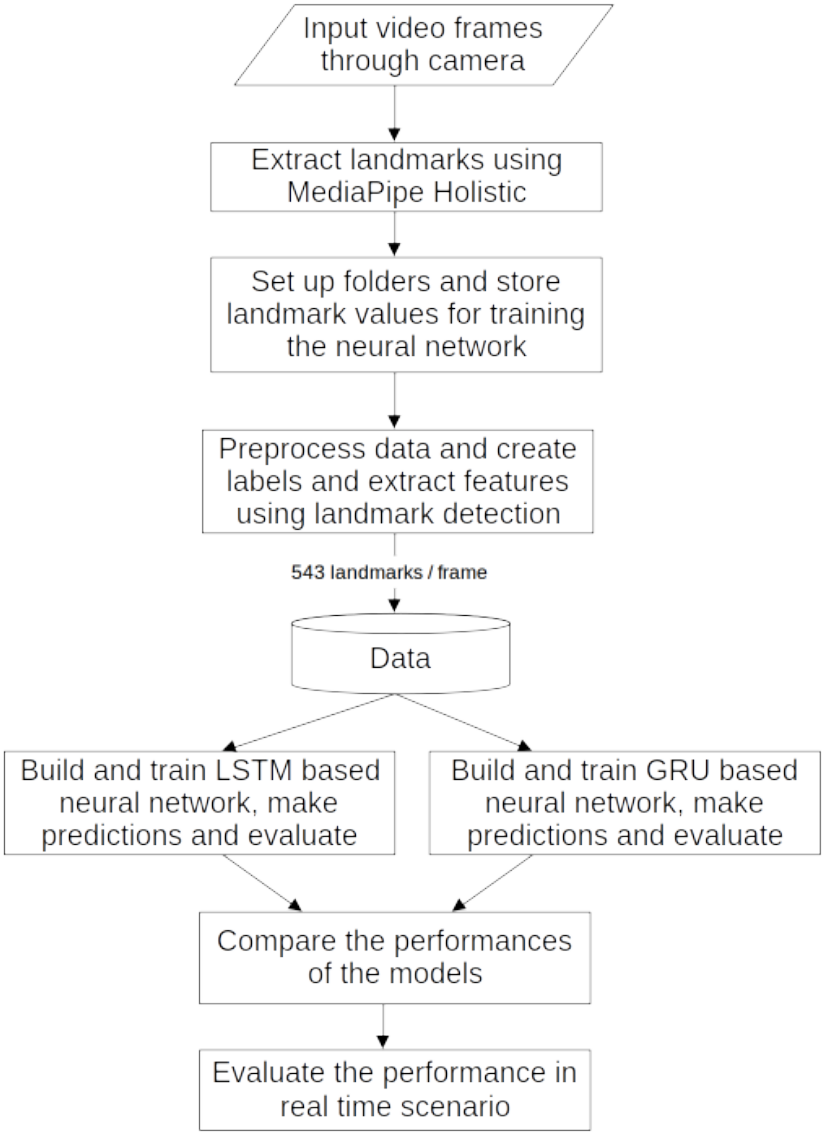


Fig. 1: Proposed framework for real-time SLR using landmark recognition with MediaPipe Holistic, GRU and LSTM.

The flow diagram of the framework is depicted in Fig. 1, where a camera is used to acquire the sign inputs. OpenCV has been used to take input from the camera and for displaying the video frames. The data is then preprocessed as described in Chapter 3.

A. MediaPipe Holistic

MediaPipe is a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. It was developed by Google. This cross-platform Framework works in Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano. Although MediaPipe is currently in alpha at v0.7, it works quite well for our purpose. MediaPipe offers open source cross-platform, customizable ML solutions for live and streaming media. Some of these solutions are Face Detection, Face Mesh, Iris, Hands, Pose, Holistic and so on.

For the purpose of our project we have used the Holistic solution of MediaPipe. The MediaPipe Holistic pipeline integrates separate models for pose, face and hand components, each of which are optimized for their particular domain. However, because of their different specializations, the input to one component is not well-suited for the others. The pose estimation model, for example, takes a lower, fixed resolution video frame (256x256) as input. But if one were to crop the hand and face regions from that image to pass to their respective models, the image resolution would be too low for accurate articulation. Therefore, MediaPipe Holistic is designed as a multi-stage pipeline, which treats the different regions using a region appropriate image resolution.

First, the human pose (top of Fig 2) is estimated with BlazePose's pose detector [7] and subsequent landmark model. Then, using the inferred pose landmarks three regions of interest (ROI) crops are derived for each hand (2x) and the face, and a re-crop model is employed to improve the ROI. Then the full-resolution input frame is cropped to these ROIs and task-specific face and hand models are applied to estimate their corresponding landmarks. Finally, all landmarks are merged with those of the pose model to yield the full 540+ landmarks.

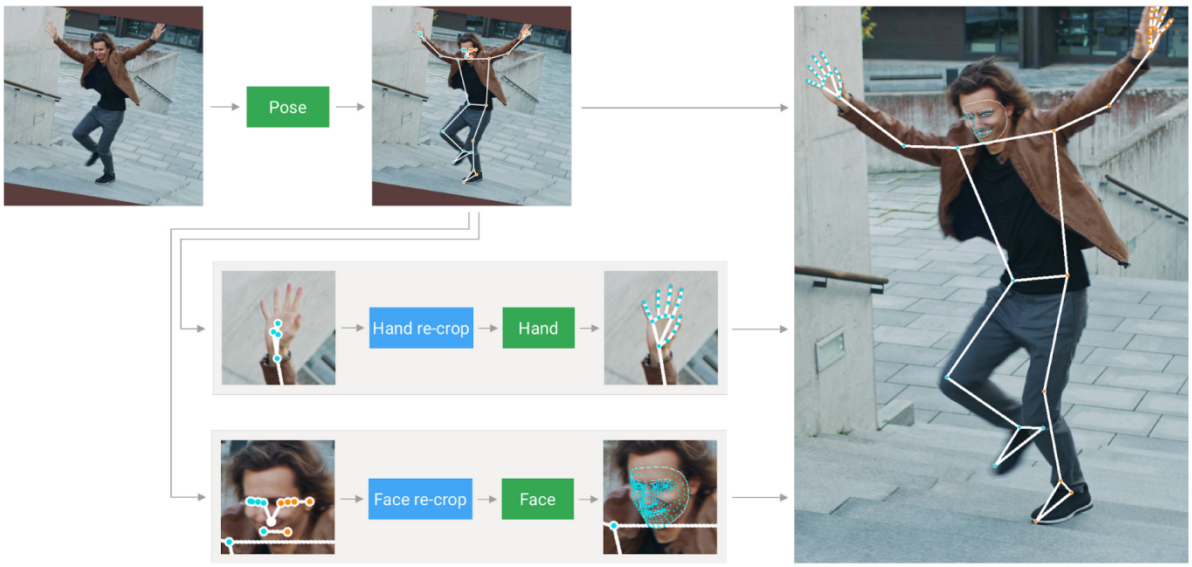


Fig. 2: MediaPipe Holistic Pipeline Overview

Using MediaPipe saved us a lot of effort which would be needed to develop our own Convolutional Neural Network (CNN) for detecting landmarks. We would also have to train the model on large image datasets or use a pretrained model and use transfer learning. Mediapipe Holistic is also quite resilient to variations in background, provided that the background is not too similar to the color of the skin or of the dress, and that the shot is well lit. Thus, using MediaPipe Holistic eased our workflow, and we were able to concentrate our efforts on developing the appropriate GRU and LSTM based neural networks to predict the signs from a sequence of frames.

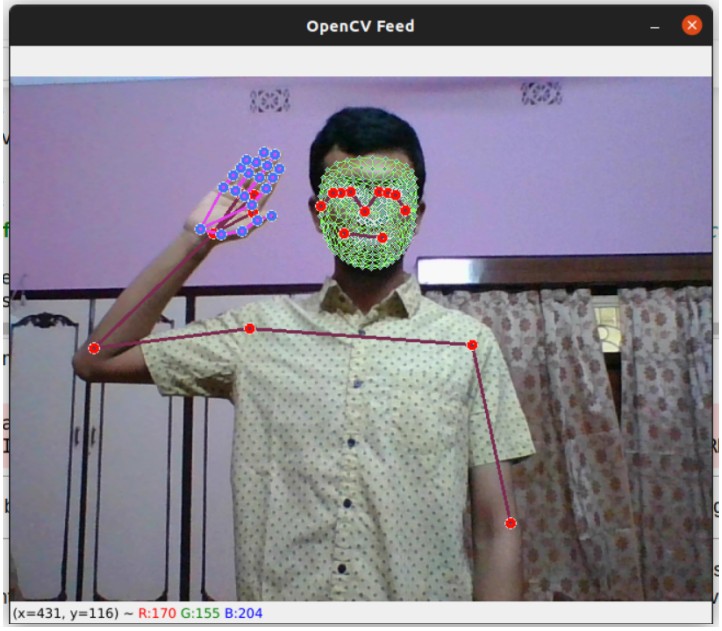


Fig. 3: Landmark detection using MediaPipe Holistic.

B. Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM) Network

GRU and LSTM architectures are variants of Recurrent Neural Network (RNN) (which is a class of neural networks which work on sequence data), which are designed to confront the vanishing gradient problem. The vanishing gradient problem in RNN makes the RNN unable to retain contextual-information over a long term. GRU and LSTM overcome this problem by using memory blocks or cells to store and access information over long periods of time. Hence, both GRU and LSTM are very suitable for continuous recognition tasks, which demand the use of long-term contextual information [8].

GRU uses two gates, an **update gate** and a **reset gate**, while LSTM uses an **update gate**, a **forget gate** and an **output gate**. LSTM is slightly more powerful than GRU, however GRU is a bit easier to train as it has less parameters than LSTM. GRU is relatively new as compared to LSTM. In some scenarios LSTM works better while in other scenarios GRU works better. GRU is more computationally efficient than LSTM since it has a less complex structure [14].

The formulas to update GRU at time t are described as follows,

$$\tilde{c}^{<t>} = tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c) \quad (1)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u) \quad (2)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}) + b_r) \quad (3)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>} \quad (4)$$

where σ is a nonlinear function. $\tilde{c}^{<t>}$ is the candidate value for the memory cell at time instance t. Γ_u and Γ_r represent the outputs of update gate and reset gate respectively. W_c is the weight matrix for the candidate value $\tilde{c}^{<t>}$. W_c consists of two weight matrices W_{ca} and W_{cx} stacked side by side. Similarly W_u and W_r are the weight matrices for the update gate, forget gate and output gate respectively and each of W_u and W_r consist of two weight matrices W_{ua} and W_{ux}, and W_{ra} and W_{rx} respectively stacked side by side . $[a^{<t-1>}, x^{<t>}]$ means that the matrix $a^{<t-1>}$ is stacked on top of the matrix $x^{<t>}$. $a^{<t>}$ is the activation at time instance <t>, $x^{<t>}$ is the input at time instance <t>. b_c, b_u and b_r are the bias vectors. $c^{<t>}$ is the value of the memory cell at time instance t.

The formulas to update LSTM at time t are described as follows,

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \quad (5)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \quad (6)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \quad (7)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \quad (8)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \quad (9)$$

$$a^{<t>} = \Gamma_o * tanh(c^{<t>}) \quad (10)$$

where σ is a nonlinear function. $\tilde{c}^{<t>}$ is the candidate value for the memory cell at time instance t. Γ_u, Γ_f and Γ_o represent the outputs of update gate, forget gate and output gate respectively. W_c is the weight matrix for the candidate value $\tilde{c}^{<t>}$. W_c consists of two weight matrices W_{ca} and W_{cx} stacked side by side. Similarly W_u, W_f and W_o are the weight matrices for the update gate, forget gate and output gate respectively and each of W_u, W_f and W_o consist of two weight matrices W_{ua} and W_{ux}, W_{fa} and W_{fx}, and W_{oa} and W_{ox} respectively stacked side by side . $[a^{<t-1>}, x^{<t>}]$ means that the matrix $a^{<t-1>}$ is stacked on top of the matrix $x^{<t>}$. $a^{<t>}$ is the activation at time instance <t>, $x^{<t>}$ is the input at time instance <t>. b_c, b_u, b_f and b_o are the bias vectors. $c^{<t>}$ is the value of the memory cell at time instance t.

C. GRU and LSTM Networks used in our project

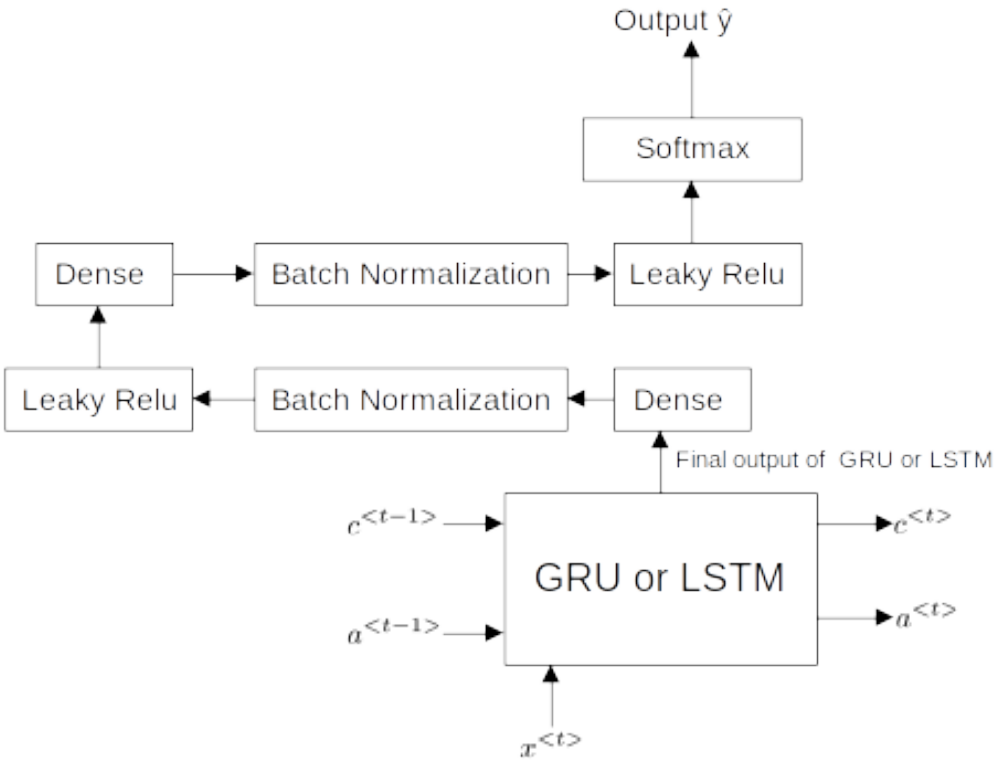


Fig. 4: Proposed model for GRU and LSTM.

We have used the same layers and hyperparameters for our GRU and LSTM models, hence we have shown both the model architecture in the same diagram. The characteristics of the various layers used in the GRU and LSTM models are mentioned below:

1. A GRU (in case of the GRU model) or an LSTM (in case of the LSTM model) layer both with 64 units (which means that the dimensionality of the output space of the GRU or LSTM layer is 64) and tanh(.) activation function. L2 regularization factor for input kernel = 0.044 and for recurrent kernel = 0.014
2. The output of the GRU or LSTM layer is fed into the first fully connected layer (i.e., dense layer) which has 64 hidden units and linear activation function. L2 regularization factor = 0.027.
3. Batch normalization is applied to the output of the first dense layer. Then "Leaky RelU" activation function is applied.
4. The output obtained after applying Leaky Relu activation function is fed into the second dense layer which has 32 hidden units and linear activation function. L2 regularization factor = 0.027.

5. Batch normalization is applied to the output of the second dense layer. Then "Leaky RelU" activation function is applied.
6. The output obtained after applying Leaky Relu activation function is fed into a softmax layer with 11 units since we have 11 categories.

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_3 (GRU)                 (None, 64)                331776

 dense_51 (Dense)            (None, 64)                4160

 batch_normalization_22 (Bat  (None, 64)               256
 chNormalization)

 leaky_re_lu_34 (LeakyReLU)  (None, 64)                0

 dense_52 (Dense)            (None, 32)                2080

 batch_normalization_23 (Bat  (None, 32)               128
 chNormalization)

 leaky_re_lu_35 (LeakyReLU)  (None, 32)                0

 dense_53 (Dense)            (None, 11)                363

=================================================================
Total params: 338,763
Trainable params: 338,571
Non-trainable params: 192
_________________________________________________________________
```

Fig. 5: Summary of GRU model

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm_13 (LSTM)               (None, 64)                442112

 dense_48 (Dense)             (None, 64)                4160

 batch_normalization_20 (Bat  (None, 64)                256
 chNormalization)

 leaky_re_lu_32 (LeakyReLU)   (None, 64)                0

 dense_49 (Dense)             (None, 32)                2080

 batch_normalization_21 (Bat  (None, 32)                128
 chNormalization)

 leaky_re_lu_33 (LeakyReLU)   (None, 32)                0

 dense_50 (Dense)             (None, 11)                363

=================================================================
Total params: 449,099
Trainable params: 448,907
Non-trainable params: 192
```

Fig. 6: Summary of LSTM model

The kernel initializer (i.e. the initialization of the matrices W_{rx}, W_{ux}, W_{fx}, W_{ax} and W_{cx}) used is "glorot uniform" and the recurrent initializer (i.e. the initialization of the matrices W_{ra}, W_{ua}, W_{fa}, W_{aa} and W_{ca}) used is "orthogonal". These initializations help reduce the problem of vanishing gradients. In order **to reduce overfitting, L2 regularization has been used** in the LSTM and GRU layers as well as in the two fully connected layers. This helped increase validation set accuracy. Applying **batch normalization greatly improved convergence**. Without using batch normalization, we had to train the models for around 1000 epochs to achieve satisfactory convergence. After we applied batch normalization, we had to train the model for only around 150 epochs to get satisfactory convergence. Also, during training without batch normalization, the loss kept spiking aggressively from time to time. Batch normalization, along with learning rate schedule described later, helped greatly reduce the problem.

To train our LSTM network, we have used the Adam algorithm, which is an optimization algorithm of mini-batch gradient descent which combines RMSprop and

gradient descent with momentum, to minimize the categorical crossentropy loss function. We have used Adam because it combines the benefits of both RMSprop and gradient descent with momentum, and helps mini-batch converge faster and perform better.

When training a model, it is often useful to lower the learning rate as the training progresses. We observed this first hand, as during training with a reasonably small but fixed learning rate, the loss kept spiking aggressively from time to time. To reduce this, a learning rate schedule called Inverse Time Decay has been used. This schedule applies the inverse decay function to an optimizer step, given a provided initial learning rate. The initial learning rate has been set to 0.002.

Finally, the model is fit to the data and the back propagation algorithm is used to update the weights. The training has been done for 1000 epochs with a mini-batch size of 64. After training, the weights of the model have been stored on secondary storage for future use.

For coding the entire model, the TensorFlow programming framework has been used.

CHAPTER 5
RESULTS

A. Performance of the proposed models

As mentioned before, our dataset consists of 1600 examples. After shuffling the data once, we took 87.5% of the dataset as the training dataset, the 6.25% as the validation dataset and the remaining 6.25% as the test dataset. This translates to 1400 training set examples, 100 validation set examples and 100 test set examples. In order to perform hyperparameter tuning, the models were trained several times with different settings of hyperparameters, like learning rate, L2 regularization factor and number of hidden units in the dense layers. After finding the optimal setting of hyperparameters, the performances of the models on the training and validation sets are demonstrated by the graphs shown in Fig. 7 and Fig. 8.

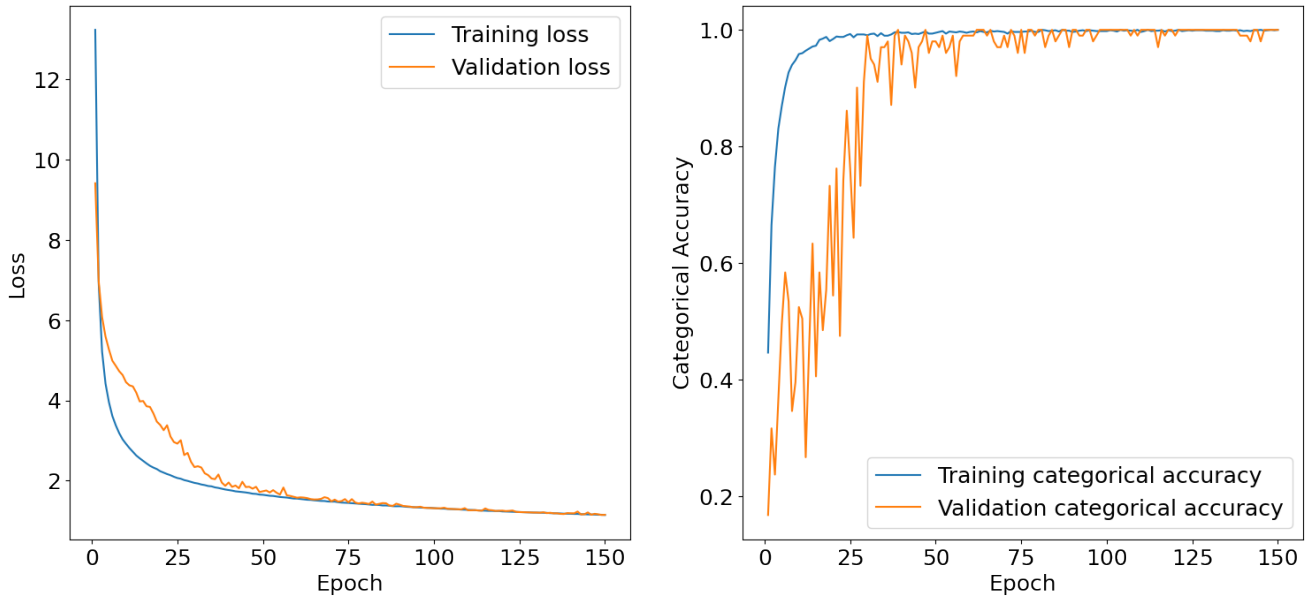


Fig. 7: Performance of GRU model on training and validation sets

For GRU, after 150 epochs,
Training set categorical accuracy = 100%
Validation set categorical accuracy = 100%
Test set categorical accuracy = 99%

	precision	recall	f1-score	support
hello	1.00	1.00	1.00	10
no action	0.80	1.00	0.89	4
thank you	1.00	1.00	1.00	14
i love you	1.00	0.91	0.95	11
again	1.00	1.00	1.00	11
food	1.00	1.00	1.00	9
me	1.00	1.00	1.00	5
want	1.00	1.00	1.00	10
forget	1.00	1.00	1.00	5
friend	1.00	1.00	1.00	7
please	1.00	1.00	1.00	14
accuracy			0.99	100
macro avg	0.98	0.99	0.99	100
weighted avg	0.99	0.99	0.99	100

Fig. 8: Classification report of GRU model

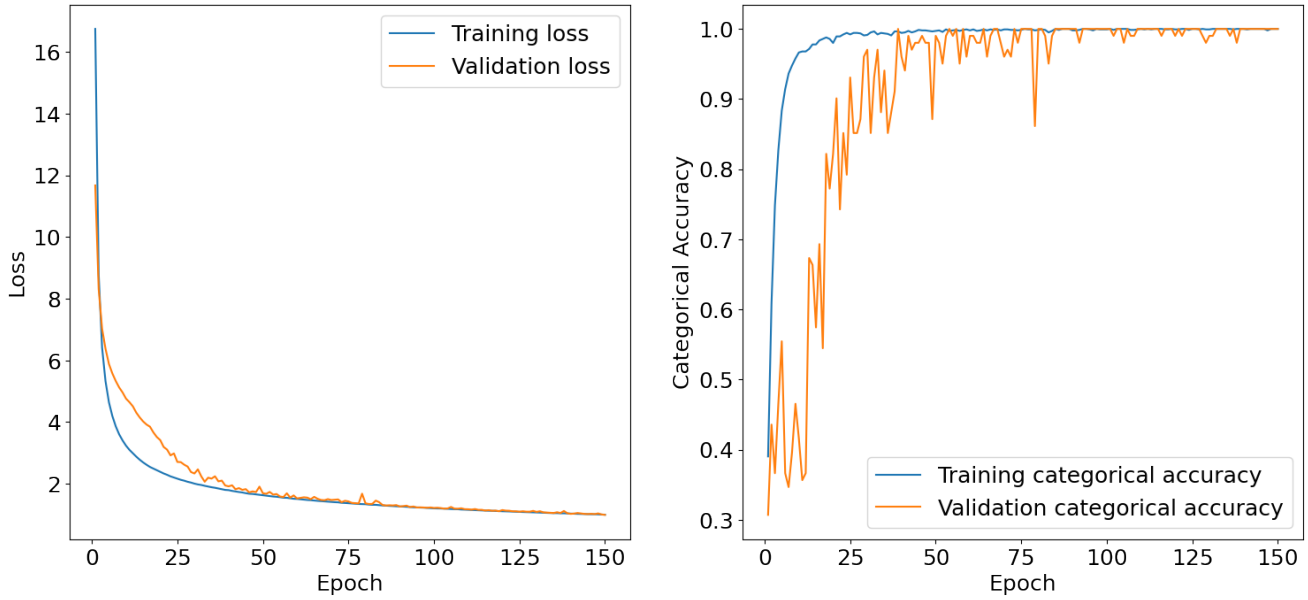


Fig. 9: Performance of LSTM model on training and validation datasets

For LSTM, after 150 epochs,
Training set categorical accuracy = 100%
Validation set categorical accuracy = 100%
Test set categorical accuracy = 99%

	precision	recall	f1-score	support
hello	1.00	1.00	1.00	10
no action	0.80	1.00	0.89	4
thank you	1.00	1.00	1.00	14
i love you	1.00	0.91	0.95	11
again	1.00	1.00	1.00	11
food	1.00	1.00	1.00	9
me	1.00	1.00	1.00	5
want	1.00	1.00	1.00	10
forget	1.00	1.00	1.00	5
friend	1.00	1.00	1.00	7
please	1.00	1.00	1.00	14
accuracy			0.99	100
macro avg	0.98	0.99	0.99	100
weighted avg	0.99	0.99	0.99	100

Fig. 10: Classification report of LSTM model

As we can see, although GRU is simpler than LSTM, in our case, it gives exactly the same accuracy as that of the LSTM model, while also being computationally less expensive.

B. Comparative analysis

For the sake of validation, we also compared the performance of the model to three existing models on SLR. Kshitij Bantupalli and Ying Xie [6] worked on an American sign language recognition system which works on video sequences based on CNN, LSTM and RNN. A CNN model named Inception was used to extract spatial features from frames, LSTM for longer time dependencies and RNN to extract temporal features. Various experiments were conducted with varying sample sizes and the dataset consists of 100 different signs performed by 5 signers and maximum accuracy of 91% was obtained. Sequence is then fed to a LSTM for longer time dependencies. Outputs of softmax layer and max pooling layer are fed to RNN architecture to extract temporal features from softmax layer. Vivek [13] developed a model for American Sign Language recognition consisting of a custom CNN model consisting of 6 convolutional layers with a dropout of 0.25. and a final dropout layer of dropout 0.5. The model was trained on a custom dataset of the authors based on the ASL dataset consisting of only static hand gestures. We also compared the model to a model developed by Lu [12] on SLR. The model consisted of a pretrained CNN named ResNet which was trained by transfer learning for the VIVA Gesture dataset followed by an RNN developed by the authors. The model was trained for 20 epochs with a learning rate of 1-e4 and ADAM for stochastic gradient descent. The batch size was set

to 48 and 8-fold cross validation was used by the authors. The authors also performed augmentation on their dataset.

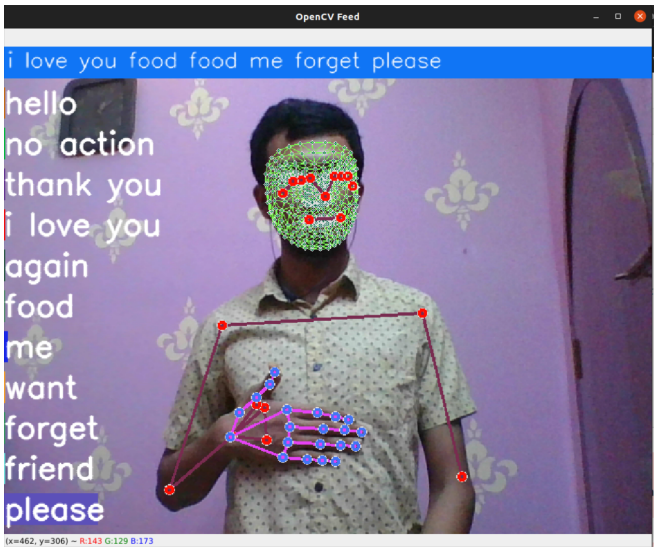
Accuracy of proposed model using Mediapipe and GRU	Accuracy of proposed model using Mediapipe and LSTM	Accuracy of Kshitij Bantupalli and Ying Xie [6]	Accuracy of Vivek et al. [13]	Accuracy of Lu et al. [12]
99%	99%	91%	84%	83%

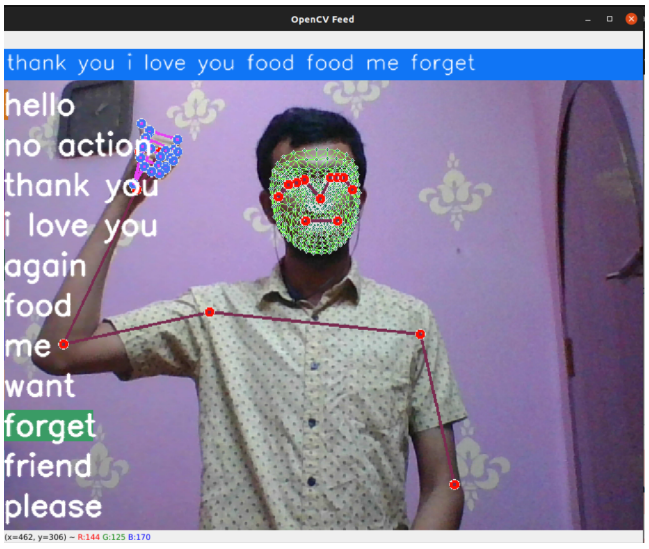
Table 1: Comparison of the performances of proposed models with three other models

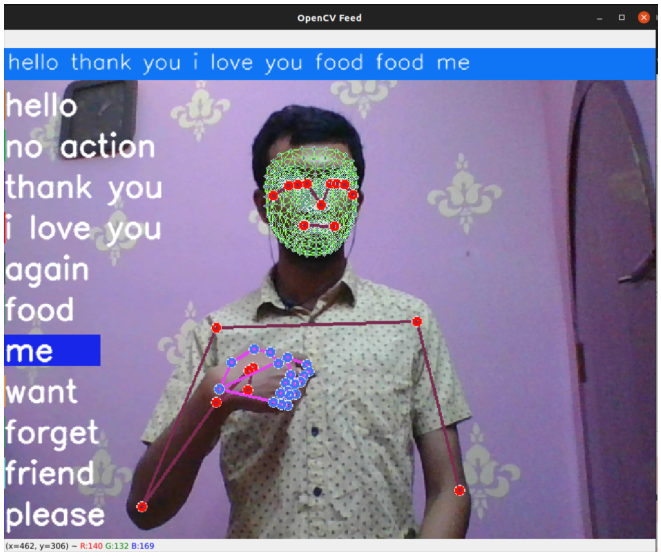
As seen from the table above, our proposed models achieve much greater accuracy than the the other three models. However, it should be noted that the results of the models of [6], [12] and [13] were obtained from a test set of 100 samples consisting of 50 signs. Our result, on the other hand, is obtained from a test set of 100 samples consisting of 10 signs and an additional category called “no action”.

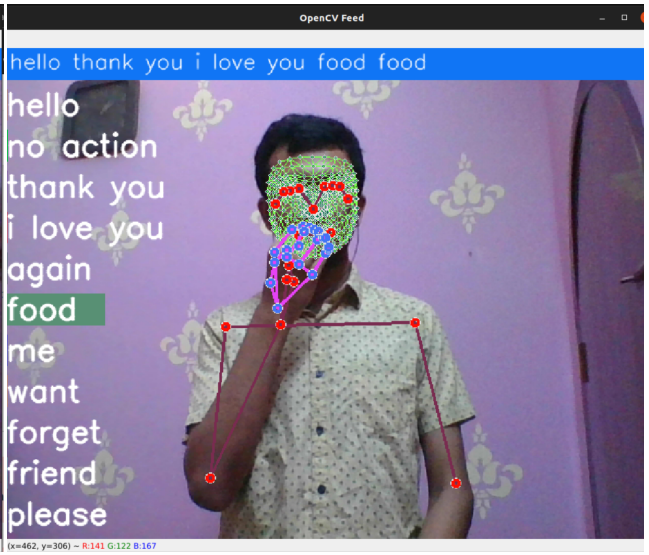
C. Real-time output

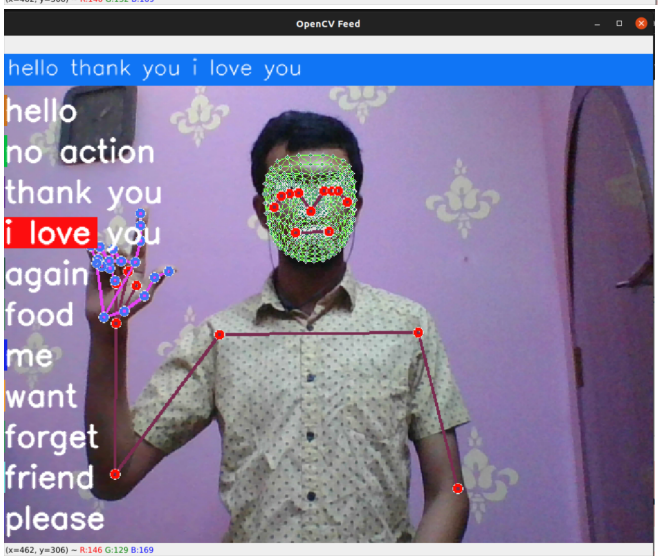
In real time, both the models perform quite well. It detects most of the signs correctly. Sometimes it gives wrong predictions between signs which involve similar hand movements like the signs for "me" and "please". For such types of actions, additional data were recorded, which increased the accuracy of predictions for these categories. Most of the training data was recorded with the signer in a standing position, hence in real time, the model performs better if the signer is in a standing position, which is normally the case while signing.

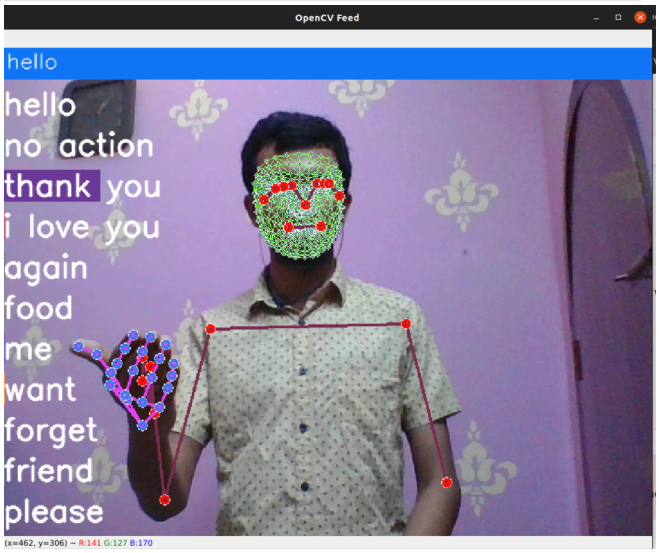

OpenCV Feed
i love you food food me forget please
hello
no action
thank you
i love you
again
food
me
want
forget
friend
please
(x=462, y=306) ~ R:143 G:129 B:173


OpenCV Feed
thank you i love you food food me forget
hello
no action
thank you
i love you
again
food
me
want
forget
friend
please
(x=462, y=306) ~ R:144 G:125 B:170


OpenCV Feed
hello thank you i love you food food me
hello
no action
thank you
i love you
again
food
me
want
forget
friend
please
(x=462, y=306) ~ R:140 G:132 B:169


OpenCV Feed
hello thank you i love you food food
hello
no action
thank you
i love you
again
food
me
want
forget
friend
please
(x=462, y=306) ~ R:141 G:122 B:167


OpenCV Feed
hello thank you i love you
hello
no action
thank you
i love you
again
food
me
want
forget
friend
please
(x=462, y=306) ~ R:146 G:129 B:169


OpenCV Feed
hello
hello
no action
thank you
i love you
again
food
me
want
forget
friend
please
(x=462, y=306) ~ R:141 G:127 B:170

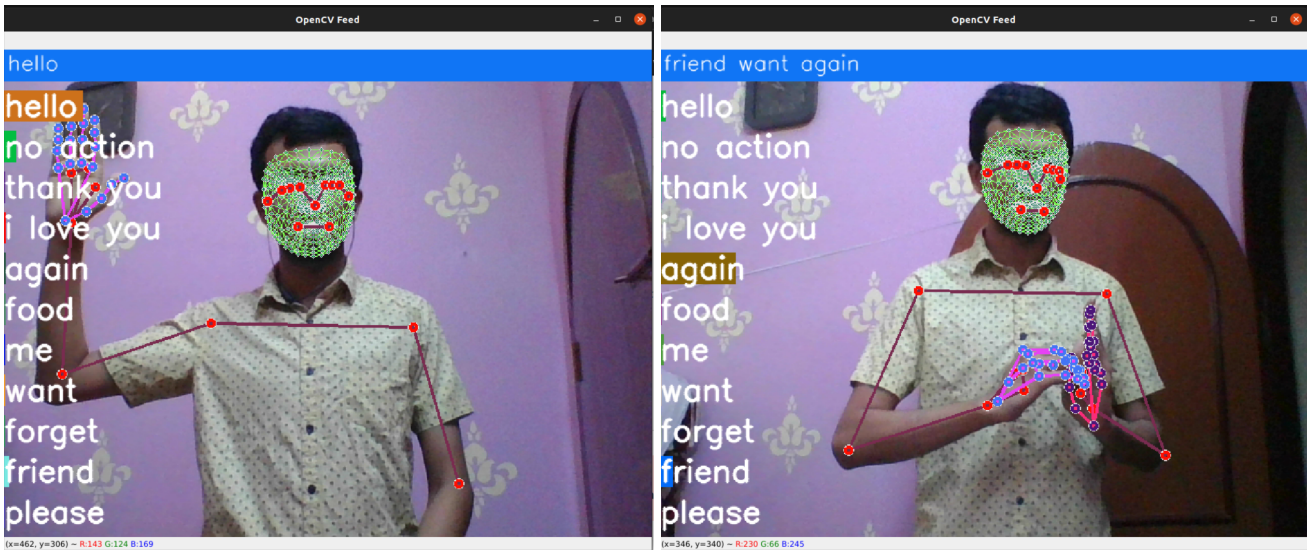


Table 2: Examples of predictions in real-time

While predicting in real time, the confidence of the predictions are displayed on the left side of the output window by the length of the horizontal bars. The last 6 predictions are displayed on the top of output window within the blue bar as can be seen in the figures of Table 2. The colors of the prediction rectangles are generated randomly. Lastly we added a feature to speak out the predictions using the gTTS (google text to speech) package in python. It means that the person with whom a deaf or mute person tries to communicate using our program, will be able to hear the words and phrases denoted by the signs. FFMPEG needs to be installed for this feature to work.

CHAPTER 6
CONCLUSION

In this project, we proposed an effective continuous ASL recognition method, which is based on the combination of landmark detection using MediaPipe Holistic and GRU and LSTM. It contains two major components, analyzing the gestures from images and classifying signs. We compared the performances of the GRU and LSTM models. The powerful landmark detection of MediaPipe Holistic and the ability of the GRU and LSTM networks to learn from contextual information helped us achieve remarkable accuracy in the experiments on our self-built dataset. We believe that our proposed method can meet all the actual application needs and our real-time system can be able to solve the problems faced by people with hearing and speech impairments easily in the near future.

CHAPTER 7
FUTURE PROSPECTS

We faced video stuttering because of lack of CPU power. This causes lots of problems while recording training videos and during real time testing, as the frames are captured after some time intervals, but the person is doing the sign language smoothly. This affects the accuracy of the model in real time, as during real time detection, the computer has to do much more processing during frames to run each frame through the MediaPipe Holistic and then through the neural network. So one of the future prospects is to train and test the performance of the model on a high end pc (ideally with a dedicated GPU). Also if we can get in up and running on google colab, it can help solve the issue. The other future prospect is that this model can be integrated into a video calling web application, which will help a deaf and mute person communicate through video calls with a person who does not understand sign language. We can also collect a lot more data and train the model to improve its accuracy in real time detection and also add signs for more words.

REFERENCES

[1] https://www.handspeak.com/word/most-used/

[2] T. Bohra, S. Sompura, K. Parekh and P. Raut, "Real-Time Two Way Communication System for Speech and Hearing Impaired Using Computer Vision and Deep Learning," 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2019, pp. 734-739, doi: 10.1109/ICSSIT46314.2019.8987908.

[3] Singha, Joyeeta & Das, Karen. (2013), "Recognition of Indian Sign Language in Live Video," International Journal of Computer Applications. 70. 10.5120/12174-7306.

[4] H. Muthu Mariappan and V. Gomathi, "Real-Time Recognition of Indian Sign Language," 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 2019, pp. 1-6, doi: 10.1109/ICCIDS.2019.8862125.

[5] S. Hayani, M. Benaddy, O. El Meslouhi and M. Kardouchi, "Arab Sign language Recognition with Convolutional Neural Networks," 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), Agadir, Morocco, 2019, pp. 1-4, doi: 10.1109/ICCSRE.2019.8807586.

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

[7] https://google.github.io/mediapipe/solutions/pose.html

[8] Graves, A., [Supervised Sequence Labeling with Recurrent Neural Networks] Springer Berlin Heidelberg, (2012).

[9] G. A. Rao, K. Syamala, P. V. V. Kishore and A. S. C. S. Sastry, "Deep convolutional neural networks for sign language recognition," 2018 Conference on Signal Processing And Communication Engineering Systems (SPACES), Vijayawada, 2018, pp. 194-197, doi: 10.1109/SPACES.2018.8316344.

[10] Nakul Nagpal,Dr. Arun Mitra.,Dr. Pankaj Agrawal, "Design Issue and Proposed Implementation of Communication Aid for Deaf & Dumb People", International

Journal on Recent and Innovation Trends in Computing and Communication ,Volume: 3 Issue: 5,pp- 147 – 149.

[11] Neelam K. Gilorkar, Manisha M. Ingle, "Real Time Detection And Recognition Of Indian And American Sign Language Using Sift", International Journal of Electronics and Communication Engineering & Technology (IJECET), Volume 5, Issue 5, pp. 11-18 , May 2014

[12] D. Lu, C. Qiu and Y. Xiao, "Temporal Convolutional Neural Network for Gesture Recognition," Beijing, China.

[13] V. Bheda and D. N. Radpour, "Using Deep Convolutional Networks for Gesture Recognition in American Sign Language," Department of Computer Science, State University of New York Buffalo, New York.

[14] https://www.coursera.org/learn/nlp-sequence-models/lecture/agZiL/gated-recurrent-unit-gru