American Journal of Electronics & Communication

Subhalaxmi Chakraborty et. al., American Journal of Electronics & Communication, Vol. III (3), 20-26

SIGN LANGUAGE RECOGNITION USING LANDMARK DETECTION, GRU and LSTM

Subhalaxmi Chakraborty
Department of Computer Science University of Engineering & Management
Kolkata, Inida
subhalaxmi2008@gmail.com

Prayosi Paul
Department of Computer Science University of Engineering & Management
Kolkata, India
prayosi0409@gmail.com

Suparna Bhattacharjee
Department of Computer Science University of Engineering & Management
Kolkata, India
suparna.bhattacharjee99@gmail.com

Soumadeep Sarkar
Department of Computer Science University of Engineering & Management
Kolkata, Inida
soumadeepsarkar1@gmail.com

Arindam Chakraborty
Department of Computer Science University of Engineering & Management
Kolkata, India
arindamchakraborty1012@gmail.com

***Abstract*— Speech impairment is a kind of disability, affects individual's ability to communicate with each other. People with this problem use sign language for their communication. Though communication through sign language has been taken care of, there exists communication gap between signed and non-signed people. To overcome this type of complexity researchers are trying to develop systems using deep learning approach. The main objective of this paper is subject to implement a vision-based application that offers translation of sign language to voice message and text to reduce the gap between two kinds of people mentioned above. The proposed model extracts temporal and spatial features after taking video sequences. To extract the spatial features, MediaPipe Holistic has been used that consists of several solutions for the detecting face, had and pose landmarks. Different kind of RNN (Recurrent Neural Network) like LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) have been used is to train on temporal features. By using both models and American Signed Language, 99% accuracy has been achieved. The experimental result shows that the recognition method with MediaPipe Holistic followed by GRU or LSTM can achieve a high recognition rate that meets the need of a Sign Language Recognition system that on the real-time basis. Based on the expectation, this analysis will facilitate creation of intelligent-based Sign Language Recognition systems and knowledge accumulation and provide direction to guide to the correct path.**

Keywords*— *Sign Language Recognition, MediaPipe, landmark detection, Recurrent Neural Network, Long Short-Term Memory, Gated Recurrent Unit *(Key words)*

I. INTRODUCTION

Sign language is an essential means of communication for deaf people. It consists of complex hand shapes, movements, hand positions, facial expressions, etc. People with speech and hearing disabilities use sign language as a nonverbal communication tool to express their feelings and thoughts to the rest of the public. However, because these members of the public find it difficult to understand their expressions, trained sign language professionals are needed in medical, legal, educational and training sessions. The demand for these services has increased in recent years. Services such as video remote interpretation using high-speed Internet connections have also been introduced, and while these sign language interpretation services are easy to use and enjoyable, there are major restrictions such as access restrictions. To high-speed Internet and compatible devices.

To address this, a landmark recognition pipeline named MediaPipe Holistic followed by GRU and LSTM has been used to recognize gestures in sign language. An American Sign Language dataset has been recorded and used based on the available videos that exists on the Internet [1] to train 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. MediaPipe Holistic has been proposed 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), we can extract temporal features from the video sequences. The performance between LSTM and GRU has been compared.

OpenCV has been used for capturing and displaying video frames and performing sign language recognition in real-time. OpenCV is an available python library that aims at real-time computer vision. The entire work has been done using Python programming language.

II. LITERATURE REVIEW

Using deep learning, image processing and also computer vision, Tanuj Bohra et al. [2] introduced a communication system that is based on the real-time two way sign language. Techniques such as skin color segmentation, hand detection, contour detection and median blur 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 model for indian sign language recognition from a live video. The system comprises three stages. Skin filtering and histogram matching are pre-processing stages. Eigenvalues and eigenvectors are being considered for feature extraction stage and Eigen value weighted euclidean distance for classification. Twenty people signed dataset that consists 480 images of 24 signs. Model had been tested on 20 videos having an accuracy of 96.25%.

Real-time Handheld Sign Language Recognition System by Dr. Gomathi V. and Mariappan H. [4] Contours are used for face, left and right hand detection by using Fuzzy C-means and Contour Recognition Algorithm. A fuzzy C-means algorithm divides the input data into a certain number of clusters. This model was implemented on a dataset containing videos recorded by 10 signers for multiple words and phrases. We were achieving 75% accuracy.

Hayani et al. [5] proposed an Arabic Sign Language recognition system by CNN and inspired by LeNet-5. The dataset contained 7869 images of Arabic letters and numerals. Various experiments were performed varying the number of training sets from 50% to 80%. With having 80% training dataset 90% accuracy was achieved. The author also compared the results obtained with machine learning algorithms such as KNN (k-nearest neighbor) and SVM (support vector machine) to show the performance of the system. This model was purely image based and can be extended to video based recognition.

Ying Xie and Bantupalli [6] worked on an American sign language recognition system that works on video sequences based on CNN, LSTM and RNN. A CNN model named Inception was used for extracting spatial features from frames, LSTM for longer temporal dependencies, and RNN for extracting temporal features. Various experiments were performed with different sample sizes and the dataset consists of 100 different characters performed by 5 signers and a maximum accuracy of 91% was achieved. The sequence is then fed into an LSTM for longer time dependencies. The outputs of the softmax layer and the max pooling layer are fed to the RNN architecture to extract temporal features from the softmax layer.

Rao et al. [9] proposes Indian Sign Language gesture recognition using CNN. This system works on videos captured by the mobile phone's front camera. The dataset is manually created for 200 ISL brands. CNN training is done with 3 different datasets. In the first batch, the dataset of only one set is given as input. The second batch has 2 sets of training data and the third batch has 3 sets of training data. The average recognition rate of this CNN model is 92.88%.

Real-time recognition and detection of American and Indian Sign Language using Sift In [10]: The author proposed a real-time vision-based hand gesture recognition system for the purpose of human and computer interaction. The system can recognize 35 different hand gestures given in American and Indian Sign Language or ASL and ISL faster with various accuracy. An RGB-to-GRAY segmentation technique was used to minimize the possibility of false detection. The authors proposed an improvised Scale Invariant Feature Transform (SIFT) method and the same was used to extract the features. The system is modeled using MATLAB. A GUI model was implemented to design and efficient user-friendly hand gesture recognition system.

Feature Extraction for Indian and American Sign Language [11] presented development and recent research of sign language based on body language and manual communication. A sign language recognition system typically goes through three steps of preprocessing, character extraction, and classification. The 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 designs whereas [13] used a CNN for stationary hand gestures.

III. Problem Statement and Dataset Description

Before 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 z magnitude uses the same scale of x.

Each pose landmark consists of a visibility attribute, which is a value in [0.0, 1.0] indicating the possibility of landmark being visible in the image.

Graphics and text files were kept separate until text was edited and formatted. Cannot use hard tabs and limit the use of hard returns to only one return at the end of a paragraph. No type of pagination can be added anywhere in the paper. Text headers - the template will do the number.

For each image, 468 facial landmarks and 33 pose landmarks were produced. For each hand, 21 landmarks were produced, assuming that particular hand was in the frame. If a particular hand was not in the frame, then MediaPipe holistic does not return any landmarks for that hand. We designed the programming logic so 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 and GRU models, we wrote an extract keypoints() function that first flattens and concatenates each category (face, pose, left hand, right hand) of landmarks into 4 numpy arrays. It then concatenates the four array numps into one numpy of length 1662 (ie 4×33 + 3×468 + 3×21×2). All 30 video frames were processed this way and the resulting numpy fields were concatenated. So we got a numpy array of shapes (30, 1662) from each video. We

recorded a total of 1600 videos, of which 90 are in the "no action" category (meaning no signs are shown), giving us a shapeless array of shapes (1600, 30, 1662) as our dataset. For some brands, we had to upload more videos to reduce overfitting and improve model performance.

IV. Proposed System

In this section, LSTM and GRU based neural network architectures have been used for continuous SLR using landmark detection with MediaPipe Holistic.

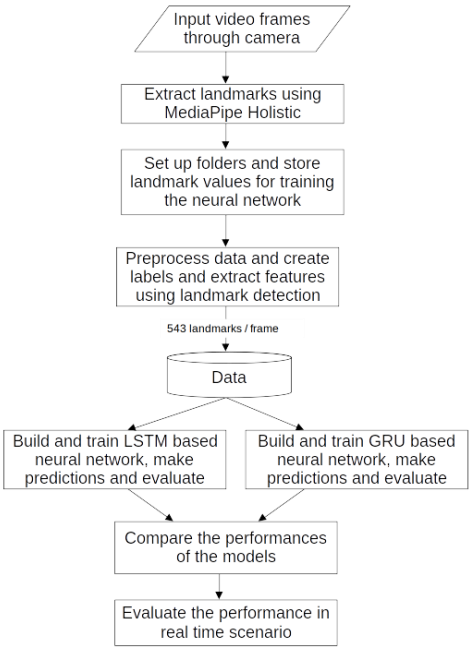


Fig. 1: Proposed framework for real-time SLR using landmark recognition with MediaPipe Holistic, GRU and LSTM.

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 pre-processed as described in Chapter 3.

A. *MediaPipe Holistic*

MediaPipe is a kind of framework for building machine learning pipelines for video, audio, etc. kind of time series data and was developed by Google. This cross-platform framework works on Desktop/Server, Android, iOS and embedded devices like Raspberry Pi and Jetson Nano. Although MediaPipe is currently in alpha version 0.7, it works quite well for our purpose. MediaPipe offers open source and customizable ML solutions for streaming and live media. Some of these solutions are Face Detection, Face Mesh, Iris, Hands, Pose, Holistic and so on.

For the purposes of our project, MediaPipe's Holistic Solution has been used. The MediaPipe Holistic pipeline integrates separate models for face, pose and hand components, each optimized for its specific domain. However, due to their different specialization, entry into one component is not suitable for others. For example, the pose estimation model takes as input a video frame with a lower fixed resolution (256x256). However, if the hand and face areas were cropped from this image to fit into their respective models, the image resolution would be too low for accurate articulation. MediaPipe Holistic is therefore designed as a multi-stage pipeline that processes different regions using region-appropriate image resolutions.

First, the human pose (upper part of Fig. 2) is estimated using the BlazePose pose detector [7] and the subsequent landmark model. Then, three regions of interest (ROIs) for each hand (2x) and face are derived using the derived pose landmarks, and a re-crop model is used to improve the ROI. Then, the full-resolution input image is cropped to these regions of interest, and task-specific face and hand models are used to estimate their corresponding landmarks. Finally, all the landmarks are merged with the landmarks from the pose model to get the full 540+ landmarks.

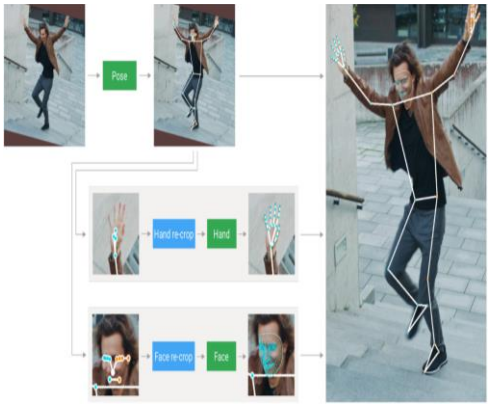


Fig. 2: MediaPipe Holistic Pipeline Overview

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, concentrate 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 Memxzory (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

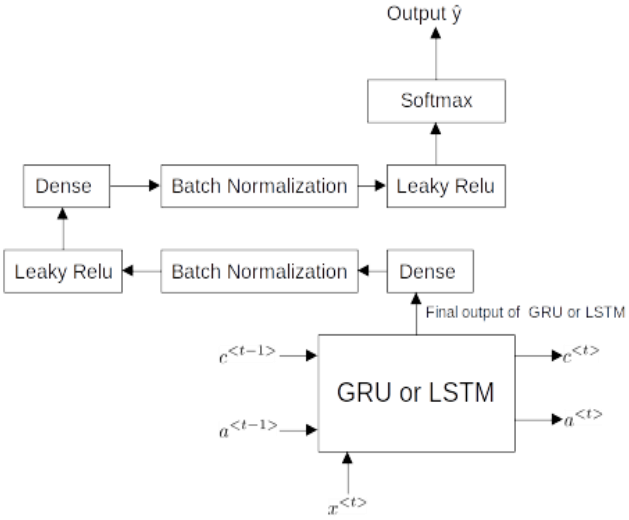


Fig. 4: Proposed model for GRU and LSTM

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 LSTM and GRU networks, the Adam algorithm has been used, which is an optimization algorithm of mini-batch gradient descent which combines RMSprop and gradient descent with momentum, to minimize the categorical cross entropy loss function. The reason behind using Adam is 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. This observation is done at 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 150 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.

V. RESULTS

A. *Performance of the proposed models*

As mentioned before, the 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.

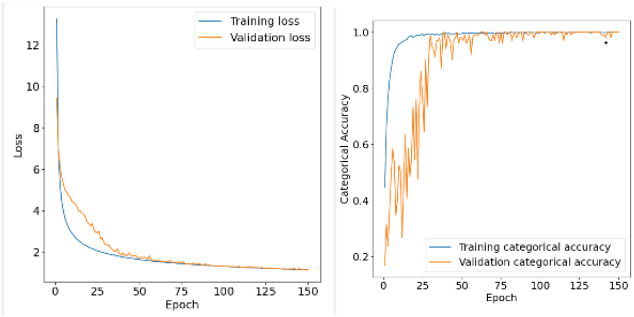


Fig. 7: Performance of GRU model on training and validation sets

	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

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

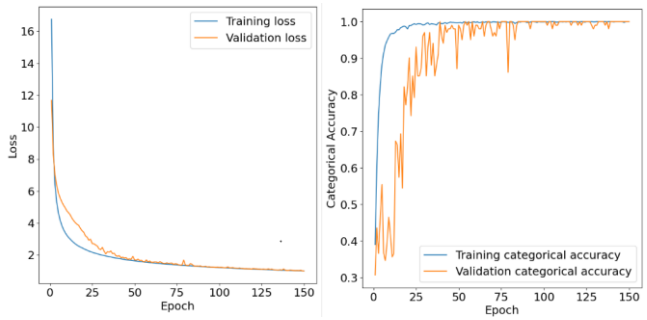


Fig. 9: Performance of LSTM model on training and validation datasets

	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

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

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, the performance of the model to three existing models on SLR has also been compared. 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.

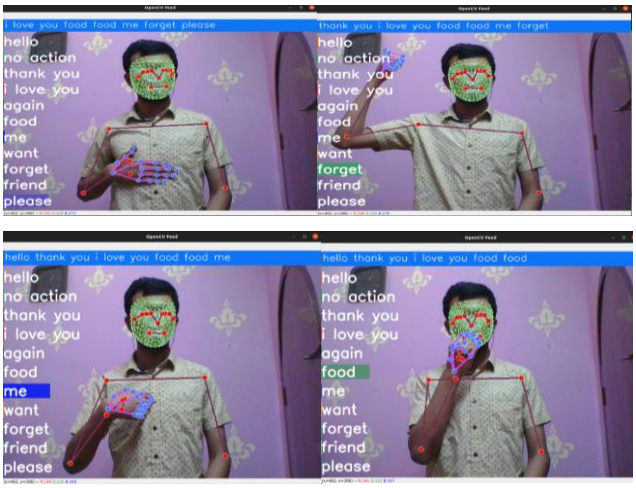
Accuracy of proposed model using Mediapipe and GRU	99%
Accuracy of proposed model using Mediapipe and LSTM	99%
Accuracy of Kshitij Bantupalli and Ying Xie [6]	91%
Accuracy of Vivek et al. [13]	84%
Accuracy of Lu et al. [12]	83%

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 200 samples consisting of 100 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.



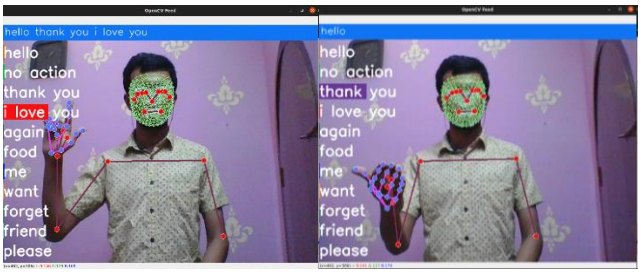


Table 2: Examples of predictions in real-time

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 a feature to speak out the predictions using the gTTS (google text to speech) package in python has been added. 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 need to be installed for this feature to work.

VI. Conclusion

In this paper, an effective continuous ASL recognition method has been proposed, 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. The performances of the GRU and LSTM models has been analysed. The powerful landmark detection of MediaPipe Holistic and the ability of the GRU and LSTM networks to learn from contextual information helped achieve remarkable accuracy in the experiments on our self-built dataset. It is believed that the proposed method can meet all the actual application needs and the real-time system can be able to solve the problems faced by people with hearing and speech impairments easily in the near future.

VII. Future Prospects

The issue that has been faced was 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 one 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. Collection of 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] Hanke, Thomas. "HamNoSys-representing sign language data in language resources and language processing contexts." In *LREC*, vol. 4, pp. 1-6. 2004.

[2] Bohra, Tanuj, Shaunak Sompura, Krish Parekh, and Purva Raut. "Real-Time Two Way Communication System for Speech and Hearing Impaired Using Computer Vision and Deep Learning." In *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp. 734-739. IEEE, 2019.

[3] Singha, Joyeeta, and Karen Das. "Recognition of Indian sign language in live video." *arXiv preprint arXiv:1306.1301* (2013).

[4] Mariappan, H. Muthu, and V. Gomathi. "Real-time recognition of Indian sign language." In *2019 International Conference on Computational Intelligence in Data Science (ICCIDS)*, pp. 1-6. IEEE, 2019.

[5] Rajasegarar, Sutharshan, Christopher Leckie, Marimuthu Palaniswami, and James C. Bezdek. "Quarter sphere based distributed anomaly detection in wireless sensor networks." In *2007 IEEE International Conference on Communications*, pp. 3864-3869. IEEE, 2007.

[6] Bantupalli, Kshitij, and Ying Xie. "American sign language recognition using deep learning and computer vision." In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 4896-4899. IEEE, 2018.

[7] Zhang, Fan, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. "Mediapipe hands: On-device real-time hand tracking." *arXiv preprint arXiv:2006.10214* (2020).

[8] Graves, Alex. "Supervised sequence labelling." In *Supervised sequence labelling with recurrent neural networks*, pp. 5-13. springer, berlin, Heidelberg, 2012.

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

[10] Nagpal, Nakul, Dr Arun Mitra, and 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* 3, no. 5 (2015): 431-433.

[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 , May2014.

[12] Lu, Dongwei, Chu Qiu, and Yi Xiao. "Temporal convolutional neural network for gesture recognition." In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pp. 367-371. IEEE, 2018.

[13] Bheda, Vivek, and Dianna Radpour. "Using deep convolutional networks for gesture recognition in american sign language." *arXiv preprint arXiv:1710.06836* (2017).

[14] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling." *arXiv preprint arXiv:1412.3555* (2014).