International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol. 9, Issue.7, July 2021 E-ISSN: 2347-2693

Proposal of a Real-time American Sign Language Detector using MediaPipe and Recurrent Neural Network

Souradeep Ghosh[1]

[1]Department of Electrical Engineering, Heritage Institute of Technology, Kolkata, India

Author's Mail Id: jeetsouradeep@gmail.com, Tel.: +91-7685920081

DOI: https://doi.org/10.26438/ijcse/v9i7.4652 | Available online at: www.ijcseonline.org

Received: 18/Jul/2021, Accepted: 20/Jul/2021, Published: 31/Jul/2021

Abstract— The predominant vocabulary of the deaf and dumb, Sign Language serves as a natural, visual language which our brain is capable of processing and deciphering linguistic details. For the past two decades, scientists have been researching the automated recognition of sign language using translating gloves and complex systems with several cameras. Most of these systems can provide partial or complete recognition of the vocabulary but aren't cost-effective for the average and below-average section of the demographic. With the advent of AI, we're trying to overcome this biasness in technology. Google's MediaPipe, which is an open-source framework for multimodal (video, audio, time-series) features with applied ML pipelines, came into existence in 2019. Using MediaPipe's Multi-hand Tracking model pipeline we can get landmarks of our fingers. This paper advocates the use of MediaPipe Hand Tracking to get hand landmarks, training a Keras RNN-LSTM model with that data to detect Sign Language of 5 trained words in real-time.

Keywords— MediaPipe, American Sign Language, OpenCV, RNN, LSTM, Real-time

I. INTRODUCTION

According to the data provided in 2019 by World Health Organisation, approximately 466 million of the world's population has disabling hearing loss. It is extrapolated by their research that by 2050, this number would surge to over 900 million. For the past centuries, people have been using translators and texts to fetch the need for communication between the disabled and non-disabled people. These methods have succumbed to limitations: Firstly, the text conversions aren't able to elucidate human expressions and proceeds very slowly which is not as comfortable as the spoken ones. Secondly, the translator method can be expensive and insolvent for the majority of the demographics and bear privacy issues to the public and government bodies.

The advent of Computer Vision and Artificial Intelligence has assisted us to overcome the technological barriers involving finger tracking and real-time image and video processing. This project aims to build and enact an AI model capable of detecting sign language in real-time with the help of Computer Vision and Google's MediaPipe Multi-Hand Tracking ML pipelines. MediaPipe helps us to track fingers of our hands and returns the coordinates of each point even in real-time. Giving primacy to finger tracking, we generated a dataset of 42 coordinates per frame (21 for each hand), for each video dataset we created for a word. Using Keras library of Tensorflow we created a RNN (Recurrent Neural Network) – LSTM (Long Short Term Memory) model [9], and trained it with the dataset we created earlier. Then using the real-time approach of MediaPipe finger tracking, we used the saved model to predict the word that we're showing. MediaPipe [10] is a framework effective to detect sign languages. This method is advantageous to the earlier used methods of detection as it only uses a single camera application of a laptop or a mobile phone to detect the sign- language.

This paper doesn't intend to recognize each and every ASL (American Sign Language). Owing to restrictions in time and computational power, our model can differentiate between five different words and also invigorates the hypothesis that if a larger dataset is used along with combined MediaPipe Hands and PoseNet algorithm, any number of signs is possible to decipher. The rest of the paper contains Section II as Related Works, Section III as Methodology (MediaPipe Implementation, Dataset Collection, Preprocessing, Model Creation, Training and Evaluation, Evaluation of test data and Real-time testing), Section IV as Results and Discussion, Section V as Conclusion and Section VI as Future Scope.

II. RELATED WORK

The need for developing a solution for sign language detection was clear to the researchers for a couple of decades. Now, we will be summarizing the previously known works on this topic and would illustrate how our Google's MediaPipe based research is different, further in the other topics.

The contour detection approach presented an OpenCV-based approach which was the earliest known approach for Object Detection (before MediaPipe). OpenCV is an Intel-based programming library that focuses on real-time

computer vision applications. In *"Hand detection using multiple proposals"* by **Arpit Mittal et al.** [1], which studies real-time hand detection claims that it can detect multi-hands although the accuracy drops when the top half part of the human body appears in the frame.

In 2015, **Ruchi Manish Gurav et al.** presented a similar approach in *"Real time finger tracking and contour detection for gesture recognition using OpenCV"* [2]. It shows that steady signs can be predicted at the rate of 30FPS but lists numerous disadvantages: the fingers that are behind the other fingers aren't detected correctly and if we display a fist, it's only detected as a fist and not fingers.

RNN combined with other AI models approach is a similar approach that we used in this project. We have used Google's MediaPipe along with RNN to detect sign language. **Sarfaraz Masood et al.** details a similar approach which utilizes the CNN approach to detect the position of the hands along with RNN to detect a pattern of signs in the paper "*Real-Time Sign Language Gesture (Word) Recognition from Video Sequences Using CNN and RNN"* [3]. The results were pretty good as the paper states the detection of 46 signs with an accuracy of 95.2 percent but the type of camera used is not defined by the authors.

HMM, Hidden Markov Model, approach was first proposed back in 1997 by **Kirsti Grobel et al.** in their research paper "*Isolated sign language recognition using hidden Markov models"* [4]. They recognized 262 independent signs with an accuracy of 91 percent. Since then, many papers have been published using this method. One of the recent publications includes the one by **Pradeep Kumar et al** [5]**,** who in their publication, "*Coupled HMM-based multi-sensor data fusion for sign language recognition",* tried different hand tracking approaches to detect 25 different sign languages with the help of HMM on a 2000 word sample and waxed an accuracy of around 90 percent.

The Glove approach advocates the use of gloves for hand tracking, with sensors in it to recognize the movement of fingers and hands and display the word the user is trying to convey. This approach uses flex-sensor and gyroscope to detect finger movement and hand orientation respectively. **S. A. Mehdi et al.** [6]**,** used seven sensors (one for each finger and two for hand orientation) to recognize alphabets in ASL in their publication "*Sign language recognition using sensor gloves"* in 2002. They obtained an accuracy close to 88 percent. Later in 2016, **Jakub Galka et al.** in their publication, "*Inertial motion sensing glove for sign language gesture acquisition and recognition"* [7], used the Glove approach combined with HMM to get an accuracy of 99 percent for 40 different gestures. The cost of sensors and the uncomfortability of wearing gloves make it an unreliable method of further research.

III. METHODOLOGY

Real-time sign language detection can be achieved by a variety of Classical and Deep Learning algorithms like Decision Tree, Random Forrest and K- Nearest Neighbour, but the accuracy of these algorithms isn't over 75 percent. So, we have used the Keras RNN LSTM model to fit our dataset of hand landmarks obtained from MediaPipe implementation. MediaPipe hand tracking was first introduced by **Fan Zhang et al.** in their publication, "*MediaPipe Hands: On-device Real-time Hand Tracking"* [8], in 2020. We have used Python programming for our development. The basic architecture of our model is depicted in Fig 1.

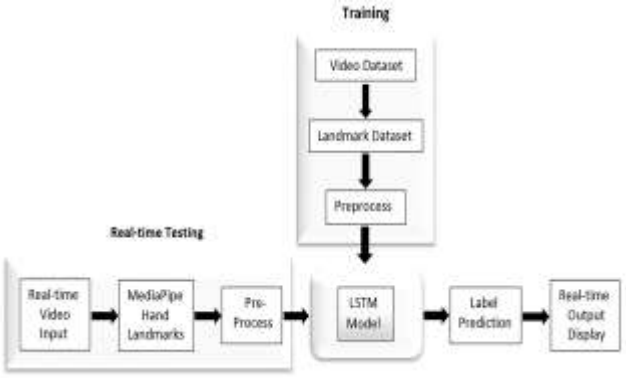

Fig 1. Basic architecture of Real-time Sign Language Detector

1. MEDIAPIPE IMPLEMENTATION

MediaPipe employs various ML Pipelines consisting of multiple models working in conjunction. A hand landmark model works on the cropped part of the image explicated by the palm detector and a series of high-precision 3D hand key points is returned. The accuracy of palm detection is higher than 95 percent according to the documentation by Google. After the palm detection, the image is passed to a CNN (Convolutional Neural Network) model [11], also known as the finger detector. The model accurately defines key point localization of 21 3D- hand-knuckle coordinates of each hand, shown in Fig 2. We have implemented this MediaPipe Hand-landmarks pipeline and tested it in our local setup in Real-time, depicted in Fig 3.

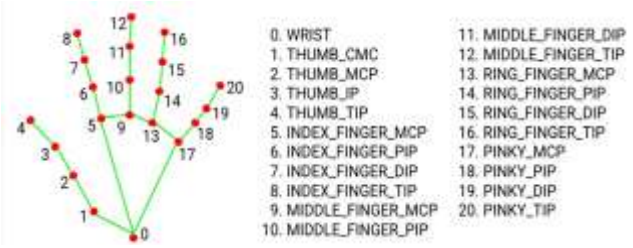

Fig 2 21 3D Key points for each hand

Fig 3 MediaPipe Hand-landmark implementation

2. DATASET COLLECTION

We produced about 126 videos each for the 5 chosen signs of ASL. These words are- teacher, think, tiger, twins and wait. The video dataset was of 30FPS each and lasted for 2-3 seconds and was pre-processed to increase the clarity and hue of the frames in it. Now, this processed video dataset was passed through the MediaPipe ML pipelines to get the hand-knuckle coordinates of each frame. We used frames at the rate of 1 out of 8 consecutive frames and got 42 coordinate values for each frame and stored them in a CSV file. We ignored the z-axis in this case as we aren't considering image depth for our instance. After the conversion, we obtained a dataset of 630 rows of coordinates. The no. of columns isn't fixed as the time period, and corresponding frames for all the videos are different. A part of the raw dataset is given in Fig 4.

1	78	79	80	81	82	83	84
68	0.72396	0.53637	0.70945	0.55412	0.71004	0.56426	0.87545
69	0.79391	0.5038	0.78004	0.51825	0.79161	0.52552	0.8229
70	0.78067	0.57183	0.76938	0.58776	0.78226	0.59276	
71	0.78146	0.55173	0.78208	0.56715	0.80541	0.5727	0.91161
72	0.75484	0.54924	0.74229	0.56066	0.74761	0.56514	0.89738
73	0.80153	0.51204	0.786	0.52093	0.78865	0.5264	0.88233
74	0.71715	0.53948	0.68859	0.5503	0.67508	0.55649	0.90342
75	0.8257	0.52875	0.80456	0.54713	0.80118	0.55744	0.90175
76	0.74365	0.50245	0.73776	0.52442	0.765	0.53548	0.75242
77	0.28012	0.48513	0.26629	0.48487	0.25392	0.48577	0.10259
78	0.24475	0.47346	0.24395	0.49019	0.21912	0.493	0.135
79	0.16649	0.46129	0.17268	0.46712	0.16773	0.47144	

Fig 4 Raw Hand-knuckle coordinates dataset

3. PREPROCESSING

The preprocessing of the CSV dataset is of two steps: handling missing values and labelling the rows. As mentioned earlier, we have an uneven number of columns in our dataset due to the different time span of videos. So, there are many empty cells in our CSV dataset. We padded our dataset with 0 so that the structure of data will be even and our model can handle it with ease. The second step involves labelling the rows with an index number associated with a particular Sign, for example, 0 for teacher, 1 for think, 2 for tiger, 3 for twins and 4 for wait. We stored these labels in another text file too for our future reference. Now, the dataset is clean and ready to be trained. A part of the cleaned dataset is given in Fig 5.

1	78	79	80	81	82	83	84	85
68	0.723955	0.536371	0.709447	0.55412	0.710037	0.564265	0.875445	0
69	0.793914	0.503803	0.780037	0.518245	0.791614	0.525516	0.822903	0
70	0.78067	0.571833	0.769377	0.587762	0.782257	0.592757	0	1
71	0.781462	0.55173	0.78208	0.567155	0.805408	0.572701	0.911606	1
72	0.754837	0.549239	0.742292	0.560662	0.74761	0.565138	0.897382	2
73	0.801531	0.51204	0.785995	0.520929	0.788647	0.5264	0.882331	2
74	0.717151	0.539478	0.688592	0.550305	0.675076	0.556493	0.903421	3
75	0.825697	0.528753	0.804561	0.547128	0.801182	0.55744	0.901748	3
76	0.743652	0.502449	0.737763	0.524421	0.765001	0.535482	0.75242	4
77	0.280123	0.485129	0.26629	0.484868	0.253916	0.485773	0.102588	4
78	0.244754	0.473464	0.243952	0.490193	0.219117	0.492999	0.134999	0
79	0.166491	0.461285	0.172685	0.467122	0.167727	0.471436	0	0

Fig 5 Cleaned Hand-knuckle coordinate Dataset

4. MODEL CREATION

We have used an RNN model which takes the Hand-knuckle coordinates as input and returns the output label for that set of coordinates. The original vanilla version of the RNN is rarely used as it suffers from a major problem. The original RNN has the tribulation of vanishing gradient, where it cannot remember the long term dependencies of the dataset. Advanced versions like LSTM (Long Short Term Memory) and GRU (Gated Recurrent Unit) [12], used to compensate for this issue. We are using an LSTM cell in our model. The basic LSTM cell architecture is given in Fig 6.

The two states of an LSTM cell are cell state and hidden state. The cell state is the memory of the LSTM cell while the hidden state is the output of the cell. In Fig 6, three inputs of each LSTM cell is depicted where: C_{t-1} is the cell state input for the memory cell in timestep t-1, h_{t-1} is the hidden state input for the memory cell in timestep t-1 and X_t is the input in timestep t. There are three gates in an LSTM cell: input gate (adds new data to the cell), output gate (outputs cell data) and forget gate (erases cell data). The function of these gates are as follows: we choose to add new content from the current input to our present cell state in the input gate, we choose what to output from our cell state in the output gate and we choose what must be removed from the h_{t-1} state, keeping only the relevant ones using the forget gate.

In our LSTM model, we used a dimensionality of 256 for the outer layer and 128 for the inner layer. These layers are accompanied by a dropout layer, a regularization layer where input and recurrent connections to LSTM are excluded from activation and weight updates during training probabilistically. The model performance is then enhanced, successfully avoiding the overfitting [14]. These layers are followed by two other layers: Flatten and Dense. Flatten layer is essential so that LSTM would be shaped as one dimension per input. The dense Layer is also called the Fully-connected layer. It ensures each neuron of layer N is connected to every neuron in the N+1 layer. We are using Keras and Tensorflow to build our model. Keras is an open-source python library that provides an interface for modelling deep learning and artificial neural networks while Tensorflow is an open-source end-to-end Machine Learning Platform. The input would pass through our trained model and the output would be predicted by it. The architecture of the model is given in Fig 7.

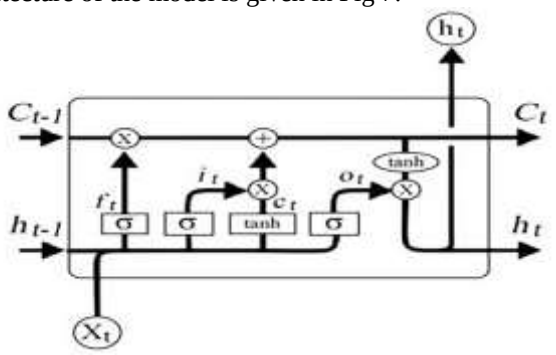

Fig 6 Basic LSTM Cell architecture

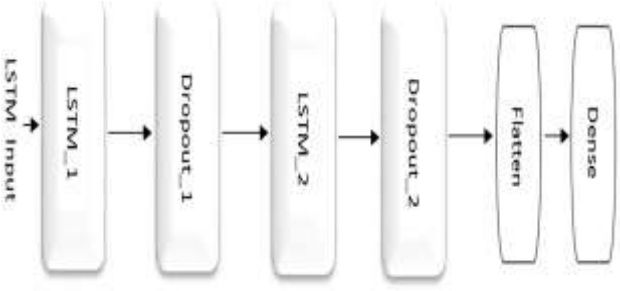

Fig 7 Architecture of our LSTM model

5. TRAINING AND EVALUATION

After accessing the Dataset CSV file we are taking 10 frames per video as our input to the model. So, the number of columns for our X variable is 420, i.e. 42*10 frames, and our Y variable is of a single column that contains the label for each video. As mentioned earlier, there are 630 videos (rows) in our dataset. We decided to keep only 5 per cent as our test data considering the small dataset we have and we split the dataset into X_train, X_test, Y_train and Y_test. So, the training dataset contains 598 rows and the test dataset contains 32 rows.

LSTM [13] is capable of handling only 3D datasets. We used Reshape function of the Numpy package to reshape the X_train and X_test values into 3D arrays. Now, the current shape of the X_train array is (598×420×1). This the final set of input for our model. As described in the model creation part, our model contains two LSTM layers of 256 and 128 units respectively each followed by a dropout layer of 0.2. These layers are followed by Flatten and Dense Layers. The output shapes and the parameters of our model are shown in Fig 8. Adam [15] optimiser is used for our model and Mean-squared loss as the loss parameter, considering it's a multi-class classification model. We have set the number of epochs to 170 with a batch size of 32 for our model and fitted the data (X_train, Y_train) in our model with accuracy as metrics. The validation set for the evaluation is again 5 per cent, which is 30 videos (rows) of our final dataset.

The parameters chosen to analyse and evaluate our model are training accuracy, training loss, validation accuracy and validation loss. The training of 170 epochs took around two and a half hours in our system and we were able to reach an accuracy of 95 per cent with a loss of 6 per cent. After the end of the training, we saved the model as .h5 file and plotted the parameters of our model. We produced two plots, Epoch vs. Accuracy and Epoch vs. Loss, for our training and validation dataset, shown in Fig 9 and 10 respectively. From the plot, the increment of the accuracy and decrement of the loss of the training and validation set suggests that the dataset perfectly fits in our model and the fact that we have not overfitted our data.

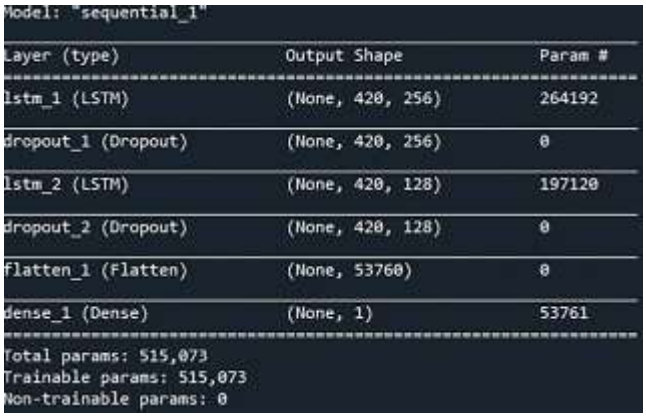

Fig 8 Output shapes and no. of parameters of different model layers

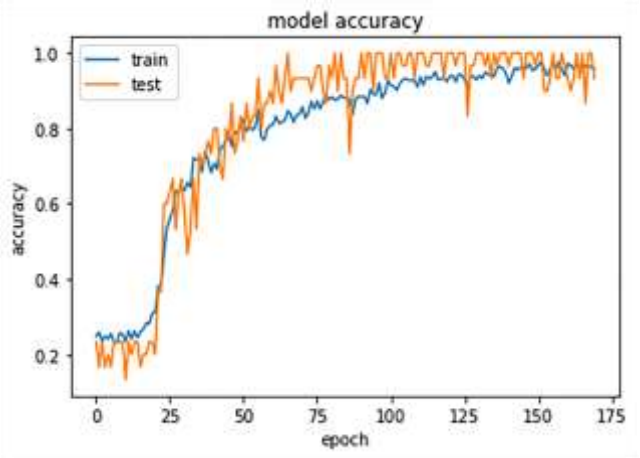

Fig 9 Epoch vs. Accuracy for Train and Test set

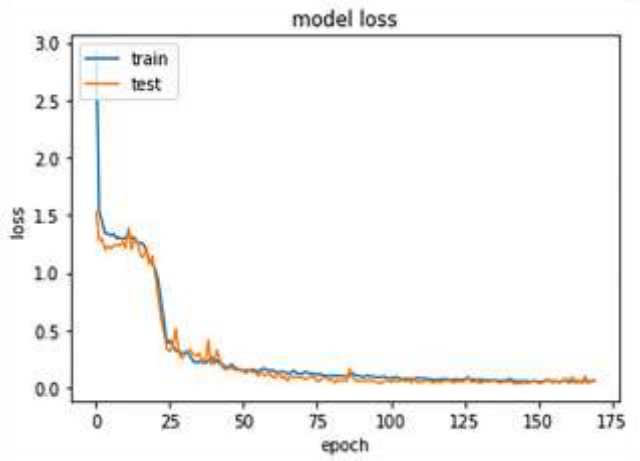

Fig 10 Epoch vs. Loss for Train and Test set

6. EVALUATION OF TEST DATA

The test and train graphs from the previous topic give us only brevity of information on how it would work on a real dataset. So, considering this smattering amount of analysis we had, we decided to use predictive analysis algorithms like Confusion Matrix along with other classification report parameters, i.e. precision, recall and F1 score, to calculate accuracy, macro average and weighted average on the test data.

The first step is to use the 32 videos we kept as X_test to predict the Y labels with our trained model. Once we receive the Predicted Y values we are rounding them off to the nearest integer labels using the round function of the Numpy package. Now the labels are ready to be plotted in a confusion matrix [16], shown in Fig 11. The confusion matrix is an N × N matrix used for the evaluation of a classification model. The Y-axis contains the real Y labels for the X_test data and the X-axis contains the rounded predicted labels from our model. As we have 5 labels for our dataset so the dimension of the confusion matrix is 5 × 5. The labels are the integers between 0-4 both inclusive. The colour gradient scale shows how each cell of the matrix is coloured depending on the quantity of data.

Now, we are gathering the detailed classification report of these predicted values. There are 4 possible cases of response for a classical model, namely: TN/ True Negative, TP/ True Positive, FN/ False Negative and FP/ False Positive. TN is the case when both the real and predicted cases are negative, TP is the case when both real and predicted cases are true, FN is the case when the real case is positive but the predicted case is negative and FP is the case when the real case is negative but the predicted case is positive. Precision is the classifier's ability to not label an instance positive if it's negative. It's the ratio of the true positives to the sum of true and false positives, given in (1).

$$\text{Precision} = TP/(TP + FP) \quad (1)$$

Recall is the classifier's ability to find all the positive instances. It's the ratio of the true positives and the sum of the true positives and false negatives, given in (2).

$$\text{Recall} = TP/(TP + FN) \quad (2)$$

The weighted harmonic mean of the Precision and Recall, given that the best score and worst score is 1.0 and 0.0 respectively, is called the F1 score [17]. So, for comparing a classification model the weighted average of the F1 score should be used and not the actual accuracy percentage. The F1 score is given by the following formulae (3).

$$\text{F1 Score} = 2 \times (\text{Recall} \times \text{Precision}) / (\text{Recall} + \text{Precision}) \quad (3)$$

The full classification report table of our model is shown in Fig 12.

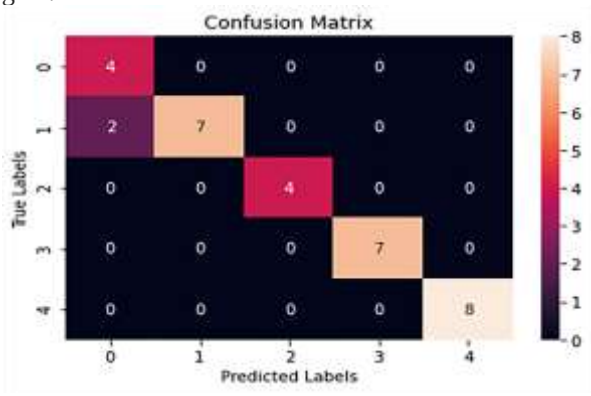

Fig 11 Confusion Matrix for evaluation of predicted results

	precision	recall	f1-score	support
0	0.67	1.00	0.80	4
1	1.00	0.78	0.88	9
2	1.00	1.00	1.00	4
3	1.00	1.00	1.00	7
4	1.00	1.00	1.00	8
accuracy			0.94	32
macro avg	0.93	0.96	0.94	32
weighted avg	0.96	0.94	0.94	32

Fig 12 Classification report table of the predicted instances

7. REAL-TIME TESTING

Now, the saved model can be used for real-time implementation. Firstly, the model is loaded onto a variable using Keras' load model method. OpenCV [18] is an open-source computer vision platform to capture, modify and augment images and videos to create a dataset and implement it in a model. By using the video capture function of OpenCV, we captured real-time video and split it into 30 fps. We took a nominal frame gap of 8 frames, similar to our dataset creation, and passed these friends consecutively to the MediaPipe hand landmarks function. Upon detecting hands and acquiring Hand-knuckle coordinates for each frame, we have 42 coordinates (21 for each hand) and stored them in a list.

As posited in the training and evaluation section, we take 10 frames per input to the model for implementation too. The 42 coordinates of each frame get appended to the list variable until a length of 420 is reached. Upon reaching the desired length, we send this variable as input to our model for prediction. The model returns a decimal value of a particular labelled class. Rounding off the predicted results we match it with the word label we previously stored in a text file and display the word as output in the same real-time video window. After displaying the output the list storing the coordinates is emptied and the process continues. The camera operation is terminated only when a wait key is pressed.

IV. RESULTS AND DISCUSSION

The real-time test was run on our setup with the following configurations:

- CPU: i5-7200U CPU (2.50GHz - 2.71 GHz)
- RAM: 16 GB (1600 MHz)
- GPU: 4 GB Nvidia GeForce 940 MX

The results from testing are compared with some of the Original ASL depicters from YouTube in the following Fig 13, 14, 15, 16 and 17.

Fig 13 YouTuber (Id: one fact ASL) vs. real-time output for word 'Wait'

Fig 14 YouTuber (Id: Signs) vs. real-time output for word 'Think'

Fig 15 YouTuber (Id: Laura Berg Life) vs. real-time output for word 'Tiger'

Fig 16 YouTuber (Id: Teaching Resources) vs. real-time output for word 'Teacher'

Fig 17 YouTuber (Id: Sharon Nason) vs. real-time output for word 'Twins'

The results obtained are decent accurate considering the minimal configurations used for real-time implementation. Our model and approach are different from other previously mentioned methods because most of the methods are not able to give good accuracy in real-time or need some high configuration camera and setup. On the other hand, we have implemented it on a simple phone camera or Webcam instead of some high configuration cameras. This makes the approach cost-effective for the general public. This research opens up new doors for a more accessible way to recognize ASL [19]. Although, the final product that people can use, is still far, the further implementation of other AI models along with MediaPipe can do the tricks.

V. CONCLUSION

The main conclusion from this paper is that the hand tracking provided by Google's Mediapipe is efficient enough to detect multiple Hand Gestures. The real-time implementation can be done with the help of simple mobile cameras or a Webcam. The videos that have higher frames per second (higher shutter speed) work more efficiently when trained. This is how we have overcome the costly methods of hand detection with a simpler approach. We have only trained our model with 5 words of 126 videos each due to time and computational limitations. With a larger and augmented dataset and higher system configurations, we can even train the model for sentences instead of words. One of the drawbacks this research faces is that we are unable to get the full arm landmarks along with upper body landmarks. If some other AI-based approach can be implemented with this approach we are sure this would surely become ready for public use.

VI. FUTURE SCOPE

There are various ways this research can be improved in the future. Firstly, the AI algorithm and the dataset used in this paper can be greatly improved. CNN or other attention mechanism-based models can also be used to implement sign language. Besides that, the dataset used in the paper is not created by a professional sign language interpreter. Also, more research needs to be done on video augmentation to get better results. More datasets can also improve the real-time accuracy of our model. User video of real-time implementation can be recorded for incrementing the dataset size in the future.

We haven't been able to create a model that can predict the whole sentence depicted by the user. This can be done easily with the introduction of other models which can detect the position of elbows, shoulders, wrists and face. TensorFlow produced a package named PoseNet [20], used to get the posture coordinates of 17 body landmarks. Dlib [21] is another python library that can be used to get the facial expression coordinates. It tracks 64 points of the face for each frame. So, combining Hand coordinates along with PoseNet and Dlib to get body posture and facial expression and creating a dataset out of them can prove to be very helpful for future research works on this topic.

REFERENCES

[1] Arpit Mittal, Andrew Zisserman, Philip HS Torr, "*Hand detection using multiple proposals*", The British Machine Vision Conference, Vol.**40**, pp.**75.**1–**75.11**, **2011**.

[2] Ruchi Manish Gurav, Premanand K. Kadbe, "*Real time finger tracking and contour detection for gesture recognition using OpenCV*", In the proceedings of 2015 International Conference on Industrial Instrumentation and Control, ICIC 2015, pp. **974**–**977**, **2015**, isbn: 9781479971657, doi: 10.1109/IIC.2015.7150886

[3] Sarfaraz Masood, Adhyan Srivastava, Harish Chandra Thuwal, Musheer Ahmad, "*Real-Time Sign Language Gesture (Word) Recognition from Video Sequences Using CNN and RNN*", Intelligent Engineering Informatics, Ed. by Vikrant Bhateja, Carlos A. Coello Coello, Suresh Chandra Satapathy, Prasant Kumar Pattnaik., Springer, Singapore, pp. **623–632**, **2018**, isbn: 978-981-10-7566-7

[4] Kirsti Grobel and Marcell Assan, "*Isolated sign language recognition using hidden Markov models*", In the proceedings of

1997 IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, Vol.**1**, pp. **162-167**, **1997**, doi: 10.1109/ICSMC.1997.625742

[5] Pradeep Kumar, Himaanshu Gauba, Partha Pratim Roy, Debi Prosad Dogra, "*Coupled HMM-based multi-sensor data fusion for sign language recognition*", Pattern Recognition Letters, Vol.**86**, Pages **1-8**, **2017**, ISSN 0167-8655, doi : /10.1016/j.patrec.2016.12.004

[6] S. A. Mehdi and Y. N. Khan, "*Sign language recognition using sensor gloves*," In the proceedings of the 9th International Conference on Neural Information Processing, 2002, *ICONIP '02.*, Vol.**5**, pp. **2204-2206,** **2002**, doi: 10.1109/ICONIP.2002.1201884.

[7] J. Gałka, M. Mąsior, M. Zaborski, K. Barczewska, "*Inertial Motion Sensing Glove for Sign Language Gesture Acquisition and Recognition*,", IEEE Sensors Journal, Vol.**16**, pp. **6310-6316**, **2016**, doi: 10.1109/JSEN.2016.2583542.

[8] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, Matthias Grundmann, "*MediaPipe Hands: On-device Real-time Hand Tracking",* Google AI Blog, **2020**.

[9] Ivan Vasilev, Daniel Slater, Gianmario Spacagna, Peter Roelants, Valentino Zocca*, "Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow*", Packt Publishing Ltd, pp. **198-212**, **2019**.

[10] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, Cl. Chang, MG. Yong, J. Lee, WT. Chang, "*Mediapipe: A framework for building perception pipelines*", Google AI Blog, **2019**, doi:1906.08172.

[11] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, Ronald M, "*Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning*", IEEE Transactions on Medical Imaging, Vol.**35**, Issue.**5**, pp. **1285-1298**, **2016**, doi: 10.1109/TMI.2016.2528162.

[12] Saurav Singla, Anjali Patel, "*Comparative Study of the Deep Learning Neural Networks on the basis of the Human Activity Recognition*", International Journal of Computer Sciences and Engineering, Vol.**8**, Issue.**11**, pp.**27-32**, **2020**.

[13] J. Sun, J. Wang, T. C. Yeh, "*Video understanding: from video classification to captioning*". In the Proceedings of the Computer Vision and Pattern Recognition, Stanford University, pp.**1-9**, **2017**.

[14] H. Li, J. Li, X. Guan, B. Liang, Y. Lai, X. Luo, "*Research on Overfitting of Deep Learning*," In the proceedings of *2019* 15th International Conference on Computational Intelligence and Security (CIS), pp. **78-81**, **2019**, doi: 10.1109/CIS.2019.00025.

[15] S. Bock, M. Weiß, "*A Proof of Local Convergence for the Adam Optimizer*," In the proceedings of 2019 International Joint Conference on Neural Networks (IJCNN), pp. **1-8**, **2019**, doi: 10.1109/IJCNN.2019.8852239.

[16] N. D. Marom, L. Rokach, A. Shmilovici, "*Using the confusion matrix for improving ensemble classifiers,"* In the proceedings of 2010 IEEE 26-th Convention of Electrical and Electronics Engineers in Israel, pp. **000555-000559**, **2010**, doi: 10.1109/EEEI.2010.5662159.

[17] D. Zhang, J. Wang, X. Zhao, X. Wang, "*A Bayesian Hierarchical Model for Comparing Average F1 Scores*," In the proceedings of 2015 IEEE International Conference on Data Mining, pp. **589-598**, **2015**, doi: 10.1109/ICDM.2015.44.

[18] M. Genovese, E. Napoli, N. Petra, "*OpenCV compatible real time processor for background foreground identification*," In the proceedings of 2010 International Conference on Microelectronics, pp. **467-470**, **2010**, doi: 10.1109/ICM.2010.5696190.

[19] Fatima Ansari, Anwar Hussain Mistry, Yusuf Mirkar, Alim Merchant, "*Real Time ASL (American Sign Language) Recognition*", International Journal of Computer Sciences and Engineering, Vol.**7**, Issue.**2**, pp.**848-851**, **2019**.

[20] S. Singh et al., "*Action Replication in GTA5 using Posenet Architecture with LSTM Cells*,"In the proceedings of 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM), pp. **544-549**, **202**1, doi:10.1109/ICIEM51511.2021.9445358.

[21] S. Sharma, K. Shanmugasundaram and S. K. Ramasamy, "*FAREC — CNN based efficient face recognition technique using Dlib," In the proceedings of* 2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), pp. **192-195**, **2016**, doi: 10.1109/ICACCCT.2016.7831628.

AUTHORS PROFILE

Mr. Souradeep Ghosh is currently pursuing Bachelor of Technology in Electrical Engineering from Heritage Institute of Technology, Kolkata, India. His research interest lies in Machine Learning, Artificial Intelligence, Computer Vision and Natural Language Processing.

