Word Level Sentence Generation using Deep Learning for Indian Sign Language

Palash Kamble
Department of Electrical Engineering
Indian Institute of Science, Bangalore, India
palashm@iisc.ac.in

Dr Rathna G N
Department of Electrical Engineering
Indian Institute of Science, Bangalore, India
rathna@ee.iisc.ernet.in

***Abstract*—Indian Sign Language refers to the language used by the speech impaired (deaf-dumb) population to communicate in the Indian Subcontinent. However, it becomes difficult for the general population to communicate well with the speech-impaired population primarily due to a lack of knowledge of sign language. So to bridge this gap of communication between the general and the speech-impaired people, in this project work, we have tried to recognize the Indian Sign Language word-by-word and thereby generate entire sentences. This project has focused on one-way communication, where the general people can understand speech impaired people with the help of our state-of-the-art models. We have attempted a three-step approach. In the first step, key points extraction of face, pose, left hand, and right hand is done from the video frames. Then in the second step, the extracted key points are given to the deep learning model such as LSTM or Transformer's Encoder to recognize which sign language (i.e., word) the key points belong. And in the final step, the recognized words are appended to the list to generate the desired sentence.**

I. Introduction

Indian Sign Language(ISL) is the sign language frequently used by the speech-impaired population for communication purposes in the Indian Subcontinent. ISL includes gestures of characters (A-Z), digits (0-9), words (commonly occurring English words). Due to a plethora of gestures used by speech-impaired people, it becomes difficult for the general population to remember each gesture and to be able to communicate well with the speech-impaired population. To date, not many state-of-the-art algorithmic models are present to solve this task of bridging the communication gap between general and speech-impaired people. This project aimed to find the limitations in the existing method and provide new approaches to achieve sentence generation tasks by capturing the gestures used in sign language. Our code for ISL recognition and sentence generation is available here

The problem statement of our work is stated as follows: Given video frames for each word, build a deep learning model to learn the sign language (or gestures) associated with the word from the video frames. With the help of a trained model, capture the words recognized from the gestures to generate an intended sentence in real-time.

II. Related Works

With the rise of deep learning for computer vision and natural language processing over the past decade, very little work has been done in translating and understanding Indian Sign Language. Most of the work established till now in the context of ISL included recognizing the gestures or sign language, either with the help of deep learning or some powerful machine learning models. The work done by Raheja et al. [1] introduced sign language recognition using SVM, which deals with dynamic gesture recognition of ISL. The work done by Hore et al. [2] used an optimized neural network for Indian Sign Language recognition. The work done by Sruthi et al. [3] introduced deep-learning based approach for the Indian Sign Language recognition task, but their work dealt with static gestures. For both static and dynamic gestures recognition, Pratik et al. [4] also introduced deep learning based approach using CNN models. Most of the recognition-based tasks are achieved using state-of-the-art CNN models. More recently, for sentence generation-related tasks, work done by Pratik et al. [5] introduced ways to translate a video sequence to its associated sentence using deep learning in the real-time scenario.

III. Limitations in previous work

The work done by Pratik et al. [5] achieved excellent results for translating video sequences to their associated sentences. But their work has the following limitations:

- As sentence prediction is made from the video sequence, not much sentence-level data is available to train the data-hungry models. A massive amount of data is required to learn sentences of varying sizes. Since the dataset was self-generated by them, it becomes inefficient to generate sequential video data to capture almost all kinds of sentences. Their work generated a dataset consisting of 96 videos for 12 different sentences. For the model to generalize well, a considerable amount of data is required.
- The second major limitation of their work is there is no correlation between words in the predicted sentence and the set of frames in the video. That is, it is not possible by the model to segregate the set of frames in the video to their associated words.

IV. Proposed Method

Reference [5] predicted a sentence from the video without the knowledge of words associated with the set of frames in the video. Their video dataset consisted of sign language associated with the entire sentence. Thereby making it difficult

to interpret which set of frames belong to which words. In our work, we proposed a word-level sentence generation approach. That is, in the real-time scenario, we recognize a word from the incoming set of frames, appending it to the list of predicted words, thereby generating the intended sentence. In a way, our work pipeline combines sign language recognition and sentence generation tasks to do sentence translation.

Deep Learning based sign language recognition tasks established by Sruthi et al. [3] and Pratik et al. [4] make use of Convolutional Neural Network(CNN) and LSTM models. Due to the complexity of CNN's and being inefficient at run-time, our work proposes making sign language recognition without using any explicit CNN architecture. We use feature keypoints extracted per frame from a video using the mediapipe package [11] and feed the extracted keypoints per frame to the sequential model to predict the associated word.

We used LSTM based sequential model [13] and encoder layer from transformer [8] to establish sign language recognition tasks. Between these two models, the transformer encoder is giving better performance.

V. Technical Details

A. Dataset Generation

The Indian Sign Language dataset we created is replicated from the freely available ISLRTC New Delhi YouTube page. The playlist found on the youtube channel consists of more than four thousand words and their associated gestures. Since there happened to be only one sample video consisting of sign language for each word, it became a need to generate a video on our own for each word so that we can create more dataset which is ideally suitable for data-hungry deep learning models. We made 50 videos per 20 gestures. Each video has 20 frames of sequential data, i.e., sign language. Then with the help of the mediapipe package, we extracted keypoints associated with each gesture per frame. We extracted keypoints from the face, pose, left hand, and right hand for each frame. A total of 1662 keypoints are extracted per frame. That is, we get a tensor of shape 20x1662 per gesture (or per video). We then feed this sequential data with a sequence length of 20 to the sequential model to recognize that particular video.

We also created the test dataset separately, but with different conditions, i.e., we created half the dataset in dim light conditions, and the remaining half was created in normal light conditions. We made 10 videos for each gestures. A total of 200 videos were made for testing purpose.

Dataset logistics for training is shown below in table I:

TABLE I
Training Dataset Logistics

TotalGestures	FramesPerVideo	KeyPointsPerFrame	TotalVideos
20	20	1662	1000

Following are the words we worked on - 'believe', 'call', 'can', 'chat', 'do', 'happen', 'happy', 'hat', 'have', 'help', 'I', 'idea', 'it', 'next', 'no', 'see', 'time', 'understand', 'what', 'you',

B. Model Architectures

We worked on two different architectures -

1) LSTM-RNN Classifier [13]
2) Transformer Encoder Classifier [8]

The work pipeline for these two architectures is described separately in the following sub-sections.

1) LSTM-RNN Classifier: LSTM-RNN classifier is the first model architecture we worked on. The workflow of the entire model is shown in Fig. 1.

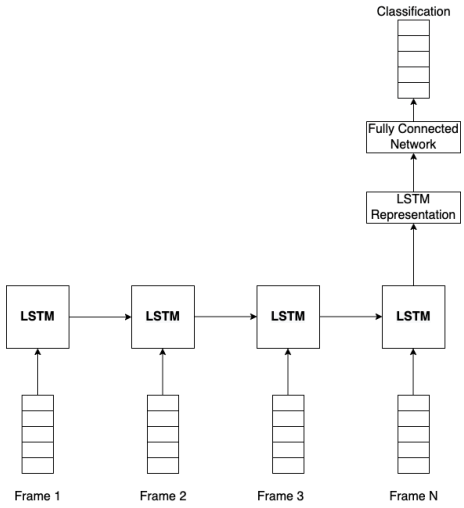

Fig. 1. LSTM-RNN Classifier

The keypoints data extracted from a video dataset having 20 frames have a tensor of shape (20x1662). So we have 20 such sequential input vectors (each vector having size 1662) to be fed to our LSTM-RNN architecture.

During time-step t, the vector input X_t from frame t, is fed to the LSTM block along with the additional inputs - hidden vector H_{t-1} obtained from time-step $(t-1)$, and memory cell state vector C_{t-1} obtained from time-step $(t-1)$ from the previous LSTM block. The output of the LSTM block at time-step t will be the hidden vector H_t and the memory cell state vector C_t.

Let, f_{LSTM} be the function representing the LSTM block that computes hidden vector and cell state vector at time step t, as shown in (1).

$$H_t, C_t = f_{LSTM}(X_t, H_{t-1}, C_{t-1}) \tag{1}$$

where H_0 and C_0 are zero initialized vectors.

After the entire sequence is fed to the LSTM-RNN, the hidden vector, H_N, which we call it LSTM representation, obtained at last time-step N (in our case N=20), is fed to the fully connected layer.

Let, f_{FC} be the function representing the fully connected layer network, which takes in the obtained hidden representation from last time-step, as shown in (2). The output of

this function is a vector of dimension equal to the number of classes (in our work, num of classes is equal to 20, since we have 20 gestures).

$$predictions = f_{FC}(H_N) \tag{2}$$

The obtained vector $predictions_{1x20}$ is then fed to the softmax layer, which outputs a probability distribution, as shown in (3)

$$predictions_i = \frac{\exp(predictions_i)}{\sum_j \exp(predictions_j)} \tag{3}$$

We then make predictions from this newly generated vector $predictions$. The class which has highest probability is chosen as the predicted class.

2) Transformer Encoder Classifier: The transformer model [8] is generally suitable for the sequence to sequence tasks. It has two components encoder and decoder. The encoder component takes in the input sequence and encodes it. The decoder component decodes the encoded sequence to its corresponding output sequence. But since we are doing a recognition task, we only leveraged the encoder component of the transformer model. The model pipeline of our work is shown in Fig. 2.

The keypoints data extracted from a video dataset having 20 frames have a tensor of shape (20x1662). This input tensor is added along with the tensor of positional encodings of shape (20x1662). The positionally encoded input tensor is then fed to the transformer-encoder classifier. This positionally encoded input tensor captures the position of vectors in the input sequence.

Let X be the input tensor of shape (20x1662)

Let P be a vector defined as -

P = [0,1,2,...,f], where f = number of frames per video (in our work f=20)

Let f_{Embed} represent the function to compute embeddings of dimension 1662, which takes in the vector P. The output is a tensor of shape (f x 1662), i.e., in our case it is (20x1662). Equation (4) shows the computation of positional encodings.

$$pos\_encodings = f_{Embed}(P) \tag{4}$$

Then we add our input tensor X to the obtained positional encodings. The newly generated input which we call it positionally encoded input tensor (5) is then fed to the transformer-encoder classifier.

$$X\_pos\_encodings = X + pos\_encodings \tag{5}$$

The obtained positionally encoded input tensor from (5) is fed to the encoder block as shown in Fig. 2. We used six encoder blocks as opposed to the paper [8] which uses eight encoder blocks because the vector dimension that the transformer is expecting (1662 in our case) should be divisible by the number of encoder blocks.

Let $f_{E(i)}$ represent the ith encoder function,
where i $\in$ {1,2,3,4,5,6}

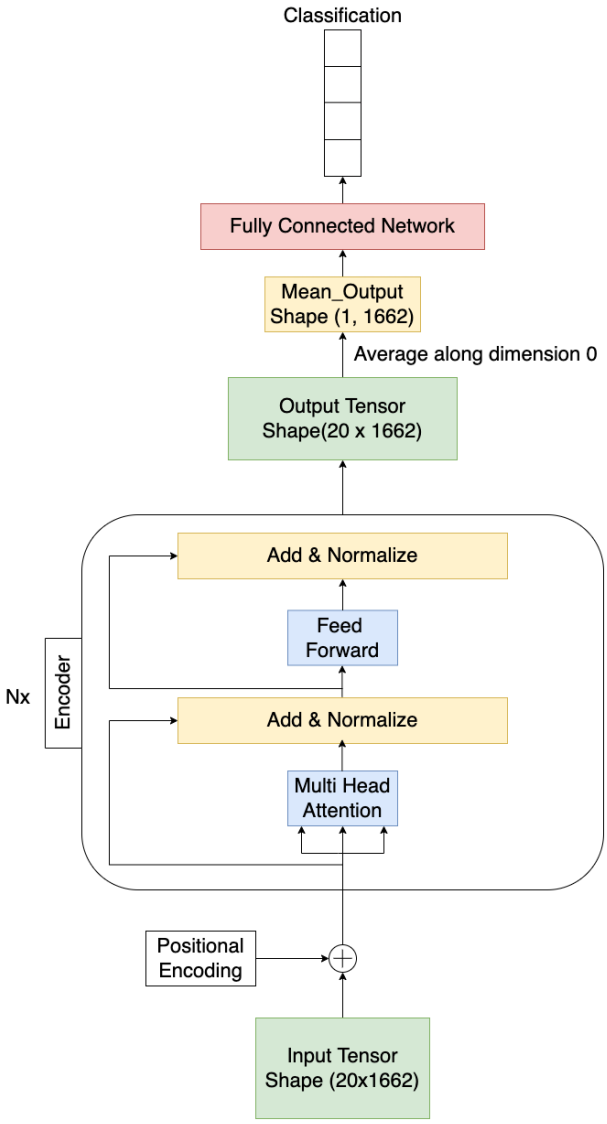

Fig. 2. Transformer Encoder Classifier

The input to the first encoder function $f_{E(1)}$ will be X_pos_encodings

$$out_2 = f_{E(1)}(X\_pos\_encodings) \tag{6}$$

Then input to each subsequent encoder block is the output obtained from the previous encoder block.

$$out_{i+1} = f_{E(i)}(out_i), i \in 2,3,4,5,6 \tag{7}$$

The output obtained from the last encoder block, (i.e., out_7 in our case) is a tensor of shape, the same as the input tensor, i.e., (20x1662).

We then compute the average of the obtained output tensor along the zeroth dimension (i.e., along the sequences) to get the mean output having dimension as 1662.

$$mean\_output_j = \sum_{i=0}^{19}(out_7(i,j)), \forall j \in 0, 1, ..., 1661 \quad (8)$$

This mean output obtained from (8) is then fed to the fully connected layer.

$$predictions = f_{FC}(mean\_output) \quad (9)$$

where, f_{FC} represents the fully connected network function.

The obtained output from (9) which is a vector having dimension equal to number of classes (i.e., gestures) is fed to the softmax layer to get the classification scores.

$$predictions_i = \frac{\exp(predictions_i)}{\sum_j \exp(predictions_j)} \quad (10)$$

We then make predictions from this newly generated vector $predictions$ obtained from (10). The class which has highest probability is chosen as the predicted class.

C. Loss function

We used cross entropy loss as our loss function since we are dealing with multi-class classification problem. Equation (12) shows the cross-entropy loss function -

Let f_L represent the loss function, which takes in two arguments, the predicted output $predictions$ and the true output $targets$.

$$loss = f_L(predictions, targets) \quad (11)$$

$$f_L = -\sum_{c=1}^{C} targets_c * log(predictions_c) \quad (12)$$

where C is the number of classes (in our case C=20)

VI. Experiments and Results

A. On LSTM-RNN Classifier

We built a two-layer LSTM-RNN model in PyTorch and trained the model with the help of hyperparameter optimization using Grid Search. We obtained the following optimal hyperparameters -

TABLE II
OPTIMAL HYPERPARAMETERS FOR LSTM-RNN

Epochs	BatchSize	LearningRate	Optimizer	HiddenSize
450	16	1e-5	Adam	128

We split the training set into training and validation sets using the split factor of 0.8. We trained the model using the optimal hyperparameters from table II. The training data vs validation data loss plot and accuracy plot is shown in the Fig. 3

The classification report obtained on test dataset is shown in Fig. 4

We can see from Fig. 4, the accuracy achieved on test dataset is **54%**. The reason for the low accuracy of the test dataset is that the test dataset was created separately involving different conditions, such as half the videos in a gesture were taken in dim light conditions, and the remaining videos were taken in normal light conditions. Also, the major reason for the low accuracy, which can be seen from the classification report, is that zero predictions were made for class **I** and class **what**. The most minor predicted class is **you**, with an f1-score of 0.16. And the highest predicted class is **do** with an f1-score of 0.95.

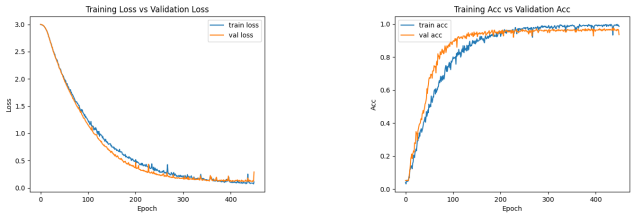

Fig. 3. Left Image: Loss Plot; Right Image: Accuracy plot

	precision	recall	f1-score	support
believe	1.00	0.30	0.46	10
call	0.60	0.90	0.72	10
can	0.62	1.00	0.77	10
chat	0.60	0.90	0.72	10
do	1.00	0.90	0.95	10
happen	0.69	0.90	0.78	10
happy	1.00	0.50	0.67	10
hat	0.00	0.00	0.00	10
have	1.00	0.40	0.57	10
help	0.83	0.50	0.62	10
I	0.00	0.00	0.00	10
idea	0.46	0.60	0.52	10
it	0.90	0.90	0.90	10
next	1.00	0.30	0.46	10
no	0.15	0.40	0.22	10
see	1.00	0.10	0.18	10
time	0.82	0.90	0.86	10
understand	0.29	1.00	0.44	10
what	0.00	0.00	0.00	10
you	0.13	0.20	0.16	10
accuracy			0.54	200
macro avg	0.61	0.53	0.50	200
weighted avg	0.61	0.54	0.50	200

Fig. 4. Classification report on test dataset for LSTM model

B. On Transformer Encoder Classifier

We performed similar Grid Search Hyperparameter optimization technique and found the following optimal hyperparameters -

TABLE III
OPTIMAL HYPERPARAMETERS FOR TRANSFORMER ENCODER

Epochs	BatchSize	LearningRate	Optimizer	HiddenSize
100	16	1e-4	Adam	128

Then we trained the model, using optimal hyperparameters from table III, and found the loss and accuracy plots shown in Fig. 5.

The classification report obtained on test dataset is shown in Fig. 6

We can see from the classification report 6, we achieved the test accuracy of **70%**, which is much better compared to our first trained LSTM-RNN model. We can even see from the classification report that the f1-scores of classes **I** and **what** are not zero, compared to our previous models output. Rather, we could see that all of the classes have been predicted.

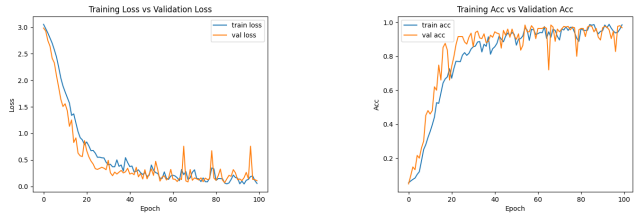

Fig. 5. Left Image: Loss Plot; Right Image: Accuracy plot

	precision	recall	f1-score	support
believe	1.00	0.60	0.75	10
call	0.56	1.00	0.71	10
can	0.50	0.70	0.58	10
chat	0.56	1.00	0.71	10
do	0.77	1.00	0.87	10
happen	0.77	1.00	0.87	10
happy	1.00	0.20	0.33	10
hat	0.91	1.00	0.95	10
have	1.00	0.30	0.46	10
help	0.88	0.70	0.78	10
I	1.00	0.90	0.95	10
idea	1.00	0.90	0.95	10
it	1.00	1.00	1.00	10
next	0.67	0.20	0.31	10
no	0.11	0.10	0.11	10
see	0.25	0.10	0.14	10
time	0.77	1.00	0.87	10
understand	1.00	1.00	1.00	10
what	1.00	0.40	0.57	10
you	0.43	1.00	0.61	10
accuracy			0.70	200
macro avg	0.76	0.70	0.68	200
weighted avg	0.76	0.70	0.68	200

Fig. 6. Classification report on test dataset for Transformer Encoder model

The confusion matrix summarizing the classification report is shown in Fig. 7

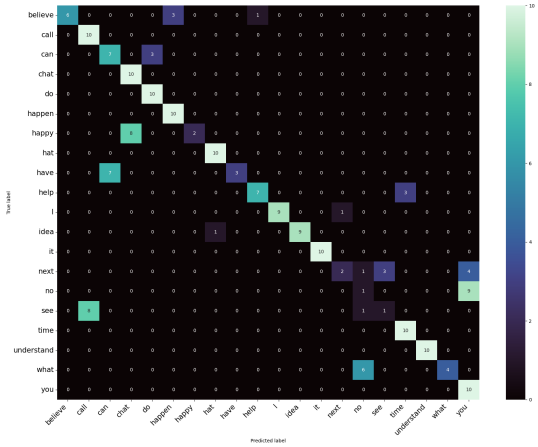

Fig. 7. Confusion matrix on predictions of test set for Transformer Encoder model

From the confusion matrix 7 and also from the classification report 6, it is evident that the class **no** is badly predicted.

VII. Real time testing

Our work pipeline for sentence generation in real time scenario is shown in Fig. 8

From the continuous live video feed occurring in real-time, we fetch the last 20 frames from the current time step and

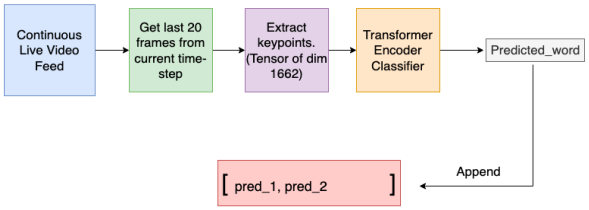

Fig. 8. Workflow to generate sentence

extract key points from all the frames. So, we get a tensor of shape (1x1662). This input tensor is fed to the transformer encoder classifier. Since the transformer encoder classifier performed better in our case, we use that in the real-time scenario. Then we get the predicted word as the model's output. We then run the following two step algorithm before appending the predicted word to the list of sentence generation.

Algorithm to append predicted word to the list of sentence generation -

1) Check if the predicted word probability is greater than the threshold value. We have put a threshold value equal to 0.75. If the probability of the predicted word is less than the threshold value, we discard the predicted word and continue fetching frames from the next time step. Else goto step 2.
2) Check if the predicted word is equal to the last predicted word from the list. If it is the same, then we discard that predicted word, else we append it to the list.

Table IV shows the word predictions done in real-time scenario.

TABLE IV
WORD PREDICTIONS DONE IN REAL-TIME SCENARIO

Predicted Words	
hat	believe
time	chat
do	idea
understand	

The following table V shows sentence generated from the words from the above table IV

TABLE V
SENTENCE GENERATION

Sentence
"hat believe time chat do idea hat understand"

The demo video of the same sentence generation can be found here. Although the sentence generated is not meaningful, but with more predicted words meaningful sentences can be generated with our real-time sentence generation workflow.

VIII. Conclusion

We proposed a novel approach for sentence generation tasks in the context of Indian Sign Language by first recognizing

the sign language with the help of detected keypoints and then appending the generated prediction to the list of sentence generation. With our approach, we can create ISL sentences of varying lengths. In our future work, we would build robust models to perform ISL recognition on more than four thousand available words.

REFERENCES

[1] Raheja, J.L., Mishra, A. and Chaudhary, A., 2016. Indian sign language recognition using SVM. Pattern Recognition and Image Analysis, 26(2), pp.434-441.

[2] Hore, S., Chatterjee, S., Santhi, V., Dey, N., Ashour, A.S., Balas, V.E. and Shi, F., 2017. Indian sign language recognition using optimized neural networks. In Information technology and intelligent transportation systems (pp. 553-563). Springer, Cham.

[3] Sruthi, C.J. and Lijiya, A., 2019, April. Signet: A deep learning based indian sign language recognition system. In 2019 International conference on communication and signal processing (ICCSP) (pp. 0596-0600). IEEE.

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

[5] Likhar, P. and Rathna, G.N., Indian Sign Language Translation using Deep Learning. In 2021 IEEE 9th Region 10 Humanitarian Technology Conference (R10-HTC) (pp. 1-4). IEEE.

[6] Sutskever, I., Vinyals, O. and Le, Q.V., 2014. Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

[7] Bahdanau, D., Cho, K. and Bengio, Y., 2014. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[8] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[9] Diederik, K. and Jimmy, B., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, pp.273-297.

[10] Bazarevsky, V., Grishchenko, I., Raveendran, K., Zhu, T., Zhang, F. and Grundmann, M., 2020. BlazePose: On-device Real-time Body Pose tracking. arXiv preprint arXiv:2006.10204.

[11] Zhang, F., Bazarevsky, V., Vakunov, A., Tkachenka, A., Sung, G., Chang, C.L. and Grundmann, M., 2020. Mediapipe hands: On-device real-time hand tracking. arXiv preprint arXiv:2006.10214.

[12] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K. and Grundmann, M., 2019. Blazeface: Sub-millisecond neural face detection on mobile gpus. arXiv preprint arXiv:1907.05047.

[13] Staudemeyer, R.C. and Morris, E.R., 2019. Understanding LSTM–a tutorial into Long Short-Term Memory Recurrent Neural Networks. arXiv preprint arXiv:1909.09586.