Sign Language Recognition: Bolstering the claims made by biometrics – *Real World Applications*

Authors: Udbhav Saxena and Shurti Jha
University at Buffalo, udbhavsa, shrutijh@buffalo.edu
For code, instructions refer to the readme file in the folder

Abstract - There is an undeniable communication problem between the hearing majority and the deaf community. Research in Sign Language Recognition (SLR) can lower this communication barrier. The task of lowering the barrier between the deaf community and the hearing majority has a broad social impact, but it is still very challenging due to complexity and large variations in had actions. If we use static images for Sign Language Recognition, we may only be able to detect some action, but not all. In American Sign Language, there are letters and words, which require actions rather than just a stationary gesture. So, in this project, we work on making system, which can detect actions. To approach this problem, we use Long Short-Term Memory (or LSTMs) which can extract temporal features from different frames of the action.

INTRODUCTION

There are a lot of people who are unable to hear or speak. For them, the most feasible and reliable option for communication is to use sign language. Sign language can be hard to interpret especially for someone who does not know the semantics. But this should not be a reason why one, who cannot communicate verbally, should not be able to communicate their ideas.

Sign Language is the only means via which deaf community can communicate. With the advances in Vision and Deep Learning, many researchers are now working in this area to solve this problem, not only to minimize the problems of deaf community but also to implement it in different fields such as authentication in the metaverse. Finding an experienced and qualified interpreter every time is difficult and expensive. Moreover, most of the people who are not deaf or does not know someone who is deaf, never try to learn sign language for interacting with deaf people. This becomes a cause of concern for deaf people. This can make them feel isolated from the community just because the other person cannot understand the semantics of sign language.

One might think that coming up with a sign language detection would be a simple task, as we could just use state-of-the-art Convolutional Neural Networks and train our model on a lot of training images and make it robust. Well, it is not as simple as it sounds. There are certain alphabets, or gestures which require motion and thus contains a temporal component. To deal with this specific problem, we employ Long Short-Term Memory (LSTMs).

PRIOR WORK

In this paper [1], the recognition was made using data gloves, location trackers etc. to measure the joint angle information, the track in the space and timing information of hands to recognize sign language. Fang G, Gao W, and Zhao D use three location trackers and two data gloves to obtain hand features and reach an average SLR rate of 91.9%. This kind of method can obtain relatively high recognition rate, but the user must wear complex data gloves and location tracker which has a bad effect on the convenience of human-computer interaction. Besides, these input devices are too expensive to promote to wide range of applications.

Figure 1 Data Gloves [1]

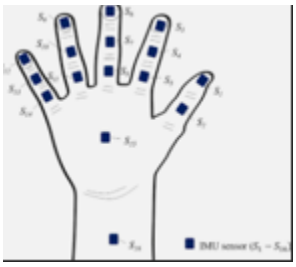

Figure 2 Sensors in Data Gloves [2]

This was by far the most intuitive method. It makes use of fundamentals of linear algebra concepts of Eigen Vectors and Euclidean Distance. According to the figure 3, the first step in the block diagram is skin filtering. In skin filtering, the filter extracts the skin-colored pixels from the non-skin-colored pixels. This method is very much useful for detection of hand, face, etc. Skin Filtering encompasses the hand cropping and binary image part too. After obtaining the binary image, we do hand to hand cropping by taking the binary image co-ordinates to crop the original image we used in the input-phase. After desired position of the image is cropped, we find the eigen vectors and eigen values of that cropped image and arrange it in a decreasing order. Finally, the classification step. This includes the Euclidean distance between the eigen vectors of test image and the input image and the minimum of each was found out to classify them.

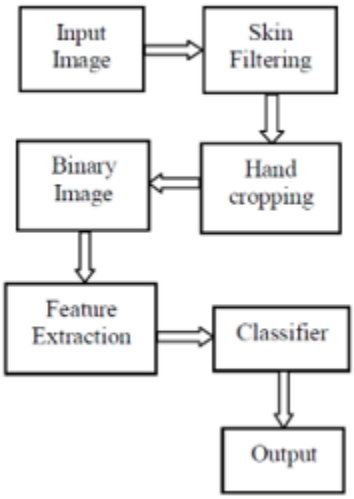

Figure 3 Block Diagram of the Model [2]

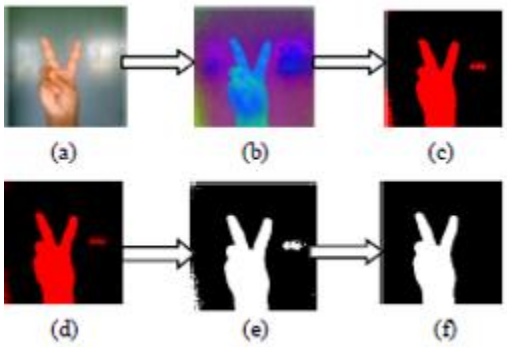

Figure 4 Skin Filtering [2]

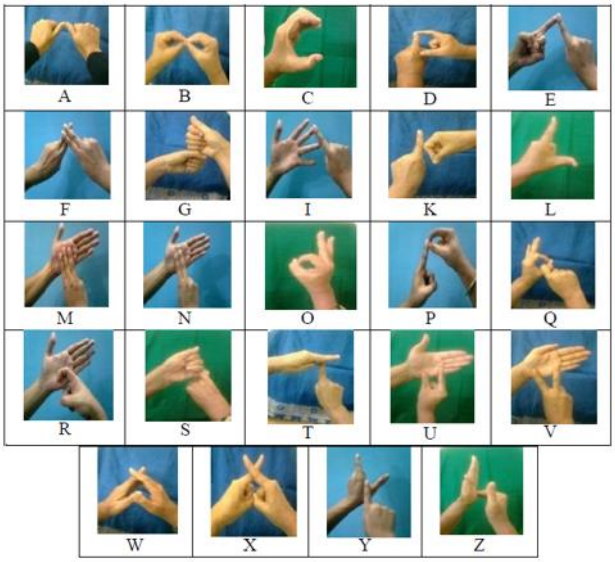

Figure 5 Dataset used for classification [2]

Drawback of this method was that it could not classify dynamic motions. There are certain alphabets in American Sign Language which needs dynamic hand motion. For example, the alphabet J and Z requires hand movement as shown in figure 6. It could only be used for static images. The other drawback is that it needs a special background which makes skin filtering easy. In real world problem, it would not be possible to recreate this every time when one is trying to communicate.

In this paper [3], state-of-the-art Convolutional Neural Networks was used for classification. This method was accurate (83%), and it was also able to make use of traditional cameras which were a non-invasive way to carry out the classification task. The idea was that a kernel window called **local receptive field** moves over each unit from prior layer. Each layer receives inputs from a set of units located in the kernel window. Parameters of kernel windows are forced to be identical for all its possible location of the previous layer, which is called **weight sharing**. The sharing of the weights reduces the number of free-variables and increases generalization capability of the network.

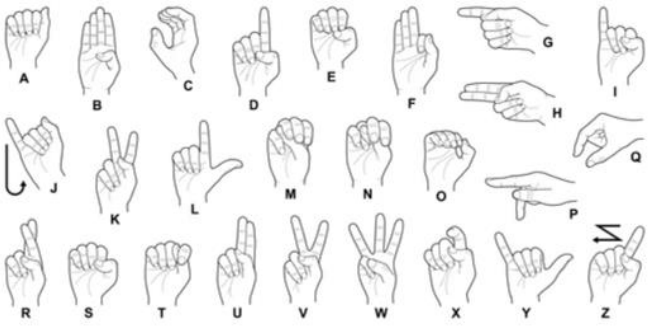

Figure 6 American Sign Language: J and Z requiring motion [3]

BACKGROUND KNOWLEDGE

In our approach we employed the use of LSTMs which gives a temporal component to the existing CNN architecture. One can think of it like a 3D convolutional neural network. LSTNs allows us to use sequence of data for training and testing purposes. This helps us train alphabets such as J and Z which requires motion.

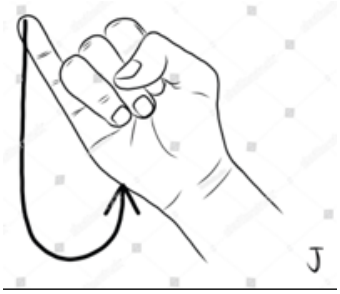

Figure 7 J in Sign Language – motion based [1]

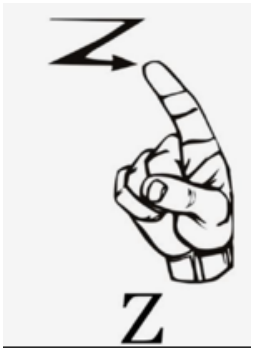

Figure 8 Z in Sign Language – motion based [1]

I. How LSTMs Work

Long Short-Term Memory or LSTM is an improvement over Vanilla Recurrent Neural Network. Normal RNNs face the problem of vanishing gradients and exploding gradients i.e., if we train a deep enough network, our model could face problems such as exploding gradients or vanishing gradients. Exploding gradients occurs when the large error accumulates which results in large value of the weights updates which makes our model very sensitive and hence may not give us a good result. Exploding gradients makes the model unstable. On the other hand, vanishing gradients occurs when there are small updates in the values of weights. In figure 9 (left) there is a recurrent neural network and in Figure 9. (right) there is an LSTM with a memory gate.

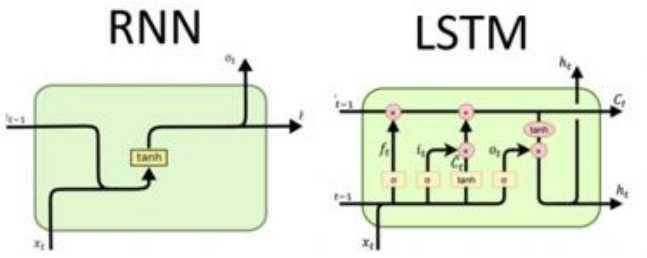

Figure 9 RNN (left) and LSTM (right) [8]

Long Short-Term Memory networks are a type of Recurrent Neural networks that can hold knowledge from previous layers. This helps us model data which needs to refer to output from previous layers. This network is especially useful in applications like machine translation, speech recognition etc. For example, we have text generation, say, we have a sentence which reads "the *man*, who was wearing shorts and who loved biking in the summer, was proud of *his* decision'. For this recurrent network might not be able to keep track of the gender and maybe unable to produce the '*his'* in real-time. To counter this problem, LSTMs will be very useful. LSTM will keep track of the gender in its memory gate and use it to generate the correct gender much later in the sentence. An LSTM can be represented as shown in figure 10.

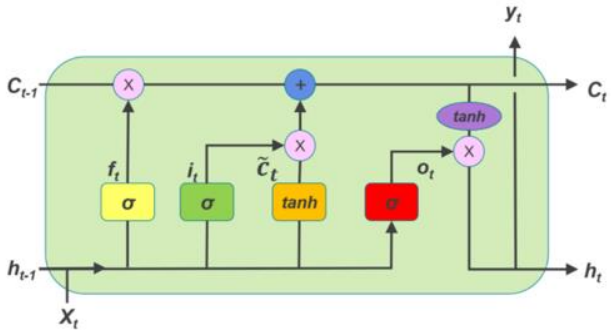

Figure 10 LSTM Structure [8]

The equations concerning the LSTM model are as follows:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \quad (1)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \quad (2)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \quad (3)$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad (4)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \quad (5)$$
$$h_t = o_t \circ \sigma_c(c_t) \quad (6)$$

First comes the forget gate f_t from equation (1), this decides whether we need to keep the information from the previous timestamp or *forget* it. x_t is the input to the current stamp. U_f is the weight associated with the input. (Ht-1) the hidden state of the previous timestamp. W_f is

the weight associated with the hidden state at current timestamp. Then comes the input gate i_t from equation (2). U_i is the weight matrix of the input. W_i weight matrix of input associated with hidden state. After the input gate, there is the output gate o_t from equation (3), it structurally like equations (1) and (2).

OUR APPROACH

Explain in brief what we did, how did we collect data, what was the architecture for our network, training our data, show the dataset, graphs, testing of our data, novel way of using this method, future work,

In this project, we implemented the idea of LSTMs in sign language recognition model and used it to train a soft biometric. A soft biometric is like handwritten signature which can be unique to an individual, but it can be replicated like a handwritten signature. A soft biometric should not be a strong enough marker for identity but for low-security and quick authentications, they do a fair job. The best part about soft biometric, especially sign language, is that it is a non-invasive process. Hard biometrics like fingerprint and face can be reserved for a more security-intense purpose. A real-life example could be, say, we ordered something expensive from an e-commerce company and the delivery man asks for an authentication. Instead of giving a soft signature in-person, we can go ahead and do our unique signature using just actions. This technology, especially when social distancing is necessary, could retrieve signatures from a distance. So, our project mainly had two goals. One being, to recognize sign language and the second being to use it as a soft biometric.

The model was trained to recognize three different signatures which was unique to a person. In total we trained 30 sequences (or videos) per signature. These sequences were in the form of matrix with pixel values. To extract features, we use mediapipe library designed by Google which can detect our hands, face, elbows, knees, and foot. But for signature purposes we only used elbows and hand data as shown in figure 11.

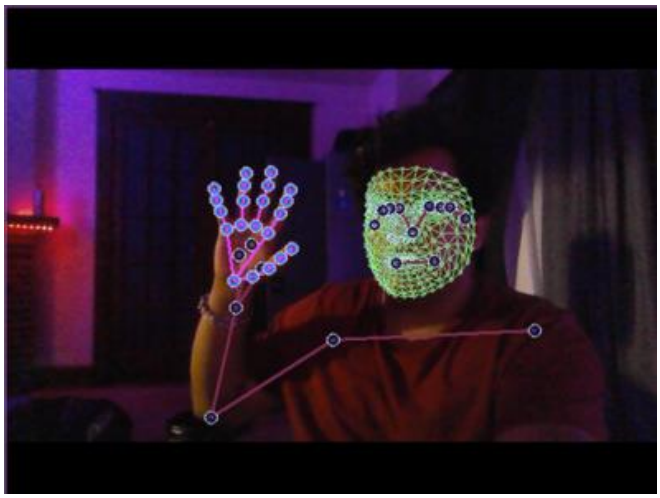

Figure 11 Detection of right hand and elbows using mediapipe

I. Collection of Training Data

With the help of mediapipe we were successfully able to extract *keypoints* from the image which were of interest and use it for training. So, for one label signature or enrolling the biometric, we collected 30 sequences of each sample and for one label, we did this 30 times i.e., for one label (say person's A signature) there were 900 frames extracted in total out of which 30 were bundled into one training sample. To be more coherent, for one signature we collected 30 videos for training purposes. In total, we had three labels for three different individuals, so a total of 90 videos were collected. A glimpse of training data format is shown in figure 12.

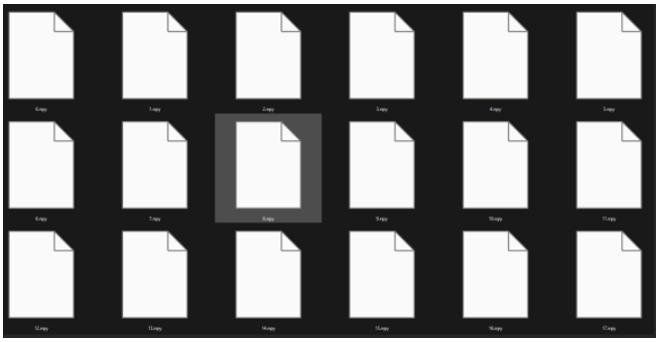

Figure 12 Each .npy file contains frames from the sequences recorded for training

Inside a *.npy* there is an array of the extracted feature points using mediapipe which are represented in figure 13. Training 30 sequence of every label, namely "J", "Z", and "Name" (for project purposes) is shown in figure 14.

```
show = np.load('0.npy')
```

```
show
```

```
array([ 0.54026133,  0.58686006, -0.74591225, ...,  0.        ,
        0.        ,  0.        ])
```

Figure 13 Inside a .npy file - Features extracted from mediapipe

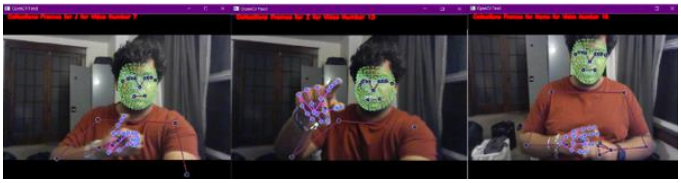

Figure 14 Training for 'J' (left), 'Z' (middle), and 'Name' (right)

II. Architecture

This architecture is a custom architecture. It contains 6 layers, out of which 3 are LSTMs layers and the other three are conventional convolution dense layers. The sequence which was recorded while collection of training data is taken as input. The training data was split into training set and testing set. Out of 90 total sequences (30 in each label and 3 labels) 85 were taken as the training set and the other 5 were used for test set.

Below is a representation of how the one training sample looks like before it goes in the input layers of the neural

network. Since there 85 of such samples, final shape of X_{train} is (85, 30, 1662). Y_{train} is one-hot-encoded and is represented in a matrix as shown in figure 17.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 30, 64)            442112

 lstm_7 (LSTM)               (None, 30, 128)           98816

 lstm_8 (LSTM)               (None, 64)                49408

 dense_6 (Dense)             (None, 64)                4160

 dense_7 (Dense)             (None, 64)                4160

 dense_8 (Dense)             (None, 3)                 195

=================================================================
Total params: 598,851
Trainable params: 598,851
Non-trainable params: 0
```

Figure 15 Custom Architecture of the model used for training the sign language model

```
X_train[0]
```

```
array([[ 0.59635198,  0.62535262, -0.97946262, ...,  0.47256044,
         0.59428394, -0.02303495],
       [ 0.59798056,  0.62500519, -0.95236671, ...,  0.44535553,
         0.57176238, -0.01487184],
       [ 0.60039496,  0.62474322, -0.93966687, ...,  0.4145886 ,
         0.53610456, -0.01346371],
       ...,
       [ 0.58965731,  0.62797928, -0.88418466, ...,  0.46975783,
         0.58249766, -0.02190135],
       [ 0.58969802,  0.62791419, -0.89712858, ...,  0.46949953,
         0.58350736, -0.02195756],
       [ 0.58971828,  0.62772757, -0.91157782, ...,  0.47037151,
         0.58346611, -0.02340985]])
```

```
X_train[0].shape
```

```
(30, 1662)
```

Figure 16 One training (X_{train}) sample description

Letter			
Z	1	0	0
J	0	1	0
S	0	0	1

```
y_train[0]
```

```
array([1, 0, 0])
```

```
y_train[0].shape
```

```
(3,)
```

```
y_train.shape
```

```
(85, 3)
```

Figure 17 Y_{train} Data Visualization

III. Results

For training purposes, we used *Adam optimizer* and for *loss function* we used *categorical crossentropy*. For LSTM input layer we used ***Rectified Linear Unit or ReLU and*** for output layer we used *softmax* since we had more than two output labels.

In this section, we trained three alphabets namely "J", "Z", and "S". Here, we were only training the model to first identify the moving alphabets correctly (refer figure 8 and figure 9) which was not possible using the static image recognition [3]. The accuracy and loss of the model is shown in figure 18.

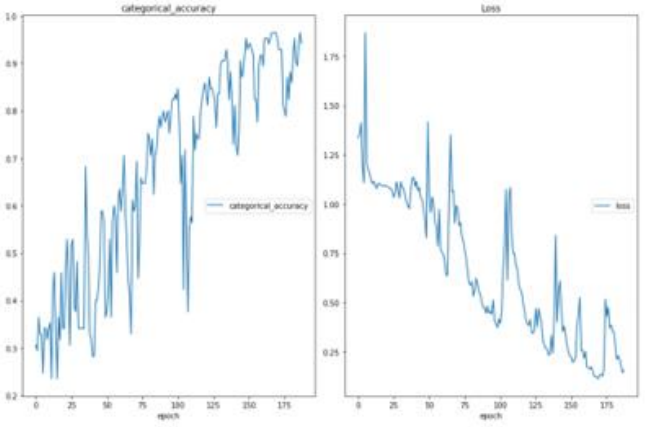

Figure 18 Accuracy (left) and Loss (right)

As shown in figure 18, after using *earlystopping*, the best accuracy we could, for the model was 96.07%. The model did a fair job recognizing all the alphabets correctly with some uncertainty. The real-time results are shown in figure 19.

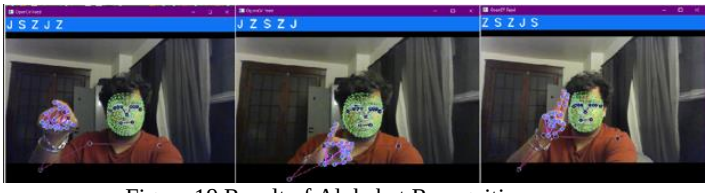

Figure 19 Result of Alphabet Recognition

In this section, we trained the model to form sequences of sentences namely, "My", "Name is", and "[Name]". The "[Name]" part is usually arbitrary as the person from the deaf community are easily able to lip read what the person mouths. Usually, the mouthing part only contains nouns. Since nouns can be arbitrary, this part is usually understood by mouthing the word [*noun*]. In figure 20, the representation of accuracy and loss is given. And in figure 21, the real-time results are shown.

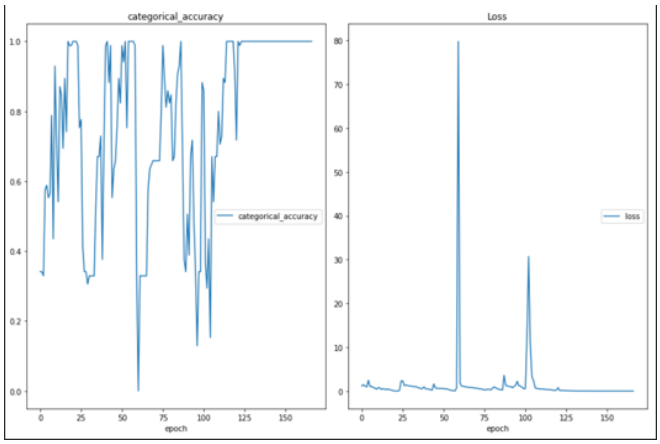

Figure 20 Accuracy (left) and Loss (right) for Sign Language Recognition

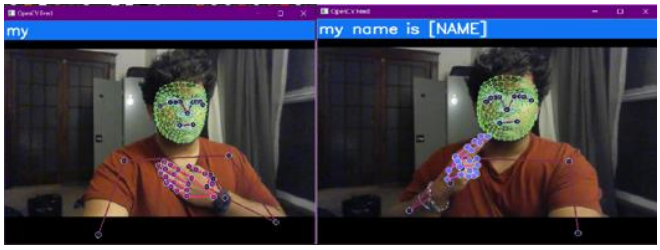

Figure 21 Results of Sign Language Recognition in real-time

Type of Recognition	Accuracy
Alphabet Recognition	86.07 %
Sign Language Recognition	99.87 %

Table 1. Accuracy of different types of recognition

USE IN THE REAL WORLD

Sign Language Recognition (SLR) can not only help the deaf community, but its application can also be spread out to fields like authentication. A lot of dating applications like Tinder, Bumble, Hinge etc. can use this technique for authentication purposes. These dating apps require a photo in specific pose and then that photo is fed into the authentication pipeline which authenticates it. An illustration of the same is shown in figure 22 and figure 23.

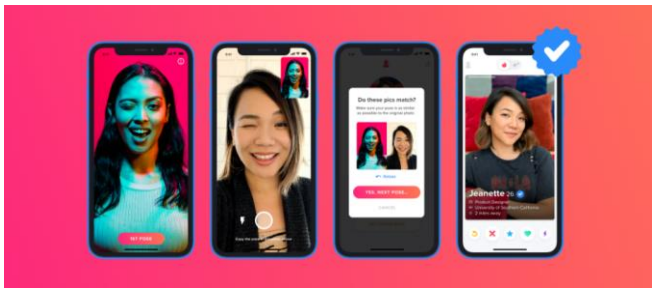

Figure 22. Tinder Authentication [9]

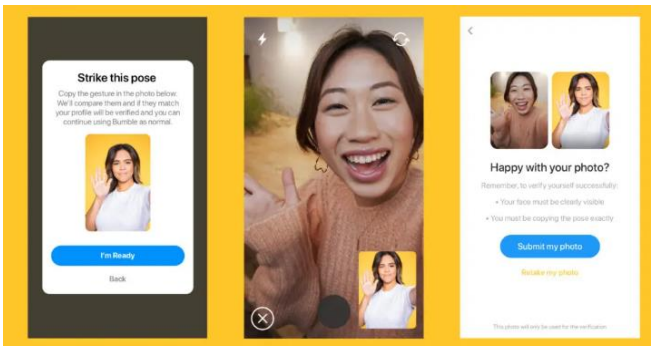

Figure 23 Bumble Authentication [10]

But the problem with current authentication method is that hackers are still able to surpass this check points and can create a fake-authentic profile of someone else. Instead, to overcome this breach, they can use the model proposed above and instead of collecting photos in a specific pose, they can collect a sequence (or video) of the user doing a specific pose. If that matches with the profile that was created (before authentication) they can go ahead are verify the user. A flow chart of the same would look something like shown in Figure 24. A flow like this can improve security and bolster the claims made by biometrics (facial recognition, for example).

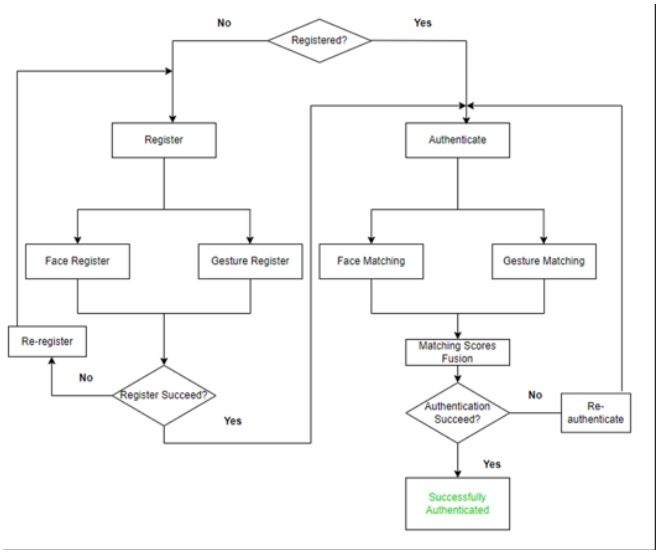

Figure 24 Flow Chart for Authentication using Gestures

Another application of SLR could be to authenticate services in the Meta Verse. In the meta verse, where there are no pens, cameras and only VR, the use of gesture recognition could be used for authenticating services. So, instead of typing the passwords, or using facial recognition, one could just enroll in a gesture signature, and use that gesture for authentication.

FUTURE WORK

While the results do seem promising, but the model is not perfect. Firstly, since we did not have access to the dataset, we had to create our own dataset for modelling. Due computation restrictions, we were only able to process this much amount of data. Because of that the model was only trained on 6 different labels namely, for Alphabet Recognition: 'S', 'J', and 'Z' and for Sign Language Recognition: 'My', 'Name is' and '[NAME]'. Collecting training data only, (without preprocessing it) took about one hour each.

Another improvement that can be done is to use Transformers [4] instead of LSTMs. The problem with LSTMs is that, while it can remember information, it processes the information one-by-one which makes it a bit slow computationally. LSTMs also does not make use of GPUs which can aid the computation by using parallel processing [CUDA]. To overcome this, a transformer can be used. A transformer is an attention-based model. In this model, the decoder has access to all the past states of the encoder. An illustration of the same is shown in the figure below (figure 25.)

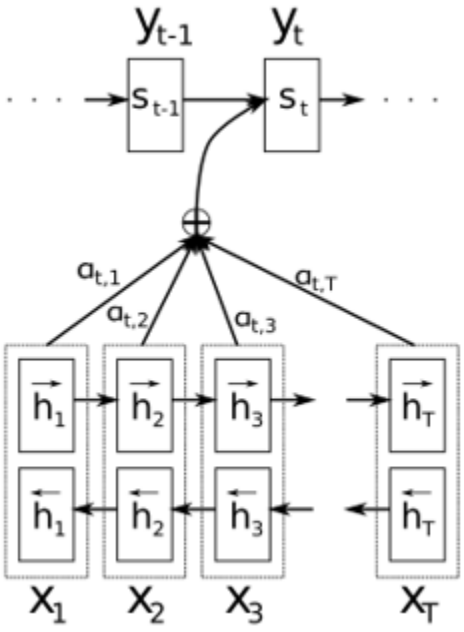

Figure 25 Transformer having access to all past states [8]

The main problem with LSTMs is that they are harder to train when compared to a transformer network, since the number of parameters in LSTMs are very large [8]. One more shortcoming of LSTMs is that it is impossible to use LSTMs for transfer learning. Also, transformers are also computationally less expensive and more accurate.

CONCLUSION

Sign Language recognition is an essential part of sign language translation. Previous methods ([1], [2], and [3]) relied heavily on static data. To overcome that part, we used LSTMs so that we could get access of sequential dimensions. Using the LSTM model, we were able to successfully make a Sign-Language-to-Text model with Alphabet Recognition achieving an accuracy of **96.07%** and for Sign Language Recognition **99.67%.** While the numbers seem promising, it was due to the lack of availability of data. It would have been better if we had access to a dataset for training the same. All the recognitions using the LSTM model were made in real-time.

ACKNOWLEDGMENTS

We would like to thank Prof. Nalini Ratha for giving us a deep understanding of how Biometrics work. Without the help of knowledge, we gained in this course, it would have been a tough task for us to solve.

CONTRIBUTION

Udbhav Saxena: Implementing the model and making the report.
Shruti Jha: Helped in editing code and made the presentations.

ACKNOWLEDGMENTS

[1] Shin, J.; Matsuoka, A.; Hasan, M.A.M.; Srizon, A.Y. American Sign Language Alphabet Recognition by Extracting Feature from Hand Pose Estimation. *Sensors* **2021**, *21*, 5856.

[2] Singha, Joyeeta & Das, Karen. (2013). Indian Sign Language Recognition Using Eigen Value Weighted Euclidean
Distance Based Classification Technique. International Journal of Advanced Computer Science and Applications. 4. 10.14569/IJACSA.2013.040228.

[3] Jie Huang, Wengang Zhou, Houqiang Li and Weiping Li, "Sign Language Recognition using 3D convolutional neural networks," *2015 IEEE International Conference on Multimedia and Expo (ICME)*, 2015, pp. 1-6, doi: 10.1109/ICME.2015.7177428.

[4] Camgoz, Necati & Koller, Oscar & Hadfield, Simon & Bowden, Richard. (2020). Sign Language Transformers: Joint End-to-End Sign Language Recognition and Translation. 10.1109/CVPR42600.2020.01004.

[5] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-Scale Video Classification with Convolutional Neural Networks," 2014 IEEE Conference on Computer Vision and Pattern

Recognition, 2014, pp. 1725-1732, doi: 10.1109/CVPR.2014.223.
[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. https://doi.org/10.1145/3065386

[7] Hochreiter, Sepp & Schmidhuber, Jürgen. (1997). Long Short-term Memory. Neural computation. 9. 1735-80. 10.1162/neco.1997.9.8.1735.

[8] Nadendla, H, *Why are LSTMs struggling to matchup with Transformers? (Oct 2020) https://medium.com/analytics-vidhya/why-are-lstms-struggling-to-matchup-with-transformers-a1cc5b2557e3#:~:text=Conclusion%3A,transfer%20learning%20in%20LSTM%20networks.*

[9] Image: *What is Photo Verification? https://www.help.tinder.com/hc/en-us/articles/360034941812-What-is-Photo-Verification-*

[10] Image: *How to get verified on Bumble? https://bumble.com/en/the-buzz/the-end-of-catfishing-introducing-photo-verification*