Sign Language Recognition Using RNN and LSTM

Corey Martin, Bhav Culleechurn
School of Computer Science
University of Guelph
Guelph, Canada
{cmarti45, bculleec}@uoguelph.ca

***Abstract*—In recent times, advances in computer vision frameworks and neural network models have allowed great progress to be made in automated translation models. Mostly, spoken languages have greatly benefitted from these improvements. However, in the task of Sign Language Recognition (SLR), progress has been slow. SLR aims at converting sign language into text in order to reduce communication barriers with hearing-impaired people in society [1]. For a long time, hearing impaired people have faced barriers in education, media and communication. Our system aims to be accessible by taking joint features from video input and using them to recognize words that are being gestured. We were able to obtain promising experimental results on the WLASL Video dataset, using a Long Short Term Memory (LSTM) and a Recurrent Neural Network (RNN) model.**

Keywords—Recurrent Neural Network, Long Short Term Memory

I. Introduction

The American Sign Language (ASL) is the predominant sign language used by the hearing-impaired community in North America. However, there are important barriers when it comes to learning ASL. Often, teaching support is limited for students enrolled in public schools [3]. This limits the opportunities for social interaction between the ASL fluent and non-fluent community. There is a need for an automated system that can assist in learning and facilitate communication between these two groups.

Automated systems are difficult to model for Sign Language Recognition because it is a complex language composed of hand-shapes, moving hand-joints which also involves facial expressions. Moreover, feature extraction is difficult due to differences in hand shapes and dimensions, and changes in orientation [6]. Methods that have attempted to map these features have included the use of the Kinect system and other motion capture methods [1]. However, the use of external apparatus makes these models less accessible and practical for real-time use. Alternatively, CNNs are often utilized to automate the process of feature extraction [9]. We propose a model that uses MediaPipe Holistic to extract those joint features from regular video inputs and use them to train LSTM and RNN models to identify ASL words that are gestured. In the following sections, we provide an overview of our methodology employed, results obtained and discussion about the findings of this paper.

II. Motivation and Objectives

Automated translation solutions for all spoken languages have reached high levels of accuracy in recent times. There have been advances in the fields of computer vision and action recognition. [12] However, such solutions have been lacking in the domain of sign-language interpretation. This has partially led to societal gaps forming for the hard-of-hearing people such as lacking teaching support in public schools [3]. Our project seeks to reduce these barriers by proposing an accessible and economical model for sign language interpretation. In the all forms of communication, the most important aspects are preserving the original meaning of the message and ensuring timely delivery. Therefore, our primary motive is to create a solution that is highly accurate and that has potential for deployment in real-time applications. To this goal, we perform a comparative study on the performance of 2 types of models, namely RNN and LSTM. We carry out various forms of hyperparameter tuning to optimize the results obtained from both algorithms.

III. Methodology

First, we describe the dataset we used, and how we preprocessed our data. Then we will present the tools we used in our project. Finally, we will provide a brief background on RNN and LSTM, and describe how we constructed and trained our models. As a high-level overview of our approach from start to end, we first extracted frames of video data using OpenCV and obtained the joint specific values from those frames with MediaPipe Holistic. We extracted and encoded our labels and then trained our models with the joint sequences to recognize our labels. We kept our testing set separate and evaluated on it at the end.

A. Dataset Description

The WLASL Video dataset is composed of around 12,000 videos gesturing 2,000 common words [2]. It is the largest dataset available for American Sign Language (ASL) recognition. Each video describes a single word and the corresponding word to video mappings are provided in a glossary file, with additional information about the video such as starting and ending frames and information about the signer. In this paper, we only considered 10 of the words in the dataset. On average, each word had about 7 videos gesturing them.

B. Python Tools Utilized

Our project relies heavily on certain libraries in python that facilitated the task of joint feature extraction, encoding into data frames and building the model.

The primary tool powering this project and making it possible is MediaPipe Holistic. It provides accurate tracking for a variety of joints including face, pose and hand motion in real-time or using pre-recorded video data. MediaPipe has sophisticated cross-platform compatibility and efficient resource control on different platforms [7]. This makes it ideal for deployment in real-time use or on mobile platforms for future work.

We used scikit-learn for constructing our RNN and LSTM models. It is a free software machine-learning framework in Python that provides accessible models for training and testing neural networks.

In order to extract the frames of visual data from the videos in our dataset, we utilized the OpenCV library. OpenCV is commonly used as an open source computer vision and machine learning library. It includes over 2500 algorithms for various forms of image manipulations in Python.

We also used the NumPy library to store our extracted joint feature data. NumPy is one of python's most used libraries for arrays and mathematical functions.

C. *Algorithms*

Briefly, we will provide some background on RNN and LSTM. RNN is a model that unlike simple feedforward neural networks, is able to capture the sequential features of data by capturing previous inputs. However, RNN suffers from the vanishing gradient problem and error blowing problem which reduces its ability to capture the long-term dependencies in data.

LSTM is built upon RNN to solve exactly that problem. They use gates to forget some of the previous inputs and prevent the vanishing gradient problem and are more suited for tasks involving long sequences of data that are defined by temporal distance [4].

D. *Parameter Initialization Methods*

This section describes the parameter initialization methods we used for our models. For the initial weights of our models, we used scikit-learn's default weights initialization. Certain weights initialization methods may be more beneficial for RNN, in order to reduce the vanishing gradient problem. Normally weights may be set to zero or at a steady state and model performance may suffer if states are not correctly initialized [8].

E. *Data Preprocessing*

We had to take various steps to preprocess the data in order to make it suitable for input to our 2 models.

We needed to extract the frames from the videos. In order to reduce the need for padding, and to provide a consistent input shape for our model, we needed to collect the same number of frames from each video. This meant that we could only collect as many frames as the shortest video in the dataset which was 15 frames. We extracted 15 frames from each video taken at consistent intervals from the starting to the ending frame.

Next, we needed to get the features from the frames we extracted which we did by using the MediaPipe Holistic library. Using MediaPipe Holistic, it is possible to extract landmark information from face, pose and hand joints.

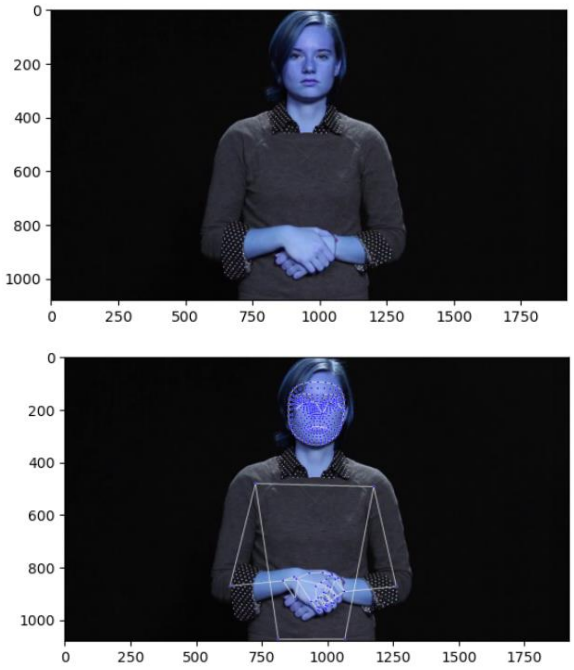

Fig. 1. Frame of sign language video before and after extracting joint features using MediaPipe Holistic

An example of this from one frame of the signing videos can be seen in Fig. 1. As shown, the model accurately maps face, pose and hand joints on the image. This is ideal for us because most sign language gestures use a combination of hand and facial gestures. We detected the positions of face, pose, left hand and, right hand landmarks from each frame to create a sequence corresponding to a word, and appended each sequence into a NumPy array. We then saved these features to make them easy to retrieve.

Further, we needed to perform one-hot encoding on the labels of videos we used. Words provided in the glossary file are strings that cannot be used as output for the training model. Instead of using all 2000 words, we only use 10 words from the glossary. We used a tensorflow utility function to generate one-hot encodings for our 10 class labels.

We also needed to augment our data due to the number of videos for each word not being sufficient. Data augmentation can reduce the risk of overfitting and help with generalization [10]. In order to do this, we performed two types of video augmentation, which were a horizontal flip and a slight random rotation. This allowed us to triple the number of videos and subsequently sample points we had for each word, while also making the model slightly more robust against different camera positionings. This resulted in 357 individual videos for our 10 class labels, or around 30 videos per word.

F. *Building our algorithms*

To build and train our LSTM model, we used scikit-learn. Our model architecture for the LSTM model includes 3 LSTM layers stacked on top of each other and 2 fully connected layers. As a baseline to compare this model to, we use another LSTM model with only one fully connected layer. We call this model LSTM1.

When training our model, we implemented the LSTM model with the Rectified Linear Unit activation function for the LSTM and fully-connected layers, and Softmax on the output layer. We used categorical cross-entropy loss as the optimization function and the Adam optimizer for training with learning rate set to default. We started with 64 nodes in the first fully-connected layer and 64 hidden inputs to the LSTM model.

We first trained our model using different learning rates and over different number of epochs and selected the best values for those two parameters. Then, we tested different values for the number of nodes in the fully-connected layer and the number of inputs in the LSTM layer.

IV. Results

This section outlines the results we obtained.

A. *Hyperparameters*

This section explains the results we obtained for different hyperparameters.

We studied the impact of varying the various hyperparameters that affect the performance of our 2 models. For LSTM, we tested different learning rates for the optimizer in the range of 0.0001 to 1.0. The optimal value we found for the learning rate was the default value of 0.01, as demonstrated in Fig. 2.

We trained both LSTM and RNN models over 500 epochs to find the number of epochs that give the lowest error rates without leading to overfitting. For LSTM, we found that the optimal number of epochs for training was around 230. Training for more epochs resulted in overfitting. RNN was able to reach comparable validation accuracy in less epochs of training.

Following this, we decided to vary the number hidden inputs in the LSTM layer. We used values ranging from 16 to 512 and the number of hidden inputs that gave the best performance was 128. The results of varying the number of hidden inputs in LSTM is shown in Fig. 3. When 128 hidden inputs were used in the LSTM layer, validation accuracy reached 83.3%.

The next hyperparameter we optimized was the number of nodes in the first fully connected layer. We tested different values of this parameter in the range of 16 to 128 and found the optimal number to be 128. The results of the different values obtained with the different numbers of hidden nodes is shown in Fig. 4. Using this, we were able to optimize our model further to obtain around 88% validation accuracy.

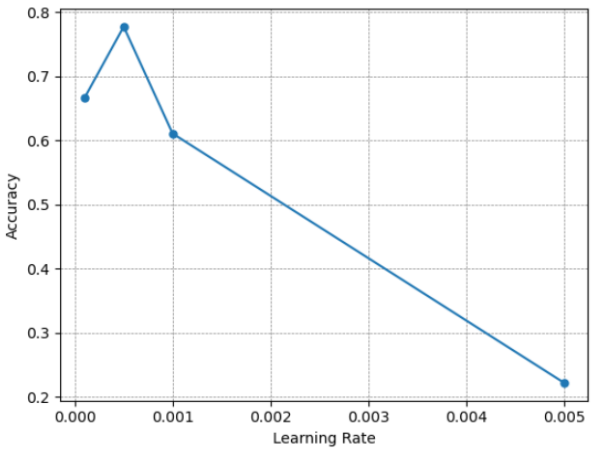

Fig. 2. Validation accuracy with different number of learning rates

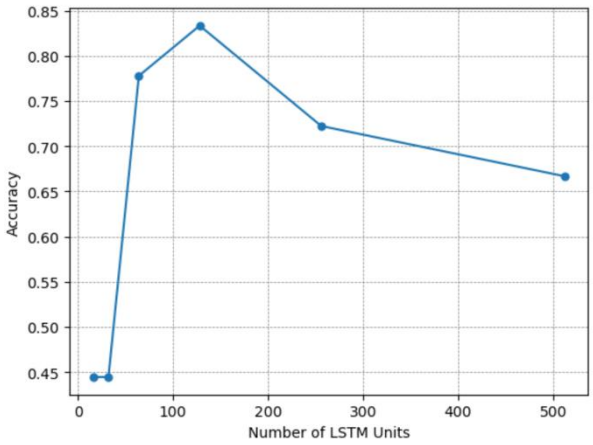

Fig. 3. Validation accuracy with different number of LSTM hidden input units.

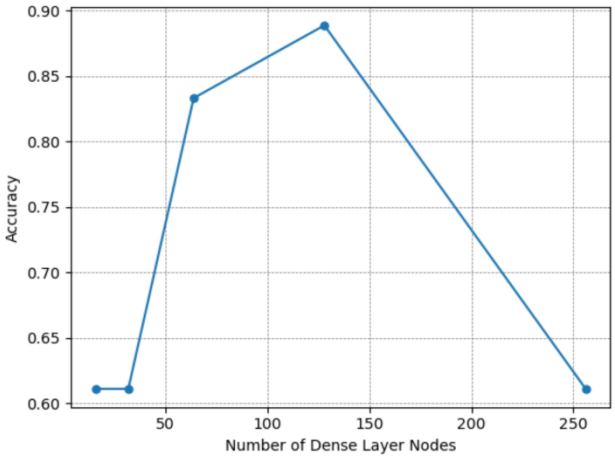

Fig. 4. Validation accuracy with different number of Nodes in the first Dense Layer.

B. *Algorithm Performance*

This section describes how our models performed.

After finding the optimal values for our models, we used the testing set to evaluate the performance of our models. The testing set included joint sequence data for

18 videos that the model had not seen in training. The testing accuracy of our different models is shown in Table 1. Our approach with LSTM2 achieved a rate of word recognition of 83.3% on the testing set.

We then evaluated our LSTM1 and RNN models on the same testing set.

The LSTM and Simple RNN both give similar results for this problem, this is because LSTM mostly performs better in cases where long sequences of data are used and differences across time are important. In this scenario the units of data are single words rather than complete sentences. For this reason both models are suitable, and little difference is seen between the two. However there is a slightly higher performance in the LSTM model, as it does still have a greater ability to find the relation between each part of the data.

The validation loss over the course of training the LSTM model consistently decreases after each epoch during the training process. During testing it was observed that at a certain point the validation loss between epochs becomes stagnant as peak performance is reached. Rather than a smooth decrease, a jagged pattern over the course of training individual epochs can be seen. The cause of this is simply the volatility of the training process. Individual epochs can have either a positive or negative effect. eventually a convergence is reached, tending towards a highly accurate model.

C. *Comparative Analysis Results*

This section describes how our models performed in comparison to each other. Fig. 5. shows testing accuracy obtained by the RNN model and the 2 LSTM models with 1 and 2 Fully Connected layers respectively. As we can observe from the results, LSTM2 and the RNN model were equally good at recognizing the gestures. We would have expected LSTM to be superior at recognizing time series data, but this was not the case. However, LSTM2 performed better than LSTM1.

TABLE I. TABLE TYPE STYLES

Method	**Models and Testing Accuracy**	
	Description	***Testing Accuracy(%)***
LSTM2 T	LSTM with two fully connected layers (Tuned)	**89%**
LSTM2	LSTM with two fully connected layers (No Tuning)	83.3%
LSTM1	LSTM with one fully connected layer	78.8%
RNN	Simple RNN model	83.3%

Values in bold indicate the highest validation accuracy.

Fig. 5. The accuracy obtained using different models in recognizing 10 words of the WLASL Videos dataset.

V. DISCUSSION

A. *Findings*

This section describes the findings of our project. We first trained a Simple RNN network without optimizing the hyperparameters. The Simple RNN network was able to achieve 83.3% validation accuracy in only about 80 training epochs as shown on Fig. 6. This model performed well with default parameters and served as a baseline to compare our other models to.

After training the RNN model, we trained an LSTM model with one Fully Connected layer. As shown in Fig. 7., this model was only able to achieve a maximum of around 78% validation accuracy and it took more epochs than RNN to reach that point.

Finally, we trained an LSTM model with 2 Fully Connected Layers instead of one. This model achieved similar results to RNN but it took more epochs to reach the same validation accuracy, as we see in Fig. 8. The LSTM with 2 Fully Connected layers achieved better performance than the model with a single Dense Layer. This shows that an end to end model is more stable [1].

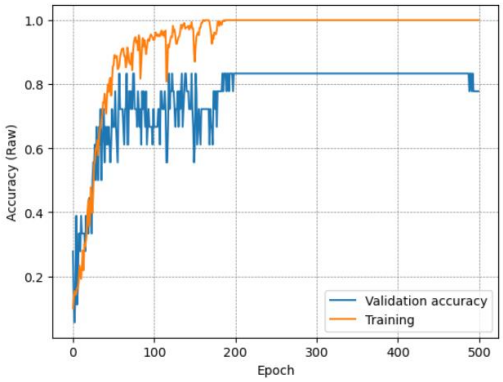

Fig. 6. Training and validation accuracy for RNN model.

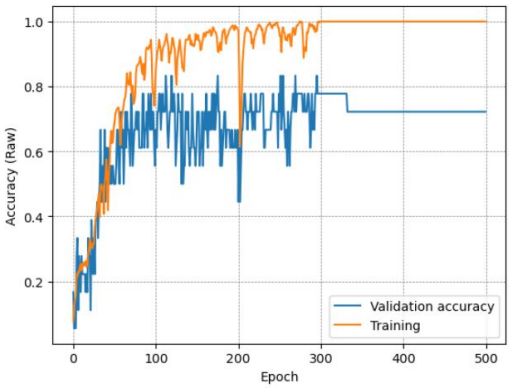

Fig. 7. Training and validation accuracy for LSTM1 model.

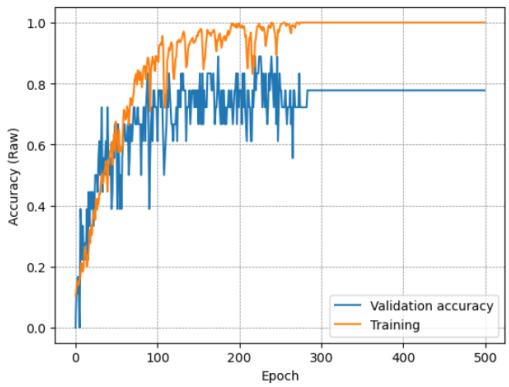

Fig. 8. Training and validation accuracy for LSTM2 model.

B. *Interpretation of the results*

This section describes the interpretation of the results we obtained. The aim of our project was to be able to accurately recognize the 10 words we selected from the WLASL Videos dataset using feature extraction from MediaPipe Holistic which provided joint tracking data. We trained our LSTM2 model, composed of 2 Fully Connected layers and obtained the highest results of 89% accuracy.

Our RNN model was able to achieve a lower accuracy of 83.3%. These results were achieved by using 15 frames from each video. Data augmentation greatly improved results as 3 frames were extracted from each single frame of video data. The reason why RNN was able to achieve similar performance to LSTM despite not performing hyperparameter tuning on RNN may be because the input sequences were not long enough for long term dependencies to take place. Research has shown before that deeper networks do not guarantee better performance [11].

C. *Pros and Cons of our solution*

This section discusses the benefits and drawbacks of the solution we implemented. A major benefit of our solution is that it is able to extract joint sequence information without the need for extraneous equipment. Other methods have used systems like the Kinect, but our model is more accessible because it can be trained on pre-existing videos as well as videos from regular mobile or webcam devices [1]. In addition, this makes the model suitable for real-time deployment, which is one of the objectives we had set to accomplish with this project.

It is common for researchers to use Convolutional Neural Networks for feature extraction in the task of sign language word recognition [5]. Another benefit of our approach is that it focuses on the most salient visual features needed for SLR. CNNs may learn to identify irrelevant features such as background information or color pixel values, whereas our model is trained on the pure positional joint values of each gesture.

A drawback of our solution is the lack of video data to help in training the model. Due to the limited availability of quality data for sign language interpretation, we had to augment the videos by applying horizontal transforms and random rotations. This is not ideal because certain sign language gestures are asymmetric, meaning that they are orientation-dependent. In those cases, the model will have difficulty identifying between the correct orientation. Moreover, the lack of videos will make it more difficult for the model to generalize to individual variances in gesturing style, such as the tempo different signers may deliver a gesture with. It is crucial to obtain more labeled sign language data to improve the model's performance.

VI. Conclusion

This section contains our concluding thoughts on the project. In this paper, we have proposed 2 models for converting videos of words in the ASL vocabulary into text. After training our models, we have shown promising experimental results in testing on unseen videos. We also realized that large datasets containing a greater number of videos per word to be recognized are crucial and can determine the performance and capacity of the model to generalize. For future work, we want to expand our model to be able to recognize more words of the ASL, train our model on a greater number of training videos to improve performance and we also want to work towards deploying real-time use of the model.

References

[1] T. Liu, W. Zhou and H. Li, "Sign language recognition with long short-term memory," 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, USA, 2016, pp. 2871-2875, doi: 10.1109/ICIP.2016.7532884.

[2] Li, D., Rodriguez, C., Yu, X., & Li, H. (2020). Word-level deep sign language recognition from video: A new large-scale dataset and methods comparison. In Proceedings of the IEEE/CVF winter conference on applications of computer vision (pp. 1459-1469).

[3] C.K.M. Lee, Kam K.H. Ng, Chun-Hsien Chen, H.C.W. Lau, S.Y. Chung, Tiffany Tsoi, American sign language recognition and training method with recurrent neural network, Expert Systems with Applications, Volume 167, 2021, 114403, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2020.114403.

[4] Su Yang and Qing Zhu "Continuous Chinese sign language recognition with CNN-LSTM", Proc. SPIE 10420, Ninth International Conference on Digital Image Processing (ICDIP 2017), 104200F (21 July 2017); https://doi.org/10.1117/12.2281671

[5] Elhagry, A., & Elrayes, R. G. (2021). Egyptian sign language recognition using CNN and LSTM. *arXiv preprint arXiv:2107.13647*.

[6] Basnin, N., Nahar, L., & Hossain, M. S. (2020, December). An integrated CNN-LSTM model for Bangla lexical sign language recognition. In *Proceedings of International Conference on Trends in Computational and Cognitive Engineering: Proceedings of TCCE 2020* (pp. 695-707). Singapore: Springer Singapore.

[7] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., ... & Grundmann, M. (2019). Mediapipe: A framework for building perception pipelines. *arXiv preprint arXiv:1906.08172*.

[8] N. Mohajerin and S. L. Waslander, "State initialization for recurrent neural network modeling of time-series data," 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 2017, pp. 2330-2337, doi: 10.1109/IJCNN.2017.7966138.

[9] Pigou, L., Dieleman, S., Kindermans, P. J., & Schrauwen, B. (2015). Sign language recognition using convolutional neural networks. In *Computer Vision-ECCV 2014 Workshops: Zurich, Switzerland, September 6-7 and 12, 2014, Proceedings, Part I 13* (pp. 572-578). Springer International Publishing.

[10] Hernandez V, Suzuki T, Venture G (2020) Convolutional and recurrent neural network for human activity recognition: Application on American sign language. PLoS ONE 15(2): e0228869. https://doi.org/10.1371/journal.pone.0228869

[11] M. C. Ariesta, F. Wiryana, Suharjito and A. Zahra, "Sentence Level Indonesian Sign Language Recognition Using 3D Convolutional Neural Network and Bidirectional Recurrent Neural Network," 2018 Indonesian Association for Pattern Recognition International Conference (INAPR), Jakarta, Indonesia, 2018, pp. 16-22, doi: 10.1109/INAPR.2018.8627016.

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