American Sign Language Recognition Using Computer Vision

Nihar Joshi niharj@umich.edu
Ryan Gudal rgudal@umich.edu
Tianyu Jiang prajnaty@umich.edu
Samantha Clark clarksam@umich.edu

1. Introduction

American Sign Language (ASL) is the primary form of communication for 1 million individuals in the Deaf community, hard-of-hearing community, as well as family or friends of people in the Deaf community. However, for the remaining U.S. population, very few people know ASL. As a result, there is a significant communication barrier between many ASL signers and the rest of Americans. Communication with signers usually requires Deaf Interpreters, who are not always available and are in high demand – the number of interpreters needed is expected to increase by 19% over 10 years [11].

To facilitate communication across these communities and attempt to fulfill the need for this high demand, we aimed to build a model that can translate ASL to written American English using computer vision techniques. More specifically, given a video of individual words or phrases in sign language, we hope to classify the sign.

Computer vision is especially apt to solve this problem because ASL is an entirely visual language. With the assistance of modern computer vision techniques it is possible for models to be trained to interpret and understand temporal and spatial signals. In recent years, researchers have been experimenting with using deep learning to solve this classification problem. Existing approaches build off the work of LeCun [4] and Krizhevsky [3] to use convolutional neural networks (CNN) that can classify images. Current researchers build off of this work and use 3-D CNNs or convolutional recurrent neural networks (CRNN) to train classification and translation models for video sequences of ASL. Additionally, there are companies that attempt to use computer vision to translate ASL. SignAll, a company that translates videos of people signing in real-time, has been working on a video translation product since 2016 and have just recently launched a pilot program [9].

To develop our translation system, we built off the work of other researchers, such as Ko and Son [2], to make a CRNN. We implemented two CRNN models both starting with a ResNeXt-50 CNN [12]. In one model we added a Long Short-Term Memory (LSTM) cell and the other model had a standard recurrent neural network (RNN).

2. Approach

2.1. Data Preprocessing

In our investigation, we used the ChicagoFSWild ASL Fingerspelling Data Set [1] [7] [6]. The data contains a collection of over 10,000 words that are signed by a diverse group of signers, and split into a series of image frames. The number of frames per video sequence varies from 2 to 296 frames and the resolution of each image varied. To streamline the input data, we resized all the input images to 224 $\times$ 224 pixels and we eliminated any words that had videos with over 45 frames or less than 5 frames. We found that the words with larger frame counts were usually pronouns and appeared very infrequently, so we did not expect our model to train accurately on these. We then sub-sampled this new set of words to create a smaller data set. Our final data set consisted of a total of 1,129 words, with 42 unique ones. Among the total word count, the training set included 872 words, the validation set included 123 words, and the test set included 134 words.

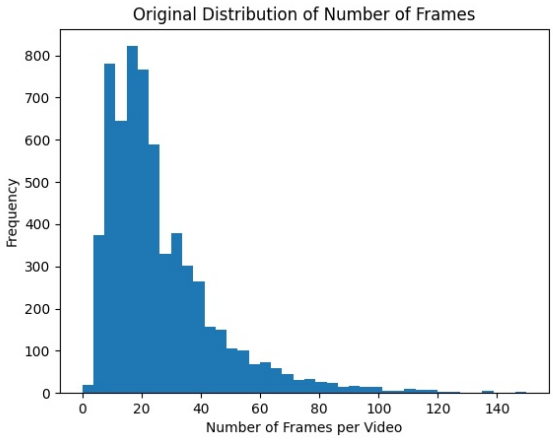

Figure 1: The original distribution of frames

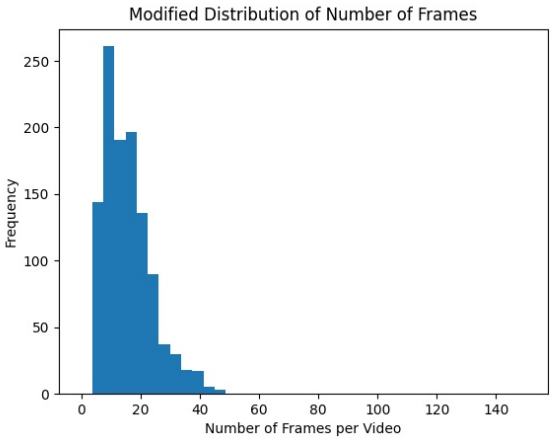

Figure 2: Frames after discarding long and short videos

2.2. Architecture

To translate the sign language videos, we used two separate architectures and compared them to a baseline weighted-random guessing algorithm. First, we passed every video sequence through a CNN to extract features from the images. We used a pre-trained version of ResNeXt-50 because of its high accuracy and the large size of the feature tensors it produced.

Next, we padded all of these feature tensor sequences to ensure that they had equal length. Video sequences that were shorter than the maximum length in a batch were padded with 0 tensors. This enabled our model to train with different batch sizes. The resulting tensors were cached and fed into our two separate architectures.

We built two separate models: an LSTM and a multi-layer RNN from PyTorch. Both of these RNNs used dropout layers and had fully-connected layers. Additionally, we fed the padded feature tensors to our models in batches of size 32. We ran each model for 20 epochs due to our computational restrictions.

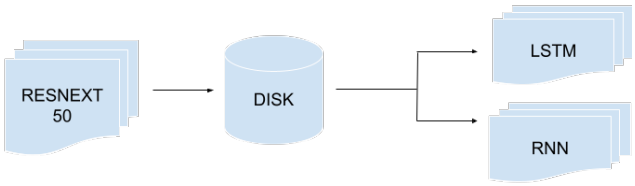

Figure 3: Neural Network Model Pipeline

To evaluate the model and perform the training step, we used a Cross Entropy Loss calculation and used PyTorch's Adam optimizer. We also used a grid searching technique to find a suitable range of values for weight decay, learning rate, dropout, and number of hidden layers. At each step, we chose the hyperparameters that maximized the accuracy of the validation data set.

This was the best hyperparameter configuration we found for the LSTM:

- Weight Decay = 0.0001
- Learning Rate = 0.005
- Dropout = 0.375
- Hidden Layers = 128

This was the best hyperparameter configuration we found for the RNN:

- Weight Decay = 0.0001
- Learning Rate = 0.001
- Dropout = 0.25
- Hidden Layers = 128

Overall, we evaluated each model based on the accuracy it achieved on the validation data set and picked the best epochs from this data to run on the final test data set.

3. Experiments

3.1. Data

We used data from the ChicagoFSWild ASL Fingerspelling Data Set, which was collected from online videos featuring several signers. These videos are captured in non-studio environments, with a variety of backgrounds and lighting conditions. This non-uniformity in the data makes it more realistic when compared to more controlled data sets such as the RWTH-BOSTON-50 data set or Purdue RVL-SLLL ASL Database.

3.2. Metrics

We primarily used Cross Entropy Loss and accuracy to evaluate our models. We decided to use Cross Entropy Loss since it combines the negative log-likelihood loss and softmax loss. The Cross Entropy Loss can be used in a multi-class classification setting, since it compares the predicted class probabilities with the true classes, and averages the loss across all examples in the test set.

We also decided to look at accuracy because we wanted to maximize the number of correct predictions in our test set. One drawback of using accuracy is that it can be difficult to get a high accuracy, since each prediction is either 0 or 1 — there is no probability in between. Additionally, the accuracy metric does not provided any weight to the correctness of each individual class.

3.3. Results

We ran a grid search on the hyperparameters of both models, and ran the training loop on the LSTM and RNN models to produce Figures 4 - 7.

As mentioned above, we found hyperparameters using a grid search over possible value ranges. Portions of Figure

5 and Figure 7 appear to have extreme oscillations in accuracy. These oscillations are expected, because updating the model slightly can cause the misclassification of a group of items at a time and can therefore reduce or increase the accuracy greatly. Slight increases in the loss can also be explained by the relatively high learning rate that we employed.

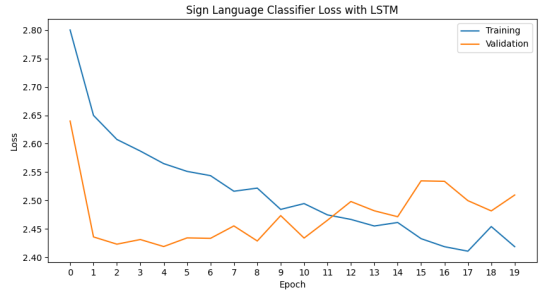

Figure 4: LSTM Loss

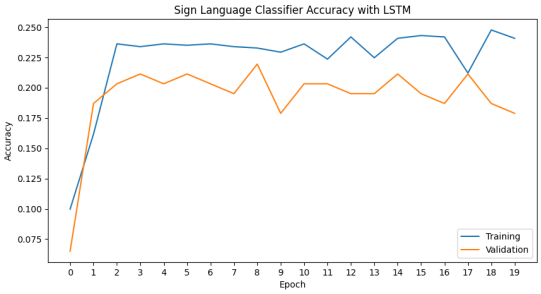

Figure 5: LSTM Accuracy

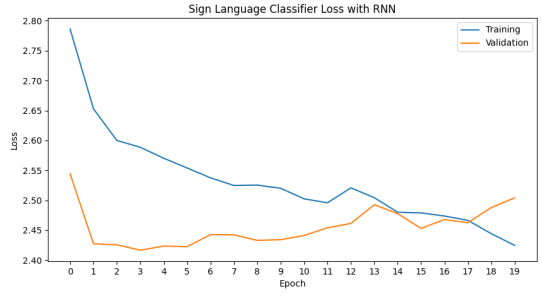

Figure 6: RNN Loss

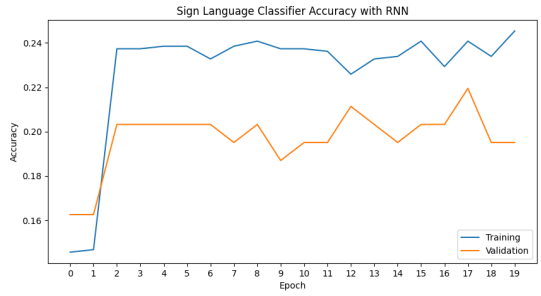

Figure 7: RNN Accuracy

Table 1. Accuracies of the 2 models

Model	Train (%)	Val (%)	Test (%)
LSTM	23.27	22.95	33.58
RNN	24.08	21.95	26.11

In general, we evaluated every single model based on the accuracies. We selected the epochs within each model that maximized the validation accuracy and reran the models with the test data set to produce a final accuracy result. Table 1 shows the complete result for accuracy scores and suggests that the LSTM had a superior performance compared to the RNN.

To evaluate the results of our different deep networks, we created two baseline algorithms: a weighted-random guessing algorithm and a modal guessing algorithm. These algorithms were evaluated on their accuracy and the results were directly compared to the performance of our deep networks.

The weighted random guessing algorithm used the proportion of word occurrences in the training set to generate random guesses for every label in the test data set. We ran this algorithm 1,000 times to create a distribution of accuracies that this model achieved. On average, the model achieved an accuracy of 11.95% on the test set.

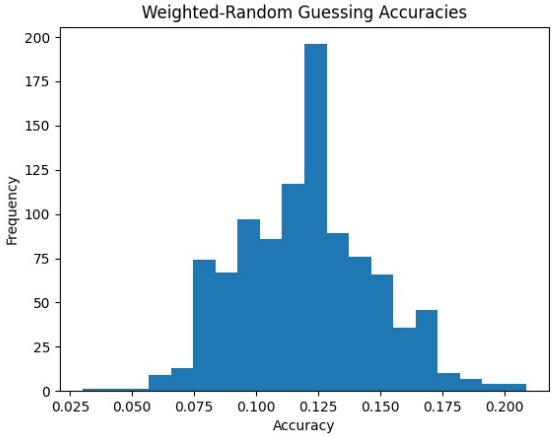

Figure 8: Accuracies based on weighted guess

Our modal guessing algorithm simply guessed the most common word that appeared in the training data set. This algorithm achieved a high accuracy of 26.86% on the test data set. The data set, however, had a very prevalent word - “asl”, which does not reflect the true occurrence of words in ASL. Therefore, this high accuracy may be misleading and we should consider other metrics when evaluating our model holistically.

3.4. Discussion

We analyzed our results by comparing the baseline metrics to the values that we found in Table 2. Overall, we noted that the random guessing algorithm performed very poorly compared to all other models. On the other hand, the modal guessing baseline metric produced a relatively high accuracy score.

Table 2. Accuracies of the models vs. baselines

	LSTM	RNN	Modal	Random
Test Accuracy	33.58	26.11	26.86	11.59

We believe that the strong performance of the modal guessing algorithm was due to the imbalance in the training and test data. When we examined the word distributions within each partition, we found that the training set contained 42 unique words, while the test set only contained a subset that included 13 of those words. In addition, the frequency of different words was not balanced.

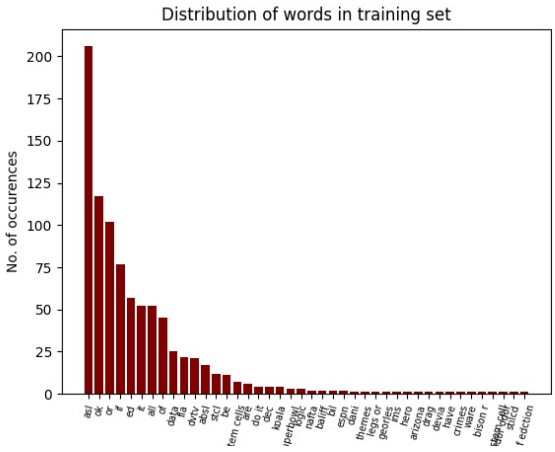

Figure 9: Word counts in training set

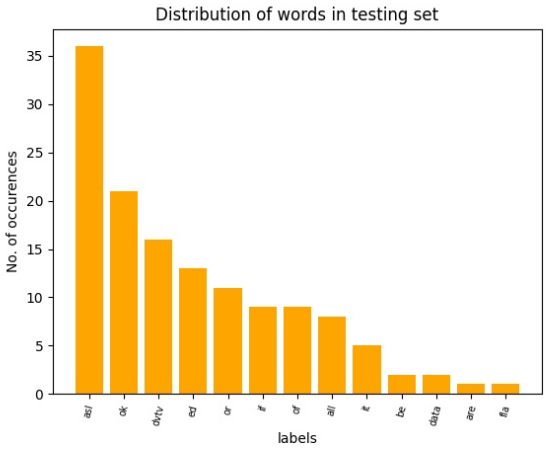

Figure 10: Word counts in test set

We believe that with a more balanced data set, the modal guessing algorithm's accuracy would drastically decrease and our two deep networks would perform better. It is important to note that the ChicagoFSWild data comes pre-partitioned by the creators. Therefore, the splits and distribution of words in our training and test set was predetermined and not altered by our own data preprocessing.

In general, an LSTM is differentiated from an RNN by the forget gates and output gates that keep track of long term memory for the model. LSTM models are thought to perform better on temporal sequences that are contingent on early data points in the sequence [8]. Although Table 2 suggests that the LSTM model vastly outperformed the RNN and our baseline models, we believe this is highly dependent on the test set that is chosen. When we ran our models on different sub-samples of the main data set, we found drastic changes in performance. Additionally, during our grid search for hyperparameters, we noticed large changes in the model's performance with small changes to the set of chosen parameters. In general, we don't believe that Table 2 is a completely accurate reflection of the LSTM model's performance. While we generally believe that the LSTM model performed better than the RNN, we believe that the margin of improvement from the RNN to the LSTM is much smaller than the Table suggests.

Overall, Table 2 indicates the the deep networks produced little to no improvement over the simple modal baseline. However, we believe it would be interesting to apply these same models and architecture to a different ASL Fingerspelling data set. Although ChicagoFSWild contains a robust variety of words, a more controlled dataset such as RWTH-BOSTON-50 [13] may give more insight into the performance difference between an RNN and LSTM in this classification task. In addition, we believe that our evaluation with the accuracy metric did not properly reflect the performance of each model. Using more complex metrics such as AUROC or Precision-Recall, may have allowed for more in-depth and nuanced analysis.

4. Implementation

We used PyTorch [5], an open source deep learning framework maintained by Facebook. In addition, we referenced the DataLoader functions from the EECS 445 (Machine Learning) course code. Finally, we referenced code from suzyahyah [10] to assist with padding variable sequences in the dataloader. The rest of the code, including the plotting, processing, and model functions, is our own work.

References

[1] J. K. J. M. D. B. G. S. B. Shi, A. Martinez Del Rio and K. Livescu. American sign language fingerspelling recognition in the wild. *SLT*, December 2018. 1

[2] S.-K. Ko, J. G. Son, and H. Jung. Sign language recognition with recurrent neural network using human keypoint detection. In *Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems*, RACS '18, page 326–328, New York, NY, USA, 2018. Association for Computing Machinery. 1

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. 1

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 12 1989. 1

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4

[6] B. Shi, A. M. D. Rio, J. Keane, D. Brentari, G. Shakhnarovich, and K. Livescu. Fingerspelling recognition in the wild with iterative visual attention. *CoRR*, abs/1908.10546, 2019. 1

[7] B. Shi, A. M. D. Rio, J. Keane, J. Michaux, D. Brentari, G. Shakhnarovich, and K. Livescu. American sign language fingerspelling recognition in the wild. *CoRR*, abs/1810.11438, 2018. 1

[8] S. Siami-Namini, N. Tavakoli, and A. S. Namin. The performance of lstm and bilstm in forecasting time series. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3285–3292, 2019. 4

[9] SignAll, 2021. 1

[10] Suzyahyah. Pad pack sequences for pytorch batch processing with dataloader, 2021. 4

[11] G. Vasquez. Demand for deaf interpreters poses opportunities to learn asl. 2021. 1

[12] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. 1

[13] M. Zahedi, D. Keysers, T. Deselaers, and H. Ney. Combination of tangent distance and an image distortion model for appearance-based sign language recognition. In W. G. Kropatsch, R. Sablatnig, and A. Hanbury, editors, *Pattern Recognition*, pages 401–408, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 4