A Deep Learning Approach to ASL-English Translation

Sunny Shah
smshah94@stanford.edu
Computational/Mathematical Engineering

Teresa Noyola
tnoyola@stanford.edu
Computer Science

1 Background

Sign language is developed as a language to communicate one's thoughts and emotions through gestures, facial expressions, and other visual cues. Being a non-verbal means of communication, sign language can be a difficult language for non-signers to understand, mandating the need for sign language interpreters, especially during events of importance, such as medical or legal appointments, or even in class. With the improvement in technology, there has been an increase in demand for such interpretation services and other accessibility services, such as closed-captioning, by those who require it. Therefore, there arises the question of developing a model that accurately translates sign language to a language of our choice.

In this paper, we propose several different models that endeavour to accurately translate American Sign Language (ASL) to English. More specifically, we discuss variations of a three-dimensional convolutional neural network (3D-CNN) and a convolutional-recurrent neural network (CRNN).

2 Related Works

3D-CNNs have been implemented to recognise sign language gestures in [1], [3], and other papers. In [1], a 3D CNN with three layers and 25 classes is used to classify Arabic Sign Language words. The model uses two 3D-Conv/Max Pool layers and a softmax layer. The model achieves 93% accuracy with kernels of size 5x5x8 (layer 1) and 5x5x4 (layer 2). In [3], 3D CNNs are used to classify hand gestures in order to aid the design of touchless interfaces in cars. The model consists of two sub-networks (one high resolution and one low resolution) where the final output is achieved by multiplying the class probabilities of the two outputs element-wise. Each subnetwork consists of four 3D-Conv/MaxPool layers, two fully-connected layers, and one softmax layer. The model achieves 77.5% accuracy on the VIVA challenge's dataset. [2] uses a 3D CNN with three 3D-Conv layers (the first two followed by subsampling) and two fully-connected layers. The model uses color, depth, and trajectory information as input. The model achieves 90.8% accuracy on a dataset built with Microsoft Kinect.

3 Data-set

We obtained our dataset from [4] that contains more than 3,300 ASL signs, comprising of a total of 9,800 tokens, all obtained from 1 to 6 signers of ASL. The videos were captured from four different angles, providing a side view, a close-up of the head region and half-speed high resolution front-view of each signer. We used the already-processed videos, due to memory constraints, wherein the videos were processed to produce video with high fidelity in the hand and face regions, due to the fact that these regions are widely-known for providing the most salient information in signs. A sample of the raw and processed frames are given below:

The videos were processed at 60 fps, while the size of the original frames were at 320×656 pixels but then cropped during the preprocessing stage to 250×300 pixels, so as to capture only the front view of the signer. The dimensions of the front view varies from signer to signer, hence, we used dimensions that worked for all the signers, which introduced some grey pixels in all the frames. We anticipate that the models used in this paper will learn that these grey pixels will contribute nothing to the labels, since they will be uniform across all the videos being fed into the models. We split the processed data, using a 80%-20% split. We then resized the images to 50×60 pixels, so that we wouldn't encounter any memory problems. Finally, we processed the labels by representing them as via their one-hot encoding; given the length of the vocabulary, we output a column vector consisting of 0's and 1's, indicating which word or phrase was being signed in the recipient video. We do not split up the phrase, since a single sign (in the video) corresponds to the phrase in English.

g

Figure 1: How the original frame (left) has been processed (right)

4 Methodology

We set up a 3D-CNN using the architecture outlined in the appendix (see Figure 2). The input to our network was a 3D tensor, with the first two dimensions representing the image height and width, and the third dimension representing the number of frames for a particular video. We experimented with both Xavier and Glorot normal initializations. We used 16 filters in the first 3D convolutional layer and 32 filters in the second 3D convolutional layer. Furthermore, we also decided to stick with a kernel size of (3, 3, 3) since our images were of a big size, relative to most CNN models. We decided to increase the number of filters as we go deeper into the network (as per the industrial trend), and we decided to stick with two 3D convolutional layers. There are two fully-connected layers after the convolutional layers. The first one has 250 hidden units and the second uses a softmax activation with 535 classes. We used categorical cross entropy loss.

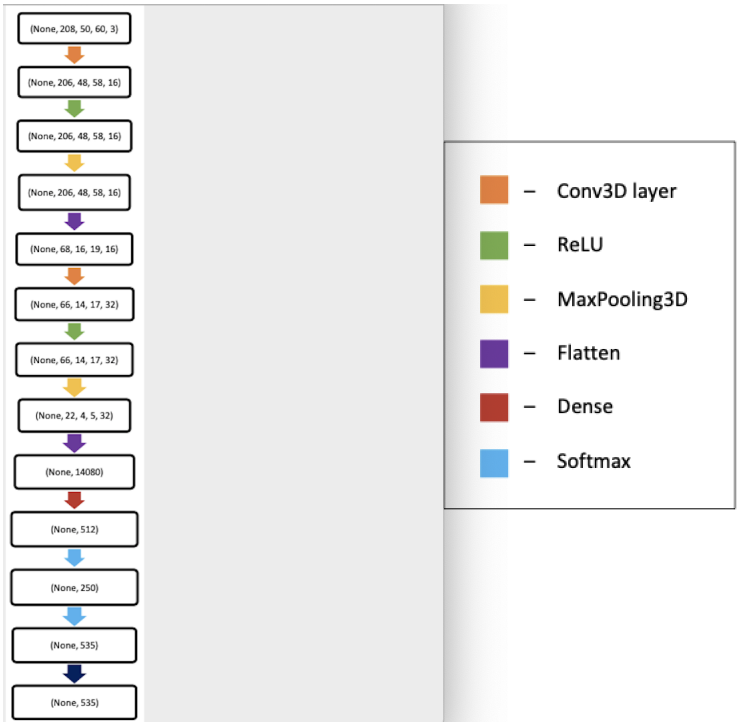
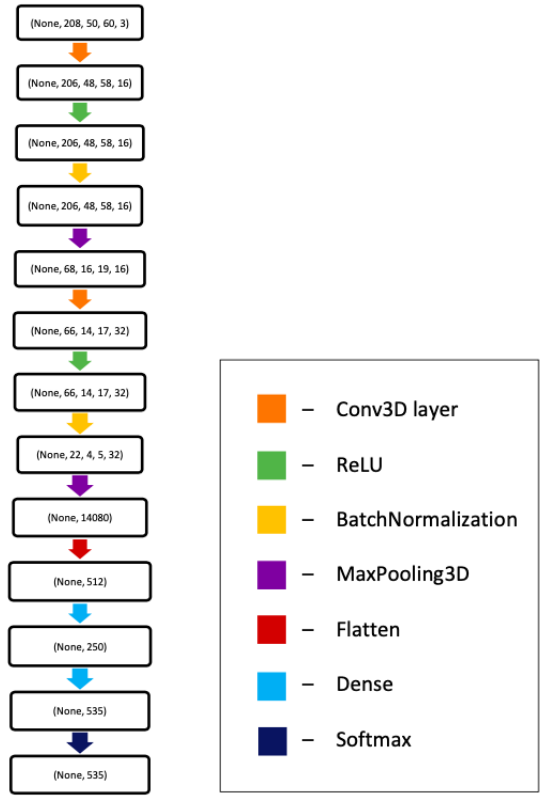


Figure 2: The architecture of the 3D-CNN

We also implemented a CRNN using the architecture outlined in the appendix (see Figure **??**). Each training example fed into our network was a 2D image representing a frame, so all the frame matrices for all the videos were concatenated together to create the input tensor. We used seven 2D convolutional layers (with 64, 128,

256, 256, 512, 512, 512 hidden units, respectively), all but one followed by a batch normalization layer and a ReLu activation layer. Four of the convolutional layers are followed by a max pooling layer. The output of the CNN is fed into the recurrent layers, which consist of two LSTMs follow by a dense + softmax layer. This architecture was chosen in the hopes of better representing temporal features of the data, since an RNN can propagate information from the past.



(a) Architecture **(b) Legend**

Figure 3: Caption

We experimented with L2 and L1 regularization, Dropout, and combinations of L2/L1 regularization and dropout (see Table 6 in appendix). We also experimented with wider and deeper networks and we tried both Adam and stochastic gradient descent optimizers. In order to find the optimal learning rate, we ran the models on five different learning rates (1.000e-7, 2.507e-5, 5.005e-5, 7.503e-5, and 1.000e-4). The results are shown in Table 7. We also re-ran the data preprocessing *without* shuffling the data to try and improve the validation accuracy. This provided more examples to train on for a single class, but also reduced the number of classes by a factor of 5. In total we ran seven experiments for each of the models, for each of the shuffled/non-shuffled datasets. We chose a batch size of 4 for the 3D CNN and a batch size of 1024 for the CRNN. We trained both models for 7 epochs. We used Adam optimization and used "accuracy" as our metric.

5 Discussion

The results of the seven experiments described above are shown in Tables 6 and 7. Our validation accuracy was extremely low for all of the experiments (discussed later), so we only report training accuracy. As shown by Table 6, the model that achieves the highest training accuracy is the 3D CNN with shuffled data and without regularization (plain network). However, we cannot conclude that this is the best model because it is highly likely that there is a problem with our data (discussed below). Figures 4 and 5 show the training accuracy and training loss for the model with the highest training accuracy. Figures 6 and 7 show the train accuracy and train loss for different learning rates (for the model with the highest training accuracy). As we can see, a learning rate of 1.000e-4 achieves the best performance.

Using Adam optimization performed better than stochastic gradient descent optimization, and creating a deeper or wider network (only attempted for the 3D CNN) did not improve performance. We also found that Xavier

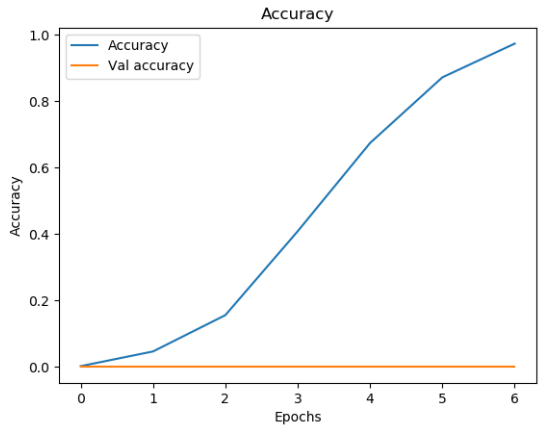


Figure 4: Train accuracy of best architecture

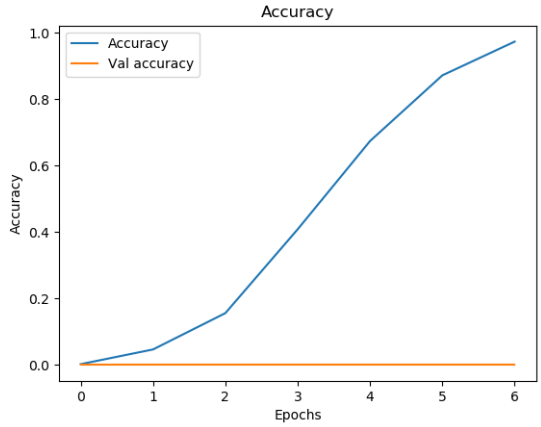


Figure 5: Train loss of best architecture

initialization worked better that Glorot normal initialization.

Our biggest concern with the results of our experiments is the extremely low validation accuracy (0% in many cases). This is either due to a problem the models or a problem with the data. However, since we got low validation accuracy for both the 3D CNN and CRNN models, and it did not improve with regularization and other attempted model improvements, we believe that the problem lies in the data. Specifically, the images in our data set have numbers at the top, grey spaces on the sides, and they are cropped unevenly within the data processing process.

6 Contributions

Sunny handled the data processing and led the implementation of the architectures. Teresa handled the training of both models for each of the seven experiments, for both the shuffled and non-shuffled data sets. Sunny created the video and Teresa and Sunny both wrote the report.

7 References

[1] Menna ElBadawy et al. “Arabic sign language recognition with 3D convolutional neural networks”. In: *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*. IEEE. 2017, pp. 66–71.

[2] Jie Huang et al. “Sign language recognition using 3d convolutional neural networks”. In: *2015 IEEE international conference on multimedia and expo (ICME)*. IEEE. 2015, pp. 1–6.

[3] Pavlo Molchanov et al. “Hand gesture recognition with 3D convolutional neural networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2015, pp. 1–7.

[4] Carol Neidle et al. "NEW Shared & Interconnected ASL Resources: SignStream® 3 Software; DAI 2 for Web Access to Linguistically Annotated Video Corpora; and a Sign Bank". In: *8th Workshop on the Representation and Processing of Sign Languages: Involving the Language Community, Miyazaki, Language Resources and Evaluation Conference 2018*. 2018.

8 Appendix

Our project is hosted on GitHub, and it can be found on the following link:

`https://github.com/sunnymshah95/ASL-English-translation`

Experiment 1 (plain network)		
	3D CNN	CRNN
Shuffled	79.79%	0.61%
Non-shuffled	44.58%	2.11%

Table 1: Training accuracy for Experiment 1

Experiment 2 (add L2 reg only)		
	3D CNN	CRNN
Shuffled	72.29%	0.23%
Non-shuffled	38.65%	0.74%

Table 2: Training accuracy for Experiment 2

Experiment 3 (add L1 reg only)		
	3D CNN	CRNN
Shuffled	61.25%	0.26%
Non-shuffled	45.81%	2.85%

Table 3: Training accuracy for Experiment 3

Experiment 4 (add dropout only)		
	3D CNN	CRNN
Shuffled	4.58%	0.08%
Non-shuffled	8.79%	1.5%

Table 4: Training accuracy for Experiment 4

Experiment 5 (add dropout + L2)		
	3D CNN	CRNN
Shuffled	5.21%	0.22%
Non-shuffled	2.66%	2.14%

Table 5: Training accuracy for Experiment 5

Experiment 6 (add dropout + L1)		
	3D CNN	CRNN
Shuffled	3.33%	0.59%
Non-shuffled	3.48%	2.85%

Table 6: Training accuracy for Experiment 6

Experiment 7 (vary learning rates)		
	3D CNN	CRNN
Shuffled	1.000e-4	2.507e-5
Non-shuffled	7.503e-5	2.507e-5

Table 7: Optimal learning rate, out of 5 tested

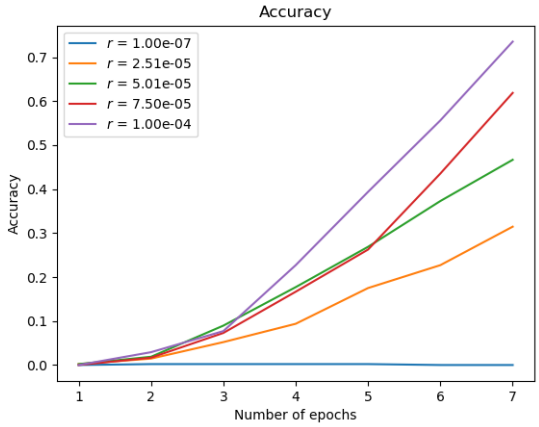


Figure 6: Train accuracy for various learning rates (plain network, best model)

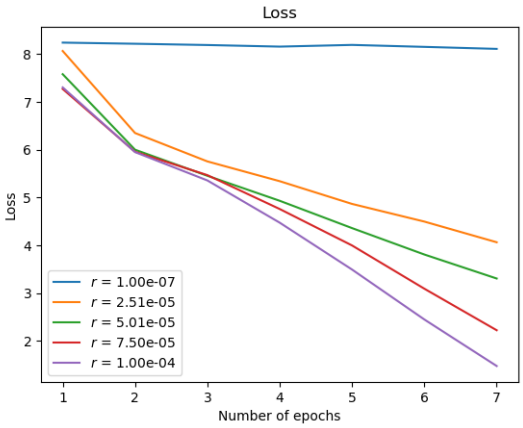


Figure 7: Train loss for various learning rates (plain network, best model)