ASL Complete Alphabet Recognition

Dima Al Dulaijan
Department of Computer Science
Boston University
Boston, MA 02215
dbld@bu.edu

Myles Cork
Department of Computer Engineering
Boston University
Boston, MA 02215
mcork@bu.edu

Faisal Mushayt
Department of Computer Science
Boston University
Boston, MA 02215
fmushayt@bu.edu

Abstract

An ASL alphabet recognition system that uses a convolutional neural network to recognize static gestures of the alphabet and computer vision to recognize dynamic gestures of the alphabet.

1 Introduction

American Sign Language (ASL) is often the primary language for deaf or non-verbal people. Without an interpreter, it becomes difficult for them to effectively communicate with others who are not fluent in sign language. A sign language interpreter application that could use common cameras built into modern computers and smartphones would help solve this problem. The main obstacle to creating an application like this is the classification of signs based on the input from the camera, which could vary in lighting, background, and other qualities. Deep learning models have had recent advances in classifying images, and thus may prove to be a good way to interpret ASL from images.

2 Related Work

With deep CNN's proving their strength in a wide variety of computer vision tasks, there have been prior attempts at using them to interpret ASL signs. Most focus on the static alphabet signs, leaving out the dynamic signs j and z. This is mainly because the static signs only require a single image to capture their meaning, while dynamic would require a video. Focusing on static images can help with the required space and time of training and predicting, and also allows for the use of models with pre-trained weights, such as those trained on ImageNet.

This approach was used by (Garcia, et al 2016) and well as (Masood, et al 2018). Garcia used a GoogLeNet model pre-trained on ImageNet, with the last few layers reinitialized with Xavier initialization and then retrained on an ASL alphabet dataset. To train their network they used log loss for their loss function. They did not mention what optimizer they used. They used a total of two datasets. The first was from the University of Surrey, and contained images of the 24 static alphabet signs from 5 different users, each with similar lighting and background, and of varying sizes. The dataset totaled about 65000 images. The second dataset, from Massey University, contained 2524 color images, each tightly cropped to the size of the hand and with backgrounds replaced with uniform black. They produced two models, one robust one for a-e, and another less robust for a-k and concluded that the training data was not varied enough to produce a robust model for all static alphabet signs. In their future work section, they mentioned more heavy image preprocessing, as well as trying other models, including VGG and ResNet. Masood built off of this, trying a VGG16 model pre-trained on ImageNet, again reinitializing and retraining the last few layers. They did not mention their loss function, but used SGD with momentum. Their dataset was the same one from Massey University that Garcia used. However, they augmented the 2524 images to create a much larger dataset of 14781, 75% of which were used for training and 25% for testing. This most likely explains their model's validation accuracy, which was much higher than similar papers. By augmenting their data and then splitting it between validation and training, there was duplicate information between the validation and training sets, leading to inflated results.

Another approach, building a custom deep CNN, was done by (Vivek, et al 2017). They used categorical cross entropy for their loss function, and did not mention what optimizer they used. Their dataset included 25 images from 5 different people, which they then removed backgrounds and augmented the images by flipping and rotating them to produce a larger dataset. They produced two models, one for letters and another for digits. They produced similar results to Garcia, concluding that environment and skin color were big obstacles, and that they would need more data in order to produce a more robust model.

3 Method

3.1 General Framework

Our method attempts to include analysis for both static and dynamic alphabet signs. A final application would start by classifying whether a sign is static or dynamic by detecting motion. We chose to focus on the following functions, those of differentiating between the static signs and the two dynamic signs. These tasks are the static and dynamic branches that can be seen in Figure 1. The static branch takes in a frame and then uses a ResNet50 CNN trained on static signs to decide between the static signs. The dynamic branch uses a motion energy based template matching classifier to determine whether the sign is j or z.

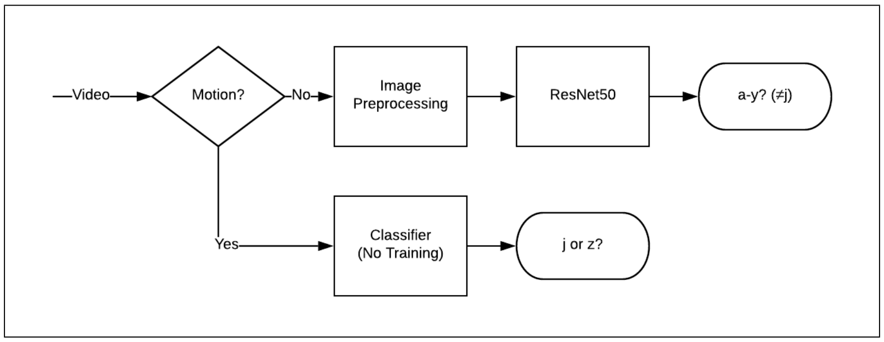


Figure 1: General framework of system.

3.2 Static Branch

For our models, we settled on using ResNet50, as in the past few years it has proven to be an effective deep CNN. This is in part due to its use of residual blocks, which can prevent gradient decay, a common problem with deep neural networks. We used categorical cross entropy for the loss function and Adam as the optimizer. We chose Adam over other optimizers as it converges much faster than others such as SGD, and has the advantages of both adagrad and RMSprop, which are avoiding local minimums and oscillations. Due to Adam?s convergence speed, we trained our model for 50 epochs, and found that our model converged around 25 to 30 epochs. We also modified regularization to try to reduce overfitting.

Our initial model was trained on a dataset containing 30 images for each sign. It included varied backgrounds, lighting, hand positions, and contained a large amount of edge cases. We split the data, 80% for training and 20% for testing. We further divided the training data into batches of 32 pictures. with and passed through the entire training set every epoch. Our next models were trained on a second dataset, containing 3000 images for each sign. Like the first dataset, it also contained images with varied backgrounds, lighting, and hand position, although it contains much less edge cases. This dataset also had images that varied in resolution and aspect ratio. Similarly to the previous dataset, the data was split 80% for training and 20% for testing, and then further divided into batches of 32 images. Again, all training data was passed through each epoch. For this dataset, we produced multiple models, one in which we preprocessed the input images to remove background and emphasize the hand, one where we did not preprocess the input images, and three others with various l2 regularization coefficients.

3.3 Dynamic Branch

Many ASL recognition systems tend to exclude J and Z, and that is simply because they are the only dynamic gestures of the alphabet. Classifying subtle motions in hand gestures is a problem greater than image classification. In order to fully encapsulate the motion, we resorted to computer vision.

First, we constructed a motion energy history image. This image is created by live frame differencing in order to subtract the background, and then captures several frames from the camera feed as the gesture is being signed. After segmenting the motion into several frames, we then combine all the captured frames into one motion energy history image. Here are some examples we created for J and Z. (Fig 2).

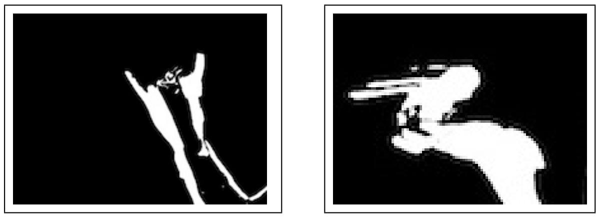

(a) J (b) Z

Figure 2: Motion energy history images

After that, we generated several templates for each letter, and used the template matching algorithm from OpenCV to detect similarities between our template and the motion energy history image generated from the user. Lastly, the gesture with the highest points of similarity to the input is then classified.

4 Results

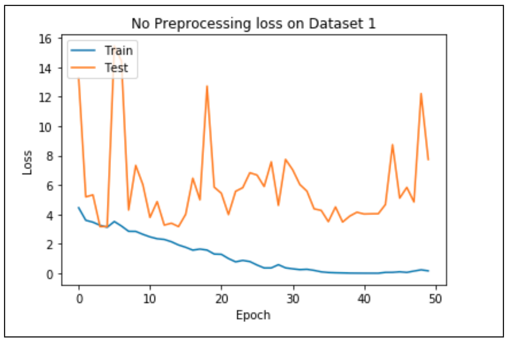
Our first attempts at classifying static signs mainly lead to bad results. The training and testing accuracies are displayed in Figure 3a, and similarly the loss in Figure 3b. We noticed that the in the training case, both metrics converged. For testing however, neither converged, and they both fluctuated a concerning amount. After some experimentation, we concluded the dataset was too small and full of edge cases. This confirms the conclusions of the other papers that the testing accuracy seemed to be low in general if the data is not representative of the features we are looking for.

Our tests on the second dataset seemed to lead to more consistent results. You can see the model accuracy and loss in Figures 4a and Figure 4b respectively. The testing accuracy converged around 0.7, and although the testing loss was fluctuating a good amount, it stayed around the same value in general. Training the model with preprocessing the images first did not lead to any improvement (Fig

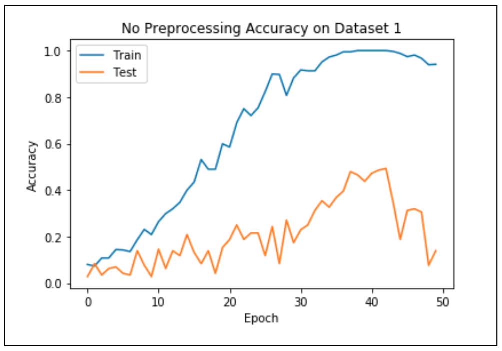
5) so we decided not to use preprocessing in the end.

One thing we noticed however was that there was a considerably big gap between the training and testing accuracy, which implies some kind of overfitting is taking place. So we figured out how to adjust the weight decay term in the ResNet model to improve regularization, but that did not lead to any favorable results. The model with the best weight decay value could only do as good as our untuned model. We believe the weight decay in the Keras ResNet model is already fine tuned to get the best results.

From experimentation, we could tell our dynamic sign classifier was accurate for the most part. It was difficult to quantitatively measure this accuracy however due to several issues. The first was the forearm matching problem. A significant number of pixels were matched on the forearm regardless of what the signer was signing, this lead to inconsistent results. We dealt with this by cropping the forearm out of the premade templates for Z. The second issue was the environmental conditions. It seemed the background, lighting, and distance from the camera all affected how the sign would be classified. This lead to some inconsistencies, but the classifier works well in general in well lit backdrops with minimal background movement.

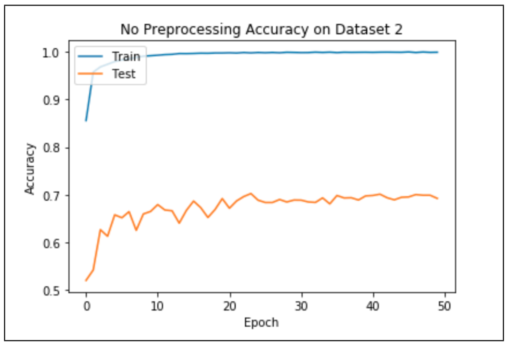


(a) No preprocessing loss

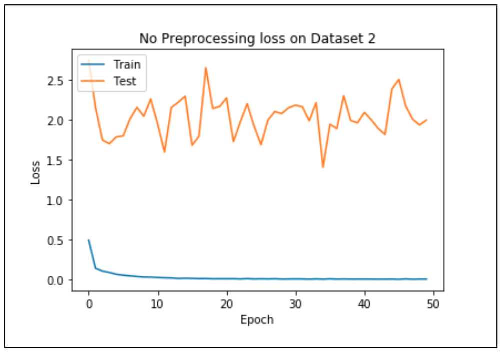


(b) No preprocessing accuracy

Figure 3: Dataset 1 results



(a) Accuracy



(b) Loss

Figure 4: Dataset 2 no preprocessing

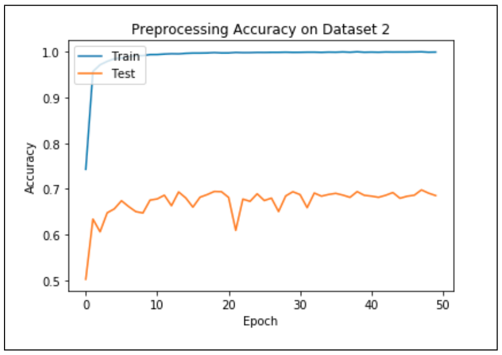


Figure 5: Dataset 2 with preprocessing

5 Conclusion

Our static methods did not provide an improvement over the other papers results. The best model, no-preprocessing on dataset 2, had a testing accuracy similar to Garcia and Bheda. Like them, we ran into similar issues, including unrepresentative datasets, as well as variations in background and other environmental conditions that contributed to low prediction accuracy. Our dynamic method is functional but we could not quantitatively calculate how accurate it is. We concluded that it was not robust as it requires certain testing circumstances to work, such as a static background and good lighting.

6 Future Improvements

Improvements need to be made to both the static branch and dynamic branch to make a robust interpreter. For the static branch, the CNN models accuracy could possibly be improved in a number of ways. A more representative and larger dataset could improve results. Fine tuning of the ResNet framework, or possibly creating a custom deep CNN may also prove to be more effective. From further research we found that motion signs could work better with using an HMM, which can take advantage of coarticulation. Motion tracking on top of pattern matching for J and Z could help as well. A complete implementation of an ASL interpreter would implement real-time sign recognition using motion detection, as mentioned in section 4.1. It would also be able to interpret words and sentences using a sequential language model for predicting letters and words.

References

[1] Garcia, B. & Viesca, S.A. (2016) Real-time American Sign Language Recognition with Convolutional Neural Networks, *in Convolutional Neural Networks for Visual Recognition*. Retrieved from Stanford University, June 21, 2019.

[2] Vivek, B. & Radpour, N. D.. (2017) Using Deep Convolutional Networks for Gesture Recognition in American Sign Language, (Report No.06836) *CoRR*, vol. abs/1710. 06836. Retrieved June 21, 2019.

[3] Masood, S. & Thuwal, H. & Srivastava, A. (2018). American Sign Language Character Recognition Using Convolution Neural Network. In: Satapa-thy S. Bhateja V. Das S. (eds.), *Smart Computing and Informatics. SmartInnovation, Systems and Technologies*, vol 78. Springer, Singapore. Retrieved June 21, 2019