# American Sign Language Predictions Using Convolutional Neural Networks and Other Models

JT Graass, Alex Wassel
Department of Computer Science, University of Virginia, Charlottesville, VA 22904
[jtg4de, aw7re]@virginia.edu

## Abstract

*Today, around one million people use American Sign Language (ASL) as their main form of communication. Used by the deaf and hard-of-hearing communities, as well as those with communication disorders, the language has received more resources and attention both online and in popular culture in recent years. However, 98% of deaf people do not receive education in sign language[1][1]. Generating image predictions has been a trending task in this field, such as predicting whether an animal is a dog or a cat. We took it a step further to recognize and label what letter of the alphabet a specific sign corresponds to. We focused on this work in predicting ASL signs, for which we obtained the dataset from Kaggle[2][2]. In our work, we explore the use of multiple machine-learning models. The input to these models are the matrix-representations of the images (signs) in the form of a numpy array. The purpose of this project was to gain further insight into what models work best for image prediction of hand signals.*

## 1. Introduction

Predicting image labels is a challenging supervised learning task for computers. In this project, we explored the implementation of a convolutional neural network (CNN), a Multi-Layer Perceptron (MLP) Classifier, a simple Neural Network model, a Random Forest Classifier, a Bagging Classifier, as well as a Voting Classifier. The reason that we tried to explore a variety of models was to gain insight into what aspects of machine learning work best for classifying images of hand signals.

Thus far, neural networks have demonstrated superior performance for this task, which is why we used them as a starting point. As MNIST datasets are a popular benchmark for image-based machine learning methods, we chose to use the Sign Language MNIST dataset to train our hand-image-recognition models.

We were inspired by one prior work that covers the implementation of a CNN for Image Recognition and Hand Gestures. While the performance of this CNN was already near perfect, we implemented our own models to how close we could get to replicating this pre-existing model. We show the results of our models through accuracy metrics, and we explain our reasoning on why we think certain models worked better than others.

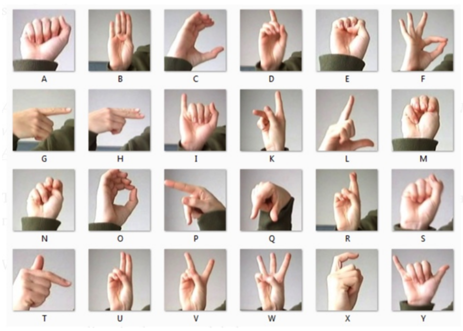


Figure 1. Here we show sample images from the Sign Language MNIST Dataset that we are using in our work. Our objective is to train various models to achieve the highest accuracies possible when predicting each of these ASL signs.

## 2. Related Work

Early work in this field include [3] where letters of the alphabet are predicted using ASL signs as input, using a convolutional neural network and max pooling. More recent works include the use of OpenCV as well as CNNs and keras [4, 5, 6]. These most recent works concentrate on recognizing the alphabet letters in real-time through a camera using signs made from one's hand.

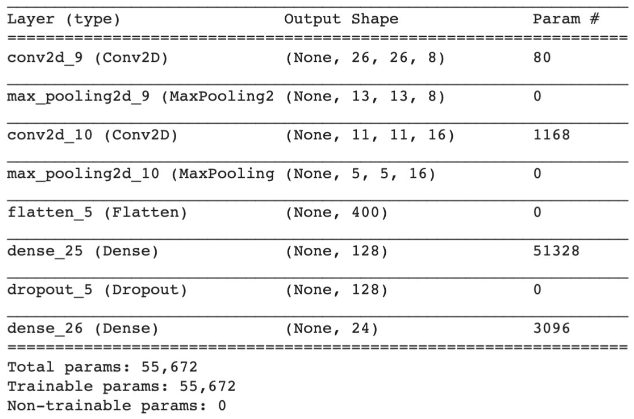
## 3. Model

While we were not able to achieve performance similar to the pre-existing CNN, we implemented some of our own

[1]https://www.newsweek.com/asl-day-2019-american-sign-language-1394695

[2]https://www.kaggle.com/datamunge/sign-language-mnist

models and achieved somewhat good accuracies. While we created and tested an MLP classifier, a simple NN, a Random Forest Classifier, a Bagging Classifier, as well as a Voting Classifier, we were far from perfect. But we do think that our implementations did reveal some insights regarding the task of classifying hand signals. For example, we have reason to think that the Random Forest Classifier is a bad model for this task since it is clear that there are certain aspects within the image of a hand signal that are more important than others. And since the Random Forest Classifier randomly splits the data rather than trying to split on the most important features, we think this classifier will fail to recognize constantly important features across hand images.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 26, 26, 8)         80
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 13, 13, 8)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 11, 11, 16)        1168
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_25 (Dense)             (None, 128)               51328
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_26 (Dense)             (None, 24)                3096
=================================================================
Total params: 55,672
Trainable params: 55,672
Non-trainable params: 0
```

Figure 2. This is the architecture of the CNN we used as our benchmark and goal.

## 4. Experiments and Results

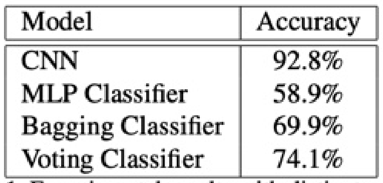
To implement our own models, we primarily used the sklearn library. We copied a high-accuracy CNN model that we found online, as mentioned in related work, and used this as our benchmark[3]. This model was able to achieve an accuracy of 92.9% on the test set. We then tried to get close to this accuracy with our own models.

The first model we implemented was an MLP Classifier that achieved an accuracy of 99.3% on the training set but only yielded an accuracy of 58.9% on the test set. We think that this is largely due to overfitting. We then implemented a simple neural network of our own that only consisted of one hidden layer as well as a "ReLu" activation function. This model failed to achieve any meaningful accuracy as the best accuracy it yielded was 5%, and we therefore abandoned it as a viable solution. We then implemented a Random Forest Classifier, and to our dismay it yielded a similar accuracy as the simple neural network.

[3]https://medium.com/the-research-nest/applied-machine-learning-part-2-a4ba715649d1

We then implemented a Bagging Classifier that yielded an accuracy of 69.9% on the test set. We think that classifying these images using bagging worked well for a number of reasons. Bagging, by definition, is the random selection of subsets from the training set and then using those subsets to train a model. This process is repeated many times. Then, the outputs of all these models are used to collectively predict the output of a given input image. One key characteristic of bagging is that the each time a random subset is selected from the training set, that subset gets replaced back into the training set. This ensures that the training set always contains every image. One trade-off of bagging is lower variance but higher bias. We think this characteristic played into this model's high accuracy.

Our final model was able to achieve even better results. We implemented a Voting Classifier using the sklearn library. Voting classifiers operate very similarly to bagging classifiers, but one key difference is that voting classifiers use multiple, distinct models to predict the outcome of one input. The way it does this is by inputting a single input through multiple models, which in our case included a Logistic Regression, a Random Forest Classifier, as well as an SVM. Then the output that the Voting Classifier predicts is the majority vote of each of the models. Our Voting Classifier achieved an accuracy of 74.1% on the test set. We think that a major reason why this voting classifier worked better was that it eliminated a lot of the variance, more so than the Bagging Classifier. And we think that minimizing variance matters especially in the context of classifying hand signals because there could be a number of outlier images in the data set that were being misclassified before.



| Model | Accuracy |
|---|---|
| CNN | 92.8% |
| MLP Classifier | 58.9% |
| Bagging Classifier | 69.9% |
| Voting Classifier | 74.1% |

Table 1. Experimental results with distinct models

## 5. Conclusion

We were very satisfied with our results. Even though were were not able to beat the pre-existing CNN model, we were able to construct a number of our own models with varying degrees of success. We were able to achieve the best success with the Voting Classifier. This project will be helpful and available for anyone in our community that would like to learn more about ASL signs using Computer Vision concepts. In the future, we are planning to further improve our project by getting more experience with OpenCV to make real-time sign predictions.

## References

[1] S. Aterfield. Asl day 2019: Everything you need to know about american sign language. In *Newsweek*, 2019.

[2] T. Boris. Sign language mnist. In *Kaggle*, 2017.

[3] X. Eric. Applied machine learning: Part 2. In *Medium*, 2018.

[4] A. Kazi. Sign language recognition using cnn and opencv. In *Medium*, 2019.

[5] M. Loïc. Slr alphabet recognizer. In *Github*, 2017.

[6] L. Marie. Sign language recognition: alphabet (python, opencv, tensorflow model inceptionv3). 2017.