# M&N's: An ASL Convolutional Neural Network

Lia Zheng, Mary Thomas

August 15, 2021

**Abstract**

A convolutional neural network (CNN) is an algorithm commonly used in machine learning for analyzing images. CNNs are meant to mimic the neurons in a human brain, and with the use of various filters, they are applicable to many tasks, including but not limited to: image processing, classification, and categorization. This project focuses on image classification. Using a dataset that contains thousands of images that depict different American Sign Language (ASL) letters, we attempt to build a CNN model with Python on Jupyter Notebooks that will be able to classify visually-similar ASL letters: M and N. We begin with a baseline model designed for dogs and cats, created by Jason Brownlee PhD on machinelearningmastery.com. From there, the original model will be adapted to run on high performance computing (HPC) systems, namely Expanse supercomputer from San Diego Supercomputer Center (SDSC). This paper will explain how the CNN is built, such as the necessary libraries, their functions, and the roles they play in building a working image classification model, as well as the challenges faced in developing the program.

# 1 Introduction

Convolutional neural networks are similar to human neurons in the way that the nodes are interconnected and restricted at the same time. An input image is processed with a CNN algorithm, which is able to pick out different parts of the image. This is done with convolutional filters—matrices where the number defines the desired output. For example, some coefficients are designed to sharpen the image, whereas some are designed for edge detection. After the convolutional layer is applied, pooling is performed in order to reduce the size of the image. Typically, max pooling (keeping the largest value) is done as it is the most effective method. As the model trains, it uses these algorithms to differentiate between the different classifications of the images [1].

# 2 Model Development

We started off with a model originally made for dogs and cats. The model was created by Jason Brownlee [2]. It used a large dataset from Kaggle containing 25,000 images of cats and dogs. That model was adapted to classify the ASL letters *M* and *N*. This model uses 6000 images in total, 3000 of each letter. The images were also from Kaggle, in a large dataset with all 26 ASL letters [3].

## 2.1 Model Design

The baseline model was chosen for multiple reasons: it's simplicity and clear instructions, the use of only two classes, and the various possibilities to experiment with the model. Because of its simplicity, the model can be adapted for a variety of uses. A more complex model would be difficult to adapt as there would be many niche components that would not translate well with other datasets. Additionally, as there are only two classes, the training model will yield more accurate classifications. The example model included one, two, and three VGG blocks—which stands for Visual Geometry Group, the researchers at Oxford who created the block architecture. These blocks contain convolution layers, as well as max pooling layers.

In ASL, the letters *M* and *N* are visually similar. The letter *M* is signed by holding a fist out with the thumb poking from in between the ring and pinkie finger. The letter *N* is the same, except the thumb is in between the middle and ring finger. Because these letters look so similar, we chose them to see how well the CNN model would classify them.

## 2.2 Building the Model

First, the image set had to be downloaded from Kaggle, then uploaded to the Lustre filesystem—SDSC *Expanse*'s global filesystem. *Expanse*'s filesystem al-

lows users to upload up to two million files [4]. The data needed to be accessed from inside JupyterLab. One challenge faced during this process was using the correct command for launching the notebook. We realized that the essential part of the galyleo command was:

```
--bind '/expanse,/scratch'
```

This allowed for the files from Lustre to be accessed inside JupyterLab.

Each image went through pre-processing, in which the size was modified to be 200 by 200 pixels for consistency. Then, all the images were saved into a NumPy array. NumPy is a Python library that is used for creating complex arrays and manipulating them. Then, the data had to be organized into folders: train and test. Each of those folders also had sub-folders labeled *M* and *N*. The training data is the data that the model will use to train. The test data is used to see how well the model performs on data that was not used in the training. In the case of this model, the test data made up 25% of the data. The library we used for this model was Keras, an open source library for creating deep learning models.

The first model was the one block VGG model. The first function *define_model()* used one convolutional layer and one max pooling layer. Because Keras's image data generator library had some deprecated functions, we added a function called *my_gen(gen)*, which is essentially an exception handler that tries to run the function *gen*, but will pass if it does not work. In the *run_test_harness()* function, the model is trained using the previous functions, and the test data will be used to compare the train accuracy and loss, and test accuracy and loss.

The next two models were the two block and three block VGG models. These added extra layers of convolution and pooling. As a result, the train accuracy went up, but the validation accuracy got further and further away from the train accuracy. This is known as overfitting—when the model captures the noise coming from the train dataset, therefore producing inaccurate results with the test data. In other words, if a model overfits, it has high train accuracy but low test accuracy. Brownlee's website gave two methods on combating overfitting: dropout regularization and image data augmentation. Dropout regularization means that during training, a percentage of the inputs will be randomly removed. Image data augmentation creates modified images of the data, in turn regularizing the model, as well as making the dataset larger.

# 3 Results

We first ran all the models using 24 epochs and 24 steps.

As seen in Table 1, this amount of training was not nearly enough to create an effective model. Thus, the model was trained again, this time with 30 epochs and 71 steps (see Table 2).

The differences were now less apparent, so graphs were included to visualize the training process (see Figures 1, 2, 3, 4, 5).

Dropout regularization did not perform well, and as a result, Image Data Augmentation did not either. Originally, the dropout percentages were 20, 20, 20, and 50. After changing the values to 10, 10, and 10, the model performed much better (see Figures 6, 7).

By using these regularization methods, the train and test graphs look much more similar, whereas without them, the train data performed with very high accuracy, while the test data had large fluctuations.

While these models can most likely be further optimized for better results, these graphs show that there is a fine line between a well-performing model versus one that overfits. Furthermore, combating overfitting can easily cause a loss in accuracy as well. Therefore, in order to create a robust model that is accurate with both the test and train data, a variety factors must be considered.

# 4 Acknowledgements

I would like to thank REHS for this wonderful opportunity to learn about and work with HPC systems. I have learned so much in such a short amount of time, and that would not have been possible without the REHS program. I would also like to thank SDSC as a whole, especially the teams who led the CIML and SDSC Summer Institutes. Those two weeks, one in the beginning of the summer, and one near the end, helped develop my understanding of the HPC systems, machine learning, and data science on HPC systems. Thank you to Leo Gu and Jenny Mar, fellow REHS interns, for their help and collaborations on our work this summer. Most of all, thank you to Dr. Mary Thomas for her mentorship and over these past eight weeks. Dr. Thomas spent many hours with the REHS interns each week, teaching and guiding us during this program. I am very grateful for all the support she has given us, and for the technical and interpersonal skills we have gained.

## References


[1] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53l, 2018.

[2] Jason Brownlee. How to Classify Photos of Dogs and Cats (with 97% accuracy. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/, September 2020.

[3] Akash Nagaraj. ASL Alphabet. https://www.kaggle.com/grassknoted/asl-alphabet, 2018.

[4] San Diego Supercomputer Center. Expanse User Guide. https://www.sdsc.edu/support/user_guides/expanse.html, 2020.


# Appendices

Table 1: 24 Epoch, 24 Step Training

| Model | Final Accuracy | Final Loss | Final Validation Accuracy | Final Validation Loss |
|---|---|---|---|---|
| One Block | 0.5807 | 0.6428 | 0.5771 | 0.6440 |
| Two Block | 0.8294 | 0.3794 | 0.7905 | 0.4052 |
| Three Block | 0.8516 | 0.3405 | 0.8165 | 0.3573 |
| Dropout Regularization | 0.5885 | 0.6335 | 0.6805 | 0.6456 |
| Dropout Regularization and Image Data Augmentation | 0.5501 | 0.6544 | 0.6544 | 0.6569 |

Table 2: 30 Epoch, 71 Step Training

| Model | Final Accuracy | Final Loss | Final Validation Accuracy | Final Validation Loss |
|---|---|---|---|---|
| One Block | 0.9909 | 0.0516 | 0.9807 | 0.0789 |
| Two Block | 0.9816 | 0.0638 | 0.9833 | 0.0613 |
| Three Block | 0.9978 | 0.0179 | 0.9947 | 0.0217 |
| Dropout Regularization | 0.6896 | 0.5272 | 0.7692 | 0.4973 |
| Dropout Regularization and Image Data Augmentation | 0.6032 | 0.5878 | 0.5717 | 0.6282 |

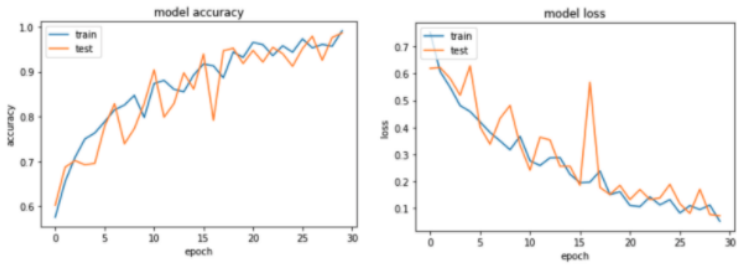


Figure 1: One Block VGG Model

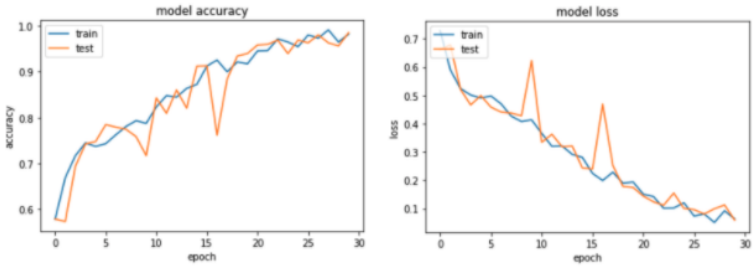


Figure 2: Two Block VGG Model

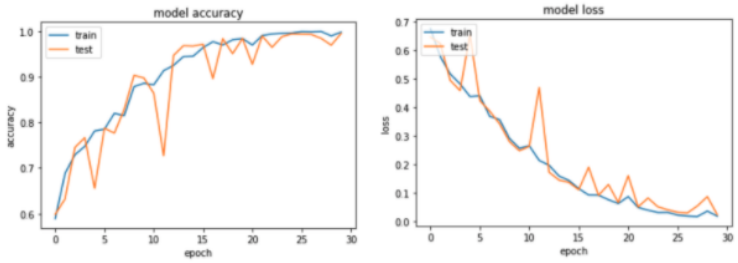


Figure 3: Three Block VGG Model

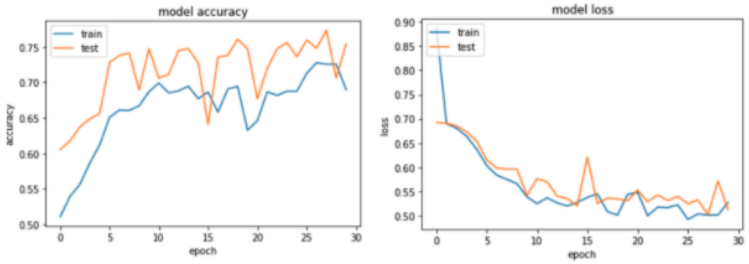


Figure 4: Dropout Regularization

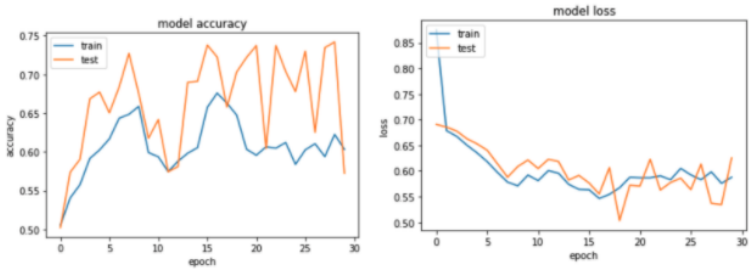


Figure 5: Image Data Augmentation

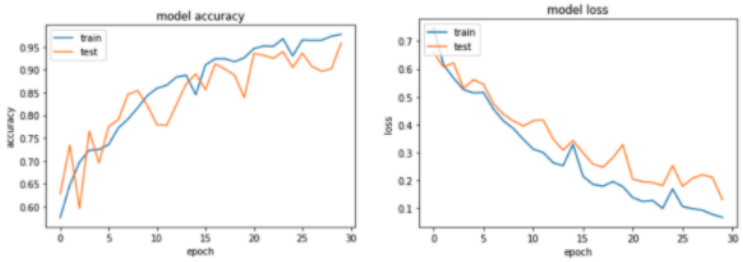


Figure 6: Revised Dropout Regularization

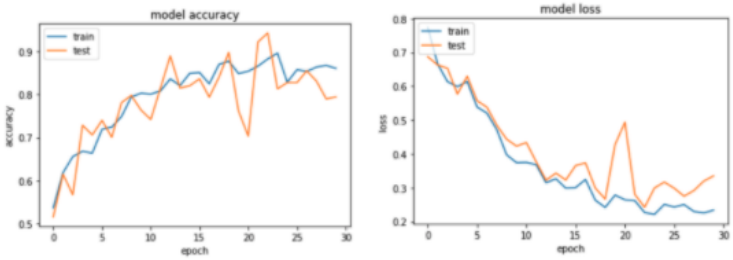


Figure 7: Revised Image Data Augmentation