Image Classification for American Sign Language

Joshua Meppiel

University of Missouri St. Louis

Abstract

This is the final report for the image classification project in CMP SCI 5390 - Deep Learning in Spring of 2023. Deep learning is a sub-field of machine learning in which artificial neural networks are used to create representations of data. These models are used to make predictions, also known as inferences, against data. They can be used for a variety of ever expanding tasks such as computer vision, natural language processing, healthcare, and financial predictive analysis. This project will use deep learning to perform the common computer vision task of image classification. Image classification takes images as inputs and infers classes from the data. In this project, I've chosen to classify images of the American Sign Language alphabet.

Contents

1 Introduction

In this project, I will explore a variety of methods to build and train neural networks using the tools TensorFlow and Keras. I will utilized the American Sign Language data set descibed in Section 2 below. The goal of the project is to learn the tooling and concepts behind deep learning and artificial neural networks.

This project is a culmination of taking the CMP SCI Deep Learning course in Spring 2023, attending Dr. Adhikari's Deep Learning Workshop at the end of 2022, and reading through the textbook Deep Learning with Python[1]

1.1 Report Layout

The remainder of this report will be structured as follows:

- Section 2: Data and its preprocessing and augmentation
- Section 3: Building an over fitting model
- Section 4: Data Augmentation and its effects
- Section 5: Data Regularization and its effects
- Section 6: Architecture priors
- Section 7: Results
- Conclusion

2 Data

2.1 Data Set

The data set used for this project is the American Sign Language data set available on Kaggle[2]. This data set contains 87,000 images of hands displaying the alphabetical characters of the American Sign Language, along with the special characters of "space", "nothing", and "delete". Each image is a 200x200 pixel RGB image in JPEG compressed format. The data set is broken down into 3 categories for training, validation, and testing. The training set contains 69,600 images. The validation set contains 17,400 images. The test set contains only 29 images, one for each class. The publishers of the data set encourage users to acquire more real world images for testing in addition to the 29 provided.

2.2 Augmentation

In addition to the 87,000 images provided within the data set, multiple methods of performing data augmentation were attempted. Available methods in Keras include random rotations, translations, flips, contrast changes and zooming. Where these augmentations are used will be described in the sections below.

3 Building an Over Fitting Model

For this task I will intentionally build an over fitting model. This exercise is to learn the process by which a model becomes over fitted to a data set. It allows us to learn the smallest possible model in terms of parameters, or weights, that will allow us to solve the problem at hand. Learning this first will allow us to expand the model to handle the problem in a real world scenario.

3.1 Validate on Training Data

First I utilize the training data both for training, but also for validation of the model during training. This process will cause the model to easily over fit, or memorize, the training data.

The smallest possible model that I was able to build to perform this task had 9,997 trainable parameters. It utilized 6 convolutional layers, each with a max pooling layer afterwards, before passing to the output layer. This model was able to achieve an accuracy of 90% after 25 epochs, **Figure 1**. The loss training and validation curves were similarly fit **Figure 2**.

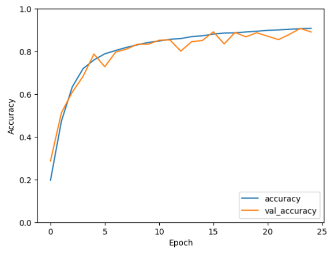

Figure 1: Accuracy per epoch.

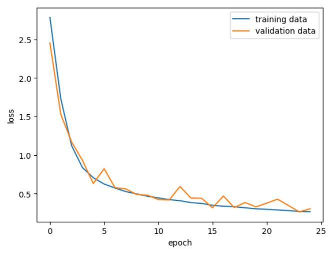

Figure 2: Loss per epoch.

3.2 Ground Truth as Input

For this task, graduate students were asked to provide ground truths as inputs to the model. This should prove that the model can fit the data to 100% accuracy. It should also reach 100% quickly and overfit the data. For this task, I used the same network architecture as the previous task, but also provided the ground truths, also known as input labels, as inputs to the model. The accuracy reached 100%, **Figure** 3, accuracy after only 92 images were input into the model.

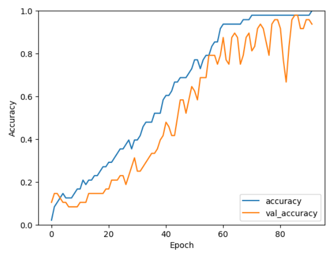

Figure 3: Accuracy per epoch.

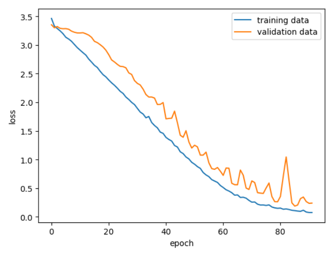

Figure 4: Loss per epoch.

4 Data Augmentation and its Effects

For this task, data augmentation was added to attempt to reach higher accuracy levels. I actually found that quite the opposite was true; adding data augmentation reduced accuracy significantly. Either a mistake was made in this process or this data set is not conducive to the data augmentation methods of random rotations and random translations that I tried.

4.1 Random Rotations

Here, the Keras Random Rotation method was utilized. The value of 10% was used for the random rotations. This reduced accuracy to about 70%, **Figure** 5.

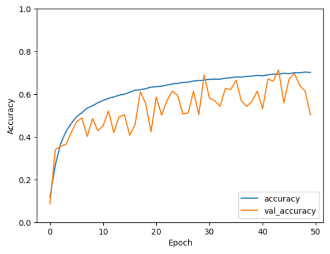

Figure 5: Accuracy per epoch.

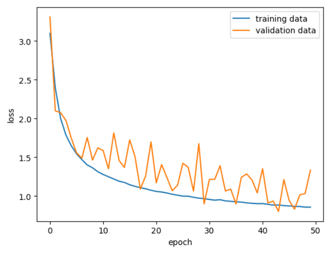

Figure 6: Loss per epoch.

4.2 Random Translations

Here, the Keras Random Translation method was utilized. The value of 0.1 was used for both the height and width factors. This reduced accuracy to **Figure** 7.

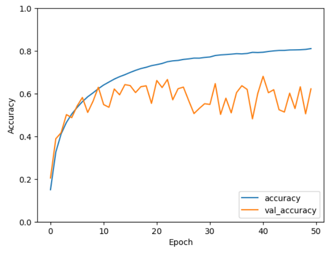

Figure 7: Accuracy per epoch.

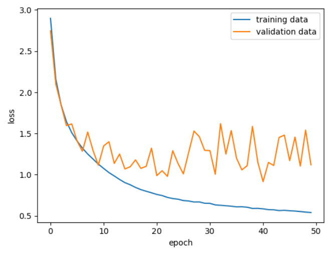

Figure 8: Loss per epoch.

5 Data Regularization and its Effects

For this task, data regularization methods were added to the model. For the best performing model, an accuracy of 98.58% was achieved, **Figure** 9. This best performing model utilized both dropout and batch normalization described below.

5.1 Dropout

Here, neurons within the neural network will randomly have their output set to zero. This causes other neurons within the same and subsequent layers to have to adjust and compensate. it is thought that this addition of dropout forces neurons to better generalize.

5.2 Batch Normalization

This method of regularization is used to compensate for internal covariate shift in a neural network. With artificial neural networks, especially very deep networks, the distribution of inputs to subsequent layers can shift to one side or the other of the network. With deeper networks, small shifts snowball into larger and larger shifts as the data passes through the network. To compensate for this internal covariate shift, the outputs of layers of neurons are normalized. That is, the means and variances of each layer's inputs are normalized. This is a process that is applied during the training step only, not during inference.

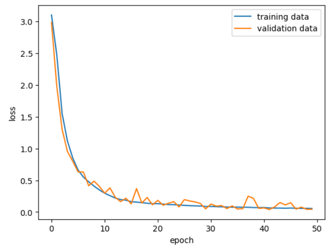

Figure 9: Accuracy per epoch.

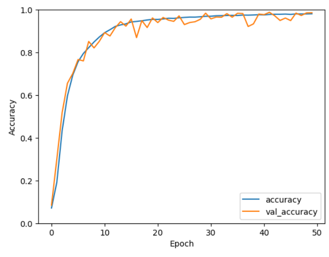

Figure 10: Loss per epoch.

6 Architecture Priors and Transfer Learning

The concept of neural network architecture priors involves taking a previously defined network architecture and utilizing it for a new problem. This can involve using just the model structure, the architecture, without the weights it may have been trained on; or, it may involve taking a pretrained model with its weights and either retraining part of it or adding additional layers to it. Both methods can be used to achieve better performance than models defined from scratch.

6.1 Untrained Model Prior

In this method, a previously defined model architecture is utilized without any trained weights. The model is then trained on the data set used for the problem at hand. This process is effectively what has taken place in this paper thus far. The model architectures of a few convolutional layers of neurons, each followed by a max pooling layer, optionally followed by dropout and/or batch normalization, and finally followed by fully connected (dense) layers of neurons, is a well defined and tested architecture prior.

6.2 Transfer Learning

This method involves taking not only a previously defined neural network architecture, but also the pretrained weights of the model. A variety of models in varying architectures are available in Keras with pretrained weights. For this exercise I utilized Xception and VGG19.

6.3 Xception

Using the pretrained Xception neural network, I added a Global Average Pool 2D layer, followed by a dropout layer, and finally an output layer of fully connected neurons equal to the number of classes. With Xception, I was able to acheive and accuracy of XXX as seen in **Figure** 11 and **Figure** 12.

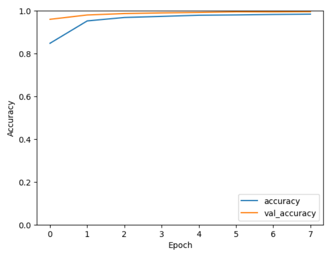

Figure 11: Accuracy per epoch.

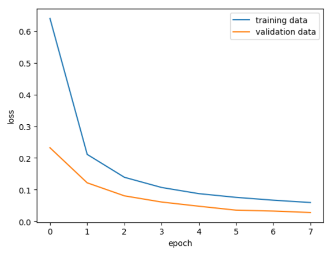

Figure 12: Loss per epoch.

6.4 VGG19

Using the pretrained VGG neural network, I added a Global Average Pool 2D layer, followed by a dropout layer, and finally an output layer of fully connected neurons equal to the number of classes. With VGG19, I was able to acheive an accuracy of YYY as seen in **Figure** 13 and **Figure** 14

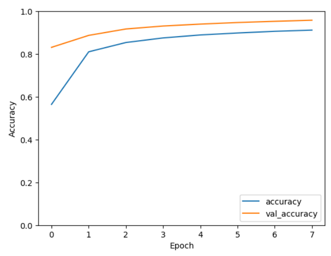

Figure 13: Accuracy per epoch.

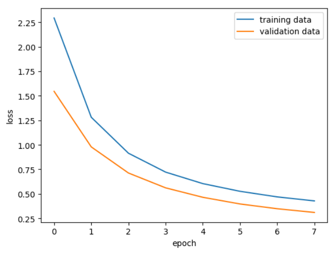

Figure 14: Loss per epoch.

7 Results

Below in **Table 1** we see the results from all of the tests. For my own architectures created from scratch, the Data Regularization model from Section 5 was the best in achieving an accuracy of 99.58%. However, the pretrained Xception model that was fine tuned onto this data set was able to achieve an even better accuracy of 99.10%.

Table 1: Results for all models

Model Name	Section	Accuracy	Loss
Overfit training set as validation	3.1	89.39%	0.29
Ground Truths as Inputs	3.2	100%	11.72
Data Augmentation (rotations)	4.1	50.32%	1.33
Data Augmentation (translations)	4.2	62.24%	1.12
Data Regularization	5	98.58%	**0.04**
Transfer Learning (Xception)	6.3	**99.10%**	**0.04**
Transfer Learning (VGG19)	6.4	95.59%	0.32

8 Conclusions

Deep Learning is a rapidly advancing field used in a variety of applications today. Computer vision, natural language processing, robotics, biological data sciences, physical data sciences, and more are all using Deep Learning as a tool to further explore their respective fields. Throughout this project, I've explored using Deep Learning for computer vision tasks. First, in intentionally over fitting a model to the training dataset. Then attempting to build a model that will train and validate on the development data set and then be evaluated on the test dataset. This has allowed me to learn about the structures and concepts used in neural networks from activation functions, layer types, optimization functions, and loss functions. I started the process on small data sets of about 1000 images during the Deep Learning Workshop in 2022. I then moved on to a few larger data sets of around 10,000 to 20,000 images between that workshop and the class. And during the class I have been using the American Sign Language dataset of 87,000 images. I'd like to continue on into datasets that contain millions of images. To do this, I would like to learn the tooling of distributed model training. This will advance my efforts to understand Deep Learning at scale including the engineering related to Machine Learning Operations (ML Ops).

References

[1] Francois Chollet. *Deep Learning with Python.* Manning Publications Co., USA, 1st edition, 2017.

[2] https://www.kaggle.com/grassknoted/aslalphabet and Akash Nagaraj. Asl alphabet, 2018.