Colby Tobin
CS6220

Assignment 4: *Problem 2.1*

GitHub for Code

https://github.com/tobincolby/CS6220-Assignment4

Data

The graphs are shown in this document, but the tabular data is in the other Excel spreadsheet attached to this submission.

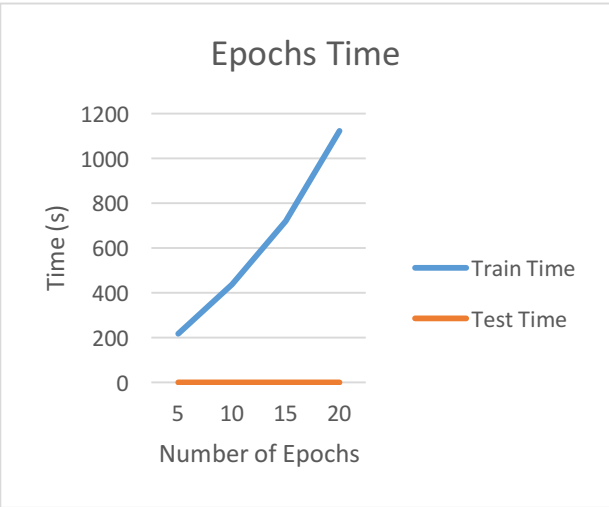
How to Run Code

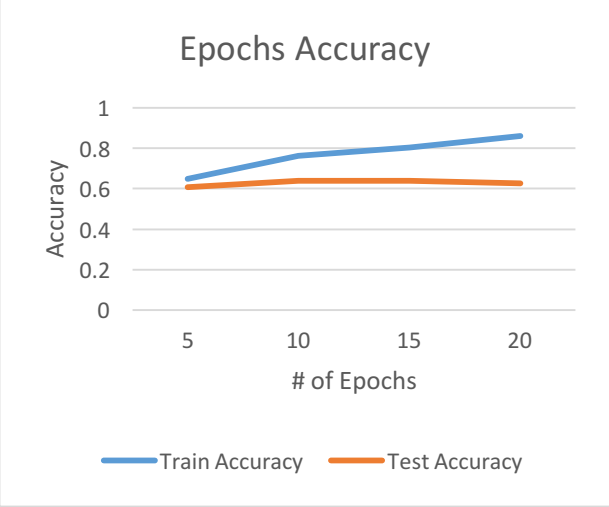
- Download GitHub files
- To train the deep learning model, run python3 train_deep_learning.py
    - This will provide you all of the train statistics
- To test the deep learning model, run python3 test_deep_learning.py
    - This will provide you all of the test statistics

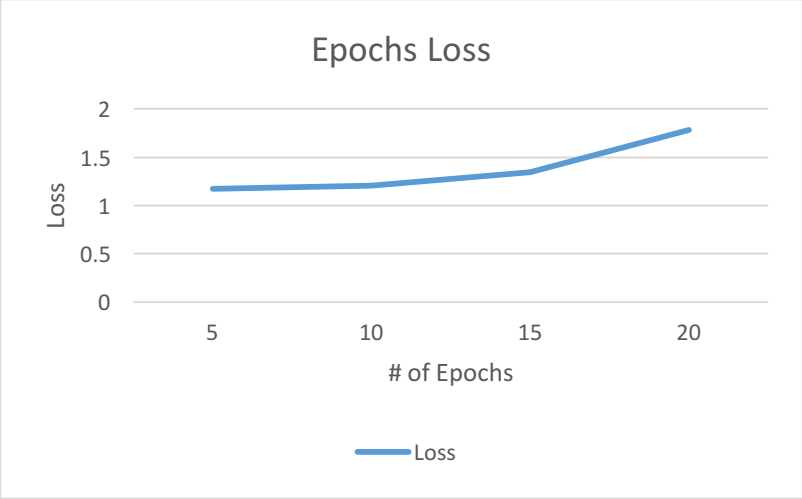
Analysis

For this analysis, it will be broken down into the analysis of each of the hyper-parameters that was varied while tuning the deep learning neural network. The hyper-parameters that were tuned were the number of epochs, the batch size, the optimizer used for learning, the number of convolutional layers, and the training set size. The number of epochs refers to the number of times that the model trained on the training set provided. The batch size refers to the number of samples that is included in each of the batches. The optimizer refers to the learning policy that will update the weights for each layer in the deep learning model. The number of convolutional layers refers to the number of layers that will perform convolutions on each sample provided. The training set size refers to the number of samples that the model will be trained on. For this problem, the Cifar10 dataset was used, and it was split into a 90/10 training/test split for validation. Outside of the parameter in question for each part of the experiment, all other parameters were maintained as constant, so only one independent variable is in question at each section.

Number of Epochs



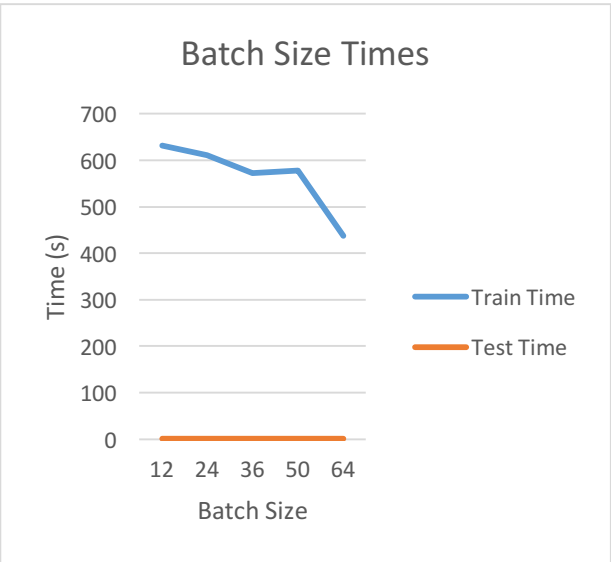


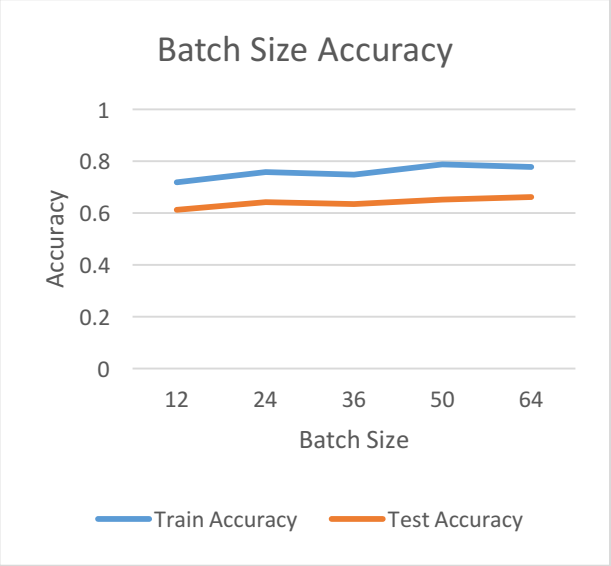


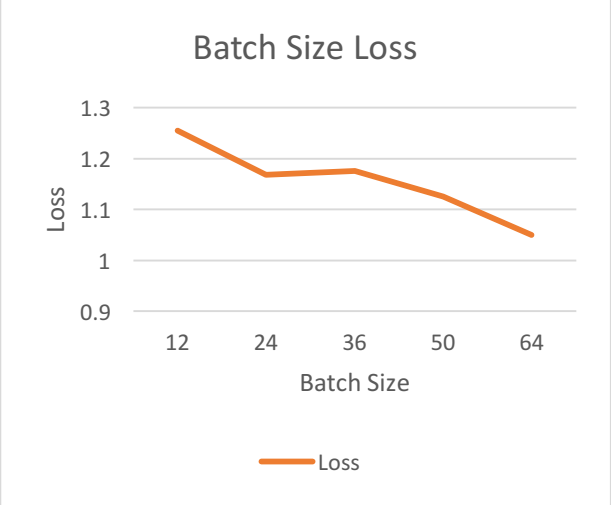
For this hyper-parameter, the number of epochs was varied at 5, 10, 15, and 20 to see how changing the number of epochs impacts the performance of the deep learning neural network. Based on the top left graph, it can be seen how the number of epochs impacts the train time and test time of the algorithm. It can be seen that the test time is not really impacted as it stays fairly consistent while the number of epochs is varied. With regards to the train time, it can be seen that the train time increases linearly as the number of epochs increases as well. Based on the graph, it can be seen that each epoch takes about the same number of time to train. When looking at the top right graph, it compares the train and test accuracy for the different number of epochs. As the number of epochs increases, the train accuracy increases along with it. When looking at the test accuracy, it can be seen that it increases up to 15 epochs. After 15 epochs, the graph begins to overfit and the test accuracy decreases at 20 epochs. Based on accuracy alone, it can be seen that the ideal number of epochs was 15 because the accuracy decreases after the epochs increases past 15. When looking at the loss graph, it can be seen that as the number of epochs increases, the loss increases slightly as well up until 15 epochs. After 15 epochs, the loss increases at a much higher rate indicating that the predictions become much worse after 15

epochs. Based on these findings, it can be seen that the ideal number of epochs for this deep learning model is 15 epochs because it has the best test accuracy and has the ideal loss for this dataset.

Batch Size



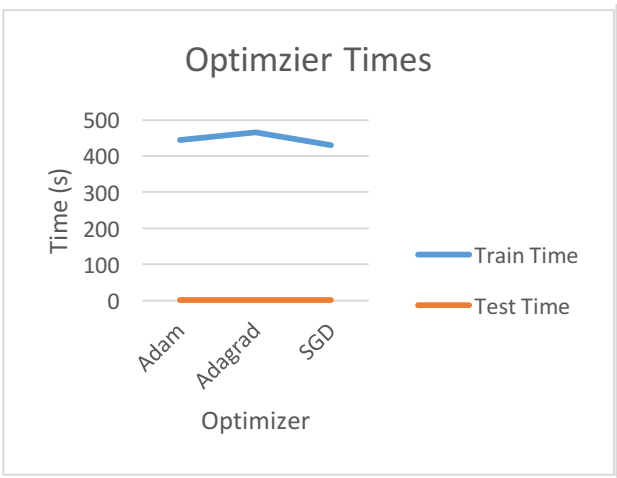


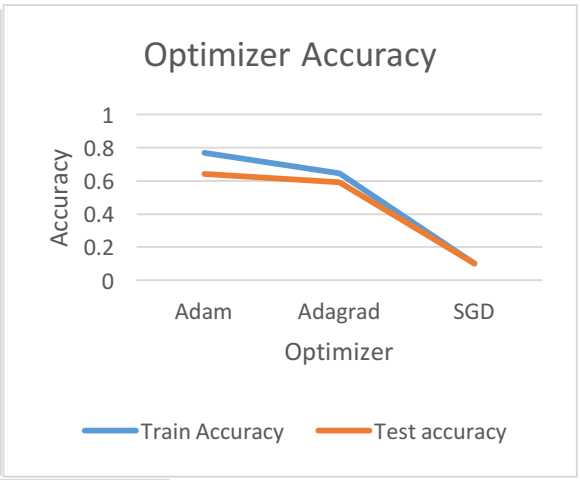


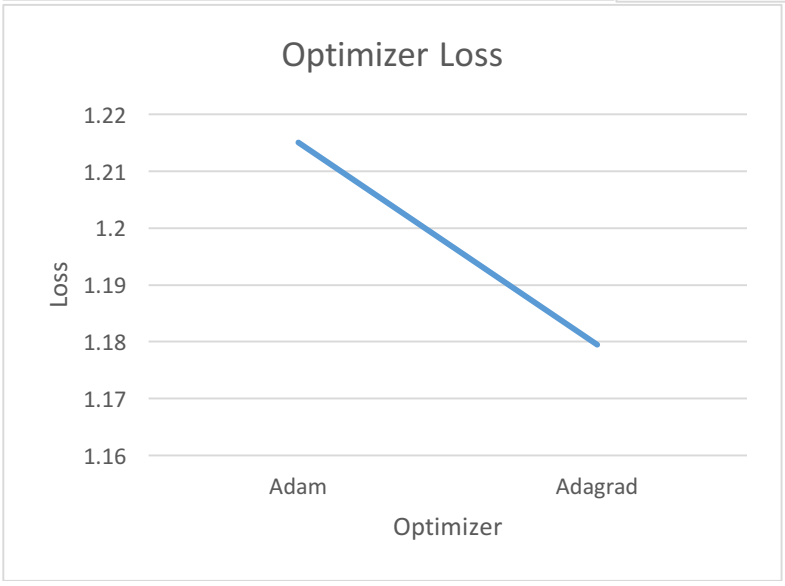
For this hyper-parameter, the batch size was varied between 12, 24, 36, 50, and 64. The batch size represents the number of samples in each batch for training. The top left graph shows how the train and test time is impacted as the batch size changes. Based on the graph, it can be seen that the test time is not really impacted as the batch size increases because the batch size deals mostly with just the training stages of the model. When looking at the train time, it can be seen that as the batch size increases, the train time decreases. This makes sense because increasing the number samples in a batch will decrease the number of batches, speeding up the train time. With this in mind, the batch size of 64 is ideal for this model. The top right graph shows how the train and test accuracy is impacted by the different batch sizes. Based on the graph, it can be seen that as the batch size increases, the train accuracy increases as well. When looking at the test accuracy, it can also be seen that as the batch size increases, the accuracy increases as well up until a batch size of 64. When looking at the loss graph, it can be seen that as the batch size increases, the loss decreases. This shows that with an increase of batch size, the predictions of the model start to become better overall. Based on the above descirptions, it makes

the most sense to have a batch size of 64 because it trains the quickest with the best predictions and the best test accuracy as well. With this in mind, a batch size of 64 is ideal for this model.

Optimizer



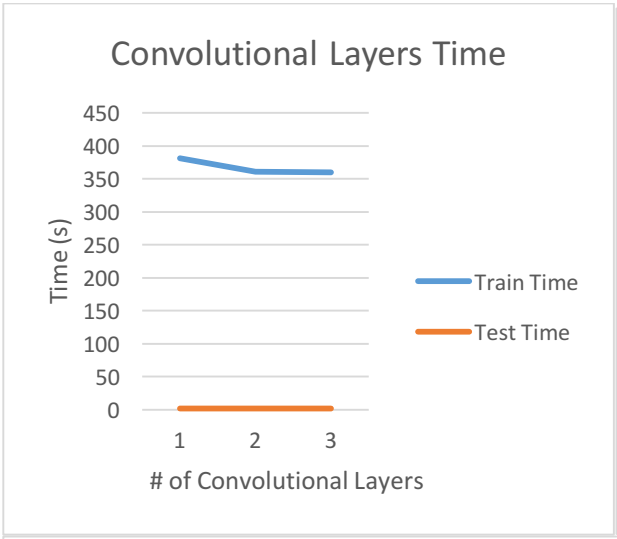


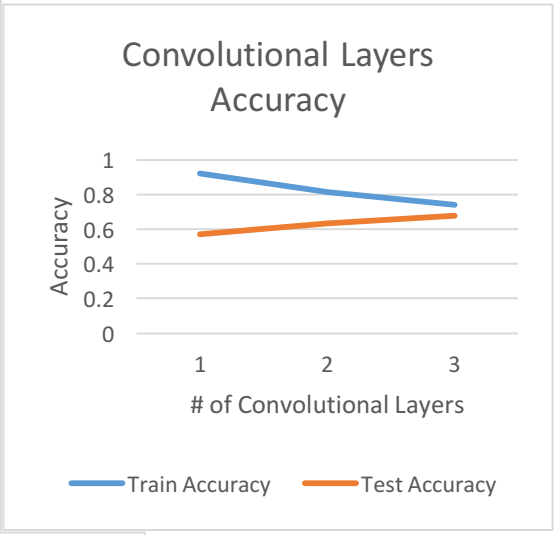


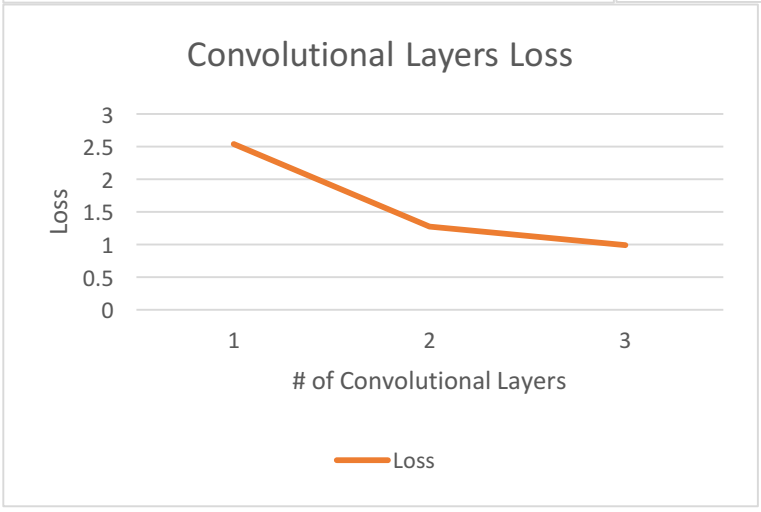
For this hyper-parameter, the optimizer used for the learning policy was varied between the adam optimizer, the stochastic gradient descent optimizer, and the adaptive gradient optimizer. The top left graph shows how the training time and testing time is impacted when the optimizer is varied. Based on that graph, it can be seen that the test time is not really impacted by which optimizer is used for training because of the test time stays fairly consistent, which makes sense because the optimizer mostly deals with training. When looking at the train time, it can be seen that the worst train time is the adaptive gradient optimizer, and the best train time is the stochastic gradient descent, and the adam optimizer is right behind it with regards to train time. The top right graph demonstrates how the accuracy of the model is impacted by which optimizer is used. Based on the graph, it can be seen that the model performs the best on the training data and test data with the adam optimizer. The adagrad optimizer performs slightly worse than the adam optimizer. However, when looking at the stochastic gradient descent optimizer, it can be

seen that it actually performs worse than random guessing for this dataset on the test data. With this in mind, SGD can automatically be ruled out as a optimizer to be used. The bottom graph deals with how the loss is impacted by the optimizer used. It can be seen that the adagrad optimizer performs with a better loss than the adam optimizier. Taking everything into consideration, it makes sense that this model uses the adam optimizer. It performs with the best accuracy on the test dataset, and is able to train quicker than the adagrad optimizer. While the adagrad optimizer has a better loss result, a more important measure is the accuracy of the model, which shows that the adam optimizer is the ideal learning policy for this dataset and this model.

# of Convolutional Layers



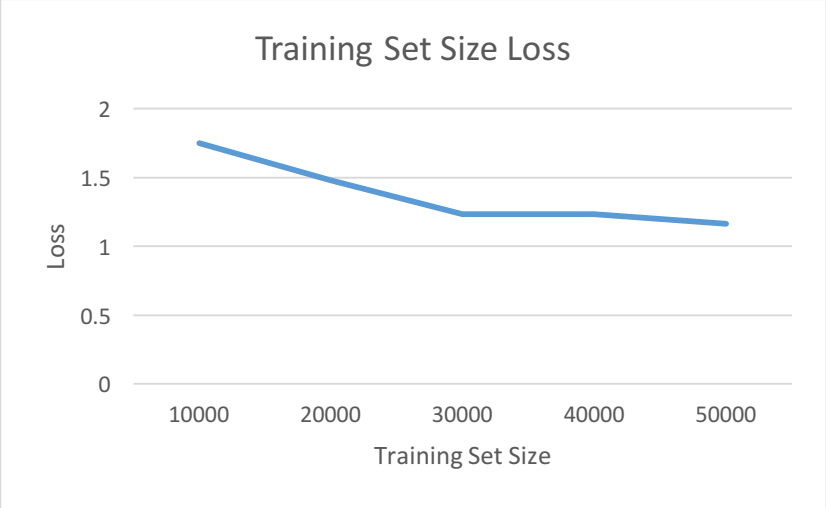




For the number of convolutional layers in the deep learning model, it was varied between 1, 2, and 3 convolutional layers where at the end of each layer, a max-pooling occurred over the last convolution. The top left graph shows how the train and test time are impacted by the different number of convolutional layers in the model. Based on the graph, it can be seen that the

test time is not really impacted by the number of convolutional layers because the test time stays fairly consistent throughout the variation in number of convolutional layers. When looking at the training time, it can be seen that as the number of convolutional layers increases, the training time decreases. This doesn't necessarily make sense because one would think that with more convolutional layers, it would take more training time, but based on the data, it can be seen that it decreases the training time. The top right graph deals with how the accuracy of the model on the test and train set is impacted by the variations in the convolutional layers of the model. It can be seen that as the number of convolutional layers' increases, the train accuracy decreases. This might be due to the fact that with less convolutional layers, the model is just learning the training data itself instead of learning what is actually in the image itself. As the model starts to learn what is in the image, the accuracy begins to adjust and decrease. This would give rise to the findings in the test dataset because as the number of convolutional layers increases, the test accuracy increases as well. This relates back to the previous explanation. The lower number of convolutional layers result in learning the data instead of learning what is actually in the image. Once the model starts learning what is actually in the image, it starts to perform better on the test dataset and give rise to better accuracy. When looking at the loss graph, it can be seen that as the convolutional layers increase, the loss decreases, which means the model is performing better predictions, so a larger number of convolutional layers is better with respect to loss. When looking at the above graphs and analysis, it can be seen that the ideal number of convolutional layers for this model is 3 because it results in a more accurate model that takes less time to train than the other number of layers.

Training Set Size



The hyper-parameter for training set size was varied between 10k, 20k, 30k, 40k, and 50k samples while the test dataset size was maintained as a constant throughout to ensure that there was only 1 independent variable. The top left graph deals with the train time and test time and how they are impacted by the training set size variations. Based on the test time, it can be seen that the training size variations do not really impact the test time because the training set size hyper-parameter deals only in the training setting. When looking at the train time, it can be seen that the variation of the training set size forms a linear graph where the increase in training set size results in a proportional increase in training time. This makes sense because as the training size increases, the model must train on more samples, resulting in a longer train time. The top right graph deals with how the accuracy of the model is impacted by the variations in the training set size. When looking at the train accuracy, it can be seen that as the training set size increases, the train accuracy seems fairly consistent. This makes sense because the model is only testing itself against its own training data for the training accuracy, so it should be able to classify its own training samples at about the same accuracy even when the number of samples increases because the number of training epochs was kept consistent. When looking at the test accuracy, it

can be seen that the test accuracy improves as the number of training samples increases. This makes sense because introducing more training samples will result in a better model that can learn a wider variety of samples and learn more about the dataset itself rather than learning the training data alone. Increasing the number of training samples decreases overfitting and results in a better model. That idea is furthered when looking at the loss graph as it shows that as the training set size increases, the loss decreases, which means that the model is resulting with better predictions. While the training time increases as the number of samples increases, it still makes sense to train the model with the full dataset because it reduces overfitting and results in a model that performs better in the real world on data that it has not seen before.

Overall Conclusions and Opinions

Based on the analysis performed above, the ideal parameters for each section have been determined. For the training set size, it is ideal to train on the entire training dataset for this model. For the number of convolutional layers, it makes sense to use 3 layers. For the optimizer, the best choice for this would be the Adam optimizer, which is typically the default for this type of deep learning model. For the number of epochs, the ideal number of epochs was 15 based on the test accuracy, test loss, and training time overall. For the batch size, the ideal batch size was 15.

For this experiment, if given more time, I would like to have been able to expand upon the batch size experiment. Based on the results, the test accuracy and the train time continued to decrease as the batch size increased. I would like to have performed more experiments with it to see if this trend continued, or if I could find a point where the test accuracy began to decrease again to truly find the optimal batch size for this dataset. Overall, this could not be done due to the computational constraints on my machine with the given time to do this assignment.

Overall, I enjoyed this assignment. It truly showed me how making small tweaks to seemingly insignificant parameters to a model could impact the overall results greatly. After doing this experiment, I wish I had a better way to run the experiments. Each of these experiments took a very long time to run, and I actually had to let my computer run over night to complete everything and gather all of the data that I needed. The only problem with this was when my machine encountered an error or went to sleep, I would have to restart and try to figure out a way to regather all of the data again for the experiment.