Identifying 32x32 Images Amongst 10 Categories Using a Convolutional Neural Network

Ryan Tran & Thomas Bryant

Problem Statement:

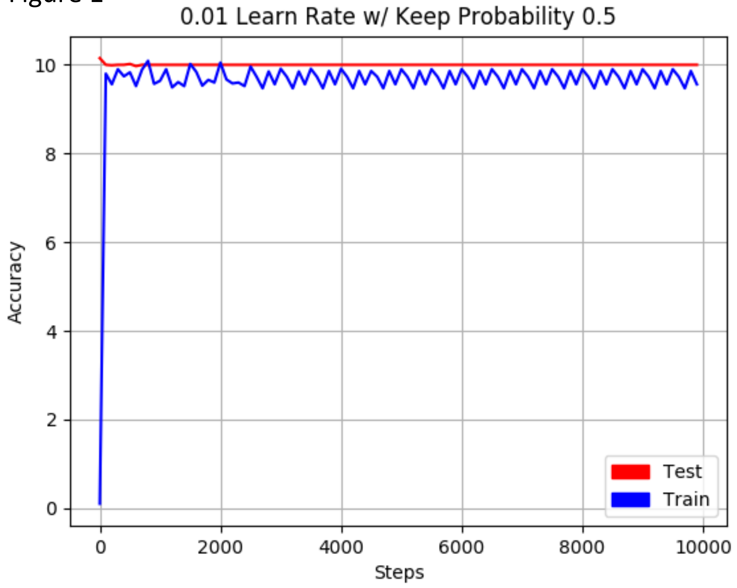
We will create a Convolutional Neural Network (CNN) using Google's API, Tensorflow, to identify 32x32 images amongst 10 categories: airplane, automobile, bird, cat, deer, dog, frog, horse ship, and truck. Motivation for this project is to learn more on how to implement Tensorflow to real world problems and have a better understanding of Convolutional Neural Networks. This project was made possible by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton for providing a dataset of well cropped images of real world objects.

Creating the Model:

The batch size and number of filters in each convolutional layer is constant throughout every test. Through experimentation, we discovered that any batch size greater than 100 and filter greater than 128 was too computationally intensive for our machines. Thus, in the following experiments we limit our changes to only the learn rate, drop rate, and number of steps.

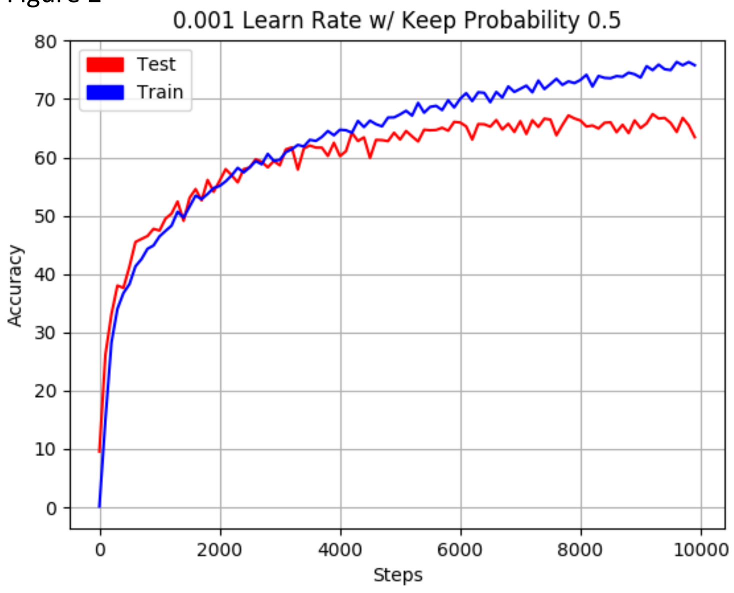
In our first model, we choose arbitrary hyperparameters because we had no assumption on what kind of results we would obtain and how our parameters would affect it. Thus, our goal was to simply create a working CNN model regardless of its results. Running our model with arbitrary hyperparameters outputted the following:

Figure 1



From figure 1 we noticed two possible issues with our model. Either we are not running enough steps or the learning rate is too high. Because this model required two hours to generate, we decided to run the model again with a lower learn rate versus running it with more steps. The results were quite insightful.

Figure 2



The accuracy of our model exponentially decays as the number of steps increase. However, it seems that our model still has the potential to learn more if we increase the number of steps drastically. Furthermore, the graph indicates there is a possibility that our model is overfitting the data because the testing accuracy exponentially decays faster than the training accuracy. In the following graphs, we increased the number of steps to 60,000 to see whether our model can continue generalize the data. The training time was approximately 8 hours.

Figure 3

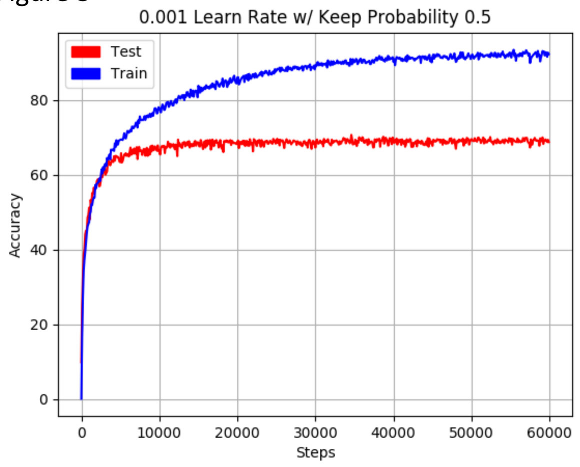
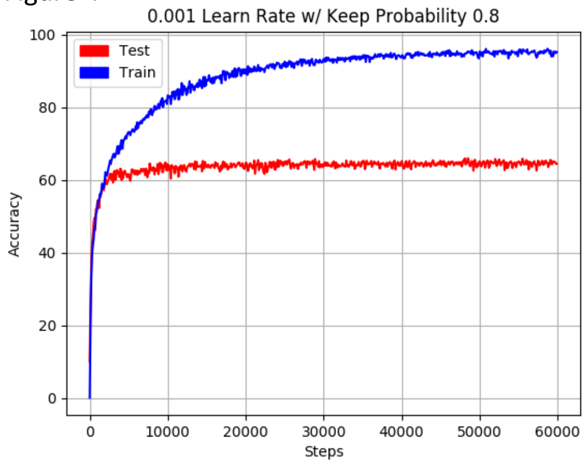


Figure 4



Going off figure 3, we discovered that our model is overfitting the data. This discovery confused us greatly because our model implements dropout to avoid this case. Thus, we decided to decrease the drop rate to check if our dropout layer is working correctly. The results are shown in figure 4. Clearly, the dropout layer is working because it is affecting the accuracy on both test and training sets. We hypothesize that increasing the drop rate and number of steps will increase test accuracy. The results are shown below.

Figure 4

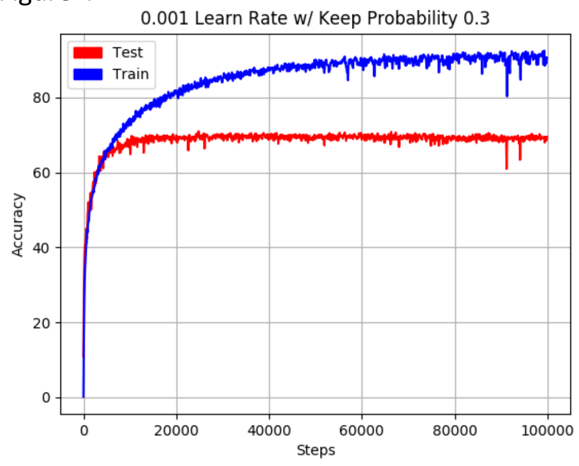
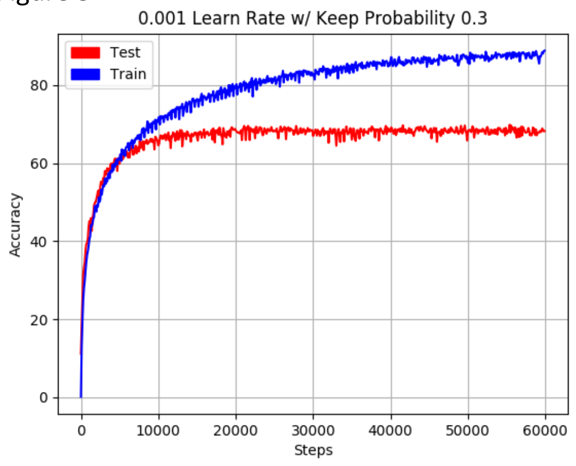
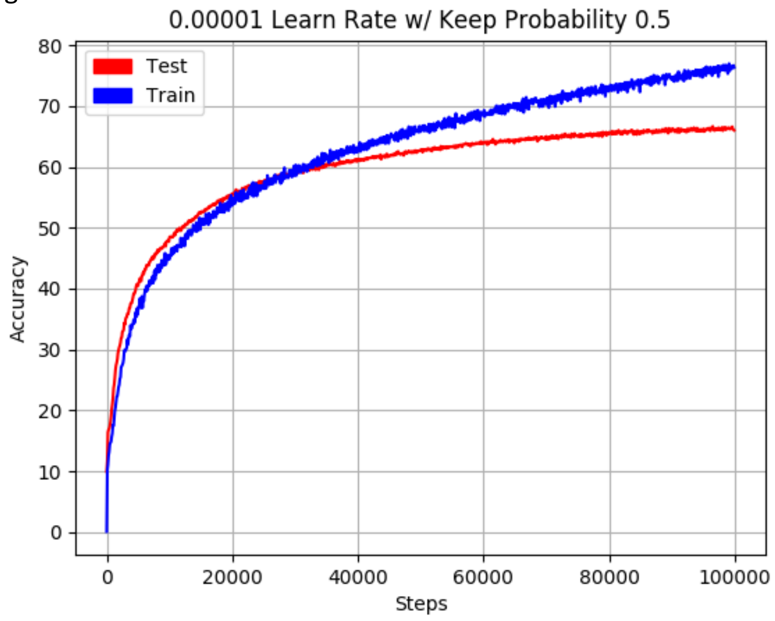


Figure 5



Unfortunately, increasing the drop rate did not improve the test rate (figure 5). In fact, the accuracy on both training and test set seem more sporadic and inconsistent. Increasing the number steps from 60,000 to 100,000 also did not improve the test accuracy (figure 4). Thus, the only hyper parameter left to change is the learning rate. In the following model, we reduced the learning rate from 0.001 to 0.00001.

Figure 6



The above model took over 14 hours to train but the results were quite surprising. Apparently, our learning rate from before was too high. We feared that out model would forever remain at a 65% test accuracy but figure 6 tells us otherwise. Furthermore, figure 6 gives us reason to believe that if we increase the step size exponentially, and maybe use a smaller learn rate, we can achieve a test accuracy above 80%. However, because of time constraints and the overly long training time, we were unable to run another model with higher steps.

Conclusion & Improvements for the Future:

Although we were unable to achieve an adequate test accuracy with our CNN, we did learn a great deal on the sensitivities of CNN. Building our model required many hyper parameters, parameters that determined whether our CNN would work properly. Our model was overfitting due to a high learning rate and lack of steps. To remedy this issue, we believe that applying an exponentially decaying learning rate will improve test accuracy. We could decrease the learning rate and increase the number of steps but we may fall in a perpetual cycle of repeatedly decreasing and increasing the parameters every time the model starts overfitting. Moreover, insufficient data could be another contributing factor to our overfitting issue. Applying noise to our dataset could improve our model because we would be artificially creating more data.

To recapitulate, our CNN may not be the most efficient model, but the experience gain through coding and experimentation has bettered our understanding on the inner works of a CNN. With this knowledge, we hope to improve our model in the near future.