Öberdrive (deep) Final Paper

CS 374 - Machine Learning & Data Mining

Oberlin College

Professor Adam Eck

Fall 2019

Brendan Fontanez, Dizzy Farbanish, Kristoph Naggert

Abstract

Vehicle related accidents are the leading cause of death among those under the age of 44, responsible for more than 30,000 deaths and 2,000,000 injuries every year. These accidents present a huge economic burden on the country as well as on the individuals involved. This issue is currently being addressed through the development of autonomous vehicles; this coupled with the vast projected growth of the self driving car sector make the topic intriguing. In this project we investigate the self driving model developed by Nvidia and create our own model for a self driving car. Through testing we find that the Nvidia model thoroughly outperforms our implementation reaching accuracies of 80 and 89.9 percent based on certain specifications, where as our model only achieved at its best approximately 22% accuracy.

Introduction

Bigger Picture

Motor vehicle accidents represent a massive burden upon the United States. Unintentional injuries, which include automobile accidents, are the third leading cause of death in the United States and are the leading cause for those aged 1-44. Each year more than 32,000 people are killed and a further 2 million are injured in vehicle crashes (Center for Disease Control 2013). A study conducted in 2010 by the National Highway Traffic Safety Administration (NHTSA) estimated that the cost of vehicle crashes was $242 billion, including 57.6 billion in decreased workplace productivity, and $594 billion due to loss of life and decreased quality of life (NHTSA 2010). The major driver of these statistics are human related, 94% of accidents come down to human error, in the US seatbelt use was significantly lower than in other developed countries. Approximately one third of crashes involved drunk driving, additionally 1 in 3 accidents were caused by speeding. Although these statistics paint a bleak portrait of the current condition of the United States, the rapid improvement of autonomous driving vehicles is cause for excitement. The automotive industry has made large strides in recent years regarding safety, current motor vehicles have technology that helps drivers avoid drifting into adjacent lanes or gives warnings when attempting to make unsafe lane changes, many have back facing cameras to make backing up saver, and some have automatic brakes to avoid hitting cars that stop suddenly. Autonomous driving technology aims to further improve upon these safety technologies. On top of the safety and economic benefits that autonomous driving vehicles provide, there is a good amount to be said about the efficiency and convenience of self driving technology. The presence of automated vehicles would improve traffic flow and reduce congestion, both major problems in today's society, in 2014 alone Americans spent nearly 7 billion hours in traffic (NHTSA). A study conducted by McKinsey & Company estimated that automated vehicles could reduce the amount of time spent driving by up to 50 minutes a day, this would have massive implications in terms of additional time being available to invest into work and family as well as a boon to the effort to reduce carbon emissions (The transportation sector was the largest contributor to greenhouse gas emissions accounting for 29% of all emissions in 2017 (United States Environmental Protection Agency 2017)).

Given the expansive nature of these problems, we predict that the market for autonomous vehicles will only grow over the coming years. In fact according to an article in *Forbes*, the self-driving car market could grow to ten times its current size over the next 6 years (*Forbes*). This means the average person will be more likely to encounter autonomous vehicles in their daily life with more and more frequency. A major motivation for this project was to gain a better understanding of how these vehicles work so that we can be better prepared to integrate them into our society and our lives.

Specific Problem

The development of autonomous vehicles as a solution to problems involving sensory and motor function controls in three-dimensional environments has been studied for many years. The biggest roadblock in the development of these technologies comes in the form of navigation; and more specifically navigation of densely populated environments. The reasons for this are plentiful: complex dynamics at traffic intersections; the difficulty of tracking the movements of many actors that may be in view at a given time; the recognition of traffic signs, lights, road markings and other postings as well as adhering to traffic laws and regulations; distinguishing between different types of vehicles also occupying the road; one also needs to address events the happen unpredictably, such as construction, drunk (or otherwise inebriated) drivers swerving into the wrong lane, or even pedestrians jaywalking; and needing to rapidly adapt to simultaneous events, such as decelerating to avoid running over a child who has wandered into the street while simultaneously needing to keep enough speed to avoid being rear-ended by a car approaching quickly from behind. However, many of these are probably too large in scope to address in this project. Therefore, we have chosen to focus on identifying road markers (i.e. lane markers and distinguishing the road from the surrounding area.) for our project. Given the vast number of possible neural network model designs, we were also interested in confirming our model's effectiveness by comparing it to a second one.

Short Summary of Solution

Our proposed solution was to design and build a neural network AI system that was able to effectively steer a vehicle. Specifically, we wanted our network to be able to identify the lines on either side of a road and keep the vehicle between them by steering it accordingly. Unfortunately, we did not have the financial resources to implement this solution in a real-world vehicle. Instead, we used a driving simulator built for testing machine learning techniques on autonomous vehicles. For this project, the specific model we decided to use the Udacity Driving Simulator. Because our neural net was going to be used for image identification, we used a convolutional neural network (CNN). As we were somewhat limited in our access to hardware and time, we simplified our neural net design into a line identification program. Each instance in our training phase was a frame from a single camera input situated at the front of the vehicle. Our goal was to have the neural net learn to recognize the lines delineating each side of the road. By manually steering the vehicle along a relatively clear, standardized stretch of highway, Udacity assigns steering values to each frame in the input image feed and returns them in a CSV file - these values acted as our instance labels. For this project we make use of a predefined implementations, the Nvidia model, as well as a simple CNN implementation created by our group.

Experiments Run/Quick Results

For each of our two models we ran 12 separate tests varying the number of epochs the model was trained for, and whether or not we bucketed the data and include dropout. Our best accuracy for the Nvidia model were achieved using 10 training epochs, and no buckets or dropout, and yielded a directional accuracy of 89.9% and an interval accuracy of 80%. The best accuracy that our implementation reached was 21.94% and 21.74% for interval accuracy and directional accuracy, respectively, the specifications used to get this accuracy were 30 training epochs, using buckets and dropout.

Background/Related Work

Our project relates to the field of autonomous vehicle research. *The Zebra*, a company that compares insurance between automotive vehicles, has a useful article defining the key classes and characteristics of autonomous vehicles. Vehicular autonomy as defined by the Society of Automobile Engineers is split into five levels: L1 is driver assistance, where the car automates small features like accelerating and breaking but the driver is almost entirely in control. Most modern vehicles are in L1. L2 is partial automation, this includes features such as cruise control with automated braking. L3 is conditional automation, the driver may give control to the vehicle in some situations but the driver must remain attentive to the behavior of the car. L4 is high automation, the vehicle can drive itself in many situations. L5 is full automation, the vehicle can drive itself effectively in all possible situations. Higher level autonomous vehicles use various types of camera and sensors to build a map of their surroundings, plan a course, and avoid obstacles (*The Zebra 2019*).

The first well documented project involving the use of deep neural networks to address the autonomous driving problem was the Autonomous Land Vehicle In a Neural Network (ALVINN). Developed at Carnegie Mellon the ALVINN model made use of a 3-layer back propagation network to accomplish the task of road recognition. The ALVINN model took in images from a camera and output the direction in which the car should head in order to stay on the road. The images used for training were simulated. Tests that use the ALVINN model were successful in following roads under certain training conditions at speeds of up to ½ meter per second (Pomerleau Dean A. 1989). This research set the foundation for future progress in the field of autonomous vehicle technology.

Since the creation of ALVINN, there has been a great deal of industry interest and investment into researching higher level autonomous vehicles. Machine learning is a significant factor in autonomous cars' ability to understand and act upon complex situations on the road. Regression algorithms, pattern recognition/classification algorithms, clustering, and decision matrix algorithms are all important for object detection, object recognition, movement prediction, and path formulation. (Anil Gupta, *IIOT world*). Some companies like Nvidia have invested huge amounts of resources specifically into autonomous vehicle research. They have been able to develop some of the most effective self-driving car models, as is demonstrated in the results of this paper.

Problem

Udacity Driving Simulator and Data Collection

As mentioned in the introduction we did not have access to a robotic car to train our model. Instead we made use of the numerous driving simulators available to the public. In our initial research we had planned on using either CARLA or deepdrive, Unfortunately, both, required a dedicated GPU in order to run, as well as operating system specifications, and none of us were in possession of computers with the ability to properly handle these programs. This roadblock forced us to pivot to a lower resolution driving simulator - The Udacity Driving Simulator. Udacity offers two separate driving modes: training and autonomous. The manuel training mode allows for an individual to control a car on a chosen track using the W/A/S/D keys, these controls correspond to forward, left, break/reverse, and right. The car has three cameras, center, left and right facing. The training mode also has a feature called 'recording mode', to track a drive all a user needs to do is to click the record button and drive. By recording a drive the simulator take images with each of the cameras at periodic intervals and saves them to an image directory, additionally steering angle, throttle, breaking, and speed are recorded and output into a CSV file with absolute paths to the corresponding images. The Udacity driving simulator has two premade tracks, a simple and a complex track. The simple track is a vaguely circular loop with only one inward turn. The complex track is also a loop, but has much more of a curving snake-like pattern that includes many sharp left and right turns and significant elevation change. For this project we create our data set by driving on the simple track for two laps, as the complexities of the other track resulted in a lot of noise and poor performance of our models when compared to the simple course. That is not to say that the simple course didn't have fundamental flaws, due to its simple nature the track consists of a disproportionate number of left turns and straight stretches, creating bias in our models. We address this issue in two ways, the first was to drive the course backwards for an equal amount of time that we drove forward to balance the number of turns. However, this doesn't solve the problems that arise from the majority of the track being straight stretches of road. The panacea for this is described in the following section on data preprocessing.

The simple track from the Udacity Training Simulator. (Source: Kocić)

Data Preprocessing

As is the case with most problems involving image recognition data cleaning and preprocessing is massively important to the performance of the models being evaluated. The absolute paths retrieved by the Udacity program are unreadable (e.g. 'D:\Desktop - Data Drive\ML Project\IMG\center_2019_12_02_12_13_57_805.jpg'), and therefore the first step for preprocessing is to correct those path names. This is accomplished by splitting the string and returning only the final index, which is the name of the .jpg, we then append a hard coded path to the image directory. Through testing we noticed that when included the left and right facing cameras introduced a lot of noise that harmed the accuracy of our models, for this reason we choose to drop the left and right images at this point in the preprocessing. It was also difficult to manipulate the model to treat all images as a single instance. For simplicity sake we are only interested in predicting the angle of steering of the car given an image, therefore we also drop throttle, reverse and speed at this point. Normally this would cause problems, however, we specifically built our data set with this in mind, to avoid the issue not accounting for speed and acceleration when training we always accelerate and never brake or reverse, and Udacity has a built in maximum speed which takes care of the problem of speed increasing indefinitely if we constantly accelerate. At this point in the process were left with an absolute path to the center camera image and an associated steering angle. Next, the data set is randomly shuffled and split into training and test sets and then further split up into training instances, training labels, test instances, and test labels. The final step before we actually train our models is to preprocess the images, to do this we loop though the training/test instances and read in the image, the image is converted from RGB to YUV color space, this step is followed by a 3, 3 Gaussian blur and then resized to the proper dimensions proportional to our blur.

In the previous section we alluded to a solution to the issue of unequal numbers of each steering classification (The overrepresentation of turning angles at or near 0, relative to left and right turns, due to long stretches of straight driving). The major problem with this is that in certain training sets where the majority of images and labels are of straight roads the model will learn to always predict a steering angle of 0. To address this we use bucketing to normalize the counts. At the time we create the training set, we loop through the labels and classify each instance as one of center, left or right, or label == 0, <0, >0. We then count the number of each classification and return the lowest number among them; this becomes the number of each type of steering class included in the training set (e.g. Say in a training set we have 50 left, 46 right and 192 straight instances, with bucketing we would only include 46 of each type in our training set). After all of this the processed images are passed into our models.

However we find that bucketing is only effective when we use our largest datasets, we find that on smaller datasets, the bucketing can drastically shrink the size of our training set. (eg. 10 left, 300 right, 400 straight), we shrink our dataset to a set of 30 instances. So we see that bucketing can be dangerous and can be damaging to datasets where labelled are skewed in a way where there is a very small minimum label. Bucketing is theoretically useful, but is often adversarial to our training process in practice.

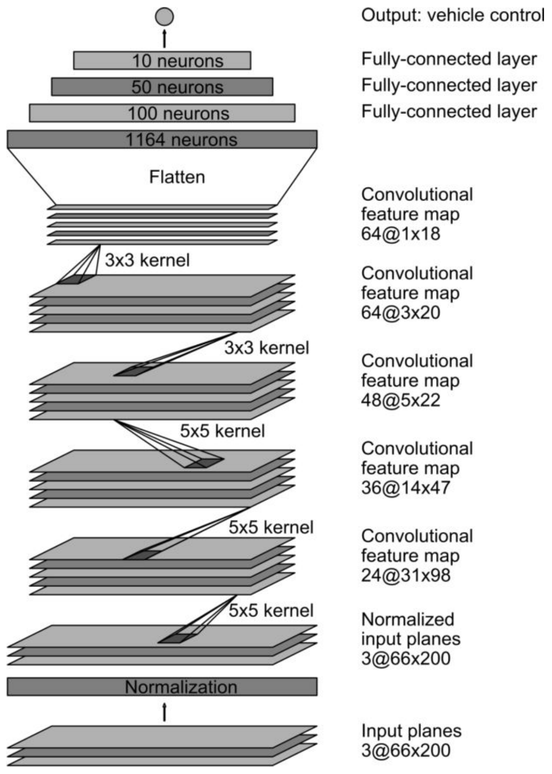
Solution

Convolutional Neural Network

Because our input data was in the form of images, the best choice of the possible machine learning techniques was a convolutional neural network, or CNN. Like any neural network, a CNN is essentially a series of layers of individual neurons. Each neuron consists of a series of weights that are repeatedly adjusted as the network is trained on a data set. A convolutional neural network is a type of neural network that is specifically designed to be suited to processing images and visual data. CNNs are commonly used for image classification and similar problems. The titular convolutional layer functions by moving a "kernel" (or explicitly sized subsection) across the input image. This allows the CNN to recognize nuances in the image data that would otherwise be missed by a more traditional neural network (Saha).

Nvidia Model

The focus of our project was to create a working implementation of the Nvidia self-driving car model and that could then be applied to our own data. In 2016 a group of developers at Nvidia created a model that made use of CNNs to map pixels, from images taken by a front-facing camera, to steering instructions for an autonomous vehicle. The weights of the model were trained to minimize the MSE between steering angle output by the program and either the command of the human driver or the adjusted steering command for off-center and rotated images. The model itself consists of 9 layers, 5 2-dimension convolutional layers, and 3 fully connected layers. The Nvidia version of the model has an image normalization layer that is hard coded; this is replaced by the data preprocessing we conduct in our project. The convolutional layers perform feature extraction and were chosen empirically through a number of experiments where layer configurations were varied. The first 3 convolutional layers are strided convolutions that use a 2x2 stride and 5x5 kernel, the final 2 convolutional layers are non-strided convolutions that use a 3x3 kernel size. After the convolutional layers the output is flattened before being passed into 4 fully connected layers, of descending size (1164, 100, 50, 10), leading to a final output neuron which is the steering angle (Figure 1) (Mariusz Bojarski 2016).



A visualization of the Nvidia CNN. (Source: Bojarski)

Group Implementation

The purpose of the group implementation was not necessarily to create a trained model that would successfully be able to keep an autonomous vehicle on the road, but instead to serve as a comparison to the Nvidia model. The Nvidia model was empirically chosen through extensive research and testing and represents the best in autonomous driving algorithms. One of our goals in this report was to highlight the performance of the Nvidia model compared to other implementations. The implementation that our group constructed consists of a single 2 dimensional convolutional layer, necessary for reading in image data, followed by a flattening step and 3 fully connected layers containing 512 neurons and an output neuron.

Experimental Setup

Keras

All model implementations were created using keras. Keras is a high-level neural network API, designed to enable fast experimentation with deep neural networks. Keras has a plethora of implementations of commonly used neural network building blocks such as layers, activation functions, optimizers and a host of tools to make working with image and text data easier to simplify the coding necessary for writing Deep Neural Networks code. In addition to standard neural networks Keras has support for CNNs and recurrent neural networks (RNNs). It also supports common utility layers like dropout, batch normalization, and pooling (Keras). This breadth of available networks and utilities made it perfect for our project.

Hyperparameters and Specifications

Here for all models we used the default Keras learning rate of 0.001, we train each model on 90% of the available data (before bucketing) leaving 10% for testing. We chose a ELU hidden layer activation and calculate loss using mean squared error (MSE). The optimizer we use is Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments (DP Kingma 2014). We vary the number of epochs that we use to train each model (10, 15, 30). In addition to varying the number of training epochs, we experiment with including dropout after the first fully connected layer (Dropout is a method of combating over fitting where the model goes over each neuron, and based on a predefined probability randomly chooses to keep or delete each neuron), and bucketing of the data.

Performance Measures

For this project we use several different measures of performance, directional accuracy, interval accuracy and the autonomous driving mode available in Udacity. Directional accuracy is defined as whether or not the model predicts the direction that the car should turn correctly. Similarly to when we normalized our training set with buckets, we again define center, left and right and we do this for both the actual steering angle as well as the predicted steering angle. Then we check to see if the classification for the prediction matches that of the actual label, if it does it is counted as correct and accuracy is calculated by taking the number of correct predictions divided by the total number of predictions. The interval accuracy measures how frequently we get within a certain distance of the actual label. To determine interval accuracy we build an interval of ±0.1 around the actual label and check to see if our predicted value falls in that interval, if it does it is counted as a correct and interval accuracy is calculated by taking the number of correct predictions divided by the total number of predictions. The final performance measure used is the autonomous mode in the Udacity Driving Simulator, the autonomous mode can be run from the terminal by passing in a trained model. This performance measurement is difficult to quantify in numeric terms, and therefore is evaluated by whether or not the car successfully drives on the road.

Results

Model	**Epochs**	**Bucketing**	**Dropout**	**Training Instances**	**Interval Accuracy**	**Directional Accuracy**	**Loss**
Nvidia	10	No	Yes	4632	0.8	0.899	0.079
Nvidia	10	No	No	4632	0.8	0.899	0.0178
Nvidia	10	Yes	Yes	1245	0.8019	0.87	0.0624
Nvidia	10	Yes	No	1245	0.06*	0.128*	0.1402*
Nvidia	15	No	Yes	4632	0.8	0.899	0.0179
Nvidia	15	No	No	4632	0.8	0.899	0.0178
Nvidia	15	Yes	Yes	1245	0.8019	0.8718	0.0625
Nvidia	15	Yes	No	1245	0.8	0.899	0.0665
Nvidia	30	No	Yes	4632	0.8	0.899	0.0178
Nvidia	30	No	No	4632	0.8	0.899	0.0178
Nvidia	30	Yes	Yes	1245	0.8019	0.8718	0.0625
Nvidia	30	Yes	No	1245	0.8019	0.8718	0.0623

Table 1.

** We suspect that these entries may have been a clerical error due to the fact that they do not resemble any of our other results. **

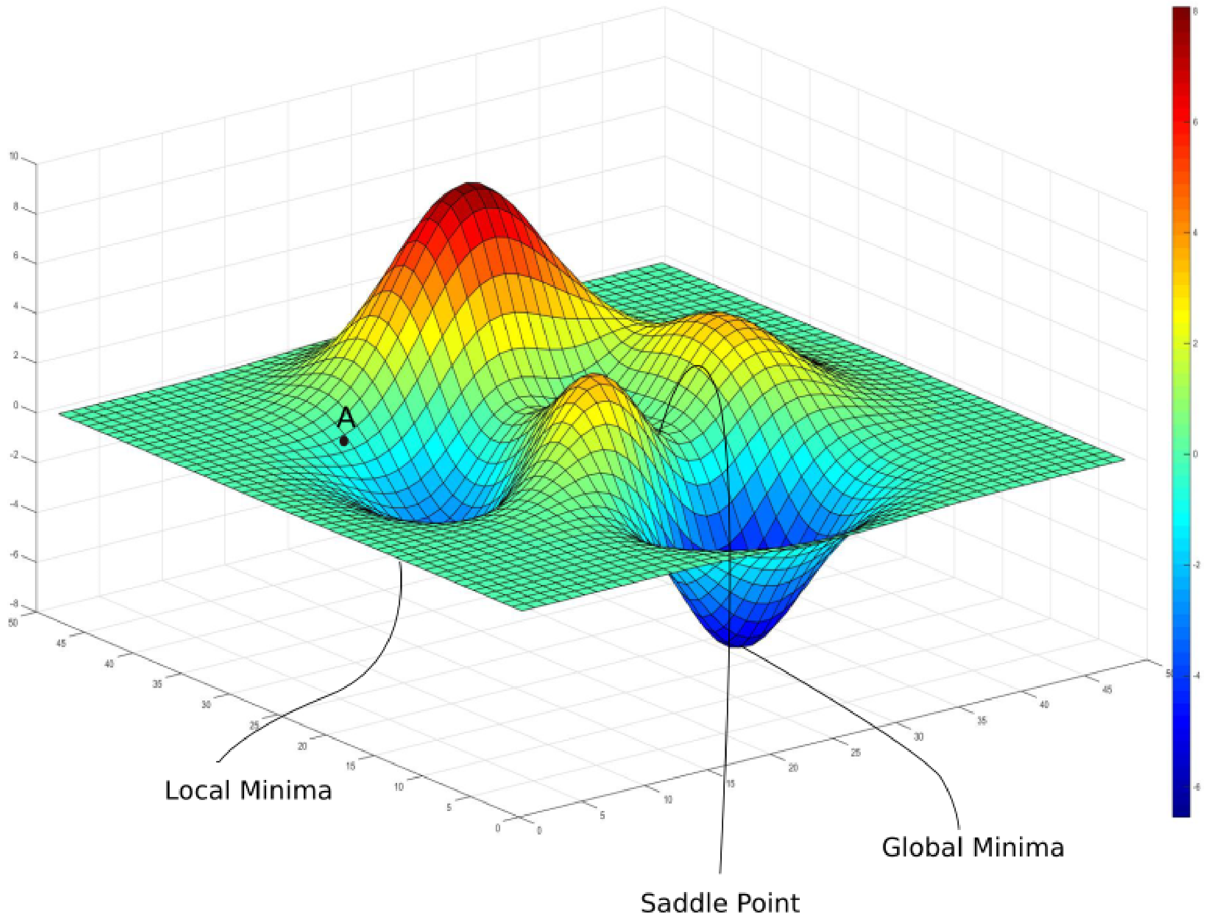
Table 1. above reports the accuracy measures for the Nvidia model given various specifications. Both of our accuracy measures change very little given our different specifications, interval accuracy ranges from 0.8 to 0.8019 (Omitting our one outlier), and directional accuracy ranges from 0.87 to 0.899. For interval accuracy the slight differences only appear in tests that made use of bucketing. However, the small variations in directional accuracy don't appear to be correlated with any one parameter. Loss also doesn't appear to change with the number of epochs, but rather with the number of available test instances.

Model	Epochs	Bucketing	Dropout	Training Instances	Interval Accuracy	Directional Accuracy	Loss
Nvidia	20	No	No	2384	0.774	0.717	0.0092
Nvidia	20	No	Yes	2384	0.79	0.81	0.0109
Nvidia	20	Yes	No	174	0.44	0.667	0.0766
Nvidia	20	Yes	Yes	174	0.44	0.722	0.0466
Nvidia	40	No	No	2384	0.82	0.977	0.0175
Nvidia	40	No	Yes	2384	0.815	0.8943	0.0043
Nvidia	40	Yes	Yes	174	0.541	0.75	0.0221

Table 2.

Table 2 describes the results of accuracy tests with a smaller training set from earlier on in the project.

These were not exactly the results we were expecting. As seen in Table 2, we have previously observed our neural network to continue increasing in prediction accuracy as the number of elapsed training epochs increased. The data in Table 1 suggests that our neural network reached minimum loss within the first ten training epochs. While this may seem implausible, we believe this may actually be the case. Our results suggest that we may have encountered a rare problem that can occur in convolutional neural nets: a local minimum of gradient descent loss. Like all gradient descent optimizers, the optimizer function we used in this project, Adam, can occasionally result in a local minimum of loss rather than the desired global minimum. This means that even after increasing the total number of training epochs, loss (and therefore accuracy) will no longer approach the most optimal point. (Kathuria) This would explain the results described in Table 1.



Visualization of a potential local minima of loss problem. (Source: Kathuria)

Despite these possible issues regarding the CNN and our optimizer our accuracies are still relatively high. This would suggest that an autonomous vehicle using our model would predict a steering angle within a tight window of the correct steering angle 80% of the time. It is difficult to quantify the meaning of this, because simply predicting the correct way to steering doesn't implicitly mean that the car would drive correctly, but the results are promising.

Model	**Epochs**	**Bucketing**	**Dropout**	**Training Instances**	**Interval Accuracy**	**Directional Accuracy**	**Loss**
Group	10	No	Yes	4632	0.0699	0.153	2.4
Group	10	No	No	4632	0.0699	0.151	0.985
Group	10	Yes	Yes	1245	0.0446	0.153	27.04
Group	10	Yes	No	1245	0.0136	0.1017	2.56
Group	15	No	Yes	4632	0.0737	0.163	1.125
Group	15	No	No	4632	0.099	0.157	1.66

Group	15	Yes	Yes	1245	0.0	0.101	12.405
Group	15	Yes	No	1245	0.0543	0.165	2.8534
Group	30	No	Yes	4632	0.0	0.128	38.09
Group	30	No	No	4632	0.0504	0.1359	1.2331
Group	30	Yes	Yes	1245	0.2194	0.2174	0.0789
Group	30	Yes	No	1245	0.1747	0.1902	0.195

Table 3.

In comparison to the Nvidia model, our group's implementation fails spectacularly. Interval accuracy ranges from 0% to a paltry 21.94% and direction accuracy isn't all that much more successful ranging from 10.1% to 21.74%. When using bucketing the interval accuracies appear to roughly increase with the number of epochs, but non-bucketed data don't follow any pattern with regard to epochs. Interval accuracy for those tests that use dropout also doesn't seem to have any association with the number of training epochs. Looking at directional accuracy none appear to be correlated with the number of epochs ran. And counter-intuitively loss appears to be mostly random.

Similarly to Table 1, the results described by Table 3 were somewhat nonuniform. However, we did observe differences in the displayed accuracy and loss for different numbers of epochs, so there did not appear to the issue of another case of local minimum loss. In this instance, the variability in the results was likely attributable to the overall unreliability of this model. As we can see from the low accuracies, this model was not particularly good at predicting direction, even after being given an ample number of training epochs.

Overall, it is clear that the Nvidia model vastly outperforms our group's implementation. The Nvidia model is statistically significantly better at every level for every specification. The 95% confidence intervals for the best interval and directional accuracy of our groups model were [0.1964, 0.2424] and [0.1945, 0.24], respectively. Whereas the 95% confidence intervals for the Nvidia model for the same specification were [0.7798, 0.824] and [0.853, 0.89], which perhaps obviously do not contain any part of the intervals calculated for the group implementation.

Conclusion

In conclusion, organization of the data and which random seed a test is run on make an impactful difference in the reported accuracies, as shown by our preliminary and final results. It appears that for large enough datasets that bucketing our data results in increased accuracy. Although we have conflicting results, from our preliminary and final findings, regarding the improvement of accuracy with increased numbers of elapsed epochs, we feel comfortable concluding that as the number of training epochs is increased the accuracy will increase to a point before plateauing; and that the lack of improvement seen in the final results is down to the chosen seed. Additionally, it appears as though when using CNNs for autonomous driving problems, relatively few epochs are needed to reach this plateau, we found that our models accuracy capped out somewhere between 20-40 epochs. And finally using our implementation of the Nvidia model we were able to achieve high accuracies of 80% and 89.9%, for interval and directional, which implies that the model could hypothetically drive a vehicle.

Future Work

The next step for this project would be to get the autonomous mode in Udacity up and running. It was one of our primary goals coming into the project to have this working, but after a decent amount of time was sunk into the issue we finally gave up, to focus on writing. Unfortunately, without this implemented our project lacks tangible meaning, because accuracy scores can only convey some information about the effectiveness of a model. Given more time in the future we would also like to examine more autonomous driving algorithms such as ALVINN and others. We would also train the car on the more complex track, because it would be more analogous to driving in the real world. In our research for this final paper we came across a number of groups who chose to crop their image inputs to include on the pixels of the road immediately in front of the car, this and including left and right images are other steps to take in the future. The last direction to take this project would be to expand to other problem within autonomous driving, such as sign or pedestrian recognition.

References

Bertoncello, Michele. “Ten Ways Autonomous Driving Could Redefine the Automotive World.” McKinsey & Company, www.mckinsey.com/industries/automotive-and-assembly/our-insights/ten-ways-autonomous-driving-could-redefine-the-automotive-world.

“Motor Vehicle Crash Deaths.” Centers for Disease Control and Prevention, Centers for Disease Control and Prevention, 6 July 2016, www.cdc.gov/vitalsigns/motor-vehicle-safety/index.html.

Matthew.lynberg.ctr@dot.gov. “Automated Vehicles for Safety.” NHTSA, 12 Aug. 2019, www.nhtsa.gov/technology-innovation/automated-vehicles-safety.

“Sources of Greenhouse Gas Emissions.” EPA, Environmental Protection Agency, 13 Sept. 2019, www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions.

Garsten, Ed. “Sharp Growth In Autonomous Car Market Value Predicted But May Be Stalled By Rise In Consumer Fear.” *Forbes*, 13 Aug. 2018, https://www.forbes.com/sites/edgarsten/2018/08/13/sharp-growth-in-autonomous-car-market-value-predicted-but-may-be-stalled-by-rise-in-consumer-fear/#153d02d3617c

Gupta, Anil. “Machine Learning Algorithms in Autonomous Driving.” Create a Culture of Innovation with IIoT World!, IIOT World, 15 Mar. 2018, iiot-world.com/machine-learning/machine-learning-algorithms-in-autonomous-driving/

Coren, Michael J. “All the Things That Still Baffle Self-Driving Cars, Starting with Seagulls.” Quartz, Quartz, 23 Sept. 2018, qz.com/1397504/all-the-things-that-still-baffle-self-driving-cars-starting-with-seagulls/

“How Do Self-Driving Cars Work?: The Zebra.” Compare Car Insurance Rates and Get Free Quotes | The Zebra, Insurance Zebra, 5 Aug. 2019, www.thezebra.com/how-do-self-driving-cars-work/

Udacity. “Udacity/Self-Driving-Car-Sim.” GitHub, 11 Jan. 2019, github.com/udacity/self-driving-car-sim.

Bojarski, Mariusz. “End-to-End Deep Learning for Self-Driving Cars.” *NVIDIA Developer Blog*, 25 Apr. 2018, devblogs.nvidia.com/deep-learning-self-driving-cars/.

Saha, Sumit. “A Comprehensive Guide to Convolutional Neural Networks-the ELI5 Way.” *Medium*, Towards Data Science, 17 Dec. 2018, towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

“Keras: The Python Deep Learning Library.” Home - Keras Documentation, keras.io/.

‘Adam: A Method for Stochastic Optimization’ Diederik P. Kingma, Jimmy Ba

Kathuria, Ayoosh. “Intro to Optimization in Deep Learning: Gradient Descent.” Paperspace Blog, Paperspace Blog, 2 Dec. 2019, blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/.

Pomerleau Dean A. “ALVINN: AN AUTONOMOUS LAND VEHICLE IN A NEURAL NETWORK”, Computer Science Department Carnegie Mellon University Pittsburgh, PA 15213 https://papers.nips.cc/paper/95-alvinn-an-autonomous-land-vehicle-in-a-neural-network.pdf

Kocić, Jelena, et al. “An End-to-End Deep Neural Network for Autonomous Driving Designed for Embedded Automotive Platforms.” *Sensors (Basel, Switzerland)*, MDPI, 3 May 2019, www.ncbi.nlm.nih.gov/pmc/articles/PMC6539483/.