Self-Driving Car Steering Angle Prediction Using Deep Learning Methods

Project Final Report

Akhil Devarakonda ADEVARA5@SEAS.UPENN.EDU
Zachary Fisher ZJF219@SEAS.UPENN.EDU
Anthony Nguyen TONYN21@SEAS.UPENN.EDU

1. Introduction

Autonomous driving algorithms are a much discussed and active area of research at the cross-section of computer vision and machine learning. Successful autonomous driving using cheap hardware (up to three RGB cameras in this case) is of significant advantage for the commercialization of self-driving vehicles. Our project aims to explore this area by using labeled dash-cam footage in order to generate accurate steering angle predictions using various deep learning techniques. For the first half of the project, we used a dataset released by Udacity that embodies challenging and diverse real-world driving scenarios (Udacity, 2016b). For the second half of the project, we explored how a neural network architecture used successfully on real-world data would transfer to simulation data. We trained and evaluated various models from training data collected in Udacity's self-driving car simulator and applied them to control steering angle of the simulated autonomous vehicle.

2. Related Work

Large public datasets and community challenges have been driven autonomous driving forward. Datasets vary from focusing on highways to urban environments, each with varying levels of difficulty. While many large scale projects incorporate radar, lidar, and a suite of cameras, we found that we are most interested in solving this task with the fewest resources.

2.1. NVIDIA

NVIDIA used a CNN architecture with five convolutional layers and three fully connected layers to accomplish steering angle predictions without the need for manual decomposing the images into their key features. The company found that less than a hundred hours of driving data from three cameras was sufficient to develop a model that operated in diverse settings including highways, residential roads, and various weather conditions. (Mariusz Bojarski & Testa, 2016)

2.2. OpenPilot

Comma.ai offers an autonomous driving system called OpenPilot that relies on a single outward facing camera and open-source software to modify a vehicle for lane-keeping and adaptive cruise control. They have over 30 hours of highway driving video publicly available labeled with different forms of sensor data. In addition to using a deep CNN, OpenPilot combines a Variational Auto-encoder and a Generative Adversarial Network to predict realistic future frames. (Schafer et al., 2018)

3. Datasets

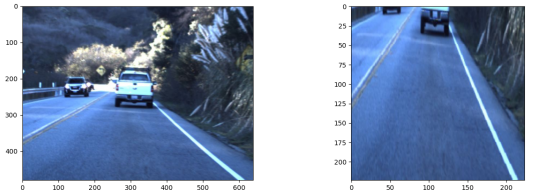
The dataset we used for the first half of the project comes from Udacity and Lyft's Perception Challenge, which can be found here. The training data contains 101,398 images that were recorded using three RGB cameras (labeled left, center, and right) mounted on the vehicle and were time-stamped with the vehicle's steering wheel angle, motor torque, GPS coordinates, and various sensor data. The test set contained 5614 images labeled with the steering angle. It is important to note that the training and test data consisted of images primarily in 'sunny' or 'overcast' lighting conditions. The steering angle label is represented as $\frac{1}{r}$ where r is the turning radius of the predicted path.

The second half of the project used training data collected from Udacity's self-driving car simulator (Udacity, 2016a). The simulator came loaded with two tracks. Track 1 is a relative simple track with mostly simple curves and no hills. Track 2 has a lot more challenging curves and is hilly. Images of these tracks are show in Figure 2. Unfortunately, for unknown reasons, the simulator did not allow us to control the throttle during training data collection; we were forced to collect the steering angle data at full throttle. We drove around the tracks and collected a total of 126,712 images, and the corresponding steering angles, for each track.

4. Methodology

4.1. Data Pre-Processing

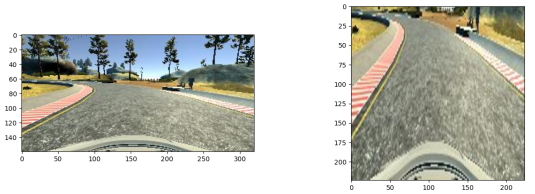
For the first half of the project, we used a dataset supplied by Udacity which contained real-world images. We reduced the training set by a third by only considering center camera images. The images were originally of size 480 x 640, but to reduce training time we cropped the part of the images above the horizon line so the model is invariant to scene changes. Finally, we resized the resulting images appropriately, depending on the size of the input the particular model required. The real-world image processing results are shown in Figure 1.



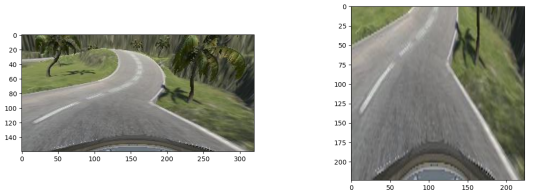
(a) Raw Image (b) Processed Image

Figure 1. Real-World Image Pre-Processing

For the second half of the project, we used the self-driving car simulator provided by Udacity. The simulator came loaded with two tracks and contained "training" and "autonomous" modes. The simulator collected the images and labels for the training data as we drove around the tracks. While a few data pre-processing methods were tried, the optimal method we found is described here. The images collected were originally sized 320 x 160 and were cropped to 320 x 100, then resized to 224 x 224 as inputs to ResNet-18. The simulator image pre-processing results are shown in Figure 2.



(a) Track 1 Raw Image (b) Track 1 Processed Image



(c) Track 2 Raw Image (d) Track 2 Processed Image

Figure 2. Simulator Track Images Pre-Processing

4.2. CNN Architecture

We determined that CNNs would be an ideal approach to this problem due to their primacy in analyzing imagery. To get a general idea of where our future work should concentrate, we implemented an assortment of pre-trained and custom CNN models. The pre-trained models included AlexNet and ResNet. While both the pre-trained models had relatively deep networks, the custom CNN consisted of two convolutional and three fully connected layers. Ultimately, the ResNet-18 model was adopted and its architecture is shown below in Figure 3. ResNet-18 is a residual network that contains 17 convolutional layers and one fully connected layer; additionally, the model employs techniques such as stride and average pooling.

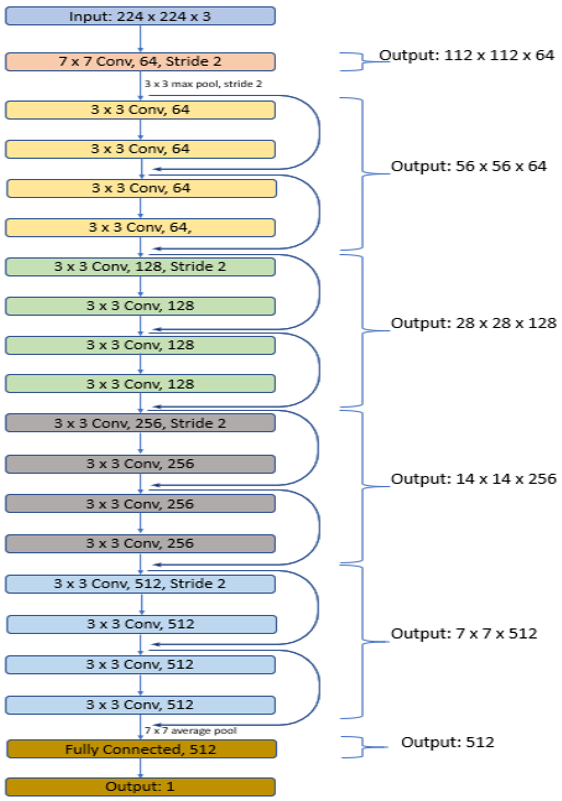


Figure 3. ResNet-18 CNN with custom FC layer

4.3. Loss Function and Hyperparameters

The original Udacity challenge for the real-world dataset used RMSE (shown in Equation 1) as a measure of model performance. As a result, their leaderboard was ranked according to a team's RMSE on the test set. To maintain consistency, we also used RMSE. In any case, since this is a regression problem and the steering angle is a continuous variable, RMSE is a good choice for fitting our model. For this reason, we also used RMSE for the

simulator datasets.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n}} \quad (1)$$

While training the various models over the real-world data, we maintained a consistent learning rate of 0.01 and used the Adam optimizer. We used an 80/20 split for the training and validation set. We also utilized a SGD batch size of 16 and trained over 8 epochs. Once the model with the lowest test loss was discovered, we froze the hyperparameters and used them on the simulator datasets.

5. Experiments and Results

5.1. Choosing a Model Architecture

First, we evaluated several neural network architectures using the real-world dataset to determine which one would result in the lowest RMSE on the test set. The networks we trained are listed in Table 1. All models were trained using identical parameters and pre-processing on the images as discussed in section 4.1. The data was shuffled before training and split into training/validation set ratio of 80/20. The Adam optimizer was used along with the RMSE loss function. The top half of the images were cropped so that only portion of the image with the road remained. We choose to crop the images in this way so that our models can be more robust to scene changes. However, this still leaves our models prone to changes in weather, season, etc.

Model	Training RMSE	Validation RMSE	Test RMSE
Pretrained AlexNet	0.207	0.257	0.198
Pretrained ResNet	0.159	0.171	0.093
Custom AlexNet	0.248	0.260	0.259
Custom ResNet-18	0.106	0.115	0.065
Custom Model	0.047	0.065	0.126

Table 1. Neural Networks Evaluation

As can be seen from Table 1, the modified ResNet-18 model had the lowest test RMSE. We customized ResNet-18 with a fully connected layer of 512 nodes; the network had approximately 11 million parameters to learn. If we had competed in the Udacity competition, our model would have placed us in sixth place on the final leaderboard. While we were pleased with the results, we were skeptical about how well our model would perform on a real car in the real world.

5.2. Deploying ResNet-18 on Real-World Test Set

To better understand our model performance visually, we overlaid the predicted and ground truth steering angles over the test images and generated a video. A couple relevant frames from the video are shown below in Figure 4.

(a) (b) (c)

Figure 4. Test Set Video

On average the model appeared to perform well. Unsurprisingly however, there were moments like the images shown in Figure 4 where the deviation between the ground truth and predicted value was so drastic that if the model was deployed on a real car the results would have been catastrophic. The prediction in Figure 4a may have been improved with more training data in low-light areas or other types of image pre-processing. Although the prediction errors shown in Figure 4b and Figure 4c were short-lived, in the real world they may have caused the car to over-correct itself and lose control.

Even though the video is a useful visual tool to evaluate our model's performance, it is difficult to assess how the model would have performed on a real car. One significant performance metric that is unclear is how robust the model is to prediction error, i.e. if the car can correct itself with the accumulation of steering error. This motivated us to search for ways to better evaluate our model performance and we decided to use Udacity's self-driving car simulator for the second part of our project.

5.3. Training Simulator Dataset using ResNet-18

After collecting training data from track 1, we trained various models using the ResNet-18 architecture. We kept the model architecture the same and only tweaked data pre-processing. The results are summarized in Table 2.

Model #	Distinction	Training RMSE	Validation RMSE
1	Baseline	0.033	0.034
2	1/2 Image Vertical Crop	0.035	0.036
3	1/3 Image Vertical Crop	0.034	0.036
4	Canny Edge Detection	0.031	0.038
5	Image Normalization	0.033	0.036
6	1/3 crop, Image Flipping	0.035	0.035
7	Track 2, 1/3 crop, Image Flipping	0.170	0.176
8	Track 2, 1/3 crop, Image Flipping and Normalization	0.181	0.185

Table 2. Model Evaluations

For model 1, the baseline model, we only used the center image with resizing for ResNet-18, a learning rate 0f 0.01, the Adam optimizer, an 80/20 split for training and validation set, an SGD batch size of 16 and trained over 8 epochs. The subsequent models had differences shown in the "Distinction" column of Table 2. Model 2 was trained on an image whose height was cropped in half. We found that this was removing part of the track

near the horizon and possibly contributing to poor performance so we reduced the vertical crop to 1/3 in model 3. Model 4 was trained using images who went through a canny edge detector so that the network only saw the edges in the image returned from the algorithm. Model 5 was trained using images that were normalized according to their mean and standard deviation. Model 6 was trained with a 1/3 image crop and random image flipping. Model 7 is exactly like model 6 except it was trained on the track 2 dataset, which is why the RMSE is drastically different than the other models. Track 2 is significantly more challenging than track 1 so we wanted to see how a model trained on it would perform on track 1. Finally, model 8 was trained on track 2 with 1/3 vertical crop and image flipping and normalization.

5.4. Deploying Trained ResNet-18 in Simulator

The simulator provided by Udacity did not come with separate test tracks. To prevent testing on the same track we trained, we simply tested the models with the car driving in the opposite direction on the track. This provided the model with images it had not seen in training. We had two metrics for model success: one lap completion around the simulator's track 1 and the speed at which the car completes the track. Figure 5 depicts a plot with all

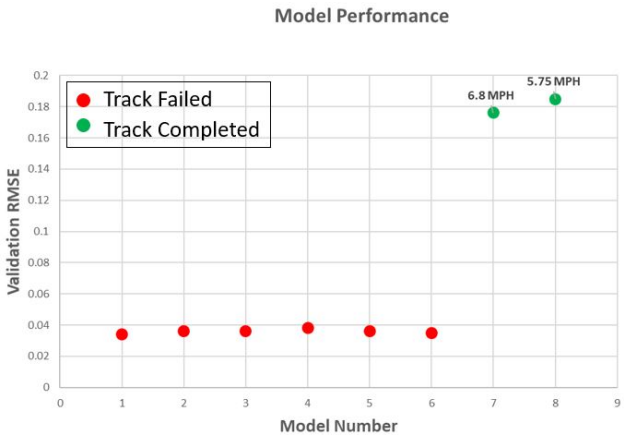


Figure 5. Model Performances

the model performances. The model's validation RMSE is shown on the y-axis and models marked as red were unable to complete one lap around track 1 while models marked green were able to complete one lap. The models that were successful are shown with the max speed at which they were able to complete the lap.

6. Discussion

In the first half of the project, we found that the ResNet-18 architecture gave us the lowest RMSE on the real-world test set and a good position on Udacity's leaderboard for the competition, which led us to adopt that architecture for the second half of the project.

The results of the tests conducted on the simulator underscore the importance of good training data and the difficulties of deploying models in the "real world". As can be seen from Figure 5, none of the models trained on track 1 were able to complete track 1 in the opposite direction (although they had no problem completing the track in the direction the training data was collected). Only the models trained on track 2 were able to complete track 1 in the opposite direction, with model 7 completing the track at an average speed of 6.8 mph. Upon inspection it became clear that models 1-6 all failed at a particularly challenging curve in track 1. Since model 7 and 8 were trained on track 2, which had a lot more curves with tighter radii, these models had no problem rounding the challenging curve in track 1. We believe if we had collected more training data at this curve, models 1-6 might have been able to complete the entire track. Additionally, we did not test models 7 and 8 on track 2 since this track would definitely have required a good model trained to predict the throttle, which was not possible due to the full throttle bug in the simulator when collecting training data. In retrospect, it would have been prudent to further explore other network architectures and perhaps build a network more tailored to this particular problem. It is possible that the deep ResNet-18 architecture may not be best suited for real time steering angle prediction due to its depth and complexity.

7. Conclusion

In this project we began by evaluating various neural network architectures to predict the steering angle of a car using a real-world dataset. We found that a custom ResNet-18 architecture had the lowest test RMSE out of all the architectures studied and our score would have placed us in sixth place on the original Udacity leaderboard. We wanted to see how this architecture would behave on a car so we used Udacity's car simulator to test it. Various pre-processing methods were applied to generate eight models using the ResNet-18 architecture. Although we were able to successfully drive the car, we found the ResNet-18 architecture may not be best suited for real time steering angle prediction due to its complexity. In future work, we would like to build models for throttle prediction to fully drive the car autonomously. Additionally, we want to use more complex self-driving car simulators that have multiple lanes, other cars and are generally more realistic. We would like to expand this work to autonomously drive a car in more challenging simulators, exploring more sophisticated techniques such as LSTM networks and reinforcement learning.

References

Mariusz Bojarski, Ben Firner, Beat Flepp Larry Jackel Urs Muller Karol Zieba and Testa, Davide Del. End-to-end deep learning for self-driving cars. 2016. https://devblogs.nvidia.com/deep-learning-self-driving-cars/.

Schafer, Harald, Santana, Eder, Haden, Andrew, and Biasini, Riccardo. A commute in data: The comma2k19 dataset. 2018. https://arxiv.org/pdf/1812.05752v1.pdf.

Udacity. Self-driving car simulator. 2016a. https://github.com/udacity/self-driving-car-sim.

Udacity. Car steering dataset. 2016b. https://github.com/udacity/self-driving-car.

Self-Driving Car Steering Angle Predictions Using Deep Learning Methods
Phase 1

Motivation: What CNN architecture and training parameters will best predict steering angles using challenging real-world dash cam video from a single RGB camera?

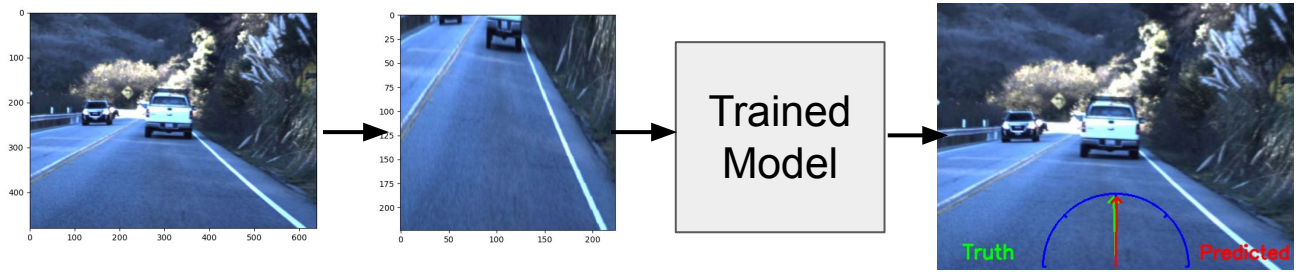
Experiments

- The following models were implemented, leading to the selection of the Custom ResNet-18 model that consisted of an additional fully connected layer at the output

Model	Training RMSE	Validation RMSE	Test RMSE
Pretrained AlexNet	0.207	0.257	0.198
Pretrained ResNet	0.159	0.171	0.093
Custom AlexNet	0.248	0.260	0.259
Custom ResNet-18	0.106	0.115	0.065
Custom Model	0.047	0.065	0.126

Results

- Predictions were largely accurate but with a few drastic errors in difficult situations

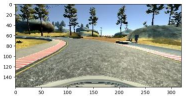


Self-Driving Car Steering Angle Predictions Using Deep Learning Methods Phase 2

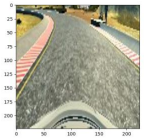
Motivation: How does the ResNet-18 architecture, which resulted in low RMSE on real-world data, actually perform when driving a car in a simulation?

Experiments

- Various pre-processing techniques were tried on track 1 and track 2 datasets

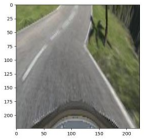


(a) Track 1 Raw Image



(b) Track 1 Processed Image

(c) Track 2 Raw Image

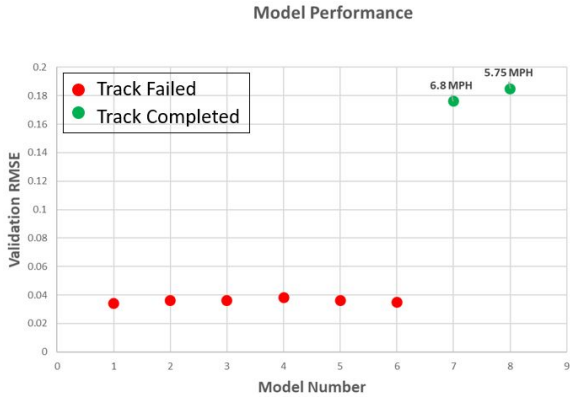


(d) Track 2 Processed Image

Model #	Distinction	Training RMSE	Validation RMSE
1	Baseline	0.033	0.034
2	1/2 Image Vertical Crop	0.035	0.036
3	1/3 Image Vertical Crop	0.034	0.036
4	Canny Edge Detection	0.031	0.038
5	Image Normalization	0.033	0.036
6	1/3 crop, Image Flipping	0.035	0.035
7	Track 2, 1/3 crop, Image Flipping	0.170	0.176
8	Track 2, 1/3 crop, Image Flipping and Normalization	0.181	0.185

Results

- Models trained on track 2, which is more challenging, were able to complete track 1.
- Model 7 performed the best, with an average speed of 6.8 mph.



- ResNet-18 may not be best suited for real time steering angle prediction due to depth and complexity.