Autonomous Steering for Vehicles using Convolutional Neural Networks

In this work, we investigate two simple and lightweight networks for autonomous steering, which, though more computationally efficient, are on par with contemporary approaches when operating in the environment provided. Additionally, we develop an easy to use ecosystem for deploying autonomous vehicles. This ecosystem can be easily extended in order to deal with increasingly difficult problems for autonomous vehicles.

I. Introduction

The primary motivation for this work is to develop a simple ecosystem for deploying a self-driving toy vehicle. An accompanying intuition is that a vehicle operating in a straightforward environment could ultimately provide a foundation upon which to better understand the problem domain as well as proposed solutions. Moreover, in the long-term, it is of interest to see whether there could be any advantages to gradually scaling-up the problem explored in our experiments. For example, whether one could re-use information learned on simpler versions of the problem, whether real, simulated, or both, via domain adaptation [10, 14].

The problem that we seek to address in our experiments is a supervised regression problem, wherein a Convolutional Neural Network (CNN), using images collected from the vehicle's front-facing camera, learns to predict steering angles adequate for keeping the vehicle within its lane as it drives at a fixed speed. Importantly, the solution to this problem is constrained to be end-to-end; that is, images are fed directly into the network after going through some fixed preprocessing pipeline, meaning that the training algorithm has no application-specific knowledge available to it. The reason for this constraint is that, in attempting to provide the network with application-specific knowledge, one might negatively impact what that network can and will learn; particularly because it is difficult for humans to account for all the nuances of driving. By allowing the training algorithm to decide what is and is not relevant, these nuances can be extracted automatically.

The key contributions of this work are its detailed explanation of a minimalistic yet functional ecosystem for developing self-driving vehicles, and its exploration of two simple architectures for the purposes of autonomous steering, which are proven to be on par with other work in the field.

The remainder of this paper will first give an overview of related work in the field of computer vision. Second, it will provide a highly detailed explanation of the various experiments performed, including a look at the dataset, an explanation of the hardware and software components involved our ecosystem, and an overview of the various network architectures, followed by our results and discussion.

II. Related Work

i. Convolutional Neural Networks

The popularity of CNNs has grown rapidly in recent years due to a convergence of three key factors. First, the emergence of large-scale datasets, such as ImageNet [9]. In the real world, objects tend to have a significant degree of variability; thus, to develop models capable of recognizing these objects, one must expose them to a similar degree of variability. Second, the growing power of computational hardware, namely GPUs, has allowed for these models to be trained within a reasonable time frame. Third, the introduction of AlexNet [5], which heralded a new dawn for CNNs in the field of Computer Vision, by winning the 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). Their network used only five convolutional layers followed by three fully-connected layers, which by modern standards is quite small. Moreover, they utilized comparatively large convolutions, such as 11x11, which have fallen out of favour, particularly after the emergence of networks like VGG [11], the winner of 2014 ILSVRC, which utilized many layers of simple 3x3 convolutions.

ii. Self-Driving Cars

The idea to use neural networks to operate autonomous vehicles is not novel, having been explored by a number of works, dating back as far as the late-1980s. [8], one of the earliest of these works, used a very small multi-layered perceptron, which took both camera and rangefinder data as inputs, and outputted the turn curvature that the vehicle must follow to reach the road’s center. Similarly, some decades later, [7] developed a new system, the DARPA Autonomous Vehicle (DAVE), which used a small convolutional neural network for choosing the direction of the vehicle. This network uses only image data mapped to a YUV space as its inputs, and either ‘left’, ‘straight’, and ‘right’ as its output. More recently [1] extended the work of [7] by devising the DAVE-2 architecture, which uses the same inputs as DAVE, but outputs the steering angle of the vehicle.

III. Experiments

Our approach to autonomous steering is centered around the use of an end-to-end network trained to drive using images from a simple track that we constructed (**Figure 1**), and associated steering angles collected through remote operation of the vehicle.

We create two networks, A and B, for the task of autonomous steering, and benchmark their performance against that of our port of the DAVE-2 [1] system, as it is a well-established end-to-end CNN for this task. Specifically, we evaluate the root mean-squared loss value of the three networks across an assortment of varying hyperparameters. Moreover, we perform a preliminary evaluation of each model’s ability to drive on an unseen track by simply counting the number of mistakes made on five laps around the track clockwise and counterclockwise. Although this metric is rough, it helps to perform an important evaluation of our vehicle, as a loss value cannot convey the network’s actual behaviour at run-time.

All of our networks were produced in Python 3.6.4 using Keras 2.1.4 [2] with a TensorFlow 1.5.0 backend.

i. Dataset

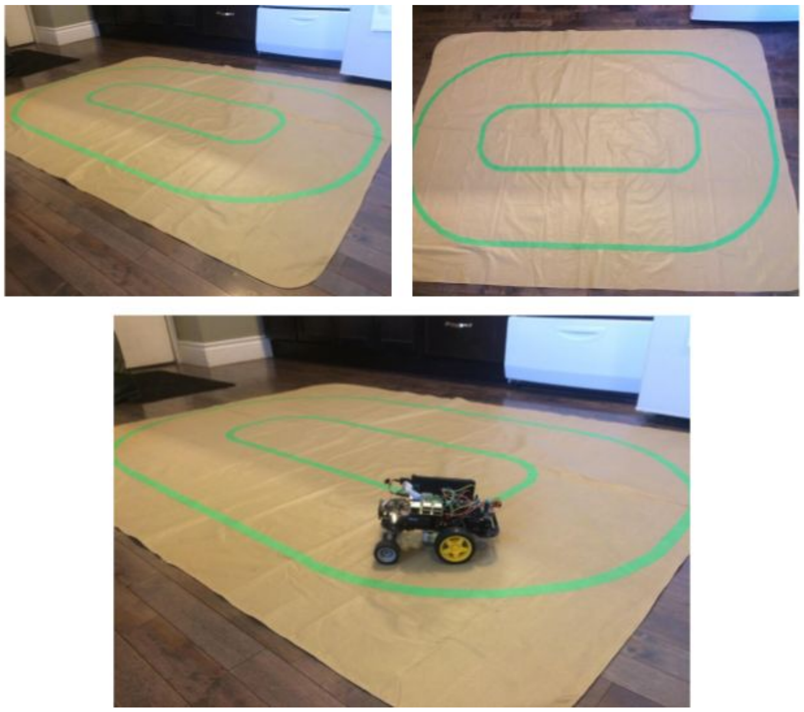

Figure 1 - an overhead view of Track #1.

The dataset consists of 2794 320x160 resolution RGB images, collected from twenty laps around Track #1; ten going clockwise and ten going counterclockwise. These images each have an associated steering angle between 40 and 140 degrees, and an associated speed between 0 and 255. As a simple preprocessing step, the steering angles are rounded to the nearest integer, and mapped to a range of -1 (140 degrees) to +1 (40 degrees). As well, the top half of each image is cropped out, as it shows primarily objects above the track, and is therefore not of use to the network.

A stratified 70/30 split is performed on the dataset in order to produce both a training and a validation set. **Figure 2** shows that the representation of each angle in the training set is very imbalanced. Thus, random upsampling and downsampling were employed in order to equalize the representation of each angle at 50 images per label. This is an important tactic, as training on such an imbalanced dataset can often produce very poor results, as it becomes advantageous for the model to always guess labels with the highest representation.

By using a generator to feed data into our model one batch at a time, we were able to randomly apply two simple and independent augmentation strategies with a 50% chance for each image. The first of these strategies was brightness adjustment, wherein a factor between 0.25 and 1.75 was chosen at random, and applied to the image's pixel values within an HSV space. Since the images in the dataset were all collected within relatively similar lighting conditions, randomly adjustments to their brightnesses were done in order to help the model generalize. The second of these strategies was horizontal flipping, wherein the image was simply mirrored, and the angle x was adjusted by the term $140 - x + 40$. Ideally, in most scenarios, a left-hand turn around a bend should have the same magnitude as a right-hand turn around the opposite bend, but due to driver error in our experiments, this might not be the case. The intuition behind using horizontal flipping is that this kind of behaviour can be encouraged.

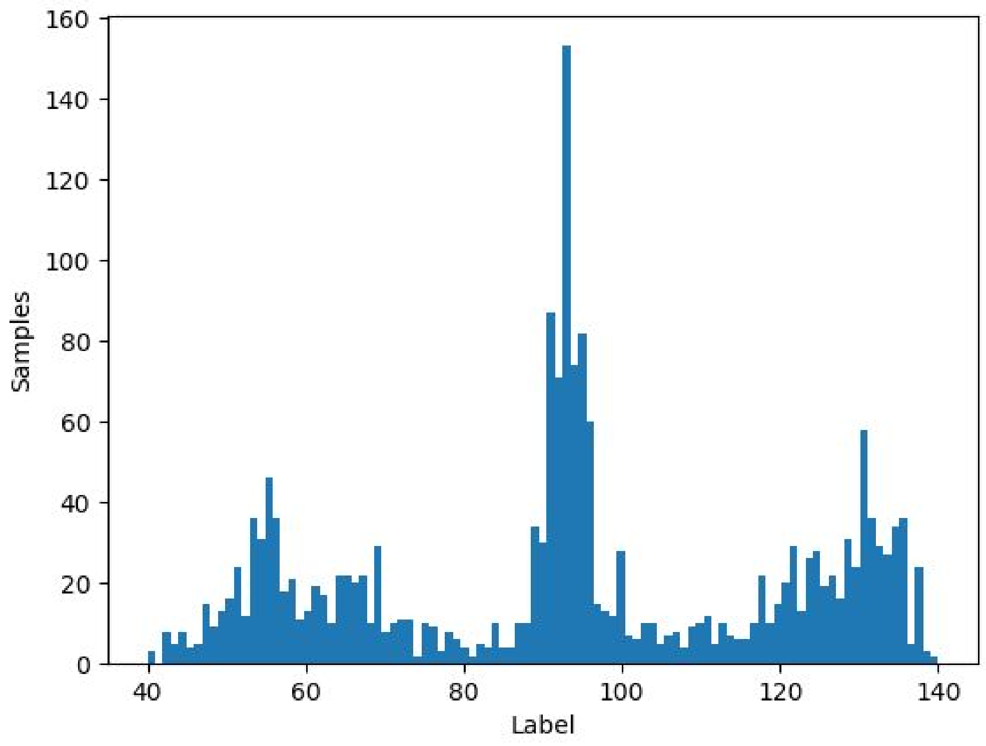

Figure 2 - the number of samples for each steering label in the training set.

ii. Hardware

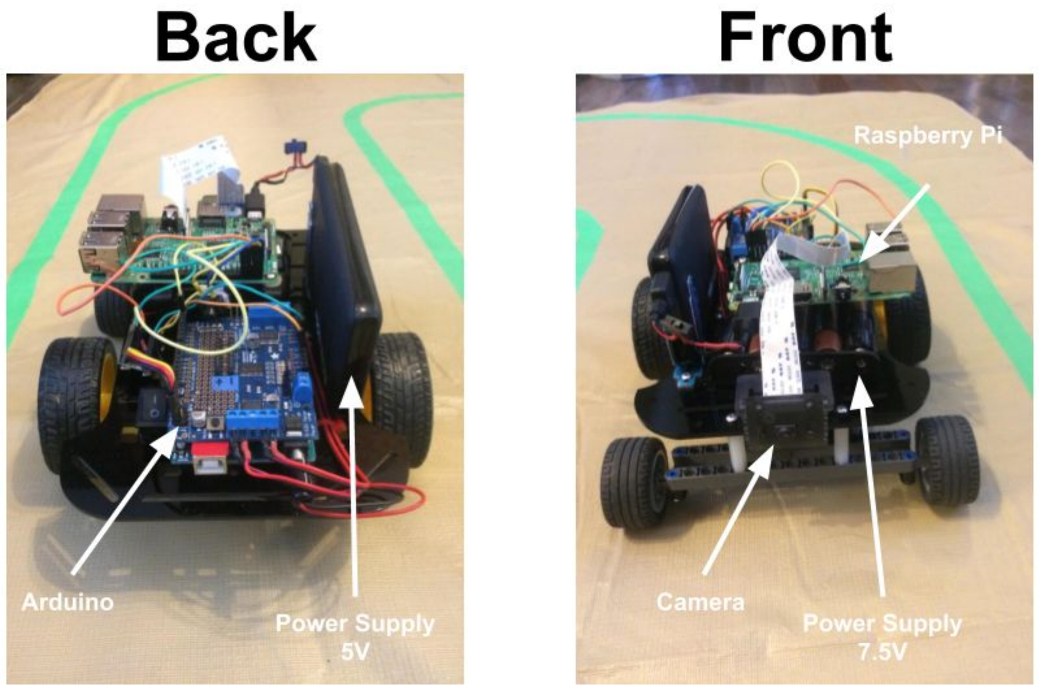

Figure 3 - a look at the vehicle.

Figure 4 shows a schematic of the vehicle's hardware components, of which the Raspberry Pi 3 Model B and Arduino Uno Rev3 are the most notable. Although interconnected, the role played by each of these devices is quite unique. The role of the Raspberry Pi is to send and receive communications to and from the other devices in the ecosystem, whereas the role of the Arduino is to process communications passed to it by the Raspberry Pi and convert them into commands for the vehicle's various motors.

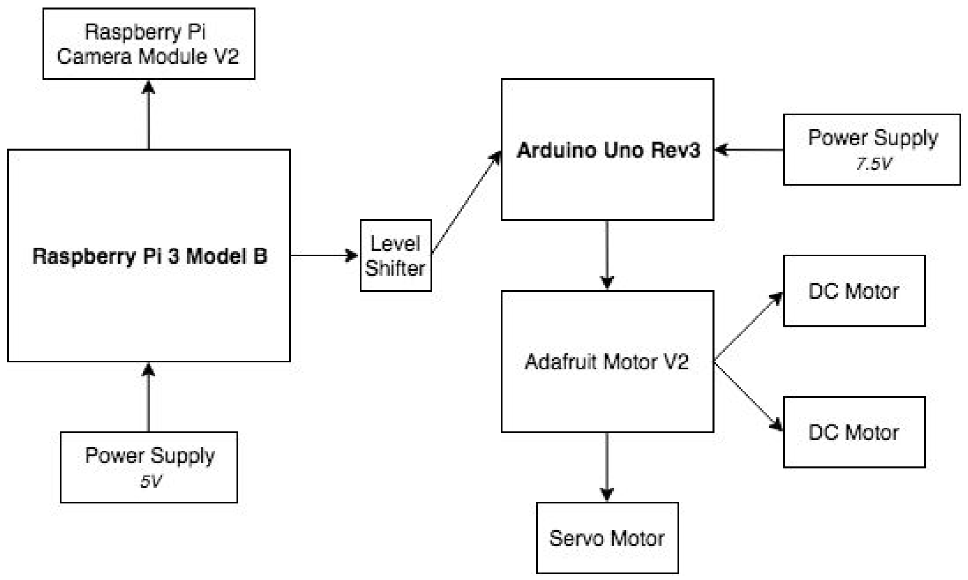

Figure 4 - a simplified schematic of the vehicle's hardware.

In general, the system has two modes of control: manual and automatic. In manual mode (**Figure 5**), the Camera Module V2 takes photographs at a rate of five frames per second. The reason why such a low frame rate was used is that, travelling at relatively slow speeds, there should not be large enough differences between frames to warrant anything higher. As the photographs are being taken, the Raspberry Pi listens for commands from the driver, which are sent over Bluetooth from a Sixaxis. This device's analogue toggles allow for steering and speed to be controlled independently, and with reasonable accuracy. As each photograph is taken, it is sent, along with the most recent steering and speed commands, to an external machine via WiFi. On the external machine, this data is simply amalgamated into a dataset for later use. Updates to the steering and speed commands are made only at the end of this process in order to avoid any mislabelling of data. Once these commands are updated, the Raspberry Pi will convert them to bytes and transmit them serially to the Arduino.

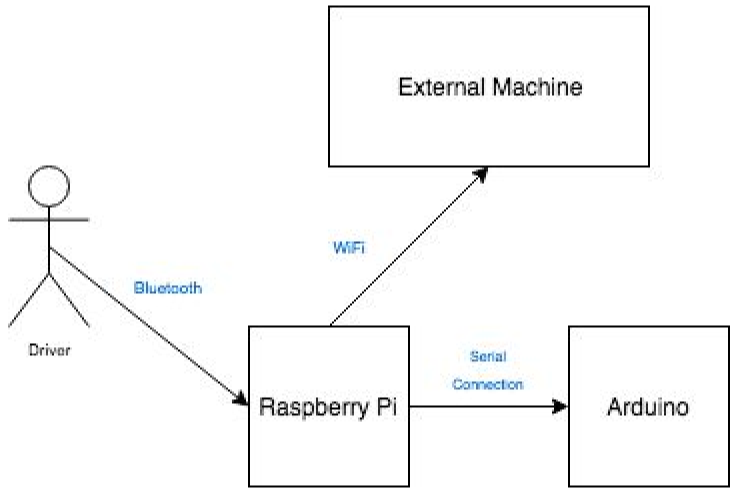

Figure 5 - the interconnections between devices operating in manual mode.

In automatic mode (**Figure 6**), the Camera Module V2 takes photographs at a rate of ten frames per second. The reason why a slightly higher frame rate was used in this scenario was so that the vehicle could react more quickly at run-time. As each photograph is taken, it is sent to an external machine, where it is fed into a model that predicts the steering command. This command is sent back to the Raspberry Pi, which converts it to bytes and transmits it serially to the Arduino.

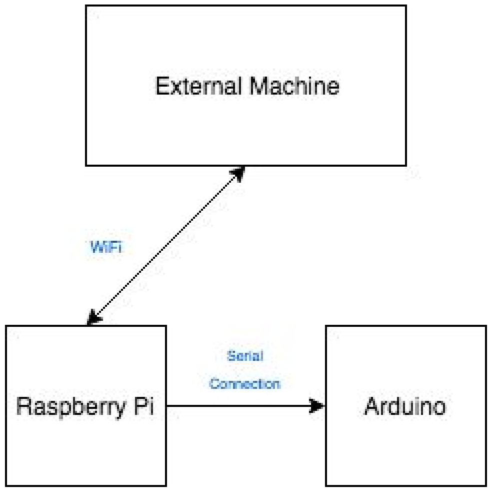

Figure 6 - the interconnections between devices operating in automatic mode.

iii. Models

In this section, we evaluate our networks vis-a-vis a benchmark network: DAVE-2. All the networks used apply the same normalization layer, which corresponds with the following function over each pixel in the input image i:

$$f(i) = i/127.5 - 1.0$$

Additionally, after each convolutional and fully-connected layer, the exponential linear unit (ELU) [3] function was applied with an α value of 1.0, where:

$$ELU(x) = \alpha * (exp(x) - 1) \text{ for x < 0}$$
and
$$ELU(x) = x \text{ for x >= 0}$$

ELU, like other members of the rectified linear family, helps to alleviate the vanishing gradient problem. Additionally, in-line with more recent additions to this family, like the leaky rectified linear unit, this function helps to combat the emergence of dying neurons, as in common with the vanilla rectified linear unit function [6]. ELU, however, provides one advantage that these other functions do not; that is, due to its allowance for saturating negative values, it tends to naturally push mean activation towards 0, which speeds up training, and eliminates the need to apply batch normalization [15] between layers.

Each of the networks below is made up of feature extracting convolutional layers followed by fully connected layers. The number and sizes of the fully-connected layers were preserved across networks in order to properly evaluate the quality of the representations produced by each of the feature extractors. Additionally, dropout (p=0.50) [13] was applied following the last convolutional layer in order to combat overfitting to the relatively small dataset.

Architecture

The architecture of the DAVE-2 (**Figure 7**) system was reconstructed from the information provided in [1]. The first fully-connected layer was removed in order to scale down the capacity of the network, as the problem being addressed here does not require that level of sophistication. Much of the data pre-processing done in [1] differs from that done in this work, and so, we focus more on comparing our implementation of the network, rather than the established performance of the DAVE-2 system.

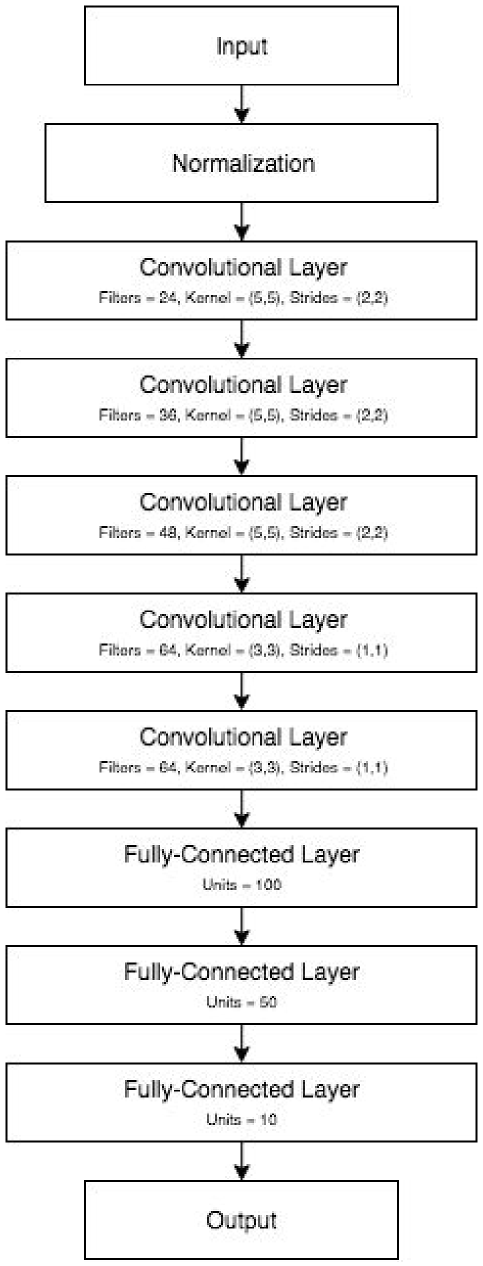
Input
Normalization
Convolutional Layer
Filters = 24, Kernel = (5,5), Strides = (2,2)
Convolutional Layer
Filters = 36, Kernel = (5,5), Strides = (2,2)
Convolutional Layer
Filters = 48, Kernel = (5,5), Strides = (2,2)
Convolutional Layer
Filters = 64, Kernel = (3,3), Strides = (1,1)
Convolutional Layer
Filters = 64, Kernel = (3,3), Strides = (1,1)
Fully-Connected Layer
Units = 100
Fully-Connected Layer
Units = 50
Fully-Connected Layer
Units = 10
Output

Figure 7 - architecture of the DAVE-2 model.

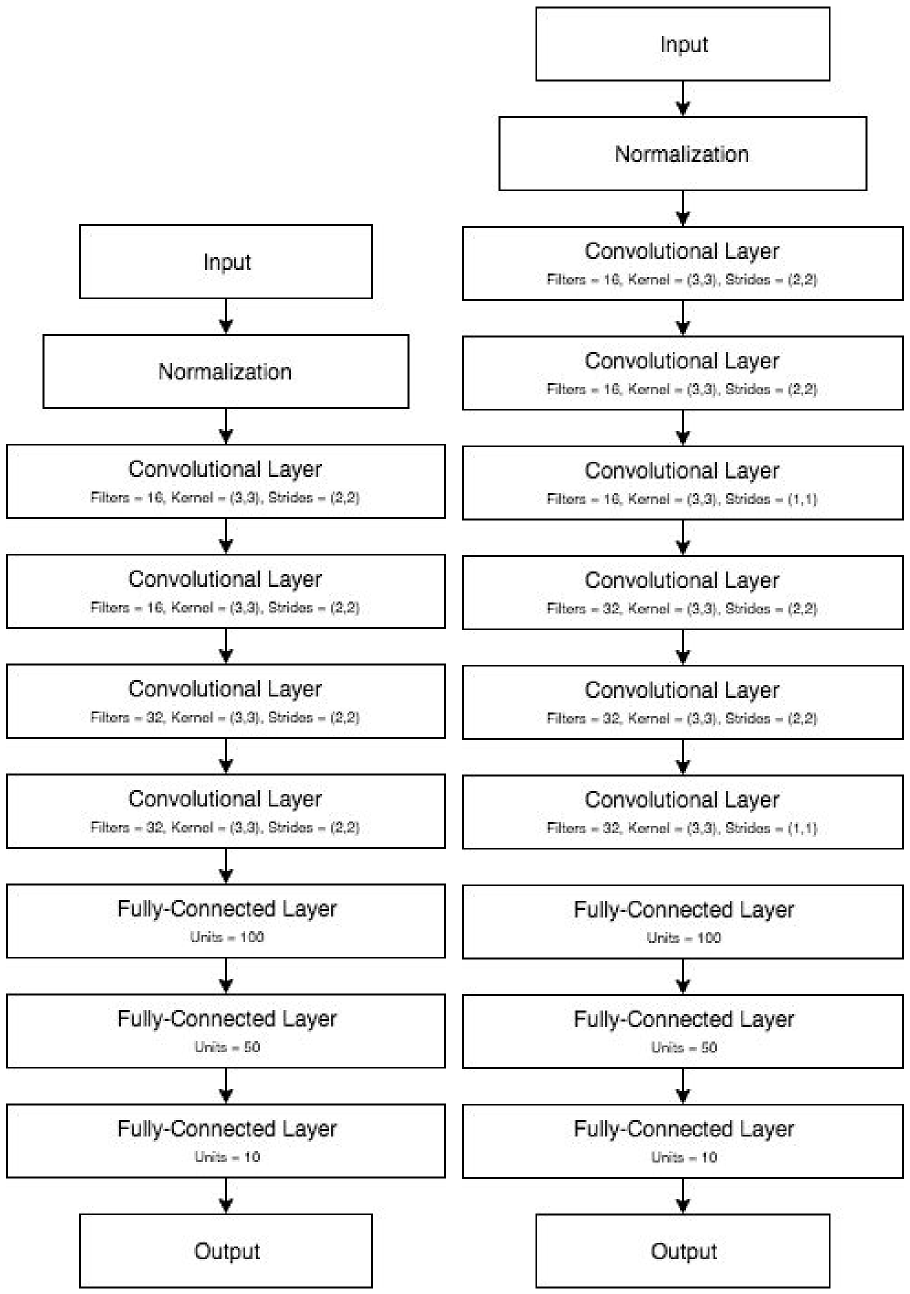
Input
Normalization
Convolutional Layer
Filters = 16, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 16, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 32, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 32, Kernel = (3,3), Strides = (2,2)
Fully-Connected Layer
Units = 100
Fully-Connected Layer
Units = 50
Fully-Connected Layer
Units = 10
Output
Input
Normalization
Convolutional Layer
Filters = 16, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 16, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 16, Kernel = (3,3), Strides = (1,1)
Convolutional Layer
Filters = 32, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 32, Kernel = (3,3), Strides = (2,2)
Convolutional Layer
Filters = 32, Kernel = (3,3), Strides = (1,1)
Fully-Connected Layer
Units = 100
Fully-Connected Layer
Units = 50
Fully-Connected Layer
Units = 10
Output

Figure 8 - architectures of the proposed networks.
(Left) Network A. (Right) Network B.

The architectures of networks A and B (**Figure 8**) draw inspiration from a variety of sources. As in [11], they adhere to a principle of constructing one's networks out of simple building blocks, and focusing primarily on the network's depth. To that end, they are both constructed using only 3x3 convolutions; though network A has a total of four convolutional layers, whereas network B has a total of six convolutional layers. One advantage to placing multiple small convolutional blocks in succession, rather than one larger convolutional block, is that one can incorporate more non-linearities into their network, which should help with discriminability. As well, assuming a fixed number of filters, the use of multiple smaller convolutions can have the same effect as one larger convolution, but require significantly fewer parameters, which can be treated as a form of regularization on the network [11]. In addition, as in [12], our networks attempt to reduce the number of components in their pipelines by replacing pooling layers with strided convolutions.

Training Regimen

For each architecture, a grid search was performed over various learning rates and batch sizes. Every model was given a maximum of 20 iterations for training, though early stopping would occur if a model's validation loss did not show signs of improvement for three iterations in a row. Adam [4] was chosen as the optimizer, along with the TensorFlow's default values of the hyperparameters β_1 , β_2 , and ε .

Loss

All the models were trained on a simple root mean-squared error (RMSE) loss function:

$$RMSE(\widehat{y}, y) = \sqrt{(\tfrac{1}{n}) * \sum_{i=1}^{n} (\widehat{y}_i - y_i)^2}$$

This function was chosen because it performs an intuitive evaluation of the network's performance; that is, it simply compares how close a predicted angle is to the desired angle.

iv. Results and Discussion

Tables 1, 2, and **3** present the validation losses for the three networks explored in this paper, and for each combination of hyperparameters. The best result from each network is highlight in green. In general, each model was able to converge to ≈ 0.10, which is equivalent to 5 degrees. Network A had the best overall performance of 0.09. Considering the computational efficiency of this network, the fact that it performs the best is surprising.

Table 1 - RMSE loss value from DAVE-2 network, given batch size (column-wise) and learning rate (row-wise). The best result is highlighted in green.

	2^0	2^1	2^2	2^3	2^4	2^5
10^{-1}	0.44	0.44	0.45	0.44	0.44	0.42
10^{-2}	0.44	0.45	0.45	0.44	0.44	0.43
10^{-3}	0.44	0.44	0.44	0.12	0.13	0.10

10^{-4}	0.10	0.10	0.10	0.10	0.12	0.10

Table 2 - RMSE loss value from network A, given batch size (column-wise) and learning rate (row-wise). The best result is highlighted in green.

	2^0	2^1	2^2	2^3	2^4	2^5
10^{-1}	0.44	0.44	0.44	0.44	0.44	0.43
10^{-2}	0.44	0.44	0.44	0.44	0.45	0.43
10^{-3}	0.11	0.10	0.10	0.11	0.11	0.12
10^{-4}	0.09	0.10	0.10	0.09	0.11	0.12

Table 3 - RMSE loss value from network B, given batch size (column-wise) and learning rate (row-wise). The best result is highlighted in green.

	2^0	2^1	2^2	2^3	2^4	2^5
10^{-1}	0.44	0.45	0.44	0.44	0.45	0.43
10^{-2}	0.44	0.44	0.44	0.44	0.44	0.43
10^{-3}	0.13	0.11	0.10	0.12	0.10	0.10
10^{-4}	0.10	0.10	0.10	0.10	NaN	0.13

For each of the networks, the best result was deployed, and the vehicle was tested on a previously unseen track (**Figure 9**). This track introduced new challenges to the vehicle, in that it introduces an unexpected bend, and also contracts for a period of time, shrinking the lane from ≈ 30.8 cm to ≈ 25.7 cm right before a typical bend. The vehicle was permitted to drive around this track ten times: five times clockwise, and five times counterclockwise. Everytime the vehicle drove over or on top of a line, it was counted as a mistake for the current network. Each network made a total of ten mistakes; each of which occurred at the bend where the lane contracts. We speculate that this issue is not that of any of the networks, but rather of the design. Since the vehicle has only one forward-facing camera, it is often unable to see the inner line of the track. Thus, we believe that when going around this bend, the vehicle defaults to the behaviours learned on Track #1, where it could afford to give ample space between itself and the outer line of the track. It should be noted that the mistakes were typically less severe when the vehicle was driving counterclockwise; that is, when the vehicle entered the bend directly from a wider portion of the track.

Without modifying any of the existing hardware, one way to improve performance vis-a-vis loss and deployment might be to train our networks on a dataset collected from a variety of tracks, each with its own unique challenges. Theoretically, by exposing the vehicle to these different challenges, one could help it to learn more general rules for driving. That being said, for the purposes of this work, the results achieved by these simple networks are very promising.

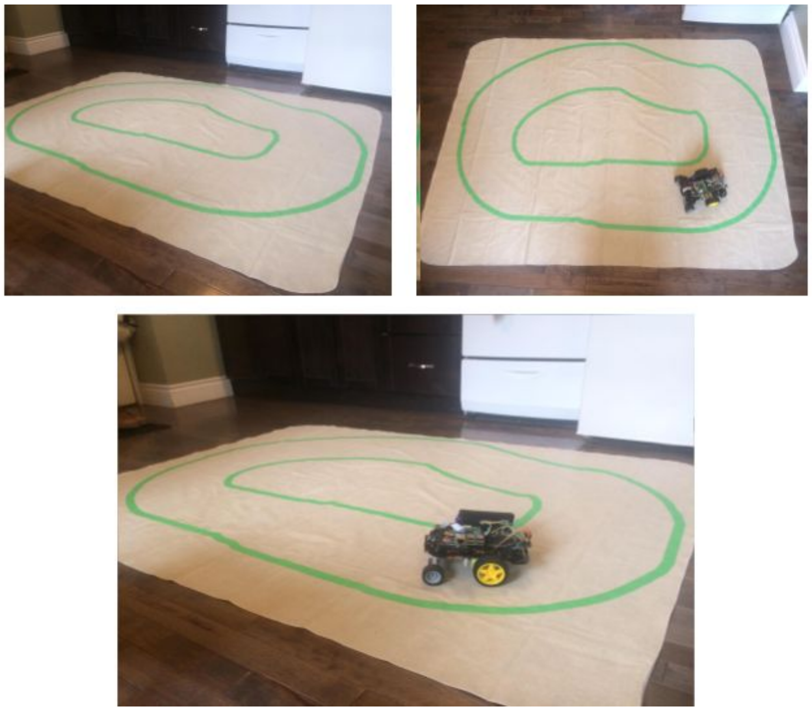

Figure 9 - an overhead view of Track #2

IV. Conclusion

In this work, we have benchmarked the performance of two simple networks for autonomous steering against the DAVE-2 network. Quantitatively, the validation loss for each of these three networks was proven to be relatively equal, despite reasonably large difference in network size. In fact, the smallest of the three networks, network A, was shown to perform slightly better than the other two. Qualitatively, there were no noticeable differences in performance, as all the networks were capable of driving the vehicle around a previously unseen track, each making only a small error when facing a challenge that the vehicle had not been exposed to during training. Furthermore, we developed a simple yet extensible ecosystem for deploying autonomous vehicles that will provide a good foundation for future work.

i. Future Work

One idea that has yet to be explored in the research is how to incorporate speed into a network for autonomous steering. In order to have speed simply be an output of the network, one could perform a separate regression atop the feature extracting layers. However, speed is often a function of situational information, such as the flow of traffic and the curvature of the road ahead, as well as procedural information, such as the speed limit. Thus, one could choose to treat speed as both an input and output of the network, potentially with feedback from the latter to the former. Since our ecosystem is already configured to collect both the steering angle and speed of the vehicle, this would be a simple extension of this preliminary work.

Another simple extension of this work would be to incorporate new sensory data, which would help the vehicle to operate in more complex environments. For example, by attaching additional cameras to the vehicle, one could capture information about its peripheries, which would allow for such applications as pedestrian tracking. Similarly, with an array of ultrasonic rangefinders, the vehicle could be taught to keep a minimum safe distance from other vehicles, while also having a failsafe in case the model failed.

References:

[1] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P.,... Zieba, K. (2016). End to End Learning for Self-Driving Cars. *arXiv preprint arXiv:1604.07316v1.*

[2] Chollet, Francois. (2015). Keras. URL: https://keras.io.

[3] Clevert, D., Unterthiner, T., & Hochreiter, S. (2015). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *Paper presented at ICLR 2016.*

[4] Kingma, D. P., & Ba, J. L. (2014). Adam: A Method for Stochastic Optimization. *Paper presented at ICLR 2015.*

[5] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Presented at NIPS 2012.*

[6] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Presented at ICML 2013.*

[7] Net-Scale Technologies, Inc. (2004). Autonomous Off-Road Vehicle Control Using End-to-End Learning. Final technical report. URL: http://net-scale.com/doc/net-scale-dave-report.pdf

[8] Pomerleau, D. A. (1989). ALVINN, an autonomous land vehicle in a neural network. Technical report. Carnegie Mellon University. URL: http://repository.cmu.edu/cgi/viewcontent.cgi?article=2874&context=compsci

[9] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., … and Fei-Fei, L. (2014) ImageNet large scale visual recognition challenge. URL: https://arxiv.org/abs/1409.0575

[10] Shrivastava, *A*., Pfister, T., Tuzel, O., Susskind, J., Wang, W., & Webb, R. (2017). Learning from Simulated and Unsupervised Images through Adversarial Training. *Paper presented at* CVPR 2017.

[11] Simonyan, K., and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *Paper presented at ICLR 2015.*

[12] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. (2015). Striving for Simplicity: the All Convolutional Net. *Paper presented at ICLR 2015.*

[13] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2015). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15, 1929-1958.*

[14] Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S.,... Darrell, T. (2015). Adapting Deep Visuomotor Representations with Weak Pairwise Constraints. *Paper presented at WAFR 2016.*

[15] Sergey, I., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *Paper presented at ICML 2015.*