Capstone Project 2 – Milestone Report

1 THE PROBLEM

The problem is simple, build a self-driving car and race with it.

The more detailed explanation problem is more personal, it is to build and test a self-driving car using artificial neural network and computer vision to control the vehicle. Using the Donkey Car kit from the donkey car website, a self-driving RC car can be built with a small budget and parts ordered from the internet. The creator of the donkey car included documentations on how to assemble and install the necessary hardware and software, allowing users with little to no experience in robotics and DIY-skills to quickly get their hands dirty on a self-autonomous vehicle project.

I stumbled onto this project while looking for Capstone Project 2 ideas. I have always wanted to explore into the self-autonomous vehicle world and how the machine learning algorithms work under the hood inside these machines. My background in engineering had provided a foundation on how a machine, such as a vehicle, is created from the design, engineering, analysis and manufacturing perspective on the hardware side, but I have very little knowledge on how the software side of it work. So, to understand more, I am using the donkey car as the platform and foundation of the Capstone Project, with the objective of learning and understanding the fundamental techniques of Computer vision coupled with machine learning using Keras to create, manipulate and validate the convolutional neural network (cNN), and ultimately find ways to improve the performance the existing cNN model.

And before I proceed further, this project could not have been made possible without the creator of donkeycar, Adam Conway and William Roscoe, as well as the creator of the donkey car simulator donkey_gym by Tawn Kramer.

2 IDENTIFY YOUR CLIENT

The straight forward answer to the potential client that this project could impact are any of the vehicle's OEMs (Original Equipment Manufacturers) such as Toyota Corp., General Motors, Volkswagen, but this could very easily expand into the public sector of government safety regulators like the National Transport Safety Board or state boards, who are trying to adapt to the implementation of self-driving vehicles on the road. And of course, companies that are already leading the self-driving car field such as Tesla, Waymo and Uber in the Silicon Valley who are the disrupting technology companies that are transforming the transportation landscape.

Last but not least, with this kind of transport autonomy, there are opportunities that could be adapted into any environments that require the transport of goods, tools, or resources from an origin to a destination, such as a factory, hospital, or warehouse.

3 DESCRIBE YOUR DATA SET

The project is based on the NVidia's End-to-End Deep Learning for Self-Driving Cars article published in 2016. The details and the history of the project I would not go into here, but that the frame work of the Convolutional Neural Network is based the architecture described in this article. The cNN network is notable being that it has 5 layers of convolutional 2D layers, with increasing feature map sizes from, 24, 36, 48, 64 and 64 again while the strided kernel unit size starts from a 2x2 stride with a 5x5 kernel for the first 3 conv2D layers, and then into a non-

strided convolution layer with a 3x3 Kernel size in the final two convolutional layers. This architecture will become more important in the simulation of the data discussed below.

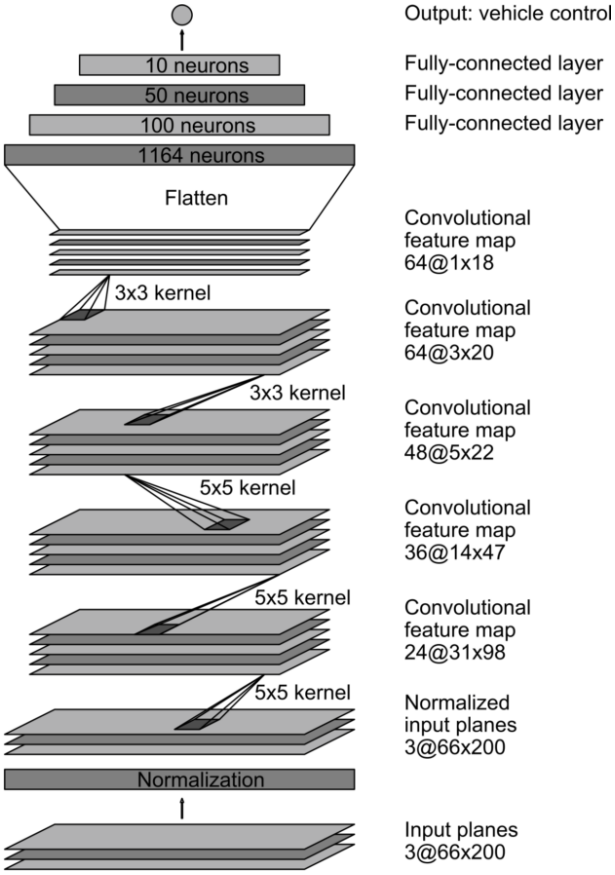

Figure 1: NVidia's End to End Self-Driving Car Model Architecture

One of the key features of this project is the acquisition of the data set. Part of the donkey car kit comes with a simulation package that is "built on the Unity game platform, uses their internal physics and graphics, and connects to a donkey Python process to use our trained model to control the simulate Donkey." However, I thought it would be beneficial to get the experience of gathering my own real-world data, so I procured some tape and created a test track.

The donkey car's camera is mounted on top of the chassis, and it generates pictures with 160 pixels wide by 120 pixels high with a depth of 3 channels, which translates to height, width, and the 3 RGB spectrums. This setup was configured gather pictures at a rate of 20 frames per

second and communicated through the raspberry Pi platform to store images of the donkeycar as it laps around the track. Assuming the car is assembled and calibrated properly, the throttle and steering of the car can be controlled via a WebSocket interface through a Wi-Fi network. All of the parameters within the vehicle can be changed through the host computer and re-upload to the raspberry pi. I have laid out a track (Figure 1) and gathered initial data. But upon analyzing the data and underestimating the battery consumption of the car, the physical track and data gathering had to end before a clean set of data was acquired.

Figures 1-3 show the onboard Donkey car camera of the test track

Figure 2

Figure 3

Figure 4

Figure 5 Indoor Track

However, once the data sets have been created, the initial stock Keras.py lay the foundation cNN models that the donkeycar developers utilized. Then it is possible to experiment and modify the neural network architecture, and evaluate the performances of the different layers.

4 DATA GATHERING

Because of the battery limitation and other system issues on the car, the data acquisition was moved on from acquiring real life data instead to utilize the onboard simulation module for training and testing purposes. However, it was in my opinion a fun exercise and experience to create an environment that generates all the training and testing data for a project from a track.

There are several caveats that needed to be mentioned between the real life vs. stimulated data. The inherent issues with using my own track creation and the own donkey car kit is that the data acquisition is determinate upon the driver's ability, as well as the various electrical and mechanical performances of the vehicle, and the track material (carpet for this case). All of these

variables will cause repeatability issues, which might render future data acquisition problematic and create discrepancy when it comes to training the data and testing the model. However, these are the **real**-world environment, and it is ideally to be training and testing on track data instead of virtual ones.

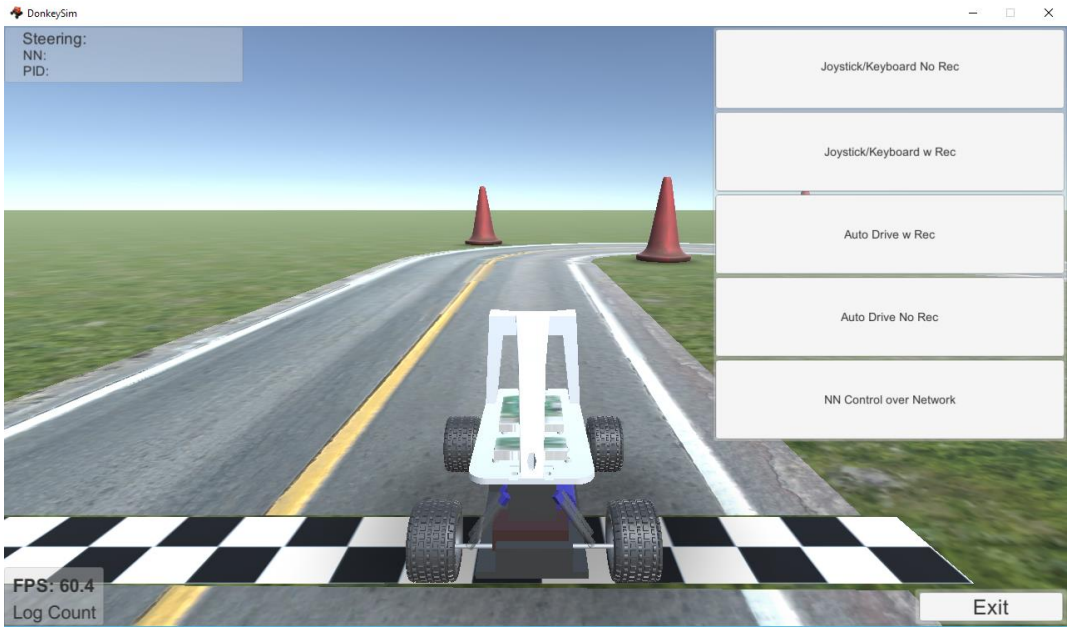

Figure 6: Simulator from Donkey_Gym

The virtual simulated data is generated from the donkey_gym package created by Tawn Kramer. (The git repository can be found here). The simulator has tracks that are procedurally generated and one can reuse the same track to test their models and different machine learning techniques. The simulator includes two separate methods to generate training data, one was user control via keyboard or controller, the other one is auto-drive method using a PID controller for the simulated donkey car that circles the track. Both methods produce images, steering and throttle data in a 3@160x120 images with .json files for steering and throttle input like a real donkey car would. These in turn can be loaded into the different python modules for training, evaluation and analysis.

5 INITIAL FINDINGS

As mentioned in Part 3, initial finding shows that having a proper training set is difficult without proper driving experiences, in addition due to battery capacity limitation, the physical data acquisition had to end pre-maturely before a valid dataset was acquired. To save on time, the rest of the training data was evaluated using the simulated package instead.

Using the latest donkey_gym version 18.9 created by Tawn Kramer, with donkeycar version 2.6.0t and TensorFlow 1.10.0.

20190328_run_05_PID has a MaxPool layer after conv2D_4 layer. This model reduced the parameter counts from 733,828 in the original model to 215,428, and a time cost of 5s per step. An improvement of 28% in run time, however the training loss suffers as a result.

20190328_run_06_PID has a 5-layer Conv2D model with dropout layer in-between. This is also the original model. But takes 7 seconds per step. To investigate further the performance of these two architectures, instead of limiting the model to 10 epochs. The model was tuned such that it was able to run with early stopping, or until it can no longer improves the model. Figure 10 and 13 shows the performance of the expanded network.

Both of these models were tested in donkey_gym under the same settings and environment, and it was found that the network with the maxpool2D layer (run_5 and run_15) performed well and made its way around the track, while the traditional model with the dropout layer (run_6 and run_16) miss the first left turn. This is an interesting development, as run_16 have better loss and accuracy than run_15, but the maxpool layer were able to correctly identify the inputs and trained the model properly to perform the left turn and complete the circuit. Which goes to show that additional training data are needed in order for the base model to perform.

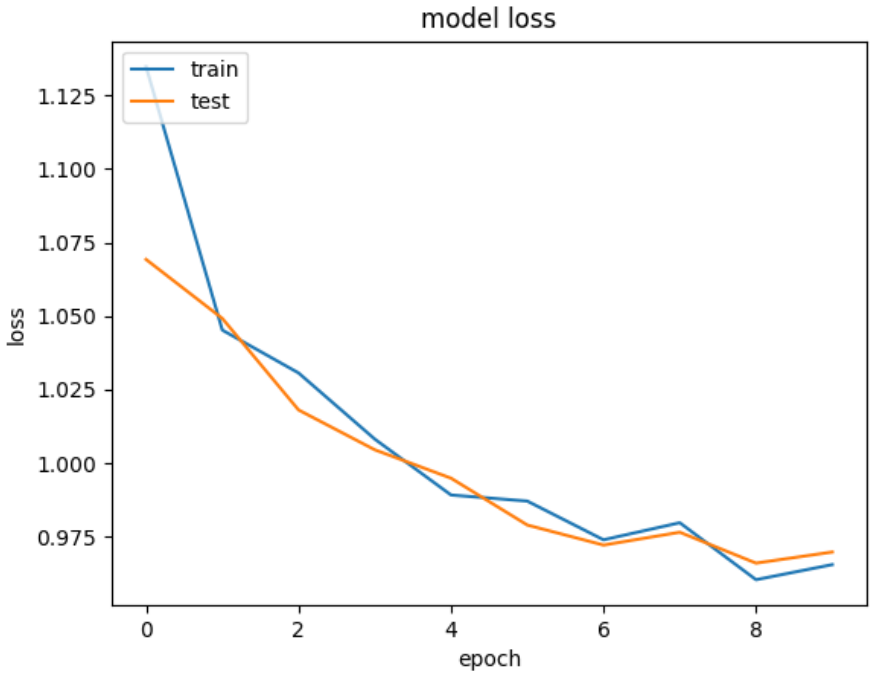

Figure 7 20190328_run_05_PID loss graph with 10 epochs and MaxPool2D layer

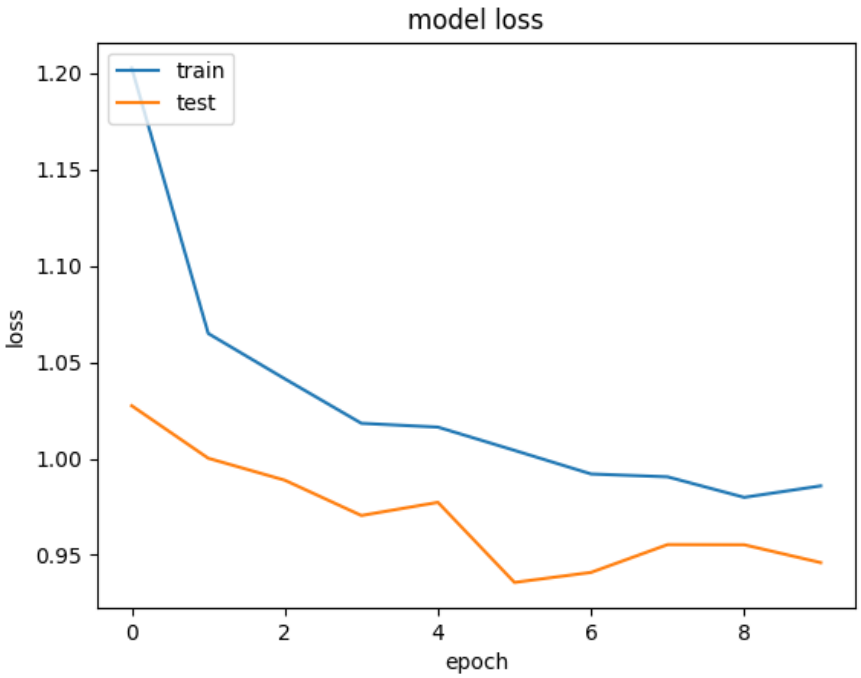

Figure 8 20190328_run_06_PID loss graph with 10 epochs

```
working on model models/20190326_run_02
found 10278 files
num train/val 8200 2078
steps_per_epoch 64 validation_steps 16
Epoch 1/50
64/64 [==============================] - 509s 8s/step - loss: 1.2368 - acc: 0.7523 - val_loss: 0.7728
 - val_acc: 0.7476
Epoch 2/50
64/64 [==============================] - 494s 8s/step - loss: 0.8622 - acc: 0.7585 - val_loss: 0.7940
 - val_acc: 0.7431
Epoch 3/50
64/64 [==============================] - 490s 8s/step - loss: 0.8145 - acc: 0.7618 - val_loss: 0.7544
 - val_acc: 0.7497
Epoch 4/50
64/64 [==============================] - 494s 8s/step - loss: 0.7870 - acc: 0.7612 - val_loss: 0.7960
 - val_acc: 0.7303
Epoch 5/50
64/64 [==============================] - 488s 8s/step - loss: 0.7904 - acc: 0.7581 - val_loss: 0.7085
 - val_acc: 0.7462
Epoch 6/50
64/64 [==============================] - 494s 8s/step - loss: 0.7680 - acc: 0.7587 - val_loss: 0.7321
 - val_acc: 0.7472
Epoch 7/50
64/64 [==============================] - 501s 8s/step - loss: 0.7628 - acc: 0.7563 - val_loss: 0.7102
 - val_acc: 0.7621
Epoch 8/50
64/64 [==============================] - 496s 8s/step - loss: 0.7425 - acc: 0.7608 - val_loss: 0.7211
 - val_acc: 0.7456
Epoch 9/50
64/64 [==============================] - 495s 8s/step - loss: 0.7532 - acc: 0.7628 - val_loss: 0.6912
 - val_acc: 0.7410
Epoch 10/50
64/64 [==============================] - 494s 8s/step - loss: 0.7465 - acc: 0.7585 - val_loss: 0.6876
 - val_acc: 0.7487
Epoch 11/50
64/64 [==============================] - 487s 8s/step - loss: 0.7302 - acc: 0.7575 - val_loss: 0.7196
 - val_acc: 0.7379
Epoch 12/50
64/64 [==============================] - 489s 8s/step - loss: 0.7241 - acc: 0.7571 - val_loss: 0.7132
 - val_acc: 0.7426
Epoch 13/50
64/64 [==============================] - 490s 8s/step - loss: 0.7173 - acc: 0.7625 - val_loss: 0.7009
 - val_acc: 0.7359
Epoch 14/50
64/64 [==============================] - 495s 8s/step - loss: 0.7248 - acc: 0.7567 - val_loss: 0.7180
 - val_acc: 0.7513
Epoch 15/50
64/64 [==============================] - 493s 8s/step - loss: 0.7184 - acc: 0.7613 - val_loss: 0.7101
 - val_acc: 0.7456
Epoch 16/50
64/64 [==============================] - 491s 8s/step - loss: 0.6971 - acc: 0.7545 - val_loss: 0.7004
- val_acc: 0.7472
```

Figure 9: Origin Donkey Network with 5 Conv2D Layers using gamepad inputs

```
working on model models/20190328_run_15_PID

Layer (type)                 Output Shape              Param #
=================================================================
img_in (InputLayer)          (None, 120, 160, 3)       0
_________________________________________________________________
cropping2d_16 (Cropping2D)   (None, 110, 160, 3)       0
_________________________________________________________________
lambda_16 (Lambda)           (None, 110, 160, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 53, 78, 24)        1824
_________________________________________________________________
dropout_98 (Dropout)         (None, 53, 78, 24)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 37, 32)        19232
_________________________________________________________________
dropout_99 (Dropout)         (None, 25, 37, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 17, 64)        51264
_________________________________________________________________
dropout_100 (Dropout)        (None, 11, 17, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 15, 64)         36928
_________________________________________________________________
pool_1 (MaxPooling2D)        (None, 4, 7, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 2, 5, 64)          36928
_________________________________________________________________
dropout_101 (Dropout)        (None, 2, 5, 64)          0
_________________________________________________________________
flattened (Flatten)          (None, 640)               0
_________________________________________________________________
dense_32 (Dense)             (None, 100)               64100
_________________________________________________________________
dropout_102 (Dropout)        (None, 100)               0
_________________________________________________________________
dense_33 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_103 (Dropout)        (None, 50)                0
_________________________________________________________________
steering_throttle (Dense)    (None, 2)                 102
=================================================================
Total params: 215,428
Trainable params: 215,428
Non-trainable params: 0
```

Figure 10: 20180328_run_15_PID model architecture

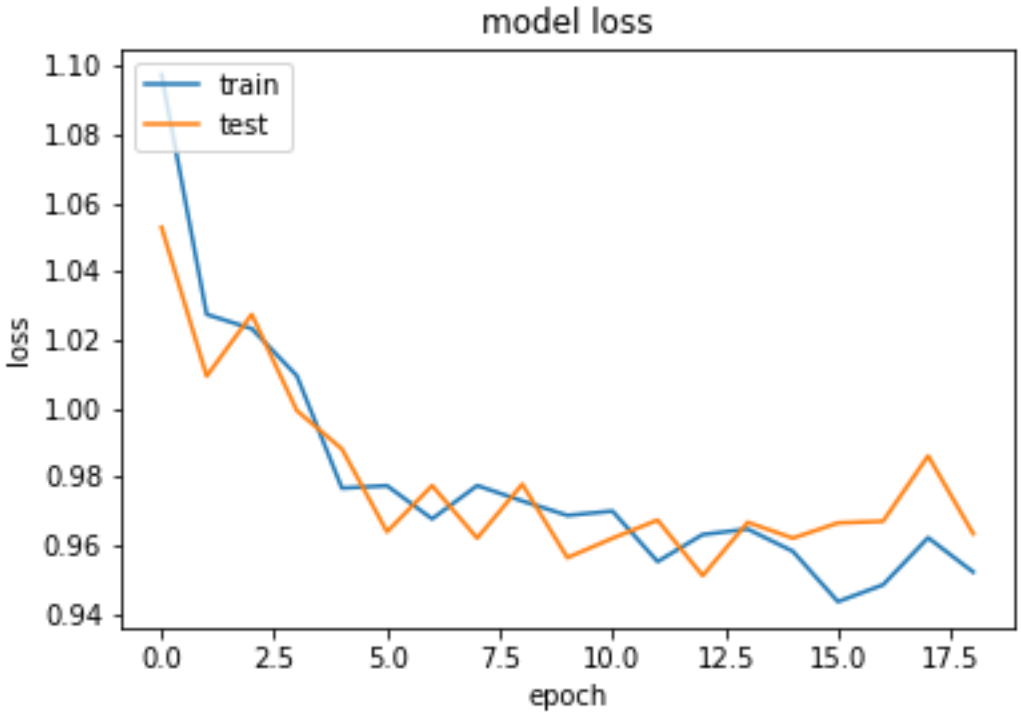

Figure 11: 20180328_run_15_PID_loss graph with expanded epochs

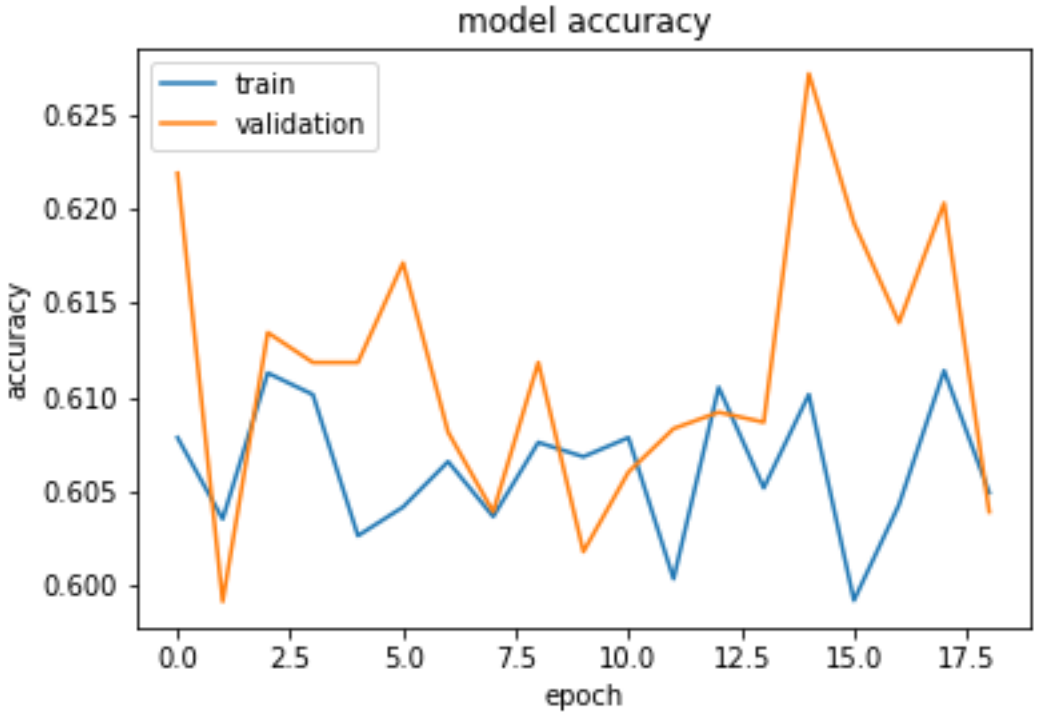

Figure 12: 20180328_run_15 accuracy between train and validation graph

```
working on model models/20190328_run_16_PID
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
img_in (InputLayer)          (None, 120, 160, 3)       0
_________________________________________________________________
cropping2d_13 (Cropping2D)   (None, 110, 160, 3)       0
_________________________________________________________________
lambda_13 (Lambda)           (None, 110, 160, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 53, 78, 24)        1824
_________________________________________________________________
dropout_78 (Dropout)         (None, 53, 78, 24)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 37, 32)        19232
_________________________________________________________________
dropout_79 (Dropout)         (None, 25, 37, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 11, 17, 64)        51264
_________________________________________________________________
dropout_80 (Dropout)         (None, 11, 17, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 15, 64)         36928
_________________________________________________________________
dropout_81 (Dropout)         (None, 9, 15, 64)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 13, 64)         36928
_________________________________________________________________
dropout_82 (Dropout)         (None, 7, 13, 64)         0
_________________________________________________________________
flattened (Flatten)          (None, 5824)              0
_________________________________________________________________
dense_26 (Dense)             (None, 100)               582500
_________________________________________________________________
dropout_83 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_27 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_84 (Dropout)         (None, 50)                0
_________________________________________________________________
steering_throttle (Dense)    (None, 2)                 102
=================================================================
Total params: 733,828
Trainable params: 733,828
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: 20180328_run_16_PID model architecture

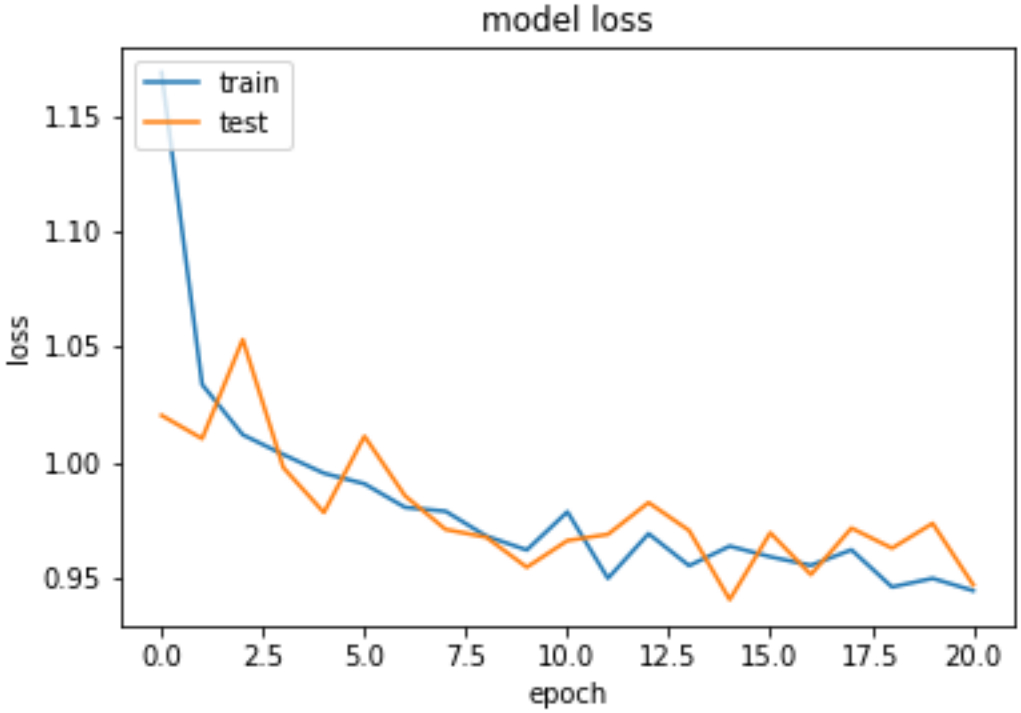

Figure 14: 20180328_run_16_PID_loss graph with expanded epochs

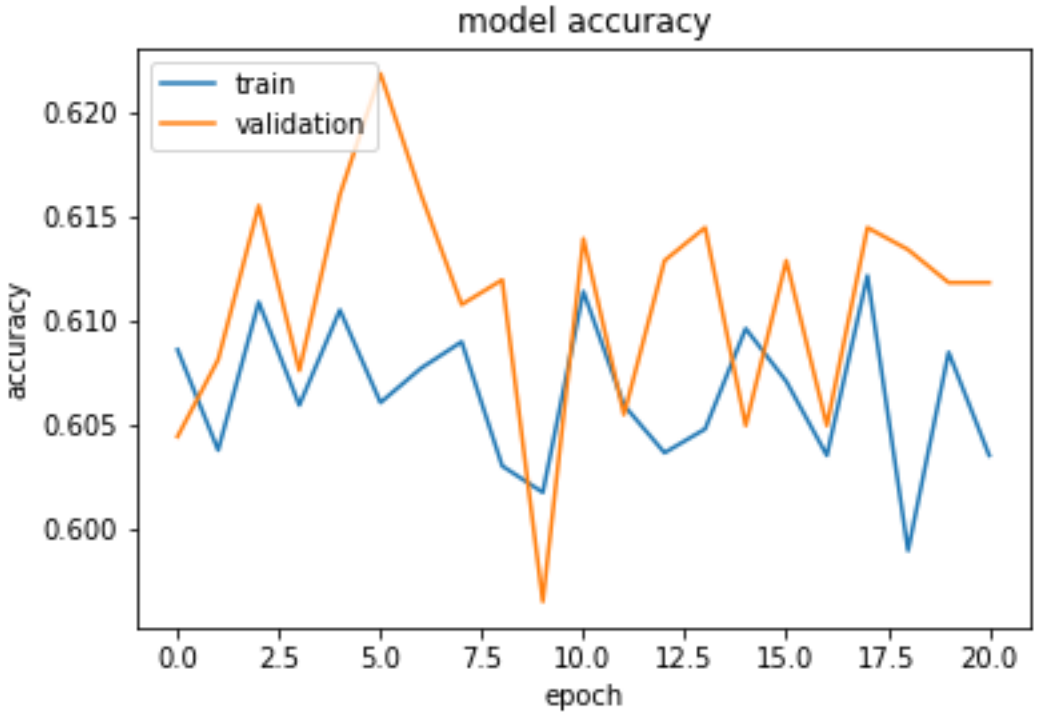

Figure 15: 20180328_run_16_PID_accuracy graph with expanded epochs

6 CONCLUSION

As can be seen, using the donkey_gym simulator can generate training data that can be used to develop and test different architectures and techniques to train the self-driving car. However, it is important to note that the simulator is as good as the data, where it excels in testing out techniques but should not be implemented directly into the real world without more testing with more real-world data.

The original layout of 5 conv2D layers with dropouts in between was a try-and-true method, but due to different software setup, I have found that changing the dropout layer after conv2D_4 to a MaxPool2D layer to help improve performance in donkey_gym dramatically. As shown in the above Figure 8. Multiple other networks were tested to improve and tweak the performance and tested in the simulator, but none were able to create the same level of smoothness as run_06. Additional networks can be modified and trained, but due to the time constraint of this particular project, only twenty or so models were tested in the current state.

The run_15 and run_16 were directly comparisons between the Maxpool2D layer vs. the Dropout layer after conv2D_4. Even though the run_16s provided at first glance a better model from its loss graph and accuracy, it ultimately was not able to drive through the first left turn, and additional training data needed to properly train the model to output the proper steering information. The run_15 maxpool2d Model managed to drive around the track repeatedly. This shows that it is possible to develop a model with an additional maxpool layer that enhances performance by lowering parameters to train and have better image collect at the expense of accuracy of the data. Carefully managing the resources and training parameters will allow an efficient model to be developed.

In hindsight, a majority of the time was spent configuring the model and making sure the right versions are being used between TensorFlow (TensorFlow 1.10), donkeycar (version: 2.6.0.t) and donkey_gym (18.9). In fact, there is still an issue where the input throttle and the generated throttles are not matching even between joystick controller inputs and PID controller inputs, and thus the models were tested on constant throttle. With that in mind, models were created to test different network architectures and parameter sizes to change running time as well as accuracy improvement. Additional works can be found in the next section.

7 NEXT STEPS

Possible next steps:

- Further training of network with more data (>10k images)
- Improve throttle output accuracy, and test data using joystick controller input.
- Test different network architectures with different hyper parameters (activation, regularizes, image processing)
- Test ResNet50, Recursive Network and other Reinforcement Learning models like DDQL.
- Implement into real world track with a working donkey car.
- Re-iterate training and testing models with real world data.

Resources

- http://docs.donkeycar.com/
- https://github.com/tawnkramer/donkey_gym
- https://devblogs.nvidia.com/deep-learning-self-driving-cars/
- https://donkeycar.slack.com/messages
- https://www.datacamp.com/courses/advanced-deep-learning-with-keras-in-python
- https://www.datacamp.com/courses/convolutional-neural-networks-for-image-processing
- https://www.tensorflow.org/api_docs/python/tf/keras
- https://www.jessicayung.com/explaining-tensorflow-code-for-a-convolutional-neural-network/