Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf.

Required Files & Quality of Code

Are all required files submitted?

The project submission includes the following files:

- Python .py files - ***model.py***
- Model files for loading autonomous mode in simulator – ***model.h5***
- ***Drive.py*** file for running the simulation
- ***PDF repor***t exploring and explaining the current implementation
- Output video – ***video.mp4***

Is the code functional?

The provided simulator and drive.py file was not modified and are reused. The car can be driven autonomously around the track by executing:

Python drive.py model.h5

Is the code usable and readable?

The ***model.py*** file contains the code for training and saving the convolution neural network. The file shows the pipeline used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

Has an appropriate model architecture been employed for the task?

The model architecture was begun from a basic neural network which consisted of a single layer as per the project guidelines to understand the simulation and running methodology. Going further, the LeNet implementation was selected but wasn't doing good enough.

As per suggestions from the coursework, The Nvidia model was explored and majority of the structure was implemented successfully. The diagram below shows the depiction of the Nvidia Model architecture designed for end to end self-driving cars.

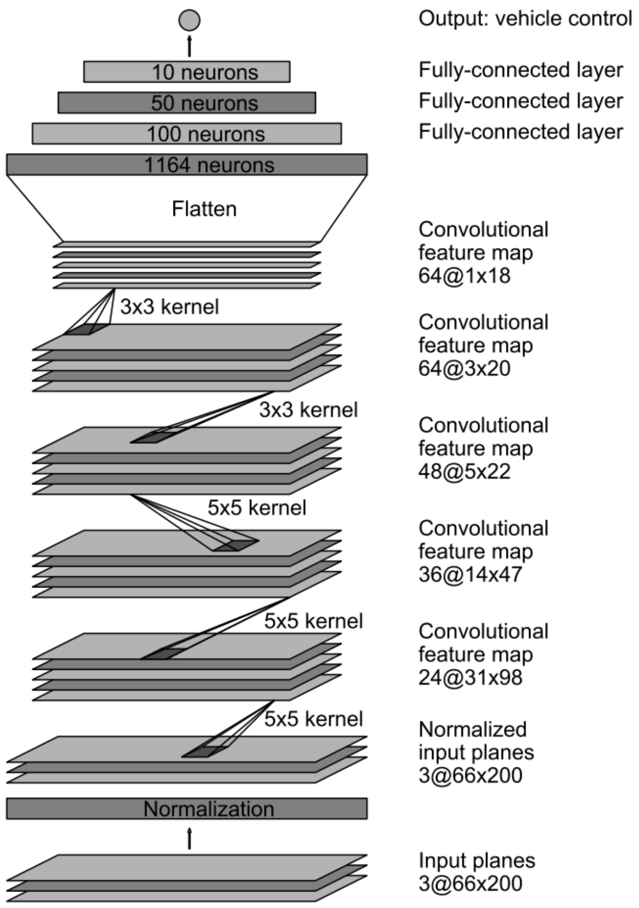


Fig1. Nvidia Architecture

The model was reproduced by taking the following steps:

1. Normalization using a Keras Lambda function
2. Cropping of the top 70 pixels and bottom 25 pixels of the images due to unwanted information such as sky and the hood of the car.
3. Convolutional Layer with 5x5 kernel and stride of 2 – 24 Filters – RELU added
4. Convolutional Layer with 5x5 kernel and stride of 2 – 36 Filters – RELU added
5. Convolutional Layer with 5x5 kernel and stride of 2 – 48 Filters – RELU added
6. Convolutional Layer with 3x3 kernel and stride of 1 – 64 Filters – RELU added
7. Convolutional Layer with 3x3 kernel and stride of 1 – 64 Filters – RELU added
8. Flattening Layer
9. Fully connected Layer – 100
10. Fully connected Layer – 50
11. Fully connected Layer – 10
12. Fully connected Layer – 1

The research material describes the changing of color spaces to LUV; however, it was not implemented. There were no signs of overfitting over 3 epochs due to the complexity of the model and the loss values of both training and validation set.

Has an attempt been made to reduce overfitting of the model?

To prevent overfitting of the model, shuffling and train/test splits have been implemented to randomize data fed into training the model. The training loss and validation loss were monitored to understand trends of overfitting or inappropriate fits.

Have the model parameters been tuned appropriately?

The Adam optimizer was chosen with default parameters and the Mean Squared error (MSE) function was used as a loss function.

Is the training data chosen appropriately?

The most important aspect during training of this model is the data fed into the structure. LeNet or Nvidia model architecture can work if the data consists of appropriate maneuvers to move around the track. This was ensured by collecting the data in training mode of the simulator.

A total of 4 laps of data were collected for training the neural network:

1. 2 Laps of driving on the center of the road
2. 1 Lap with majority of recovery driving covered- Data of car drifting off and recovering to the center of the road was recorded and used in implementation
3. 1 Lap with typical on the edge driving with high speeds - This was done to cover a combination of driving on center and recovery driving to return to center during sharp turns

In addition to recording this training data, more training data was added by doing the following:

1. In addition to center images, both left and right images were used with appropriate steering angles obtained by manipulating the steering angles recorded with a correction factor
2. Also, the images were flipped to introduce clock wise driving data due to the nature of the track. The steering measurements were flipped in sign to accordingly represent clockwise driving and corresponding steering inputs.

This resulted in total of 48216 images and steering measurements- 80 percent for training and 20 % was used in validation.

Architecture and Training Documentation

Is the solution design documented?

The overall model architecture can be logically classified into the following steps:

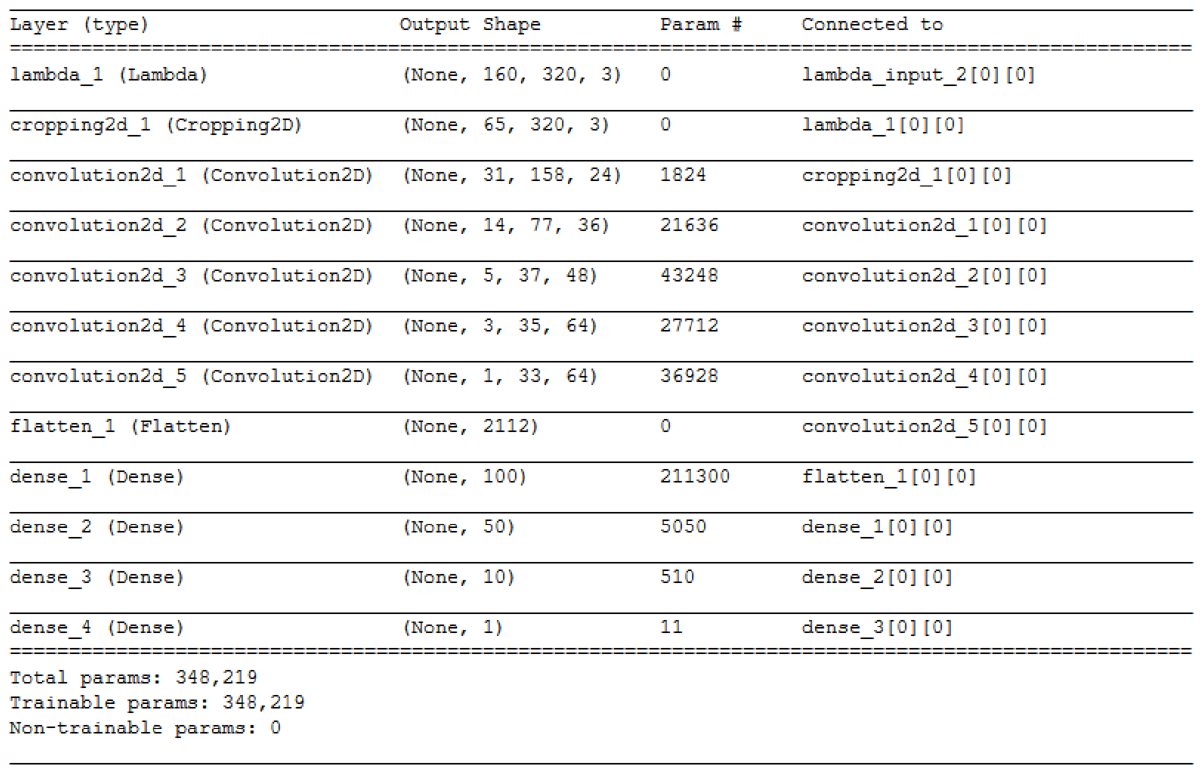
1. ***Image Normalization*** – Normalization makes it easier for the network to generalize different images due to different color schemes. Also, performing this normalization within

the network architecture allows for this scheme to be altered in network architecture and be accelerated via GPU processing

2. ***Feature Extraction*** – The convolutional layers are included to perform feature extraction. The current architecture (as explained by Nvidia) is implemented with real life driving images and was understood by extrapolation to be more effective on simulator-based images as well. The convolutional layers were identified empirically through a series of experiments with varied layer configurations
3. ***Steering Angle Control***– The fully connected layers are designed to function to be like a controller for steering. Fully connected layers condense the information flow to generate steering angles.

Essentially, the current end to end implementation of steering angle from camera images broadly consists of the above mentioned 3 steps. However, it is impossible to define a clean break between the layers that perform feature extraction and steering angle control.

Is the model architecture documented?



```
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)   0           lambda_input_2[0][0]
____________________________________________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 65, 320, 3)    0           lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 31, 158, 24)   1824        cropping2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 14, 77, 36)    21636       convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 37, 48)     43248       convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 3, 35, 64)     27712       convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 1, 33, 64)     36928       convolution2d_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 2112)          0           convolution2d_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           211300      flatten_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          dense_3[0][0]
====================================================================================================
Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0
____________________________________________________________________________________________________
```

Fig2. Model Architecture Visualization

The above figure represents the visualization and the different characteristics of the model architecture.

[***Note***: The total number of parameters is high and hence, efforts will be made to simplify the architecture upon completion of the coursework to understand the simplest neural network to achieve equal accuracy. The current model works but it is highly inefficient in terms of number of parameters.]

Is the creation of the training dataset and training process documented?

As described in the previous section, the most important aspect during training of this model is the training data. This was understood through repeated hours of tuning the model architecture with no substantial gains.

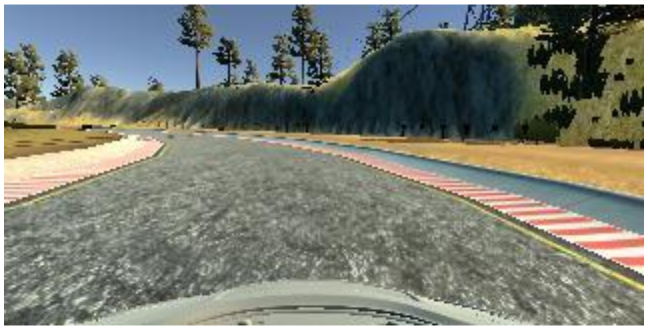
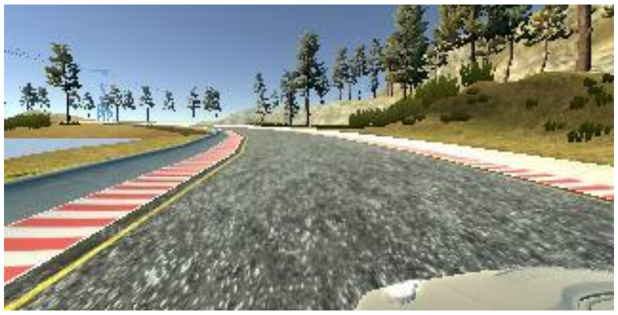


Fig3. Center Dash image



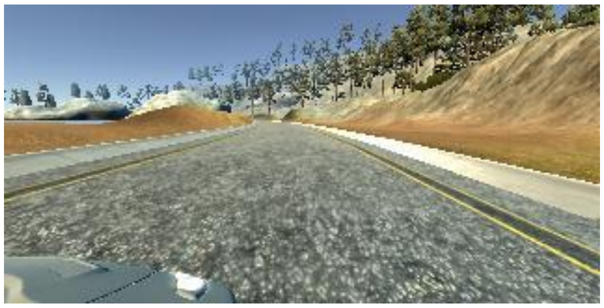


Fig4. Left and Right Dash cam images

A total of 4 laps of data were collected for training the neural network:

1. 2 Laps of driving on the center of the road
2. 1 Lap with majority of recovery driving covered- Data of car drifting off and recovering to the center of the road was recorded and used in implementation
3. 1 Lap with typical on the edge driving with high speeds - This was done to cover a combination of driving on center and recovery driving to return to center during sharp turns

In addition to recording this training data, more training data was added by doing the following:

1. In addition to center images, both left and right images were used with appropriate steering angles obtained by manipulating the steering angles recorded with a correction factor
2. The images were flipped to introduce clock wise driving data due to the nature of the track – more counter clockwise turns than clockwise. The steering measurements were flipped in sign to accordingly represent clockwise driving and corresponding steering inputs.

This resulted in total of 48216 images and steering measurements- 80 percent for training and 20 % was used in validation.

The robustness of the data has really made the trained model very appropriate. The robustness is ensured by equalizing the number of recovery both from left and right edges of the track (achieved through flipped image augmentation).

In addition to the above, the shuffling and the train/test split helped determine if the model was overfitting or underfitting. The adam optimizer chosen negated the need for manual tuning of the learning rate.

Simulation

Is the car able to navigate correctly on test data?

Yes, the car can navigate the track with the tire always on the drivable portion of the track surface. The car does not pop out of the ledges and does safe maneuvers around sharp turns. This can be observed in **video.mp4** file.

Self-Evaluation

This project was very enjoyable and personally very pleased with the results. Training the car to drive itself, with relatively little effort and virtually no explicit instruction, was extremely rewarding.

Shortcomings:

1. The current model does not perform well in the second track - Needs further data collection from track 2 or better generalization
2. Current model architecture is complex and can be simplified.
3. LeNet layer needs further tuning – Crashes depending upon the data fed into for training – The data that works makes the steering angle very wobbly

Further improvements:

1. Reduce complexity of architecture- Further add dropouts and pooling layers to prevent overfitting and make the model for generalized
2. Train the model with track 2 data and attempt to run it on track 1
3. Train the model with real world data with other vehicles, pedestrians and environmental conditions to improve understanding and robustness of tuning the system end to end.
4. Combining the last 3 projects of the Nanodegree program to do Lane detection, vehicle tracking and self-driving – Very excited to attempt this and understand how all the different methods work together to create the self-driving car.