Behavioral Cloning

How to train your Car

We will utilize Deep Learning techniques to predict the angle of car turn and compare it with ground truth and train over iterations to have model learn right weights. Once the weights are set with minimal error, we will let the car loose on a track to see how it performs in autonomous mode.

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

Implementation

Here I will consider describe how I addressed each point in my implementation.

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing

```
python drive.py model.h5
```

3. Submission code is usable and readable

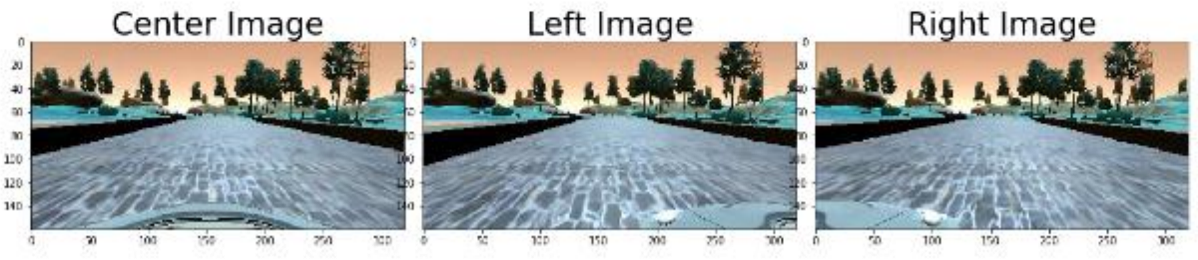
The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. Training Data and extracting images

I attempted to collect the data and drove multiple laps and spend considerable time in training with my original data, but there were issues on the sharp turn. I noticed that throttle and break data was not proper when I collected the data. Then I switched to the data provided by Udacity and I could see change in throttle more responsive to turns and speed.

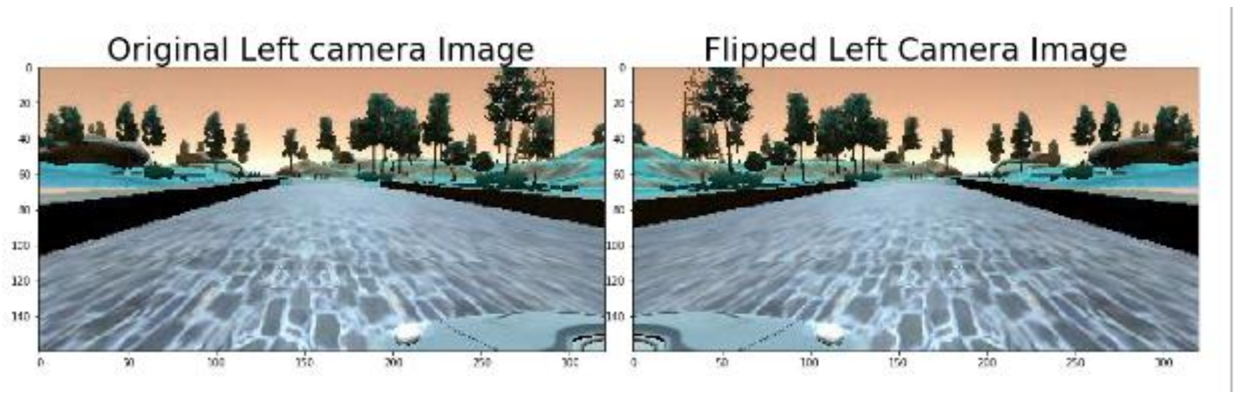
Lines 38-50 in model.py opens the driving_log.csv and extract the right, left and center image paths as well as angle data in lists. I use the shuffle method to shuffle the data and create some randomness in samples. Then I create the validation data set with 90% for training and 20% for validation [line 62].

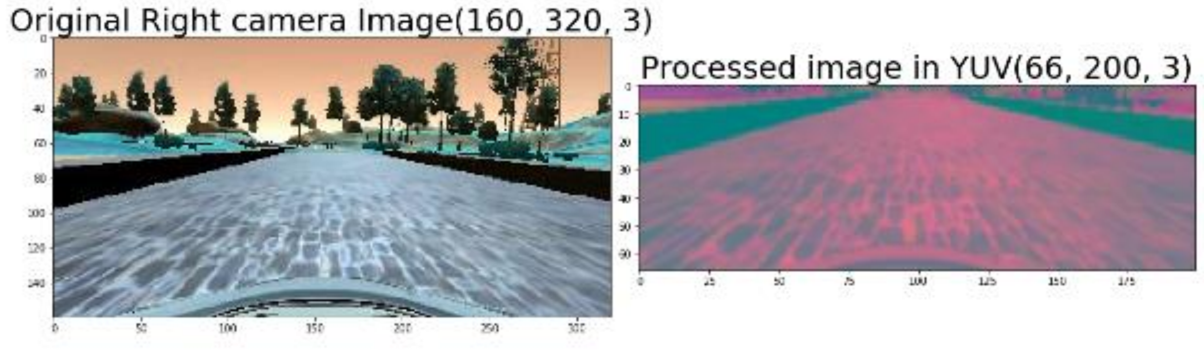


2. Data augmentation

- As advised by Udacity, I used the left left and right cameras with correction. If the measurement of angle was less than -0.15, I made correction of original measurement – 0.25. Similarly, for right camera, if angle > 0.15, I made the correction of +0.25. After correction in angles, I added the data paths for center, right and left and respectively for angles and created training data as X_train(data paths for images) and y_train (measurements)

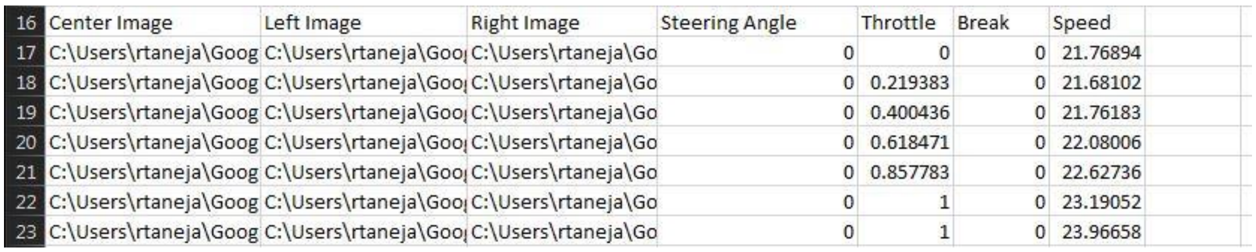
- As advised in Udacity lectures and other blogs, pre-processing the training data was very effective in getting the result. Here is the following pre-processing I applied to my data:
    - Generating random brightness
    - Flipping images
    - Cropping images to remove unnecessary details like trees and upper portion of image
    - Converting to YUV space for Nvidia model architecture and resizing it for 200,66 to match network's input shape.




Original Right camera Image(160, 320, 3)
Processed image in YUV(66, 200, 3)

3. Conclusion on data collection and augmentation

Collecting training data was the hardest part. I got the feeling that the way you drive in training mode affects how your car reacts in autonomous mode. Keeping car in the center of the track and not making many turns is not a good way of training. I felt that constantly taking right and left turns helped greatly in making decisions in autonomous mode; especially training on lot of data where car is coming inside towards the center from the boundary. Also, training for different types of boundaries helped a lot. Otherwise the car won't certain type of boundaries. Training data was eventually chosen to keep the vehicle driving on the road.



16	Center Image	Left Image	Right Image	Steering Angle	Throttle	Break	Speed
17	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0	0	21.76894
18	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.219383	0	21.68102
19	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.400436	0	21.76183
20	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.618471	0	22.08006
21	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	0.857783	0	22.62736
22	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	1	0	23.19052
23	C:\Users\rtaneja\Goog	C:\Users\rtaneja\Goo	C:\Users\rtaneja\Go	0	1	0	23.96658

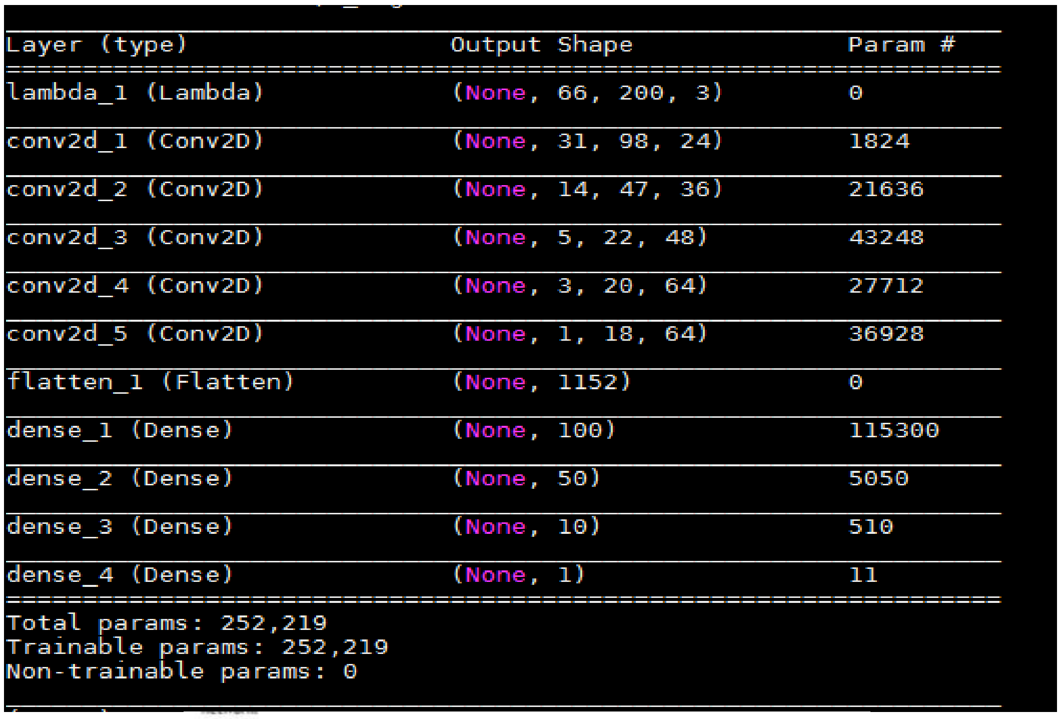
Model Architecture and Training Strategy

1. Solution Design Approach

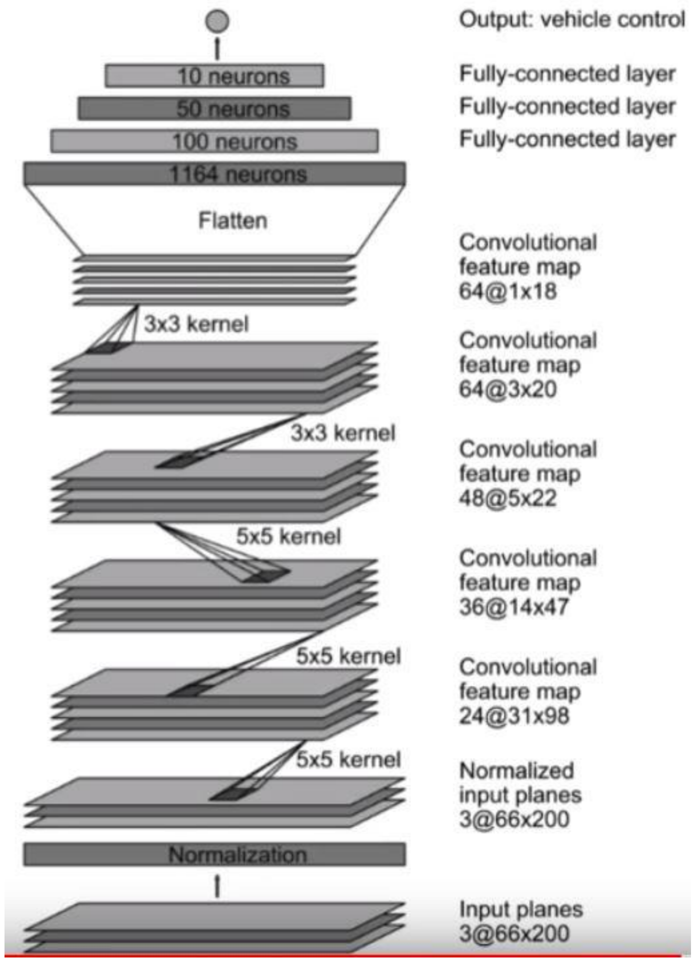
The overall strategy for model architecture was to have the lowest delta between training and validation runs. To gauge how well the model was working, I split my image and steering angle data into a training and validation set.

- My first step was to use Flatten and Dense layer to see how a very basic model performs. The result was very poor as expected and the car didn't even run and mse was very high for both testing and validation set.
- My second approach was to use Convolution layer and use 6 filters and 5x5 kernels with 'relu' as the activation layer. I also used a lambda function to perform image normalization on my training data set. The mse training and validation became lower but looked like i will still have very far to go. Difference between training and validation mse was also high indicating overfitting.
- I used the Nvidia's neural network architecture which has 5 Convolution2D layers and 3 FC layers. When I started training with this, the mse dropped a lot indicating this is a good network choice here. The delta between Training and Validation was also very less indicating that my model is now generalizing.
- **I had to use regularization in Dense layers as a method to reduce overfitting.**

The final architecture looks like this:



```
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 66, 200, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               115300
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
```



3. Generators: I created two generators, one for training and other for validation. Per batch, I took a random offset and preprocessed it, then flipped the image and appended it to existing data set. Lines 148-178 shows the generator part of pipeline. I used the batch size of 128.

4. I modified the drive.py with a similar preprocessing function so that the image size is 66x200 and image is in YUV color space as required by Nvidia model. Line 51-56 in drive.py is where I have added pre-processing function. I did play with various throttle values as suggested in some blogs, but I found out that keeping it to original is best way to achieve desired output.

5. I created the run_epic file and video file named run_epic.mp4 which shows that car driving autonomously.

Other thoughts and things to do in Future

- I believe having the right training data set for this project was the biggest task. I would like to learn how Udacity created the right data set.

- My car is not very straight and kind of takes too many lefts and rights, I would like it to smoothen a bit.