Udacity Self-Driving Car Engineer Nanodegree - Behavioural Cloning Project

GitHub: https://github.com/harivamsi9/Project-Behavioral-Cloning-using-Deep-Learning-for-Self-Driving-Cars
VideoOutput: https://www.youtube.com/watch?v=MB_5WTdvasI&feature=share

Introduction

The objective of this project is to teach the computer to drive car on on the basis of data collected in simulator provided by Udacity. Turns out Udacity has provided a sample dataset (God Bless those folks!) for people like us. Here we apply the concepts of Deep Learning and Convolutional Neural Networks to teach the computer to drive car autonomously.

We feed the data collected from Simulator to our model, this data is fed in the form of images captured by 3 dashboard cams center, left and right. The output data contains a file data.csv which has the mappings of center, left and right images and the corresponding steering angle, throttle, brake and speed.

Using Keras Deep learning framework we can create a model.h5 file which we can test later on simulator with the command "python drive.py model.h5". This drive.py connects your model to simulator. The challenge in this project is to collect all sorts of training data so as to train the model to respond correctly in any type of situation.

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` - The script used to create and train the model.
- `drive.py` - The script to drive the car.
- `model.h5` - The saved model.
- `WriteUpReport_UdacityBehaviorCloning` as pdf file as per the rubrics
- `Run1.mp4` - A video recording of your vehicle driving autonomously at least one lap around the track.

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
```

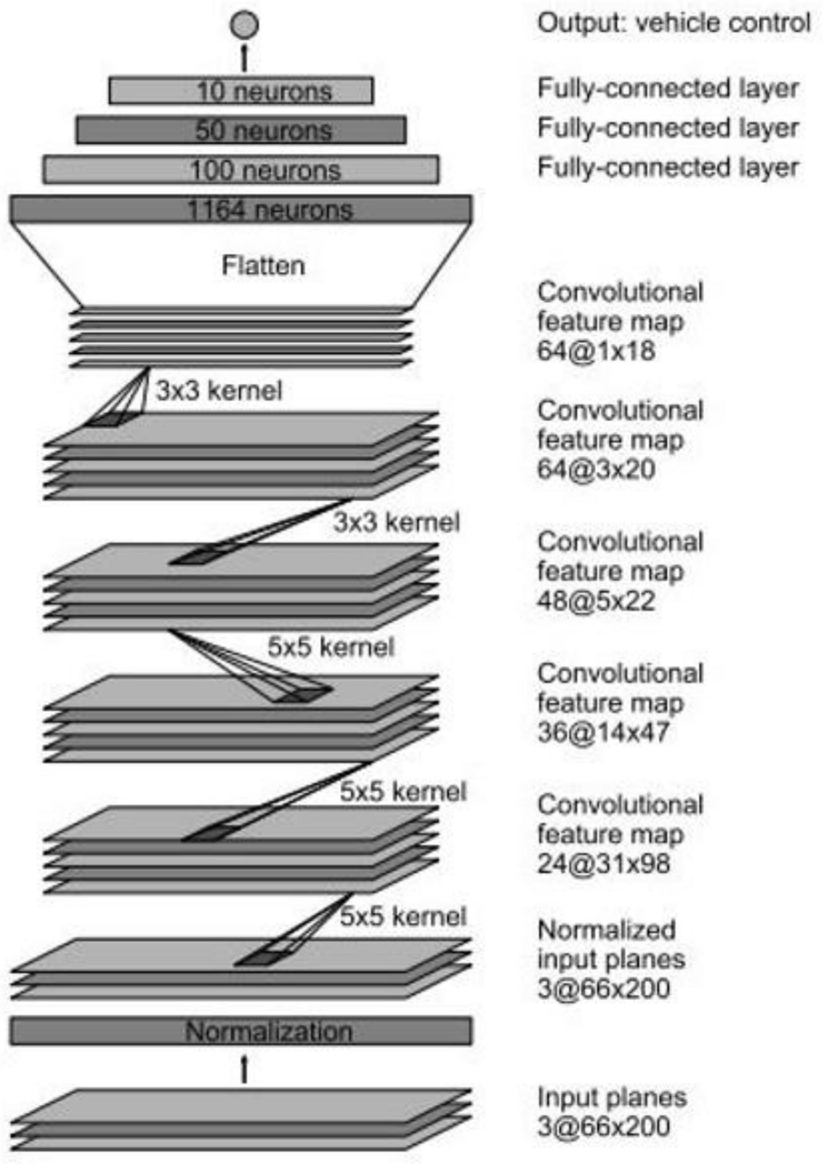
3. Submission code is usable and readable

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

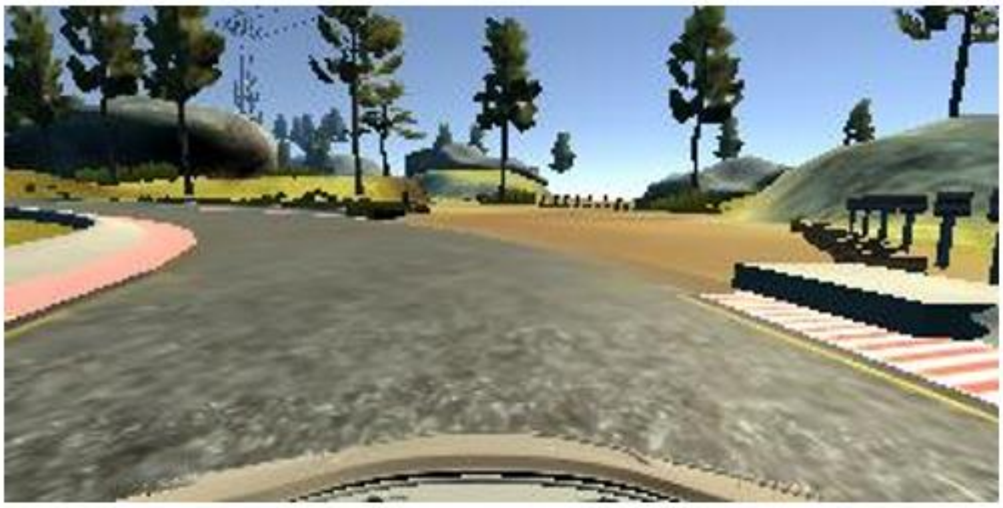
Model Overview

- I decided to test the model provided by NVIDIA as suggested by Udacity. The model architecture is described by NVIDIA here. As an input this model takes in image of the shape (60,266,3) but our dashboard images/training images are of size (160,320,3). I decided to keep the architecture of the remaining model same but instead feed an image of different input shape which I will discuss later.



Loading Data

- I used the the dataset provided by Udacity
- I am using OpenCV to load the images, by default the images are read by OpenCV in BGR format but we need to convert to RGB as in drive.py it is processed in RGB format.
- Since we have a steering angle associated with three images we introduce a correction factor for left and right images since the steering angle is captured by the center angle.
- I decided to introduce a correction factor of 0.2
- For the left images I increase the steering angle by 0.2 and for the right images I decrease the steering angle by 0.2
- Sample Image



Preprocessing

- I decided to shuffle the images so that the order in which images comes doesn't matters to the CNN
- Augmenting the data- i decided to flip the image horizontally and adjust steering angle accordingly, I used cv2 to flip the images.
- In augmenting after flipping multiply the steering angle by a factor of -1 to get the steering angle for the flipped image.
- So according to this approach we were able to generate 6 images corresponding to one entry in .csv file

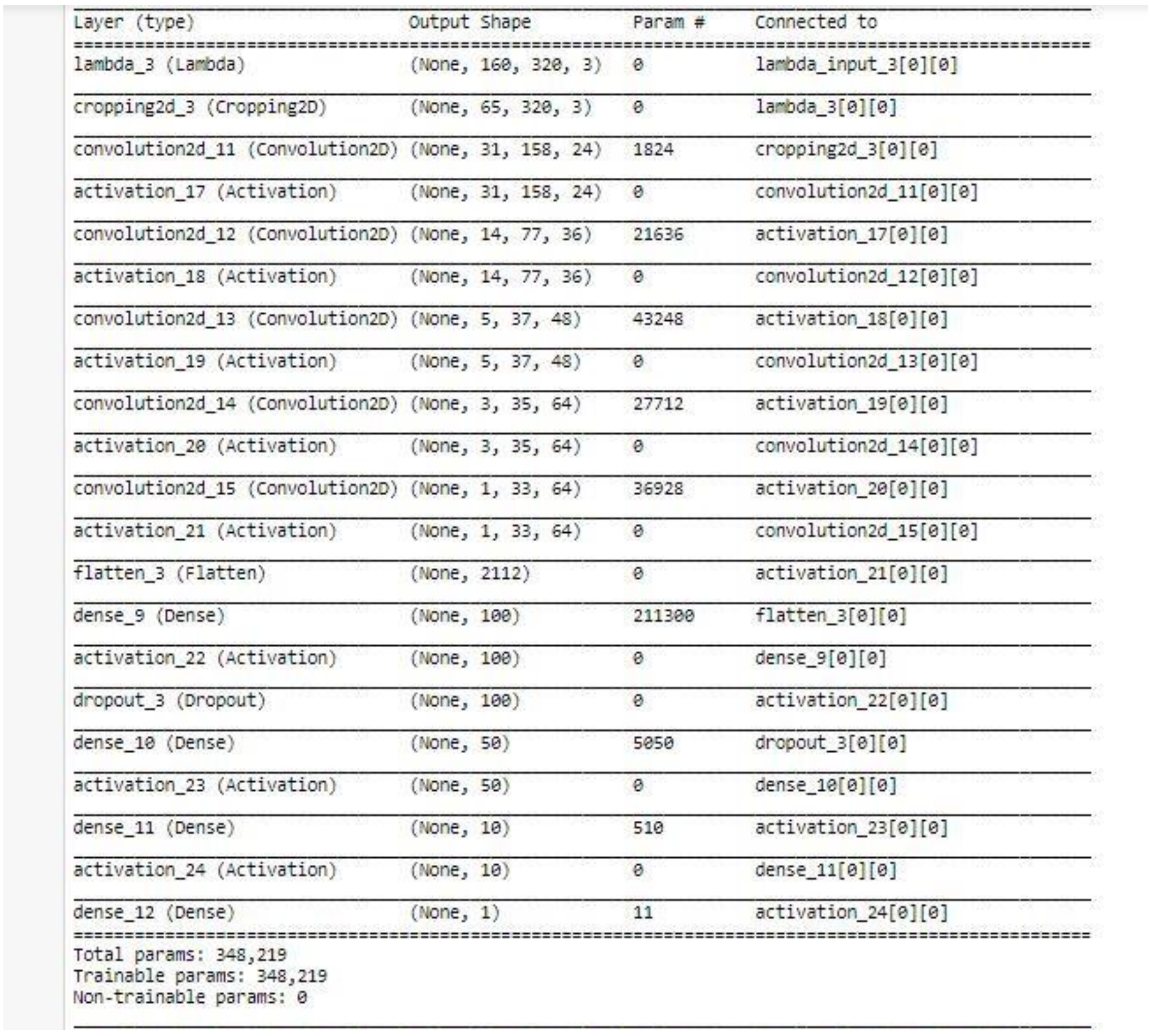
Creation of the Training Set & Validation Set

- I analyzed the Udacity Dataset and found out that it contains 9 laps of track 1 with recovery data. I was satisfied with the data and decided to move on.
- I decided to split the dataset into training and validation set using sklearn preprocessing library.
- I decided to keep 15% of the data in Validation Set and remaining in Training Set

- I am using generator to generate the data so as to avoid loading all the images in the memory and instead generate it at the run time in batches of 32. Even Augmented images are generated inside the generators.

Final Model Architecture

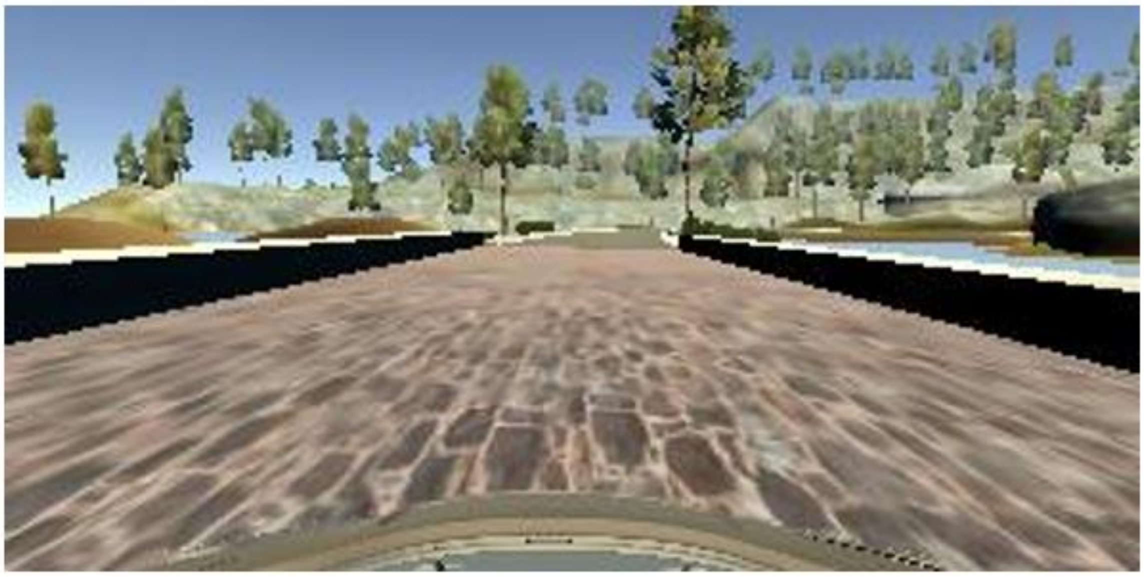
- I made a little change to the original NVIDIA architecture, my final architecture looks like in the image below



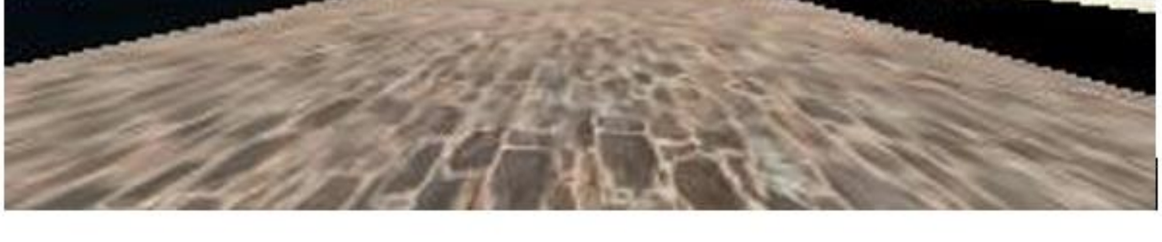
```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_3 (Lambda)                (None, 160, 320, 3)   0           lambda_input_3[0][0]
____________________________________________________________________________________________________
cropping2d_3 (Cropping2D)        (None, 65, 320, 3)    0           lambda_3[0][0]
____________________________________________________________________________________________________
convolution2d_11 (Convolution2D) (None, 31, 158, 24)   1824        cropping2d_3[0][0]
____________________________________________________________________________________________________
activation_17 (Activation)       (None, 31, 158, 24)   0           convolution2d_11[0][0]
____________________________________________________________________________________________________
convolution2d_12 (Convolution2D) (None, 14, 77, 36)    21636       activation_17[0][0]
____________________________________________________________________________________________________
activation_18 (Activation)       (None, 14, 77, 36)    0           convolution2d_12[0][0]
____________________________________________________________________________________________________
convolution2d_13 (Convolution2D) (None, 5, 37, 48)     43248       activation_18[0][0]
____________________________________________________________________________________________________
activation_19 (Activation)       (None, 5, 37, 48)     0           convolution2d_13[0][0]
____________________________________________________________________________________________________
convolution2d_14 (Convolution2D) (None, 3, 35, 64)     27712       activation_19[0][0]
____________________________________________________________________________________________________
activation_20 (Activation)       (None, 3, 35, 64)     0           convolution2d_14[0][0]
____________________________________________________________________________________________________
convolution2d_15 (Convolution2D) (None, 1, 33, 64)     36928       activation_20[0][0]
____________________________________________________________________________________________________
activation_21 (Activation)       (None, 1, 33, 64)     0           convolution2d_15[0][0]
____________________________________________________________________________________________________
flatten_3 (Flatten)              (None, 2112)          0           activation_21[0][0]
____________________________________________________________________________________________________
dense_9 (Dense)                  (None, 100)           211300      flatten_3[0][0]
____________________________________________________________________________________________________
activation_22 (Activation)       (None, 100)           0           dense_9[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)              (None, 100)           0           activation_22[0][0]
____________________________________________________________________________________________________
dense_10 (Dense)                 (None, 50)            5050        dropout_3[0][0]
____________________________________________________________________________________________________
activation_23 (Activation)       (None, 50)            0           dense_10[0][0]
____________________________________________________________________________________________________
dense_11 (Dense)                 (None, 10)            510         activation_23[0][0]
____________________________________________________________________________________________________
activation_24 (Activation)       (None, 10)            0           dense_11[0][0]
____________________________________________________________________________________________________
dense_12 (Dense)                 (None, 1)             11          activation_24[0][0]
====================================================================================================
Total params: 348,219
Trainable params: 348,219
Non-trainable params: 0
____________________________________________________________________________________________________
```

- As it is clear from the model summary my first step is to apply normalization to the all the images.
- Second step is to crop the image 70 pixels from top and 25 pixels from bottom. The image was cropped from top because I did not wanted to distract the model with trees and sky and 25 pixels from the bottom so as to remove the dashboard that is coming in the images.

Sample Input Image-



Cropped Image-



- Next Step is to define the first convolutional layer with filter depth as 24 and filter size as (5,5) with (2,2) stride followed by ELU activation function
- Moving on to the second convolutional layer with filter depth as 36 and filter size as (5,5) with (2,2) stride followed by ELU activation function

- The third convolutional layer with filter depth as 48 and filter size as (5,5) with (2,2) stride followed by ELU activation function
- Next we define two convolutional layer with filter depth as 64 and filter size as (3,3) and (1,1) stride followed by ELU activation funciton
- Next step is to flatten the output from 2D to side by side
- Here we apply first fully connected layer with 100 outputs
- Here is the first time when we introduce Dropout with Dropout rate as 0.25 to combact overfitting
- Next we introduce second fully connected layer with 50 outputs
- Then comes a third connected layer with 10 outputs
- And finally the layer with one output.

Here we require one output just because this is a regression problem and we need to predict the steering angle.

Attempts to reduce overfitting in the model

After the full connected layer I have used a dropout so that the model generalizes on a track that it has not seen. I decided to keep the Dropoout rate as 0.25 to combact overfitting.

Model parameter tuning

- No of epochs= 5
- Optimizer Used- Adam
- Learning Rate- Default 0.001
- Validation Data split- 0.15
- Generator batch size= 32
- Correction factor- 0.2
- Loss Function Used- MSE (Mean Squared Error as it is efficient for regression problem).

After a lot of testing on track 1 I was convinced that this is my final model.

Output Video

The video is available in workspace itself "/home/workspace/CarND-Behavioral-Cloning-P3/run1.mp4" – this was recorded using the Video.py file.

I have screen recorded the simulation output and you can see that in here:
https://www.youtube.com/watch?v=MB_5WTdvasI&feature=share