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.

Model Architecture

My model consists of a convolution neural network with 3x3 and 5x5 filter sizes and depths between 24 and 53 (model.py lines 18-24) The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Lambda, Conv2D, Dropout

def model(loss='mse', optimizer='adam'):
    model = Sequential()
    model.add(Lambda(lambda x:  (x / 127.5) - 1., input_shape=(70, 160, 3)))
    model.add(Conv2D(filters=24, kernel_size=5, strides=(2, 2), activation='relu'))
    model.add(Conv2D(filters=36, kernel_size=5, strides=(2, 2), activation='relu'))
    model.add(Conv2D(filters=48, kernel_size=5, strides=(2, 2), activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=3, strides=(1, 1), activation='relu'))
```

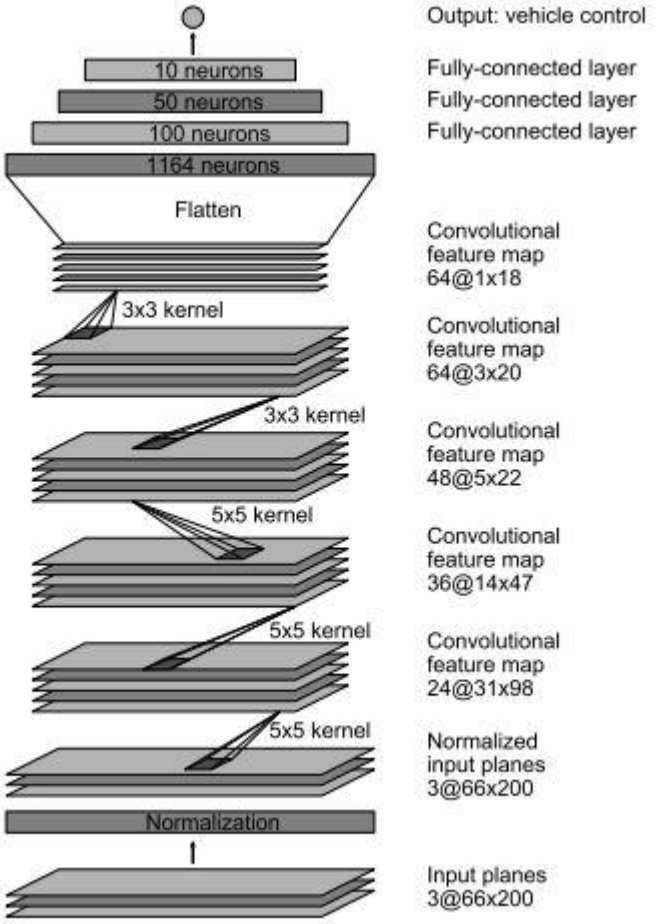
```
model.add(Conv2D(filters=64, kernel_size=3, strides=(1, 1), activation='relu'))

model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1))

model.compile(loss=loss, optimizer=optimizer)

return model
```

Attempts to reduce overfitting in the model

To prevent overfitting, I used several data augmentation techniques like flipping images horizontally as well as using left and right images to help the model generalize. The model was trained and validated on different data sets to ensure that the model was not overfitting. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

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.



Pre –processing of image

To create the training data, I used the Udacity sample data as a base. For each image, normalization would be applied before the image was fed into the network. In my case, a training sample consist of three images:

1. Center camera image
2. Left camera image
3. Right camera image

Conversion of Raw Image(BGR) to RGB

Resized Image

Creation of the Training Set & Training Process

- No of epochs= 2
- 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

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

I found that this was sufficient to meet the project requirements for the first track. The model was then tested on the track to ensure that the model was performing as expected.