BEHAVIOR CLONING PROJECT

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 (well commented)
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- Behavior_Cloning.pdf summarizing the results

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 validating the convolution neural network for predicting the steering angle. The file shows the pipeline I used for training and validating the model, and it is well commented.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 3x3,5x5 filter sizes and depths between 32 and 64 **(model.py lines 88-112).** My model is inspired from one developed by Autonomous Driving team at Nvidia.

The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer **(code line 91)**.

2. Attempt to reduce overfitting in the model

I have used dropout layers after the fully connected layer to avoid overfitting. **(code line 107, 109).** To keep a sanity check, every time the model was trained, it was checked by running through the simulator. Training and Validation data has been used with a split rate of 80-20.

3. Model Parameter Tuning

I have used Adam optimizer, so the learning rate has not been set manually. (Model.py line 113)

4. Appropriate Training Data

I have used the center, right and left lane driving data and steering angle for training model. To counter the effect of left steering bias, I have also used data augmentation by flipping the images and taking the opposite sign of the steering measurement.

Model Architecture and Training Strategy

1. Solution Design Approach

I started by a fairly simple architecture to see if everything works correctly using only 2 convolution layers and a few dense layers. When the model started to work, I trained it on Nvidia model and saw that it was overfitting, for which I augmented the data as explained in (Appropriate Training Data). Then I also introduced some dropout layers in the fully connected layers, after which the mean squared error was minimized and got a fairly good result when viewed on the simulator.

2 .Model Architecture :

The final model architecture (model.py lines 88-112) consisted of a convolution neural network with the following layers and layer sizes,

LAYER	**DESCRIPTION**
Input	RGB image of Size (160 x320x3)
Cropping2D (to remove the unuseful information)	**Output = 90x320x3**
Lambda Layer (Normalization)	**Output = 90x320x3**
Convolutional 5x5@24	1x1 Stride, padding= 'valid ' **Output = 86x316x24**
RELU	Activation Layer
MaxPooling	**Output = 43x158x24**
Convolutional 5x5@36	1x1 Stride, padding= 'valid ' **Output = 39x79x36**
RELU	Activation Layer
MaxPooling	**Output = 20x40x36**
Convolutional 5x5@48	1x1 Stride, padding= 'valid ' **Output = 16x36x48**
RELU	Activation Layer
MaxPooling	**Output = 8x18x48**
Convolutional 3x3@64	1x1 Stride, padding= 'valid '

	Output = 6*16*64
RELU	Activation Layer
Convolutional 3x3@64	1x1 Stride, padding= ‘valid ’ **Output = 4*14*64**
MaxPooling	**Output = 2x7x64**
Fully Connected	Output = 896 Nodes
RELU	Activation Layer
Fully Connected	Input nodes=896, **output nodes = 100**
Dropout	Rate =0.2
Fully Connected	Input nodes = 100, **output nodes = 50**
Dropout	Rate =0.2
Fully Connected	Input nodes = 50, **output nodes = 10**
Fully Connected	Input nodes = 10, **output nodes = 1**

3. Creation of the Training Set & Training Process

My generated data was not giving good result so I have used the provided data. I have used the center, left and right lane images and steering angle for model creation. I have also augmented the data, by flipping the images and taking the negative of the steering angle. Furthermore, the training and validation data has been split in the ratio of 80-20%. My model was able to achieve the desired loss by 20 epochs and I used an adam optimizer so that manually training the learning rate was not required.