File Submitted and 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
- writeup_report.pdf summarising the results
- video.mp4 that captures testing on model by launching Udacity provided simulator and entering autonomous mode.

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.

I used Python generator function to generate data for training rather than storing the training data in memory.(model.py, lines 73-112)

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My Model is NVIDIA inspired CNN architecture. It consists of 9 layers,5 convolutional layers and 4 fully connected layers. The input images are split into YUV planes, resized to 66*200 and zero-mean normalised before passed to the network(model.py lines 54-70). Model uses 2×2 stride and a 5×5 kernel in first three convolution layer and a 1x1 stride and a 3x3 kernel in last two convolution layer. (model.py lines 123-154)

The model includes ELU layers to introduce nonlinearity.

2. Attempts to reduce overfitting in the model

Data Augmentation is the technique to avoid overfitting. Following techniques are used to add more data to the model.

1. Flipping data would not only ensure generalisation of the model but also make car ride along the centre of the road and NOT drift towards left or right.(mode.py, lines 101-105)

2. Augmenting the dataset with steering angle > 0.15 or less than -0.15 by increasing steering angle by a random value between -1 and 1.(model.py, lines 10-49).

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py code line 51,151) . The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.(video.mp4)

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 150).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road, driving smoothly around the curves.

Left and right camera images steering angles were corrected by +0.20 and -0.20 respectively for car to ride along the centre of the road.

For details refer creation of the training set and training process in the next section.

Architecture and Training Documentation

1. Solution Design Approach

My first step was to use a convolution neural network model. I chose NVIDIA CNN as suggested by Udacity.

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 used data augmentation techniques as described in later section to avoid problem of overfitting.

Data augmentation improved the results and mean squared error on validation set was reduced and was at par with mean squared error on the training set.

The final step was to run the simulator to see how well the car was driving around track one.The vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 123-147) consisted of a NVIDIA inspired convolution neural network as seen below.

Layer	Input	Kernel	Stride
Conv1	66*200*3	5*5	2*2
Conv2	31*98*24	5*5	2*2
Conv3	14*47*36	5*5	2*2
Conv4	5*22*48	3*3	1*1
Conv5	3*20*64	3*3	1*1
Flatten	1*18*64		
FC1	1164		
FC2	100		
FC3	50		
FC4	10		

3. Creation of the Training Set & Training Process

I increased data points by capturing training data using Udacity provided simulator, capturing images from the left and right side of the road and smoothly driving along the curves.

After the collection process, I had 8036(Udacity)+4035(recovery) = 12061 number of data points includes both Udacity data and recovery data. I augmented the data set by perturbing the steering angles on Udacity and Recovery data set for images with non zero steering angle. After eliminating data points with zero steering angles and increasing images with non zero steering angle, I was left with 24110 data points(24110*3 = number of images).(model.py, lines 10-49)

I then selected single image from each data point by randomly choosing between left, right and centre and provided +0.20 and -0.20 correction on the left and right camera images resp. (model.py, lines 83-93). Thus total number of images = 24110 are selected by this process.

I then preprocessed these images by splitting RGB images to YUV. I cropped the image, 45 from top and 25 from bottom. I then resized images to 66* 200 to suit NVIDIA CNN model. I then normalised the images such that pixels values have zero mean.

I then did data augmentation by flipping the images and inverting the steering angle (multiply by -1). This increased dataset count to 48220.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. I used an adam optimizer and total number of epochs as 2.