Behavior Cloning Project **Hardik Patel**
Udacity – Self Driving Car – Term 1

(I used verbiage from the writeup template, and changed it where required)

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 to create and train the model - I describe the model below
- drive.py for driving the car in autonomous mode – I did not change this file in any way
- model.h5 - a trained convolution neural network – Model created by running model.py
- writeup_report.pdf – This file summarizing the details of the project
- run1.mp4 – Video of the car driving.

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 run1
```

I use the "run1" directory to store all the images from the drive in autonomous mode to create a video, as mentioned in the lesson.

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. An appropriate model architecture has been employed

My model consists of a convolution neural network with varying filter sizes between 24 and 64. This model is shown in lines 83-97 in model.py

The model includes RELU layers to introduce nonlinearity in the convolutional layers, and the data is normalized in the model using a Keras lambda layer (code line 81).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting, and both data sets are created using the generator. The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 111).

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 in both directions.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

The solution was obtained by reading in all three images of an instance, left, right and center, and applying corrections of 0.2 to the left and right images. The main part to tune was the convolution network.

2. Final Model Architecture

The final model architecture (lines83-97) used is the Nvidea architecture from then lesson, with three dropout layers added to reduce overfitting. The architecture is a convolution network, with five convolution layers. The reason I chose this network is because it is an established network for problems like this. However, I needed to add the dropout layers as the network was overfitting.

The complete network architecture is given below. I started with the LeNet architecture, which was easy to implement as I had implemented it for an earlier project. However, the model couldn't correctly traverse the track even after hevily modifying LeNet. The method of modification was trial and error, such as including max pooling layers, dropout layers, increasing the number of dense layers (which made the model train slower and created much larger files for model.h5)

After playing around with LeNet, I decided to experiment with the NVidia architecture from the lesson, which looked more promising in the first run. It failed on the river, however, it was able to run the rest of the track with no problem. This architecture was chosen mainly because it was recommended and validated by many people across Udacity forums. Since it worked pretty much as is, I didn't need to change hyper parameters like filter size and more. However, to combat problems while driving I decided to collect more data for the architecture to learn. Finally, to fix the problem of overfitting, I implemented maxpooling layers, for which I had to change some hyperparameters in the model, and dropout layers, which were easier to implement. After the maxpooling layers, the model became worse, hence dropouts were introduced. Changing this hyperparameters, had negative effects on the model.

Without the dropout layers the car would fall into the river. This indicated overfitting, and I chose dropouts to combat the problem.

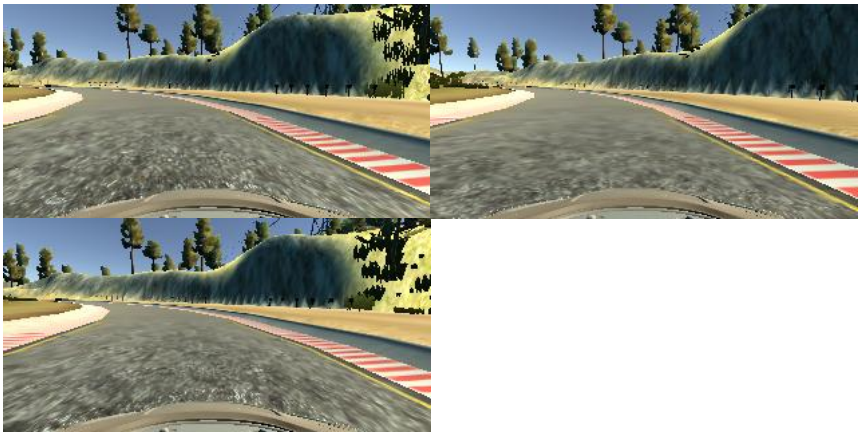
Using model.summary(), the final architecture is as follows –

Layer (type)	**Output Shape**	**Param #**	**Connected to**
lambda_1 (Lambda)	(None, 160, 320, 3)	0	lambda_input_1[0][0]
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0	lambda_1[0][0]
convolution2d_1(Convolution2D)	(None, 43, 158, 24)	1824	cropping2d_1[0][0]
convolution2d_2(Convolution2D)	(None, 20, 77, 36)	21636	convolution2d_1[0][0]
convolution2d_3(Convolution2D)	(None, 8, 37, 48)	43248	convolution2d_2[0][0]
convolution2d_4(Convolution2D)	(None, 6, 35, 64)	27712	convolution2d_3[0][0]
convolution2d_5(Convolution2D)	(None, 4, 33, 64)	36928	convolution2d_4[0][0]
flatten_1 (Flaten)	(None, 8448)	0	convolution2d_5[0][0]
dense_1 (Dense)	(None, 100)	844900	flatten_1[0][0]
dropout_1 (Dropout)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dropout_1[0][0]
dropout_2 (Dropout)	(None, 50)	0	dense_2[0][0]
dense_3 (Dense)	(None, 10)	510	dropout_2[0][0]
dropout_3 (Dropout)	(None, 10)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	11	dropout_3[0][0]

3. Creation of the Training Set & Training Process

To capture good driving behavior, I recorder 4 laps in both directions and 2-3 recovery laps after that. I had approximately 37000 total images to use to train

I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover if it were to leave the center of the road. Here is what it looks like -



As these images show. The car was taken to the edge of the road and then turned around to the center.

To augment the data sat, I also flipped images and angles to generate more data and have the car generalize going left or right, so the car would not learn the track but rather learn how to drive.

Since most of the time the car was following a straight path, I included a threshold to reduce bias for small steering angles as seen in line 29.

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 5 epochs as after 5 there was no meaningful gains.