Behavioural Cloning

The goals of this project are:

- Use the simulator to collect data of driving behavior
- Build a CNN 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
- Summarize the results with a written report

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 (added an image pre-processing function)
- **model.h5** - Containing a trained convolution neural network
- **writeup_report.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 saving the convolution neural network. The file shows the pipeline I used for training and validating the model.

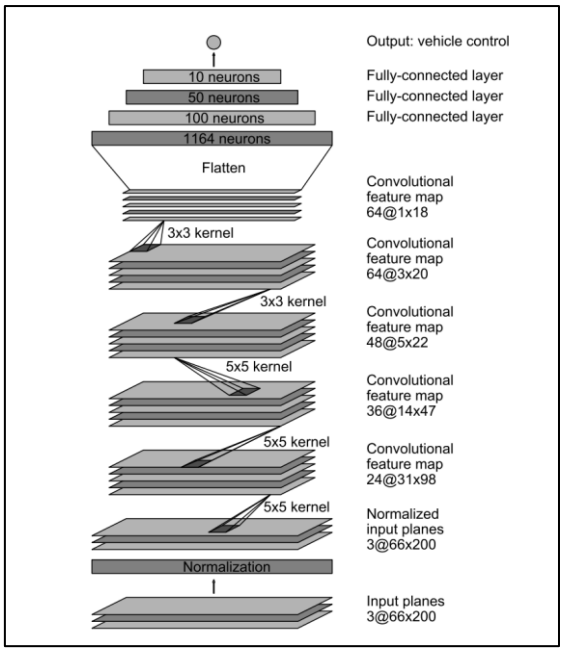
Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I used NVIDIA architecture to create a model to be trained and able to drive the car autonomously.
I used Keras Lambda function to parallelize the image normalization process. (model.py lines 79).
Lambda function can take each pixel of the image and apply normalization formula given below

Normalized pixel = pixel value / 127.5
Centered Pixel Value = Normalized pixel – 1.0

The NVIDIA CNN architecture is as shown in the figure below



The normalized data is fed into three 5x5 Convolution layer with 2x2 stride followed by two 3x3 kernel convolution layer with no strides. Before applying the dense operation, the Flatten function is applied. The Dense operation with outputs 1164, 100, 50, 10, 1 are applied. (model.py lines 80-91).

2. Attempts to reduce overfitting in the model

The model contains dropout layers with probability of 0.5 to reduce overfitting (model.py lines 86).

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py line 99-104). 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 98).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. (model.py line 46-69).

I used the combination of images from centre camera, left camera and right camera. The left and right steering angle was obtained from centre steering angle by subtracting 0.25 for left and adding 0.25 for right.

Using left and right camera input will help in steering left or right smoothly than a harsh left or right turn.

Next, I Augmented the input data by taking the flipped images and negating the input steering angle. By doing this the bias towards the left turn shall be prevented since the track is loop in anticlockwise direction.

Model Architecture and Training Strategy

1. Solution Design Approach

My first step was to use a convolution neural network model like the NVIDIA architecture.

To gauge how well the model was working, I split my image and steering angle data into a training and validation set.
The screen shot of training and validation accuracy for 5 epochs is given below

```
Epoch 1/5
6912/7028 [============================>.] - ETA: 0s - loss: 0.0376 - acc: 0.2277C:\Users\nithi\Miniconda3\envs\car
ining.py:1569: UserWarning: Epoch comprised more than `samples_per_epoch` samples, which might affect learning resu
void this warning.
  warnings.warn('Epoch comprised more than '
7104/7028 [==============================] - 60s - loss: 0.0377 - acc: 0.2272 - val_loss: 0.0312 - val_acc: 0.2240
Epoch 2/5
7104/7028 [==============================] - 62s - loss: 0.0274 - acc: 0.2266 - val_loss: 0.0262 - val_acc: 0.2375
Epoch 3/5
7104/7028 [==============================] - 64s - loss: 0.0267 - acc: 0.2279 - val_loss: 0.0231 - val_acc: 0.2271
Epoch 4/5
7104/7028 [==============================] - 63s - loss: 0.0242 - acc: 0.2280 - val_loss: 0.0189 - val_acc: 0.2510
Epoch 5/5
7104/7028 [==============================] - 63s - loss: 0.0247 - acc: 0.2286 - val_loss: 0.0233 - val_acc: 0.2442
```

The 5 epoch is chosen because if more epochs there was no further improvement in the model performance.
The last step was to run the simulator to see how well the car was driving around track one.
There were a few places where the vehicle crossed the drivable line. By collecting more data driving backwards in the track this issue was fixed.

At the end of the process, the vehicle can drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture consists of following layers

1. 5x5 kernel Convolution layer with with 2x2 stride and output depth 24
2. 5x5 kernel Convolution layer with with 2x2 stride and output depth 36
3. 5x5 kernel Convolution layer with with 2x2 stride and output depth 48
4. 3x3 kernel Convolution layer with with no stride and output depth 64
5. 3x3 kernel Convolution layer with with no stride and output depth 64
6. One Flatten Function with ouput (1152,)
7. Dropout layer with probability 0.5
8. Dense operation with output 1164
9. Dense operation with output 100
10. Dense operation with output 50
11. Dense operation with output 10
12. Dense operation with output 1

Here is a visualization of the architecture

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 66, 200, 3)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 31, 98, 24)	1824	lambda_1[0][0]
convolution2d_2 (Convolution2D)	(None, 14, 47, 36)	21636	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 22, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 3, 20, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 1, 18, 64)	36928	convolution2d_4[0][0]
flatten_1 (Flatten)	(None, 1152)	0	convolution2d_5[0][0]
dropout_1 (Dropout)	(None, 1152)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 1164)	1342092	dropout_1[0][0]
dense_2 (Dense)	(None, 100)	116500	dense_1[0][0]
dense_3 (Dense)	(None, 50)	5050	dense_2[0][0]
dense_4 (Dense)	(None, 10)	510	dense_3[0][0]
dense_5 (Dense)	(None, 1)	11	dense_4[0][0]

3. Creation of the Training Set & Training Process

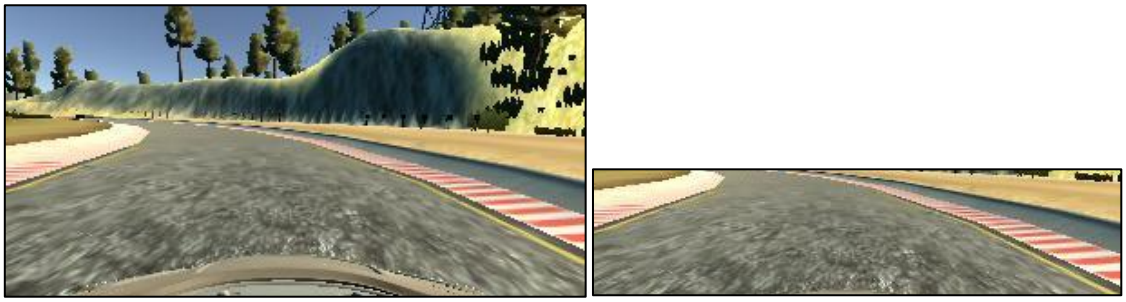
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

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 steer left or right smoothly.
These images show left, center and right camera views.

To augment the data sat, I also flipped images and angles thinking that this would prevent the left bias.
For example, here is an image that has then been flipped

After the collection process, I had 52,716 number of data points. I then preprocessed this data by

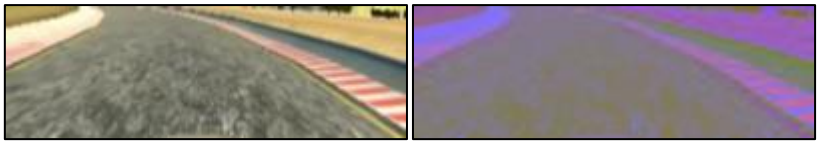
i) Cropping the image: Cropping to 75 x 320 size by removing 65 pixels of the top portion which includes sky and 20 pixels of bottom portion which includes car hood. The original and the cropped image is as shown below



ii) Applying Gaussian: Blur to make the image smooth and reduce noise. The cropped image and blurred image are as shown below

iii) Resizing the image: Image is resized to 200x66 as described by NVIDIA architecture. The Blurred image and resized image is as shown below

iv) Converting color to YUV: The image is converted to YUV scale as mentioned by NVIDIA architecture. The resized image and image converted to YUV scale is as shown below



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. The ideal number of epochs was 5 by trial and error method. I used an Adam optimizer so that manually training the learning rate wasn't necessary.