Behavioural Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behaviour.
- 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
- Summarize the results with a written report

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.
- run1.mp4 video showing trained car running on track 1.
- Behavioural Cloning Project.pdf writeup report summarizing the results.

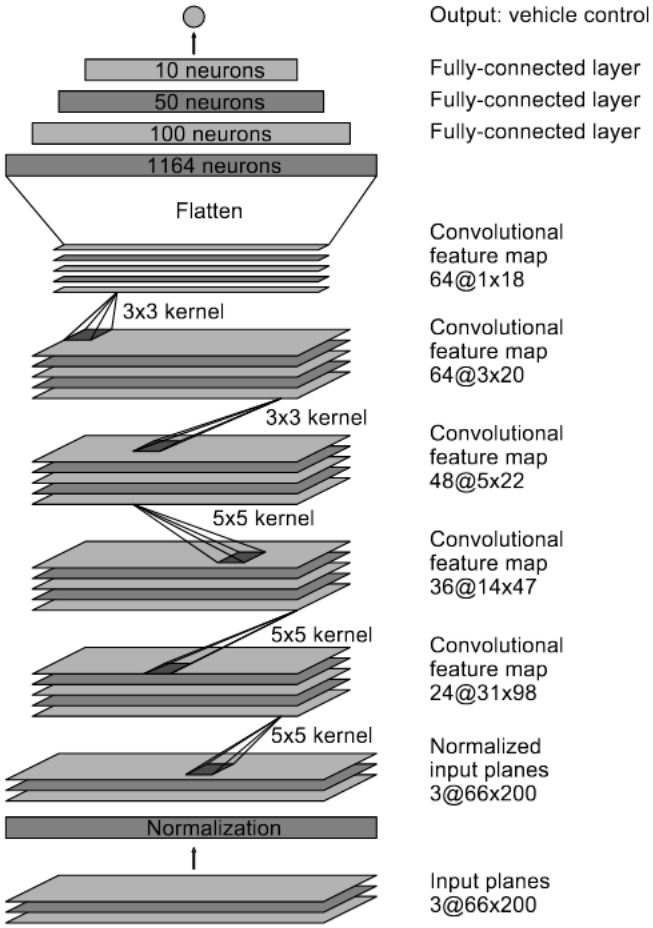
Model Architecture and Training Strategy:

1. An appropriate model architecture has been employed:

I used a modified version of the Nvidia autonomous driving model:

Layer	Output Shape
Cropping2D	160x320x3
Lambda resize	66x200x3
Lambda yuv colorspace conversion	66x200x3
Lambda Normalize	66x200x3
conv2D	31x98x24
conv2D	14x47x36
conv2D	5x22x48
conv2D	3x20x64
conv2D	1x18x64
dropout	1x18x64
flatten	1152
dense	100
dropout	100
dense	50
dense	10
dense	1

Original Nvidia Model:



Relu activations were used in the convolution layers to introduce nonlinearity. Since we reduced the problem to a regression problem , I used MSE(mean squared error) as the error function to optimise. Adam optimiser was used to train the network.

2. Attempts to reduce overfitting in the model:

The model contains 2 dropout layers to reduce overfitting with a keep_prob of 0.5 . 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.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. However, I experimented with different batch sizes and epochs for training and found the best results with a batch size of 64 and epochs set to 23.

4. Appropriate training data

Initially I tried to train the model with the training data provided by Udacity for track1.However I found that the data had left bias(due to circular track) and the model was not generalizing well. The car struggled to stay on track on more than a few occasions. Then I decided to record my own data for training. The Udacity simulator was used to record 3 laps in regular direction, 3 laps in the opposite direction and 1 lap of recovery from the left and right sides of the road. The left , right and central camera images where collected in **IMG** folder while data related to steering angle, throttle, brake and speed was captured in **driving_log.csv** file. In total 30240 images were used for training the network. For this project we are only using the steering angle while keeping other features constant.

Image from central camera

Image from left camera

Image from right camera

5. Solution Design Approach

I started off by using the Udacity provided training data and the default Nvidia model as described in their blog. A few extra layers were added to increase performance. I used a Cropping layer to crop unwanted part of the image so as to focus training only on the road.

As recommended by Nvidia developers, the images were converted to YUV color space .The images were then resized to 66x200x3 as the model expected an input of the same size. Images were then normalized and fed to a series of convolution layers to produce the prediction for steering angle. This model was trained for a few epochs and the model weights were saved in a model.h5 file. I then fed the model data to drive.py to output steering angles for driving the car in Udacity simulator in autonomous mode. While the car drove well initially ,it went off the track and into the sand. To solve the issue I created some more training data by running the car in training mode as described above. I then trained the model on the center camera images and checked the result again. I found that the car quickly drove off the track and kept driving. To combat the issue ,I decided to use the left and right camera images for training too. I added/subtracted an offset of 0.2 degree from the center camera measurements to assign measurements to left and right camera images. This way I could train the network as if all the cameras are in the front of the vehicle. On training the model by using above approach I found that the car was able to drive beautifully without going off-track. A run1.mp4 video is shared with the project files showing the car driving without any issues on track 1.

Discussion:

After few epochs of training the car did very well on the track 1 of the simulator. However when the same model was used to run car on track 2(challenging track),the car did not stay on track for very long. Track 2 has a lot of sharp turns and non-marked roads. Also the sudden steep elevations and slopes were difficult to handle for the car. It seems that to tackle such roads we need to use the remaining features in the dataset i.e. throttle, brake and speed. This will help the car to slow down on elevated and low slope roads. I did not feel the need to introduce data augmentation in this project till now. However augmented data might help to better generalize the model. A few laps of training data from track 2 will also surely help improve the model.