#Behavioral Cloning

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior
- 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

Rubric Points

###Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

###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 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.md or 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, and it contains comments to explain how the code works.

###Model Architecture and Training Strategy

####1.

An end to end learning algorithm Nvidia Dave2 architecture has been employed at first to introduce the normalization function, to initialze the image.

My model consists of 3 convolution neural network(3x3 filter size) and depths at 16/32/64 separatedly deployed then flatten the layers.

The model includes 4 Densed(full connect) layers + 3 RELU activated to introduce nonlinearity.

####2. Attempts to reduce overfitting in the model

a separate method is considered to conquer overfitting:
if the steering angle <=0.2 or >=0 at adjacent 4 frames, those images/data will be dropped. which function is introduced at the beginning of code to initialize the images pool.

####3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.
as mentioned above, the steering angle parameter to reduce redundancy and the correction factor to adjust 3 cameras respond shall be well-tuned. which will be deemed as general parameters after the whole model deployed.

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the default dataset. within which a combination of center lane driving, recovering from the left and right sides of the road included. but still I failed at some sharpest turn at my first time smooth run, so I relearned additional images and add them up to the original images pool to further train my model. for more information see the next section.
an augment technic is introduced to the training images pool, to flip the effective images and angles and yield data antomatically.

####1. Solution Design Approach
The overall strategy for deriving a model architecture was to ...
I used convolution neural network model similar to the Nvidia Dave2. I thought this model might be appropriate because I saw the papers once mentioned in the forum then I read about the contents, I assume it is a proofed state-of-art solution to combine 3 cameras input to yield a best driving performance with just lines of codes.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set by a factor 0.2. without too much effort, I got a ideal MSE at lowest level < 0.02. This implied that the model was properly set without underfitting. To combat the overfitting, I modified the model with the redundancy-removal method as former mentioned, also also count on the Relu activation in my covnet structure.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track. especially the sharpest one at the forked dirt road, in order to improve the driving behavior. I rechecked my code, and correct the colorspcase followed with a normalization term, and I crop my image a bit with the cropping function to filter out the sky and the bonnet. and even add up more training images shooted nearby that sharpest turn to get more robust manoeuvre records. to yield a more flexible model.

at last I combat a lot to try minimize my image to a proper size, to reduce the workloads for the training. at finally I realized the I'm not having an proper pipeline, which is like: resize the image -> normalize the image inside covnet -> crop image inside covnet.

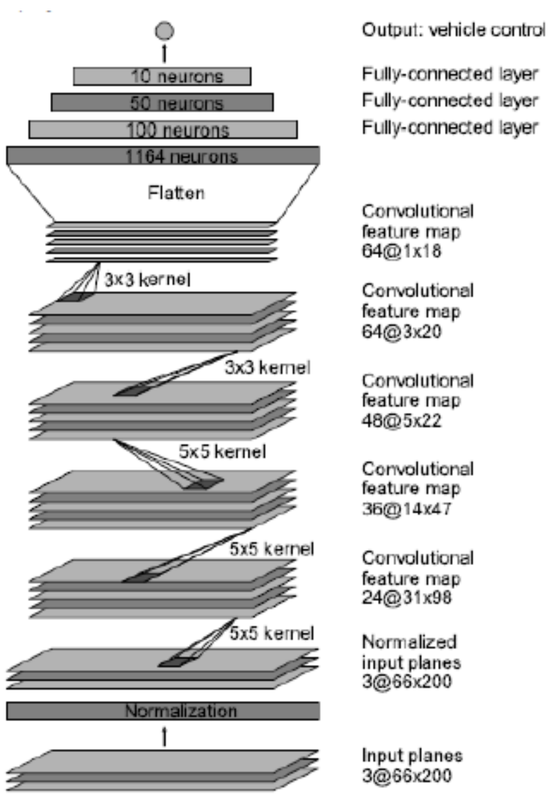
the resize term is put ahead, which will sacrifice the useful pixels before cropping, so I flip the sequency, which is like: crop the image -> resize the image -> normalize the image inside covnet. At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

the resize parameter takes loads of time to tune, I tried 160x80, 70x70, 65x65, 60x60, 40x40, 30x30, 25x25, 24x24, 22x22, 20x20 time and again, with 20x20 setup the convex becomes malfunction as such a small image size can not bear convet deeper layers. so I get the extremity at 24x24, at which my car drives good whereas ensures the h5 file at its minimum size.

####2. Final Model Architecture

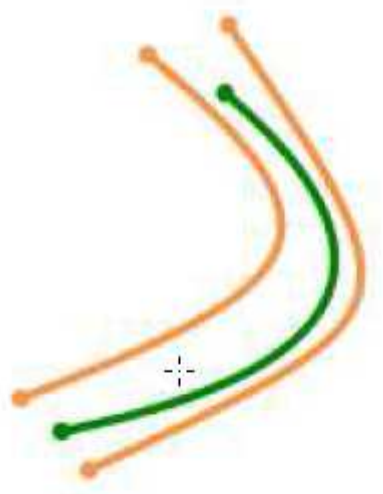
The final model architecture

The final model architecture consisted of a convolution neural network similar to below images which is copied from the parper of Nvidia pipeline. Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)



####3. Creation of the Training Set & Training Process

To capture good driving behavior, I also introduced the augment technic, to flip the existing images and corresponding angles, and record it as a new input.
for above mentioned addtional training images shooted at the specific sharpest turn, I steer to the road side then recover from side to center while driving pass the turn in around 300 frames. without a 2nd try, I just record this manoeuvre once.



green line is the car’ s track, and the other color is the edge of the sharp turn
Then I go back to train may model again and open the simulator. it succeed to ran cross the whole map.

I finally randomly shuffled the data set and put 32*2*3*6 = 9000+ of the data into a train and validation, The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 10 base on my experiences but since my model is at its smallest level, so I train 20 epochs without adding up to much computational power.