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

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
* mymodel.h5 containing a trained convolution neural network
* writeup_report.md or writeup_report.pdf summarizing the results

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
```sh
python drive.py mymodel.h5
```

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 is essentially NVIDA model with few modification though I started with LeNet first. The NVDIA model consists of a convolution neural network with several 3x3 filter sizes and depths between 32 and 128 (model.py lines 63-79)

```
    # normalization
    model.add(Lambda(lambda x: (x / 255 - 0.5),
input_shape=(160,320,3)))
    # crops at top and bottom, output shape = (75, 320, 3)
    model.add(Cropping2D(cropping=((CROPTOP,CROPBOTTOM), (0,0)),
input_shape=(160,320,3)))

    # convolutional layers
    model.add(Convolution2D(24,5,5,subsample=(2,2),activation="relu"))
    model.add(Convolution2D(36,5,5,subsample=(2,2),activation="relu"))
    model.add(Convolution2D(48,5,5,subsample=(2,2),activation="relu"))
    model.add(Convolution2D(64,3,3,activation="relu"))
    model.add(Convolution2D(64,3,3,activation="relu"))

    # flattening
    model.add(Flatten())

    # fully connected layers with dropouts
    model.add(Dense(100))
    model.add(Dropout(0.3))
    model.add(Dense(50))
    model.add(Dropout(0.3))
    model.add(Dense(10))
    model.add(Dropout(0.3))
    model.add(Dense(1))
```

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 58).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 74, 76, 78).

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 (model.py line 81).

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 ...

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to first use LeNet and see how it behaves, play with some hyperparameters to get a feel of how things can improve and try to see what max fit I can get.

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.

To combat the overfitting, I modified the model so that it can have few dropout layers

Once I could see I am going getting results I want – full loop driving autonomously, I switched to NVDIA model and played with few hypermaters to get it to work.

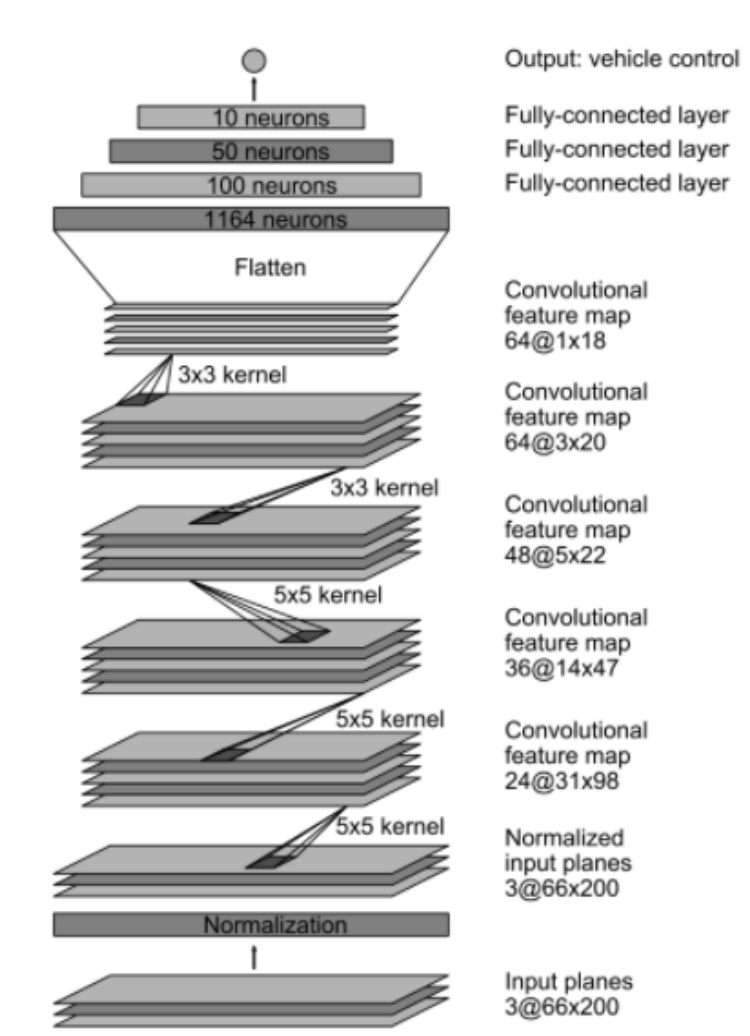
The final step was to run the simulator to see how well the car was driving around track one.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

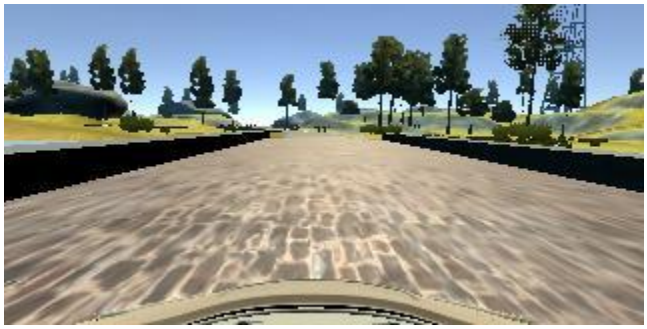
The final model architecture (model.py lines 63-79) consisted of a convolution neural network with the following layers and layer sizes ...

Here is a visualization of the architecture

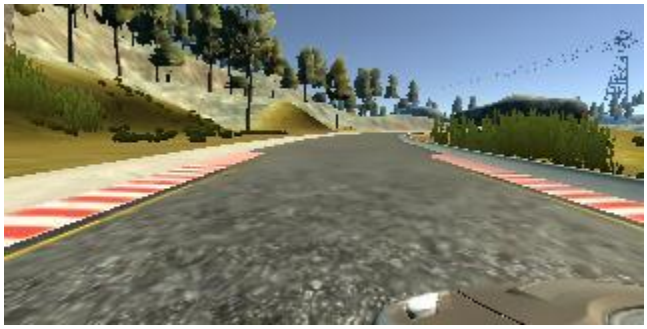


3. Creation of the Training Set & Training Process

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 .... These images show what a recovery looks like starting from ... :



Then I repeated this process on track two in order to get more data points.

To augment the data sat, I also flipped images and angles thinking that this would create more data for neural network to learn from.

After the collection process, preprocessed this data by cropping the area dominated by sky (upper part) 60 pixel and bottom 20 pixels mostly car image itself in the image.

I finally randomly shuffled the data set and 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 as evidenced by ... I used an adam optimizer so that manually training the learning rate wasn't necessary.