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

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

```sh
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 appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 filter sizes and depths between 24 and 48 (model.py lines 90-92), and a convolution neural network with 3x3 filter sizes and depths between 48 and 64 (model.py lines 93-94)

The model includes RELU layers to introduce nonlinearity (code line 90-94), and the data is normalized in the model using a Keras lambda layer (code line 88).

The model includes Fully Connected layers(code lines 96-100).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 95).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 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 a adam optimizer, so the learning rate was not tuned manually (model.py line 103).

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

My first step was to use a convolution neural network model similar to the LeNet. I thought this model might be appropriate because LeNet works well on traffic sign classifer project.

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 turned to the nVidia Autonomous Car Group Mode which was designed to train a real car to drive autonomously.

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, to improve the driving behavior in these cases, I flipped the images and steering angles to balance the training data.

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

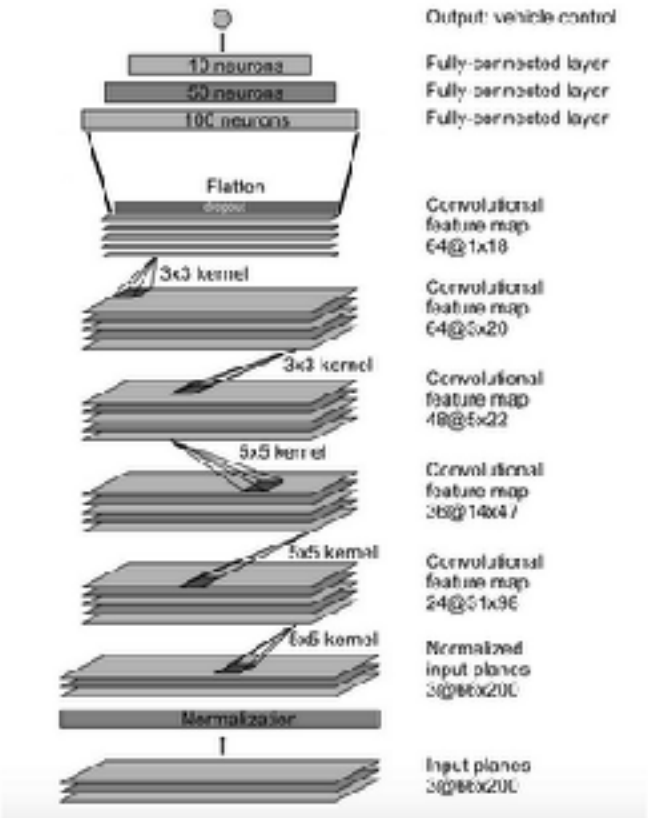
2. Final Model Architecture

The final model architecture (model.py lines 87-100) consisted of a convolution neural network with the following layers and layer sizes:

Layer	Description
Input	160x320x1 Gray Image

Lambda (x/255 - 0.5)	Output 160x320x1
Cropping2D	Output 90x320x1
Convolution2D 5x5	Output 43x158x24, Activation relu
Convolution2D 5x5	Output 20x77x36, Activation relu
Convolution2D 5x5	Output 8x37x48, Activation rRelu
Convolution2D 3x3	Output 6x35x64, Activation relu
Convolution2D 3x3	Output 4x33x64, Activation relu
Flatten	Output 8448
Dense	Output 120
Dense	Output 50
Dense	Output 10
Dense	Output 1

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)



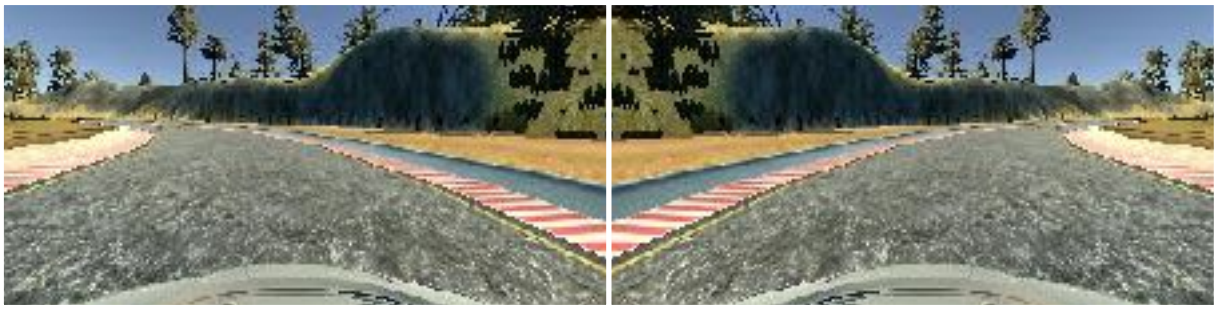
3. Creation of the Training Set & Training Process

To capture good driving behavior, I 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 recover to center lane road. These images show what a recovery looks like:

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 help the model to generalize better. For example, here is an image that has then been flipped:



After the collection process, I had X number of data points. I then preprocessed this data by converting the image color into grayscale, and using EqualizeHist to improve the image contrast.

I finally randomly shuffled the data set and put 30% 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 an adam optimizer so that manually training the learning rate wasn't necessary.