Behavioral Cloning

Writeup Template

You can use this file as a template for your writeup if you want to submit it as a markdown file, but feel free to use some other method and submit a pdf if you prefer.

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

[//]: # (Image References)

[image1]: ./examples/placeholder.png "Model Visualization"
[image2]: ./examples/placeholder.png "Grayscaling"
[image3]: ./examples/placeholder_small.png "Recovery Image"
[image4]: ./examples/placeholder_small.png "Recovery Image"
[image5]: ./examples/placeholder_small.png "Recovery Image"
[image6]: ./examples/placeholder_small.png "Normal Image"
[image7]: ./examples/placeholder_small.png "Flipped Image"

Rubric Points
Here I will consider the [rubric points](https://review.udacity.com/#!/rubrics/432/view) 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
```sh
python drive.py model.h5
```

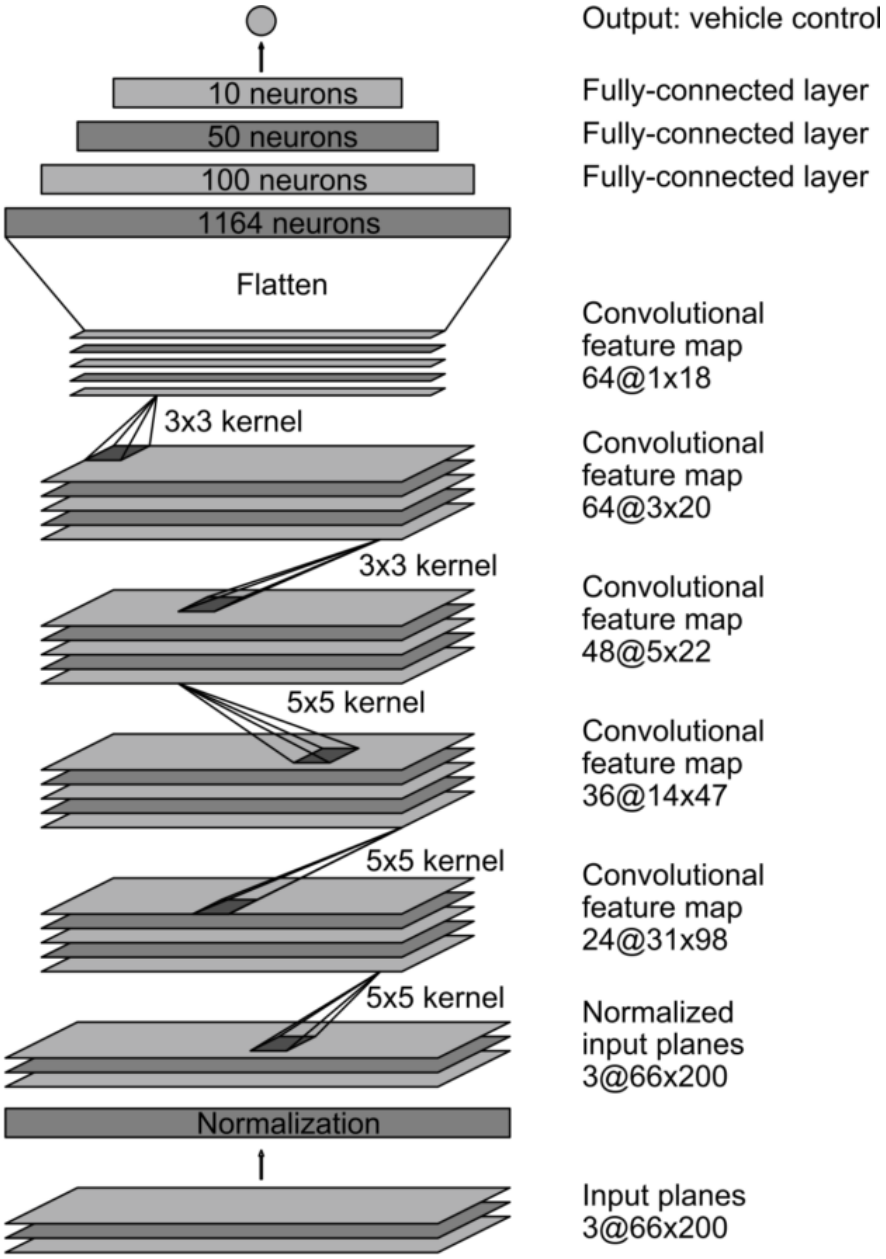
```

#### 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

For the actual network architecture, the Nvidia model is used because of its simplicity and demonstrated ability to perform well on self-driving car tasks. The architecture is slightly modified to add batch normalization layers instead of Dropout before ReLU activations are applied.
My model consists of 5 convolution layers with batch normalization follwed by three fully connected layers.I am using MSE as loss function and Adam as gradient optimizer.

#### 2. Attempts to reduce overfitting in the model
In order to prevent overfitting I drove my car around the track for almost 20 minutes and collected more than 500MB of data for training and validation set. I am also applying Batch Normalization for all convolution as well as fully connected layers for my model.

#### 3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

#### 4. Appropriate training data

Training data was chosen to keep the vehicle driving middle 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 ...

My first step was to use a simple convolution neural network with 3 convolution layers and 2 fully connected layers for regression.
But I found that I needed to go deeper so I ended up choosing almost the same architecture as given by Nvidia DAVE-2 System research paper.
I thought this model might be appropriate because it was successful for Nvidia so I was really interested to implement the same architecture for my model.

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 by applying Batch Normalization.

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 so I gathered training data again and trained my second time with more 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

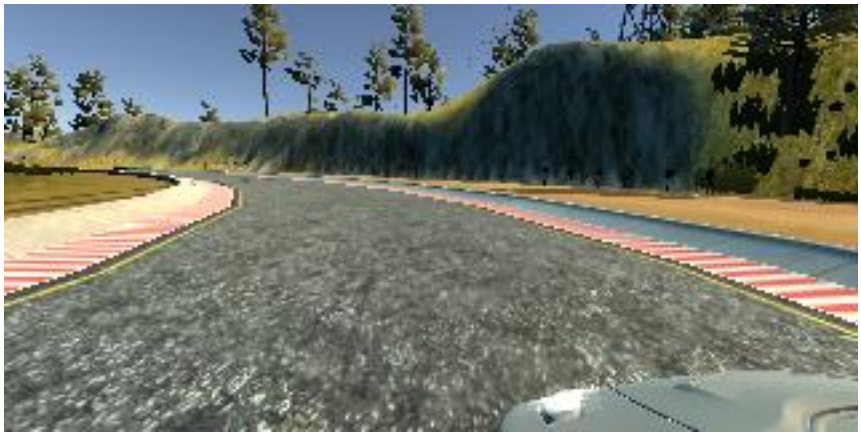
My final model consists of 5 convolution layers followed by three fully connected layers with batch normalization for faster learning and higher accuracy. I am using MSE as loss function and Adam as gradient optimizer.



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 itself in case it deviates from its path.



Left Sided View

Right Sided View

Center View

To augment the data sat, I also flipped images and angles thinking that this would provide better symmetric view.

After the collection process, I had 18081 number of data points. I then preprocessed this data by merging all image paths in one list and putting angle measurements in another list.

I finally randomly shuffled the data set and put 20% of the data into a validation set and 80% in training 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 3 as evidenced while training.
I used the Adam optimizer so that manually training the learning rate wasn't necessary.