#**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](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

My model consists of a convolution neural network with 5x5,and 3x3 filter sizes and depths between 24 and 64 (model.py lines 83-96)

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 81).

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

The model was trained and validated on different data sets (code line 71) and the model went through dropout to avoid 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 105).

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used the sample data provided by Udacity

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 watch the loss and the validation loss then try on the simulator to watch the performance.

My first step was to use a convolution neural network model similar to the Nvidia. I thought this model might be appropriate because it worked well with Nivida self-driving car and was also recommended by Udacity project module.

To combat the overfitting, I modified the model so that it has validation data that are different from the training data and watch the loss on both sets and made dropout layers even in between convolutional layers using SpatialDropout2D.

Then I cropped the image to make the model only sees the road.

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 multiplied the amount of the data (make the model train on the data multiple times)

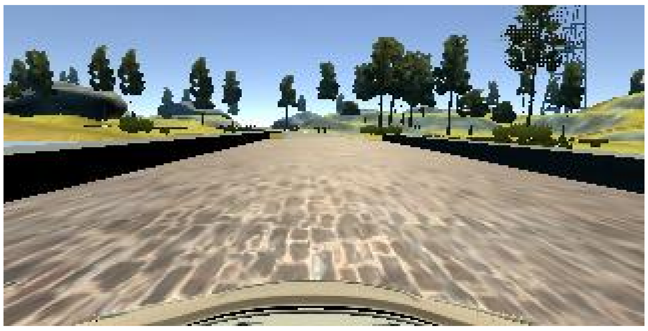
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

####2. Final Model Architecture

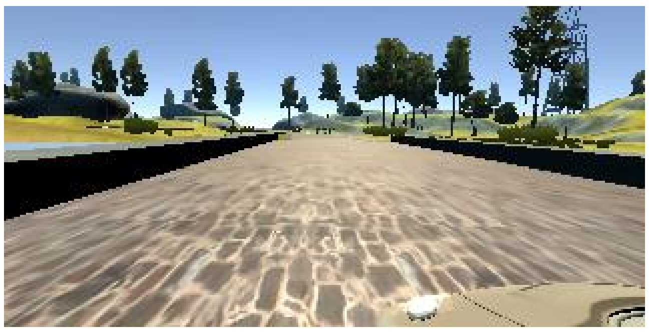
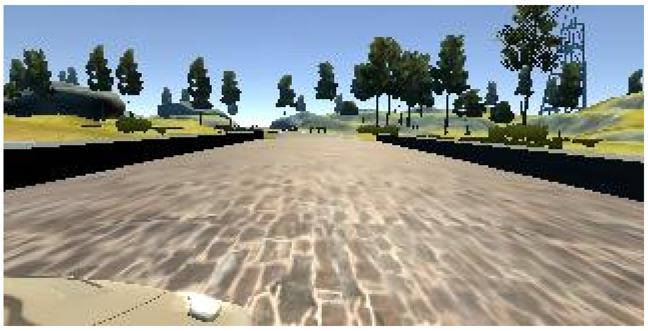
The final model architecture (model.py lines 74-85) consisted of a convolution neural network with the following layers and layer sizes …
1- 24 depth (hxw)=(5x5) subsample=(2,2)
1- 36 depth (hxw)=(5x5) subsample=(2,2)
1- 48 depth (hxw)=(5x5) subsample=(2,2)
1- 64 depth (hxw)=(3x3)
1- 64 depth (hxw)=(3x3)

####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:

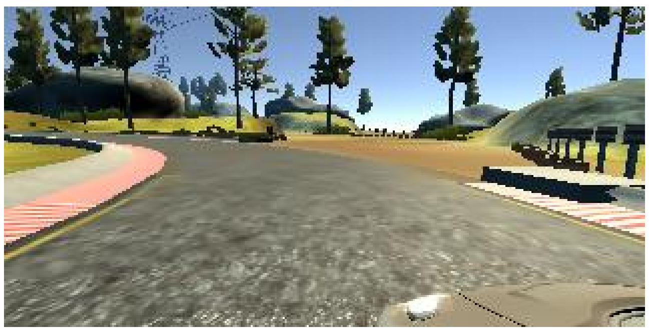
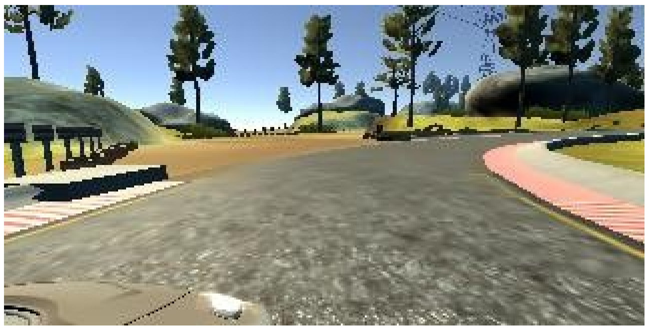


I then included the images fro the left and right cameras so that the vehicle would learn to recover from the sides of the road the following images are from the left and right cameras respectively.

Then I repeated this process by copying the data.

To augment the data sat, I also flipped images thinking that this would create a more generalized data and balance the left turnings with the right turnings For example, here is an image that has then been flipped:

Etc ....

After the collection process, I had 96420 number of data points. I then preprocessed this data by ...
1- cropping the images to focus only on the road
2- normalize the input

I finally randomly shuffled the data set and put 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 chosen by the “EarlyStopping” callback function. I used an adam optimizer so that manually training the learning rate wasn't necessary.