#**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.pdf summarizing the results
* video.mp4 output of running the simulator in autonomous mode.

####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 4 convolution neural networks with 32, 20, 15,10 filters with 3x3 kernel size and depths between 32 and 128 and 4 fully connected layers including the output layer (model.py lines 84-127)

The model includes RELU layers to introduce nonlinearity (code line 84-127), and the data is normalized in the model using a Keras lambda layer (code line 82).

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

The model contains dropout layers and max pool layers in order to reduce overfitting (model.py lines 84-127).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code lines 31, 68-69). 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 130).

####4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, and recovering from the left and right sides of the road to enable the network to make a decision when run into such situations.

###Model Architecture and Training Strategy

####1. Solution Design Approach

The overall strategy for deriving a model architecture was to achieve a best model to drive the car in autonomous mode.

My first step was to use 3 convolutional neural networks model similar to the discussion in the course, one for edges and second for shapes and third for higher level details. I thought this model might be appropriate as most of the problems during human recognition goes to similar process.

I had taken care of overfitting by adding dropout and maxpool layers

And I have 3 fully connected layer followed by an output layer
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 an average mean squared error on the training set and validation set.

I have experimented with the layer sizes and left and right image, measurement correction and finally decided that 4 convolution layers are performing better.

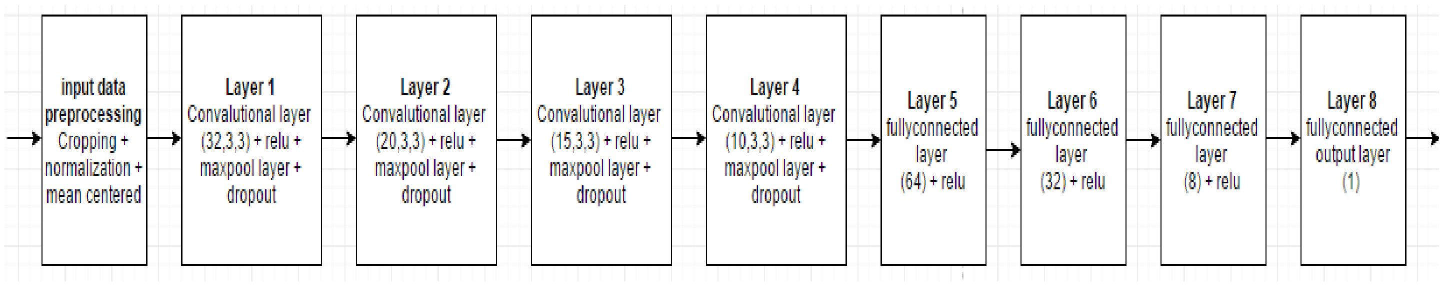
The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track and to improve the driving behavior in these cases, I adjusted the left and right measurement correction.

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

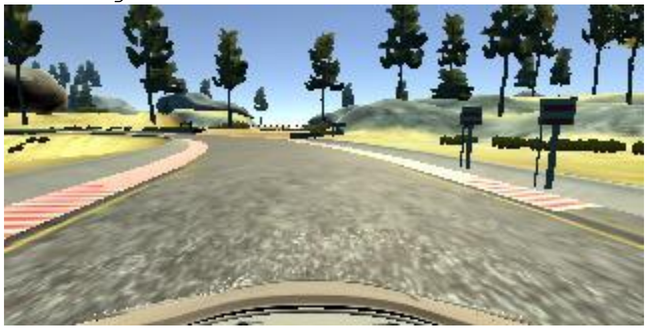
####2. Final Model Architecture

The final model architecture (model.py lines 84-127) consisted of 4 convolution neural networks with 32, 20, 15,10 filters with 3x3 kernel size with relu activation, maxpooling and dropout after every layer and depths between 32 and 128 and 4 fully connected layers including the output layer.

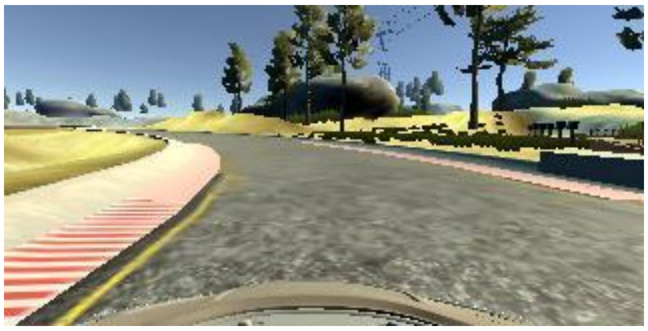
Here is a visualization of the architecture

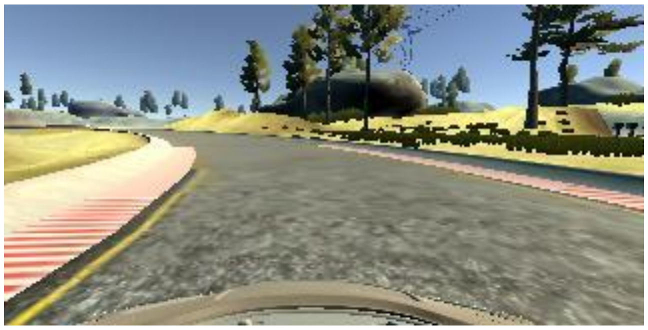

input data preprocessing Cropping + normalization + mean centered
Layer 1 Convalutional layer (32,3,3) + relu + maxpool layer + dropout
Layer 2 Convalutional layer (20,3,3) + relu + maxpool layer + dropout
Layer 3 Convalutional layer (15,3,3) + relu + maxpool layer + dropout
Layer 4 Convalutional layer (10,3,3) + relu + maxpool layer + dropout
Layer 5 fullyconnected layer (64) + relu
Layer 6 fullyconnected layer (32) + relu
Layer 7 fullyconnected layer (8) + relu
Layer 8 fullyconnected output layer (1)

####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 do the same. These images show what a recovery looks like starting from left to center again:





After the collection process, I had 3626 number of data points. I then preprocessed this data by including the flipped images and left, right measurement corrected images. After all the processing, I had 14504 (4x3626) number of data points.

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. I used an adam optimizer so that manually training the learning rate wasn't necessary.