Behavioral Cloning

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 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
* behavioral_cloning_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

```
$ > 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. Preprocssing data

I collected my own training data using the simulator provided by Udacity. Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving(2/3 laps), recovering from the left and right sides of the road to the center, collected data from sharp curves specifically and driving car counter clockwise manually few laps on track 1.

During trial/testing I collected data from track 2 and combined with earlier data, this didn't give me better results.
I also tried using data provided by Udacity combined with recovery data, didn't help much.

2. An appropriate model architecture has been employed

The cameras in the simulator capture 160 pixel by 320 pixel images. We do not want to feed noise such as sky, hills, hood of the car to the network so I use Cropping2D layer from Keras to crop 70 rows pixels from top and 25 rows pixels from bottom.

In Keras, lambda layers can be used to create arbitrary functions that operate on each image as it passes through the layer. lambda layer is used for image normalization

train/valid split
80% of the data is separated for training and 20% for validation

Basic idea is network takes in images from center, left and right camera and outputs a new steering angle for the car.
Taking images from all 3 cameras can teach the model how to steer if the car drifts off to the left or the right.

I first tried my own simple architecture with 2 convolution layers followed by pooling and 2 fully connected layers and an output layer. This didn't give good results. Second architecture I tried is based on NVIDIA model. It looks like this.

- Image normalization
- Cropping
- Convolution: 5x5, filter: 24, strides: 2x2, activation: RELU
- Convolution: 5x5, filter: 36, strides: 2x2, activation: RELU
- Convolution: 5x5, filter: 48, strides: 2x2, activation: RELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: RELU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: RELU
- Fully connected: neurons: 100
- Fully connected: neurons: 50
- Fully connected: neurons: 10
- Fully connected: neurons: 1 (output)

Final model summary:

```
______________________________________________________________________
______________________________
Layer (type)                      Output Shape          Param #
Connected to
======================================================================
==============================
lambda_1 (Lambda)                 (None, 160, 320, 3)   0
lambda_input_1[0][0]
______________________________________________________________________
______________________________
cropping2d_1 (Cropping2D)         (None, 65, 320, 3)    0
lambda_1[0][0]
______________________________________________________________________
______________________________
convolution2d_1 (Convolution2D)  (None, 31, 158, 24)   1824
cropping2d_1[0][0]
______________________________________________________________________
______________________________
convolution2d_2 (Convolution2D)  (None, 14, 77, 36)    21636
convolution2d_1[0][0]
______________________________________________________________________
______________________________
convolution2d_3 (Convolution2D)  (None, 5, 37, 48)     43248
convolution2d_2[0][0]
______________________________________________________________________
______________________________
convolution2d_4 (Convolution2D)  (None, 3, 35, 64)     27712
convolution2d_3[0][0]
______________________________________________________________________
______________________________
convolution2d_5 (Convolution2D)  (None, 1, 33, 64)     36928
convolution2d_4[0][0]
______________________________________________________________________
______________________________
flatten_1 (Flatten)               (None, 2112)          0
convolution2d_5[0][0]
______________________________________________________________________
______________________________
dense_1 (Dense)                   (None, 100)           211300
flatten_1[0][0]
______________________________________________________________________
______________________________
dense_2 (Dense)                   (None, 50)            5050
dense_1[0][0]
______________________________________________________________________
______________________________
dense_3 (Dense)                   (None, 10)            510
dense_2[0][0]
______________________________________________________________________
```

```
____________________________________
dense_4 (Dense)                  (None, 1)            11
dense_3[0][0]
================================================================================
====================================
Total params: 348,219
```

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer.
I tried using dropout layers with keep_prob 0.5 but car was all wobbly and unstable. Dropout didn't produce better results in my case. I did a lot of trial/test with batch_size and epoch_size with variations in amount of training data.

3. Model parameter tuning

Adam optimizer with a learning rate of 0.001(deafault).
epoch_size : 6
batch_size : 32
Correction factor for steering left/right angle : 0.25

4. Attempts to reduce overfitting in the model

I tried using dropout layers with keep_prob 0.5 but car was all wobbly and unstable. Dropout didn't produce better results in my case.

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.

5. Image Augmentation:

Since the track is left turn bias, flipping images and taking the opposite sign of the steering measurement helped with it. For example:

Image from center camera

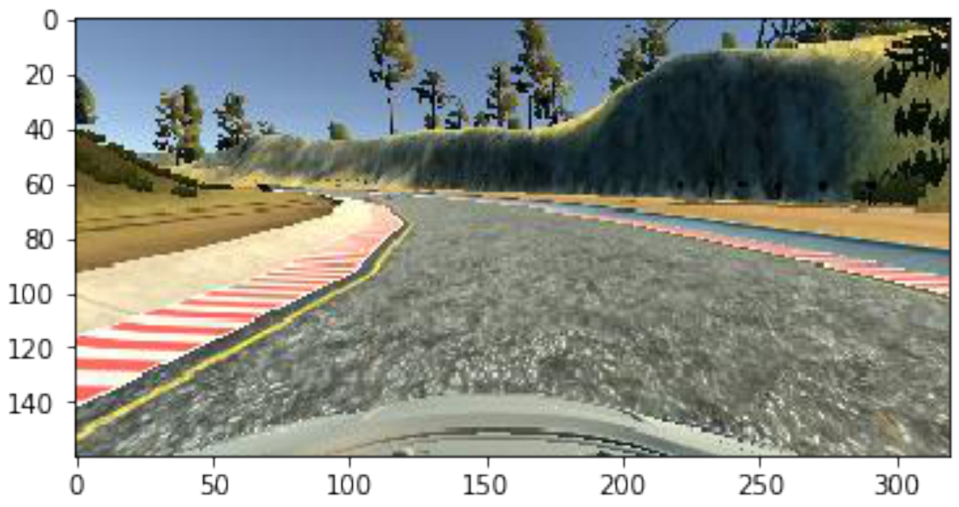


Image from left camera

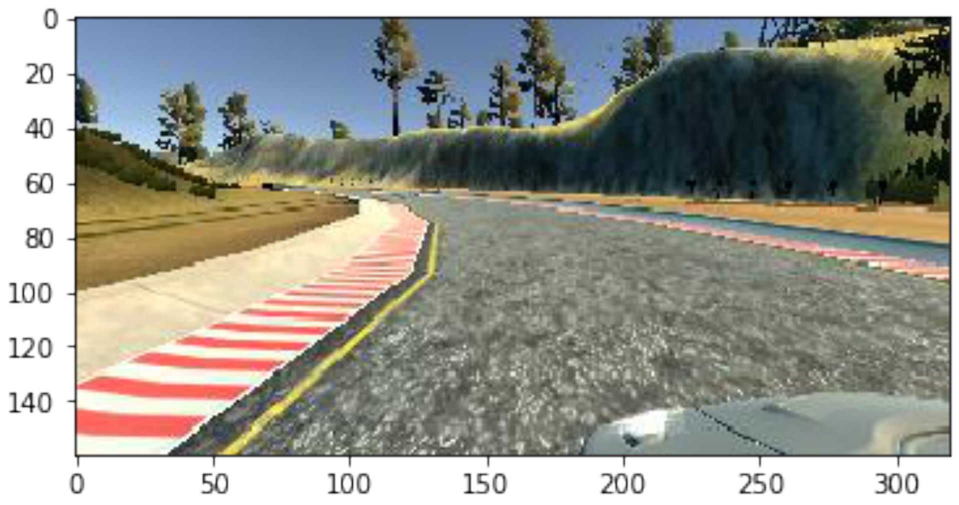
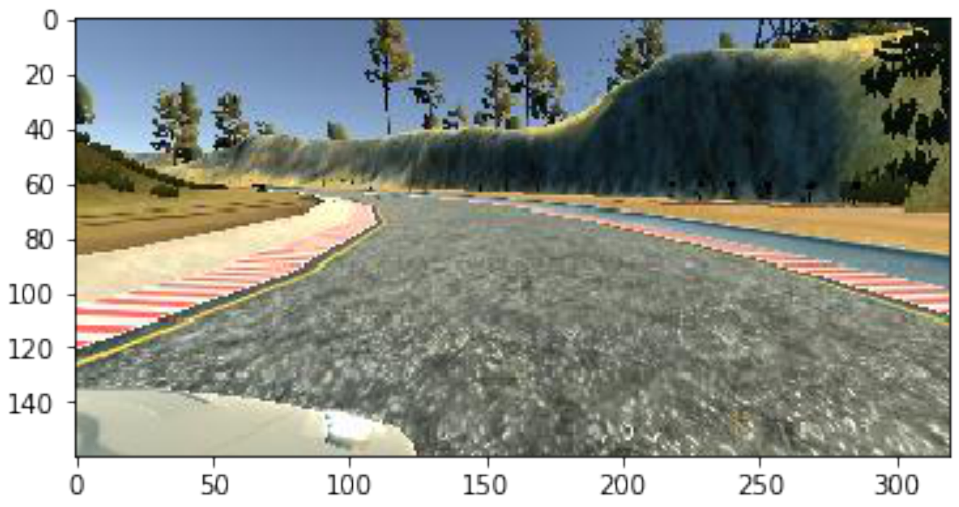
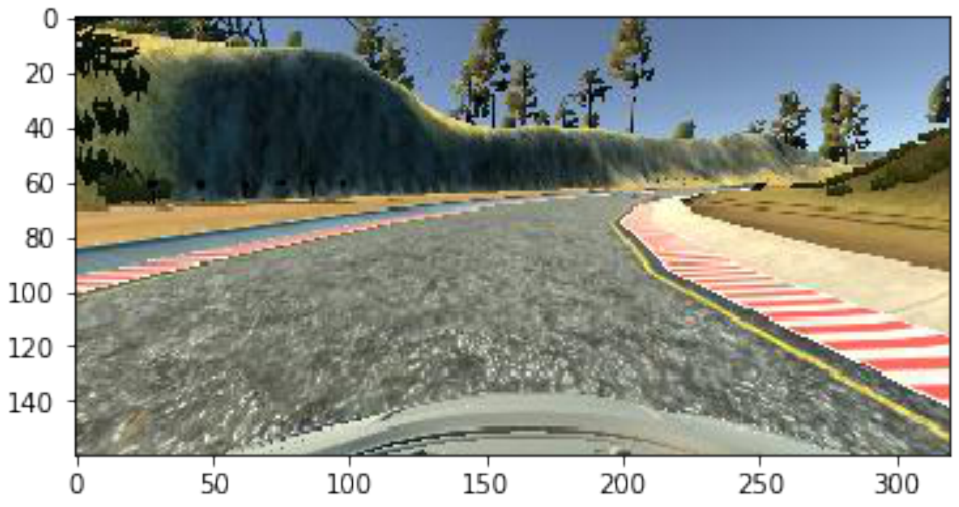


Image from right camera



Flipped



Todo : I want to continue working on data augmentation generating new images with methods like adding random brightness, transforming images, resizing, adding random shadows etc and spend more time on generalizing network so that it works on second/ challenge track as well.