Behavioral Cloning

Writeup

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

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:

- | |____final_run.mp4 - Video file for submission
- | |____model.h5 - Model for submission
- | |____Model.ipynb - The main python notebook with code
- | |____model.json - Model's JSON for submission
- | |____model.py - The code to train model derived from python notebook
- | |____dataAugmentation.py - The file to augment the data before we run model.py to train the model
- | |____ScreenRecording.mp4 - The SCREEN RECORDING of my execution at 30fps
- | |____video.py - Udacity's video.py
- | |____drive.py - Udacity's drive.py modified to suite model.h5 input format
- | |____Writeup.ipynb - The Writeup for submission
- | |____Writeup.pdf - The Writeup for submission

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 is structured.

The `dataAugmentation.py` file contains code to augment the data before we run `model.py`

The Entire code is present in `Model.ipynb` and that is the source of truth. I worked from a notebook. For the submissition I created the above Python files

Model Architecture and Training Strategy

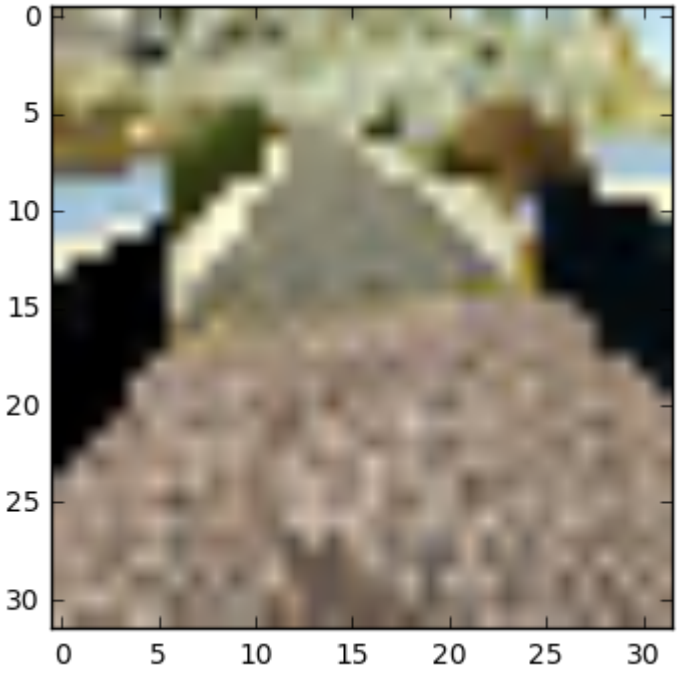
1. An appropriate model architecture has been employed

My Model summary is as follows

```
____________________________________________________________________________
____________________________________
Layer (type)                     Output Shape          Param #     C
onnected to
============================================================================
================================
lambda_16 (Lambda)               (None, 32, 32, 3)     0           l
ambda_input_16[0][0]
____________________________________________________________________________
____________________________________
convolution2d_15 (Convolution2D) (None, 31, 31, 3)     39          l
ambda_16[0][0]
____________________________________________________________________________
____________________________________
flatten_16 (Flatten)             (None, 2883)          0           c
onvolution2d_15[0][0]
____________________________________________________________________________
____________________________________
dense_31 (Dense)                 (None, 512)           1476608     f
latten_16[0][0]
____________________________________________________________________________
____________________________________
dropout_18 (Dropout)             (None, 512)           0           d
ense_31[0][0]
____________________________________________________________________________
____________________________________
dense_32 (Dense)                 (None, 1)             513         d
ropout_18[0][0]
============================================================================
================================
Total params: 1,477,160
Trainable params: 1,477,160
Non-trainable params: 0
____________________________________________________________________________
____________________________________
```

```
    model.add(Lambda(lambda x: x/127.5 - 1., input_shape=(row,col,c
h)))
    model.add(Convolution2D(3, 2, 2))
    model.add(Flatten())
    model.add(Dense(512, activation='elu'))
    model.add(Dropout(0.5))
    model.add(Dense(1))
```

My model consists of a convolution neural network with 2x2 3 filters. I selected this small amount because my input is a small 3*232. Also the number 3 because my input is a 3channel input and the model can choose different R,G,B channels as appropriate. The 32*32*3* input helps us during runtime as its faster to process less number of pixels. Below is an example of what the network sees



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

A Dropout is introduced to give more redundant neurons. It is also helps in generalizing the network.

I went with a simple network with less parameters. This is because as the number of parameters grow the data required grows too. I wanted to accomplish the task with simple networks. This also gives added benifit of being faster at run time.

2. Attempts to reduce overfitting in the model

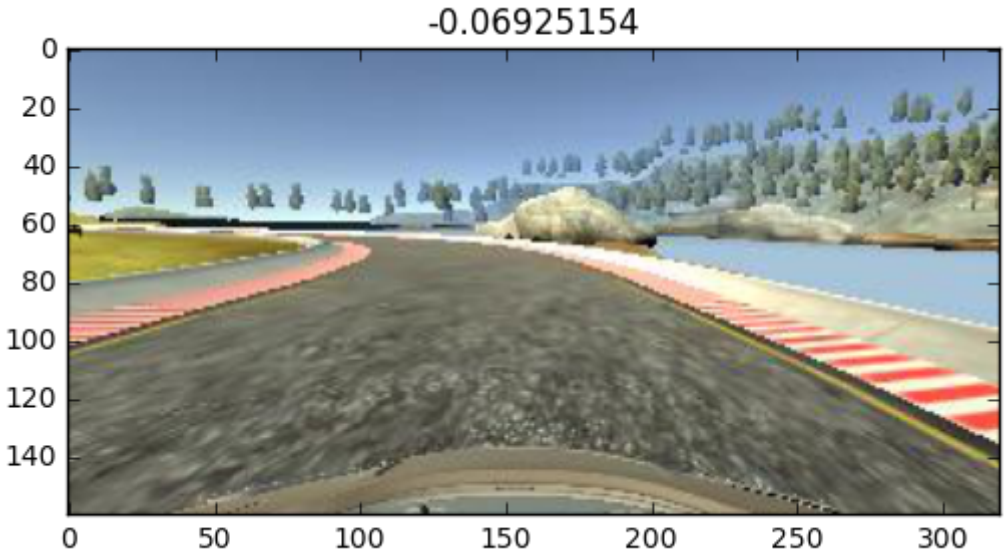
The model contains dropout layers in order to reduce overfitting as shown above

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.

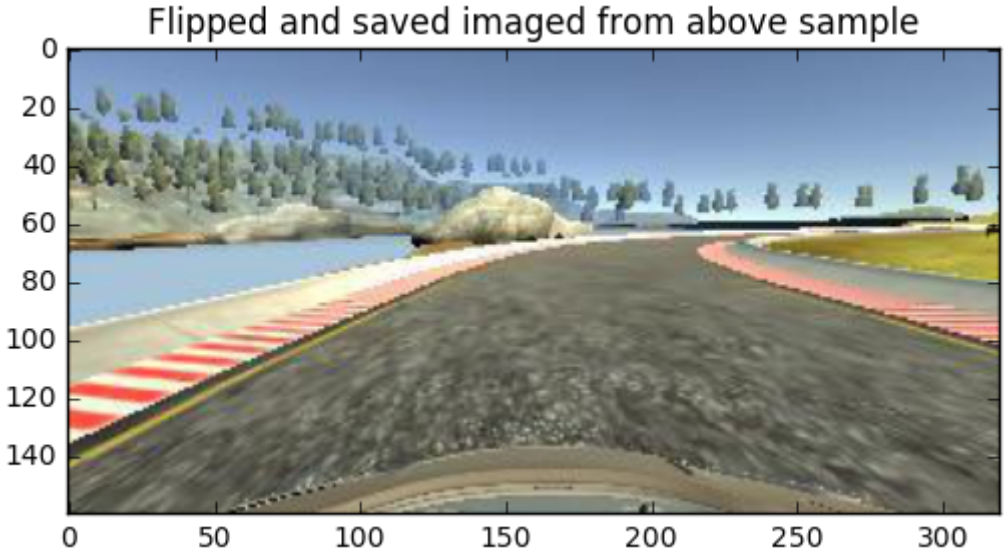
Overfitting was avoided by many methods

1. I flipped all the images, to avoid left oversteer
2. I shifted the images randomly between -50 to +50 px @0.004angle/px steer.

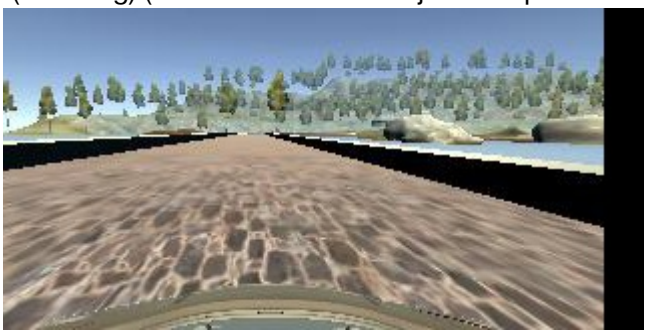
Example image



Example image Flipped



Example of a shift +20px (0.08 deg) (not related to above. just independent example)



3. Model parameter tuning

The model used an adam optimizer with mean squared error loss function. I choose some of its parameters. especially the epsilon parameter helped me by reducing it.

```
model.compile(optimizer=Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsi
lon=1e-08, decay=0.0), loss='mse')
```

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

The overall strategy for deriving a model architecture was to have a simple Conv network which can learn and generalize.I didnot want to use any existing architectures and wanted to find a simpler one. Many people have got good results with VGG and comma.ai architecture. I took insipiration from these and stripped down the networks to have less parameters.

My first step was to use a convolution neural network model similar to the VGG. I thought this model might be appropriate because my peers have had success. Also there are easily available pretrainined networks. However I quickly realized that the model is too complex for the limited amount of data.

I used UDACITY's provided data. However i found that this didnot handle well corners. So I collected my own data for corners and recovery in different sets.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set.

To combat the overfitting, I modified the model so that it has Lamda which normalizes input & drop outs & RLU for complex function

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 collected more data in recovery for sharp turns if vehical goes out.

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

2. Final Model Architecture

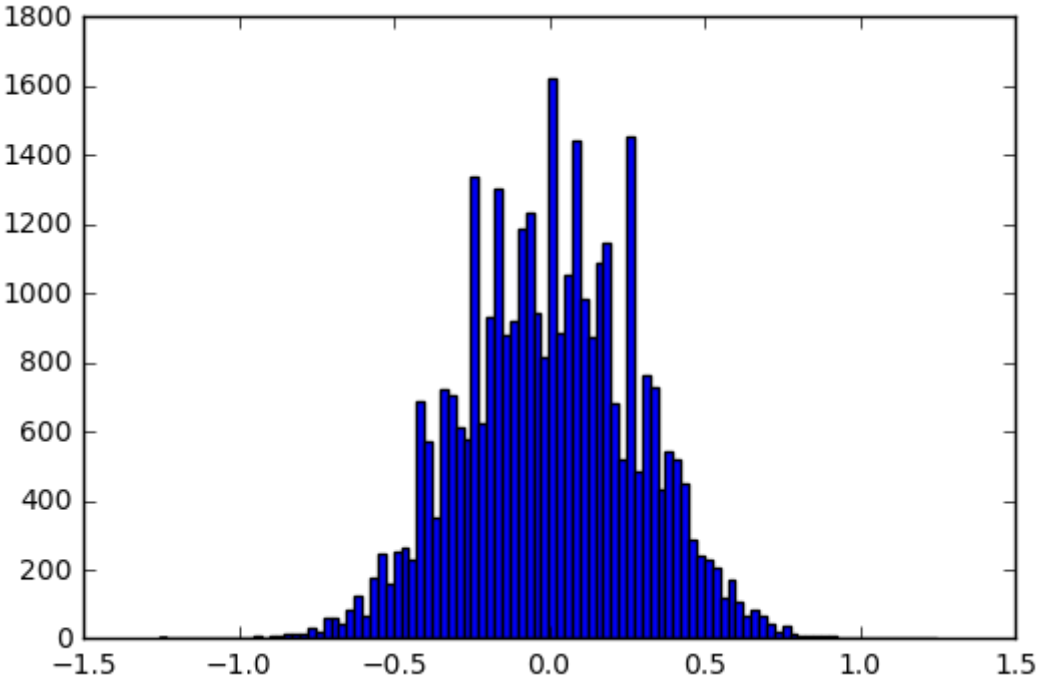
```
____________________________________________________________________________
____________________________________
Layer (type)                     Output Shape          Param #     C
onnected to
============================================================================
================================
lambda_16 (Lambda)               (None, 32, 32, 3)     0           l
ambda_input_16[0][0]
____________________________________________________________________________
____________________________________
convolution2d_15 (Convolution2D) (None, 31, 31, 3)     39          l
ambda_16[0][0]
____________________________________________________________________________
____________________________________
flatten_16 (Flatten)             (None, 2883)          0           c
onvolution2d_15[0][0]
____________________________________________________________________________
____________________________________
dense_31 (Dense)                 (None, 512)           1476608     f
latten_16[0][0]
____________________________________________________________________________
____________________________________
dropout_18 (Dropout)             (None, 512)           0           d
ense_31[0][0]
____________________________________________________________________________
____________________________________
dense_32 (Dense)                 (None, 1)             513         d
ropout_18[0][0]
============================================================================
================================
Total params: 1,477,160
Trainable params: 1,477,160
Non-trainable params: 0
____________________________________________________________________________
____________________________________
```

3. Creation of the Training Set & Training Process

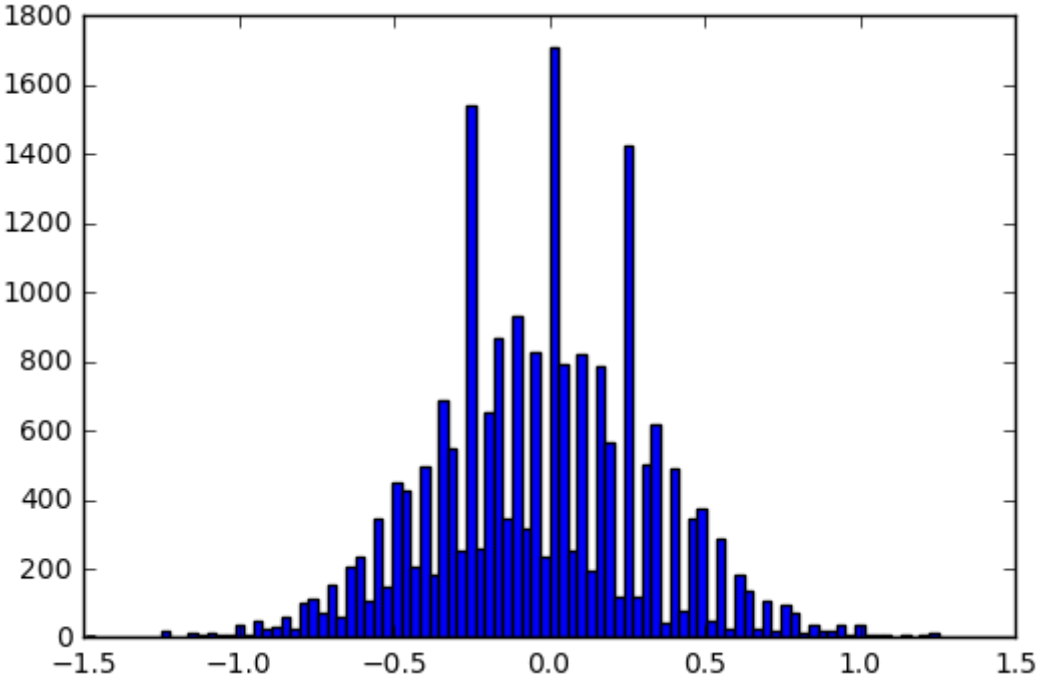
I used

A) Udacity's data.

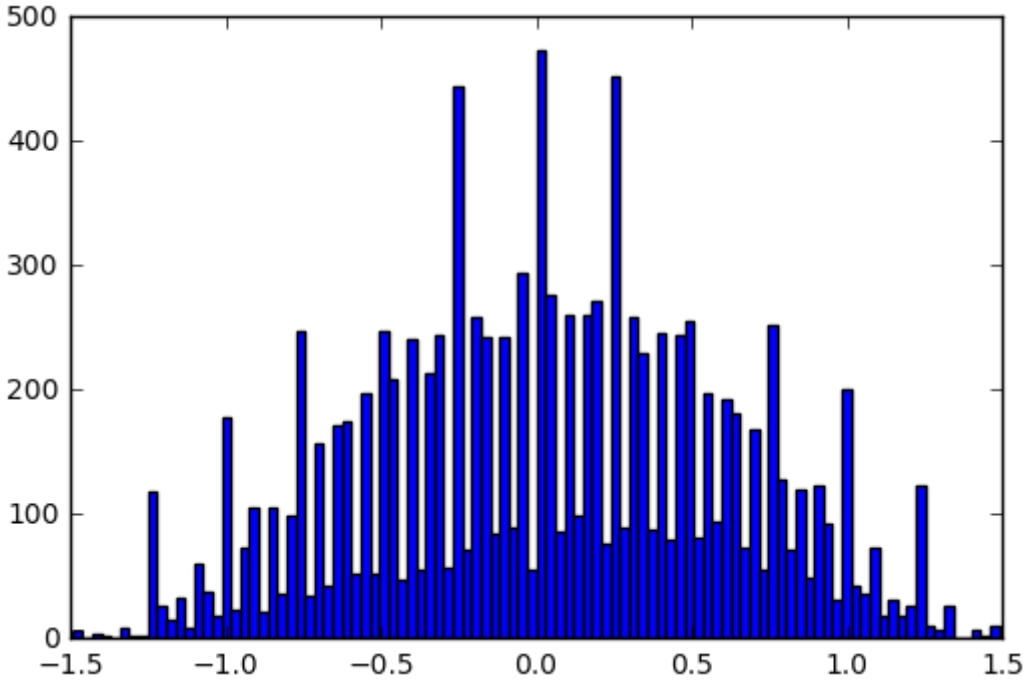
Number of rows = 33867



B) My Own Data with data around corners. Number of rows = 21555

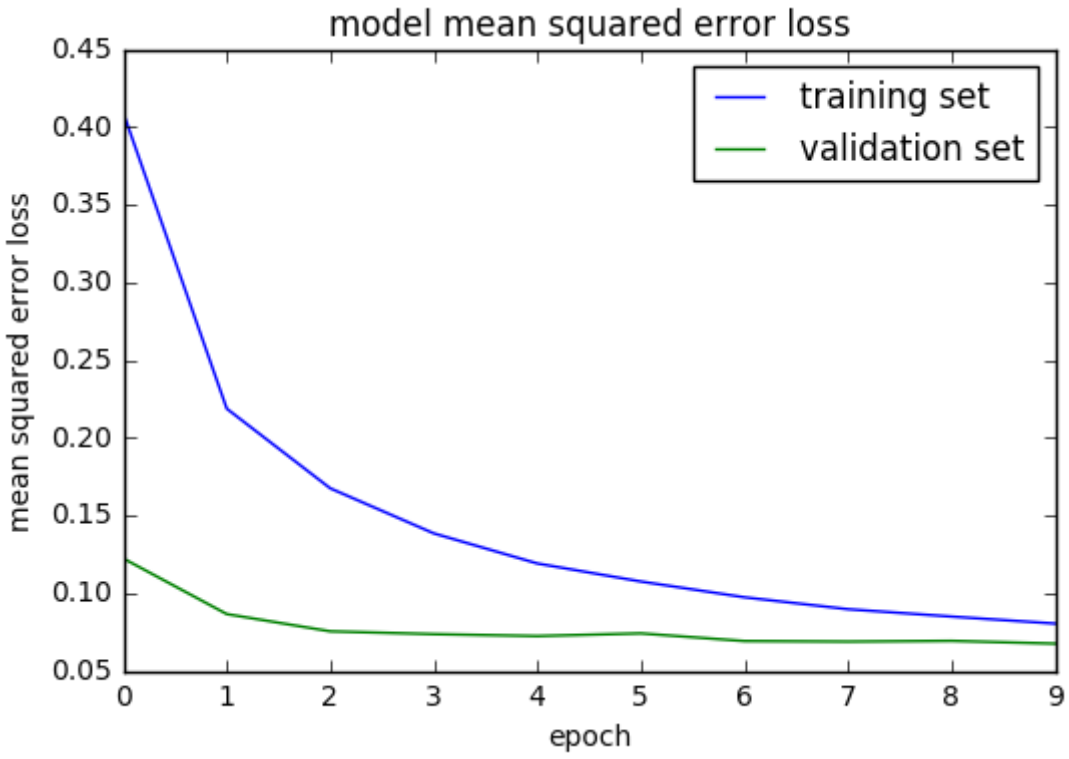


C) Recovery data by driving towards the center from edges Number of rows = 11889



I finally randomly shuffled the data set and put 10% 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 10 as evidenced by both training/validation set error platued as shown in the diagram



In [ ]: