Behavioral Cloning Project

The goals 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

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
- 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.

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 the 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 both 3*3 and 5*5 filter sizes, and depths between 12 and 60 (model.py lines 74-78). The model includes Relu layers to introduce nonlinearity (code line 74-78), and the data is normalized in the model using a Keras lambda layer (code line 70-72).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 79). The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 15-16). 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 86)

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. Moreover, I used multiple cameras i.e. from center, left and right to capture images and took them as training data. For the left and right cameras, I also gave an offset of steering wheel angle to improve normalization.

Model Architecture and Training Documentation

1. Solution design documentation

The overall strategy for deriving a model architecture was to use a basic neural network model as a reference and then tried to modify its parameters so that the mean squared errors of training and validation sets were both small enough.

I took Nvidia architecture as a reference since it is recommended in this paper. My first model had a low mean squared error on the training set but a high mean squared error on the validation set. That implied my model suffers a little bit overfitting. To fix this problem, I used dropout and the mean squared error of validation set decreased. The final step was to run the simulation to see if the car can drive around track one. There were a few spots where the vehicle fell off the track, for example the sharp right/left turn and the bridge. To improve the autonomous driving behavior in these case, I collected more data, which steered the vehicle back to the middle of the road.

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

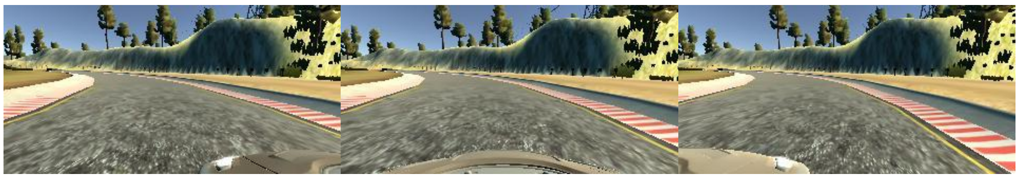
2. Model architecture documentation

The final model architecture consisted of a convolution neural network with the following layers and layer sizes:

Input: 160*320*3
Normalization:
Conv layer with Relu: 12 filters with 5*5 filter size
Conv layer with Relu: 24 filters with 5*5 filter size
Conv layer with Relu: 36 filters with 5*5 filter size
Conv layer with Relu: 48 filters with 3*3 filter size
Conv layer with Relu: 60 filters with 3*3 filter size
Dropout: 20%
Flatten: 540 nodes
Fully connected layer: 540 * 50
Fully connected layer: 50 * 25
Fully connected layer: 25 * 10
Fully connected layer: 10 * 1

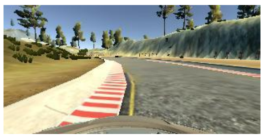
3. Training dataset and process documentation

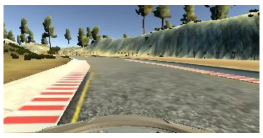
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example of center lane driving with multiple cameras:

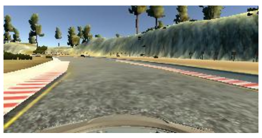


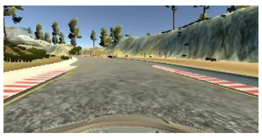
Left Center Right

I then recorded the vehicle recovering from sides of road back to center so that the vehicle could learn to steer back to road center. These images show a recovery from left side to road center.



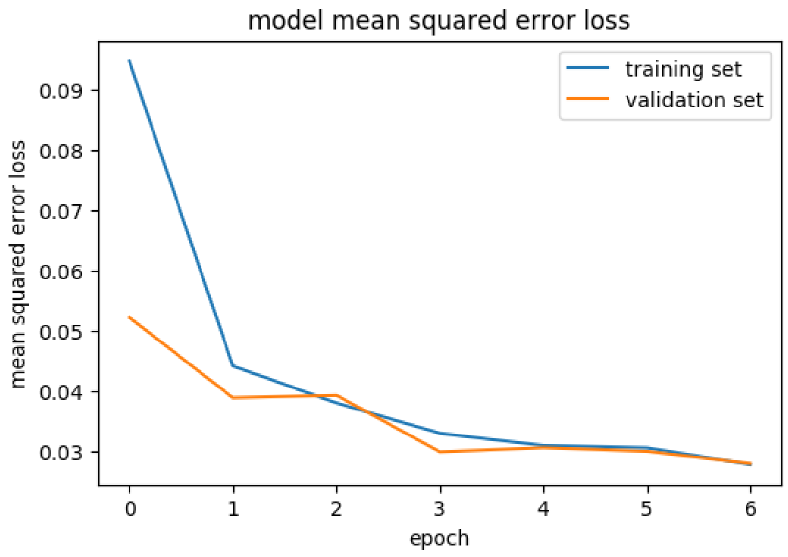






To augment the data set, I also flipped the image and angles (code line 48-53) and then preprocessed these data by choosing an interested area that excludes the sky and the hood of the vehicle (code line 73).

I finally randomly shuffled the data and put 20% of the data into a validation set. The validation set helped to determine if the model suffered over/under fitting problem. The mean squared error of training and validation is showed as:


model mean squared error loss
training set
validation set
mean squared error loss
epoch