Behavioral Cloning

Writeup

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior and some correction 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
* writeup.pdf summarizing the results
* video.mp4 file recording of the vehicle driving autonomously around the track.

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 was created with Keras's Sequential model(line 65~97). The structure based on the one from commaai

- A Lambda layer is used to normalize the images and a crop layer to ignore the useless infomation in images.
- Three convolution layers were implemented:
  a. 16 filters of kernel size 8x8, 4x4 stride, same padding b. 32 filters of kernel size 5x5, 2x2 stride, same padding c. 64 filters of kernel size 5x5, 2x2 stride, same padding
- Flatten layer
- Dropout layer
- Fully connected layer (512)
- Dropout layer
- Fully connected layer (128)
- Dropout layer
- Fully connected layer (1)

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (lines 80,86,92 in *model.py*).

The model was trained and validated on different data sets to ensure that the model was not overfitting. 20% of datasets were chosen as validation datasets.

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 over MSE loss. The model parameter was not tuned manually.

4. Appropriate training data

Training data was chosen:

1. Drive the vehicle on the center of the road (both direction).
2. Due to the test results showed the car was not able to recover in some cases, I drived the car in some

recovering cases and record the data.
3. Additional datasets were included to deal with the fail situations.

It is very difficult to do continous steering input on keyboard. The quality of trainning data I saved was not good which bring more difficluties into the model training.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to adpat the reference model, feed with suitable data, test and adapt model or training data according to test results.

My first step was to use a convolution neural network model similar to the one from commaai. I thought this model might be appropriate because its very good test results during vehicle test.

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 a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model to add one addtional dropout layer(line 92).

Then I test the model, the overfitting problem solved. But the performance is not good enough. So I change the activation to ReLu and introduce addtional fully connected layer. I am not able to explain the reasons of these modifications. They were came from several test tries.

I also tried to use valid padding in convolution layers to speed up training. But the results were not good so I did not use valid padding.

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 drive the car to in similar cases and generate data. The new data were used to train the model again.

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 65-97) consisted of a convolution neural network with the following layers and layer sizes:

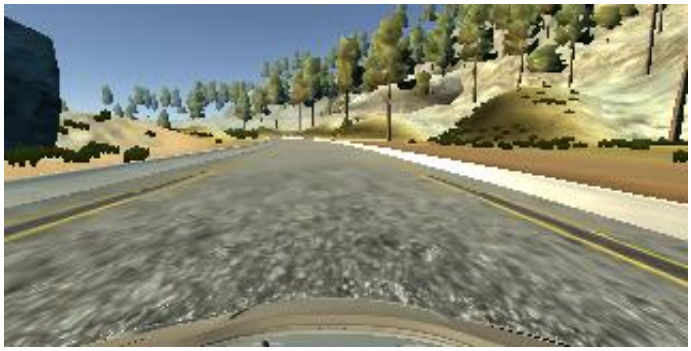
- A Lambda layer is used to normalize the images and a crop layer to ignore the useless infomation in images.
- Three convolution layers were implemented:
  a. 16 filters of kernel size 8x8, 4x4 stride, same padding b. 32 filters of kernel size 5x5, 2x2 stride,

same padding c. 64 filters of kernel size 5x5, 2x2 stride, same padding

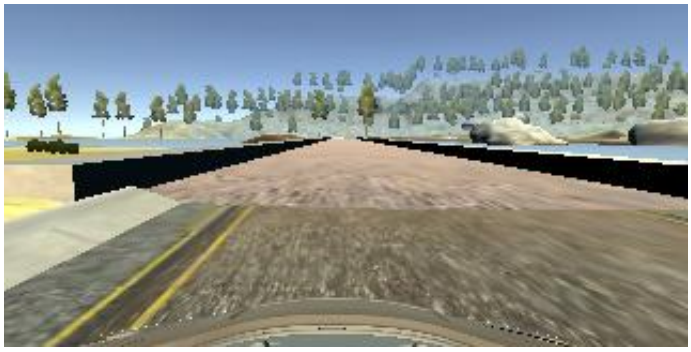
- Flatten layer
- Dropout layer
- Fully connected layer (512)
- Dropout layer
- Fully connected layer (128)
- Dropout layer
- Fully connected layer (1)

3. Creation of the Training Set & Training Process

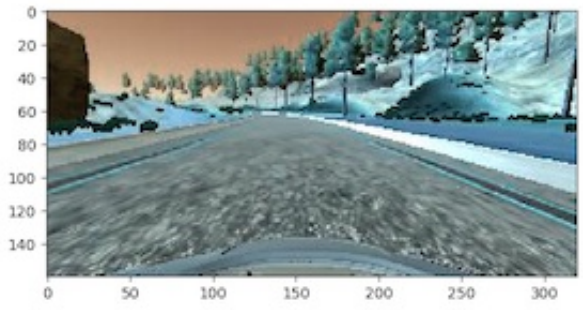
To capture good driving behavior, I first recorded two laps on track one using center lane driving on both directions. Here is an example image of center lane driving:

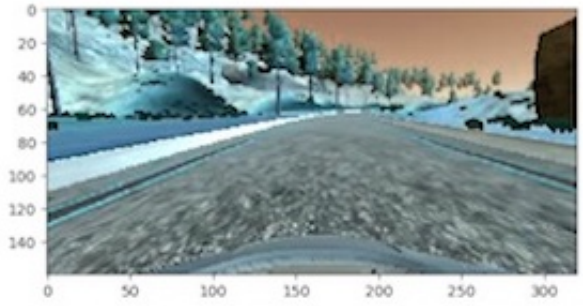


I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover from dangerous situations. These images show what a recovery looks like starting from almost crash to the bridge and were recovered mannually:



To augment the data sat, I also flipped center images and steering angles thinking that this would help to improve the model quality and prevent potential over fitting. For example, here is an image that has then been flipped:





After the collection process, I had 16820 number of data points. I then preprocessed this data by normalization and cropping.

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 number of epochs was 10 as evidenced by the acceptable loss value from training and validation.

- Epoch 10/10 13456/13456 [==============================] - 94s - loss: 0.0160 - val_loss: 0.0229

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