Behavioral Cloning

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

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 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. *An appropriate model architecture has been employed*

A final architecture is:

Layer	Model.py Code Line
Input 60x320x3	56
Convolution 12x5x5, valid, stride 1, RELU	57
Max Pooling 2x2, valid, stride 2	58
Convolution 24x5x5, valid, stride 1, RELU	59
Max Pooling 2x2, valid, stride 2	60
Convolution 48x5x5, valid, stride 1, RELU	61
Flatten	62
Fully Connected,120	63
Fully Connected,84	64
Fully Connected,10	65
Fully Connected,1	66

2. Attempts to reduce overfitting in the model

The model is trained on three cameras of three laps of entire track and flipped augmented data (code line 41-46), without the augment, the model is overfitting.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 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(0.001) was not tuned manually (model.py line 68).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of three camera images of three laps and augment of flipped images total of 20,556 samples.

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 use a base architecture of LeNet and two laps of driving on the track. Adjusting the architecture and training data according to the performance.

My first step was to use a convolution neural network model similar to the LeNet, I thought this model might be appropriate because it's a simple model can be trained fast and effectively pick up the main characters in an image as a classifier. I only tuned the last fully connected layer output as 1 to fit the steering angel output.

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 add the augment data as flipping the images and imply with negative measurement to double the training set.

Then I tested the result on the track, it still can't pass the turn after the bridge, instead it runs into the open area behind the track. So, I decided to change my model to a little more complex by tuning to deeper convolutional layers and add one more convolutional layer.

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

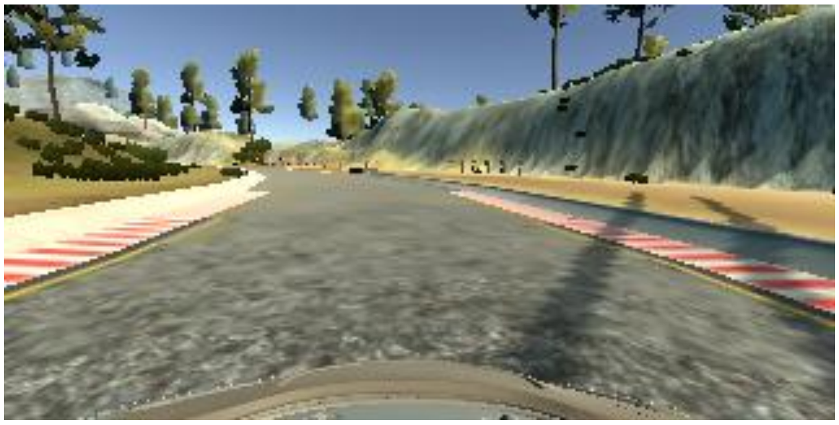
2. Final Model Architecture

The final model architecture (model.py lines 18-24) consisted of a convolution neural network with the following layers and layer sizes:

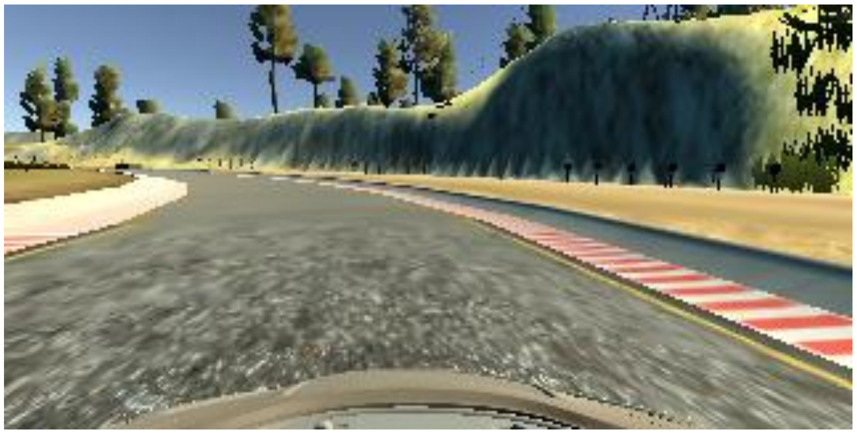
Layer	Model.py Code Line
Input 60x320x3	56
Convolution 12x5x5, valid, stride 1, RELU	57
Max Pooling 2x2, valid, stride 2	58
Convolution 24x5x5, valid, stride 1, RELU	59
Max Pooling 2x2, valid, stride 2	60
Convolution 48x5x5, valid, stride 1, RELU	61
Flatten	62
Fully Connected,120	63
Fully Connected,84	64
Fully Connected,10	65
Fully Connected,1	66

3. Creation of the Training Set & Training Process

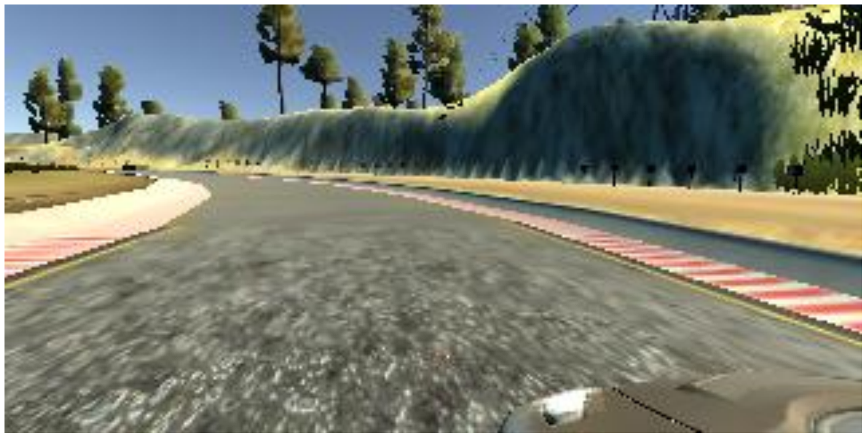
To capture good driving behavior, I first recorded three laps on track one using center lane driving. Here is an example image of center lane driving:



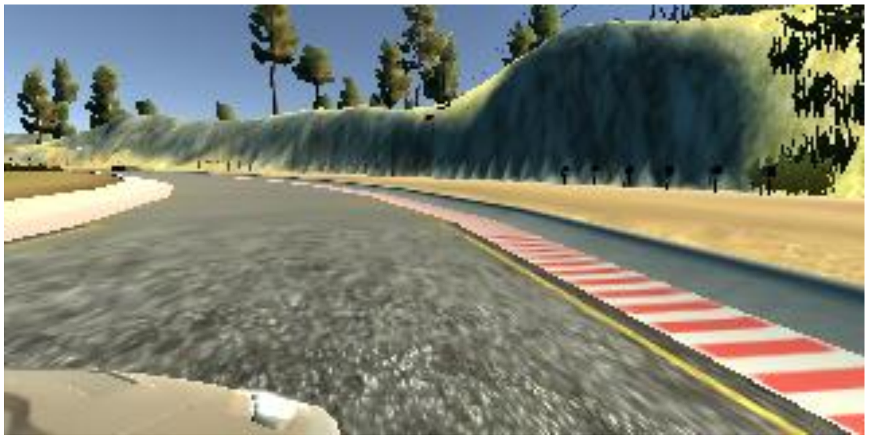
I then append the three images from center, left and right cameras to the data set(code line 22-36) with a correction(0.2) so that the vehicle would learn to recover when it is close to the side of the track. The left and right camera images are trained as center camera image. These images show three camera images and measurements at the same location:



Center, steering angel: -0.132

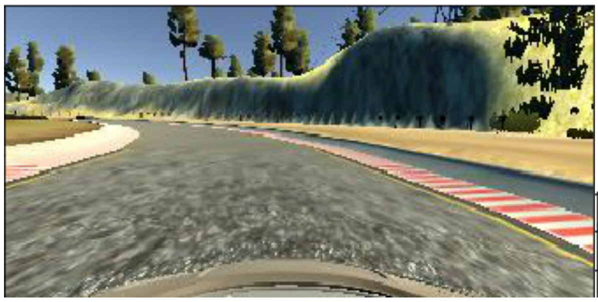
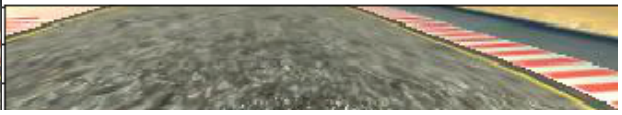


Left, steering angel: -0.068

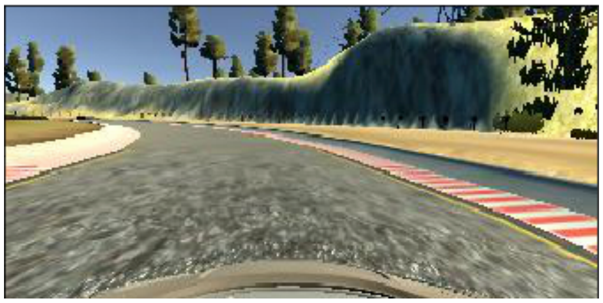
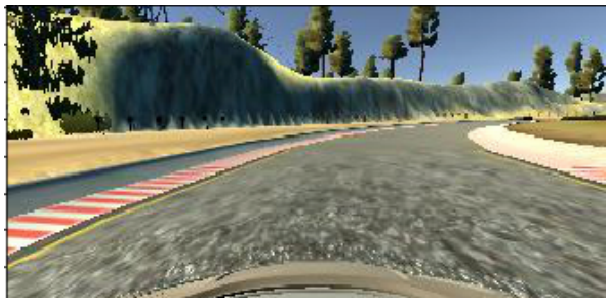


Right, steering angel: -0.332

In addition, because the convolution neural network is flexible with size of the input images, so I crop the image as 60x320x3 during preprocessing, so the deep learning is more focused on front road not the sky or trees. For example:

Lastly, to augment the data sat, I also flipped images and angles thinking that this would avoid the bias of driving in one direction and also generate moredata. For example, here is an image that has then been flipped:

After the collection process, I had 20,556 of data points. I then preprocessed this data by normalizing(x:x/255.0-0.5).

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 ideal number of epochs was 2 as evidenced by run1.mp4 and the loses are around 0.01in both training set and validation set. I used an “adam” optimizer so that manually training the learning rate wasn't necessary.