Behavioral Cloning

Project Report

Submitted files

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.pdf summarizing the results

Running the model

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
```

Model code

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. Model architecture

My model consists of a 5 convolution layers and 4 fully connected layers. There are 3 convolutional layers with 5x5 filter sizes and depths 24, 36 and 48 respectively and 2 convolutional layers with 3x3 filter sizes and depth 64) (model.py lines 87, 89, 91, 93 and 95).

The model includes RELU layers (activation = ‘relu’) and the data is normalized in the model using a Keras lambda layer (code line 78).

4 fully connected layers consists of 100, 50, 10 hidden units and 1 output unit.

2. Attempts to reduce overfitting in the model

After each convolutional or hidden fully connected layer the model contains dropout layers in order to reduce overfitting.

Different drop out rate were tested: 0.2, 0.5 and 0.8 and with the drop out rate of 0.2 the best results were achieved.

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.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 108).

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 and data augmentation.

For details about how I created the training data, see the next section.

5. Solution Design Approach

The overall strategy for deriving a model architecture was to start with simple model and small amount on the data and stepwise increase the data size and the model complexity.

My first step was to use a convolution neural network model similar to the LeNet. I thought this model might be appropriate because LeNet proved to be a good model for image recognition.

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 include dropout. I tried several values for dropout rate: 0.2, 0.5 and 0.8. It turned that the dropout rate of 0.2 was the optimal one.

Then I also increased the number of data samples by driving 1 lap. I also increased the complexity of the model by using NVIDIA like model.

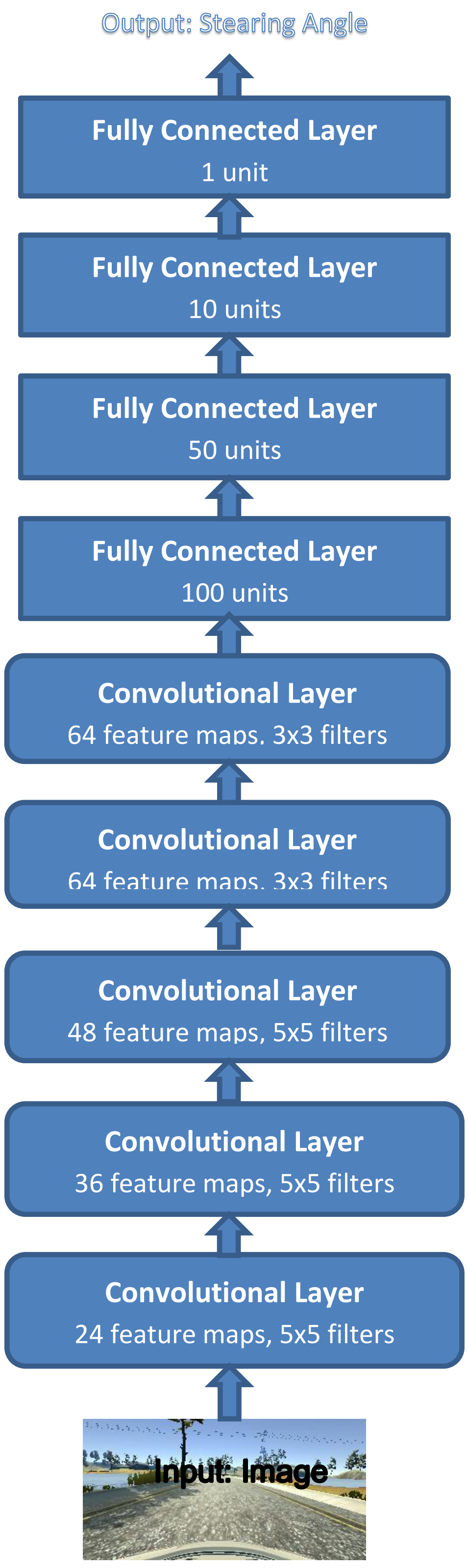
The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track like sharp curves. To improve the driving behavior in these cases, I collected another lap of the training data. Since it was still not enough I augmented the data size by flipping the images and also using left and right cameras. The car could now drive autonomously longer but still not for the whole lap.

Finally, after increasing training size for another lap (in total 3 laps + already provided data) the vehicle was able to drive autonomously around the track (for a long time) without leaving the road.

6. Final Model Architecture

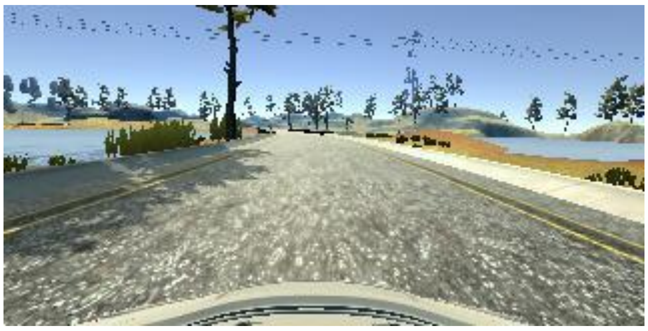
The final model architecture (model.py lines 87-105) consisted of 5 convolution layers and 4 fully connected layers. There are 3 convolutional layers with 5x5 filter sizes and depths 24, 36 and 48 respectively and 2 convolutional layers with 3x3 filter sizes and depth 64) (model.py lines 87, 89, 91, 93 and 95).

Here is a visualization of the architecture:

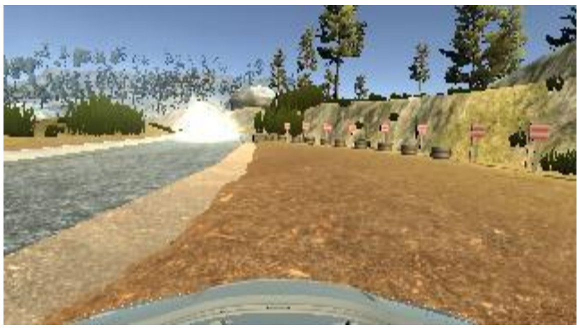


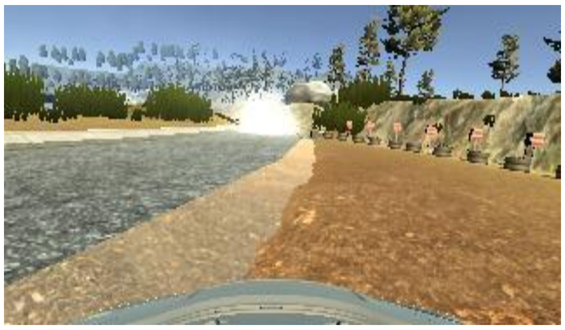
7. Creation of the Training Set & Training Process

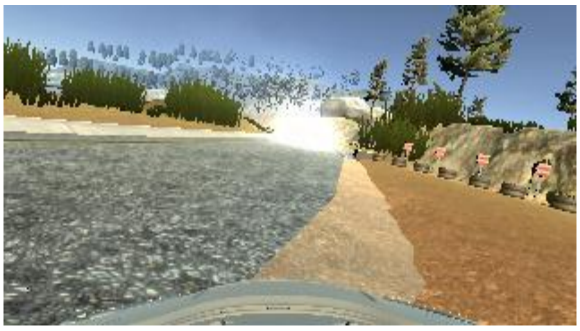
To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

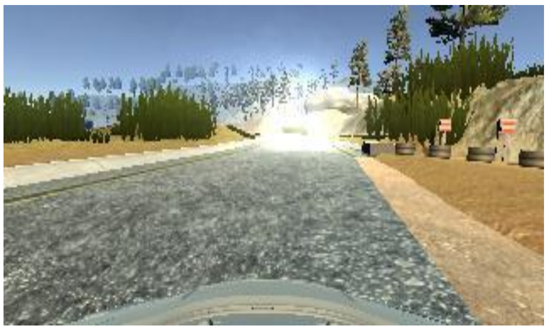


I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to return to the center. These images show what a recovery looks like



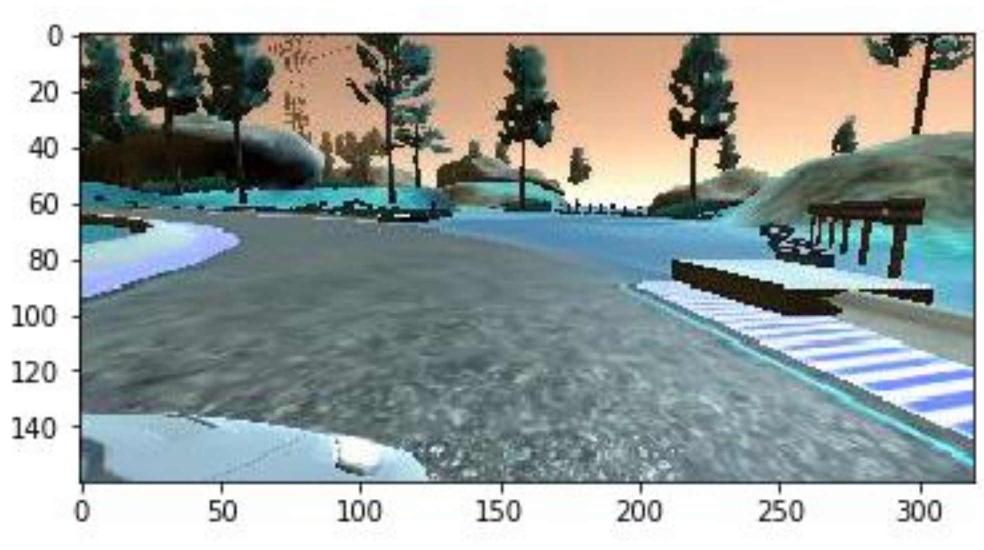




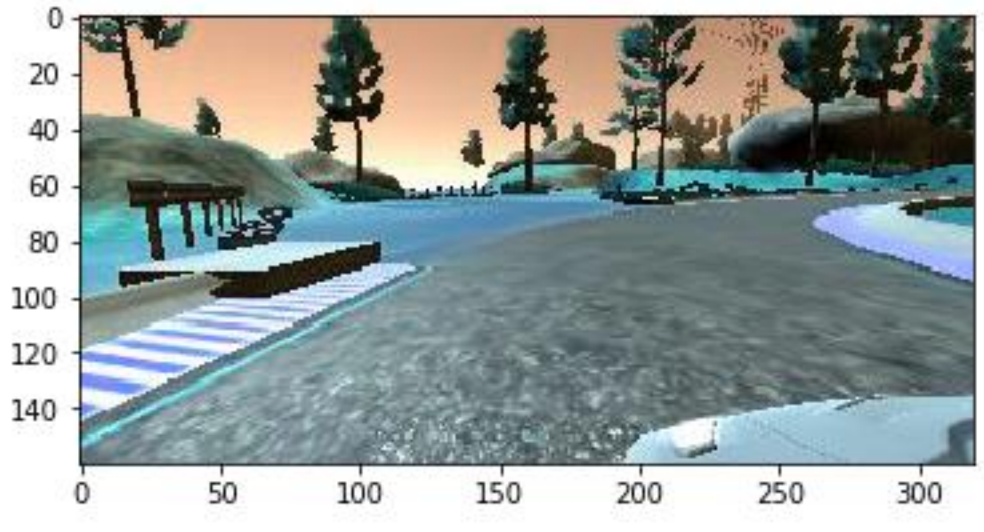


To augment the data sat, I flipped images and angles as well as used not only the central camera but also the left and the right camera (with some optimization of the corrections angle). This would increase the number of samples and improve the network performance.

For example, here is an original image:



And here the flipped image:



Since after 2 laps and data augmentation it was still not possible to drive autonomously the whole lap I collected another one lap of the date.

Finally it was possible to drive autonomously not only a single lap but the vehicle was able to drive autonomously several hours on the first track.

After the collection process, I had total of 125826 data points. I then preprocessed this data by the 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 ideal number of epochs was 5 as evidenced by decreasing the validation error. I used an adam optimizer so that manually training the learning rate wasn't necessary.