Sebastian Nietardt

Udacity Self-Driving Car Nanodegree

Project 3

March 12, 2017

Behavioral Cloning

Files Submitted & Code Quality

Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- model.py containing the script to create and train the model
- run1.mp4 containing a video of the car, autonomously driving around track 1
- writeup_report.pdf summarizing the results

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
```

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

An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes with a stride of 2x2 and 1x1 respectively. The depths are between 24 and 64. It includes RELU layers to introduce nonlinearity. The source code can be found in the file model.py lines 71 - 75.

In addition, the data is normalized in the model using a Keras lambda layer (code line 67).

Attempts to reduce overfitting in the model

The model contains two dropout layers in order to reduce overfitting (model.py lines 81 & 85). The inputs in each dropout layer were kept with a probability of 70%. The model was validated with validation data and tested by running it through the simulator and ensuring that the vehicle could stay on the track.

Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 90).

Appropriate training data

The data to train the model needed to be generated by driving the vehicle in the simulator myself. The training data is supported to contain all the important information to keep the vehicle driving on the road. That means the data must show how to drive the car in the middle of the road as well as how to recover from the side of the road. More details about the training data will be described in the next section.

Model Architecture and Training Strategy

Solution Design Approach

My first step was to use a convolution neural network model similar to the NVIDIA autonomous car architecture. I thought this model might be appropriate because they use it for real autonomous driving. The data was split into a training and validation set to measure the loss during training and check for overfitting. In order to prevent overfitting, I also added dropout layers to the model.

I played around with the layers and added the activation RELU to the fully connected layers but it increased the loss. It looks like the layers at the end of the model needs the inputs without an activation function to predict the angle as good as possible. That is why I decided not to use the RELU activation function in the fully connected layers. Instead, I added two additional fully connected layers to the model and increased the size of each fully connected layer. It helped to decrease the validation loss.

After finishing the model and running the simulator, the vehicle is able to drive autonomously around the track without leaving the road.

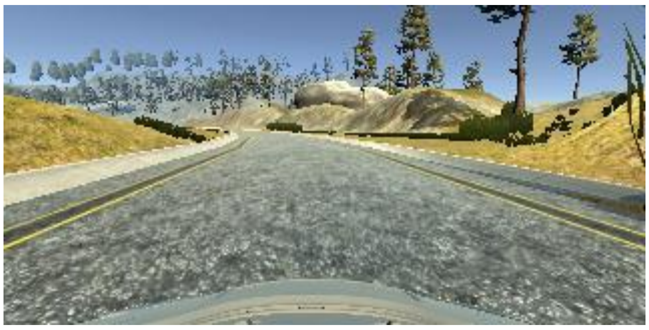
Final Model Architecture

My final model consists of several layers which are described in the following table. The source code to the model can be found in the file model.py lines 65-87.

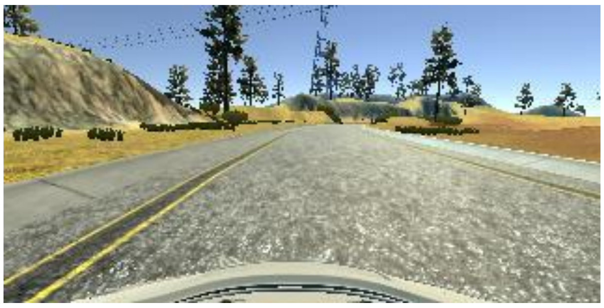
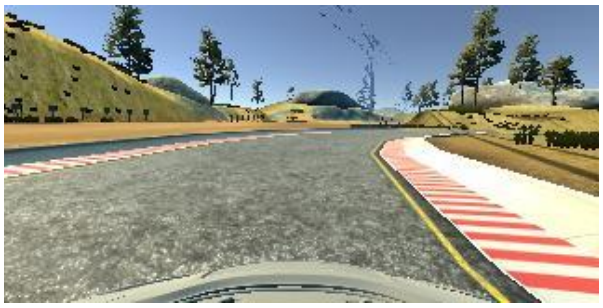
Layer	Description
Input	160x320x3 color images
Lambda	Data normalization to interval [-1:1]
Cropping	160x320x3 → 65x320x3
Convolution 5x5	2x2 stride, valid padding, outputs 31x158x24
RELU	
Convolution 5x5	2x2 stride, valid padding, outputs 14x77x36
RELU	
Convolution 5x5	2x2 stride, valid padding, outputs 5x37x48
RELU	
Convolution 3x3	1x1 stride, valid padding, outputs 3x35x64
RELU	
Convolution 3x3	1x1 stride, valid padding, outputs 1x33x64
RELU	
Flatten	Output 2112
Fully connected	Output 160
Fully connected	Output 160
Dropout	70% keep probability on training
Fully connected	Output 80
Fully connected	Output 80
Dropout	70% keep probability on training
Fully connected	Output 40
Fully connected	Output 1

Creation of the Training Set & Training Process

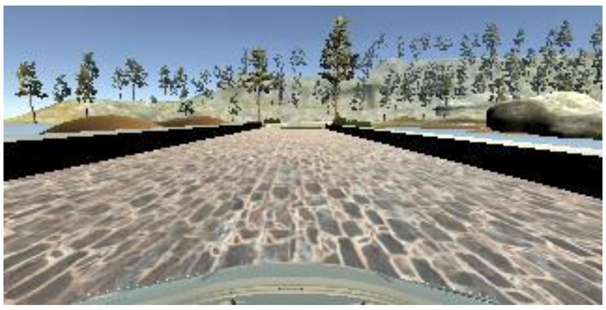
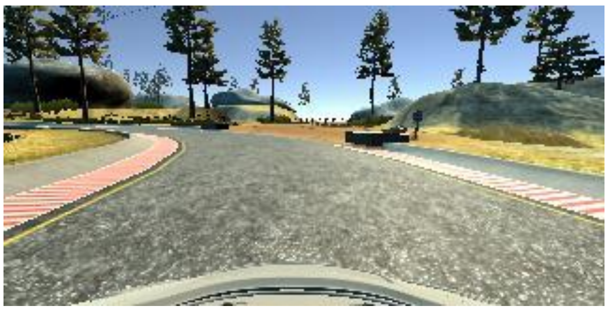
The data to train the model needed to be generated by driving the vehicle in the simulator myself. I drove two laps in each direction trying to drive in the middle of the road. This is an image of center lane driving:



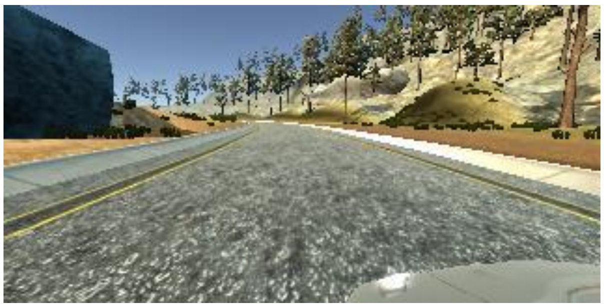
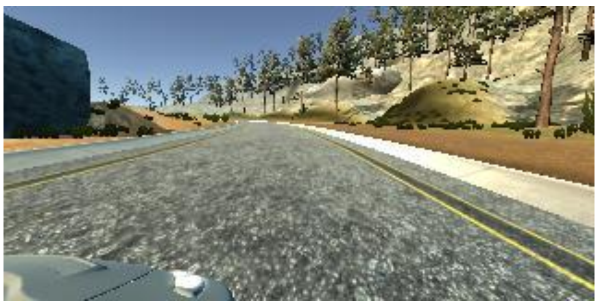
Afterwards I generated some data of recovering situations. In order to generate that data, I drove one lap in each direction, taking videos driving from the sides of the road to the middle of the road. The following two images are sample images. The left image has been taken recovering from the left side of the straight road and the right image has been taken recovering from the right side in a curve. Since recovering should be a smooth process, the angle in these images is not that different from other pictures. One can hardly see that the car is driving towards the center of the road.

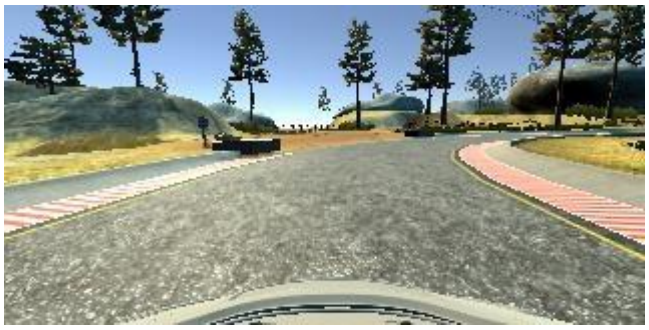
After that I took some additional videos of driving over the bridge and around curves in each direction.

The left and right camera images have also been used to create more recovery data. The steering angle for these images needed adjustment to steer the car back to the center of the road. For the left camera image, the value 0.2 was added to the angle. For the right camera image, the value 0.2 was subtracted from the angle. These are example images from the left and the right camera. Both images were taken at the same time and show the road from both camera perspectives.

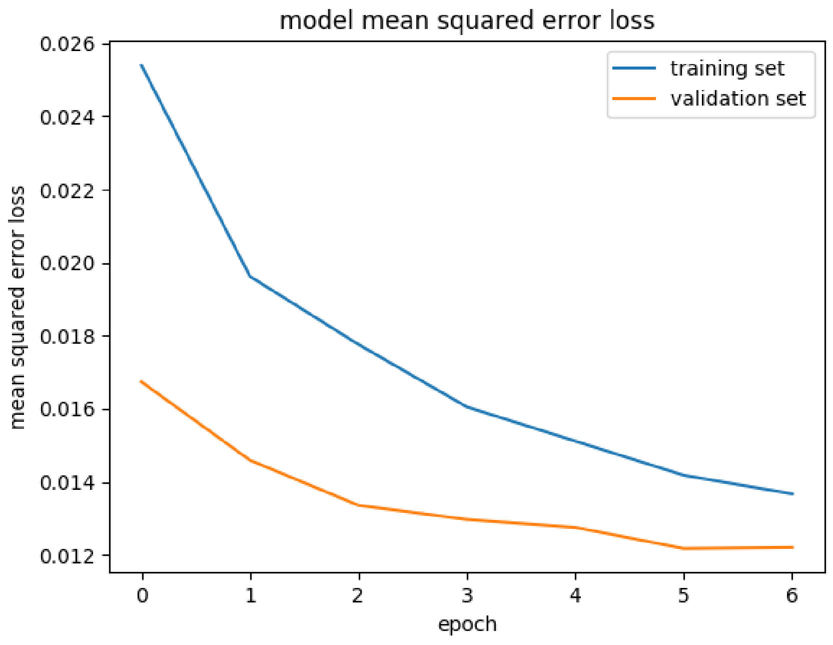



To augment the data set, I also flipped the images thinking it would generalize the data. Here is the flipped image of the curve from above:



After the collection process, I had 75414 number of data points. Then I preprocessed this data by dividing its values by 127.5 and subtracting 1. This brought the values of the images into an interval between -1 and 1.

The model used an adam optimizer so that manually training the learning rate was not necessary. The data was randomly shuffled and 20% of the data were put into a validation set. I used this training data for training the model. The validation set helped to determine if the model was overfitting or underfitting. The ideal number of epochs was seven. After seven epochs, the validation loss did not improve anymore or even increased. The following plot shows the loss after each epoch:


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