Term 1-P3: Behavioral Cloning
Sebastian Lena

Overview

Project #3 involves behavioral cloning, using a deep neural network to teach a car to drive in a simulator provided by Udacity. The simulator can be found in:

https://github.com/udacity/self-driving-car-sim

This simulator was developed in unity3d, a game engine.
The simulator has cameras that capture the images from center, right and left side of the virtual car.

My code for this project is publicity available and can be found here:

https://github.com/B0WS3R/CarND-Term1-P3-Behavioral-Cloning

Introduction

The idea for this project is to make a car drive around a track by building a neural network architecture (thanks to Nvidia!) and then training it using all the data obtained by a human driving around it.
We'll use the simulator provided by Udacity to get the driving data, then we'll use the simulator to test our trained model.

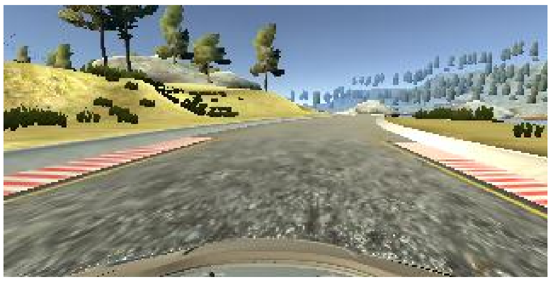
Obtaining training data

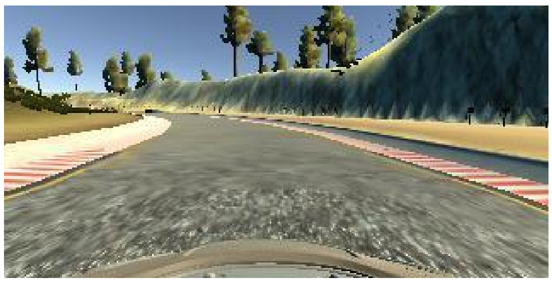
Each image is captured with the accompanying steering angle. Later the images are be fed into the network, and the network's job is to match the appropriate steering angle.
When the images are loaded each image gets a copy that has been flipped horizontally with steering angle negated. (this because the test track has a turn bias to the left side)

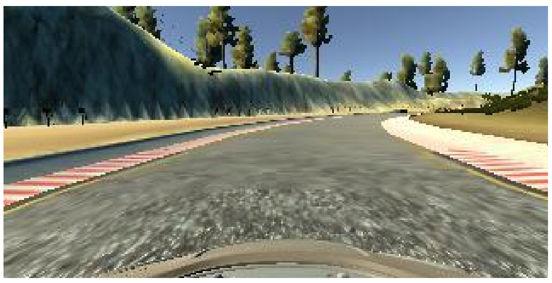
I evaluated different scenarios and collecting the data from all of them:

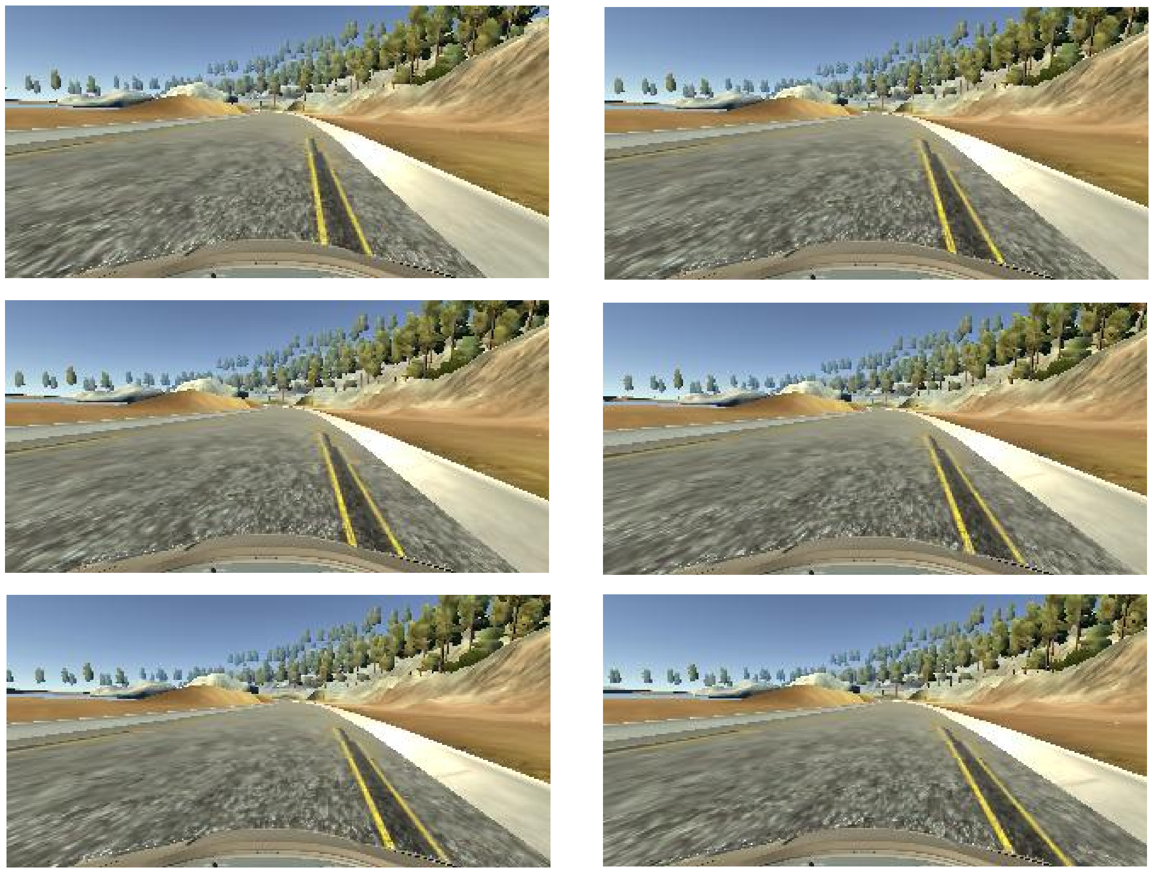
```
scenarios = [
                'tl-normal',
                'tl-backward',
                'tl-right',
                'tl-left',
                'tl-corrective',
        ]
```

I drove the track #1 first in normal direction (forward), then in backward direction. This represents the most important data for the neural network.
Also I collected a corrective data driving from the edge of the road (in both sides, left and right), from the edges to the center of road (here I tried to teach the car how re-enter to the center of the road), and data from the cornes (bridge) and curves.
It's important don't record when the car goes outside the road because in this case the car will clone this behavior (I learned this by trial and error).
The goal of all this was to help the model with the concept of "staying on the road."


<front-center>


<center to left>


<center to right (flipped)>


<corrective sequence, re-enter to center of road>

I only used de images from the center of the car.
I don't collect data from second track.

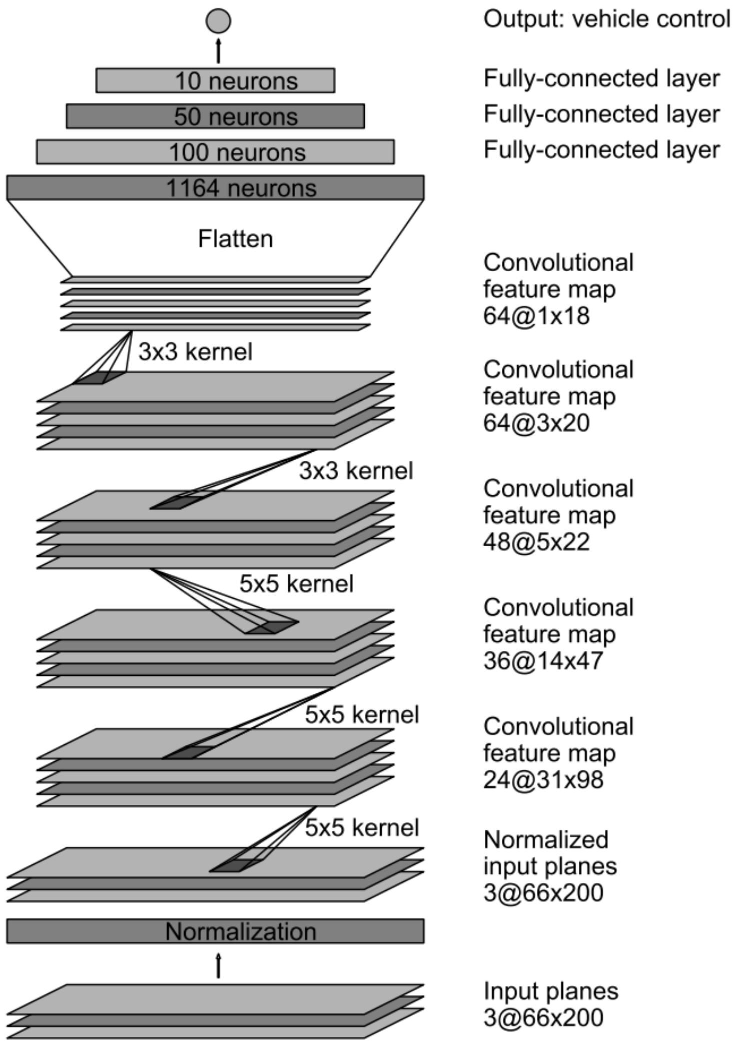
Preprocessing data

As part ot the training pipeline, the images were cropped to remove the top and bottoms part and normalized.

Model architecture

The chosen model architecture was NVIDIA's network for End-to-End Learning for Self Driving Cars, here is the paper:

https://arxiv.org/pdf/1604.07316v1.pdf



The model look like:

Input: pass in the 320x160 RGB images
Cropping: remove the top 60 pixels (sky) and bottom 25 pixels (car hood). This information isn't useful for determining steering angles
Normalization: pixel values [0, 255] are scaled down to [-0.5, 0.5].
Convolution: 24 features, 5x5 filters, stride of 2

Convolution: 36 features, 5x5 filters, stride of 2
Convolution: 48 features, 5x5 filters, stride of 2
Convolution: 64 features, 3x3 filters, stride of 1
Convolution: 64 features, 3x3 filters, stride of 1
Flatten: flatten the space down to one dimension for the fully-connected layers.
Fully-Connected: 1164 values from linear combinations of previous values
Fully-Connected: 100 values from linear combinations of previous values
Fully-Connected: 50 values from linear combinations of previous values
Fully-Connected: 10 values from linear combinations of previous values
Output: 1 steering angle from linear combinations of previous values

Each convolutional layer and fully-connected layer has **RELU** activation, also a **dropout layer** with 25% drop-rate to help prevent overfitting.

Training the model

I used **Keras** to build and train the model because with keras is quick and simple.
The Adam optimizer was used with a **learning rate = 0.001**, and the **MSE** (mean-squared-error was minimized).
The data was shuffled and the model was trained in randomly batches of **64** images and **7** epochs.
For validation, **20%** of data was reserved.
In total the model was trained using around **~ 50K** images (48981).

Results and conclusions

After the training, the model was capable of doing several laps to circuit 1 and never left the track.
The car made a minimals corrections on the straight-aways and had a couple handling near of edges.
For the second track was unable to drive correctly, obviously because the training data was generated in the track 1. I believe this model could handle driving on the second track if I were to provide it with the correctly training-data (from the second track)
I found this project easy to implement in keras following the nvidia paper, but very difficult to get the correct training data on the simulator and get the correct behavioral driving.