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

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_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.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network with 5x5 and 3x3 filter sizes and depths between 24 and 64 (model.py lines 40-52)

The model includes RELU layers to introduce nonlinearity (code line 43-47), and the data is normalized in the model using a Keras lambda layer (code line 41). The model was cropped to remove the hood of the car and top of the image where there is no road using cropping2D (code line 42).

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting. Images were flipped to reduce bias towards turning a particular direction because the track I trained on mostly turns one direction. 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 54).

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 roads and used the mouse so that I did not have jerky driving behavior.

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 transfer learning for a convolutional neural network that is suited to the application of self driving cars. I used

the NVIDIA end to end learning model because it performs the same task as this simulator but in the real world.

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 while the model was performing very well after 3 epochs the validation loss was increasing and starting to overfit so I lowered the amount to 3 epochs.

The final step was to run the simulator to see how well the car was driving around track one. The car went around the track without running into anything! I attribute this to selecting a well suited model and doing as much data augmenting and preprocessing as I could think of.

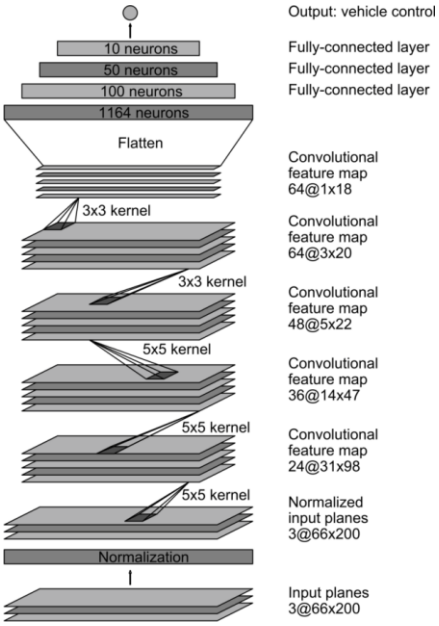
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 40-52) consisted of a convolution neural network with the following layers and layer sizes:

- Normalization lamda layer
- Cropping2d layer
- Convolution2D 24,5,5
- Convolution2D 36,5,5
- Convolution2D 48,5,5
- Convolution2D 64,3,3
- Convolution2D 64,3,3
- Flatten
- Fully connected dense 100
- Fully connected dense 50
- Fully connected dense 10
- Fully connected dense 1

Here is a visualization of the architecture

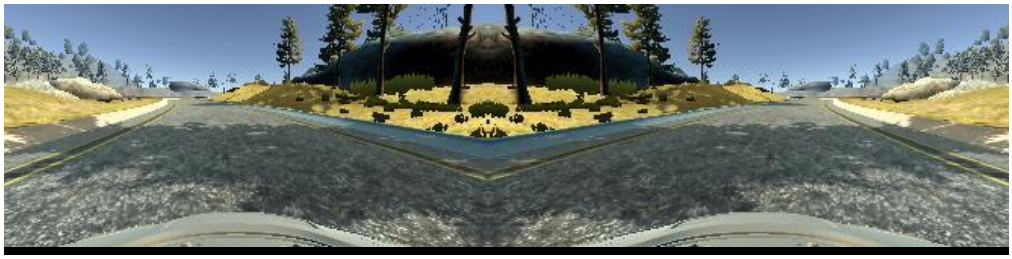


3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded four laps on track one using center lane driving going at full speed while steering with the mouse. I traveled 2 laps in each direction. 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 go back to the middle of the road if it was drifting.

To augment the data set, I also flipped images and angles thinking that this would ... For example, here is an image that has then been flipped:



After the collection process I preprocessed this data by including the left and right camera images. I put a correction factor of 0.2 on each camera’s steering angle. This tripled my amount of data and allowed the car to learn how to do slight turns towards the middle from the edges of the road.

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 3 as evidenced by the validation loss increasing at 4 epochs. I used an adam optimizer so that manually training the learning rate wasn't necessary.