Behavioral Cloning
Report by Cristian Cucchiella

Objective

The focus of this project is to built a convolutional neural network with Keras and tensorflow with the goal to create a model that can predict the correct steering angle of a car given a single input image. The idea is to use an end-to-end pure deep learning approach in order to let the model learn the relevant features of the road while driving and recording the steering angle. A regression model can then be used to predict the steering angle feeding an image from a front camera.

The project consisted in the following steps:
- Use of a simulator to collect data (composed of images and steering angle) 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 extracted from the collected data
- Test that the model successfully drives around the tracks without leaving the road

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

The 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
- **track1.mp4** a video recording of the car driving on the first track in autonomous mode
- **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 used for training and validating the model. Moreover it contains comments explaining the code.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

I used the nVidia Autonomous Car Group model, and the car drove the complete first track after just three training epochs.

2. Attempts to reduce overfitting in the model

Instead of applying regularisation techniques like Dropout or Max pooling, I preferred to keep the training epochs low: three epochs only. In addition, I split data into training (80%) and validation (20%).
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 146).

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I collected the training data driving on track 1 in the center of the lane and in both direction. The simulator provides three different images: center, left and right cameras. Each image was used to train the model.
For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

I used the nVidia Autonomous Car Group model.The only modification was to add a new layer at the end in order to have a single output, as required.
The data has been augmented by adding the same image flipped with a negative angle (lines 85). In addition to that, the left and right camera images where introduced with a correction factor on the angle to help the car go back to the lane (lines 50). The car continues to experience issues with the "dashed" line. I needed more data, but it was a good step.

2. Final Model Architecture

A model summary is as follows:

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 160, 320, 3)	0	lambda_input_2[0][0]
cropping2d_1 (Cropping2D)	(None, 90, 320, 3)	0	lambda_1[0][0]
convolution2d_1 (Convolution2D)	(None, 43, 158, 24)	1824	cropping2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 20, 77, 36)	21636	convolution2d_1[0][0]
convolution2d_3 (Convolution2D)	(None, 8, 37, 48)	43248	convolution2d_2[0][0]
convolution2d_4 (Convolution2D)	(None, 6, 35, 64)	27712	convolution2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 4, 33, 64)	36928	convolution2d_4[0][0]

flatten_1 (Flatten)	(None, 8448)	0	convolution2d_5[0][0]
dense_1 (Dense)	(None, 100)	844900	flatten_1[0][0]
dense_2 (Dense)	(None, 50)	5050	dense_1[0][0]
dense_3 (Dense)	(None, 10)	510	dense_2[0][0]
dense_4 (Dense)	(None, 1)	11	dense_3[0][0]

Total params: 981,819
Trainable params: 981,819
Non-trainable params: 0

3. Creation of the Training Set & Training Process

To have more data, on track 1, I collected data driving both forward and backward.
All these data was used for training the model with three epochs. The data was shuffled randomly.

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:

Resources

http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf
https://devblogs.nvidia.com/deep-learning-self-driving-cars/