Teaching a simulator how to drive

Andreas Rubin-Schwarz

***Abstract*—One center piece in the quest for autonomous cars is called behavioral cloning. By turning human driving in machine readable information it is possible to train a computer to learn to steer a car in the correct direction. While a lot of different approaches have been undertaken to solve this task, a very promosing avenue that has been developed in the last decade is deep learning. Deep neural networks seem to have great capabilities to act on image data. In this project a convolutional neural network (CNN) is created that is capable of driving a vehicle in a computer simulation without human intervention. The final model scores a Mean Squared Error of .0176.**

***Keywords*—autonomous driving, behavioral cloning, convolutional neural network, cars, steering angle prediction**

I. Project Overview

The following project leverages image and steering data collected from a car simulator to create a convolutional neural network that learns to predict the correct steering angle and ultimately is able to drive itself. Fig. I shows a sample image of the dataset. Code and weights can be found in following files:

1) **model.py**: script to create and train the model
2) **drive.py**: for driving the car in autonomous mode
3) **model.h5**: contains the trained convolution neural network
4) **evaluate.py**: contains the evaluation code

Using a Udacity provided simulator and the drive.py file, the car can be driven autonomously around the track by executing following command:

```
sh python drive.py model.h5
```

FIG. I. Sample Image

A. Model Architecture and Training Strategy

The final model draws inspiration from the NVIDIA paper End-to-End Deep Learning for Self-Driving Cars [1]. It contains four convolutional layers with a filter size of 5x5 and 3x3 as well as increasing depths from 24 to 64. Afterwards the information is funneled through four dense layers with decreasing size. While the original architecture works with fewer and smaller dense layers, the additional layers seem to have a smoothing effect on the predicted values. After implementing the additional depth of the network the car steers less 'choppy'. A cropping layer removes some of the unnecessary information by reducing the image size to 90x320. Figure III shows the image after cropping. Afterwards normalization is applied to help the model converge quicker. An overview of the architecture can be seen in Figure II and Table I.

TABLE I. Model Summary

Layer	Output Shape	Parameters
Cropping	(90, 320, 3)	-
Normalization	(90, 320, 3)	-
Conv2d	(43, 158, 24)	1,824
Conv2d	(20, 77, 36)	21,636
Conv2d	(8, 37, 48)	43,248
Conv2d	(6, 35, 64)	27,712
Conv2d	(4, 33, 64)	36,928
Flatten	(8448)	1824
Dense	(256)	2,162,944
Dense	(128)	32,896
Dense	(64)	8,256
Dense	(10)	650
Dense	(1)	11

Cropping and normalization is done on the fly as part of the model graph. ReLu activation is used in all layers for quicker learning.

B. Attempts to reduce overfitting in the model

To reduce overfitting the model contains a dropout layer after each dense layer with a dropout probability of 40 % and a dropout layer after each convolutional layer with a lower dropout probability pf 10 %. During training, the model was evaluated through a validation set containing 20 % of all collected data. The learning curve can be seen in Figure IV. During training, the model was trained and validated on different data sets to ensure that the model was not overfitting. After training, the model was used to steer the simulator on the training track by itself with a speed setting of 18 mph. To further validate the result a testing set was created containing an entire round of the first racing track.

C. Model parameter tuning

The final model uses adam optimization with a learning rate of 0.001 and implements early stopping to determine the number of epochs. A batch size of 128 is being used as well as two dropout schemes for convolutional and dense layers to help prevent overfitting.

FIG. II. ARCHITECTURE OF CONVOLUTIONAL NEURAL NETWORK.

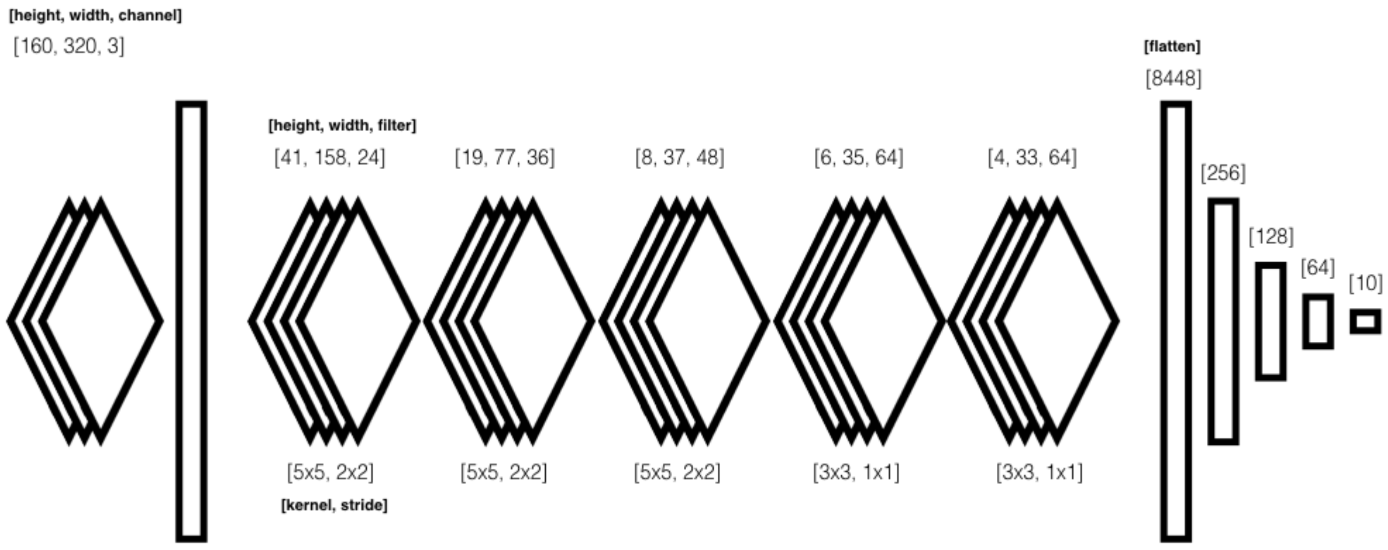


FIG. III. CROPPED IMAGE

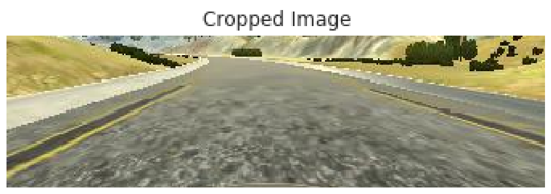
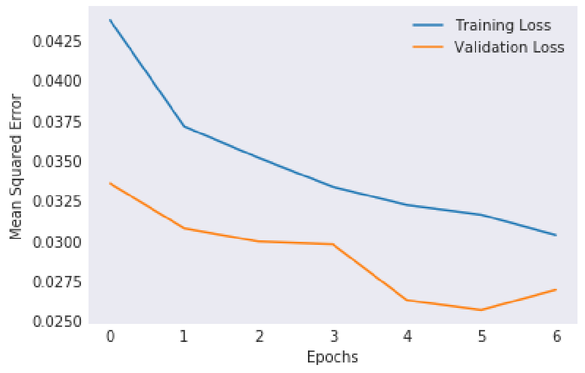


FIG. IV. LEARNING CURVE



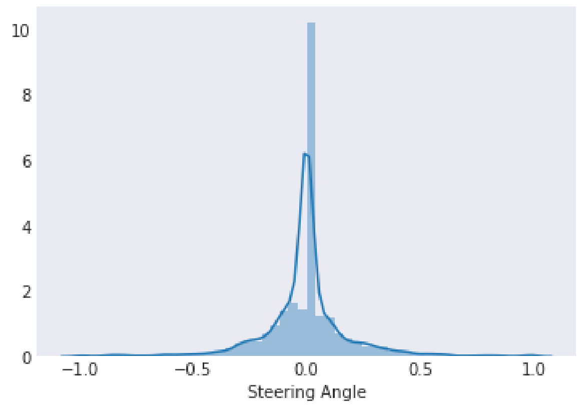
D. *Appropriate training data*

A sufficient amount of training data was chosen to keep the vehicle driving on the road. A combination of center lane driving, recovering from the left and right sides of the road as well as different camera angles have been used. For details about how the training data, see the next section.

II. CREATION OF THE TRAINING SET & TRAINING PROCESS

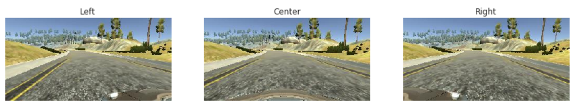
The creation of the training set was done in an iterative way. Before recording, a set of rounds were driven to get a feeling for the track and the controls. 'Normal' or good driving behavior was first recorded using mouse movements as well as keyboard movements. After the first set of recordings an initial network structure was implemented to get a baseline result for how good the imagery works with a simple network structure. While recording, it became obvious that most images showed uncritical situations where the car was going straight forward. (See Figure V).

FIG. V. DISTRIBUTION-PLOT OF STEERING ANGLES IN OBSERVATIONS.



The first track also seems to introduce a one-sided bias since most of the curves are leaning towards one direction. To mitigate those effects, following steps have been taken:

FIG. VI. DIFFERENT CAMERA PERSPECTIVES OF SIMULATOR.



1) Record driving behavior in curves rather than on straight patches of the track.
2) Capture critical moments such as leaving the road and recovering from both sides in an attempt to have the model learn more difficult situations.
3) Adding reverse laps to the data set
4) Leveraging all camera angles with a steering correction value of .2 for measurements[1]

To help the model generalize better, it was also trained on data gathered from the second racing track. After the collection and pre-processing steps the data set contained 83,352 observations. To preserve a certain randomness, the data was shuffled and split into training as well as validation sets. To determine the final amount of epochs, early stopping was leveraged. Due to using adam optimization manual training of the learning rate wasn't necessary.

III. EVALUATION AND OUTLOOK

To evaluate the final model two measures were used. First a test set containing a new lap was recorded. This test set contains 15,117 images. The evaluation code can be found in $evaluate.py$. The final model scored a Mean Squared Error of 0.0176. To further validate the model, the simulator was connected and steering angles were predicted based on new image data turning the simulator into an autonomous vehicle. The model learned to stay centered in the lane, steer away from hazardous terrain and is able to master challenging curves at a simulated pace of 18 mph. In all consecutive five rounds the car never left the lane and stayed within the boundaries (which actually outperforms my mouse-performance).

REFERENCES

[1] Mariusz Bojarski & Davide Del Testa & Daniel Dworakowski & Bernhard Firner & Beat Flepp & Prasoon Goyal & Lawrence D. Jackel & Mathew Monfort & Urs Muller & Jiakai Zhang & Xin Zhang & Jake Zhao & Karol Zieba, *End to End Learning for Self-Driving Cars* CoRR. 2016.

[1]Figure VI shows the different angles of any given image. For any image only the center steering angle is given, to estimate driving angles as well as simulate left and right shifting maneuvers from the far right or far left .2 is being subtracted or added to the initial steering angle that corresponds to the center image.