Writeup for Behavioral Cloning project

Part of Udacity self driving car nanodegree

Sven Eriksson

May 21, 2017

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

I have done all of these things and the purpose of this report is to describe that.

1 Files Submitted & Code Quality

The code I wrote for this project is located in the folder 'code'. The file 'drive.py' for driving the car in autonomous mode is located directly in the top folder. In the same top folder one can also find this report, 'writeup_report.pdf' and 'model.h5' that contains the trained neural network.

My code is devided up into 3 files. 'model.py' that describe the model and calls the training functions from Keras. 'dataLoader.py' that reads the

comma separated files containing the names and angles for all images. 'dataLoader.py' does also divede the dataset into a training and validation set. Finally it also contains the code for creating the generators for training and validation. The last file is 'config.py' that contains the parameters I changed during training the network.

All three files contains comments and are structured in a way to make it easy to understand.

2 Model Architecture and Training Strategy

My model is based on the one NVidia used in their article 'End to End Learning for Self-Driving Cars' [1].

I did a few modifications to it in order to adapt it to this task and to try out a few things for testing and learning purposes.

2.1 Network architecture

2.1.1 Normalization and grayscale

All elements in the three color layers, red, blue, and green, were normalized to be around zero by dividing them by 255 and subtracting 0.5 from the result.

A forth layer containing the image in grayscale was created by taking the average of the three color layers in each pixel. I am aware that the network can learn to see the images in grayscale by in each pixel having the same value for each color layer. But I wanted to test how to concatenate layers and to see if it helped by manually defining this grayscale layer. I don't think it did as the result was similar and the training time was slightly longer.

Theese four layers were then used together as input for the convulisional part of the neural network.

2.1.2 Convultional layers

Input size is 66 pixels as hegith by 320 pixels as width by 4 'colors'. The activation function for these layers is 'relu'. This input is wider than the

one NVidia used and has an additional 'color'. So I decided to increase the feature depth in each layer. This has probably increased the training time but it also allows for potentially more features.

Feature depth	Filer	Stride
36	5 by 5	2 in both directions
48	5 by 5	2 in both directions
64	5 by 5	2 in both directions
80	5 by 5	1 in both directions
80	5 by 5	1 in both directions

Output size is 1 pixels as hegith by 33 pixels as width by 80 as feature depth. This is then flattened and used as input to the following fully connected layers.

2.1.3 Fully connected layers

For the fully connected layers I experimented with a few different activation functions and even though I didn't really see any difference I ended up using 'tanh' for all fully connected layers except the output layer that had no activation function.

There fully connected layers are 50% larger than those in the NVidia article [1]. This is done to deal with the input to the fully connected layers that is nearly twice the size of the one found in NVidia's article.

Number of neurons	Type
150	Internal layer
75	Internal layer
15	Internal layer
1	Output layer

2.2 Attempts to reduce overfitting in the model

An attempt to reduce overfitting has been done by adding a dropout layer between allmost all layers.

The model was trained and validated on different but similar data sets as I randomly took images from the same track.

2.3 Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

The only value that was tuned was the adjustment angle for images from the right and left camera. But that is modifing a parameter for the training data rather than the model.

2.4 Appropriate training data

Training data was randomly picked from my several runs with manual control on the easier of the two tracks.

3 Model Architecture and Training Strategy

3.1 Solution Design Approach

As I had previously encountered the NVidia article in my work I used the network described there as a starting point.

I added dropout layers to prevent overfitting. All layers were also increased in order to adjust for the difference in input size between NVidia's article and our simulation.

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

3.2 Creation of the Training Set & Training Process

I focused on driving somewhat in the center of the road during all runs and used a adjustment angle images from the right and left camera. There adjustment angles were tuned in such a way so that the network would learn to steer towards the middle should it ever see a similar image.

For a visualization of the raw images given to the network see figure 1. The network does then crop the images in a way that can be seen in figure 2.

(a) Image from the left 'camera'.

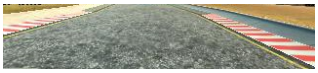
(b) Image from the center 'camera'.

(c) Image from the right 'camera'.

Figure 1: Raw images from the 3 'cameras'. All are 160 by 320 pixels.

(a) Image from the left 'camera'.

(b) Image from the center 'camera'.



(c) Image from the right 'camera'.

Figure 2: Cropped images from the 3 'cameras'. All are 66 by 320 pixels.

For each image I created a flipped copy by the time of loading it. The flipped and unflipped images were independntly picked for training and validation and for different batches. When creating a flipped copy of an image I also created a flipped copy of the recorded angle, with our without adjustment angle depending on the which camera that took the image. The distribution of angles of this augmented dataset can be seen in figure 3.

When the model failed to drive as intended I gathered more of the same kind of data and also tuned to adjustment angle to teach the network to stay towards the middle.

The successful runs around the easy track can be seen in the video clips in this github repository.

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, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

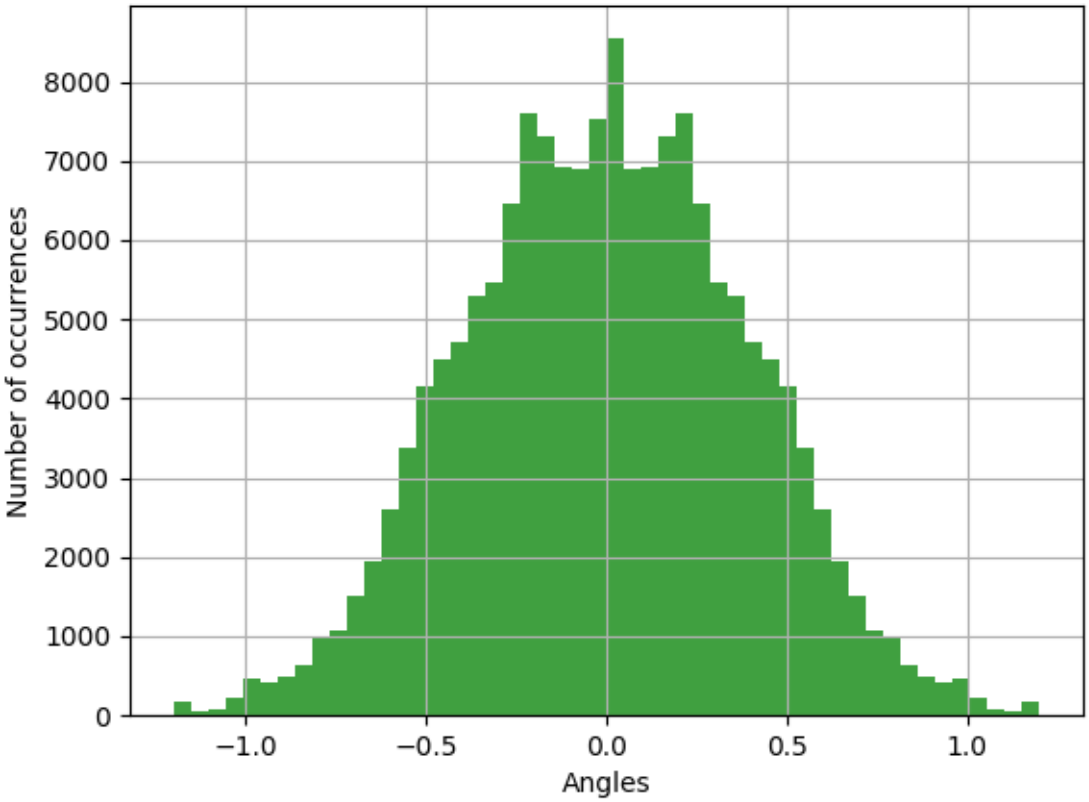


Figure 3: Histogram of the distrubition of angles in my augmented dataset. It is symmetrical due to how it was created with flipped images.