Implement of End to End Learning for Self-Driving Cars on Caffe deeplearning framework

Ruzhuo Wang
University of California, Riverside
rwang085@ucr.edu

Abstract

In the target paper [5], they trained a convolutional neural network (CNN) [3] to map raw pixels from a single front-facing camera directly to steering commands. The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. They never explicitly trained it to detect, for example, the outline of roads. Their result is said to be 98% autonomy value for a typical drive in Monmouth County NJ from their office in Holmdel to Atlantic Highlands.

So I want to find out how they achieve that much high autonomy value, since the target paper do not provide any source code, so I build 3 different but similar architecture of Pilotnet as is mentioned in the targeted paper, and apply two kinds of preprocessing methods for dataset to implement the Pilotnet on Caffe [1].

1. Introduction

CNNs have revolutionized pattern recognition. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of handcrafted feature extraction followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

End to end (E2E) [4] deep learning is the idea of instead of engineering a lot of presentations, you can output more complex things.

In this project, I build 3 similar but different architecture of Pilotnet, first one is the same as it is introduced in the target paper, second one is similar with one github project of SullyChen [6] and the third one is what I build myself. I also apply two preprocessing methods for the original dataset, aiming at improving the performance.

1.1. Three different network architecture

I build an architecture that is the same as the target paper introduced, because I think it may have the same performance as is described in the target paper, but the experiment results show that it has the worst performance compared with the architecture I build myself and the architecture of SullyChen's project.

After I figure out that the original architecture does not have good performance, I try to modify the architecture and build a new architecture based on the original one by adding some layers, it do increase the train accuracy(the test result that Caffe framework gives out, this is more like test the network on the training dataset), but it has over-fit problem.

In order to furthermore increase the training accuracy and solve over-fit problem, I find out that SullyChen successfully apply Pilotnet on the Tensorflow framework, so I build an architecture the same as he build in Tensorflow project, and it outperforms the other two architectures for train accuracy, but still can not solve the over-fit problem.

1.2. Two different preprocessing methods for dataset

I have the original driving dataset that comes from SullyChen's project, in order to improve the training accuracy and solve the over-fit problem, I want to manually extract the feature for the Pilotnet to furthermore increase the training accuracy and solve the over-fit problem, so I apply two preprocessing methods for the original dataset.

One preprocessing method is canny edge detector [2]. By applying canny edge detector on the original image, I can extract the curve of the edge in the original image include the road curve.

One preprocessing method is image-cropping. By applying this method on the original image, I can extract a smaller image that most of the contains is the road curve.

2. Related work

In SullyChen's Github repository, he successfully train a modified Pilotnet using Tensorflow framework, the training loss is shown in Figure1. That is the reason why I want to build a architecture the same as his.

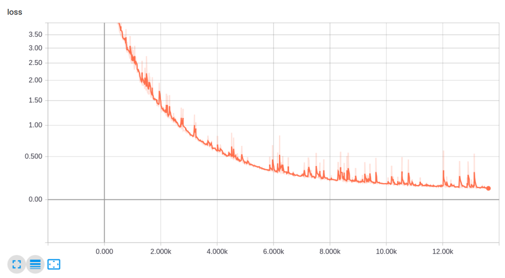


Figure 1. The training loss for SullyChen's project

3. System framework

3.1. Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

3.2. Network architecture

I build three different but similar Pilotnet architecture as is introduced bellow.

3.2.1 Original architecture

The original network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. In the target paper, they use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. As is shown in Figure2.

Figure 2. Original architecture

3.2.2 My architecture

My network consists of 20 layers, compared with original architecture, I delete one normalization layer, add 7 relu layers, add two new normalization layers and two dropout layers. As is shown in Figure3.

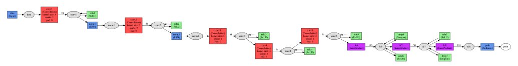


Figure 3. my architecture

3.2.3 SullyChen's architecture

SullyChen's network consists of 17 layers, compared with original architecture, SullyChen delete one normalization layer, add 7 relu layers, add two dropout layers. As is shown in Figure4.

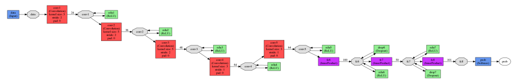


Figure 4. my architecture

3.3. Important preparation before training

3.3.1 The preparation of training and testing dataset

I write a python script(datasetreader.py) to divide the dataset into two parts, 7/8 of the driving dataset has been treat as training dataset, the other 1/8 of the driving dataset has been treat as validation dataset, the driving dataset is consist of 45568 images that captured from a front camera on a driving car. I download this dataset from SullyChen's github repository.

3.3.2 The label method for dataset

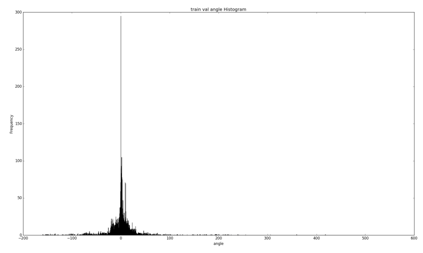


Figure 5. Steering angle distribution

Because the Pilotnet is an end to end network, so it require labeled dataset as training and testing dataset, and for Caffe deeplearning frame work, it require positive integer as label, so I write a python script(label.py) to transfer the steering angle from float number into 0 to 100 labels. As is shown in Figure5, the steering angle is like a Gaussian distribution, the majority steering angle is around 0, so I map the steering angle with the value within -140 to 140 into 1 to 98, the steering angle with the value less than -140 into 0, the steering angle with the value between within 140 to 200

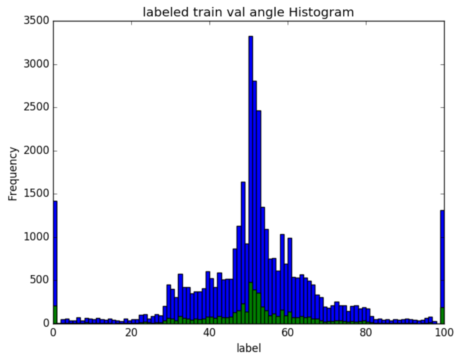


Figure 6. Labeled angle distribution

into 99, the steering angle with the value bigger than 200 into 100. The mapped steering angle is shown in Figure6, the green bar is the labeled validation dataset, the blue bar is the labeled train dataset.

4. Experiment

4.1. For different network architectures

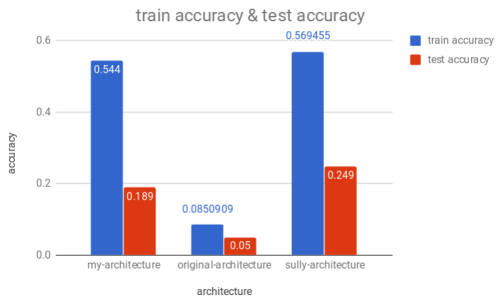
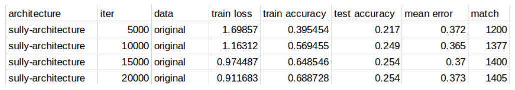


Figure 7. The accuracy performance

I train the three different network on the same training dataset, and focus on the train accuracy and test accuracy. The accuracy result in Figure7 shows that SullyChen's architecture outperforms other two. The data in Figure7 is the accuracy performance for three different architectures trained and tested on the original dataset for 10000 iterations.

4.2. Over-fit experiment

In Figure8, I can tell that, when iterations increase, the train loss decrease, the training accuracy increase, the train loss here is the test result that Caffe framework give me, but when I test the pre-trained model myself using a python script(prediction.py), and compute the test accuracy, mean error and matched label number using a python script(compute_prediction_accuracy.py), the result tell me there exit over-fit problem for pre-trained model.



architecture	iter	data	train loss	train accuracy	test accuracy	mean error	match
sully-architecture	5000	original	1.69857	0.395454	0.217	0.372	1200
sully-architecture	10000	original	1.16312	0.569455	0.249	0.365	1377
sully-architecture	15000	original	0.974487	0.648546	0.254	0.37	1400
sully-architecture	20000	original	0.911683	0.688728	0.254	0.373	1405

Figure 8. Steering angle distribution

Also in Figure9, I can tell that, most of the matched label are on the left side, right side and center, mainly because training dataset are located at those positions. The data in figure8 is the prediction result for SullyChen's architecture of 20000 iterations trained and test on the original dataset.

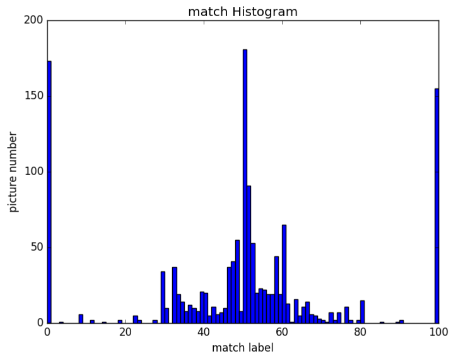


Figure 9. Matched label distribution

4.3. For different dataset preprocessing methods

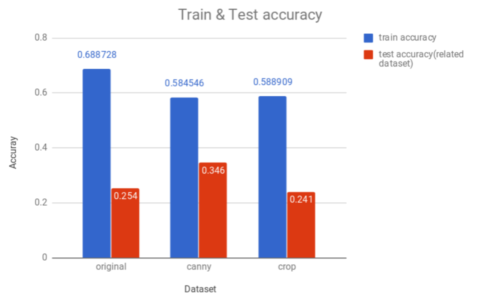


Figure 10. The accuracy performance for different preprocessing methods

As is shown in Figure10, Figure11 and Figure12, after applying canny preprocessing methods on the original dataset, the train accuracy that Caffe framework give out decrease, and the train loss the Caffe framework give out increase, but if I use the related validation data to test the accuracy of pre-trained model, the test accuracy increase(still not high) and mean error decrease, it means that canny preprocessing method can somehow solve the over-fit problem.

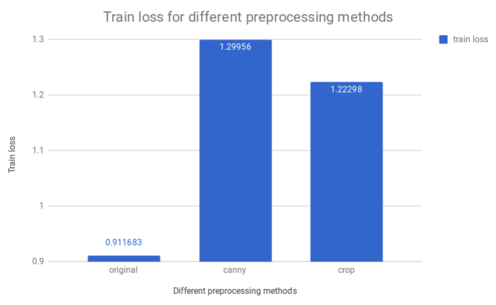


Figure 11. Train loss for different preprocessing methods

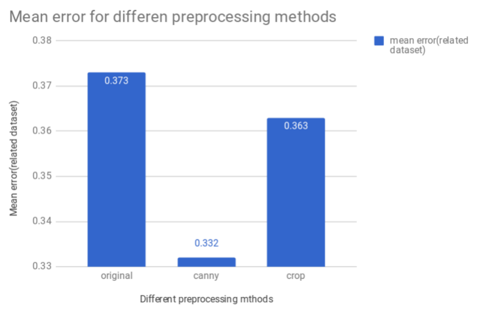


Figure 12. Mean error for different preprocessing methods

But after applying crop preprocessing method on original dataset, the train accuracy that Caffe framework give out decrease, and the train loss the Caffe framework give out increase, meanwhile, the test accuracy decrease(still not high) and mean error increase, it means that crop preprocessing method do not solve the over-fit question.

5. Conclusion

It is hard to build a workable CNN, the small difference in the architecture will result in huge difference in the CNN performance.

Dataset is quite important for the training of the CNN network, for this dataset, the labeled angle with relatively more training data has a high matched label number in the prediction result.

Canny preprocessing method can somehow solve the over-fit problem for this specific architecture, meanwhile, crop preprocessing method seems not that helpful.

6. Future work

For dataset, maybe I should find some way to augment the dataset, maybe apply the preprocessing methods for augmenting the dataset is a acceptable approach.

For the label method, it may be possible to map the steering angle as Gaussian distribution, so the labeled dataset can be more smooth, which may improve the Pilotnet performance.

It is important to be more familiar with the Caffe deep-learning framework, because the train accuracy(the test accuracy Caffe gives out) and the prediction accuracy I use python script computing is different.

References

[1] Caffe. `http://caffe.berkeleyvision.org/`.

[2] Canny edge detector. `https://en.wikipedia.org/wiki/Canny_edge_detector`.

[3] Convolutional neural network. `https://en.wikipedia.org/wiki/Convolutional_neural_network`.

[4] What does end to end mean in deep learning methods? `https://www.quora.com/What-does-end-to-end-mean-in-deep-learning-methods`.

[5] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, 2016.

[6] SullyChen. Autopilot-tensorflow. `https://github.com/SullyChen/Autopilot-TensorFlow`.