End-to-End Learning to Steer for Self-Driving Cars with Small and Efficient Networks

Norman Di Palo, MSc in Artificial Intelligence and Robotics

September 28, 2017

Abstract

In this work, I trained different neural network architectures to learn how to autonomously steer a car given a training dataset of several hours of human driving. I compared one of the most famous models, the NVIDIA architecture, with a small and parameters-efficient CNN, SqueezeNet, modified for this task. Finally, I compared their results to a new architecture that adds recurrent layers to the SqueezeNet model. The models are compared both in terms of performance that in the number of parameters needed and training time required.

1 Introduction

The field of computer vision has seen numerous breakthroughs in the recent years following the recent advancement in neural networks and deep learning research. In particular, Convolutional Neural Networks have proven to be very useful tools to perform a large variety of tasks typical of computer vision, such as obejct recognition and detection, semantic segmentation, depth estimation, and so on. These improvements had an impact on many areas of robotics, too, enhancing the ability for autonomy and learning in various environments. One field that has grown particularly in the recent years is the field of **self-driving cars**. Many companies, worldwide, are investing heavily on these new advancements of Artificial Intelligence to develop cars with growing level of autonomy. The advent of semi or fully autonomous veichles in the streets will not only create a different experience for the user, but will eventually change dramatically the urbanistic scenario of cities, rethinking completely public transportations as well as area designed for parking, allowing cities to be more human-centered and less car-centered. Short and long distance transportation will feel a great impact from the advancements of research in these areas, allowing for a more efficient use of resources and time. These are some reasons why I believe this is an important field to focus on, not only for its technological side, but for its social and urbanistic impacts too.
In this work, I designed and implemented different experiments involving various network architecture, used to learn in an end-to-end fashion how to steer

autonomously a car in various scenarios, from highways to city streets. The networks often used in the research fields are often very large in terms of parameters and thus of memory needs. Recent CNNs that achieved the best results on classification tasks have millions and millions of parameters, requiring enormous training time, memory usage and time needed for inference. This is not acceptable in the fields of robotics and thus of self driving car, that need an embedded solution that is fast and slim in terms of memory. Moreover, the networks used in self-driving cars need to be updated often and quickly, often using cellular internet connections that do not allow large data transfer. The two main architectures that were confronted are the NVIDIA model [1], one of the firsts deep CNN used for this task, that has shown remarkable results, and a modified version of the SqueezeNet architecture [2], a CNN developed recently that obtained AlexNet-like accuracy on ImageNet with 50 times less parameters. As said, this aspects are crucial to develop solutions that can perform in real time on hardware installed on a car. Lastly, I added recurrent layers on top of the modified SqueezeNet architecture to allow the network to determine steering angles based not only on the current frame, but on the temporal sequence of the last frames too.
The dataset used is made of pictures of several hours of driving in various scenarios, from highways to city streets. Each picture has an associated steering angle. This is the only information used to train the network end-to-end. I compared the results of the solutions in terms of performance (mean squared error on validation set), number of parameters and training time required. Lastly, I run a script that takes as input a video extracted from the dataset an computes in real time the steering angle, thus showing the ability of the architecture to smoothly run in real time even on a laptop using only CPU.

2 Global Architectures Comparisons

The networks architecture, as said, is designed to learn end-to-end how to steer taking as input images. Thus, the general architecture takes as input an image with the 3 RGB channels, performs a series of operations, such asconvolutions, pooling, reshaping, as finally outputs a single real valued number, corresponding to the angle of steering. The problem can be defined as a regression problem in the space of images. The loss function is the mean squared error of the predicted output and the real one, with in addition a small regularization of the parameters of the network.
Both networks have an initial structure made of convolutional and pooling layers: as it is known in the literature, this part of the network extracts features from the image, combining in a hierarcical way the features extracted from the previous layers. After this part, the bi-dimensional features get flattened into a single vector, that passes through a series of densely connected layers. The final output is then the real value of the steering. The NVIDIA and the SqueezeNet models share many common features, but have quite distinctive tracts as well. While the NVIDIA model resembles more closely a classical

Figure 1: (left) An example extracted from the dataset. (right) Features extracted from the first layer of convolution of the NVIDIA model (taken from the NVIDIA paper)

CNN, the SqueezeNet model introduces the Fire module, a group of filters that can improve the performance while using a small number of parameters.
Overall, the NVIDIA architecture uses around 450.000 parameters, a small number compared to the CNNs used for image classification, while SqueezeNet, after some modifications that I performed running several experiments, uses only 100.000 parameters. Both are able to perform real-time inference on my laptop, using an Intel Core 2 Duo at around 10 FPS.

3 Data Analysis and Preprocessing

The dataset is composed of various hours of driving in different settings, from highways to street cities. The images are 455x256 RGB, sampled at around 10 FPS. Before using this data with the networks, I performed some analysis: as it often happens in real driving scenarios, the majority of the data represents very small steering angles, thus a straight driving. This can be harmful during the training process, making the model biased towards these angles. Thus, I reduce partially the number of these frames corresponding to small steering angles, but different experiments didn't show an appreciable improvement.

Then, I performed some data preprocessing and augmentation. I cropped the images, erasing the upper part that is mostly useless in the driving process, being in made of sky, trees or buildings. To perform data augmentation, in order to give the model a larger and more various dataset, I applied a random change in the brightness of the image, thus simulating different light conditions of the road. In this way, the model is able to generalize to various environments and weathers. The NVIDIA paper suggests also rotating randomly the image or sometimes mirroring it, but I believe this is not stricly useful, and the mirroring part could potentially be harmful in settings where it is important to keep the correct lane and steer accordingly.
Some works suggest to transform the images into grayscale. While this reduces considerably the dimension, color information can be quite useful for detecting lanes, cars and obstacles.

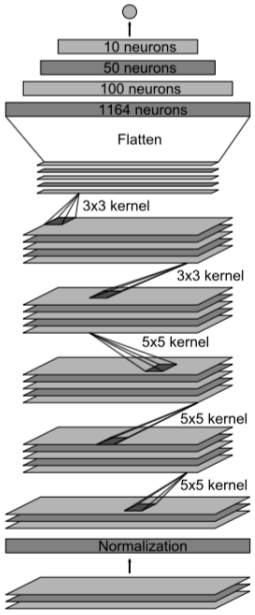

Figure 2: The NVIDIA CNN architecture.

4 NVIDIA Model

The paper by NVIDIA researchers was one of most succesful results in the field of self-driving car using deep networks and tested on a real car. This opened the field to other architectures, experiments and papers in the field of computer vision for self-driving cars and, more widely, for autonomous robotics, such as semantic image segmentation or monocular depth estimation.
I reproduced the experiments and the architecture used in that work, changing slightly the image preprocessing phase and using a smaller dataset for computational reasons. The network architecture is described in the figure, it consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers.: the image is initially normalized, and then goes through a series of convolutional layers, made of strided convolutions in the first three convolutional layers with a 22 stride and a 55 kernel and a non-strided convolution with a 33 kernel size in the last two convolutional layers. Finally, the output matrix of features gets flattened and goes through densely connected layers, with the final output being the real valued steering angle. All the activation functions used are ReLUs. The weights are trained by minimizing the mean squared error with the ground truth value, with the Adam optimizer.
The training was done with a batch size of 16, and lasted 10 epochs. After that, the validation error decreased very slowly, and given my very limited computational resources, I tried to reduce the duration of experiments in order to

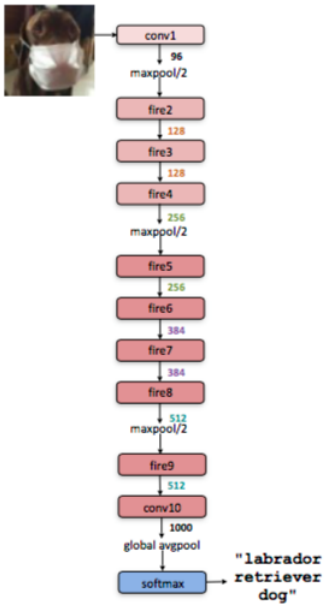

Figure 3: Original SqueezeNet architecture. In the modified model, the final layer is substituted with a densely connected layer with a single output and no activation, while the number of filters in the Fire modules is strongly reduced.

perform more of them. Each epoch took around an hour on an Intel Core 2 Duo. The learning curve is shown in the figure, compared to the SqueezeNet learning curve. Interestingly, after 10 epochs this architecture performed slightly worse than the SqueezeNet model, even having around 4 times more parameters.

5 SqueezeNet Model

The SqueezeNet architecture is an interesting and recent model, developed to achieve high accuracy in classification tasks, even having a very small number of parameters compared to many other models. The key building block of the architecture is the Fire module, described in the original paper. It is made of a set of 1x1 filters, and then 1x1 and 3x3 filters, inspired partially by the Inception module. [3]
As said, this network was initially designed for classification tasks. These kind of tasks require the extraction of quite complex features from an image, in order to recognize, for example, dogs, cars, faces, and so on. Thus, the number of different convolutional filters is quite high, and so is the number of parameters. Intuitively, we need less of these features in the driving task. The kind of information to extract from an image is more limited, being mostly the shape of the road and lanes, and the presence of obstacles or cars. So I reduced

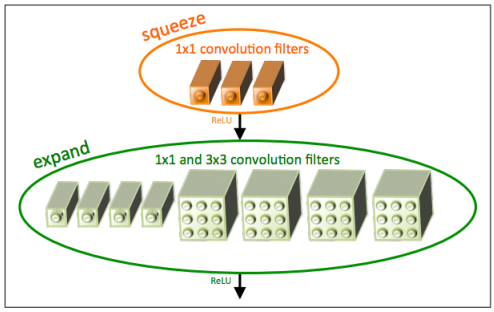

Figure 4: The Fire block of the SqueezeNet network.

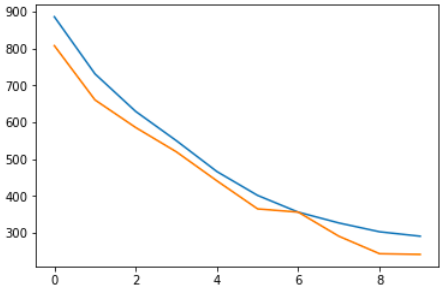

Figure 5: Comparison plots of validation loss of NVIDIA model (blue) and SqueezeNet (orange) after each epoch.

drastically the number of different filter, even of a magnitude of 10x. This resulted in an even slimmer network, made of around 100.000 parameters, that performed anyway quite well on the driving task.

The modified architecture has as output a real valued number, given as output from a densely connected layer with no activation function, while the original model has a softmax layer. The training phase is identical to the previous model: the Adam optimizer is used to minimize the mean squared error of the output. The batch size is 16, as before, with 10 epochs. This time, each epoch took around 40 minutes. The validation error, computed on 1/10 of the training set, was shown to be always lower than the NVIDIA case. (details can be seen in the figure)

6 Recurrent layers on CNN architecture

All the models explained before inferred the steering angle using only the current frame. As experience suggests, driving is a dynamical behaviour based on the history of previous control inputs and observations. Thus, I implemented an architecture that works in a similar way, by adding recurrent layers after the

convolutional layers of the NVIDIA model. As said, the convolutional layers extracts features from the image, creating a matrix of features, that is then flattened into a single vector. I inserted an LSTM layer after the convolutional layers that takes as input the temporal sequence of these vectors containing the extracted features, and outputs a single real valued number, i.e. the steering angle, or in other experiments another vector that is then given as input to different layers of fully connected nodes. This allows the network to infer the steering angle based on a series of frames, 4 frames in the evaluated case. I used the pretrained NVIDIA model from the previous experiments, freezing all the layers but the recurrent layer, and trained it again on the dataset. While the dataset in the previous cases were shuffled and single images were extracted, in this case 4 consecutive images are choosen by picking a random starting frame from the dataset. The preprocessing steps are the same as before.
The training phase of this new architecture was more unstable of the previous cases, and the final loss on a validation set was found to be very similar to the classic model, sometimes even higher. I plan to furtherly investigate this kind of network architectures in the future, trying to obtain more computational power to perform longer experiments without having to wait for an entire day.

7 Conclusion and Further Work

In this works I evaluated different kind of architectures in the task of end-to-end learning of steering for a self-driving car. In these uses, the main bottleneck with respect to classic CNN architectures are hardware constraints, as well as real-time requirements. The hardware embedded on cars is not very powerful, and needs to be updated often in a distribuited way using cellular internet connections that often do not allow large data transfers. Thus, a redesign of common architectures has to be explored in order to obtain high performances with small memory and computational requirements.
The SqueezeNet modified architecture performed slightly better than the NVIDIA one, even having 4x less parameters. Both architectures were tested in real time on 10 FPS videos, using an Intel Core 2 Duo, showing how they can easily work on possibily GPU-accelerated car mounted hardware. Along with the video, a moving picture of a steering wheel was shown, giving a visual feedback of the behaviour of the model. The movements were quite realistic and fluid, thus showing how the model is able to imitate the human driving.
In the near future I plan to extend the use of this architecture to perform semantic segmentation of street images, extremely useful in city settings, and monocular depth estimation, useful as well for a fast 3D understanding of the environment.
The code for the experiments can be found in the Jupyter Notebook, with the link to download the dataset aswell.

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* 2016.

[2] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, Kurt Keutzer *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size* 2016

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich *Going Deeper with Convolutions* 2014