# Self-Driving Car

Đorđe Marjanović
Fakultet tehničkih nauka
Novi Sad, Srbija
dj.m@gmx.com

***Abstract*—Recent advances in Deep Neural Networks (DNNs) have led to the development of DNN-driven autonomous cars that, using sensors, cameras, etc. can drive without any human intervention. Most major manufacturers including Tesla, Ford, BMW, and Google are working on building and testing different types of autonomous vehicles. Autonomous cars or self-driving cars is being introduced to society more and more and will be the next big step in the progression of personal cars. This work demonstrate autonomous driving in a simulation environment by commanding steering and throttle control inputs from raw images. Supervised learning was used to map images to steering and throttle inputs based on example data collected from simulator. The training data was augmented with adjusted images which allow the model to generalize better. Model selection was used to determine an appropriate neural network architecture and this process is documented. The capability of the model to drive autonomously in a real-time closed-loop simulation was evaluated. This work demonstrates the capability of a deep convolutional neural network to accomplish aspects of autonomous driving using only direct image data as input.**

***Keywords—self-driving car, autonomous car, deep learning, convolution neural networks, behavioral cloning***

## I. Introduction

Significant progress in Machine Learning (ML) techniques like Deep Neural Networks (DNNs) over the last decade has enabled the development of safety-critical ML systems like autonomous cars. Several major car manufacturers including Tesla, GM, Ford, BMW, and Google are building and actively testing these cars. Recent results show that autonomous cars have become very efficient in practice and already driven millions of miles without any human intervention [1]. Twenty US states including California, Texas, and New York have recently passed legislation to enable testing and deployment of autonomous vehicles [2]. As self-driving car technology improves, it is likely that more cars on the road will be equipped with the technology to autonomously drive, but more importantly, drive safely and reliably.

This project explores the possibility of employing behavior cloning to construct a model to achieve steering and throttle prediction using only input images. The goal is to provide a proof of concept, that image data can indeed provide the information required to drive a car autonomously.

As a secondary unfold, the project explores the notion of what successful autonomous driving entails. Various metrics indicating the distance from the center of the lane and the average speed while staying on the track were evaluated. The project drew inspiration from a challenge introduced in the Udacity self-driving car nanodegree [3], having the prime objective of autonomous driving by predicting steering.

## II. Related Work

The first step towards the area of autonomous vehicle navigation was taken while designing the Autonomous land vehicle in Neural Network (ALVINN) [4]. Although the computing power at the time was limited, the network demonstrated the use input pixels for autonomous vehicle navigation.

In 2016, NVIDIA released a paper regarding a similar idea that benefited from ALVINN. In the paper [5], the authors used a relatively basic CNN architecture to extract features from the driving frames. The layout of the architecture can be seen in Figure 1. Augmentation of the data collected was found to be important. The authors used artificial shifts and rotations of the training set. Left and right cameras with interpolated steering angles were also incorporated. This framework was successful in relatively simple real-world scenarios, such as highway lane-following and driving in flat, obstacle-free courses.

There are more research initiatives applying deep learning techniques in autonomous driving challenges. Another line of work is to treat autonomous navigation as a video prediction task. Comma.ai [6] has proposed to learn a driving simulator with an approach that combines a Variational Auto-encoder (VAE) [7] and a Generative Adversarial Network (GAN) [8]. Their approach is able to keep predicting realistic looking video for several frames based on previous frames despite the transition model being optimized without a cost function in the pixel space.

Moreover, deep reinforcement learning (RL) has also been applied to autonomous driving [9], [10]. RL has not been successful for automotive applications until some recent work shows the deep learning algorithms ability to learn good representations of the environment. This was demonstrated by learning of games like Atari and Go by Google DeepMind [11], [12]. Inspired by these work, [9] has proposed a framework for autonomous driving using deep RL. Their framework is extensible to include RNN for information integration, which enables the car to handle partially observable scenarios. The framework also integrates attention models, making use of the glimpse and action networks to direct the CNN kernels to the places of the input data that are relevant to the driving process.

## III. CNN Architecture

CNNs are feedforward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells, motivates their architecture. CNN architectures come in several variations; however, in general, they consist of convolutional and pooling (or subsampling) layers, which are grouped into modules. Either one or more

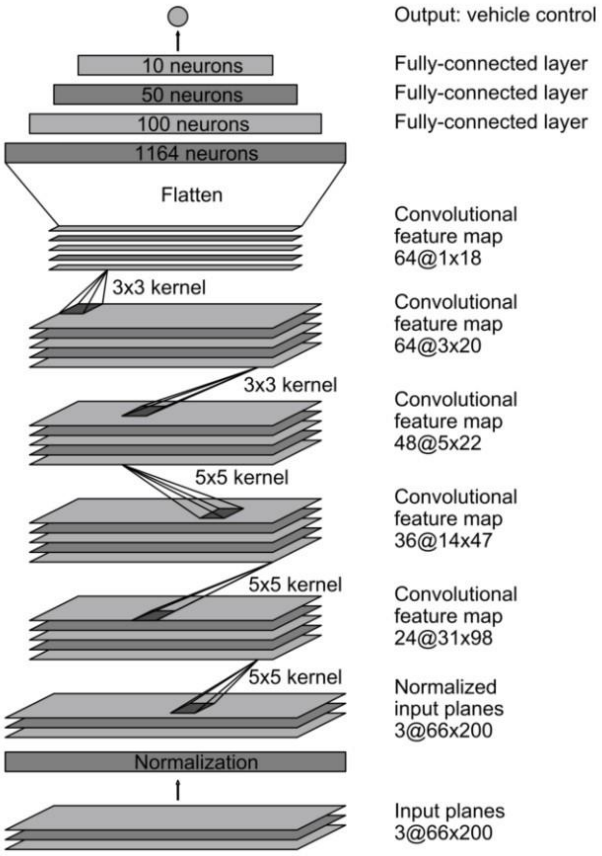


Figure 1. CNN architecture

fully connected layers, as in a standard feedforward neural network, follow these modules. Modules are often stacked on top of each other to form a deep mode.

### A. *Convolutional Layer*

The convolutional layers serve as feature extractors, and thus they learn the feature representations of their input images. The neurons in the convolutional layers are arranged into feature maps. Each neuron in a feature map has a receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights. Inputs are convolved with the learned weights in order to compute a new feature map, and the convolved results are sent through a activation function [14].

### B. *Pooling Layer*

The purpose of the pooling layers is to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. Pooling is an important component of convolutional neural networks for object detection. Initially, it was common practice to use average pooling aggregation layers to propagate the average of all the input values, of a small neighborhood of an image to the next. However, in more recent models, max pooling aggregation layers propagate the maximum value within a receptive field to the next layer. Formally, max pooling selects the largest element within each receptive field. Figure 2 illustrates the difference between max pooling and average pooling. Given an input image of size 4 × 4, if a 2 × 2 filter and stride of two is applied, max pooling outputs the maximum value of each 2 × 2 region, while average pooling outputs the average rounded integer value of each subsampled region. While the motivations behind the migration toward max pooling are addressed in [13] [14], there are also several concerns with max pooling, which have led to the development of other pooling schemes. [14]

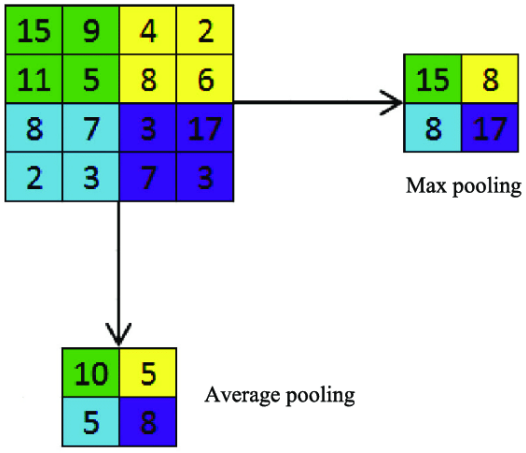


Figure 2. Average versus max pooling

### C. *Training*

CNNs, and ANNs in general use learning algorithms to adjust their free parameters (i.e., the biases and weights) in order to attain the desired network output. The most common algorithm used for this purpose is backpropagation. Backpropagation computes the gradient of an objective (also referred to as a cost/loss performance) function to determine how to adjust a network's parameters in order to minimize errors that affect performance. A commonly experienced problem with training CNNs, and in particular DCNNs, is overfitting, which is poor performance on a held-out test set after the network is trained on a small or even large training set. This affects the model's ability to generalize on unseen data and is a major challenge for DCNNs that can be assuaged by regularization. [14]

## IV. Data Generation & Simulator

Data generation is inspired by Nvidia's Self-Driving Car's data collection process. They attached 3 cameras to a car. One is placed in the center and the other 2 are placed on each side of the car. They recorded the steering wheel's data-capturing the steering angle. So, captured data:

1. Images from center, left and right cameras
2. Steering angle
3. Speed of the car
4. Throttle
5. Brake

Imitation of this system is Udacity's Self Driving Car simulator which will be used in this project. It is open-source 3D driving simulator rendered in Unity for their online course on using deep learning to train an autonomous driving agent. The viewpoint for collecting data is from behind the car, while labeled images are generated from a first-person

drivers perspective of the road. The project features two built-in tracks, as well as a custom track creation module. This simulator has few customizability and does not feature other cars. The simulator has 2 modes - Training mode and Autonomous mode. In the training mode, the simulator captures the images from the 3 cameras, speed value, throttle value, brake, and steering angle. These recorded values are stored in a CSV file.

## V. Data Preprocessing

In the driving mode for data collection, the vehicle was driven in the center of the lane. The large majority of the training data corresponds to zero steering input. There are some left steering image but right steering images are fewer. If model is trained on this data without accounting for the imbalance, there is a high probability that your model will have a great bias towards the 0-steering angle images, the car did not have enough examples to learn how to return to the center of the road if for any reason it drifted apart from center. Those are following solutions for this problem: cutting down the straight drive images, left and right camera images have to be included, augment turn images. Augmentation implies: the turn images by flipping them horizontally, adjusting of steering angle by steering correction factor (+0.2/-0.2 for left/right images), image translation and brightening [17].

Simulator saves images in RGB format, as NN in this case accepts images in YUV encoding, RGB images are converted to YUV. Figure 3 shows how RGB and YUV image look like.

RGB Image

YUV Image

Figure 3. RGB and YUV image

The images are normalized (image data divided by 127.5 and subtracted 1.0) to avoid saturation and make gradients work better.

Not all the informations in the image are required or maybe some information might hinder the learning of Neural Network. The only thing the Neural Network needs to concentrate on is the road while ignore everything else (the sky, the car's bonnet and the side view of the road). This data will be cropped out of image to retain only the road information in the image. The images before and after cropping are shown in Figure 4.

Before cropping

After Cropping

Figure 4. Images before and after cropping

## VI. Network Architecture

Neural network training it's weights to minimize the mean squared error between the steering command output by the network and the command of either the human driver. The design of the network is based on the Nvidia model [5] shown in Figure 1. Following adjustments are added to the model: Lambda layer for image normalization, dropout layer to avoid overfitting after the convolution layers. Final model looks like as follows:

- Image Normalization
- Convolution: 5x5, filter: 24, strides: 2x2
- Convolution: 5x5, filter: 36, strides: 2x2
- Convolution: 5x5, filter: 48, strides: 2x2
- Convolution: 3x3, filter: 64, strides: 1x1
- Convolution: 3x3, filter: 64, strides: 1x1
- Dropout (0.5)
- Fully connected: neurons: 100
- Fully connected: neurons: 50
- Fully connected: neurons: 10
- Fully connected: neurons: 1 (output)

Activation function for every layer (except output) is ELU (Exponential linear unit) which is more suited for this task than RELU, and ELU takes care of the Vanishing gradient problem. Total number of parameters for this model is 252219. This application is wrote in Python 3, Keras is used as deep learning framework.

## VII. Training, Validation and Evaluation

The data collected during the Data Generation part involved collecting all the camera images, steering angle and others while a human driver was driving the car. Application is going to train a model that clones how the human was driving the car - essentially clones the driver's behavior to different road scenarios. This is called **Behavioral Cloning**. To formally define it - Behavioral cloning is a method by which human skills can be captured and reproduced in a computer program.

Based on input images, the Neural Network would output a single value - the steering angle. Essentially, based on the input images, the Neural Network decides by what angle the car must be steered. This output value is compared with the steering data collected from human driving to compute the error in the Neural Network's decision. With this error, the model uses Backpropogation algorithm to optimize the parameter (weights) of the model to reduce this error.

Mean-squared error was used as loss function to measure how close the model predicts to the given steering angle for each image. To optimize this loss, the Adam optimizer was used [16] with learning rate = $1.0e^{-4}$. Adam with set parameters (decay – can be set as learning rate/batch size) computes adaptive learning rate.

Collected data is splitted into training and validation data in proportion 80:20. ModelCheckpoint from Keras is used to save the model only if the validation loss is improved which is checked for every epoch.

Evaluation of neural network is done in same simulator that follows the Server-Client architecture as shown on Figure 5. The server will be the simulator and the client would be the Neural Network, or rather the Python program. This whole process is a cyclic feedback loop. The simulator outputs the images, the python program analyses it and outputs the steering angle and the throttle. The simulator receives this and turns the car accordingly. And the whole process goes on cyclically.

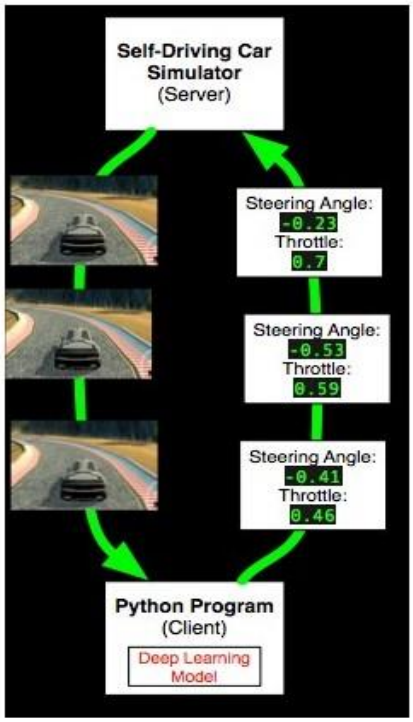


Figure 5. Simulator (Server) – Neural Network (Client)

## VIII. Conclusion and Future Work

This work demonstrated autonomous steering and throttle control in a closed-loop, real-time simulation using only raw images as input to predict steering angle. This proof-of-concept shows the power of deep convolutional neural networks to learn the necessary visual features without traditional visual processing steps such as lane detection. The goal of autonomous driving in this simulation was accomplished, but this work could be extended in following ways.

The main lack of behavioral cloning is that it cannot be better than human driver whose behavior is imitated, so the next main goal will be implementation of self-driving car with Deep Reinforcement Learning. With reinforcement learning car learns from itself and interaction with environment by getting rewards from its actions either positive or negative and during training model predictions getting better and better. Future work will consider adding LSTM layer for throttle control predictions. In real environment model would have to be able to handle the different conditions – snowy, rain etc. Generate adversarial models, GANs, could be used to transform a summer training set into a winter one. Additionally, GANs could be used to generate more scenes with sharp angles. Another important extension to this work would be to use a simulation environment with more richness and variety to demonstrate the potential of extending this to real world driving.

## References


[1] 2017. The Numbers Don't Lie: Self-Driving Cars Are Getting Good. https://www. wired.com/2017/02/california-dmv-autonomous-car-disengagement. (2017).

[2] 2017. Autonomous Vehicles Enacted Legislation. http://www.ncsl.org/research/ transportation/autonomous-vehicles-self-driving-vehicles-enacted-legislation. aspx. (2017).

[3] Udacity-An open source self-driving car https://github.com/udacity/ self-driving-car-sim .

[4] D. A. Pomerleau. Alvinn, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, Computer Science Department, 1989.

[5] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

[6] E. Santana and G. Hotz. Learning a driving simulator. arXiv preprint arXiv:1608.01230, 2016

[7] D. P. Kingma and M. Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014

[9] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. Autonomous Vehicles and Machines, Electronic Imaging, 2017

[10] S. Shalev-Shwartz, S. Shammah, and A. Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. arXiv preprint arXiv:1610.03295, 2016

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016

[13] Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In Proc. Computer Vision and Pattern Recognition Conference (CVPR'07). IEEE Press, 2007

[14] Rawat, Waseem, and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review." *Neural computation* 29.9 (2017): 2352-2449.

[15] Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004. Final technical report. URL: http://net-scale.com/doc/net-scale-dave-report.pdf

[16] D. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014

[17] Du, Shuyang, Haoli Guo, and Andrew Simpson. "Self-driving car steering angle prediction based on image recognition." *Department of Computer Science, Stanford University, Tech. Rep. CS231-626* (2017).