Behavioural Cloning Applied to Self-Driving Car on a Simulated Track using the Nvidia model

Alan Afif Helal
IT Department
Computer Engineering
Federal University of Espírito Santo
Vitória, Espírito Santo
Email: alan@helal.com.br

***Abstract*—The idea to build intelligent systems that model human behavior has been studied for over a century. Despite all the progress in the field, there is a long way to go in trying to model how the human brain works. This work aims to build a convolutional neural network implementing the model proposed by Nvidia for self-driving cars. Six different scenarios were tested to evaluate the model performance. After training the neural network was able to drive the autonomous car satisfyingly.**

I. Introduction

Deep learning is a subarea of Artificial Intelligence (AI) that can be used for many daily tasks [1]. One of the main challenges faced nowadays is how to train an AI to drive cars autonomously. A lot of projects for autonomous car uses Light Detecting And Ranging (LiDAR)[1] for positioning but there are other implementations based only of pictures from cameras placed around the car [2].

One of this implementations is proposed by Nvidia[2]. The model consists of a trained convolutional neural network (CNN) to map raw pixels from a single front-facing camera directly to steering commands. Therefore, there is no need of a LiDAR to locate the car on the road. It's a cheaper and less complex solution.

This project aims to implement the Nvidia and evaluate its perfomance on a driving simulator. For training of the neural network, a data-set with more than 10,000 photos taken by the driving simulator was used. Training requires computing power hardly found in personal computers in our homes. To solve this problem, Google Colab was used for training.

This work was structured as follows:

- Section 2 explains the creation of the neural network, the handling of training and testing datasets, and the techniques used to improve the performance of the neural network.
- Section 3 shows the results found in each of the six analyzed scenarios.
- Section 4 discuss the results and possible explanations for the problems encountered.
- Section 5 presents the conclusion and suggestions for future work.

[1]https://en.wikipedia.org/wiki/Lidar

II. Materials and Methods

Since we are going to train a Convolutional Neural Network to clone the driving behavior of a human, we must provide a data-set containing images of how a human drives. We can get this images from the driving simulator. For this work the driving simulator provided by Udacity[2] was used.

The simulator has two modes, training and autonomous, and two different tracks as shown in Figure 1

Fig. 1. Main screen of the driving simulator provided by Udacity

For this work, more than 10,000 photos from 3 different cameras were used. Two cameras are located on the left and right side of the simulated car and one camera is located in the middle of the windshield of the simulated car. Figure 2 shows an example of photos taken by each one of the cameras at the same time.

Figure 3 shows the model proposed by Nvidia. The first layer of the network performs image normalization. The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. The five convolutional layers are followed with three fully connected layers leading to an output control value which is the inverse turning radius [2].

[2]https://github.com/udacity/self-driving-car

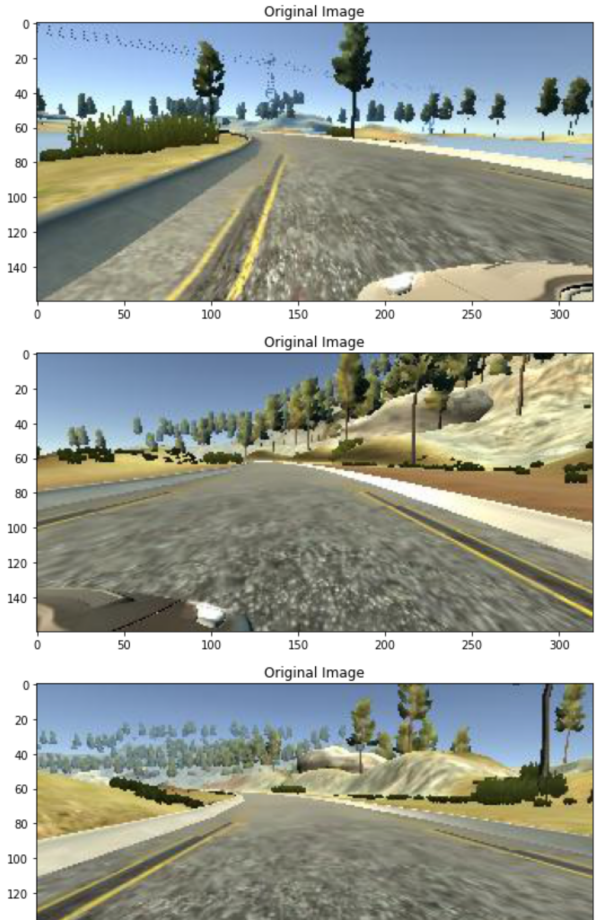

Fig. 2. Three different views of the same location taken by each camera placed around the simulated car.

The training of the neural network consists of Images being fed into a Convolutional Neural Network which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the Convolutional Neural Network are adjusted to bring the Convolutional Neural Network output closer to the desired output [2], as shown in Figure 4.

Activation functions are used in neural networks to compute the weighted sum of input and biases, which is used to decide if a neuron can be fired or not. In this work two activation functions were used:: ELu in the convolution layer and fully connected layers, and softmax in the output layer.

The Exponential Linear Unit. (ELU) activation function has been the most widely used activation function for deep learning applications. It is a faster activation function with better performance and generalization in deep learning [4]. The ELU activation function is given by Equation 1.

$$f(x) = \begin{cases} x_i, \ if \ x_i \geq 0 \\ \alpha(e^x - 1), \ if \ x_i < 0 \end{cases} \tag{1}$$

The Softmax function is an activation function used for multivariate classification tasks that compute the probability

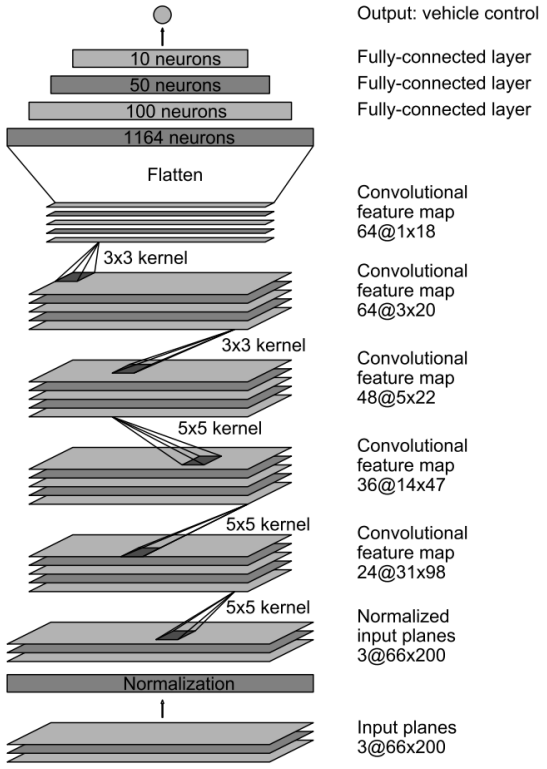

Fig. 3. Nvidia architecture.

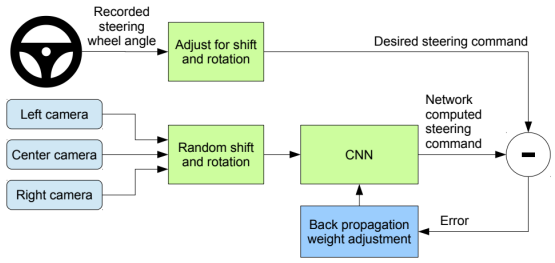

Fig. 4. Training the Convolutional Neural Network. Source: Nvidia [2]

distribution from a vector of real numbers. The Softmax function produces an output which is a range of values between 0 and 1, with the sum of the probabilities been equal to 1 [4]. The softmax activation function is showed in Equation 2.

$$f(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \tag{2}$$

Python[3] programming language with Keras[4], Tensorflow[5] and Numpy[6] libraries were used to program the neural network. Since we need a lot o computational power, we used

[3]https://www.python.org
[4]https://keras.io
[5]https://www.tensorflow.org
[6]https://numpy.org

the Google Colaboratory[7] platform. That way, we do not need to set up an environment for the training.

It is known that we can use GPU[5] as a hardware accelerator to perform faster training. Unfortunately, not everyone has access to High-End GPUs at home. But this issue has been worked around through Google Colab. For a few limited time, for free, we chose GPU as our hardware accelerator to perform the neural network training.

A. Batch Normalization

Training a neural network is a very expensive and complicated task. For this, several techniques make this task less complicated. Batch normalization[6] allows you to reach training thresholds with fewer training steps. Therefore, this technique was used to reduce time and computational cost in training.

B. Data Augmentation

Fortunately, the training data-set can be easily expanded using the Data Augmentation[7] technique. With it, new images created digitally just by rotating the original image horizontally can be introduced in training. These digitally created images closely match the photos in the data-set, as we have several photos with the same animal walking around the camera. The use of Data Augmentation is an effective technique to considerably improve the performance of convolutional neural network training. Figure 5 shows an example of how Data Augmentation works when applied to one of the images from the training data-set.

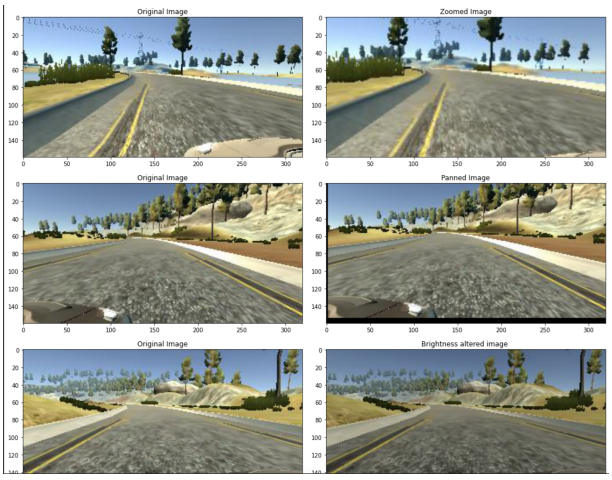

Fig. 5. Example of a digitally created image from the training data-set

C. Dropout Layer

The dropout technique is a simple way to avoid neural network overfitting after the training. It improves the performance of the neural network training in some cases [8]. On the other hand, the use of Dropout on Convolutional Neural Networks fails to obtain a noticeable performance improvement [9]. The effectiveness of dropout for convolutional neural networks is further reduced [9] if you use other regularization techniques such as the one discussed in the previous subsections on this paper. It is not a general rule, but in our tests, the use of the dropout technique resulted in a simple improvement in the accuracy of the network. Therefore, it was decided to keep a dropout layer between the first and second dense layer.

D. Image Pre-processing

The images used in the dataset have many elements that are not useful for training, such as:

- the top of the photo has sky and trees
- the bottom of the photo shows a part of the car
- the sides of the photo have part of the environment beyond the track

All these items mentioned above do not contribute positively to training. The important thing for learning is to know the limits of the track as we are interested in defining the position of the car's steering wheel. For this, the image will go through a pre-processing to remove these elements from the image. Additionally, the image color can also be changed to a scale that facilitates training. Then the image's color scheme is changed, a Gaussian filter is applied, the image is resized and then normalized. Thus, the image that will be used in training has only the elements important for training and has a size substantially smaller than the original image, as shown in Figure 6.

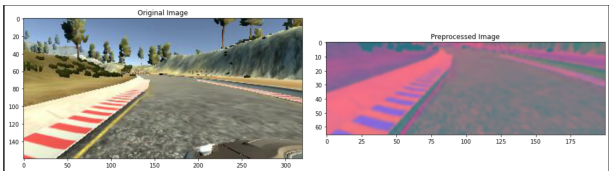

Fig. 6. Original image and image pre-processed contained only useful information for the neural network training.

E. Configuring the data-set for Performance

Since we are dealing with a huge amount of data, we need to configure the data-set for performance. We applied buffered prefetching to avoid I/O blocking when loading data. For this, we used two methods defined in the Keras library: cache() and prefetch(). The cache() method keeps the images in memory after they're loaded off disk during the first epoch of training. This technique ensures the data-set does not become a bottleneck while training the model. Given that the dataset is too large to fit into memory, this method is used to create a performant on-disk cache. The prefetch() method overlaps data preprocessing and model execution while training.

F. Optimizer and Loss Function

Before the model compilation, we need to choose the optimizer and the loss function. The Adam optimizer is often used due to its computational efficiency and little memory requirements. Additionally, it is well suited for problems that

[7]https://colab.research.google.com

are large in terms of data and/or parameters[10]. Since we have three classes, the loss function used is the Sparse Categorical Cross Entropy.

G. Training and Validation

For training, the data-set was divided between training and validation. We used 80% of the images in the training data-set to train the neural network, and 20% for validation. The training consisted of 10 epochs of learning.

H. The Neural Network

If you wish to use the neural network presented in this paper, there is a step-by-step guide provided in the author's GitHub[8].

III. Results

To evaluate the model, the original Nvidia paper proposed the following metric [2], show in Equation 3:

$$autonomy = (1 - \frac{(number\_of\_interventions) \times 6}{elpased\_time[seconds]}) \times 100 \tag{3}$$

These interventions occur when the simulated vehicle departs from the center line by more than one meter. When this happens, there is a need of human intervention to keep the car in the middle of the road. Since the simulator used in this work doesn't allow human intervention on autonomous mode and there is no way to find when the car is one meter away from the middle of the road, another way to evaluate the model was proposed.

Six scenarios were tested to evaluate the neural network training:

- Scenario 1: Speed limit of 10 mph on Track 1
- Scenario 2: Speed limit of 20 mph on Track 1
- Scenario 3: Speed limit of 30 mph on Track 1
- Scenario 4: Speed limit of 10 mph on Track 2
- Scenario 5: Speed limit of 20 mph on Track 2
- Scenario 6: Speed limit of 30 mph on Track 2

Track 1 was used to get the photos used to train the neural network. Track 2 is a complete different track, used to test the model generalization (the car should be able to drive on any kind of roads). Track 1 is a circuit and on Track 2 there is a starting and finishing line. The test consist of:

- One complete lap on the track for each scenario.
- Collect the number of times the car went off road.
- For each lap, verify if the car was able to complete the lap or if it crashed.

The autonomy of the car was measured using Equation 4, where offroad is the number of times the car went off-road and time is the time to complete one lap, in seconds:

$$aut = \begin{cases} 0, \; if \;\; car \;\; crashed \\ (1 - \frac{offroad \times 6}{time[seconds]}) \times 100 \; if \;\; car \;\; didn't \;\; crash \end{cases} \tag{4}$$

[8]https://github.com/HelalBR/CognicaoVisual

A. Scenario 1

This scenario was testing the trained model on the same track used to gather the training data-set with a speed limit of 10 miles per hour. The results of each lap (time to complete and times that the car went off road) are show in Table I. This was the best scenario with an mean autonomy of 92,59%.

Run	Time (s)	Off road	Autonomy (%)
1	275,34	3	93,46
2	274,88	3	93,45
3	275,46	3	93,47
4	276,44	5	89,15
5	275,02	4	91,27
6	274,94	3	93,45
7	275,52	4	91,29
8	275,87	3	93,48
9	275,34	3	93,46
10	275,61	3	93,47

TABLE I: Time to complete the lap and times that the car went off road on Track 1 with speed limit of 10 mph

B. Scenario 2

This scenario was testing the trained model on the same track used to gather the training data-set with a speed limit of 20 miles per hour. The results of each lap (time to complete and times that the car went off road) are show in Table II. This scenario had an mean autonomy of 83,08%.

Run	Time (s)	Off road	Autonomy (%)
1	149,76	4	93,46
2	151,34	4	93,45
3	150,54	4	93,47
4	149,67	4	89,15
5	151,12	4	91,27
6	153,98	4	93,45
7	154,77	5	91,29
8	152,87	4	93,48
9	154,44	5	93,46
10	155,03	5	93,47

TABLE II: Time to complete the lap and times that the car went off road on Track 1 with speed limit of 20 mph

C. Scenario 3

This scenario was testing the trained model on the same track used to gather the training data-set with a speed limit of 30 miles per hour. The results of each lap (time to complete and times that the car went off road) are show in Table III. In some runs the car crashed and did not complete the lap, represented by a dash on Table III. This scenario had an mean autonomy of 58,59% considering only the runs that the car did not crash and completed the lap.

Run	Time (s)	Off road	Autonomy (%)
1	106,88	7	60,70
2	-	-	-
3	-	-	-
4	110,12	8	56,41
5	-	-	-
6	109,87	8	56,31
7	107,49	7	60,93
8	-	-	-
9	-	-	-
10	-	-	-

TABLE III: Time to complete the lap and times that the car went off road on Track 1 with speed limit of 30 mph

D. Scenario 4

This scenario was testing the trained model on a different track used to gather the training data-set with a speed limit of 10 miles per hour. The results of each The results of each run (time to go from the start line to the finish line and times that the car went off road) are show in Table IV. In some runs the car crashed and did not complete the lap, represented by a dash on Table IV. This scenario had an mean autonomy of 91,16% considering only the runs that the car did not crash and completed the lap.

Run	Time (s)	Off road	Autonomy (%)
1	426,87	5	92,97
2	430,14	7	90,24
3	428,12	6	91,59
4	427,41	6	91,58
5	431,81	7	90,27
6	429,65	7	90,22
7	-	-	-
8	432,77	7	90,30
9	427,76	5	92,99
10	531,94	7	90,28

TABLE IV: Time to complete the lap and times that the car went off road on Track 2 with speed limit of 10 mph

E. Scenario 5

This scenario was testing the trained model on a different track used to gather the training data-set with a speed limit of 20 miles per hour. The results of each run (time to go from the start line to the finish line and times that the car went off road) are show in Table V. In some runs the car crashed and did not complete the lap, represented by a dash on Table V. This scenario had an mean autonomy of 70,88% considering only the runs that the car did not crash and completed the lap.

Run	Time (s)	Off road	Autonomy (%)
1	214,88	9	74,87
2	-	-	-
3	-	-	-
4	223,76	12	67,82
5	219,70	11	69,96
6	-	-	-
7	-	-	-
8	-	-	-
9	-	-	-
10	-	-	-

TABLE V: Time to complete the lap and times that the car went off road on Track 2 with speed limit of 20 mph

F. Scenario 6

This scenario was testing the trained model on a different track used to gather the training data-set with a speed limit of 30 miles per hour. Unfortunately, the car crashed on every run. Since the evaluation used on the other scenarios computes the time to complete the track and the times the car went off road, this scenario had an autonomy of 0.0%, i.e. the car was not able to complete the track without human intervention.

IV. Discussion

The idea of this work was to implement and test the model proposed by Nvidia on a simulated environment. One of the main differences about this model, regarding autonomous cars, is the fact that it uses only images captured by cameras placed around the car.

The data-set used to train the neural network was made of photos taken by the cameras when a real human was driving on Track 1. That is, the data-set is biased by how good the person was driving during the creation of the data-set. Each entry of the training data-set consists of three photos (taken by the left, right and middle camera), the normalized angle of the gas pedal, the normalized angle of the wheel and the normalized angle of the reverse pedal (if the car is going forward, the reverse pedal angle is zero).

Since the data collection was made driving a simulated car on a computer, the steering angles are not very not very accurate as they were obtained using the keyboard's directional arrows. This interferes directly on the quality of the trained model. If was used a real wheel attached to the computer, the training data-set would contain better information about the steering angle to feed the neural network.

Even with these limitations, the neural network using the model proposed by Nvidia was able to clone a human behavior when it comes to driving cars. It is a good starting point for all areas: from teaching about autonomous cars to developing new cheaper technologies.

V. Conclusion

We conclude that deep learning could enable the inexpensive, high-volume, and even real-time collection of different aspects of human behavior, especially about how we drive.

The Convolutional Neural Network Was able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection (no need of explicit labels during training). The model was able to generalize the learned behavior on a different track.

A. Future Works

For future work, it is suggested to use a training data-set with more hours of driving on different tracks. Also, it is important to test the model on different types of track, like highways, dirt road, residential area etc. Since the model only uses the image from cameras to locate the car, it is important to test the model on different climate conditions like rain or fog.

ACKNOWLEDGMENT

The author would like to thank Prof. Dr. Alberto Ferreira de Souza (Federal University of Espírito Santo) for assisting the neural network implementation, and Udacity for making the autonomous car simulator available.

REFERENCES

[1] Shinde, P. & Shah, S. A Review of Machine Learning and Deep Learning Applications. *2018 Fourth International Conference On Computing Communication Control And Automation (ICCUBEA)*. pp. 1-6 (2018)

[2] Bojarski, M., Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. & Zieba, K. End to End Learning for Self-Driving Cars. (2016)

[3] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, California, USA: 3rd International Conference on Learning Representations, 2015.

[4] C. Nwankpa and W. Ijomah and A. Gachagan and S. Marshall, *Activation Functions: Comparison of Trends in Practice and Research for Deep Learning*, 2018.

[5] A. Gajurel and S. Louis and F. Harris, *GPU Acceleration of Sparse Neural Networks*, 2020.

[6] S. Ioffe and S. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, Lille, France: 3Proceedings of the 32nd International Conference on Machine Learning, 2015.

[7] A. Mikolajczyk and M. Grochowski, *Data augmentation for improving deep learning in image classification problem*, International Interdisciplinary PhD Workshop (IIPhDW), 2018.

[8] N. Srivastava and G. Hinton and A. Krizhevsky and I. Sutskever and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15, 2014.

[9] S. Cai and Y. Shu and W. Wang and G. Chen and B. Ooi, *Efficient and Effective Dropout for Deep Convolutional Neural Networks*, 2020.

[10] D. Kingma and P. Diederik and J. Ba, *Adam: A Method for Stochastic Optimization*, California, USA: 3rd International Conference for Learning Representations, 2015.