Self-driving through a deep recurrent convolutional neural model.

Luna-Álvarez Antonio
Depto. de Ciencias Computacionales.
CENIDET-TecNM
Cuernavaca, Mexico
jesus.luna18ce@cenidet.edu.mx

Mújica-Vargas Dante
Depto. de Ciencias Computacionales.
CENIDET-TecNM
Cuernavaca, Mexico
dantemv@cenidet.edu.mx

***Abstract*—The autonomous driving in cars is a task of great complexity due to the multiple factors to be considered. With the popularization of deep learning, great advances have been made in this domain, achieving acceptable results. However, a constant defect in the works reported in the literature is the lack of consideration of time as a factor of great importance, considering only current scenes being a reactive method. This document shows the experimentation with a recurrent convolutional neuronal model, which contemplates previous actions to take the current decisions.**

***Index Terms*—Deep Learning, Convolutional Model, Recurrent Model, Time-distributed, Self-driving, Autonomy.**

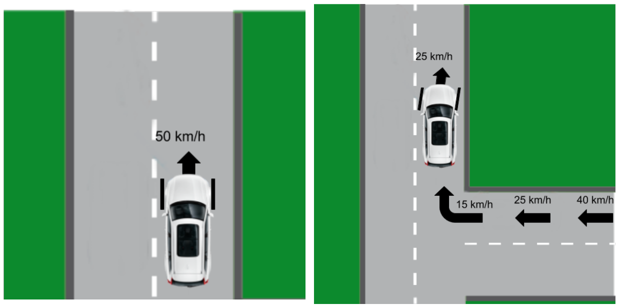
I. Introduction

Autonomous vehicles are robotic systems that must have the ability to navigate through the environment for which they were designed, independently and avoiding obstacles or situations that compromise the safety of the vehicle and its passengers. The main component of a self-driving car is the decision module, which can be controlled by a classification model that will mainly predict the steering angle and vehicle speed [1]. In the study of the state of the art it has been observed that of the most reliable classification models are given by a Neural Network Based on Deep Learning (DNN).

Although good self-driving results have been documented using deep learning, the importance of time in the driving task has been ignored. A person with sufficient experience driving in urban areas foresees the following actions in advance, for example braking distances. In the same way it makes decisions taking into account the actions taken in previous immediate times. For example, the speed and angle of the vehicle after taking a curve, as shown in Fig. 1.

Similar to how a human driver does it, a neural model dedicated to self-driving receives the image of the road as input. To interpret this image a Convolutional Neural Network (CNN) is used. This type of neural network extracts the most important characteristics, as could be the lines of the road. The information is processed by layers of classification in the architecture of the network, learning these characteristics and the commands (steering, acceleration or others) as a class. In practice, the network associates the input images with which it was trained and returns the command, as seen in Fig. 1a.

However, this doesn't store the predictions of previous steps, which are necessary to make appropriate decisions to the situation. As shown in Fig. 1b, although the car is currently on a straight road, because I make a turn the speed is lower and it's necessary to correct the angle of the wheels.



(a) Scene that sees a reactive model.

(b) Scene that sees a model based on time.

Fig. 1: Difference between reactive and time-based models.

The objective of this document is to show the recurrent convolutional neuronal paradigm and its performance in comparison with a reactive convolutional neuronal paradigm shown in the literature, in the task of self-driving.

II. Background

A. Convolutional Neural Network

The convolutional neural networks (CNN) are based on the convolution of signals in two dimensions and the detection of filter-based characteristics (called kernel) that they learn through training. To extract the characteristics, the kernel performs the 2D convolution operation, expressed in (1), to the input image.

$$y(n_1,n_2) = \sum_{k_2=0}^{N-1}\sum_{k_1=0}^{M-1} x(k_1,k_2)\ h(n_1-k_1,n_2-k_2) \qquad (1)$$
$$0 \leq n_1 \leq N-1,\ 0 \leq n_2 \leq M-1$$

Where N and M are the dimensions of the image, n_1 and n_2 represent the column and row indices of the pixel being processed, k_1 and k_2 the kernel indexes.

This type of neural network has great advantages of automation in the task of extracting image information. In this paradigm, no known image processing filter is applied, but through the training of the neural network, it learns the image processing filters. In general, the first convolution layer learns to detect edges, while the second can learn to detect more complex shapes that can be formed by combining different edges, such as circles and rectangles. The third layer and beyond learn much more complicated characteristics based on the characteristics generated in the previous layer [2].

To train the convolutional layers, the backpropagation through the convolutional layer technique is used. This is very similar to backpropagation for a multilayer perceptron network, the only difference being that the weight connections are scattered, since the different input areas share the same weights to create a map of output characteristics. In general, the map of characteristics is obtained through the function expressed in (2). Where w represents the weights of the kernel and a first map of characteristics obtained.

$$S_{ij} = \sum_{n=1}^{2}\sum_{m=1}^{2} w_{(3-m)(3-n)} \cdot a_{(i-1+m)(j-1+m)} \tag{2}$$

For convolutional layers, the assigned activation function is the Rectified Linear Unit Function (ReLU) which is expressed in (3) where x represents the input value. This function allows normalizing the data by not admitting negative values and thus avoid noise that may contain the input image.

$$ReLU = ln(1 + e^{x}) \tag{3}$$

On the other hand, the kernel weights are adjusted by calculating the descending gradient which is expressed in (4). Where L represents the error function.

$$\frac{\partial L}{\partial w_{ij}} = \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{\partial L}{S_{ij}} \frac{\partial S_{ij}}{\partial w_{ij}} \tag{4}$$

B. Recurrent Neural Network

Recurrent Neural Networks (RNN) are designed to use and learn from sequential information. Generally, an RNN can be viewed as a weighted, directed, and cyclic graph that contains three different types of nodes, which are; the input, hidden and output nodes [3].

There are three types of essential tasks that can be performed with this type of network [4]:

1) Sequence recognition: A particular output pattern occurs when an input sequence is specified.
2) Sequence playback: The network must be able to generate the rest of a sequence when you see part of it.
3) Temporal association: In this case a particular output sequence must be produced in response to a specific input sequence.

The RNN suffer from an inherent problem: they don't capture the long distance dependencies between the inputs. This is because gradients in instances of long sequences have a high probability of going quickly to zero, called vanishing of the gradient, or tending to infinity called explosion of the gradient. A simple solution to the gradient explosion problem is to add a threshold that limits exponential growth. However, it isn't enough for the problem of gradient fading, so some variants of the classic RNN model have been proposed.

The Long Long-term Memory (LSTM) architecture was originally proposed by Hochreiter and Schmidhuber in 1997 [5], and is widely used today due to its superior performance in accurate modeling of short and long term dependencies.

An LSTM neuron is composed of 5 different non-linear components, which interact with each other in a particular way. In this paradigm, 4 components are called gates, they protect and control the information in the cell, which stores the memory of the neuron. The gates are implemented by a sigmoid function and a inner product. To control the behavior of each gate, one trains with a set of parameters and by calculating the gradient descending. The 5 gates of an LSTM cell are expressed in (5).

$$Forget\ gate : \sigma_f[t] = \sigma(\mathbf{W}_f x[t] + \mathbf{R}_f y[t-1] + b_f) \tag{5a}$$

$$Candidate\ state : \tilde{\mathbf{h}}_f[t] = g_1(\mathbf{W}_h x[t] + \mathbf{R}_h y[t-1] + b_h) \tag{5b}$$

$$Update\ gate : \sigma_u[t] = \sigma(\mathbf{W}_u x[t] + \mathbf{R}_u y[t-1] + b_u) \tag{5c}$$

$$Cell\ state : \mathbf{h}[t] = \sigma_u[t] \odot \tilde{\mathbf{h}}[t] + \sigma_f[t] \odot \mathbf{h}[h-1] \tag{5d}$$

$$Output\ gate : \sigma_o[t] = \sigma(\mathbf{W}_o \mathbf{x}[t] + \mathbf{R}_o y[t-1] + \mathbf{b}_o) \tag{5e}$$

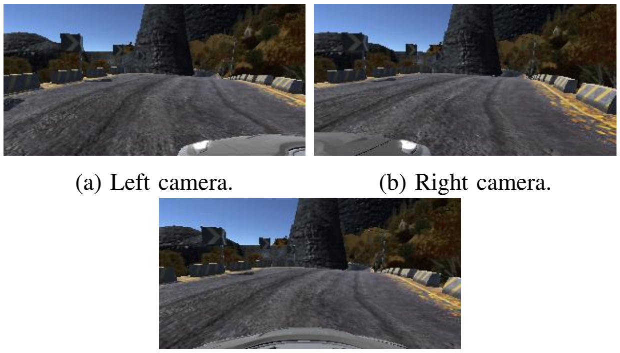
$$Output : \mathbf{y}[t] = \sigma_o[t] \odot g_2(\mathbf{h}[t]) \tag{5f}$$

Where $x[t]$ is the input vector at time t. W_f, W_h, W_u and W_o are rectangular weight matrices that are applied to the input of the LSTM cell. R_f, R_h, R_u and R_o are square matrices that define the weights of the recurrent connections, while b_f, b_h, b_u and b_o are vectors of bias. The function $\sigma(\cdot)$ is sigmoid, while $g_1(\cdot)$ and $g_2(\cdot)$ are point nonlinear activation functions generally implemented as hyperbolic tangents that reduce the values in $[-1, 1]$. Finally, $\odot$ represents the inner product [6].

III. System description

A. Simulation & Data

The platform used to experiment with this approach is the Udacity simulator. This simulator presents a great advantage by providing a training module, which consists of a manual driving in which it can be recorded through three simulated cameras located in the front of the vehicle, each image has a size of 320 x 160 pixels and 3 color channels. The images obtained from the simulator can be seen in Fig. 2, these three images represent a pattern of the driving database. The simulator captures 10 patterns per second, considering that the maximum speed of the vehicle is 30 mi/h, approximately each instance has a difference of 1 meter.


(a) Left camera.

(b) Right camera.

(c) Center camera.

Fig. 2: Images obtained from manual driving in the Udacity simulator.

The Udacity simulator provides a file in CSV format which contains the absolute path where that images are stored, as well as a vehicle steering command expressed in radians in a closed range $[-0.43, 0.43]$ (in the simulator is seen as $[-25°, 25°]$) which take the role of the class. Considering only 2 floating points, there's a difference between classes of $0.01\ r \Rightarrow 0.5729°$, obtaining a total of 87 classes and avoiding a drastic turn of the vehicle.

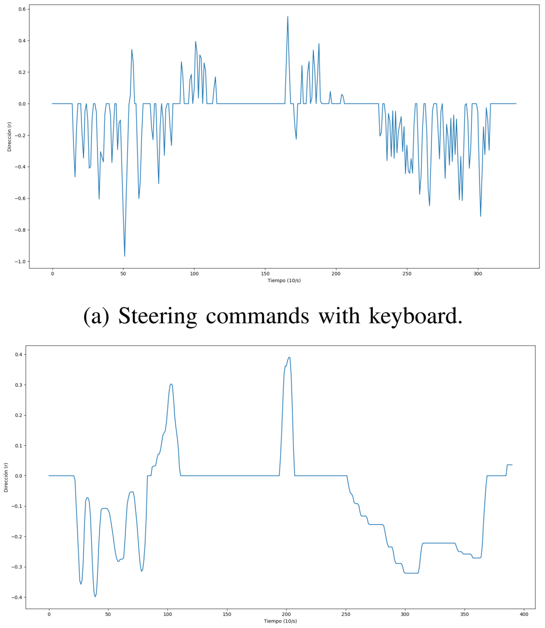
The manual driving is done through the control with the keyboard for the actions of turning, accelerating and braking. However the keyboard doesn't have the sensitivity to make slight turns, it's necessary to press the key several times in short periods to maintain the correct direction. These actions become noise in the class as shown in Fig. 3a. To solve this problem, a videogame steering wheel was used. Fig. 3b shows the steering commands obtained in the same fragment of the road driven with the steering wheel.

Now with respect to the evaluation of driving and because the developers of Udacity don't provide data on the length of the tracks, manual driving was performed at a constant speed of $30 mi/h$ (maximum speed), measuring the time to complete the The longitude result for the first track of $1.28\ km$ ($0.8\ mi$) and $2.99\ km$ ($1.86\ mi$) for the second was reached. This information is very important to evaluate the performance of models trained in the task of driving.

B. Preprocesing

The simulator only has two tracks: a simple one of approximately 2 miles and the other more complicated of double length. Therefore, if a model trained in these scenarios was tested in another environment with different lighting and road conditions, the network wouldn't give the expected results. Therefore, a preprocessing is carried out in two stages:

1) A clipping is made in the input image (which is 320 x 160 pixels) by removing 25 pixels in the lower part and 60 in the upper one, thus eliminating the front part of the vehicle that appears in the image and the sky that doesn't provide useful information, as shown in Fig. 4.



(a) Steering commands with keyboard.

(b) Steering commands with joystick.

Fig. 3: Vehicle steering commands (class).

The image is also scaled to a new one of 200 x 66 pixels, this to make the processing in the convolutional layers a little lighter and thus obtain a data vector of smaller size.

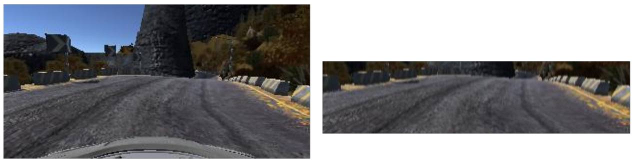


Fig. 4: Image clipping.

2) In this stage, the selection of one of the views shown in Fig. 2 is made. This serves as an information increase, since if the manual driving is carried out ideally maintaining a good position in the lane, cases in which the vehicle is near the edge of the road or invading the opposite lane wouldn't be memorized by the network and wouldn't respond adequately before this situation. Therefore a selection of one of the three views and their respective direction adjustment, expressed in (6), represents three different patterns.

$$f(r) = \begin{cases} r + 0.2 & \text{if } view = 0 \\ r - 0.2 & \text{if } view = 1 \\ r & other \end{cases} \tag{6}$$

In addition, a random rotation is performed on the X axis by adjusting the direction command given a random value x, $r = r + (x * 0.002)$. This visually represents a more closed or open curve as the case may be.

There may also be a rotation of the image generating a scene contrary to the existing one. The direction command is adjusted given $r = r{*}{-1.0}$, totally inverting the direction, as well as the original image.

C. Recurrent Convolutional Model

Because the design of an architecture requires extensive experimentation and extensive knowledge of the subject, the model proposed by [7] was used, which is used to decide turning angles in static images of roads with extreme climatic conditions. This model shown in Fig. 5 is made up of 5 convolutional layers, it doesn't have pooling layers. For the classification, 3 LSTM layers are used and finally 2 Multilayer Perceptron, the last one with a neuron to give the command as an output.

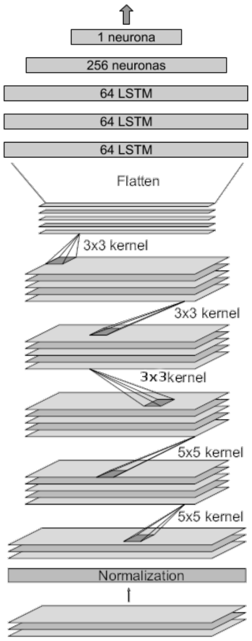


Fig. 5: Chauffeur model.

Similar to [8][9], the model depends on distributed time entries. This means that the network requires as input a short succession of chrono-evaluated images. As each image is represented as a 3D tensor, this model requires a 4D tensor related to a scalar, which is the class, in this case the turn command of the last image of the sequence. In Fig. 6 the required data structure is shown.

Therefore, each layer is subdivided into n layers that are governed by the same functions, where n is equivalent to the number of steps. Usually, the output of the last convolutional layer is a matrix, but in this approach the output is represented as a 3D tensor. However, the LSTM layers require a matrix as input, for this a layer of flatten is used. This layer reduces the dimensions of the tensioner as shown in Fig. 7.

IV. Experiments & results

In this experiment 2 databases were generated. The first contains a total of $97,186$ patterns, with a total of $291,558$ images obtained between the three views. Given that the simulator provides between 9 and 10 patterns per second, there's an approximate total of 2.7 hours of driving on the two tracks. The second database contains only data from the second track, which is longer and more complicated. This database contains $24,793$ patterns, $74,379$ images and represents 41 minutes of driving.

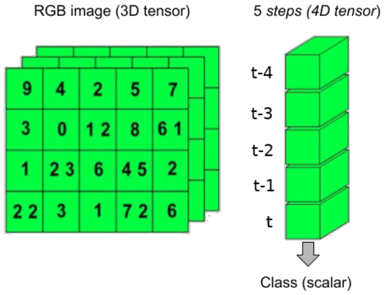


Fig. 6: Data structure required.

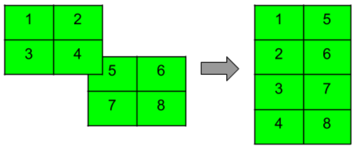


Fig. 7: Transformation made by the flatten layer.

To compare the results, we take the model proposed in [10][11] shown in Fig. 8. This model called Pilotnet is the largest reference in the self-driving domain and unlike the Chauffeur model (Fig. 5), it doesn't have convolutional layers of distributed time and its classification is based only on multilayer perceptron.

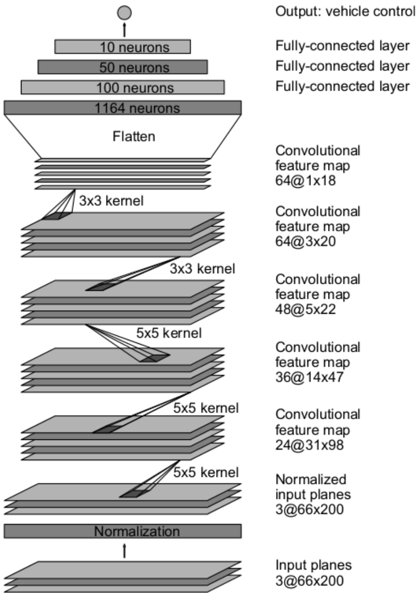


Fig. 8: Pilotnet model [10].

The tuning of parameters is the same for both models:

- Epochs: 10.
- Iterations: 10,000.
- Lot: 40 patterns.
- Learning rate: 1.0^{-5}.
- Training set: 80%.
- Validation set: 20%.

To compare the results with the Chauffeur model with different steps, I trained with 3 and 5 steps using the same parameter configuration.

In the study of the state of the art, few metrics related to the evaluation of autonomy were found. In [10], the metric shown in (7) is proposed, which is the most referenced in the literature.

$$autonomy = (1 - \frac{interventions \cdot 6}{elapsed\ time}) \cdot 100 \tag{7}$$

The metric is based mainly on the corrections made by the driver to avoid collision of the vehicle. This number of interventions is multiplied by a constant of 6 seconds, which represents the average time an average driver takes to correct the course of the vehicle, according to a study carried out by them.

Throughout the tests carried out in this work it was observed thatn't necessarily an intervention is usually 6 seconds in the simulation, even though it's in real time. Due to these variations, the metric proposed in the literature turns out to be less reliable, therefore it's proposed to measure the autonomy based on the intervention time of the driver to correct the address using the expression 8. This proposed metric evaluates the autonomy similarly to the previous one, with the difference that it takes an absolute time of interventions.

$$autonomy = 100 - \frac{total\ intervention\ time}{elapsed\ time} \cdot 100 \tag{8}$$

As a third autonomy metric, it's proposed to perform the test without the driver's intervention to measure the percentage of the route that completes the vehicle on the runway until the first collision using the expression (9). In this way there's a reference on what each model He has learned from the road.

$$completed\ road = \frac{s \cdot t}{d} \cdot 100 \tag{9}$$

Where s is the average speed of the vehicle, t represents the driving time until the first collision or completing the route and d is the total distance of the test track.

First, the test was made allowing the driver's interventions. To activate the manual driving, press the ”x” button on the steering wheel, this deactivates the neuronal model commands and initializes the counting of the intervention times. Similarly, by pressing the ”○” button, print the values of the calculated metrics until the moment of the button event. This test was carried out with the two models, the configurations of 3 and 5 steps of the Chauffeur model and with both databases, the results obtained are shown in Table I.

TABLE I: Metrics of autonomy obtained.

Model	**Metrics**			
	NVIDIA		Proporsal	
	Road 1	Road 2	Road 1	Road 2
Pilotnet (shortest db)	44.55%	83.11%	82.09%	96.35%
Pilotnet (largest db)	**100%**	77.57%	**100%**	90.05%
Chauffeur 3 steps (shortest db)	76.88%	85.96%	90.26%	92.48%
Chauffeur 3 steps (largest db)	**100%**	**100%**	**100%**	**100%**
Chauffeur 5 steps (shortest db)	88.89%	88.78%	95.43%	96.23%
Chauffeur 5 steps (largest db)	**100%**	**100%**	**100%**	**100%**

In a second stage, each model is tested without intervention. When starting the simulation, a chronometer is initialized and it's finalized when the vehicle stops. Every second the speed is sampled to obtain an average speed, this to calculate the percentage of the completed road. The results of this experiment are reported in Table II.

TABLE II: Metrics of road completed obtained.

Model	**Road completed**	
	Road 1	Road 2
Pilotnet (shortest db)	9.47%	7.52%
Pilotnet (largest db)	**100%**	30.44%
Chauffeur 3 steps (shortest db)	33.61%	18.07%
Chauffeur 3 steps (largest db)	**100%**	**100%**
Chauffeur 5 steps (shortest db)	34.37%	16.93%
Chauffeur 5 steps (largest db)	**100%**	**100%**

Remembering that you have 87 classes, a difference of 0.01 in the prediction represents an absolute error in the metrics that are usually used for the evaluation of neural networks, such as precision, accuracy, etc. However, in the tests, a minimal error doesn't have such a drastic effect on driving, that is why this type of evaluation doesn't turn out to be ideal.

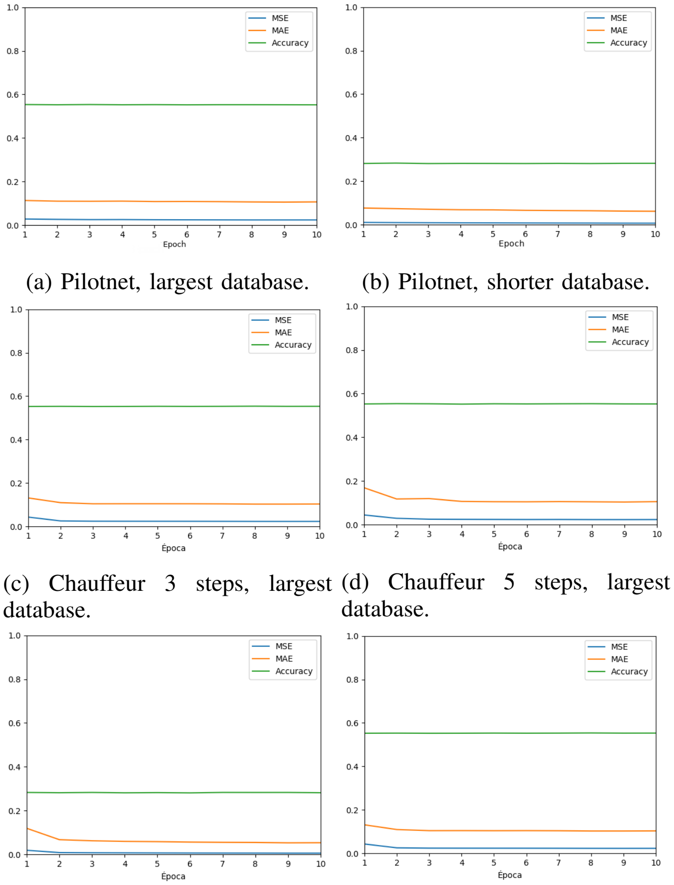
On the other hand, the metrics dedicated to measure the error in the prediction adapt much better to the problem and show better the performance of these networks, since they calculate the difference between the expected and the obtained. A small angle of difference doesn't present as much error, that is why these are the ones that were contemplated to evaluate the training of the networks. The metrics used are: Mean Squared Error (10), Mean Absolute Error (11) and Root Mean Squared Error (12).

$$MSE = \frac{1}{n}\sum_{i=1}^{n} = (\hat{Y}_i - Y_i)^2 \tag{10}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} = |\hat{Y}_i - Y_i|^2 \tag{11}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} = (\hat{Y}_i - Y_i)^2} \tag{12}$$

Performing the analysis of the results obtained, the training with the shortest database obtains better results in the error metrics as seen in Fig. 9b, 9e & 9f, but in practice and according to the Tables I & II, the training with the base of largest data are the ones that drive the car better. It's also observed that the Accuracy Metric presents very variable results and it doesn't stop to have relation with the rest. Another observation is that the RMSE (12) isn't included in the graph since it gives the same results as the MSE (10) in all cases.



(a) Pilotnet, largest database.

(b) Pilotnet, shorter database.

(c) Chauffeur 3 steps, largest database.

(d) Chauffeur 5 steps, largest database.

(e) Chauffeur 3 steps, shorter database.

(f) Chauffeur 5 steps, shorter database.

Fig. 9: Training metrics obtained.

In the qualitative results, the Chauffeur model shows superiority to the reactive model. Although the experiments done with the 3 and 5 step configurations with the largest database obtain the same results, the driving behavior of the 3-step model is very erratic (as if the driver was in a drunken state). This driving doesn't respect the lanes and although it doesn't collide, it drives near the limits of the road and occasionally moves in zigzag. These behaviors are corrected in the 5-step model, which behaves similarly to a human driver. However, in training with the shortest database it's more sensitive to unknown situations, which doesn't happen with the previous one.

V. Conclusion

As a conclusion, a reactive model isn't enough to completely solve a complex problem such as driving. It was shown that a model created to label static images can be adapted to process small video fragments and predict subsequent actions. Although this driving is done in environments without obstacles, it can be used on free roads.

The observation of the erratic behavior of the 3-step model motivated a study of the literature with the intention of finding metrics dedicated to the evaluation of driving behavior. In this study we didn't find any method to evaluate this behavior, so as a future work, the design and experimentation of a metric to evaluate the autonomy as a driving behavior will be carried out.

References

[1] T. Litman, *Autonomous vehicle implementation predictions*. Victoria Transport Policy Institute Victoria, Canada, 2017.

[2] S. Pattanayak, Pattanayak, and S. John, *Pro Deep Learning with TensorFlow*. Springer, 2017.

[3] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio, "Architectural complexity measures of recurrent neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1822–1830.

[4] I. Bonet Cruz, S. Salazar Martínez, A. Rodríguez Abed, R. Grau Ábalo, and M. M. García Lorenzo, "Redes neuronales recurrentes para el análisis de secuencias," *Revista Cubana de Ciencias Informáticas*, vol. 1, no. 4, 2007.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. Springer, 2017.

[7] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ACM, 2018, pp. 303–314.

[8] A. Hassan and A. Mahmood, "Convolutional recurrent deep learning model for sentence classification," *IEEE Access*, vol. 6, pp. 13 949–13 957, 2018.

[9] P. H. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *31st International Conference on Machine Learning (ICML)*, 2014.

[10] 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.

[11] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*, 2017.