2020 17th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)
Mexico City, Mexico. November 11-13, 2020

Self-driving through a Time-distributed Convolutional Recurrent Neural Network

Antonio Luna-Álvarez
Tecnológico Nacional de México-CENIDET
Cuernavaca-Morelos, México
jesus.luna18ce@cenidet.edu.mx

Dante Mújica-Vargas
Tecnológico Nacional de México-CENIDET
Cuernavaca-Morelos, México
dante.mv@cenidet.tecnm.mx

Manuel Matuz-Cruz
Tecnológico Nacional de México-Tapachula
Tapachula-Chiapas, Mexico

Jean Marie Vianney Kinani
Instituto Politécnico Nacional-UPIIH
San Agustín Tlaxiaca-Hidalgo, Mexico

Eduardo Ramos-Díaz
Instituto Politécnico Nacional-UPIIH
San Agustín Tlaxiaca-Hidalgo, Mexico

***Abstract*—This paper proposes an approach based on the use of time dimension within a Convolutional and Recurrent Hybrid Neural Network to carry out the driving of a simulated vehicle in real time. Convolutional layers are transformed into time-distributed layers that process a series of images in parallel and are related in time and space by recurrent layers. The experimentation showed an approximate 10% improvement over other autonomous driving approaches.**

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

I. Introduction

Autonomous vehicles are robotic systems that must have the ability to navigate the environment for which they were designed, avoiding obstacles or situations that compromise the safety of the vehicle and its passengers [1]. The main component of a self-driving car is the decision module, which can be controlled by a classification model that predicts the steering angle and vehicle speed [2]. In the study of the state-of-the-art, it has been observed that the most reliable classification models are given by a Neural Network Based on Deep Learning. Although good self-driving results have been documented using Deep Learning, as documented in the articles [3]–[5] with reactive models, the importance of time in the driving task has been forgotten. A person with enough experience driving in urban areas foresees the following actions in advance, for example stopping distances. In the same way, it makes decisions taking into account the actions carried out in the immediate past.

To perform a human-like conduction, it is necessary to add the time factor to the Neural Network. For this purpose, this paper proposes the use of the time-distributed convolutional layers approach [6], which process time series of t frames in parallel in the fourth dimension, subsequently relating the time steps with space through recurrent layers to control the vehicle direction.

The rest of the document is organized as follows: in Section II the concepts necessary to understand the proposed approach are explained, in Section III the base and improved models are detailed, in IV the data, evaluation metrics and results obtained in the experimentation are detailed, to finalize with the conclusions in Section V.

II. Background

A. Convolutional Neural Network

Convolutional Neural Networks (CNN) base their operation on the use of adjustable filters called kernel, these serve as parameters w called synaptic weights in other architectures such as the multilayer perceptron. A filter w convolves the inputs of the tensor $x \in \mathbb{R}^{2+}$ in such a way that it reduces their size and extracts new high-abstraction features. For an input 2D image operate through:

$$h(n_1, n_2) = \sum_{w_2=1}^{N} \sum_{w_1=1}^{M} x(w_1, w_2)\ h_k(n_1 - w_1, n_2 - w_2) \quad (1)$$

Where N and M are the dimensions of the convolution kernel that satisfies $0 \leq w_1 \leq N-1$, $0 \leq w_2 \leq M-1$. n_1 and n_2 represent the column and row indices of the pixel being processed. At this point, the input x^2 has been transformed into a layer h of k feature maps defined from the number of filters used.

In this paradigm, through training, the neural network learns the image processing filters. The first convolution layer learns to detect edges, the second can detect shapes like circles and rectangles. Third layer and beyond learn much more complicated based on the generated in the previous [7]. The assigned activation function is the Rectified Linear Unit (ReLU) $S_{ij} = \max(0, h_{ij})$. This Function allows to normalize the data by not admitting negative values and thus avoiding noise. In learning, the backpropagation is spread out between the kernel weights w and adjusted for i epochs by the Gradient Descent method expressed in (2), where L represents the loss function and α represents the learning rate.

$$w_{i+1} := w_i - \alpha \frac{\partial L}{\partial w_i} \quad (2)$$

978-1-7281-8987-1/20/$31.00 2020 IEEE

B. Recurrent Neural Network

Recurrent Neural Networks (RNN) are designed to learn from sequential information. They are interpreted as weighted, directed and cyclic graphs with three types of essential nodes: input, hidden and output [8]. Generally are used for three tasks [9]: sequence recognition, sequence playback and temporal association. Simple RNNs cannot capture distant dependencies because the gradients obtained from long sequences tend rapidly to zero or to infinity, which is called gradient vanishing and explosion. The Long Short-Term Memory (LSTM) [10] provides a solution thanks to its composition of five non-linear components, which interact with each other in a particular way and are distributed in t time instances. It contains four components called gates that control the information within the neuron, operating it through the inner product and a non-linear activation function such as sigmoid. The gates fulfill a specific function and operate by:

$$Forget : \sigma_f[t] = \sigma(\mathbf{W}_f x[t] + \mathbf{R}_f y[t-1] + b_f) \quad (3)$$

$$Candidate : \tilde{\mathbf{h}}_f[t] = g_1(\mathbf{W}_h x[t] + \mathbf{R}_h y[t-1] + b_h) \quad (4)$$

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

$$Output : \sigma_o[t] = \sigma(\mathbf{W}_o \mathbf{x}[t] + \mathbf{R}_o y[t-1] + \mathbf{b}_o) \quad (6)$$

While the hidden state is abstracted into the cell state (7) and the neuron output is obtained by (8).

$$Cell\ state : \mathbf{h}[t] = \sigma_u[t] \odot \tilde{\mathbf{h}}[t] + \sigma_f[t] \odot \mathbf{h}[h-1] \quad (7)$$

$$Output : \mathbf{y}[t] = \sigma_o[t] \odot g_2(\mathbf{h}[t]) \quad (8)$$

Where $x[t]$ is the input vector at time step 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 non-linear activation functions such as hyperbolic tangents that reduce the values in $[-1, 1]$. Finally, $\odot$ represents the inner product [11]. To control the behavior of each gate, one trains with a set of parameters and by calculating the gradient descent.

III. Proposed approach

A. Base model

Since the design of a neural network architecture requires extensive experimentation and knowledge in the application domain, the Chauffeur model proposed by [12] was taken as a basis. Originally, this model was used to predict steering angles in static images on roads with extreme weather conditions, such as fog and heavy rain. The model is a hybrid of five CNNs for extraction of features maps, three LSTM layers that process and memorize the training data patterns, and two Multilayer Perceptrons (MLP) for classification. The complete model is shown in Figure 1.

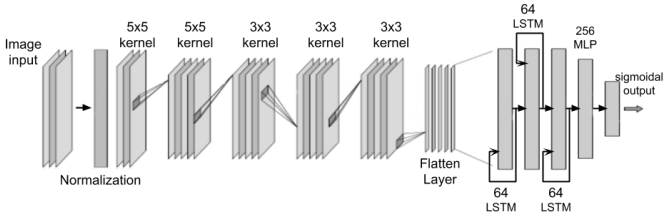


Fig. 1: Chauffeur model [12].

The model receives an input X with dimension $w \times h \times 3$ according to the dimensions of the images in the database and the RGB channels. The first layer $C1$ generates k feature maps obtained from the convolution of a filter w^k of size $m \times n \times d$ for each pixel and color channel in X plus a bias b^k. Applying w filters of size 5×5 in the first layer generates $k = 24$ feature maps of size $w - \lfloor 5/2 \rfloor \times 2 \times h - \lfloor 5/2 \rfloor \times 2$, for each index i, j in the feature map k:

$$C1^k_{i,j} = f\left(\sum_{m=1}^{w_w}\sum_{n=1}^{w_h}\sum_{d=1}^{3} X_{m+i,j+n,d} \times w^k_{m,n,d} + b^k\right) \quad (9)$$

where the function $f(\cdot)$ represents the activation function ReLU. Consecutively $C2$ produces 32 maps of $w_{C2} = (w_{C1} - 5)/2 + 1 \times h_{C2} = (h_{C1} - 5)/2 + 1$ from the 24 generated from $C1$. Following this method $C3$ extracts $k = 48$ maps with filters w of 3×3, $C4$ $k = 64$ with filters of 3×3, and $C5$ $k = 128$ maps with 3×3 filters. For the extracted information to be memorized by the LSTM layers, it is necessary to perform a dimension reduction. The feature maps generated in $C5$ are compacted in the F layer of *Flatten*, this transforms $F : C5^{m \times n \times d} \mapsto C5^{(m \times n \times d)}$. So that in F a vector of $m \times n \times d$ values is extracted. These serve as input for the $R1$ layer, which consists of 64 LSTM neurons and propagates in the same way as (3) to (8).

The output of the third LSTM $R3$ layer is passed to the MLP layers, the first one receives a vector of 64 values in 256 neurons activated by the sigmoid function, the output of this layer is 256 values that are sent to the last layer with a single neuron that normalizes to an output at $[-1, 1]$ by means of the Hyperbolic Tangent function:

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (10)$$

In this way, the steering control is taken with -1 a full turn to left, 0 a central position and 1 to the right. However, so that this model can operate in a continuous simulation it is necessary to add the time dimension, this is done by converting the 3D convolutional layers into 4D time-distributed convolution layers.

B. Time-distributed Convolutional Recurrent Model

Similar to [13], [14], the model depends on distributed time inputs. 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 steering command of the last image of the sequence. Figure 2 shows the required data structure.

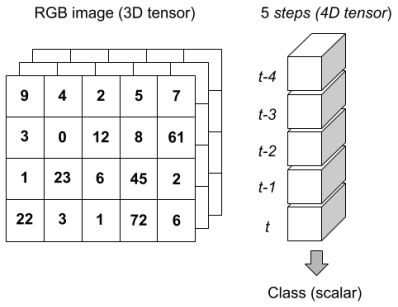


Fig. 2: Data structure required.

In Figure 3 the model with the proposed approach is shown as input x a 4D tensor similar to the one shown in Figure 2 that is normalized and distributed through the time dimension.

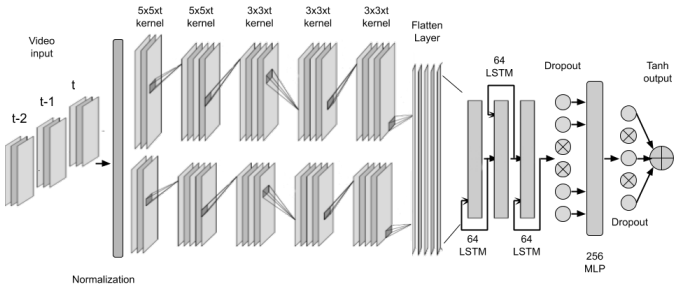


Fig. 3: Proposed time-distributed Chauffeur model.

The most notable difference is the expansion of the convolutional layers that appear to be a parallel architecture. The convolution (9) adds a superindex t indicating the time step to process, the operation is rewritten for $C1$ as:

$$C1_{i,j}^{t,k} = f\left(\sum_{m=1}^{w_w}\sum_{n=1}^{w_h}\sum_{d=1}^{3} X_{m+i,j+n,d} \times w_{m,n,d}^{t,k} + b^{t,k}\right) \tag{11}$$

Starting with layer $C2$, each map is reduced in size for all convolutions using the subsampling method, which replaces the pooling layers. The jump factors dW and dH are added to the convolution filter $m \times n$, in this way a reduced characteristics map is obtained whose dimension is given by:

$$w_{C2} = \frac{w_{C1} - m}{dW} + 1 \tag{12}$$

$$h_{C2} = \frac{h_{C1} - n}{dH} + 1 \tag{13}$$

Applying $k = 32$ convolution filters w of 5×5 and subsampling $dW = 2 \times dH = 2$, the layer $C2$ is expressed as:

$$C2_{i,j}^{t,k} = f\left(\sum_{m=1}^{5}\sum_{n=1}^{5}\sum_{d=0}^{32} C1_{dW*i+m,dH*j+n,d}^{t,k} \times w_{m,n,d}^{t,k} + b^{t,k}\right) \tag{14}$$

The feature maps generated in $C5$ are compacted in the Flatten layer F, this transforms $F : C5^{d\times m\times n\times t} \mapsto C5^{(d\times m\times n)\times t}$ from:

$$F_i^t = \sum_{d=0}^{128}\sum_{m=0}^{38}\sum_{n=0}^{36} C5_{d,m,n}^{t} \tag{15}$$

So in F a t vectors of $175,104$ values are extracted. These serve as input for the layer $R1$ which is now time-distributed, this means that the information in $t-1$ adjusts $\mathbf{R}$ and is at the same time auto-adjust the synaptic weights w in time t. In this part a dropout layer is added, which under a random parameter p scales the information obtained in the previous layers, having the probability of reducing the value of some neuron to 0. $D1$ consists of a total of 256 neurons:

$$\begin{aligned} \mathcal{D}_i &= \mathcal{R}3 \times p_i \\ 0.2 &\leq p_i < 1.0 \end{aligned} \tag{16}$$

The next layer is made up of 256 MLP neurons with sigmoid activation function $\sigma(\cdot)$. Then a Dropout layer appears again at $D2$ which passes the information to the output layer. The output layer consists of a single MLP neuron with a Hyperbolic Tangent activation function.

C. General setup

For training parameters setup of the neural model, the recommendations mentioned by [15] were taken into account, considering the number of instances in the database detailed in Section IV-A, the size network and the hardware used. In addition, the stochastic descending gradient method for backpropagation is replaced by a more sophisticated method that guarantees to accelerate learning. The parameters are:

- Time steps: 5.
- Epochs: 10.
- Iterations: 10,000.
- Batch: 40 patterns.
- Learning rate: 1.0^{-5}.
- Training set: 80%.
- Validation set: 20%.
- Optimizer: Stochastic variant of the ADAM [16] optimizer called Adaptive Noise Moment [17]

IV. Experiments

A. Simulation and data

For the driving data acquisition, the Udacity simulator [18] was used, this provides a training module which consists of a manual driving where it is recorded by means of three parallel cameras at a rate of 10 fps with size 320×160 in 8-bit RGB format. Considering the frame acquisition rate and that the maximum speed of the vehicle is $30\frac{mi}{h} \Rightarrow 13.41\frac{m}{s}$,

the difference between each instance captured during the movement is $\approx 1.3\ m$. In Figure 4 an instance captured by the cameras is shown.

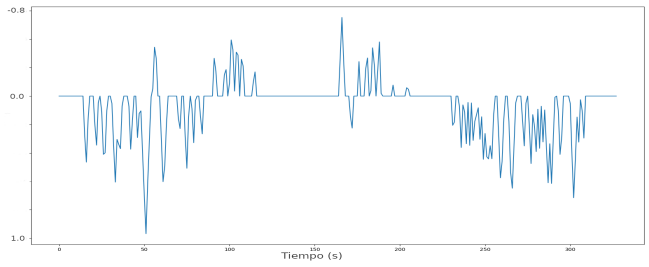
(a) Left view.

(b) Right view.

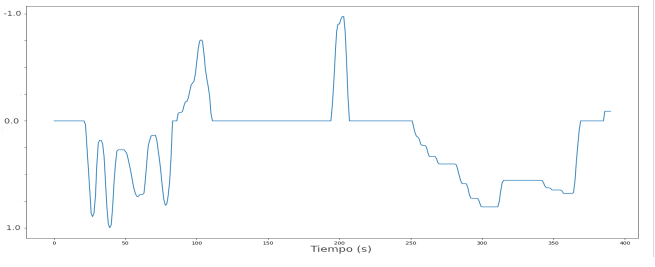
(c) Center view.

Fig. 4: Images obtained from manual driving in the Udacity simulator.

The Udacity simulator provides a CSV format file which contains the absolute path where the images are stored, as well as a vehicle steering command on $[-0.43, 0.43]$ in radians which represents the class. This class is normalized to $[-1, 1]$ to be replicated by the non-linear Hyperbolic Tangent function. Control is via keyboard for steering, acceleration and braking actions. However the keyboard does not have the sensitivity to make slight turns, it is 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 Figure 5a. To solve this problem, a steering wheel joystick was used. Figure 5b shows the steering commands obtained in the same fragment of the road driven with the steering wheel.



(a) Steering commands from keyboard.



(b) Steering commands from joystick.

Fig. 5: Vehicle steering commands (class).

Since Udacity developers do not provide information about the length of their two roads, constant speed driving of $30\frac{mi}{h}$ was performed and measuring the time to complete one lap was measured, the resulting length for first road is $1.28\ km$ and $2.99\ km$ for the second. This information is very important to evaluate the performance of models trained for autonomous driving. Based on these characteristics, a database was created with approximately 2.7 hours of driving on the roads that represent a total of $291,558$ images with different views of the road.

B. Preprocesing

The simulator only has two roads, so if a model trained in these scenarios was tested in another environment with different lighting and road conditions, the Neural Network would not give the expected results. Therefore, a preprocessing is carried out in three stages:

1) A clipping is made in the input image by removing 25 pixels in the bottom and 60 in the top, thus eliminating the front part of the vehicle that appears in the image and the sky that does not provide useful information. The image is also scaled to 200×66 pixels, this to make the processing on the convolutional layers a little lighter.
2) In this stage, the selection of one of the views shown in Figure 4 is made. This serves as information augmentation, since it is necessary to have instances where the vehicle is close to the edge of the road or encroaching on the opposite lane. Therefore a selection of one of the three views and its respective direction setting, expressed in (17), represents three different training patterns.

$$f(r) = \begin{cases} r + 0.2 & \text{if } view = 0 \\ r - 0.2 & \text{if } view = 1 \\ r & other \end{cases} \tag{17}$$

3) A random rotation is performed on the X axis by setting the direction command given a random value x, $r = r + (x \times 2^{-3})$. 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 adjusts given $r = r \times -1.0$, totally reversing the direction.

C. Metrics

In the state-of-the-art, some metrics related to the evaluation of autonomy were found. In [19], the metric shown in (18) is proposed, which is the most referenced in the literature.

$$autonomy = \left(1 - \frac{interventions \cdot 6}{elapsed\ time}\right) 100 \tag{18}$$

This metric is based on the interventions made by the driver to avoid collisions and scaled at a constant of 6 seconds, which represents the average time to stabilize the vehicle. Through experimentation, it was observed that the interventions are faster than 6 seconds, so starting from (18) the metric (19)

is proposed, which measures the absolute time of intervention giving a more accurate estimate.

$$self - driving = \left(1 - \frac{total\ intervention\ time}{elapsed\ time}\right) 100 \tag{19}$$

As a third autonomy metric, it is proposed to perform the test without driver interventions to measure the percentage of road that the vehicle completes on the road until the first collision using the expression (20). In this way, reference is made to what each model has learned from the road.

$$completed\ road = \left(\frac{s * t}{d}\right) 100 \tag{20}$$

Where s is the average speed, t the driving time to the first collision or completing the road, and d is the total distance of the test road. First, the test was carried out allowing the driver to intervene.

To compare the results, the NVIDIA Pilotnet model proposed in [19], [20] is taken. This model does not have time-distributed or recurrent layers. The training parameters are the same, with the difference that it was trained with the ADAM optimizer [16]. For further experimentation, the Chauffeur base model was trained with 1 time step and same parameters, such that it receives an input image as Pilotnet. As a fourth comparison method, Cirl [21], a vision-based reinforcement learning-trained controller, was used.

D. Results

Each model was trained with the proposed database and its configuration mentioned above, for the Cirl method the pre-trained model was used. The experimentation consisted of 10 autonomous driving tests on the two roads with driver intervention and another 10 without it. The Control is done by telemetry using the Python 3.5 language with Tensorflow and Keras frameworks, where the metrics are also captured. The training, as well as the telemetry control was executed on a PC with Intel Core i7-7700 processor with 32 GB RAM and 1 TB SDD, it is also boosted by a GTX Titan X graphic processing unit with 3072 CUDA cores and 12 GB of dedicated memory. For the intervention experimentation, it presents the mean of the results obtained in Table I.

TABLE I: Autonomy results obtained.

Model	**Metrics**			
	Autonomy		Self-driving %	
	Road 1	Road 2	Road 1	Road 2
Pilotnet	*83.11%*	74.55%	*86.35%*	82.09%
Cirl	84.26%	*66.81%*	91.61%	*73.01%*
Base Chauffeur	85.96%	76.88%	92.48%	90.26%
Enhanced Chauffeur	**90.89%**	**88.78%**	**96.43%**	**96.23%**

As can be seen in Table I, the proposed approach considerably improves autonomy results compared to reactive models and reinforcement learning. The reactive Pilotnet model presents the lowest performance for road 1, which has the least complexity, this happens because it lacks the ability to learn time sequences making difficult actions of greater complexity such as sharp curves and drastic lighting changes. On the other hand, Cirl learning reinforcement is presented as the least suitable for road 2, which contains tighter curves and more complex variations because the circumstances in which he was trained did not contain these characteristics, perhaps a training on this road improves performance. results although it would be heavier and more time consuming. The results also show that adding time-distributed convolutional layers improves the autonomy results when considering short time periods; This is observable in the comparison of the base model and the proposed one. The autonomy results contrast by 6.68% compared to the base model.

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

TABLE II: Percentage of road completed.

Model	**Road completed**	
	Road 1	Road 2
Pilotnet	51.32%	30.44%
Cirl	64.49%	*11.27%*
Base Chauffeur	*33.61%*	18.07%
Enhanced Chauffeur	**100%**	**97.93%**

The experimentation shows that the proposed approach presents the best performance, complete road 1 without interventions in all experiments and road 2 had a single experiment completed only 79.31%. It is worth mentioning that for the first series of tests, the interventions were carried out from when the vehicle presented risk situations, in this second experimentation it was allowed to continue in a risk situation and no collisions occurred. On the other hand, the Pilotnet model traveled the shortest distance on both roads, while Cirl had the greatest difficulty on road 2. The base Chauffeur model requires adjustments from the beginning as it takes very rigid directions as it lacks time dependence.

V. Conclusion

In the experimentation, it was observed that the proposed approach improves on average 10.8% the autonomy metrics, this due to the consideration of the passage of time in the driving task. An observation of this experimentation is that the configuration of 3 time steps of the model presents certain unsafe behaviors, however when increasing to 5 time steps these behaviors are reduced and a more human-like driving is

appreciated, so as future work plans to experiment with the variation in the dimension of time, as well as to include road obstacles and various vehicle sensors such as LiDAR scanners.

Acknowledgments

The authors thank to CONACYT, as well as Tecnológico Nacional de México/Centro Nacional de Investigación y Desarrollo Tecnológico for their financial support through the project so-called "Controlador Difuso para ajuste de coeficientes de rigidez de un modelo deformable para simulación en tiempo real de los tejidos del hígado humano".

References

[1] C. B.S. T. Molina, J. R. de Almeida, L. F. Vismari, R. I. R. González, J. K. Naufal, and J. B. Camargo, "Assuring fully autonomous vehicles safety by design: The autonomous vehicle control (avc) module strategy," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, IEEE, 2017, pp. 16–21.

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

[3] M. G. Bechtel, E. McEllhiney, and H. Yun, "Deeppicar: A low-cost deep neural network-based autonomous car," *arXiv preprint arXiv:1712.08644*, 2017.

[4] S. Chen, S. Zhang, J. Shang, B. Chen, and N. Zheng, "Brain-inspired cognitive model with attention for self-driving cars," *IEEE Transactions on Cognitive and Developmental Systems*, 2017.

[5] V. Alexeev, A. Staravoitau, G. Piskun, and D. Likhacheuski, "End to end learning for a driving simulator," *foreignlanguage russian Reports of the Belarusian State University of Informatics and Radioelectronics*, no. 2 (112), 2018.

[6] A. Koeppe, F. Bamer, and B. Markert, "An intelligent nonlinear meta element for elastoplastic continua: Deep learning using a new time-distributed residual u-net architecture," *Computer Methods in Applied Mechanics and Engineering*, vol. 366, p. 113 088, 2020.

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

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

[15] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

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

[17] D. Mújica-Vargas, A. Luna-Álvarez, J. de Jesús Rubio, and B. Carvajal-Gámez, "Noise gradient strategy for an enhanced hybrid convolutional-recurrent deep network to control a self-driving vehicle," *Applied Soft Computing*, p. 106 258, 2020.

[18] A Udacity, *Self-driving car simulator built with unity*, 2018.

[19] 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.

[20] 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.

[21] X. Liang, T. Wang, L. Yang, and E. Xing, "Cirl: Controllable imitative reinforcement learning for vision-based self-driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 584–599.