End-to-End Time Distributed Convolution Neural Network Model for Self Driving Car in Moderate Dense Environment

Willy Dharmawan
Graduate of Electrical Engineering and Computer Science
Kanazawa University
Kakuma, Kanazawa, Japan
willy.dharmawan@bppt.go.id

Hidetaka Nambo
Graduate of Electrical Engineering and Computer Science
Kanazawa University
Kakuma, Kanazawa, Japan
nambo@blitz.ec.t.kanazawa-u.ac.jp

***Abstract*—** Vehicle control in Autonomous Car requires the following command to make sure that the car can accomplish a specific task, such as taking a turn, stop on the traffic light, following lanes, and changing lanes. This serial command indicates that a self-driving car should not be addressed as a context-based problem that theoretically needs a temporal system that can accommodate multiple frames.

Based on this added complexity of the problem, we propose a network that can accommodate the sequential input of images. Thus, we apply a time distributed model of Convolutional Neural Network (CNN), to recognize a visual problem, followed by LSTM that can capture temporal state dependencies.

By modifying the Carla environment, we can capture frame per frame images with detailed information of throttling, speed, steering angle, brake, and some states such as direction, speed limit, and traffic light state. We use the Carla control agent so that it can automatically capture all of the images from the camera and those of information. We demonstrate that this rough approach can perform well in the Carla environment with moderate dense traffic. It can reach the destination faster than the ground truth and standard convolution model in just 93.978 seconds. Although the driver agent performance is a bit rough with around 13.27 of speed above score, it shows a better steering control, which means better stability.

Keywords*—*Time Distributed, LSTM, CNN, Carla, Autonomous Car

I. Introduction

Autonomous car system, which split up into spatial perception and control module, has become a part of the development of vehicle control rule-based solution [1], [2], [3], [4], [5], as well as a traditional solution [6]. Then, in the recent year, Nvidia proposed a newer solution [7], applying a high dimensional feature extraction using Convolutional Neural Network (CNN) and map the result into steering control.

This efficacy of learning action policies from mapping pixel images is appealing because it directly mimics the demonstrated performance, without accounting to the agent environment. Even though it can learn vehicle control from the input of images, we think that self-driving car does not just rely on single information of images. It is a sequence problem that requires multiple data of images. It shows in [8] that a sequential based model has slightly better accuracy in comparison to just CNN based model. This result shows in the implementation of Fully Connected Network-Long Short-Term Memory (FCN-LSTM) architecture on discrete action-driving experimentation. Therefore, in this work, we explore a time distributed based model which combines CNN and LSTM, accommodating multiple input and multiple outputs.

The differences of time distributed model with stacked CNN are the LSTM layer part and sequential input in time steps. The agent will learn a control parameter from the subsequent information. The model extracts the high dimension features, and LSTM will determine the importance of each feature. From this process, the model will yield the control parameter. This methodology fits with the autonomous car problem because it is a sequential problem that needs multiple images to know the best policy.

Many factors in the self-driving car environment cannot be reflected solely through a single front-view camera sensor. So that, we address this problem into a discrete state such as go left, right, go straight, and lane-follow and define these into three parameters output, steering angle, throttle, and brake. Also, the Carla environment provides some parameters, such as *direction state*, *speed*, *speed limit*, and *traffic light state* in the lane. Thereby, we can just simply make some adjustments on the code and create a program to acquire our training data and test our model. In contrast, it requires another algorithm or mechanism to get those specific states of the agent.

There is also a problem regarding on how to set a temporal model on time steps. If we try to use a standard dense layer sequentially, the weights and biases might be changed, and it makes the output flattened with each time step mixed. Thus, we use the time distributed layer to wrap the model, to get output separately by time steps.

We evaluate this proposed model by comparing it to a spatial based model and ground truth data on how it differs in test performances in the town environment. The experiment results show that time Distributed model has better stability in steering angle in comparison to just a regular stacked CNN model. However, this combination of CNN-FCN-LSTM does not perform well in controlling speed, because it has the highest score in *above speed control*. Consequently, it can reach the predefined destination faster than the ground truth and CNN based model.

II. End-To-End Self Driving Car

End-To-End Learning has been known as a straight forward way and more into brute force technique for solving a complex problem by taking advantage of deep learning structure [9]. It comprises of several layers that constitute an artificial neural network, which imitates a distributed approach to solve a problem.

The autonomous car is one of the remarkable examples of a complicated problem. To define this particular system, Alexandru Serban et al. create a multi-layers diagram [10]. It starts with the extraction of sensor fusion data to get relevant features (e.g., object detection), then the car will have information on the surrounding environment (*world model*). From this onward, the system will generate a behavior model that is useful for decision making in the planning layer. Finally, the control layer interfaces the control through the actuator.

The end-to-end learning model simplifies these multilevel problems. The first attempt in 1989, Autonomous Land Vehicle in a Neural Network (ALVINN) [11], manages to do well in simple roads with few obstacles. It just comprised of a shallow network that predicted actions from pixel inputs. Nevertheless, this successful show the potential of neural networks for autonomous navigation.

With the development of deep learning, mainly CNN based model [12], Nvidia proposed a similar idea that fully utilizes convolutional networks [7][13] to extract the features from driving images. This stacked convolutional layer is adjusted so that it can map the high dimensional features into vehicle control, a steering angle. It shows a successful result with a straightforward scenario such as highway lane following and driving in obstacle-free courses.

Following that basis, this mechanism expands with a multi-modal multi-task framework [6], which combines CNN based networks with FCN-LSTM. They address end-to-end steering control with new speed prediction. They use Udacity and SAIC dataset to evaluate their model and shows that their model has a better mean absolute error in steering angle prediction.

The recent works focus on the variation of CNN based model on various performance evaluation. There is an ablation test of imitation learning in a successive percentage introduced by F. Codevilla et al. [14]. On multi-task learning, S. Chowdhuri et al. [15] presents multi-modal multi-task learning with a comprehensive evaluation of multiple types of datasets. This model develops into Multi-task learning from Demonstration (MT-LfD) [16] combining ResNet and Soft Attention to measure visual affordance.

Our work wraps the multi-modal and multi-task function with the Nvidia basis model on context learning. Driving behavior becomes the evaluation parameter. These parameters are defined through the accessible information in testing simulation.

III. Methodology

A. Working Flow

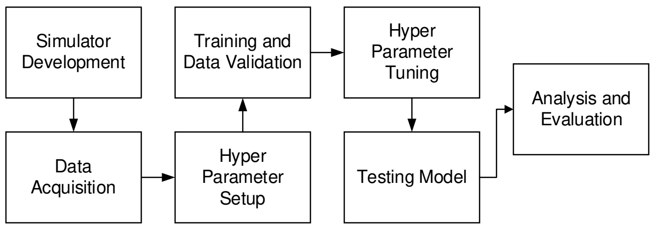


Fig. 1. Project working flow

This work starts from setting up and overwritten simulator program tailoring up to the test designed. It requires an understanding of the Carla environment to add some of the codes into Carla's python classes. Some of these classes are *roaming_agent*, *basic_agent*, *local_planner*, and *autonomous_controller*. These sections are essential parts to adjust to implementing an autonomous car algorithm.

All of these modifications are also useful for helping the user in capturing and labeling the data to create a training set and test set, which commonly requires a lot of time. When the dataset is ready, predefined hyperparameters need to be configured, such as *number of epochs*, *learning rate*, *batch size*, *loss function*, etc. Later, the obtained data is trained and validated with the ratio of 7:3, follows *K-fold cross-validation* recommendation to get a good proportional dataset. The engineering process is taken place in the tuning process to achieve the best model performance.

After all parts of the design are set, training and data validation, as well as testing model, are executed. We track and save the results to multiple training logs. Through these saved data, the model is analyzed and evaluated.

B. Simulation Environment

In this work, we simulate the model in Carla Simulator [17] version 0.9.5, which has a numerous improvement in comparison to 0.8.4 version, such as python API, waypoint ID, autopilot system, etc. We also use a navigation-based Proportional Integral Derivative (PID) controller, which employs a control loop on sensor feedback for the autonomous system. It is hardcoded AI that works imitating the human driver, such as stopping when red light, running while green light, keeping the speed up in the different lane, and distance up to the front car. Moreover, we can record all acquired data, such as agent states and parameters, images, and configuration information.

Specifically, we emulate the model in the Town02 map with clear noon weather conditions. By detail, the weather parameter has cloudiness 15, precipitation 0, wind intensity 0.35, sun azimuth angle 0, and sun altitude angle 75. We also set up an overall number of vehicles into 50 random means of transport, ranging from regular cars, bicycles, motorbikes, and trucks. These vehicles spawn in different positions. Their movement is also spreading randomly and follow the way of an expert agent moved. So, it follows the traffic rule and speed limit.

In the environment, there are varieties of landmarks, such as buildings, trees, warehouses, high buildings, open space areas, and many more. All these landmarks are part of self-driving car challenges, how the agent can withstand the irregularities in the environment as well as keeping the lane and distance with non-player vehicles while maintaining the rule such as speed limit and traffic light.

Non-player vehicles have deterministic behavior. This kind of unrealistic nature will make an expert agent easily derives its policy. As our test focuses on driving experience on a different model to the target goal, we don't put stochastic nature to these vehicles. Instead, apart from the standard capture (without added noise), temporal noise is added to the expert agent to cover all of the possible states in the test session.

All of these data are captured on the PC with a slightly higher spec, i7 6700 3.4 GHz, ram 16 GB, GPU 1080 GTX in python 3.7. The PC is also equipped with 3rd parties library for training and testing purposes (*Tensorflow-gpu*, *Keras*, and library dependencies). Our test on this computer can achieve performance around 30 to 115 fps depends on the non-player crowd.

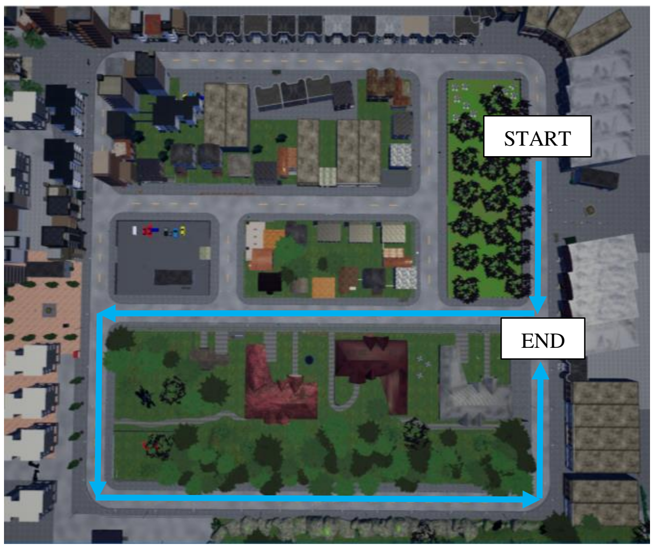
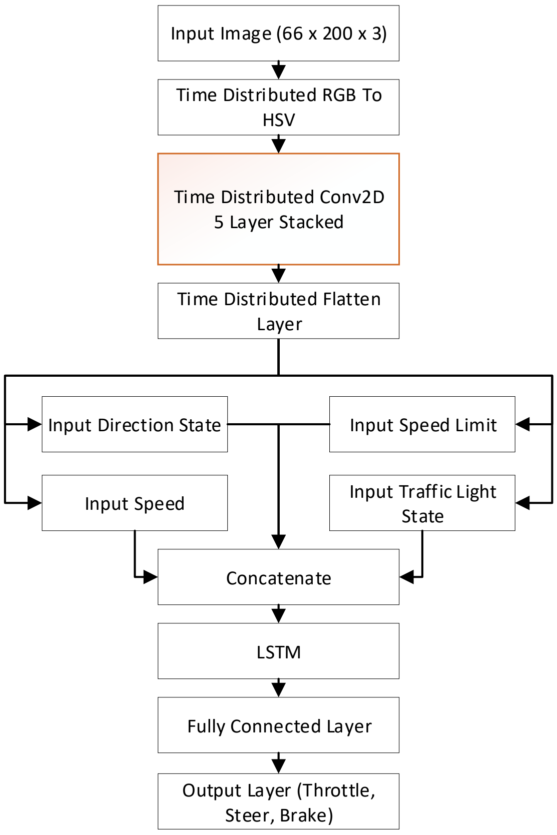


Fig. 2. Example of the arbitrary route generated randomly in the Carla environment.

C. Model Architecture



Input Image (66 x 200 x 3)
Time Distributed RGB To HSV
Time Distributed Conv2D 5 Layer Stacked
Time Distributed Flatten Layer
Input Direction State
Input Speed Limit
Input Speed
Input Traffic Light State
Concatenate
LSTM
Fully Connected Layer
Output Layer (Throttle, Steer, Brake)

Fig. 3. Design of Time Distributed model which comprised of 5 stacked Conv2D layer and LSTM.

The design of our model architecture comprised of Long-term Recurrent Convolutional Network design [14] with some tuning and idea of the Pilotnet model [7]. It includes three control parameter which defines the state of the autonomous agent.

Firstly, we crop and resize the input image into 66 x 200 x 3, adhering Nvidia designed model [7] as it has been tuned in that way to achieve a good result. Then, it follows with HSV convert, to remove noisy information. After that, the six stacked convolutional layers will extract images into high-level features of data. The main differences between normal and time distributed are:

- Time distributed is a wrapper which applies time slice of an input. In other words, the layer put on a dense layer on n timesteps depends on the configuration.
- The weights and biases will not change until the batch of the sample concerning time steps shifts into another distribution. This part is also consequent with a standard layer without time steps.

After flattening the output of CNN, the four states (direction, speed limit, speed, and traffic light state) are concatenated. In the following part, LSTM will function as a context descriptor on the sequence of images. Finally, a fully connected layer will map these into the driving control parameter.

D. Training Data Acquisition

Consider a controller that interacts with the environment over discrete time steps. At each time step t, the controller receives an observation o_t and takes an action a_t. The training data is a set of observation action pairs with respect to:

$$D = \{\langle o_t, a_t \rangle\}_{t=1}^{N} \quad (1)$$

generated by the expert agent. The assumption is that the expert has accomplished the driving from a certain point until it reaches the destination. The autonomous agent will mimic the path or lane that the expert usually used to reach the goal.

We make our recorder to capture the driving images and another parameter. We use the sensor RGB or front view camera provided by Carla. To avoid redundancy in image acquisition, we sampled ten frames per second (FPS) for every episode.

There are three types of images that we captured, training data in default setup condition with another vehicle, without vehicle and noise added. These three differences in the dataset are necessary for the sake of covering all the possible positions of the agent so that it can recover from an unwanted situation or place. We acquired this data by randomly set up start location and stop location.

Fig. 4. The sample image, captured through the front view camera in Carla's environment.

IV. Result and Analysis

In this section, we provide the result of our test simulation, which started from model training until experiment analysis.

A. *Model Training and Validation*

After a long process of data acquisition, we obtained 101,360 images with detail such as frame number, speed, throttle, brake, speed limit, traffic light state, direction, simulation time, etc. Then, we filter out into just six parameters, which have a composition of direction state, traffic light state, speed limit state, and speed as X, while throttle, steer and brake work as Y. Using Keras, we train our model with the following configuration.

TABLE I. TIME DISTRIBUTED BASED MODEL CONFIGURATION

Hyper Parameter	***Configuration***
Learning Rate	0.00012
Batch Size	16
Loss	Mean Square Error
Optimizer	Adam

In this training process, the total number of parameters is 802,199. We use the Early stopping module to reduce overfitting as well as get the best number of epochs in training before it stops improving. In our first experiment with the Nvidia model, the training ends after it reaches 40 epochs, with training loss 0.0054 and validation loss 0.024. The result shows that the model can converge very well. Furthermore, it can achieve training loss ~ validation loss, which means that the model is neither overfitting nor underfitting.

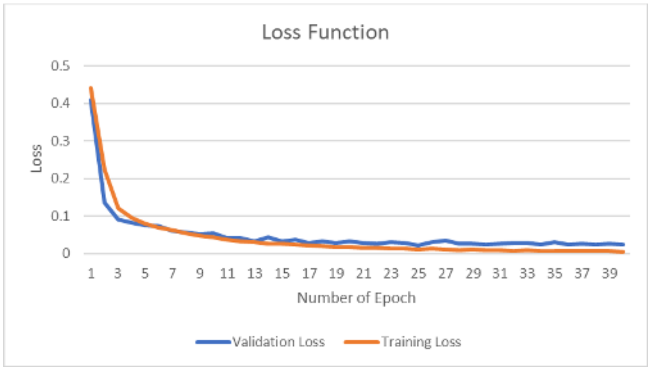


Fig. 5. Loss function graph of Time-distributed based model training.

B. *Route Data Statistics*

We also compile data statistics of the route which the agent passes. Based on Figure 5, the road dominantly shows a straight path with some movement of steering angle. In Figure 6, it also indicates that the PID or the ground truth agent is braking quite often in the test route, as there are many vehicles around, even though it experiences red light not so often. On the other hand, the course mostly has a lane with a 30 Km/h speed limit with dominance in the lane-follow path.

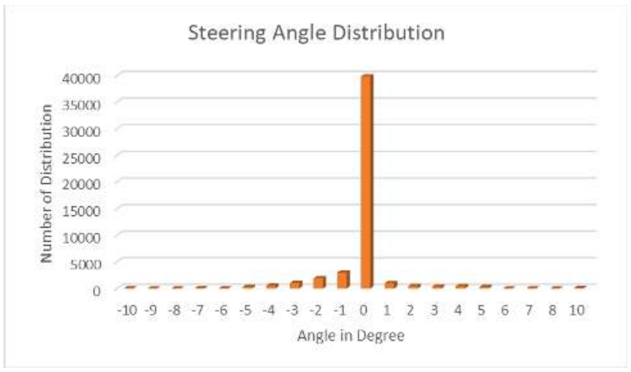


Fig. 6. Steering angle distribution in training data.

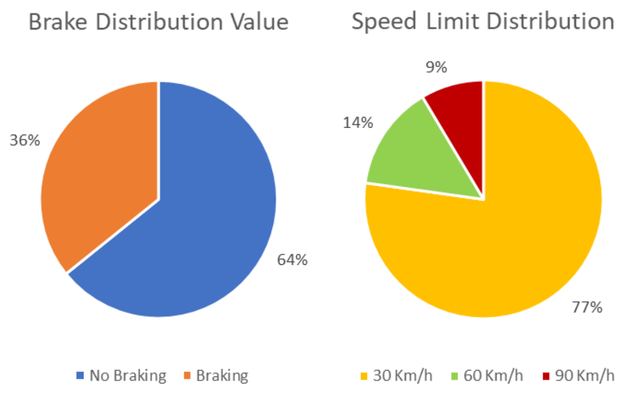


Fig. 7. Brake and speed limit distribution value on the designated routes in simulation.

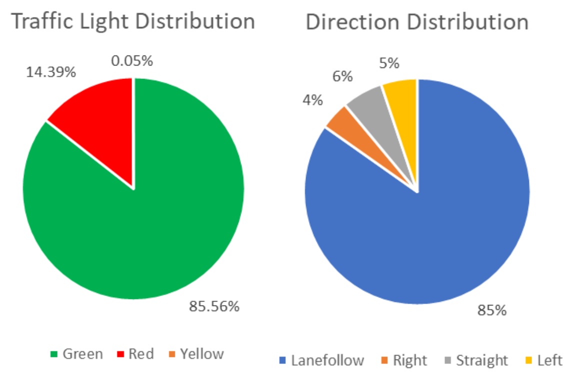


Fig. 8. Percentage of encountered traffic light distribution and direction distribution experienced by the agent on the designated routes.

C. *Visualization Activations*

One of the means to know how our deep CNN can successfully classify or predict the input images is by knowing what our CNN model sees the input. We can visualize the intermediate actions with Keras model that we have trained and take batches of images as input. This representation also gives a view of how our CNN model decomposed images into a visual concept of features.

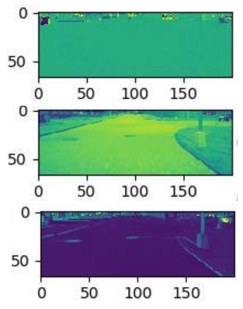


Fig. 9. The output of HSV converter from RGB images before getting into the CNN layer.

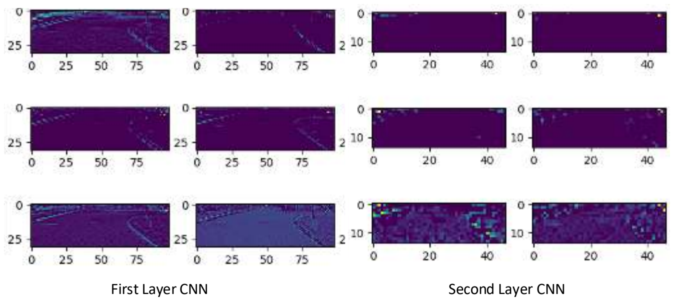


Fig. 10. Sample of visualization activation after pass-through from the first to the second layer of CNN.

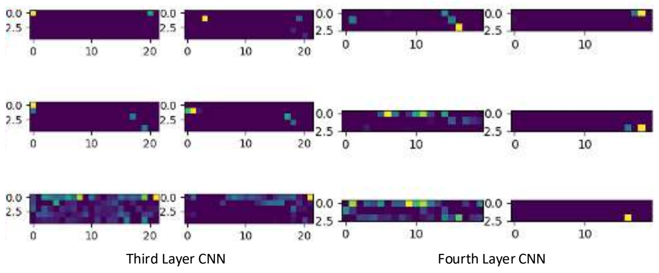


Fig. 11. Sample of visualization activation after pass through the third to the fourth layer of CNN.

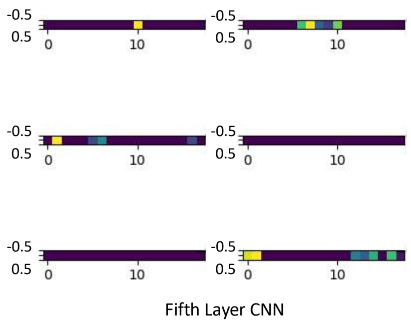


Fig. 12. Sample of visualization activation after pass through the fifth layer of CNN.

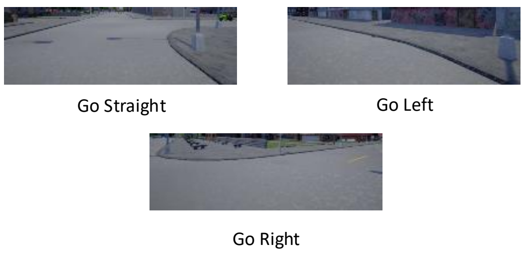


Fig. 13. Sample of output classification using the trained model.

D. *Test Simulation*

We run the simulation and arbitrarily set up the route concerning the map of Town02. We use a PID based autonomous sensor (manually coded Artificial Intelligent (AI)) as a ground truth. We evaluate the performances of time distribute model by comparing with ground truth, and CNN based [7] data test simulation. The data is sampled through ten of successful driving from setup position A to B. Through all of this round of the simulation, we calculate steering score, which we obtain from a mean of summing absolute steering value. There are also *speed above* and *below the score*, obtained through calculating the mean of the difference between *speed* and *speed limit* value.

A higher steering score means that the agent moves more agitated, and the speed score indicates haste of the agent. Consequently, a higher speed below the rating shows how good the agent can stick to the speed limit concerning the current lane. Conversely, higher in *speed above score*, point out that the car cannot keep up with the speed limit.

TABLE II. MEAN OF PERFORMANCE COMPARISON OF TIME DISTRIBUTED MODEL TO GROUND TRUTH DATA AND STACKED CNN MODEL

Perf. Parameter	***Ground Truth***	***Time Distributed***	***Stacked CNN***
Steering Score	1.34	3.71	4.21
Speed Score	6.264	9.65	10.5
Speed Below Score	18.013	15.269	17.02
Speed Above Score	3.605	13.274	8.418
Real Time Duration (s)	141.303	109.1841	138.586
Total Crossed Lines	0	6	3

According to Table II, ground truth result shows an excellent example of performance with zero total crossed lines, low steering score, and low-speed control. The agent can also control the speed limit problem very well, by having a high low speed below the score and low speed above score, which adheres to the traffic rule. Although time generally distributed inferior in comparison to this ground truth data, the model poses a better result in comparison to stacked CNN. It has a better steering score, and speed score compares to stacked CNN, but it has low control in the speed limit, which causes to cross lines many times. Consequently, the time distributed agent shows a peculiar result in the time of reaching the destination. It remarks the best time to reach the goal. This unexpected behavior comes from how time distributed agent handles the speed in every lane with a different speed limit.

In the following table, a time distributed agent can achieve 93.978 s in the same route with good steering score and speed score. Nevertheless, it traverses the line 8 times with a high speed above score. Whereas stacked CNN can achieve the best time 98.29 with a slightly high steering score, but it is more stable by not crossing the line at all.

TABLE III. THE FASTEST TEST OF AN AGENT ON A DIFFERENT MODEL

Perf. Parameter	*Time Distributed*	*Stacked CNN*
Steering Score	3.9	4.4
Speed Score	8.91	8.61
Speed Below Score	14.83	13.14
Speed Above Score	14.42	8.29
Real Time Duration (s)	93.978	98.29
Total Crossed Lines	8	0

V. CONCLUSION AND FUTURE WORKS

In this paper, we develop and evaluate end-to-end time distributed based model, a combination of CNN FCN-LSTM with multiple input and states, and multiple-output prediction. There are four states included in this test, direction, speed limit, traffic light, and speed. There is also a parameter that we predict from all these states, Steering, Braking, and Throttling. Basically, from our experiment, these three parameters are sufficient to define the movement of the cars.

Experiments show that the time distributed agent can reach the destination exceptionally faster than the PID control based navigation and stacked CNN based model. As a result, it yields a *low speed below the score* and a *higher speed above score*. Though, it also displays that the agent is less agitated and better speed score than the stacked CNN model.

Overall, from the steering and speed score, the result indicates that time distributed model slightly better than the stacked CNN model. Still, it suffers from speed limit management problems. Although finished faster in some cases is the top priority, stability in casual car driving would be far more critical.

On the other hand, we find that we should add more variables into the data set, such as weather and day (night, afternoon and morning), to know the robustness of time Distributed model toward a change of visual light perception and noisy environment. This robustness should also be tested with a stochastic nature non-player agent with a more comprehensive evaluation of driving performance.

Another thing that we concern about is the absence of pedestrians, which is also part of the city environment. While on our test, Carla 0.9.5 still not yet provides pedestrian features, so that in the future work, this part should be added.

REFERENCES

[1] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in Proceedings of the IEEE International Conference on Computer Vision, 2015, pp.

[2] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue et al., “An empirical evaluation of deep learning on highway driving,” arXiv preprint arXiv:1504.01716, 2015.

[3] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, and V. N. Murali, Deeplanes: End-to-end lane position estimation using deep neural networks,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2016, pp. 38–45.

[4] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun, “3d traffic scene understanding from movable platforms,” IEEE transactions on pattern analysis and machine intelligence, vol. 36, no. 5, pp. 1012–1025, 2014.

[5] H. Zhang, A. Geiger, and R. Urtasun, “Understanding high-level semantics by modeling traffic patterns,” in Proceedings of the IEEE International Conference on Computer Vision, 2013, pp. 3056–3063.

[6] Z. Yang, Y. Zhang, J. Yu, J. Cai, and J. Luo, “End-to-end Multi-Modal Multi-Task Vehicle Control for Self-Driving Cars with Visual Perceptions,” arXiv:1801.06734v2, 2018.

[7] M. Bojarski, et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.

[8] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” arXiv preprint arXiv:1612.01079, 2016.

[9] T. Glasmachers, “Limits of end-to-end learning,” arXiv preprint arXiv:1704.08305, 2017.

[10] A.C. Serban, E.. Poll, and J. Visser, “A Standard Driven Software Architecture for Fully Autonomous Vehicles,” 2018 IEEE International Conference on Software Architecture Companion (ICSA-C), IEEE, 2018.

[11] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in Advances in neural information processing systems, 1989, pp. 305–313.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.

[13] M. Bojarski, et al., “Explaining how a deep neural network trained with end-to-end learning steers a car,” arXiv 2017, arXiv:1704.07911.

[14] F. Codevilla, M. Muller, A. Lopez, V. Koltun, and A. Dosovitskiy, “End-to-end Driving via Conditional Imitation Learning,” International Conference on Robotics and Automation (ICRA), 2018.

[15] S. Chowdhuri, T. Pankaj, and K. Zipser, “MultiNet: Multi-Modal Multi-Task Learning for Autonomous Driving,” arXiv:1709.05581, 2019.

[16] A. Mehta, S. Adithya, and S. Anbumani, “Learning end-to-end autonomous driving using guided auxiliary supervision,” arXiv 2018, arXiv:1808.10393.

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. L´opez, and V. Koltun, “CARLA: An open urban driving simulator,” In Conference on Robot Learning (CoRL), 2017.