Autonomous Driving on a Direct Perception System with Deep Recurrent Layers

Bruno Correia Almeida
Autonomous Computational Systems Lab, Aeronautics Institute of Technology (ITA)
São José dos Campos, Brazil
bruno.correia.almeida@gmail.com

Paulo André Lima de Castro
Autonomous Computational Systems Lab, Aeronautics Institute of Technology (ITA)
São José dos Campos, Brazil
pauloac@ita.br

ABSTRACT

Controlling a car in traffic is a complex task that is hard to treat with classical algorithms. With the fast development of computer vision techniques, deep artificial neural networks have been use on this task with the use of a frontal camera on a car. Few research projects with these techniques, however, go beyond the perception task and are tested on a driving environment.

The use of recurrent neural networks combined with convolutional neural networks has achieved good results in analysis of videos and image streams because it can capture temporal features. We achieve a better performance with the use of recurrent layers combined with a convolutional network in a autonomous driving system based on direct perception, which is composed by two steps: the perception of indicators on the image from the car frontal camera and the control of the car that directly uses these indicators to drive the car on a simulator.

Compared to the original work we achieved a percentual decrease of the mean absolute error on the indicators perception task of 36%, with the use of a deeper convolutional architecture with residual connections, and 48% when adding recurrent layers to this network. In the autonomous driving test on the simulator we found that the best system was composed by an hybrid of these two networks, using recurrent layers only for part of the indicators.

The system developed on this project, *PyDeepDriving*, is an extension of the original system from the Princeton Vision & Robotics Group, *DeepDriving*. We open sourced it to facilitate the research of autonomous driving systems based on direct perception.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Control methods*; *Scene understanding*; *Computer vision problems*; Transfer learning;

KEYWORDS

Machine Learning, Autonomous Driving, Video Analysis

APPIS 2019, January 7–9, 2019, Las Palmas de Gran Canaria, Spain

ACM ISBN 978-1-4503-6085-2/19/01…$15.00
https://doi.org/10.1145/3309772.3309790

ACM Reference Format:
Bruno Correia Almeida and Paulo André Lima de Castro. 2019. Autonomous Driving on a Direct Perception System with Deep Recurrent Layers. In *2nd International Conference on Applications of Intelligent Systems (APPIS 2019), January 7–9, 2019, Las Palmas de Gran Canaria, Spain.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3309772.3309790

1 INTRODUCTION

Autonomous driving has great potential to change the way we live. They are implemented by interpreting the space around the car with different sensors like RADAR [15], LIDAR [11], GPS and cameras. Advanced control systems are employed to make use of all this information to identify obstacles, lanes and plan the car's route.

The usage of deep neural networks has shown great results in many applications, as shown in [10]. It excels in many computer vision tasks and has recently been used on the autonomous driving problem, as seen on [7] and [1]. A deep neural network is usually used to extract information from the camera and then, together with other sensors, a scene understanding for the car is achieved.

The use of recurrent networks (RNNs) combined with deep convolutional neural networks (CNNs) has achieved good results in analysis of videos and image streams [3] because it can capture temporal features. But while the fast progress of computer vision techniques has encouraged the use of these techniques on research of autonomous driving systems on a car with a frontal camera, few projects go beyond the perception task and few are tested on a driving environment. In this project we were able to attest the performance using deep convolutional and recurrent networks by not only measuring the perception error of these networks but also by driving on the simulator.

In this project we explore the direct perception approach for self driving cars, created on the *DeepDriving* project [2], to run an autonomous car using just visual information, without any extra sensors, and with a simple approach. The goal is use different neural network architectures for this problem and analyze the usage of the motion information for this approach via recurrent neural networks, as well as the viability of using transfer learning [17]. To achieve this we developed a new version for [2], which we named *PyDeepDriving*, to enable the use of modern deep learning frameworks, that is open source and distributed on https://github.com/bcalmeida/PyDeepDriving.

2 FOUNDATIONS

To give context on the vision based autonomous driving solutions, we can classify them in three categories from the most complex to the most simple:

- **Mediated Perception**, in which a scene is analyzed to extract as much information as possible using detection, lane finding, segmentation and other algorithms and used together with other sensors like LIDAR, RADAR, GPS, etc. A scene understanding for the surrounds of the car is created and from that a controller takes the decision for the driving commands, like on [14] and [4].
- **Direct Perception**, the method introduced in *DeepDriving* [2], learns a mapping from an image to several meaningful indicators of the road situation, including the angle of the car relative to the road, the distance to the lane markings, and the distance to cars in the current and adjacent lanes. With this compact but meaningful representation as perception output, it uses a very simple controller to make driving decisions and drive the car smoothly.
- **Behavior Reflex** are approaches where a direct mapping is constructed from the input to a driving action. It is seem on the classic ALVINN [12] and more recently on NVIDIA's end-to-end learning [1]. This method fails to capture a bigger picture of the situation. This level of abstraction fails to capture what is really happening and shows a erratic driving behavior.

In this project we use convolutional neural networks with residual connections [6]. Deep Residual Networks (ResNets) use a building block composed of a skipping connection, that can be defined by $y = F(x, Wi) + x$ where x is the input vector, y the output and Wi are the parameters, that allows deeper networks while avoiding the problem of vanishing gradients.

We use a long short-term memory (LSTM) recurrent neural network architecture [5]. The architecture used in our experiments is given by the following equations, where for each element in the input sequence, each layer computes the following:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t c_{(t-1)} + i_t g_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, $h_{(t-1)}$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time 0, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function.

In this project we also investigate the use of transfer learning, as described in [17], a technique with widespread use nowadays in computer vision. With it we use pretrained features from on another dataset, enabling the use of less data. Despite dealing with computer graphics from a simulator in this project, the earlier layers of a network acts as pretty general shape detectors which can generalize well to various datasets.

3 SYSTEM ARCHITECTURE

3.1 TORCS

TORCS, *The Open Racing Car Simulator* [16], is an open source driving and racing simulator that is used mostly for research. It supports the creation of bots and can be extended by researchers to implement features needed for their work. The great advantage of using a simulator is the ease of testing your model, avoiding the need to deploy on a real car, reducing the time and cost of the experiments.

In *DeepDriving* [2] they modified TORCS to show lane markings as well as added bots that simulate traffic cars instead of racing cars. This modified version of TORCS is used in this project.

3.2 PyDeepDriving

To do all the experiments in this project we developed a new version of *DeepDriving*, from [2], which we named *PyDeepDriving*. It enables the use of modern Python based deep learning frameworks, provides ways of creating different direct perception systems with different indicators and provides better modularity and extensibility. While the original version, *DeepDriving*, used a modified outdated version of the Caffe framework [8], that had implemented not only the neural network model but also the driving controller and all the communication with TORCS in a monolithic and hard to modify way. Figure 1 shows a high level architecture of the projects.

With that we provide an easier and more pleasant way of working with a direct perception autonomous driving and hope that it contributes to further research and interest in this approach. We noticed there is an interest in the community on the original *DeepDriving* because of the opportunity to work on the whole stack of a driving system, but the tools provided were out of date and harder to use. *PyDeepDriving* is published on https://github.com/bcalmeida/PyDeepDriving.

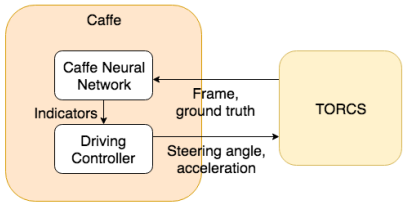
3.3 Indicators

When the car is in lane, centered between the lane markings, the model looks for the distances/indicators described on the list below. They are shown on Figure 2a and 2b.

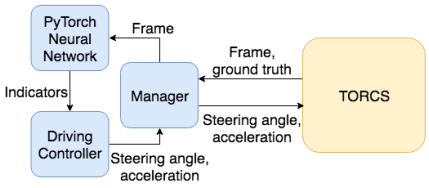
(1) toMarking_LL: distance to the left lane marking of the left lane
(2) toMarking_ML: distance to the left lane marking of the current lane
(3) toMarking_MR: distance to the right lane marking of the current lane
(4) toMarking_RR: distance to the right lane marking of the right lane
(5) dist_LL: distance to the preceding car in the left lane
(6) dist_MM: distance to the preceding car in the current lane
(7) dist_RR: distance to the preceding car in the right lane

When the car is on a lane changing situation, getting close or on the lane markings, the model looks for the distances/indicators described below. On Figure 2c and 2d we can see these indicators.

(1) toMarking_L: distance to the left lane marking
(2) toMarking_M: distance to the central lane marking
(3) toMarking_R: distance to the right lane marking
(4) dist_L: distance to the preceding car in the left lane

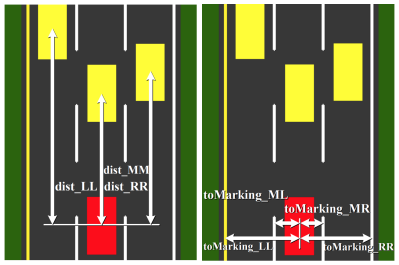


(a) DeepDriving [2] architecture

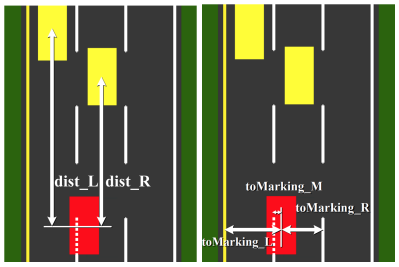


(b) PyDeepDriving architecture

Figure 1: Architecture of the original *DeepDriving* and the new developed *PyDeepDriving*



(a) `dist` indicators when In Lane **(b) `toMarking` indicators when In Lane**



(c) `dist` indicators when On Mark **(d) `toMarking` indicators when In Lane**

Figure 2: Representation of the indicators for the perception system in the different states, from [2]

(5) dist_R: distance to the preceding car in the right lane

There are also 2 other indicators that are used in both in lane and on mark situations:

(1) angle: angle in radians between the car and the tangent of the road segment

Table 1: Summary of the indicators used in each state

In Lane State	On Mark State	Any State
dist_LL	dist_L	angle
dist_MM	dist_R	fast
dist_RR	toMarking_L	
toMarking_LL	toMarking_M	
toMarking_ML	toMarking_R	
toMarking_MR		
toMarking_RR		

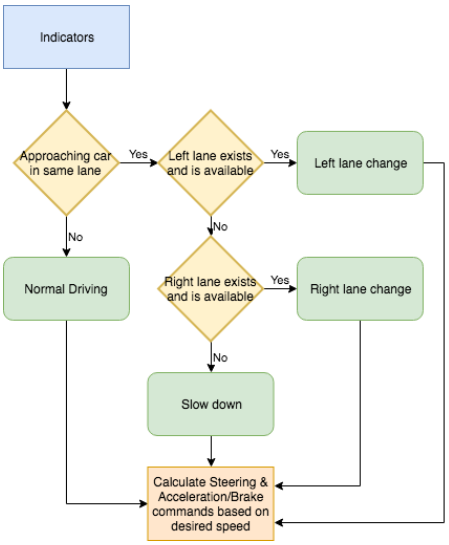


Figure 3: Diagram of the controller behavior

(2) fast: boolean stating if the road is straight or if there is a curve ahead

There are further implementation details, e.g. when the car is changing lanes there is an overlap area in which the car is in both in lane and on mark state, but these are not included here for not providing greater insight on how the indicators work. For further details refer to the original work on [2]. All the indicators are summarized on Table 1.

3.4 Controller

The controller used follows a simple high level logic, which is shown on the diagram on Figure 3, in which the car changes lanes according to the need of maintaining a baseline speed. Not shown on the diagram is that the car tries to stay on the center line. The direct perception system shows that it is possible to create a simple way of controlling, with vision only, a car given a proper choice of indicators for the image.

4 EXPERIMENTS AND RESULTS

To evaluate the performance of neural networks with recurrent layers on the perception system as well as the performance of deeper convolutional neural networks than used in [2] we need

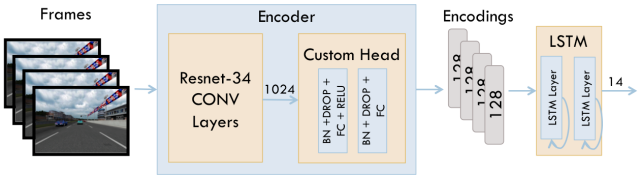


Figure 4: Architecture of the Resnet-LSTM model

to analyze the system not only on the perception task, but on the actual driving simulation as well. The following sections describes the experiments regarding this two aspects: perception and driving.

4.1 Perception

To assess the performance of recurrent neural networks and the performance of deeper convolutional neural networks, we trained a model with a deep CNN with residual connections, which we call the *Resnet* model, and a model using a similar CNN combined with LSTM layers, named *Resnet-LSTM*.

For all the experiments in this section we used we used 223298 images from the dataset provided by [2], which contains frontal camera images labeled with all the 14 real value indicators as described in section 3.3. It is captured with a human driver at a 10 frames per second rate in one, two or three lane roads.

The Resnet model is composed of the convolutional layers of a Resnet-34 based architecture [6] with 34 layers, pre-trained on ImageNet [13], and a fully connected custom head with 2 layers. The custom head was determined empirically by training performance. Dropout and batch normalization techniques helped minimizing overfitting.

We found that transfer learning with pre-trained weights helped the training process by avoiding overfitting without needing to add more training images. Despite the computer graphics of the images being really different from those on the ImageNet competition, general shape detectors are learned on the earlier layers of the network. The original work, *DeepDriving* [2], did not use transfer learning.

The Resnet-LSTM model uses a deep convolutional network as an encoder of images with the same architecture as the Resnet model, but with a different ouput dimension. The Resnet-LSTM model architecture can be seen on Figure 4. This combination of CNN and RNN is seem on video analysis [3], as the network can learn temporal features from the video. We also tried feeding the previous network output concatenated with the image encoding to the LSTM, but we didn't see any improvement despite the higher complexity. Each image in a sequence is encoded to a vector of length 128 and then used as input to a 2-layer LSTM network. Higher encoding sizes didn't show better results, as the values are too low level for the LSTM to process.

We also need to compare the perception performance with the original work from [2]. They shared few metrics about the network performance on the perception task stating that the network performance test was made by watching it driving on the simulator. It does, however, provide the Mean Absolute Error (MAE) of it's fully trained Alexnet [9] based network when tested on a simpler situation, with 2 lanes only, when comparing with other methods on the paper. We use these numbers and scale it with the indicators ranges to be able to compare errors of indicators with different magnitudes, as shown on Table 2. For example, while the indicator `angle` is in the range -0.5 to 0.5, all the `dist` indicators, like `dist_M`, are on the range 0 to 75.

Since the error metric for the `fast` indicator is not provided, we calculate the average MAE for all indicators and the average MAE for all except the `fast` indicator. We can see these values on Table 3. From these numbers we see that the `fast` indicator has a higher error when compared to any other indicator. This is because the `fast` indicator should have been treated as a categorical variable and not a continuous variable between 0 and 1, and also because the definition of this indicator, which is whether the road is curved, is too broad and has cases of slightly curved roads that are hard to define. The `fast` indicator should have been trained, then, with a separate head on the network that had a classification loss function, instead of been treated as a regression like the other indicators.

From Table 3 we can see that there is a percentual decrease of the average mean absolute error of the indicators of 36% with the Resnet model and a decrease of 48% with the Resnet-LSTM model.

4.2 Driving

To test the driving performance we collect metrics during the autonomous driving on TORCS on *PyDeepDriving*. We collect more than 17000 frames to be analyzed from allowing the system to drive in a three lane track situation with others AI-controlled cars. We analyze the average speed of the car and average perception error, as a way of making the driving analysis more quantitative. We do not count metrics like number of collision because they rarely occur given the simplicity of the direct perception system and environment.

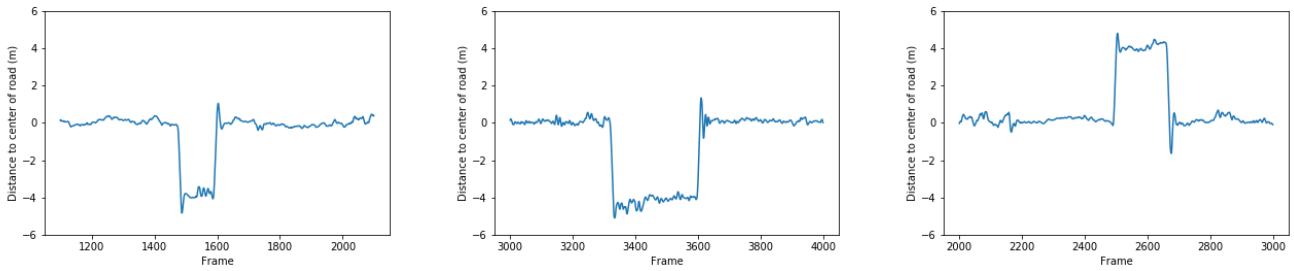
The average speed differences on Table 4 are small because during long simulations most of the time the car is on the middle lane at full speed. The Average MAE are less than expected because on the simulation great part of the time the car is driving without cars around, which is easier on for the perception network, while the dataset for the network is collected by a human driver that deliberately puts the car around more traffic.

When driving with the Resnet-LSTM as the perception network, we notice that the car slightly oscillates on its lane and sometimes overshoots when changing lanes. We can compare it with the Resnet model by looking at Figure 5a and Figure 5b, in which we have the distance of the car to the center of the road for a number of frames from the simulation. In the analyzed section we see the car driving on the middle lane and, for a brief moment, on the right lane. We can see the overall higher oscillation of the Resnet-LSTM car, and a overshoot and following oscillation when moving back to the center lane. This behavior makes it a worse driving system on simulations compared to the Resnet model, despite having a better perception error metric as shown on Table 4. This explains the lower average speed shown on Table 4 that Resnet-LSTM has, since it wastes time on unnecessary oscillations.

The oscillating behavior is caused by the inertia that the stateful LSTM causes on the network inferences. While this results in a better error metric because of the motion information that it captures, it presents a problem when it mispredicts the motion of the

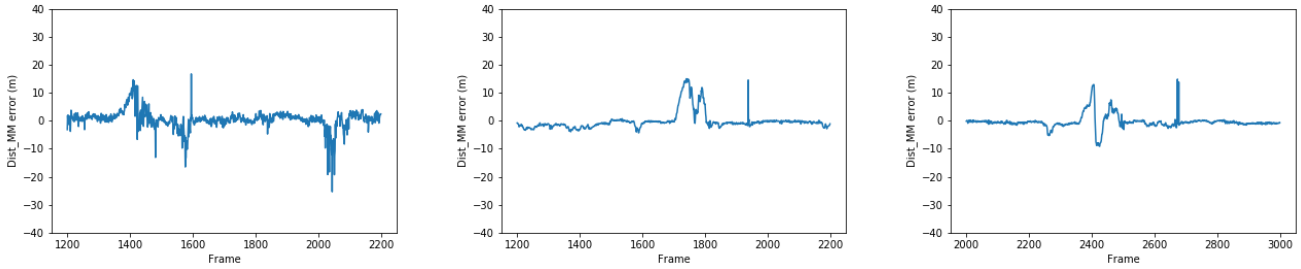
Table 2: Scaled MAE per indicators for each model

Model	to_L	to_M	to_R	d_L	d_R	to_LL	to_ML	to_MR	to_RR	d_LL	d_MM	d_RR	angle	fast
Alexnet	0.075	0.056	0.070	0.117	0.143	0.038	0.031	0.032	0.037	0.068	0.063	0.106	0.033	*unavailable*
Resnet	0.046	0.048	0.050	0.034	0.036	0.04	0.038	0.037	0.046	0.044	0.042	0.047	0.043	0.291
Resnet-LSTM	0.037	0.036	0.040	0.028	0.031	0.027	0.032	0.031	0.034	0.031	0.037	0.034	0.047	0.227



(a) Resnet **(b) Resnet-LSTM** **(c) Resnet + Resnet-LSTM**

Figure 5: Distance to center of road in meters during part of the of the driving simulations



(a) Resnet **(b) Resnet-LSTM** **(c) Resnet + Resnet-LSTM**

Figure 6: Error of the `dist_MM` indicator during part of the driving simulations

Table 3: Average of Scaled MAEs for each model

Model	Average MAE	Average MAE without `fast`
Alexnet	*missing info*	0.066
Resnet	0.060	0.042
Resnet-LSTM	0.048	0.034

Table 4: Mean speed and error metric on driving simulations

Model	Mean Speed	Average MAE
Resnet	19.052	0.049
Resnet-LSTM	18.972	0.041
Resnet + Resnet-LSTM	19.166	0.045

car because of a sudden change. The error propagates in the direct perception driving system that lacks a sensor control system and just uses the actual value from the perception network.

The distance indicators, which represent distances to car in front, are less susceptible to this problem because there are fewer abrupt changes in the distance to the other cars in the environment and the values of these indicators only produce car movement decisions in specific situations. For example, when the car is the center lane the value of `dist_LL` does not affect the car behavior, as well as a large value of `dist_MM`.

With that we can build a model that builds on the strength of both models by using both models and using the Resnet-LSTM distance indicators and the other indicators from the Resnet model. We call this model *Resnet + Resnet-LSTM*. From Table 4 the average MAE is between both, as expected, and the average speed for this model is the highest. In Figure 5c we see a similar behavior to the Resnet model regarding the car movement, while on Figures 6a, 6b and 6c we see the error on the `dist_MM` metric, which shows a smoother behavior for Resnet-LSTM and Resnet + Resnet-LSTM.

5 CONCLUSIONS

The use of recurrent neural networks combined with deep convolutional neural networks has achieved good results in the analysis of videos and image streams [3] because it can capture temporal features. But while the fast progress of computer vision techniques has encouraged the use of these techniques on research of autonomous driving systems with a car with a frontal camera, few projects go beyond the perception task and are tested on driving environment. In this project we were able to attest the performance using deep convolutional and recurrent networks by not only measuring the perception error of these networks but also by driving on the simulator.

We proposed two networks, Resnet and Resnet-LSTM, for the perception of a direct perception system, which resulted in a percentual decrease of the average mean absolute error of the indicators inference of 36% and 48% respectively. With that we can attest to the performance of deeper networks in the perception problem and the increase in performance by using a recurrent layer that captures motion information from frame to frame.

This project, however, shows that it is not possible to naively just use the network with best performance and it is necessary to deeply understand how each indicator behaves and the drawbacks of each network. We showed that just using the superior Resnet-LSTM model did not work well, but a combination of the Resnet and Resnet-LSTM, allowing the latter to infer the `dist_*` indicators, performed the best.

As seem on Table 3. The `fast` indicator has a higher error than any indicator by a large margin. This is because the `fast` indicator should have been treated as a categorical variable and not a continuous variable between 0 and 1, and also because the definition of curve of a road in too broad and there are roads that are only slightly curved. The `fast` indicator would then be trained with a separate head on the network that had a classification loss function, instead of been treated as a regression problem like the other indicators.

This project also contributes to the research of direct perception autonomous driving systems by providing the community with an easier environment with *PyDeepDriving*. We noticed there is an interest in the community on the original work, *DeepDriving*, because of the opportunity to work on the whole stack of a driving system, but the tools provided were out of date and harder to use.

Future work could include boolean indicators that could indicate the state of the car (In Lane or On Mark) and also that could replace the `fast` indicator, and trained jointly with the other networks by using a loss function that combines the regression and classification problem. With the work developed in *PyDeepDriving* it is also easier to create a simpler set of indicators, write a Python controller and test if it achieves similar results.

ACKNOWLEDGMENTS

The authors would like to thank Chen et al. and the Princeton Vision & Robotics Group for providing the original *DeepDriving* source code, and CAPES (National Council for the Improvement of Higher Education) for their support.

REFERENCES

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[2] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In *Proceedings of the IEEE International Conference on Computer Vision*. 2722–2730.

[3] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2625–2634.

[4] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. 2013. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research* 32, 11 (2013), 1231–1237.

[5] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385* (2015).

[7] Brody Huval, Tao Wang, Sameep Tandon, Jeff Kiske, Will Song, Joel Pazhayampallil, Mykhaylo Andriluka, Royce Cheng-Yue, Fernando Mujica, Adam Coates, et al. 2015. An Empirical Evaluation of Deep Learning on Highway Driving. *arXiv preprint arXiv:1504.01716* (2015).

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*. ACM, 675–678.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[11] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. 2011. Towards fully autonomous driving: Systems and algorithms. In *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 163–168.

[12] Dean A Pomerleau. 1989. *Alvinn: An autonomous land vehicle in a neural network*. Technical Report. DTIC Document.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115, 3 (2015), 211–252.

[14] Shimon Ullman. 1980. Against direct perception. *Behavioral and Brain Sciences* 3, 3 (1980), 373–381.

[15] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics* 25, 8 (2008), 425–466.

[16] Bernhard Wymann, E Espi, C Guionneau, C Dimitrakakis, R Coulom, and A Sumner. 2013. TORCS, The Open Racing Car Simulator.

[17] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In *Advances in neural information processing systems*. 3320–3328.