Deep learning and control algorithms of direct perception for autonomous driving

Der-Hau Lee, Kuan-Lin Chen, Kuan-Han Liou, Chang-Lun Liu & Jinn-Liang Liu

ONLINE FIRST

Applied Intelligence
The International Journal of Research on Intelligent Systems for Real Life Complex Problems

ISSN 0924-669X

Appl Intell
DOI 10.1007/s10489-020-01827-9

Springer

Applied Intelligence
https://doi.org/10.1007/s10489-020-01827-9

Deep learning and control algorithms of direct perception for autonomous driving

Der-Hau Lee[1] · Kuan-Lin Chen[2] · Kuan-Han Liou[2] · Chang-Lun Liu[2] · Jinn-Liang Liu[2]

Abstract

We propose an end-to-end machine learning model that integrates multi-task (MT) learning, convolutional neural networks (CNNs), and control algorithms to achieve efficient inference and stable driving for self-driving cars. The CNN-MT model can simultaneously perform regression and classification tasks for estimating perception indicators and driving decisions, respectively, based on the direct perception paradigm of autonomous driving. The model can also be used to evaluate the inference efficiency and driving stability of different CNNs on the metrics of CNN's size, complexity, accuracy, processing speed, and collision number, respectively, in a dynamic traffic. We also propose new algorithms for controllers to drive a car using the indicators and its short-range sensory data to avoid collisions in real-time testing. We collect a set of images from a camera of The Open Racing Car Simulator in various driving scenarios, train the model using this dataset, test it in unseen traffics, and find that it outperforms earlier models in highway traffic. The stability of end-to-end learning and self driving depends crucially on the dynamic interplay between CNN and control algorithms. The source code and data of this work are available on our website, which can be used as a simulation platform to evaluate different learning models on equal footing and quantify collisions precisely for further studies on autonomous driving.

Keywords Self-driving cars · Autonomous driving · Deep learning · Image perception · Control algorithms

1 Introduction

The direct perception model proposed by Chen et al. [1] maps an input image (high dimensional pixels) from a sensory device of a vehicle to fourteen affordance indicators (a low dimensional representation) by a convolutional neural network (CNN). Controllers then drive the vehicle autonomously using these indicators in an end-to-end (E2E) and real-time manner. This paradigm falls between and displays the merits [1–3] of the mediated perception [4–8] and behavior reflex [9–13] paradigms. We refer to these papers, some recent review articles [14–18], and references there for more thorough discussions about these three major paradigms in the state-of-art machine learning algorithms of autonomous driving.

We instead study the interplay between CNNs and controllers and its effects on the overall performance of self-driving cars in training and testing phases, which are not addressed in earlier studies. CNN is a perception mapping from sensory input to affordance output. Controllers then map key affordances to driving actions, namely, to accelerate, brake, or steer [1].

These two mapping algorithms are generally proposed and verified separately since automotive control systems are very complex varying with vehicle types and levels of automation [14–19]. A great variety of simulators have been developed for simulation testing of autonomous cars in

✉ Jinn-Liang Liu
jinnliu@mail.nd.nthu.edu.tw;
http://www.nhcue.edu.tw/~jinnliu/

Der-Hau Lee
derhaulee@yahoo.com.tw

Kuan-Lin Chen
mark0101tw@gmail.com

Kuan-Han Liou
jason839262002@yahoo.com.tw

Chang-Lun Liu
leo28833705@yahoo.com.tw

[1] Department of Electrophysics, National Chiao Tung University, Hsinchu, Taiwan

[2] Institute of Computational and Modeling Science, National Tsing Hau University, Hsinchu, Taiwan

Published online: 08 August 2020

various aspects such as mobility dynamics, path planning, urban traffic, freeway traffic, traffic scene, and safety assessment [17]. However, there are very few [20] open source simulators like The Open Racing Car Simulator (TORCS) [21, 22] and Car Learning to Act (CARLA) [20] that can be used to develop an E2E simulation platform with both machine learning and controller tools for research investigation and verification.

Chen et al. have developed the open source platform named DeepDriving [1] that integrates the CNN AlexNet [23] to TORCS. This platform allows real-time simulation of a CNN pre-trained ego agent (called Host here) driving along with other TORCS agents (called Agents). The main difference between Host and Agents is that they use estimated and true affordance indicators, respectively, to autonomously control their own driving dynamics. It is even more importantly that DeepDriving allows researchers to extend, improve, or verify machine learning as well as control algorithms in a consistent and unambiguous way.

We propose here an E2E simulation architecture that allows multi-task (MT) learning [3, 24], applies to different CNNs predicting different indicators for performance evaluation, and includes new control algorithms to quantify and avoid collisions using sensory data and fewer indicators than those in [1]. We show that the architecture can be used to evaluate different CNNs on equal footing and the algorithms can avoid collisions for Host and Agents in testing phase.

Fig. 1 illustrates the simulation cycle of the architecture: (i) a traffic scene presents to a camera and sensors of self-driving car producing an image and other traffic data, respectively, (ii) the image feeds into a CNN producing driving indicators, (iii) the indicators and data put into a controller moving the car, and (iv) a new traffic scene displays to the car after movement.

End-to-end learning was first introduced by Pomerleau [9] in 1989 for road-following task using deep NN (DNN) and camera and LiDAR on a vehicle. LeCun et al. [11] proposed an E2E system in 2006 for obstacle avoidance task using CNN and camera on a 50 cm off-road truck. In the last five years, this methodology dramatically gains its popularity in academic as well as industrial sectors [19].

Table 1 displays a comprehensive list of E2E learning models in chronological order surveyed by Grigorescu et al. in 2019 [19] and added and compared by us from the aspects of driving tasks in learning (LTask) and testing (TTask) phases, neural networks (NN), sensory input (Sensor) to NN, and real-time testing (TTime) environment, where Dist is for distances from other cars and/or road/lane centerline, Accl for acceleration, Drive for steering, changing lanes, and overtaking, and R for recurrent. TTime is defined as whether the trained model is driving dynamically in real time (Yes) or not (No) using the controllers of a real (RYes), robotic (BYes), or simulated (SYes) car. Table 1 excludes reinforcement learning models in [19].

It is very dangerous and extremely costly to quantify collisions of self-driving cars in real-world traffic in the validation phase for a trained model. And yet, the collision is one of the utmost important tasks of autonomous driving that needs to be carefully analyzed and precisely quantified at least in the testing phase for evaluating learning and control algorithms. There is no earlier work in Table 1 addressing specifically this issue with quantitative analysis. However, it is unfair to compare different models trained on different hardware and software platforms that vary widely [14–19]. For example, Table 1 has already included quite different categories in LTask, TTask, Sensor, and TTime, let alone a great deal of learning tasks missing in the table but studied in a large volume of other publications such as

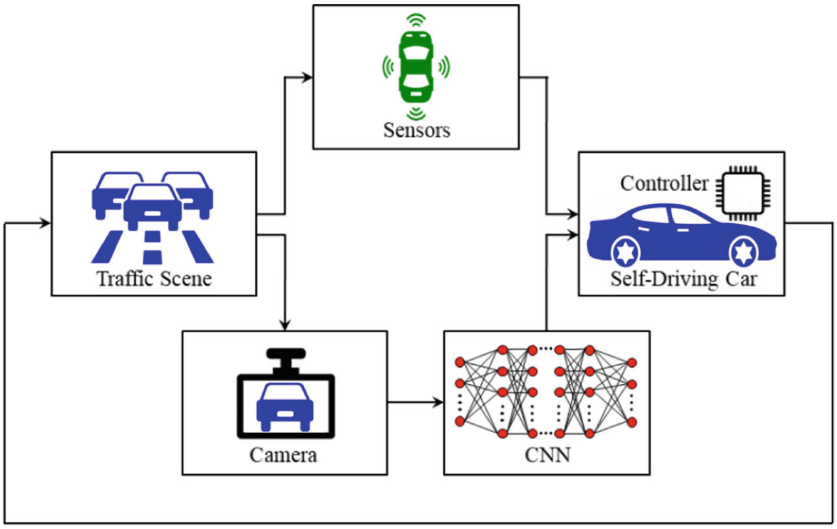

Fig. 1 End-to-end and real-time architecture of direct perception consisting of CNN and front-facing camera, short-range radars, and controllers of self-driving car. The arrows indicate the data flow of image from camera to CNN and then both driving indicators from CNN and sensory data from sensors to controllers

Table 1 Comparison of end-to-end learning models

Author	LTask	TTask	NN	Sensor	TTime
Pomerleau [9]	Steer	Steer	DNN	Camera, LiDAR	RYes
LeCun et al. [11]	Steer	Steer	CNN	Camera	BYes
Chen et al. [1]	Steer, Dist	Drive	CNN	Camera	SYes
Bojarski et al. [12]	Steer	Steer	CNN	Camera	RYes
Al-Qizwini et al. [2]	Steer, Dist	Drive	CNN	Camera, Radar	SYes
Xu et al. [25]	Motion	Motion	C+RNN	Camera	No
Eraqi et al. [26]	Steer	Steer	RNN	Camera	No
Rausch et al. [27]	Steer	Steer	CNN	Camera	No
Sauer et al. [3]	Steer, Dist	Steer	CNN	Camera, Planner	SYes
Codevilla et al. [13]	Steer, Accl	Steer	CNN	Camera, Planner	BYes
Hecker et al. [28]	Steer	Steer	C+RNN	Camera, Planner	No
Bechtel et al. [29]	Steer	Steer	CNN	Camera	BYes
This Work	Steer, Dist	Drive	CNN	Camera, Radar	SYes

road, lane, 2D and 3D object, and sign detection, semantic segmentation, localization, driving style recognition, etc. to name a few [14–19]. We thus only select the work in [1–3] for a suitable comparison with our model.

The CNN-MT model shown in Fig. 2 unifies and generalizes earlier CNN models [1–3] of single-task learning to include multiple related tasks such as a regression task for estimating perception indicators and a classification task for driving decisions. Please note the difference between learning tasks in Fig. 2 and driving tasks in Table 1. The generalized model is based on a minimization of different loss functions for different tasks (see below). It allows to use existing tasks or to add new tasks and hence can be used to systematically investigate different models in both training and testing phases for a clear comparison. Our results show that the proposed model performs better than earlier models in inference efficiency and driving stability.

In summary, the contributions of the present work include (i) a deep learning model for multiple learning and driving tasks using various CNNs to predict affordance indicators from camera images, (ii) control algorithms for achieving zero collisions with few indicators from CNNs and radar data from self-driving car, (iii) comprehensive comparisons of different CNNs on the performance metrics of their dynamic inference accuracy and computatinal complexity and efficiency, and (iv) an optimal (novel) CNN-MT model for autonomous driving based on the present and previous studies of the direct perception paradigm.

2 Related Work

Chen et al. [1] proposed 14 indicators (heading angle, 5 distances to preceding cars, 7 distances to lane markings in a three-lane highway, and a Boolean "fast" that is not optimized by the gradient descent method) in two coordinate (in lane and on lane) systems. They only used camera images as inputs to AlexNet [23] modified for autonomous driving (denoted by AlexNet+14 herein). The output estimated indicators from AlexNet+14 are then used in a controller that drives Host in a TORCS traffic with (up to 12) Agents that use true indicators. They also proposed a controller logic for driving Host and Agents. They have collected about 450,000 images from 12 hours of human driving in TORCS on 7 different tracks. The maximum speed of Host in their end-to-end simulation is 72 km/h (62 mph).

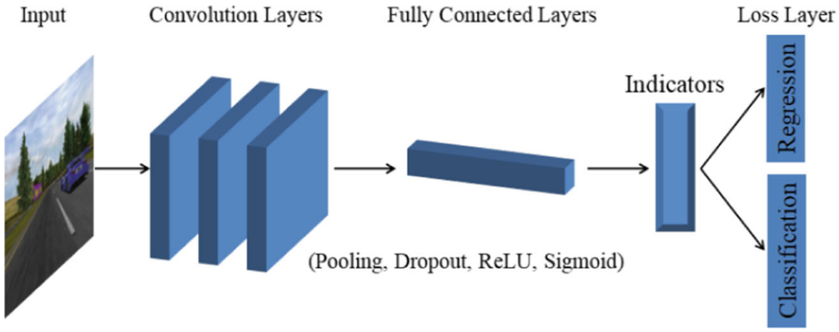

Fig. 2 A CNN multi-task learning model that combines a regression task for learning perception indicators and a classification task for driving decisions

Based on DeepDriving, Al-Qizwini et al. [2] proposed 5 indicators (heading angle and 4 distances to lane markings), where the 5 distance indicators to preceding cars in [1] are removed and two coordinate systems are reduced to one. In addition to cameras, they used other sensory devices in Host (like lidar and long and short range radars in real cars) to replace these 5 indicators and thus provide the controller more accurate measures of surrounding Agents. They have compared GoogLeNet [30], VGGNet [31], and Clarifai [32] and shown that GoogLeNet performs the best for the root mean squared error (RMSE) of their 5 indicators in training phase. They have also compared GoogLeNet to AlexNet+ with the original 14 and their 5 indicators and shown that RMSEs of these three models are comparable in between 0.01 and 0.02 with GoogLeNet slightly better. The controller has been modified to detect Agents within 60 meters from Host using its sensors and to allow the speed of three Agents larger than that of Host. They collected 510,112 images from 14 hours of a label-collecting agent but did not publish the data and code.

Sauer et al. [3] generalized the direct perception approach to include high-level driving commands such as "turn left at the next intersection" provided by sensor devices in advanced navigation systems [33]. They proposed 6 affordance indicators, namely, heading angle, distance to the vehicle ahead, distance to lane centerline, red light, speed sign, and hazard stop to deal with complex urban environments. The loss function is defined as the sum of the mean absolute error of the first three indicators and the cross entropy of the last three. They have developed a controller that decouples longitudinal control (throttle, brake) using the car-following model in [1] with a proportional–integral–derivative controller and lateral control (steering) using the Stanley controller. The maximum speed of Host in their simulation using CARLA [3] with a front facing camera is 20 km/h in single-lane traffic in driving direction.

3 Control and CNN Algorithms

We use a front-facing camera and a few short-range radars to design CNN and control algorithms and show that radars can improve driving stability defined by the damage model in TORCS, where the damage number is a measure of an agent colliding with other agents or road obstacles [22].

Depicted in Fig. 2, the CNN-MT learning model

$$\min_{\tilde{\mathbf{y}},\tilde{\mathbf{p}}} \left\{ \frac{\lambda_R}{2M} \sum_{m=1}^{M} \|\mathbf{y}-\tilde{\mathbf{y}}\|_2^2 - \frac{\lambda_C}{M} \sum_{m=1}^{M} \sum_{k=1}^{2} p_k \log(\tilde{p}_k) \right\} \qquad (1)$$

combines a regression (R) task for learning I perception indicators in the output vector $\tilde{\mathbf{y}} = (\tilde{y}_1, \cdots, \tilde{y}_I)$ and a binary classification (C) task for learning the probability $\tilde{\mathbf{p}} = (\tilde{p}_1, \tilde{p}_2)$ of a one-hot (truth) vector $\mathbf{p} = (p_1, p_2)$ to make a decision for overtaking $\tilde{p}_1 > 0.5$ or not $\tilde{p}_2 > 0.5$, for example. The one-hot vector $\mathbf{p}$ can be defined by a safety distance D of Host from Agents ahead determined by radars for overtaking or by other factors for other decisions. As an example, we define $p_1 = 1$ and $p_2 = 0$ for overtaking if $D > 10$ m. The probability $\tilde{\mathbf{p}}$ of D is learned solely from the images of the front-facing camera. Therefore, these two learning tasks are highly correlated. Here, λ_R and λ_C are weighting factors of their corresponding loss functions, M is a batch size of input images, $\mathbf{y}$ is the ground truth vector of I perception indicators, $\mathbf{p}$ is dynamically determined by TORCS, and $\tilde{p}_k$ are outputs of the softmax function.

3.1 Affordance indicators and control algorithms

We use three types of indicators, namely, Host's heading angle (Angle), its distance to the road centerline (toMiddle), and its distance to the direct preceding car in a certain lane i (Di). For example, there are $I = 5$ indicators, i.e., Angle, toMiddle, D1, D2, and D3 on a three-lane road as shown in Fig. 3, for the regression learning task defined in (1). The probability indicator $\tilde{\mathbf{p}}$ of the safety distance D in (1) replaces three distance indicators D1, 2, and 3 for the classification learning task.

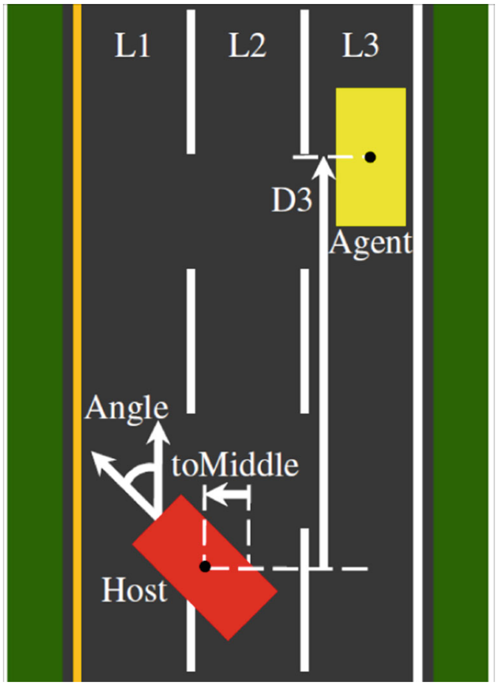

Fig. 3 Affordance indicators for a regression learning task of autonomous driving: Angle (Host's heading angle), toMiddle (Host's distance to the road central line), D1 (Host's distance to the direct preceding Agent in Lane 1 (L1)), D2, and D3 on a three-lane highway. A probability indicator $\tilde{\mathbf{p}}$ replaces three distance indicators D1, 2, and 3 for a classification learning task (see text)

The indicator toMiddle is a critical value for all cars in TORCS to steer and drive on the track [21]. It is used in [3] but not in [1, 2]. The indicators D1, D2, and D3 are different from the five indicators in Figs. 3c and 3e in [1], since we use only one coordinate system as in [2] instead of two.

Controllers are crucial in normal and race autonomous driving [1–3, 21, 22, 34–36]. We change TORCS controllers, which imitate human drivers in racing with sensor information [34–36], for highway driving based on these indicators. Our controllers result from a series of simulation tests targeting at zero damages in TORCS. We investigate the controllers in [1] and find collisions taking place due to a lack of using the sensor information of neighboring Agents in Host.

In particular, we find that the following state information of Agents in Host provided by the sensors of TORCS [21] is essential for our learning algorithms to achieve zero damages in our simulation test.

Agent_State0: There are no Agents in a range of 60 meters.

Agent_State1: A slower Agent is directly in front of Host within the range and with overtaking distance for Host.

Agent_State2: A slower Agent is directly in front of Host without overtaking room for Host. This state concerns front-rear collisions.

Agent_State3: An Agent is very close to Host in the lateral direction. This state concerns lateral collisions.

TORCS provides not only sophisticated physical and 3D graphical engines but also many sensors (*angle*, *speed*, *opponents*, *damage* etc.) and effectors (*steer*, *accel*, *brake* etc.) [34] for model developers to design a variety of normal driving and racing controllers in simulated self-driving traffic [1, 34–36]. These sensors and effectors are customized to simulate corresponding electronic or mechanical devices in state-of-the-art vehicles [37].

We integrate the indicators Angle, toMiddle, D1, D2, and D3 (whose estimated and true values are used by Host and Agents, respectively), the effectors *steer*, *accel*, and *brake* (whose values are determined by our and TORCS algorithms for Host and Agents, respectively), and the sensor *opponents* (for calculating the *Agent_State0* to *Agent_State3* of Agents) in Algorithms A1-A8 in Appendix to design controllers that yield zero damage for Host as well as all 20 Agents in testing phase.

For example, we change the original STEER algorithm of TORCS to Algorithm A1 that returns a value of the effector *steer* using the input values of the five indicators by calling the procedure GETOFFSET in Algorithm A2. Algorithm A2 in turn calls Algorithm A8 for the values of *Agent_State0* to *Agent_State3* and determines whether Host should overtake or stay in the current lane and slow down. Algorithm A1 changes the input Angle according to the values of toMiddle, *offset*, *lane_width* (4 m used here), and *road_width* (13 m). Namely, Host steers left first to overtake if toMiddle $>$ *lane_width* or right secondly if toMiddle $<$ $-$*lane_width*.

The lateral and longitudinal control Algorithms A1-3 and A4-7, respectively, are constantly calling the Angent state Algorithm A8 to avoid collisions in a dynamic traffic. Fig. 4 shows that the controllers in [1] incur both longitudinal (top row) and lateral (bottom) collisions without integrating Host's perception indicators and sensory information of surrounding Agents.

3.2 CNN algorithms

AlexNet+14 in [1] modifies the original AlexNet [23] by switching pooling and local response normalization layers and adding one more fully-connected (FC) layer so that the last four FC layers have 4096, 4096, 256, and 14 units. AlexNet+14 yields estimated values (real numbers) of 14 affordance indicators from an input image. The estimated values are then measured in the mean absolute error (MAE) with the ground truth values provided by TORCS. MAEs are reduced by the stochastic gradient descent optimizer in training phase.

We retain 14 output neurons for 5 indicators in order to compare our CNNs and controllers with those in [1], where the absolute error of the indicator toMiddle is weighted 9 times that of the other four. In our experience, toMiddle is more important to learn than the others as its better values yield better Host dynamics in the driving stability of following, curving, and overtaking. The errors of the indicators can be corrected by short-range TORCS sensors to achieve zero damages. The maximum speeds of Agents and Host are 72 and 74 km/h, respectively.

The authors in [30] proposed nine inception modules for GoogLeNet. Each module consists of 1x1, 3x3, and 5x5 convolution kernels and one 1x1 projection layer, which can capture map features at different scales and reduce dimensions and thus remove computational bottlenecks effectively. Moreover, GoogLeNet has a global average pooling after the last convolution layer, which averages out the channel values across the convolutional feature map and hence reduces the total number of parameters drastically. In general, the parameters and memory of a pre-trained GoogLeNet are ∼6 million and ∼20 MB, respectively, compared to ∼60 million and ∼200 MB for AlexNet [23]. The authors in [2] used the original GoogLeNet.

We modify GoogLeNet (called GoogLeNet+) by adding one FC layer (having 128 units) before each of three sigmoid layers that yield two auxiliary (Loss0, Loss1) and one main (Loss2) Euclidean loss layers (see, e.g., Appendix A in [2] for GoogLeNet architecture). These three FC-Sigmoid-Loss structures of GoogLeNet+ are thus (i) FC (4096) → FC (4096) → FC (128) → Sigmoid (5) → Loss0,

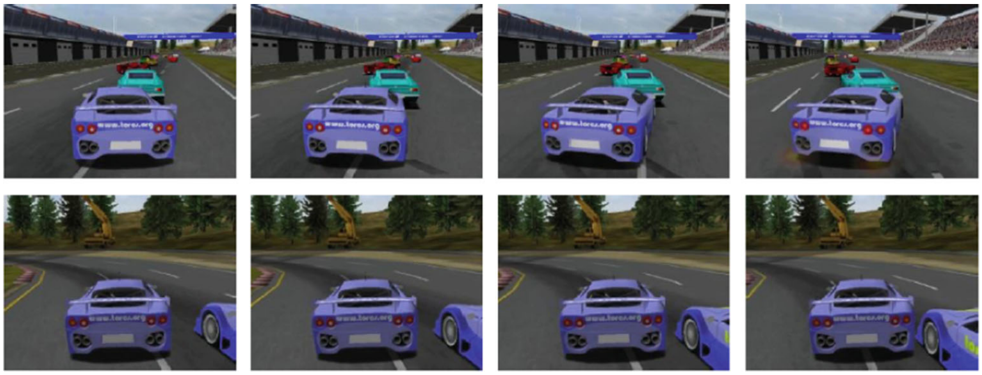

Fig. 4 Two sequences of images showing frontal (top row) and lateral (bottom) collisions between Host and Agent using the controllers in [1]

(ii) FC (4096) → FC (4096) → FC (128) → Sigmoid (5) → Loss1, and (iii) FC (4096) → FC (128) → Sigmoid (5) → Loss2. The added FC (with 128 units) layers reflect the regression task of GoogLeNet+MT to learn 5 (or less) perception indicators in Fig. 2 instead of the classification task of the original GoogLeNet.

Table 2 summarizes the differences between our and earlier models. The learning task RC (regression and classification) between ours and that in [3] is different since the driving tasks in Table 1 and the driving environment are different. The perception indicators are also different, where 3R refers to go straight, turn left, and turn right and 3C to Host's two distances to the vehicle ahead and the road centerline in a single lane and Host's angle to the road tangent [3].

To avoid collisions, the original control algorithms in a simulator need to be changed according to learning tasks, speed, and indicators as denoted by CARLA+ and TORCS+ with the + sign in Table 2. The main differences between ours and those in [1, 2] are RC- vs R-task learning, TORCS+ vs TORCS (unchanged), and different indicators as discussed above. Since our model in (1) is more general than those in [1, 2], we can implement different CNNs (AlexNet+14 in [1] and our AlexNet+5, GoogLeNet5, AlexNet+MT, and GoogLeNet+MT), quantify their performance, and fairly compare them in the same driving scenario and conditions.

Table 2 Comparison between related models in learning task (R for regression, C for classification), speed (in km/h), CNN, simulator, and perception indicators, where the + sign denotes modifications of the respective CNN and controller

Ref.	Task	Speed	CNN	Simulator	Indicators
[1]	R	72	AlexNet+	TORCS	14R
[2]	R	72	AlexNet+ GoogLeNet	TORCS	5R
[3]	RC	20	VGG16	CARLA+	3R3C
Ours	R	74	AlexNet+5 GoogLeNet5 GoogLeNet+5	TORCS+	5R
	RC	74	AlexNet+MT GoogLeNet+MT		2R1C

The difference between CNN5 (with 5R in Table 2) and CNNMT (2R1C) in our model is that CNN5 learns all 5 indicators in Fig. 3 using the regression loss function only ($\lambda_R = 1$ and $\lambda_C = 0$ in (1)) whereas CNNMT learns two regression indicators Angle and toMiddle (2R) and a binary classification (1C) probability vector $\widetilde{\mathbf{p}}$ in (1) with $\lambda_R = 10$ and $\lambda_C = 1$. Since CNNMT does not learn the distance indicators D1, D2, and D3, the number of output neurons is three (2R+1C), which is the smallest number in Table 2, i.e., CNNMT is most efficient. Since D1, D2, and D3 are not learned, we replace them by the ground-truth distances from the sensors in the control Algorithm A2.

4 Experimental setup

A key issue in supervised machine learning is data collection and labeling. For simulation-based autonomous driving research, data comes from a host car by a human driver or a robotic (AI) agent. An AI agent can be thought as a perfect human-like driver [33-35]. We used an AI agent to collect ~500,000 labeled images on seven different tracks for training as shown in Fig. 6 in [1].

There are three driving scenarios for collecting our data. First, a host agent drives on seven empty tracks in a zigzag manner for training the principal indicators Angle and toMiddle. Second, the host follows closely another very slow and zigzagging AI car in front. Third, the host drives normally on tracks with other AI cars (up to 20). The host drives on each track multiple times to collect data. To obtain different traffic images, AI agents are programmed with various driving behaviors.

We use a different track in testing phase to assess the pre-trained CNN agent with the ground truth data collected by itself. The test data contains about 3000 images. For the assessment, we use MAE of the indicators predicted by CNN algorithms on this static data (sMAE). MAE can also be computed dynamically during driving (dMAE). However, dMAE is larger than sMAE due to asynchronous frequencies between CNN computing (complexity, computer, and speed dependent) and TORCS image generation (computer and speed dependent). CNN and TORCS frequencies are >15 and >30 Hz, respectively, on our computer.

Effectiveness is a critical issue for machine learning models in their deployments to real-time applications such as self-driving cars. Our goal here is to study and compare various integrated models of CNNs, perception indicators, learning and driving tasks, and control algorithms, from which we can select an optimal model. In addition to the accuracy metrics dMAE, we further perform experiments to compare these models based on the performance metrics of model's size (the total number of weights and biases), computational complexity (the total number of multiply-and-accumulate operations per image), and inference speed (frames per second) [39].

5 Results

We train GoogLeNet5 and GoogLeNet+5 using a fine-tuning technique [38] to adapt the self-driving problem. We first train the scratch network by stochastic gradient descent with the batch size bs = 32, the momentum m = 0.9, and the learning rate starting from lr = 0.01 and decreasing by a factor of 0.96 every 32000 iterations. The training process stops after 50k iterations as shown in the inset in Fig. 5. Since the pre-trained network captures general features in its early layers [38], we only fine tune the last FC layers in the re-training process, i.e., all previously trained weights are used as initial guesses except that of the last FC layers set to zero. The loss curve in Fig. 5 shows that error fluctuations (i.e., spiky peaks in the inset) are effectively reduced by fine tuning and errors decrease sharply within 1k iterations. The total number of iterations for GoogLeNet by this two-step training is only 70k to reach sMAE ≈ 0.01 compared to 300k in [2] with comparable errors. Each point in all loss curves is an average of 11 consecutive errors.

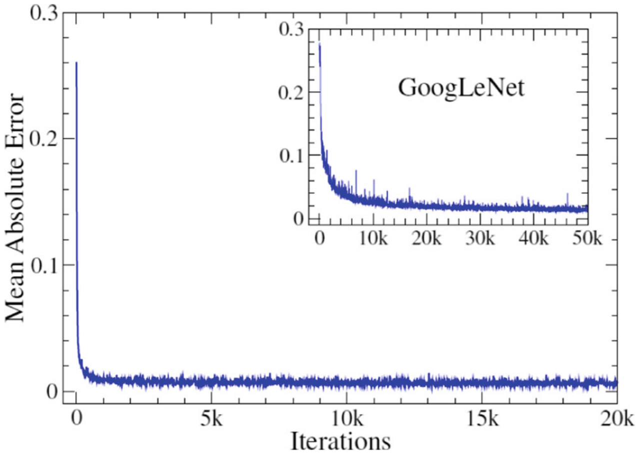

Fig. 5 Mean absolute errors of the five indicators determined by GoogLeNet5 in two-step training first from scratch (in the inset) with 50k and then from fine-tuning with 20k iterations

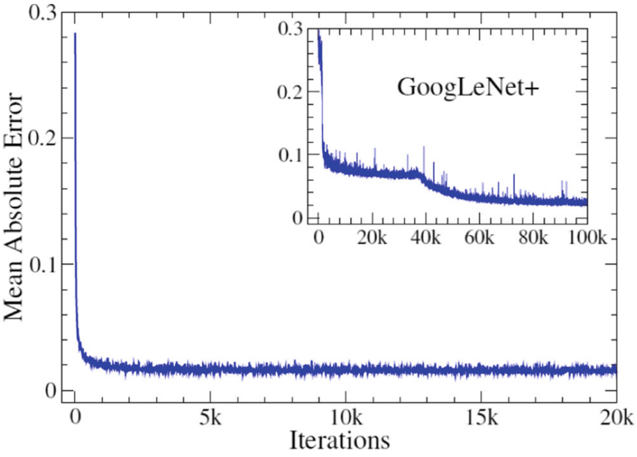

Fig. 6 Mean absolute errors of the five indicators determined by GoogLeNet+5 in two-step training first from scratch (in the inset) with 100k and then from fine-tuning with 20k iterations

In the fine-tuning process, the values of bs and m are the same as those of the pre-trained network except lr. The starting learning rate for fine-tuning FC layers is lr = 0.01, while lr = 0.001 for other layers. In this way, the new added FC layer learns ten times faster than that of the pre-trained network.

The training losses of GoogLeNet+5 and AlexNet+5 are shown in Figs. 6 and 7 having sMAE ≈ 0.02 with 120k and 200k iterations, respectively. The hyperparameters of AlexNet+5 are the same as those in [1] except m = 0.5.

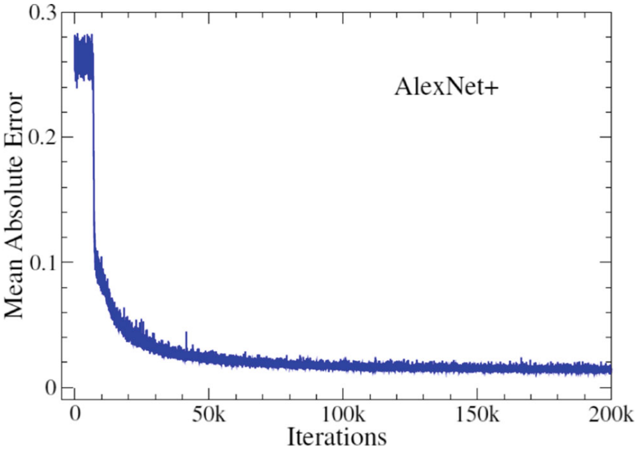

Fig. 7 Mean absolute errors of the five indicators determined by AlexNet+5 in one-step training with 200k iterations

Table 3 Static mean absolute errors in testing phase

Indicator	AlexNet+5	GoogLeNet5	GoogLeNet+5	Alex/GoogLeNet+MT
Angle	0.034	0.041	0.029	0.038/0.025
toMiddle	0.539	0.389	0.347	0.310/0.315
D1 ($\widetilde{\mathbf{p}}$)	6.864	5.190	6.055	(0.007/0.008)
D2	7.048	3.227	3.155	
D3	8.388	5.905	5.450	

GoogLeNet5 is the best among these three CNNs in training loss. However, GoogLeNet+5 yields better Angle and toMiddle in testing phase as shown in Table 3, where Angle is in radians and others are in meters. Angle and toMiddle are principal indicators for Host driving stably around curves, in overtaking, to avoid collissions, and in following as shown in Algorithms A1, A2, A3, and A4, respectively. Larger errors in D1, D2, and D3 are corrected by the sensor information of Agents in Host as shown in Algorithms A1, A2, and A8.

Table 3 also shows that two multi-task CNNs, namely AlexNet+MT and GoogLeNet+MT, perform better than three single-task CNNs in terms of the common two indicators Angle and toMiddle, where the errors 0.007/0.008 of the predicted probability vector $\widetilde{\mathbf{p}}$ are absolute errors, i.e., $\|\widetilde{\mathbf{p}} - \mathbf{p}\|_\infty$. The profiles of training loss (not shown) for Alex/GoogLeNet+MT are similar to those in Figs. 6 and 7. GoogLeNet+MT is overall the best among five CNNs in this table.

Table 4 presents the dMAEs of AlexNet+14, AlexNet+5, GoogLeNet+5, AlexNet+MT, and GoogLeNet+MT, where distLL = D1, distMM = D2, distRR = D3, and toMiddle is a combined indicator for the two road marking (distL and distR) and seven lane marking (denoted by toMarkX) indicators in [1]. AlexNet+14 is better than AlexNet+5 since AlexNet+14 uses two coordinate systems that are however more complicated in designing robust and stable control algorithms to avoid collisions, especially without using sensory information. GoogLeNet+5 is better than AlexNet+5 in D1, D2, and D3, i.e., in depth perception but worse in lateral perception (toMiddle).

Table 4 Dynamic mean absolute errors in testing phase

AlexNet+14		Alex/GoogLeNet+5		Alex/GoogLeNet+MT	
Indicator	dMAE	Indicator	dMAE	Indicator	dMAE
Angle	0.035	Angle	0.043/0.045	Angle	0.041/0.032
distLL	7.970	D1	8.315/7.566	toMiddle	0.340/0.336
distMM	6.188	D2	9.233/6.758	$\widetilde{\mathbf{p}}$	0.012/0.012
distRR	8.540	D3	10.198/8.374		
distL	2.870	toMiddle	0.397/0.464		
distR	2.822				
toMarkL	0.319				
toMarkM	0.374				
toMarkR	0.314				
toMarkLL	0.291				
toMarkRR	0.252				
toMarkML	0.257				
toMarkMR	0.261				

The errors of depth perception indicators predicted by these CNNs are in general quite large as shown in the table. By contrast, AlexNet+MT and GoogLeNet+MT are better than previous three CNNs using only the lateral perception (regression) indicators (Angle, toMiddle) and the overtaking (classification) indicator $\widetilde{\mathbf{p}}$. The indicators D1, D2, and D3 are not explicitly but implicitly (via the safety distance $D > 10$ m) learned by AlexNet+MT and GoogLeNet+MT. Again, GoogLeNet+MT is overall the best in Table 4.

Table 5 shows that the control Algorithms A1 to A8 can yield zero collisions for all 5 CNNs using either 5 indicators (5R in Table 2) or 2-regression and 1-classification indicators (2R1C) after one loop on the test track, whereas the original controller in [1] incurs 413 damage points.

Finally, we summarize in Table 6 the performance of four CNNs used in our model framework of (1). The metrics for the comparison between different CNNs consists of the total number of weights and biases (Weights) that determines the memory requirement of a CNN, the total number of multiply-and-accumulate operations per image (MACs) that determines the computational complexity of the CNN, the dynamic mean absolute error of the indicator Angle (Error) predicted by the CNN, and the total number of frames per second (FPS) processed by the CNN in testing phase. Here, MACs are obtained from the benchmark of MACs for the original AlexNet and GoogLeNet in Table II in [39], i.e., $0.85 = 0.724 \times (71.55/61)$G (giga) for AlexNet+5, for example.

Table 5 Damage points in testing phase

CNN	Damage
AlexNet+14	413
Alex/GoogLeNet+5, GoogLeNet5, Alex/GoogLeNet+MT	0

Table 6 Performance of proposed CNNs in the metrics of Weights (the total number of weights and biases in million (M)), MACs (multiply-and-accumulate operations in giga (G) per image), Error (dMAE of Angle), and FPS (frames per second)

Metrics	AlexNet+5	GoogLeNet+5	AlexNet+MT	GoogLeNet+MT
Weights	71.55M	6.11M	73.65M	6.36M
MACs	0.85G	1.25G	0.87G	1.3G
Error	0.043	0.045	0.041	0.032
FPS	20	15	20	18

The prediction accuracy (Error) and speed (FPS) of driving tasks by a CNN are vital factors for its deployment into self-driving cars. Based on our intensive studies shown in this section, we conclude that GoogLeNet+MT performs the best among 8 CNNs in Table 2.

6 Conclusion

We have proposed an end-to-end deep learning model for self-driving cars by integrating multi-task (MT) learning, convolutional neural networks (CNNs), and control algorithms based on the direct perception approach to autonomous driving. The CNN-MT model can be used to evaluate the inference performance of different CNNs using different sets of perception indicators for different learning tasks (regression or classification) to avoid collisions and make driving decisions in real-time highway traffic. The control algorithms use the perception indicators inferred by a CNN from images of the front-facing camera of a car with its sensory information of surrounding cars. The performance metrics consists of the parameter size and computational complexity of the CNN, the dynamic errors of the indicators predicted by the CNN, and the total number of frames per second processed by the CNN.

We have modified two well-established CNNs, namely AlexNet+ and GoogLeNet+, proposed a multi-task (regression (R) and classification (C)) learning formula and two sets (5R and 2R1C) of few indicators (5 or 2 indicators for perception and 1 indicator for driving decision), and compared our CNNs, control algorithms, and indicators with those in earlier work. We have comprehensively studied AlexNet+14R, AlextNet+5R, AlextNet+2R1C, GoogLeNet5R, GoogLeNet+5R, and GoogLeNet+2R1C in terms of the accuracy in both training and testing phases and the inference performance in testing phase. GoogLeNet+2R1C performs the best among these CNNs on the performance metrics. Here, 2R denotes two lateral indicators Angle and toMiddle for the regression task and 1C denotes one longitudinal indicator for the classification task to make a probable decision for overtaking using implicitly the other three longitudinal indicators (distances to preceding cars in three lanes). The CNN does not explicitly learn these three indicators in training phase but learns 1C with a preset safe distance to the preceding cars.

Compared with earlier work, our model is novel as it provides a general framework to systematically, comprehensively, and quantitatively compare and select an optimal CNN among different CNNs using different indicators for different learning and driving tasks in different types of traffics.

Acknowledgments We thank the reviewers for valuable comments to improve the paper.

Appendix : Control algorithms

Algorithm A1 Calculating steer value

```
procedure STEER(Angle, toMiddle)
    offset = GETOFFSET(D1, D2, D3); //Algorithm A2
    Angle −= (toMiddle − offset)/road_width;
    if (toMiddle > lane_width) then
        Angle −= (toMiddle − lane_width)/road_width;
    else if (toMiddle < − lane_width) then
        Angle −= (toMiddle + lane_width)/road_width;
    else Angle −= toMiddle/road_width;
    steer = Angle/steer_lock; //steer_lock = 0.366 rad
    steer = FILTERS(steer); //Algorithm A3
    return steer;
end procedure
```

Algorithm A2 Calculating offset for overtaking

```
procedure GETOFFSET(D1, D2, D3)
    AGENTSTATE(); //Algorithm A8
    if (Agent_State == 1) then
        if (toMiddle > 1.5) then
            offset −= parms; //Agent is near or in left lane
        else if (toMiddle < − 1.5) then
            offset += parms; //Agent is near or in right lane
        else
            if (D2 < 10) then //middle lane is occupied
                if (D1 > 10) then
                    offset += parms;
                else if (D1 < 10 && D3 > 10) then
                    offset −= parms;
            else offset goes slowly to zero;
    else offset goes slowly to zero;
    return offset;
end procedure
```

Algorithm A3 Steering filter for collision avoidance

```
procedure FILTERS(steer)
   AGENTSTATE(); //Algorithm A8
   if (Agent_State == 3) then
      if (Agent is near) then
         diff_yaw = agent_yaw − host_yaw;
         //agent_yaw given by TORCS
         psteer = diff_yaw/steer_lock;
         steer = parm1 × steer + parm2 × psteer;
   return steer;
end procedure
```

Algorithm A4 Calculating acceleration value

```
procedure ACCEL(Angle)
   allowed_speed = ALLOWEDSPEED(Angle);
   //Based on estimated road tangent angle = Angle
   //+ Host's yaw angle. Speed in m/s.
   if (current_speed > allowed_speed) then
      accel = allowed_rpm/current_rpm;
   else accel = 1;
   accel = TCS(accel); //AlgorithmA5
   return accel;
end procedure
```

Algorithm A5 Traction control system

```
procedure TCS(accel)
   slip = driven_wheels_speed − current_speed;
   if (slip > TCSslip) then
      accel −= min(accel,(slip − TCSslip)/TCSrange)
   else //TCSslip = 2m/s
      accel = accel; //TCSrange = 10m/s
   return accel;
end procedure
```

Algorithm A6 Calculating brake value

```
procedure BRAKE()
   if (current_speed > allowed_speed) then
      brake = min(1, current_speed − allowed_speed);
   else
      brake = 0;
   AGENTSTATE(); //Algorithm A8
   if (Agent_State == 2) then brake = 1;
   brake = ABS(brake); //Algorithm A7
   return brake;
end procedure
```

Algorithm A7 Anti-lock braking system

```
procedure ABS(brake)
   if (current_speed > ABSspeed) then
      slip = current_speed − avg_4wheels_speed;
      if (slip > ABSslip) then
         brake −= min(brake,(slip − ABSslip)/ABSrange)
   else //ABSspeed = 3 m/s
      brake = brake; //ABSslip = 2 m/s
   return brake //ABSrange = 5 m/s;
end procedure
```

Algorithm A8 Determining agent state

```
procedure AGENTSTATE()
   check D_exact; //Exact distance between Host and
   //Agent given by TORCS
   Agent_State = 0;
   if (D_exact < 60 && D_exact > − 60) then
      if (D_exact > 4.5) then
         Agent_State = 1;
         if (Host and Agent in same lane) then
            if (needed brake distance > D_exact) then
               Agent_State = 2;
      else if (D_exact < 4.5 && D_exact > −4.5) then
         Agent_State = 3;
   return Agent_State;
end procedure
```

References

1. Chen C, Seff A, Kornhauser A, Xiao J (2015) Deepdriving: Learning affordance for direct perception in autonomous driving. Proceedings of the IEEE International Conference on Computer Vision, pp 2722–2730
2. Al-Qizwini M, Barjasteh I, Al-Qassab H, Radha H (2017) Deep learning algorithm for autonomous driving using GoogLeNet. IEEE Intelligent Vehicles Symposium (IV), pp 89–96
3. Sauer A, Savinov N, Geiger A (2018) Conditional affordance learning for driving in urban environments. Proceedings of the 2nd Conference on Robot Learning PMLR 87:237–252
4. Ullman S (1980) Against direct perception. Behav Brain Sci 3:373–381
5. Leonard J, How J, Koch O (2008) A perception-driven autonomous urban vehicle. Journal of Field Robotics 25:727–774
6. Jazayeri A, Cai H, Zheng JY, Tuceryan M (2011) Vehicle detection and tracking in car video based on motion model. IEEE Trans Intell Transp Syst 12:583–595
7. Geiger A, Lenz P, Stiller C, Urtasun R (2013) Vision meets robotics: the KITTI dataset. Int J Robot Res 32:1231–1237
8. Huval B, Wang T, Mujica F (2015) An empirical evaluation of deep learning on highway driving. arXiv:1504.01716
9. Pomerleau DA (1989) ALVINN: An autonomous land vehicle in a neural network. Advances in Neural Information Processing Systems, pp 305–313

10. Pomerleau DA (1991) Efficient training of artificial neural networks for autonomous navigation. Neural Comput 3:88–97
11. LeCun Y, Muller U, Flepp B (2006) Off-road obstacle avoidance through end-to-end learning. Advances in Neural Information Processing Systems, pp 739–746
12. Bojarski M, Del Testa D, Zhang X (2016) End to end learning for self-driving cars. arXiv:1604.07316
13. Codevilla F, Miiller M, Dosovitskiy A (2018) End-to-end driving via conditional imitation learning. IEEE International Conference on Robotics and Automation, pp 1–9
14. Taylor CJ, Košecká J, Blasi R, Malik J (1999) A comparative study of vision-based lateral control strategies for autonomous highway driving. Int J Robot Res 18:442–453
15. Vahidi A, Eskandarian A (2003) Research advances in intelligent collision avoidance and adaptive cruise control. IEEE Trans Intell Transp Syst 4:143–153
16. Bagloee SA, Tavana M, Asadi M, Oliver T (2016) Autonomous vehicles: Challenges, opportunities, and future implications for transportation policies. Journal of Modern Transportation 24:284–303
17. Hussain R, Zeadally S (2019) Autonomous cars: Research results, issues, and future challenges. IEEE Communications Surveys & Tutorials 21:1275–1313
18. Janai J, Güney F, Behl A, Geiger A (2017) Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. arXiv:1704.05519
19. Grigorescu S, Trasnea B, Cocias T, Macesanu G (2019) A survey of deep learning techniques for autonomous driving. Journal of Field Robotics, early view
20. Dosovitskiy A, Ros G, Koltun V (2017) CARLA:, An open urban driving simulator. Proceedings of the 1st Conference on Robot Learning, PMLR 78:1–16
21. Wymann B, Espié E, Sumner A (2000) TORCS: The open racing car simulator. Software available at http://www.torcs.sourceforge.net
22. Wymann B (2006) TORCS manual installation and robot tutorial. Manuscript, Available from: http://torcs.sourceforge.net/index.php
23. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, pp 1097–1105
24. Evgeniou T, Pontil M (2004) Regularized multi–task learning. Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp 109–117
25. Xu H, Gao Y, Yu F, Darrell T (2017) End-to-end learning of driving models from large-scale video datasets. Proceedings of IEEE conference on computer vision and pattern recognition, pp 2174–2182
26. Eraqi HM, Moustafa MN, Honer J (2017) End-to-end deep learning for steering autonomous vehicles considering temporal dependencies. arXiv:1710.03804
27. Rausch V, Hansen A, Hedrick J (2017) Learning a deep neural net policy for end-to-end control of autonomous vehicles. American Control Conference (ACC), pp 4914–4919
28. Hecker S, Dai D, Van Gool L (2018) End-to-end learning of driving models with surround-view cameras and route planners. Proceedings of the European Conference on Computer Vision, pp 435–453
29. Bechtel MG, McEllhiney E, Yun H (2018) Deeppicar: A low-cost deep neural network-based autonomous car. IEEE Inter. Conf. on Embedded and Real-Time Computing Systems and Applications, pp 11–21
30. Szegedy C, Liu W, Rabinovich A (2015) Going deeper with convolutions. Proc IEEE Conf Comput Vis Pattern Recognit, pp 1–9
31. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556
32. Zeiler M, Fergus R (2014) Visualizing and understanding convolutional networks. European Conference on Computer Vision, Springer, Cham, pp 818–833
33. Boriboonsomsin K, Barth M (2009) Impacts of road grade on fuel consumption and carbon dioxide emissions evidenced by use of advanced navigation systems. Transp Res Rec 2139:21–30
34. Loiacono D, Togelius J, Saez Y (2008) The WCCI 2008 simulated car racing competition. IEEE Symposium on Computational Intelligence and Games, pp 119–126
35. Cardamone L, Loiacono D, Lanzi PL (2009) Learning drivers for TORCS through imitation using supervised methods. IEEE Symposium on Computational Intelligence and Games, pp 148–155
36. Muñoz J, Gutierrez G, Sanchis A (2010) A human-like TORCS controller for the simulated car racing championship. Proceedings of the IEEE Conference on Computational Intelligence and Games, pp 473–480
37. Rosique F, Navarro PJ, Fernández C, Padilla A (2019) A systematic review of perception system and simulators for autonomous vehicles research. Sensors 19(3):648
38. Yosinski J, Clune J, Bengio Y, Lipson H (2014) How transferable are features in deep neural networks? Advances in Neural Information Processing Systems, pp 3320–3328
39. Sze V, Chen YH, Yang TJ, Emer JS (2017) Efficient processing of deep neural networks: a tutorial and survey. Proc IEEE 105:2295–2329

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.