Towards Neuroimaging real-time driving using Convolutional Neural Networks

Development of CNNs for autonomous steering in the TORCS environment

Carlos Fernandez Musoles
Centre of Computational Intelligence (CCI)
DeMontfort University
carlos.fernandez.musoles@gmail.com
Leicester, UK

***Abstract*—The majority of previous attempts to autonomous driving in the TORCS environment involve the use of pre-calculated data such as the exact distance to opponents or the actual position of the car with respect to the center of the track. As humans, we drive based on locally available information that is gathered by our senses, mainly sight. This information is not precomputed and it is left to the agent to make sense of it. This work explores and evaluates the development of autonomous steering using visual-only input in real time driving. Convolutional Neural Networks (CNNs) have been proven to excel in categorical image classification, but little has been done on how they could perform in continuous spaces such as deciding steering values. The results presented here show that CNNs are indeed capable of performing well on such spaces and can be trained from example. Various modifications proved to enhance network accuracy and hence driving performance, such as applying edge detection filters to the input image, using a weighted average method to select network outputs and including unusual situations in the training data, making the neurovisual agent much more robust and capable of higher generalization power.**

Keywords—convolutional neural networks, neuroimage, visual input, autonomous driving, TORCS

I. Introduction

The Open Racing Car Simulator (TORCS) has become an important tool for AI researchers interested in the development of autonomous driving strategies. Previous successful submissions include Finite State modular approaches [1], fuzzy-based controllers [2], coevolution strategies for fine tuning car parameters [3] and feedforward neural networks that learn by imitation [4][5]. Most of the previous approaches use accurate track information —such as the width of the current segment and the distance to track limits—, opponent metrics —such as angle between our car and next driver— and any other data that is available in the TORCS environment. However, as humans we drive fundamentally based on local information available to us via our senses, mainly sight. Efforts to restrict the amount of data provided to drivers are seen in [6], though they still pre-process the information making drivers passive receptors of sensory information. The aim of this project is to investigate the feasibility of an autonomous driver that uses visual-only information to navigate tracks in TORCS.

Strategies that use visual input have been applied in a few scenarios: from neurovisual control of a Quake II bot [7] to an attempt to General Game Play for Atari videogames [8]. In the driving domain, ALVINN [9] is one of the first implementations to use visual information as the main input of a neural network for steering. In the TORCS environment, [10] showed how a recurrent neural network layout could be constructed using evolutionary algorithms to navigate a simple track using visual information. Although showing promise, these strategies are somewhat simplified in either the nature of the input —ALVINN uses distance sensors to aid visual input — or its complexity —simple network topologies. Those simplifications were imposed in no small part due to the high dimensionality nature of the visual input and the computational expense of processing it in real time. Nowadays those limitations are being overcome with the advent of massive parallel computation and the use of GPUs for General Purpose programming (GPGPU).

GPGPU has made possible to implement more complex models in Reinforcement learning, genetic algorithms and most of all neural networks. Deep neural network models have been shown to be ideal to process complex visual data [11], particularly in image classification [12] and pattern recognition [13][14]. Convolutional Neural Networks (CNNs) are one such models; they consist of a series of partially-connected layers with local connectivity and shared weights to allow detecting patterns irrespective of their position in the image. CNNs are naturally good at feature extraction using multiple local maps that arise from the original image. Recently CNNs have been shown to be useful in the field of reinforcement learning. One very promising approach uses CNNs as a replacement for Q-value tables in Q-Learning [15]. [16] shows how deep reinforcement learning —i.e. RL that uses deep networks— has the potential to be used in real time driving when using visual input. Although their general learner system offers extremely good results in a wide variety of games —including TORCS— its generalization power *within* a training session remains unclear; the system is trained and tested on a single scenario — in the case of TORCS, on a particular track— and performance on unseen situations is not shown.

This work explores and evaluates the use of convolutional network models to deal with real-time autonomous driving in TORCS using exclusively visual input to steer a vehicle. The remaining sections are organized as follows: Section II describes the proposed driving model; Section III briefly explains the learning framework, from data gathering to network training; Section IV defines the experimental design employed to evaluate the performance of the multiple iterations of the CNN driver; Section V presents the experimental results from performance tests on various scenarios; the discussion of the results and limitations is carried out in Section VI; and Section VII summarizes the conclusions reached in this work, as well as listing areas of future development.

II. PROPOSED SOLUTION

A. Overview

The system implemented, *Steer-CNN*, is a modified TORCS bot that uses a neural network to control steering. In its current implementation, *Steer-CNN* focuses on predicting steering commands only, and all other driving decisions, such as acceleration, brake and gear change, are made using a hardcoded policy.

Figure 1 shows the decision diagram, run at every frame. The system takes as input the TORCS rendered image from a third person perspective —from the top of the car facing forwards- and passes it to a pre-processing module, which crops it to discard the borders and applies one or more filters to it —see Section V for tests on different filters. The final image is a greyscale 1-byte depth 144x100 structure that acts as the input for the CNN. Each input unit value is scaled —so values from 0 to 255 are transformed to 0 to 1. The CNN outputs a single value that is used as the steering command, which is combined with the remaining driving decisions from the hardcoded policy and returned to the engine.

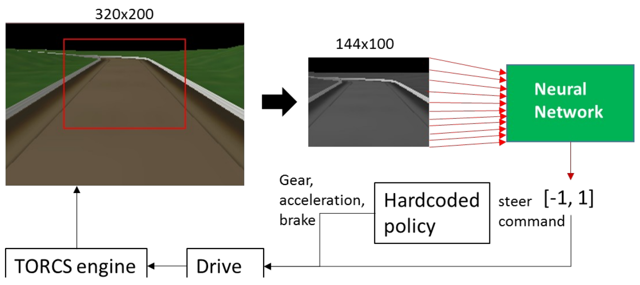

Fig. 1. Steer-CNN driving diagram, from TORCS render to driving decision

B. Convolutional Network Architecture

The CNN is based on the work by [17], the *LeNet-5*, which achieves very good accuracy rates on handwritten recognition tasks. One feature that makes it a good candidate for our problem is its ability to detect patterns irrespective of their position on the image —a common trait of CNNs due to the use of shared weights. The original model has been scaled up —number and size of feature maps— to deal with higher resolution images:

- Convolutional layer (CN) 1: 144x100 inputs, 30 output channels, 10x10 kernels.
- Subsampling layers: down sampling x2, stride 2.
- CN 2: 67x45 inputs, 50 channels, 6x6 kernels.
- Multilayer perceptron (MLP): 2 layers (31000x500) with 100 or 200 output units.

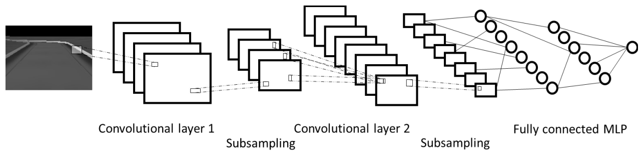

Fig. 2. CNN architecture for Steer-CNN based on LeNet-5, showing two convolutional layers, 2 subsampling layers and a fully connected MLP.

LeNet-5 is a qualitative classification system, i.e. with a categorical output (0 to 9, corresponding to the digit seen). Thus, the last MLP layer contained 10 units; the final decision of the network was chosen based on which unit had a higher activation (winner-takes-all selection). *Steer-CNN* needs to operate on a continuous space: the range of possible steers in TORCS, from -1 to 1. Hence, to be able to use this model, the steering space is discretized into a number of steps. Even though theoretically this reduces the resolution power of the system, in practice this has little impact on driving performance, provided the number of steps is high enough: 100 or more divide the 2 units range interval into steps of less than 1 degree per step, which is a more than acceptable resolution when driving.

III. LEARNING FRAMEWORK

A. Data gathering

The CNN is trained from a hardcoded line-following driving policy. Whilst driving, 10 data samples are gathered per second, recording: 1) *Input image*: the image is stored as an array of pixel values, where each is a 1-byte greyscale value (0-black, 255-white); and 2) *Expected output*: discretized steering value determined by the driving policy.

Three different data sets are generated, two using a hardcoded line-following policy (1 and 2) and one from one of the CNN drivers (3) —see Section V for a discussion on this.

- TRAINING_SET_1: data generated from a single track (dirt6, 9875 samples).
- TRAINING_SET_2 data generated from different tracks (dirt6, dirt4, speedway1 and e-track4, 11672 samples).
- TRAINING_SET_3: data generated from different tracks (dirt6, speedway1 and dirt4, 7482 samples).

The tracks were chosen based on their high variety of race situations and differences in track width, to provide a heterogeneous dataset to the CNN.

B. Network training

The CNN is trained using Stochastic Gradient Descent, a batched algorithm that uses a very small subset of the training data on each iteration to modify the weights and bias of the network. It has been shown to be resilient to overfitting [18]. The batch size used is 32 and training sessions were run for 50000 iterations.

Prior to training, the data is normalised so all pixel information is provided within the range 0 to 1.

IV. Experimental design

This work measures and compares the impact that various design decisions have on the predictive accuracy of the CNN and driving performance. Five different experiments are run to test the following: 1) Pre-processing the image prior to feeding the CNN; 2) scaling up the network outputs; 3) optimal output selection policy; 4) using multiple tracks in the training set. 5) learning from *Steer-CNN* driver.

In all scenarios, *Steer-CNN* driver uses the network architecture described in Section II, unless specifically stated otherwise. The training set used is indicated.

Performance is measured in real-time driving on one lap over *known* and *unknown* tracks -a track is considered *known* when any part of it has been used during training. To add statistical significance, each test is repeated 5 times and the results averaged. The following metrics are calculated:

- Mean Square Error (MSE): Average square distance between expected and actual network output expressed as a percentage of the total steering space; the expected value is the steering command calculated by the hardcoded line-following policy.
- Time stuck: the car is deemed *stuck* when its speed is below 5 km/h and out of the track bounds. Should this happen, the driver has a simple hardcoded recovery policy to put it back in track.
- Damage: resulting from collisions.
- Lap time.

The following tracks were used to test performance: dirt6, speedway, oval e-track5, dirt1, e-road, cg-track2 and street1. For simplicity, when the results are not shown for a particular track it indicates the candidate was not able to complete the track (DNF result).

V. Results

A. Pre-processing the input image

The first experiment measures whether pre-processing the image has an impact on the final driving performance. The rationale behind this test is, in TORCS, different tracks tend to have different colour schemes (dirt races have brown roads, whereas city races have tarmac) and environments (out of boundaries objects, but still visible, such as trees, buildings, etc.); this may impact the generalization power of the network. If the image is pre-processed to abstract certain key elements, generalization may get improved.

Two CNNs are trained and compared, each using a different pre-processing filter: 1*) B&W*: black and white image (no filter); and 2) *Edge*: edge detection filter. Both of them are trained using TRAINING_SET_1. Both networks used 100 steps to discretize the steer space, hence the CNNs had 100 output units. After training, the network error for *B&W* was 1.07 on average (of a total of 100) and 1.0611 for *Edge*.

The results in Table 1 show that edge detection is a good filter as it helps improve generalization. *B&W* is slightly better on the known track on all metrics (dirt6), but it is incapable of finishing an unknown track (speedway1), whereas *Edge* completes it without major problems.

	Dirt6 (known)		**Speedway1 (unknown)**	
	B&W	*Edge*	*B&W*	*Edge*
Damage	888.4	1783.6	DNF	672
Stuck time	3.42785	3.913444	DNF	2.90737
Lap time	168.7346	172.6464	DNF	162.4996
MSE	0.278749	0.296858	DNF	0.39448

Table 1. Results from testing B&W vs Edge drivers

The results also demonstrate that edge information is sufficient to drive around a track. Furthermore, as less data needs to be store, edge detection training data size is reduced (75MB per 85MB for B&W). Edge detection filter is used to process the input image for the remaining experiments.

B. Scaling up outputs

The output space —possible steering values in TORCS— is a continuous range from -1 to 1. As discussed, LeNet-5-type of networks are categorical and hence the space needs to be discretised. To understand the impact of discretising the output space, two CNN models are compared, both with identical topology but varying in the number of output units on the last layer:

1) CNN_100: 100 output units
2) CNN_200: 200 output units

Since the complete steering range is 0.734 radians, a CNN with 100 units splits the space with gaps of 1.32 degrees, setting the steering resolution to 1.32. The CNN with 200 units results in a steering resolution of 0.66 degrees.

CNN_100 is the network resulted from the edge detection experiment. CNN_200 is trained with the same dataset as CNN_100 (TRAINING_SET_1) and achieves an average classification error of 1.2833. Since the total error is now 200, this suggest the network is able to predict steering angles with more precision (error value in percentage of 0.641 vs 1.0611).

Table 2 compares the results of CNN_200 based on those obtained in the previous experiment, evidencing a significant improvement over CNN_100.

	Dirt6	Speedway1
Damage	-1313.4	-54
Stuck time	-2.44602	-0.15845
Lap time	-47.5512	-72.1228
MSE	-0.1181	-0.102934

Table 2. CNN_200 results expressed as the difference from those of CNN_100.

An increased steering resolution –from 100 to 200- leads to an enhancement of driving performance (reduced lap time, stuck time and damage taken) and prediction accuracy (less MSE). The remaining experiments use 200 output units in their CNN architectures.

C. Optimal output selection policy

On a LeNet-5 network, the final output is chosen by a winner takes all policy, i.e. the unit with higher value is selected as the final output. Fig 3 (left) shows an example of how the winner-takes-all selection criterion determines the final output. Although this yields good results in digit image classification —where the differences between classes is clear — it is not clear whether this is the best strategy for a continuous space such as steering.

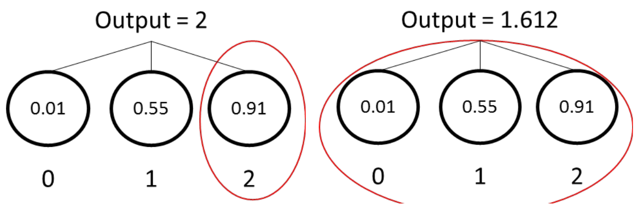

Fig. 3. Lefg: winner-takes-all output selection criterion example with 3 output units firing at different rates and the final output; right: Same example but using the weighted average output selection criterion, yielding a different output value

On a complex and continuous decision space it is difficult to unequivocally assign a steering value to a specific image, and a network could be understood as a way of calculating steering recommendations, using the unit output value as the network's confidence on each steering value. Thus, a winner-takes-all selection criterion would reduce the network's ability to incorporate this. To understand whether the network is internally capable of capturing this subtlety, we compared the classification results of a winner-takes-all selection strategy with a weighted average that uses the values from all output units (and uses the value itself as the weight). Fig 3 (right) shows the selected output on the same example as above but using weighted average.

As we are testing whether the network *already* captures this, no further training is required. Instead, the network error rate for CNN_200 over the training set is recalculated using the weighted average vs winner-takes-all.

	Winner-takes-all	**Weighted average**
Error rate	10.39%	12.83%
Average error	1.2833	0.9096

Table 3. Error rate and average error of the CNN_200 network over the training set using two output selection criteria.

The error rate in Table 3 is calculated based on the difference between expected output (0 to 199) and actual output. For the winner-takes-all, the error rate is simply the times the expected is not equal to the actual output. The weighted average computes a decimal value, and hence a simple comparison would not be accurate; instead it is considered not correct when the difference with the expected value is 0.5 or greater —equivalent to rounding the decimal output to an integer and doing a direct comparison with the expected value. The results indicate that the network achieves better accuracy using the weighted average, with a lower average error (0.9096 vs 1.2833). Although the network reaches a higher error rate (12.83%), the lower average error indicates that those mistakes are only between 0.5 and 1 units from the expected value.

To see whether this improvement translates into better driving, the real time MSE whilst driving is measured in a *Steer-CNN* using each selection criterion.

	MSE (winner-takes-all)	**MSE (weighted average)**
Dirt6	0.357523	0.350811
Speedway	0.994827	0.975517

Table 4. Real-time prediction error measured for both selection criteria.

Results in Table 4 suggest the weighted average is a more accurate selection method.

Networks use weighted average selection criterion for the remaining experiments.

D. *Diversity of training data*

So far all CNNs have been trained using data from a single track (TRAINING_SET_1). It includes a variety of race situations that make the CNN generalize to driving on other track (speedway1). This experiment evaluates whether training with a more diverse dataset improves performance.

Two identical CNNs are compared (image pre-processed with edge detection, 200 output units, using weighted average selection criterion), each trained with a different dataset:

1) CNN_1: Using TRAINING_SET_1 (single track)

2) CNN_2: Using TRAINING_SET_2 (multiple tracks)

After training, both networks achieved similar accuracies, with CNN_1 average error of 0.9096 and CNN_2 average error of 0.835.

Table 5 compares the results of CNN_2 to those of CNN_1 obtained in experiment 2 (single track training). As CNN_2 trained with both Dirt6 and Speedway1, the drivers were tested on Dirt4 (known to CNN_2) and Oval E-track 5 (unknown to both).

	Dirt4	**Oval e-track5**
Damage	-34.8	0
Stuck time	-0.31651	0
Lap time	-5.922	-0.7192
MSE	-0.33332	-0.06628

Table 5. CNN_2 results expressed as the difference from those of CNN_1.

CNN_2 results in better driving and prediction accuracy on Dirt4 —expected, as it trained on that track- and on Oval e-track5, an unknown track. Thus, more variety in the training set not only enhances driving —as more tracks are known— but also seems to improve performance on unknown tracks.

E. *Learning from Steer-CNN driver*

A limitation of using a perfect driver to generate the training data is that the network only learns from good examples. This is useful for the network to know what to do in normal situations, but when driving, as the classification error accumulates, the car could get into unusual situations where the network has to deal with a type of input never seen before. An example of an unusual situation is when the car is driving outside the track. Naturally the performance of the network on those situations is poor –and in fact it accounts for most of the situations where *Steer-CNN* gets stuck. [19] uses a staggered approach, whereby a basic learner is first trained from a hardcoded policy, and then its driving is used to add data points to the training data, which is then used to train the next generation. Here we use a modification of that process, where a new training set is generated in a combined effort, using an acceptable but imperfect driver —trained from a hardcoded policy— to navigate around the track, but recording the expected output from the perfect driver using the line-following policy. This approach creates both normal and unusual situations (as occasionally the driver would get stuck) and records what the expected steering action should be in both.

The following experiment tests whether driving can be improved by using unusual situations in training. TRAINING_SET_3 was generated by letting CNN_2 drive on various tracks and recording the steering value from the hardcoded line-following policy. Table 6 shows the results of the CNN trained with TRAINING_SET_3 as compared with those of CNN_2 from experiment D. The new policy does not get stuck in either of the tracks and the driving performance is considerable better as a result. After including unusual situations in training, now when *Steer-CNN* faces a barrier it is able to react and steer away successfully.

	Dirt6	**Speedway1**
Damage	-1610.4	-62.4
Stuck time	-2.11205	-1.12179
Lap time	-21.5004	-3.8248
MSE	-0.06577	0.066418

Table 6. Results of Steer-CNN trained using unusual situations as compared to a network trained on various tracks from a perfect driver (CNN_2)

But the more remarkable result is the increased power of generalisation, as *Steer-CNN* is now capable of driving on a number of tracks that could not be completed before. CNN_1 and CNN_2, trained with a hardcoded policy, got stuck in different parts of the following tracks: Dirt1, E-Road, CG Track2 and Street1. The new CNN trained with unusual situations easily navigates through those tracks with performance shown in Table 7.

	Dirt1	**E-Road**	**CG Track2**	**Street1**
Damage	1077	44	13	362
Stuck time	2.52231	0	1.09932	0
Lap time	53.528	128.534	109.862	139.544
MSE (winner-takes-all)	1.03679	0.632822	0.62544	0.574434
MSE (weighted average)	0.980776	0.587153	0.60101	0.545807

Table 7. Driving performance of Steer-CNN trained with unusual situations

VI. Discussion

LeNet-5 type of convolutional networks have been proven very robust and accurate in categorical image classification. However, it remained unclear whether this type of CNN could be useful in a continuous space, such as steering values on a -1 to 1 range to control driving. This work has built evidence to confirm that LeNet-5 type of CNNs are indeed applicable as continuous evaluation functions based on visual input.

Several key factors that enhance the CNN performance have been identified, grouped in two categories: quantitative — elements that increase classification accuracy or better driving performance— and qualitative improvement —changes that result in the network doing something it was incapable of doing before.

Using varied training data, weighted average criterion for CNN output selection and better steering resolution by further discretizing the steering space are amongst the quantitative improvements that enhance driving performance and reduce network prediction error.

It is worth noting that the weighted average criterion for aggregating the output value from the network outputs is functionally equivalent to adding a final layer to the network with a single unit and interpret that unit as the steering value. The focus of this work was not on optimizing the network architecture but to evaluate whether a classification CNN such as LeNet-5 was able to operate on continuous spaces, and hence no further layers were added, favoring the output selection methods. However, the author acknowledges incorporating this into the network may yield better results, as the selection criterion would become part of the learning.

Although the increase in performance brought by the quantitative improvements is significant, it is by itself insufficient to build an effective autonomous CNN driver, as *Steer-CNN* was unable to complete many of the TORCS tracks. Pre-processing input image and using unusual situations in the training set fundamentally changed performance. The two qualitative changes considerably enhanced the generalization power of the CNN and allowed *Steer-CNN* not only to perform better in the tracks it could already finish (dirt6, speedway1 and oval e-track5), but also to complete arguably difficult tracks it could not before (dirt1, e-road, CG Track2, street1). The results are even more impressive if the topology of those tracks is taken into account, with very narrow lanes, out of bounds objects and even fake paths.

The single most influential optimization was using unusual situations in the training set, as this allowed *Steer-CNN* to recognize them and know how to correct them. One example of such situations is when facing a wall: as the perfect driver would have not encounter this situation, the CNN would struggle to select an appropriate steering value; training from a previous version of itself, *Steer-CNN* is able to avoid crashing.

Since the scope of the project was to demonstrate the suitability of CNNs in real time driving, the current implementation focuses exclusively on steering using a single image frame. However, this limitation does not allow the driver to consider other relevant information such as current speed, or previous steering actions. As a consequence, occasionally, particularly when attempting to avoid collisions, *Steer-CNN* decisions result in meander-like driving. Sudden and drastic changes in steering without considering speed leads to some spinning. Another design decision was the delegation of non-steering driving actions —such as accelerating, braking and gear change- to a hardcoded policy. Although in practice this allowed us to focus on evaluating navigation, it imposed a limitation on global performance, as comparison with other TORCS bots was not possible in global terms –lap time is greatly influenced by the speed the car traverses the track in.

VII. Conclusion and further work

Steer-CNN results have demonstrated that a visual-only driving policy using CNNs is feasible for real-time decision making in the TORCS environment. In addition, this work has laid out several optimization elements that lead to quantitative and qualitative improvements on driving performance and network accuracy:

- Network generalization is improved when pre-processing the image to abstract relevant features.
- Using weighted average to select CNN output units instead of a winner-takes-all method yields lower prediction error and produces more accurate driving.
- A varied training data set tends to improve performance both in known and unknown tracks.
- Using a suboptimal driver to gather training data in conjunction with a hardcoded policy significantly improves driving performance, allowing the driver to complete difficult tracks that were not possible before.

Even though the results are promising, further work must be carried out to fully realize an autonomous neurovisual driver. The following are areas of potential improvement:

- Incorporate other sensory data to the policy as well as previous steering history and generate CNNs to control other driving decisions –accelerate and brake
- CNN architecture: experiment with other CNN architectures and explore the use of automation tools to optimize CNN hyperparameters and layers.

REFERENCES

[1] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Pérez, “A modular parametric architecture for the TORCS racing engine,” *CIG2009 - 2009 IEEE Symp. Comput. Intell. Games*, pp. 256–262, 2009.

[2] L. Cardamone, P. L. Lanzi, D. Loiacono, and E. Onieva, “Advanced overtaking behaviors for blocking opponents in racing games using a fuzzy architecture,” *Expert Syst. Appl.*, vol. 40, no. 16, pp. 6447–6458, 2013.

[3] L. Cardamone, D. Loiacono, and P. L. Lanzi, “Applying cooperative coevolution to compete in the 2009 TORCS Endurance World Championship,” *2010 IEEE World Congr. Comput. Intell. WCCI 2010 - 2010 IEEE Congr. Evol. Comput. CEC 2010*, 2010.

[4] J. Munoz, G. Gutierrez, and A. Sanchis, “Controller for TORCS created by imitation,” *CIG2009 - 2009 IEEE Symp. Comput. Intell. Games*, pp. 271–278, 2009.

[5] C. Athanasiadis, D. Galanopoulos, and A. Tefas, “Progressive neural network training for the Open Racing Car Simulator,” *2012 IEEE Conf. Comput. Intell. Games, CIG 2012*, pp. 116–123, 2012.

[6] D. Loiacono, L. Cardamone, and P. L. Lanzi, “Simulated Car Racing Championship: Competition Software Manual,” no. April, 2013.

[7] M. Parker and B. D. Bryant, “Neurovisual control in the Quake II Environment,” *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. August, pp. 44–54, 2012.

[8] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A Neuroevolution Approach to General Atari Game Playing,” vol. 6, no. 4, pp. 1–18, 2013.

[9] D. a Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” *Adv. Neural Inf. Process. Syst. 1*, pp. 305–313, 1989.

[10] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, “Evolving large-scale neural networks for vision-based reinforcement learning,” *Proceeding fifteenth Annu. Conf. Genet. Evol. Comput. Conf. - GECCO ’13*, p. 1061, 2013.

[11] C. Nebauer, “Evaluation of convolutional neural networks for visual recognition.,” *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 685–696, 1998.

[12] S. Zhu, Z. Shi, C. Sun, and S. Shen, “Deep neural network based image annotation,” *Pattern Recognit. Lett.*, vol. 65, pp. 103–108, 2015.

[13] K. Zhang, Q. Liu, Y. Wu, and M.-H. Yang, “Robust Visual Tracking via Convolutional Networks,” *arXiv*, vol. 25, no. 4, pp. 1–18, 2015.

[14] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, “HCP: A Flexible CNN Framework for Multi-label Image Classification,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8828, no. XX, pp. 1–1, 2015.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. a Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous Methods for Deep Reinforcement Learning,” *arXiv*, pp. 1–28, 2016.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[18] M. Hardt, B. Recht, and Y. Singer, “Train faster, generalize better: Stability of stochastic gradient descent,” *srXiv:1509.01240*, pp. 1–24, 2015.

[19] S. Ross, G. J. Gordon, and J. A. Bagnell, “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning,” *Aistats*, vol. 15, pp. 627–635, 2011.