Robustifying the Deployment of tinyML Models for Autonomous mini-vehicles

Miguel de Prado *‡ miguel.deprado@he-arch.ch
Manuele Rusci † manuele.rusci@unibo.it
Romain Donze * romain.donze@he-arc.ch
Alessandro Capotondi § alessandro.capotondi@unimore.it
Serge Monnerat * Serge.Monnerat@he-arc.ch
Luca Benini †‡ lbenini@iis.ee.ethz.ch
Nuria Pazos * Nuria.Pazos@he-arc.ch

* He-Arc Ingenierie, HES-SO †DEI, University of Bologna ‡Integrated System Lab, ETH Zurich §UNIMORE

2021 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-7281-9201-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ISCAS51556.2021.9401154

***Abstract*—Standard-size autonomous navigation vehicles have rapidly improved thanks to the breakthroughs of deep learning. However, scaling autonomous driving to low-power systems deployed on dynamic environments poses several challenges that prevent their adoption. To address them, we propose a closed-loop learning flow for autonomous driving mini-vehicles that includes the target environment in-the-loop. We leverage a family of compact and high-throughput tinyCNNs to control the mini-vehicle, which learn in the target environment by imitating a computer vision algorithm, i.e., the expert. Thus, the tinyCNNs, having only access to an on-board fast-rate linear camera, gain robustness to lighting conditions and improve over time. Further, we leverage GAP8, a parallel ultra-low-power RISC-V SoC, to meet the inference requirements. When running the family of CNNs, our GAP8's solution outperforms any other implementation on the STM32L4 and NXP k64f (Cortex-M4), reducing the latency by over 13x and the energy consummation by 92%.**

***Index Terms*—Autonomous driving, tinyML, robustness, and, micro-controllers**

I. Introduction

Autonomous driving has made giant strides since the advent of deep learning (DL). However, scaling this technology to micro- and nano- vehicles poses severe challenges in terms of functionality and robustness due to their limited computational and memory resources of the on-board processing unit [1], [2].

Micro-Controller Units (MCUs) are typically chosen on small unmanned vehicles to balance the mW power cost of the sensing front-end and keep the system-energy low to extend the battery life. Traditionally, driving decisions have been off-loaded and carried out remotely, implying energy-expensive, long-latency, and unreliable transmissions of raw data to cloud servers [3]. Off-system transfers can be prevented by processing data on-board and directly driving the motor controllers. Thus, tiny machine learning (TinyML) has appeared as a new field to address these challenges and tackle on-device sensor data analysis at hardware, algorithmic and, software level [4], [5].

A major challenge for the design of autonomous driving decision models is that the real-world environment change over time: the data distribution that has been initially learned might not match the underlying distribution of the current environment, e.g., the car driving through different landscapes or weather conditions. Hence, there is an increasing need to adapt to ever-changing environments to make vehicles more robust and efficient over time.

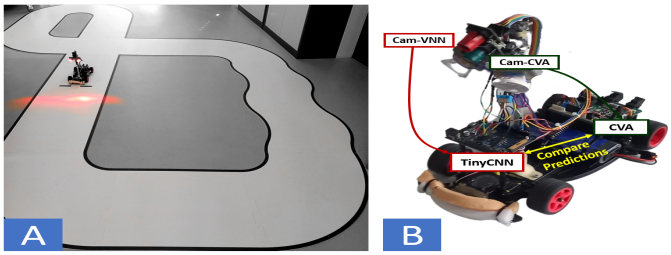

Fig. 1: **Automotive application use-case.** A) Mini-vehicle running on circuit track. B) Close-loop learning task. One linear camera feeds the GAP8 (tinyCNN) and another the NXP k64f (CVA).

A. Goal specification: Robust low-power Autonomous Driving

We aim to shed light in the robustification of DL methods for autonomous systems deployed on dynamic environments. We specifically focus on enabling the deployment of tinyCNNs to a low-power autonomous driving vehicle. We base our design on the NXP cup framework [6], an autonomous racing competition that offers a solid ground to test new ideas that can be reproduced, ported to other autonomous devices, e.g., drones, or scale up to bigger vehicles. Our vehicle consists of a battery-powered mini-car that needs to detect autonomously 7 different states in a controlled circuit: *GoStraight, TurnLeft, TurnRight, CrossingStreets, StartSpeedLimit and StopSpeedLimit* as shown in Fig. 1. The vehicle contains a linear camera and an on-board NXP k64f MCU [7] to detect the current state, compute the required action and drive the actuators accordingly.

The process was originally based on a conventional and handcrafted computer-vision algorithm (named as CVA) that predicts accurately only under stable light conditions. The fragility to light condition is due to the nature of the CVA as it calculates derivatives on the input image which requires good contrast. The lack of robustness to light conditions is countered by setting the camera with a variable acquisition time that adapts to the lighting conditions (controlled by a PID) to obtain clear images. However, the variability and length of the time ($\geq$2 ms) limit the vehicle's agility. Thus, we take a tinyML approach and aim at replacing the conventional CVA by a tinyCNN to: *i)* improve the robustness to lighting variations, and *ii)* increase the performance, i.e., actions/sec, by learning challenging features from short and constant acquisition times.

978-1-7281-9201-7/21/$31.00 ©2021 IEEE

Authorized licensed use limited to: UNIVERSITA MODENA. Downloaded on June 22,2022 at 09:19:39 UTC from IEEE Xplore. Restrictions apply.

B. Contributions

We present an end-to-end flow of data, algorithms, and deployment tools that facilitates the deployment and enhance the robustness of a family of tinyCNNs for an autonomous low-power mini-vehicle. Thus, our contributions are the following:

1) We introduce a closed-loop learning flow that enables the tinyCNNs to learn through demonstration. By imitating an expert with access to good-quality images, the tinyCNNs gain robustness to lighting conditions while having only access to a fast-rate camera, thereby doubling the system's throughput.

2) We leverage GAP8 [8], a parallel ultra-low-power RISC-V SoC, to meet the CNN inference computing requirements by adding it as a System-on-Module (SoM) on the NXP platform.

Further, we compare our deployment solution on GAP8 (50 MHz) with two Cortex-M: an STM32L4 (80 MHz) and an NXP k64f (120 MHz). We present the Pareto-optimal front where our solution dominates all other implementations, reducing the latency by over 13x and the energy consumption by 92%.

II. Related work

We find 3 main topics related to this work in the literature:

1) High-Performance Autonomous Driving: There exist multiple CNN approaches for autonomous driving [9], ranging from standard-size to small-scale vehicles. On the higher end, Nvidia and Tesla introduced PiloNet [10] and AutoPilot [11], requiring dedicated platforms such as TESLA FSD chip and NVIDIA drive, which provide TOPs of computing power and tens of GBytes of memory for their large CNN solutions. Other approaches such as DeepRacer [12], F1/10 [13], or DonkeyCar [14] require GOPs and hundreds of MBytes that cover using platforms like Nvidia Jetson, Raspberry PI or Intel Atom. By constrast, we focus on end-to-end CNN solutions suitable for very low-power vehicles with MCUs featuring MOPs and up to a few MBytes, which is an unexplored field.

2) Learning Methodology: Imitation Learning (IL) leverages the idea of a student learning from an expert that gives directives through demonstration [15], [16]. In this context, ALVINN [17] proposed a CNN-based system that learns to infer the steering angle from images taken from a camera on-board. Similarly, PilotNet [10] and J-Net [18] employed a system that collects the driver's signal to label a training dataset with on-board cameras. However, these approaches only use the expert to label the training datasets. Instead, we propose a closed-loop learning flow where the learner confronts the expert in real drive and gradually improves through demonstration. In this direction, Pan et al. proposed [19] where they optimize the policy (online) of an reinforcement learning agent that imitates an expert driver with access to costly resources, while the agent has only access to economic sensors. However, their approach is not compatible with our use case where low-power systems cannot hold such level of computation and memory.

3) Low-power DL deployment: Multiple software stacks have been introduced on inference tasks for resource-constrained MCUs. In this context, STMicroelectronics has released X-CUBE-AI to generate optimized code for low-end STM32 MCUs [20]. Similarly, ARM has provided the CMSIS-NN [21], which targets Cortex-M processors and provides a complete backend for quantized DL networks exploiting 2x16-bit vector MACs instructions [22], [23]. Recently, the CMSIS-NN dataflow has been ported to a parallel low-power architecture, PULP, originating PULP-NN [24], which exploits 4x8-bit SIMD MAC instructions on a parallel processor, such as the GAP8. In this work, we leverage the GAP8 for the autonomous driving use case and provide a quantitative energy-latency-quality comparison of these solutions.

	LeNet5	**VNN4**	**VNN3**	**VNN2**	**VNN1**
# Parameters (K)	72,85	6.04	0.97	1.29	0.48
Complexity(KMAC)	181.25	163.41	28.69	5.82	7.5

TABLE I: **Vehicle Neural Network (VNN) family**

III. Robust navigation with tinyML

We aim to take a tinyML approach and replace the initial CVA solution by a tinyCNN. First, we evaluate an initial setup and verify the challenge of porting DL methods on MCUs.

A. Initial Evaluation and Challenges

1) Data collection: We manually collect and label 3 initial datasets, each containing around 1000 samples per class (driving action) for the training set and 300 for the test set. The first dataset, *Dset-2.0*, contains samples with a fixed acquisition time of 2.0 ms - clear enough images - where the CVA can still predict well the required action. On the other hand, the second and third datasets, *Dset-1.5* and *Dset-1.0*, hold more challenging samples (low contrast) with 1.5 ms and 1.0 ms acquisition times where the CVA fails to predict well, and thus, we aim to use a CNN instead.

2) Training: We choose LeNet5 [25] for our initial evaluation as it is small and well-known CNN architecture, which is also used in [18]. We use PyTorch as training environment with cross-entropy loss function, SGD optimizer, data augmentation and dropout to avoid over-fitting. Thus, we obtain an accuracy of 99.53% on the *Dset-2.0* test set, but only 84.12% and 81.27% on the more challenging *Dset-1.5* and *Dset-1.0* test sets.

3) Deployment: We employ CMSIS-NN (Int8) as a backend to execute LeNet5 on the NXP k64f. The execution time turns out to be 5.4 ms, far too long compared to the $\approx$2 ms achieved by the conventional CVA on the same platform and conditions.

Given LeNet5's fragility to lighting conditions and its long latency, the initial CNN setup provides no benefit compared to the original CVA. To address these challenges, first, we create a family of tinyCNNs for efficient MCU deployment. Next, we introduce the GAP8 as a SoM to accelerate the CNN inference and finally, we detail the closed-loop learning methodology.

B. Vehicle Neural Network (VNN) Family

We modify LeNet5's topology by varying the number of convolutional (conv) layers and the stride to shrink the model size, the number of operations, and the latency. To have a higher tolerance to the diffusion of features in low-light conditions, we opt for a relatively large kernel size (k=5) for both the conv. and the pooling layers, having the latter a stride of 3 to

Authorized licensed use limited to: UNIVERSITA MODENA. Downloaded on June 22,2022 at 09:19:39 UTC from IEEE Xplore. Restrictions apply.

reduce the number of activations. As a result, we have created a family of tinyCNN called Vehicle Neural Networks (VNNs), see Table I, containing a range of layers that span from 1 to 3 convs. followed by one fully-connected layer for the final classification. Networks and datasets have been open-sourced[1]

C. System-on-Module (SoM) setup

We leverage GAP8 [8], a parallel ultra-low-power RISC-V SoC, to meet the CNN's inference computing requirements (we describe GAP8 and efficient deployment in Section IV). We add GAP8 as a SoM on the NXP platform and set up the system with two synchronized cameras (calibrated to have the same view of the circuit), one feeding the NXP and another feeding the GAP8, see Fig. 1. We use the GAP8 as a CNN accelerator, which is only in charge of inferring the VNN on the input image, while the NXP controls all the other sensors and actuators. The results from the VNN are constantly transferred via UART from the GAP8 to the NXP to control the actuators.

D. Closed-loop Learning System

Initially, our family of VNNs achieves an accuracy of 78%-83% on *Dset-1.0*, a drop of 15%-20% compared to the VNNs trained on the *Dset-2.0* setup. Thus, we need a learning strategy to enhance the VNNs' robustness on the low-contrast images from the *Dset-1.0* setup. Hence, we propose a closed-loop learning system as a way to gradually improve the quality of our datasets and boost the VNNs' accuracy.

We implement this technique by collecting valuable data from the sensors at runtime, training (offline) the model from scratch on the cumulative set of data, and pushing back the updates to the deployed VNN, see Fig. 2. Thus, we pursue two main objectives: *i)* improve the robustness to lighting variations, and *ii)* increase the performance, i.e., actions/sec, by learning challenging features from shorter acquisition times.

We have experimentally tested[2] that the CVA accurately predicts provided adequate light conditions. We can assume the predictions of the CVA as ground-truth and follow an imitation learning (IL) approach where we use the CVA as a teacher to help the VNNs learn better features. Hence, we decouple the original system and propose a setup with two cameras:

- Cam-CVA: set with a variable (and long) acquisition time that adapts to the lighting conditions (controlled by a PID) and always provides clear images. This camera feeds the CVA running on the NXP.
- Cam-VNN: set with a short and constant acquisition time that captures the lighting variability of the environment. This camera feeds the VNN on the GAP8.

By confronting the results of both algorithms while the mini-vehicle runs in a lighting-changing environment, we can improve the generalization capacity of the deployed VNN. Moreover, we can also leverage IL to increase the VNNs' accuracy on the more challenging *Dset-1.5* and *Dset-1.0* setups, which, in turn, improves the system's throughput (actions/sec). We carry out the IL methodology at runtime and collect those input images that lead VNN's predictions to differ from those of the CVA. The samples get automatically labeled by the CVA and are sent over to the PC. We include the new samples in the training set to reinforce those classes where the VNN has failed. The learning procedure can be repeated multiple times. However, we find that the frequency of new sample discovery decreases over time as the deployed solution progressively improves. Thus, we show an experiment with several phases.

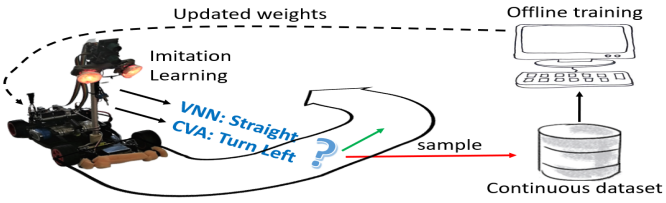

Fig. 2: **Closed-loop learning pipeline.** End-to-end closed-loop learning cyclic methodology via Imitation Learning.

E. Experimental setup and Results

We demonstrate the closed-loop learning methodology by first training a VNN on *Dset-1.0*, the most challenging setup. Next, we combine our three *Dsets: 2.0, 1.5, and 1.0*, and train the VNNs on it to make them have access to richer data distribution. Then, we deploy a VNN on the mini-vehicle and set *Cam-VNN*'s acquisition time to *1 ms*. We make the vehicle run in a varying-light environment while the VNN and CVA results are confronted. Those samples where the VNN failed are collected at every stage and merged to *Dset-1.0* training set to conform an incremental dataset that we show in 3 phases: *+25%*, *+50%* and *+100%* (new samples wrt. the original *Dset-1.0*). In the last stage, i.e., *+100%*, we heavily alter the light conditions of the environment to improve the robustness of the VNN. We train the VNN from scratch at each phase for 1000 epochs on the complete set and send an update of the weights to the vehicle before starting the next phase. We perform each training three times to account for the variability in the random initialization.

Fig. 3-A displays the results obtained on the *Dset-1.0* test set. Initially, our family of VNNs achieves an accuracy of 78%-83%. After training the VNNs on the combined dataset (*Dset-All*), most of the VNNs are able to generalize better, and their accuracy noticeably improves due to the richer diversity of light conditions. Further, when leveraging the closed-loop learning methodology through IL and training the networks on the reinforced dataset, VNN3, VNN4, and LeNet5 reach a top accuracy of 94.1%, 97.4%, and 98.3%. By contrast, VNN2's accuracy remains mostly constant while VNN1 decays at the end, probably due to their shallow topology and lower capacity, failing to learn from more challenging data. Overall, the closed-loop learning approach allows an increase in accuracy of over 15% in *Dset-1.0*, matching the accuracy of the CVA on *Dset-2.0* while doubling the throughput of the system.

Fig. 3-B illustrates how VNN4 learns and forgets features. We train VNN4 on the data from *Dset-1.0* setup, i.e., same as in Fig. 3-A, and compare VNN4's accuracy on *Dset-1.0* and *Dset-2.0* test sets, which contain samples with different acquisition

[1] *https://github.com/praesc/Robust-navigation-with-TinyML*

[2] Not possible to benchmark CVA statically on our *Dsets* as it uses previous samples to predict the current one, i.e., it works on a continuous data stream.

Authorized licensed use limited to: UNIVERSITA MODENA. Downloaded on June 22,2022 at 09:19:39 UTC from IEEE Xplore. Restrictions apply.

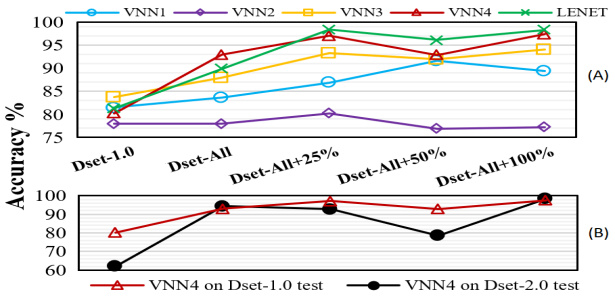

Fig. 3: **Closed-loop learning evaluation.** X-axis is shared. A) VNNs accuracy on *Dset-1.0* test (8-bit). B) VNN4 trained on *Dset-1.0* learns/forgets features from *Dset-2.0*.

time and therefore, lighting conditions. We can observe that, at first, when the VNN4 has only been trained on *Dset-1.0* initial's data, the accuracy on *Dset-2.0* is poor. Nonetheless, when VNN4 receives data from *Dset-All*, it generalizes better and performs well in both test sets. However, as the experiment goes on and VNN4 only sees data from the *Dset-1.0* setup, it tends to forget and slowly decreases its accuracy on *Dset-2.0* test set. Finally, by heavily altering the lighting conditions on the *100%* point, VNN4 can rehearse with data similar to those of *Dset-2.0* and "remembers". This experiment depicts a challenge of continuous learning. It slightly shows the effects of catastrophic forgetting [26] and how rehearsal techniques can tackle this issue [27], which we will address in the future.

IV. MCU Efficient Deployment

We first describe the compression of the VNNs and GAP8's capabilities to achieve an efficient deployment. Then, we offer a comparison between different MCUs when running our VNNs.

A. Network Compression

Training of CNNs is normally carried out using floating-point data types. Such types need specific arithmetic units which may not even be present in MCUs due to their large area and costly energy consumption. Quantization is a compression method that reduces the storage cost of a variable by employing reduced-numerical precision. In addition, low-precision data types can improve the arithmetic intensity of the CNNs by leveraging the instruction-level parallelism. Thus, we have quantized our VNN models to fixed-point 8-bit to reduce memory footprint and power consumption [28], [29]. We have employed post-training quantization where the weights can be directly quantized while the activations require a validation set (sampled from circuit runs) to determine their dynamic range. Thus, we have observed a low accuracy loss (<3%) on the initial *Dsets* and negligible loss (<1%) after the closed-loop learning phases.

B. MCU Hardware/Software Inference Platforms

We have leveraged the GAP8 [8], which is based on the PULP architecture. It includes a RISC-V core, acting as an MCU, and an 8-core RISC-V (cluster) accelerator featuring a DSP-extended ISA that includes SIMD vector instructions such as 4x8-bit Multiply and Accumulate (MAC) operations. Besides, the cluster is equipped with a zero-latency 64kB L1 Tightly Coupled Data Memory and an 512 kB L2 memory.

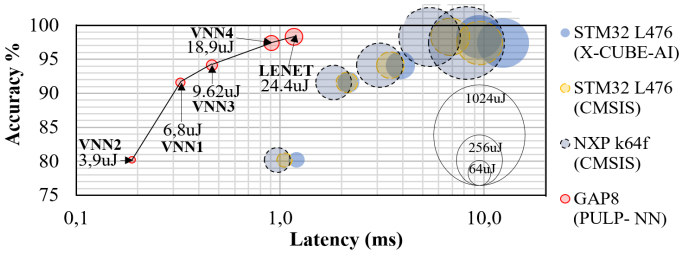

Fig. 4: **Accuracy-Latency-Energy Trade-off.** Accuracy (w.r.t. *Dset-1.0*, y-axis), latency (x-axis), and energy-consumption (balloon area) for an inference of the VNNs. Black line highlights Pareto front.

We compare the GAP8 (at 50 MHz) with two other classes of MCUs: a *low-power single-core* MCU (STM32 L476 at 80 MHz [30]), and a *high-performance single-core* MCU (NXP k64f at 120 MHz [7]). Besides, we also compare different inference backends supported on these devices. We report the GAP8 platform coupled with the PULP-NN [24], against the STM32 supporting either X-Cube-AI [20] or CMSIS-NN [21] and, the NXP coupled with the CMSIS-NN [21].

C. Energy-Accuracy-Latency Trade-off

Thanks to the high accuracy obtained through the closed-loop learning methodology, we can now employ the camera with a short-acquisition time ($1\ ms$) and set it as our latency target for the classification task. Fig. 4 summarizes the accuracy, latency, and energy measured on the different MCUs. All VNNs deployed on GAP8 meet the $1\ ms$ target and establish the Accuracy-Latency Pareto frontier, dominating all the other implementations on STM32 L476 and NXP k64f. Remarkably, only VNN2 performs under $1ms$ on the NXP k64f ($0.97ms$) due to its shallow topology and the higher clock frequency of the NXP. Interestingly, we observe that X-Cube-AI backend is up to 27.8% slower than CMISIS-NN on the STM32 L476.

Looking at the energy consumption of a single classification, VNN4 on GAP8 (8 cores, 50 MHz) consumes $18.9\mu J$, -65.2% less compared to VNN2 on NXP k64f (120MHz) while being over 20% more accurate. The same VNN2 on GAP8 consumes $3.9\mu J$, -92.8% less than the NXP k64f. The usage of STM32 L476 (80MHz) is only possible if executing VNN2 and relaxing the target latency up to $1.5ms$. Yet, it consumes the same amount of energy as VNN4 on GAP8, while the latter provides -13.5% latency and +21.4% accuracy.

V. Conclusion and Future Work

We have shed light on the robustification of tinyCNN models deployed on a low-power driving use case. We have introduced a closed-loop learning methodology that enables the tinyCNNs to imitate an expert, improving their robustness to lighting conditions in the target deployment scenario. Further, we have proposed a parallel ultra-low-power platform to meet the latency specifications, which consume as little as $3.9\mu J$ for a single inference. we envision that this methodology can be applied for other autonomous use cases, e.g., drones, to improve the robustness and efficiency of tinyML methods. As future work, we aim to perform the training of the CNNs on-chip towards a continuous learning scenario.

Authorized licensed use limited to: UNIVERSITA MODENA. Downloaded on June 22,2022 at 09:19:39 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini, "A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones," *IEEE Internet of Things Journal*, 2019.

[2] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018.

[3] O. Mutlu, "Processing data where it makes sense in modern computing systems: Enabling in-memory computation," in *2018 7th Mediterranean Conference on Embedded Computing (MECO)*. IEEE, 2018, pp. 8–9.

[4] "Tinyml," 2020. [Online]. Available: https://www.tinyml.org/summit/

[5] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv preprint arXiv:2003.04821*, 2020.

[6] "Nxpcup," 2020. [Online]. Available: https://nxpcup.nxp.com/

[7] "NXP K64F," 2019. [Online]. Available: https://www.nxp.com

[8] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2018, pp. 1–4.

[9] S. Kuutti, R. Bowden, Y. Jin, P. Barber, and S. Fallah, "A survey of deep learning applications to autonomous vehicle control," *arXiv preprint arXiv:1912.10773*, 2019.

[10] M. Bojarski, P. Yeres, A. Choromanska, K. Choromanski, B. Firner, L. Jackel, and U. Muller, "Explaining how a deep neural network trained with end-to-end learning steers a car," *arXiv preprint arXiv:1704.07911*.

[11] "Auto Pilot," 2019. [Online]. Available: https://www.tesla.com/autopilot

[12] "DeepRacer." [Online]. Available: https://aws.amazon.com/deepracer/

[13] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*.

[14] "DonkeyCar." [Online]. Available: github.com/autorope/donkeycar

[15] "Introduction to Imitation Learning," 2019. [Online]. Available: https://blog.statsbot.co/introduction-to-imitation-learning-32334c3b1e7a

[16] "ICML 2018: Imitation Learning Tutorial," 2018. [Online]. Available: https://sites.google.com/view/icml2018-imitation-learning/

[17] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," in *Advances in neural information processing systems*, 1989.

[18] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," *Sensors*, vol. 19, no. 9, p. 2064, 2019.

[19] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *Robotics: science and systems*, 2018.

[20] STMicroelectronics, "X-CUBE-AI," https://www.st.com/en/embedded-software/x-cube-ai.html, accessed: 2019-09-12.

[21] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[22] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[23] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual wake words dataset," *arXiv preprint arXiv:1906.05721*, 2019.

[24] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-nn: Accelerating quantized neural networks on parallel ultra-low-power risc-v processors," *arXiv preprint arXiv:1908.11263*, 2019.

[25] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, p. 5, 2015.

[26] "Continual Learning," 2018. [Online]. Available: https://medium.com/@culurciello/continual-learning-da7995c24bca

[27] V. Lomonaco, "Continual learning with deep architectures," Ph.D. dissertation, alma, 2019.

[28] M. Rusci, A. Capotondi, F. Conti, and L. Benini, "Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus?" in *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2018, pp. 1–2.

[29] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *arXiv preprint arXiv:1905.13082*, 2019.

[30] "STMicroelectronics STM32L476xx," 2019. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l476je.pdf

Authorized licensed use limited to: UNIVERSITA MODENA. Downloaded on June 22,2022 at 09:19:39 UTC from IEEE Xplore. Restrictions apply.