Towards Efficient On-Board Deployment of DNNs on Intelligent Autonomous Systems

Alexandros Kouris
Dept. of Electrical & Electronic Engineering
Imperial College London, UK
a.kouris16@imperial.ac.uk

Stylianos I. Venieris
Samsung AI Center, Cambridge
s.venieris@samsung.com

Christos-Savvas Bouganis
Dept. of Electrical & Electronic Engineering
Imperial College London, UK
christos-savvas.bouganis@imperial.ac.uk

***Abstract*—With their unprecedented performance in major AI tasks, deep neural networks (DNNs) have emerged as a primary building block in modern autonomous systems. Intelligent systems such as drones, mobile robots and driverless cars largely base their perception, planning and application-specific tasks on DNN models. Nevertheless, due to the nature of these applications, such systems require on-board local processing in order to retain their autonomy and meet latency and throughput constraints. In this respect, the large computational and memory demands of DNN workloads pose a significant barrier on their deployment on the resource- and power-constrained compute platforms that are available on-board. This paper presents an overview of recent methods and hardware architectures that address the system-level challenges of modern DNN-enabled autonomous systems at both the algorithmic and hardware design level. Spanning from latency-driven approximate computing techniques to high-throughput mixed-precision cascaded classifiers, the presented set of works paves the way for the on-board deployment of sophisticated DNN models on robots and autonomous systems.**

I. Introduction

Deep neural networks (DNNs) are emerging as the dominant model for numerous machine vision tasks. Their widely demonstrated state-of-the-art accuracy has led to their deployment as a core component in a broad range of real-world applications. At the forefront of this emergence, intelligent autonomous systems have successfully nudged in various applications of the industrial landscape, such as Amazon's robot-run warehouse and Samsung's Retail Bot, and actively lead the development of research for novel applications, such as Uber's driverless vehicle, Google X's autonomous delivery drone and Samsung's healthcare robot (Samsung Bot Care). However, being constantly challenged by tasks of increasing complexity, there is a continuing need to extract even higher accuracy out of new models. To push the accuracy performance of DNNs, larger and more complex networks have been introduced that offer higher learning capacity at the expense of increased compute and memory requirements.

These excessive computational demands introduce challenges when building intelligent autonomous systems. Such systems typically consist of remote/mobile agents (such as robot platforms) with limited computational capabilities and strict power and/or payload constraints. This challenge is amplified by firm low-latency requirements that are imposed for mission-critical decision making, as in the case of autonomous navigation (Fig. 1a), where the agent interacts with real-world unstructured environments and needs to rapidly respond in external events (*e.g.* to avoid collisions). Additionally, the high rate of information perceived from the agent's sensors introduce high-throughput requirements in order to absorb this data to the maximum possible extend (Fig. 1b), broadening effectively the agent's perceptual capabilities.

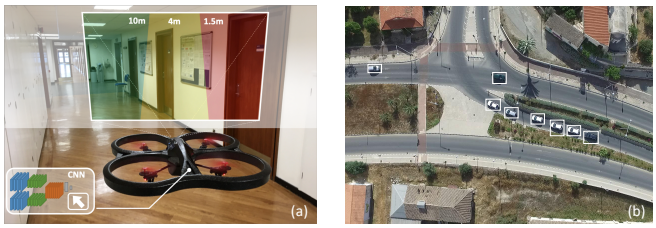

Fig. 1. Deep learning-powered UAV applications: (a): Autonomous Navigation [2] (b): Vehicle Detection [3]

To remedy this situation, powerful embedded processors have emerged, spanning from mobile CPUs to embedded GPUs and full-custom neural accelerators. Nevertheless, the inefficiencies of programmable architectures on one hand and the fixed functionality and high cost of custom chips on the other, set a limit to their suitability for on-board processing on autonomous systems. In this respect, a promising alternative platform that strikes a balance between the high performance of specialised hardware and the flexibility of processors is reconfigurable hardware in the form of FPGAs [1].

In this context, the challenge of on-board deployment of deep learning models is effectively reduced to the problem of establishing an efficient mapping of DNN inference on embedded FPGA platforms that: *a)* reduces the response time to meet the safety requirements for making mission-critical decisions, *b)* enables the near-sensor processing of high-resolution images obtained by the agent's on-board high frame-rate camera and *c)* complies with the low-power constraints of the agent.

Overall, the future intelligent autonomous systems are dependent on innovations: *a)* at the algorithmic level, leading to further advancements in the achieved task-level accuracy of machine vision models, and *b)* in the establishment of advanced development tools, with the ability to provide efficient mappings of state-of-the-art models to FPGAs (or other embedded processing platforms), considering different optimisation objectives such as latency and throughput. As illustrated

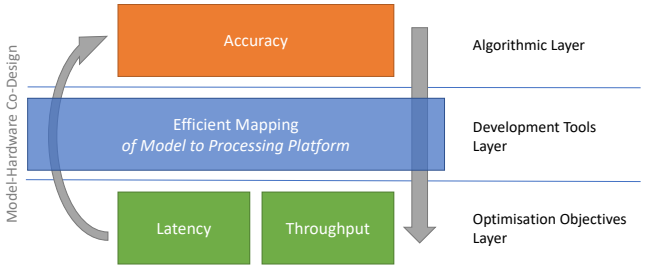

Fig. 2. Intelligent autonomous system development stack.

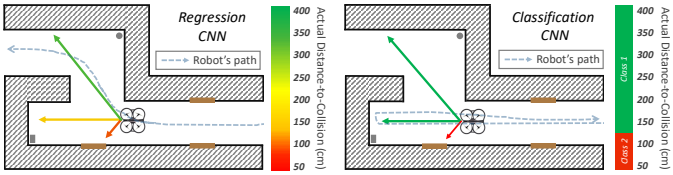

Fig. 3. The proposed Regression CNN [2] provides distance-to-collision predictions of finer granularity compared to relevant classification-based approaches from the literature. A representative example of how the motion planner utilises this additional information to make more informed action decisions on the task of autonomous navigation is depicted here.

in Fig. 2, these development tools can either establish a (faithful or approximate) forward mapping of a given model on a targeted processing platform, or ultimately subsist as part of a closed-loop model-hardware co-design process that co-optimises the system's accuracy and performance by iteratively adjusting both the model itself and its underlying hardware implementation.

Towards this direction, in our recent work, we have tackled a number of critical problems at all levels of the development stack (Fig. 2) to enable the efficient deployment of DNNs on intelligent autonomous systems via the use of FPGAs. The rest of the paper presents a high-level view of our work, broadly including: 1) novel DNN architecture design for autonomous drones at the algorithmic level [2], as well as compute-aware DNN design co-optimising accuracy and performance on the targeted processing platform [3] (discussed in Sec. II); 2) toolflows for the automated generation of latency-optimised (faithful and approximate) DNN inference accelerators for latency-critical tasks, including both single- [4, 5] and multi-DNN [6] scenarios (discussed in Sec. III); and 3) the exploitation of the resilience of CNNs to reduced-precision arithmetic to boost the inference throughput [7, 8] (discussed in Sec. IV).

II. Deep Learning-powered Drones

Over the past decade, the broad commercial availability of inexpensive micro aerial vehicles (MAVs), such as camera-equipped quadcopters, has promoted UAVs to an emerging remote sensing platform. At the same time, the advancements of deep learning models in machine vision empowered UAVs with increased autonomy that extended their applicability on real-world tasks, including infrastructure inspection, and search and rescue operations.

Towards Autonomous UAV Navigation. A task of major importance in UAV missions is autonomous navigation. To achieve robust navigational autonomy, real-time obstacle detection (and ultimately avoidance) is required to handle unmapped or dynamic objects and structures in the agent's environment. Recent literature, residing in the algorithmic layer of Fig.'s 2 stack, has introduced DNN models that demonstrate state-of-the-art performance in UAV navigation in unseen environments, with enhanced generalisation capabilities [9]. In this direction, we have recently proposed a novel two-stream CNN architecture [2] that predicts the distance-to-collision towards multiple directions, by extracting spatio-temporal features from the UAV's on-board forward-looking camera video stream (Fig. 1a). These values are fed to a novel local motion planner, that controls the UAV's yaw and linear velocity, to achieve collision-free navigation.

Our approach handles the task of autonomous navigation as a multistage process, and can be used in a standalone setting to navigate in unseen environments (exploration), or in combination with a global motion planner standing in for the collision avoidance system.

Compared to other state-of-the-art approaches that employ a coarser classification task between navigable and non-navigable space [10], the introduced regression CNN offers more information-carrying predictions during perception, enabling that way the motion planner to make more insightful decisions on the autonomous navigation task (Fig. 3).

However, the efficient deployment of such computationally demanding models on mobile robot platforms poses formidable challenges, especially in the case of UAVs that are restricted by limited payload capabilities and a narrow power envelope. Moreover, in many applications, low-latency requirements (*e.g.* for mission-critical decisions) as well as deployment in remote areas, predetermine the establishment of a wireless link between the remote agent and a base server station to be prohibitive (in latency terms) or even impossible.

To deal with these challenges, a lot of attention has been given on the deployment of deep learning models in embedded processing systems (near-sensor processing). One conventional direction comprises optimising the deployment of existing models using i) custom hardware accelerators and ii) approximation techniques. Examples of such approaches, recently developed by our group are summarised in Sec. III.

Hardware-Aware DNN Design. At the same time, domain-specific model-system co-design forms an alternative direction that is gaining increasing attention in the literature. Along these lines, in our recent work [3], we conducted a Neural Architecture Search (NAS) on the task of vehicle detection from UAV imagery (Fig. 1b), exploring the trade-off between performance and accuracy by tuning the CNN's architecture (in terms of number and size of filters per layer) and the resolution of the input image. Employing this compute-aware design methodology, the resulting model provides the highest accuracy or best performance, among other architectures all complying to the same resource budget by efficiently utilising the computational resources of the target platform.

III. Latency-Critical Decision Making

To achieve high throughput, a common practice in resource-constrained systems involves the offloading of computation to powerful cloud-based servers. However, the tight response-time requirements posed by autonomous machines dictate placing latency at the forefront. Safety-critical decision-making tasks such as speed and steering angle control require a very short loop between perceiving the environment (*e.g.* scene recognition, object detection) and acting (*e.g.* steering angle adjustment). To this end, performing inference locally is an emerging alternative approach that is capable of alleviating the time and battery overhead of cloud computing. By mapping all computations on-device, the latency and power overhead of exchanging data with the cloud together with the strict requirements of constant Internet connectivity are removed with a substantially reduced end-to-end latency.

Currently, there is a wide availability of commodity hardware platforms with powerful compute capabilities that lie within the power and form-factor budget of autonomous systems. The candidate processing systems span from the 10s of watts of the latest embedded GPUs (*e.g.* NVIDIA Jetson TX1, TX2 and Xavier) and the 5-watt profile of commodity mobile processors (*e.g.* Qualcomm Snapdragon 845/855 and Samsung Exynos 9810 SoCs) down to the sub-watt performance of neural accelerators (*e.g.* the Intel Neural Compute Stick (NCS) 2 with a TDP of around 1 W).

Despite their advantages, each platform exhibits a number of limitations that challenge their on-board adoption as the main inference engine. On the one side of the spectrum, programmable architectures, such as the mobile CPUs and GPUs, offer flexibility in terms of functionality and often are accompanied by a mature development toolchain. Nonetheless, the achieved programmability comes at the expense of inefficiencies [11] and often suffers from substantial latency variability [12] that does not meet the latency guarantees of mission-critical tasks such as obstacle avoidance. Furthermore, throughput-driven platforms, such as embedded GPUs, typically require batch processing to reach their peak performance, with a substantial performance drop when batch size of 1 is used as is typical in latency-critical tasks. On the other end, although ASIC-based neural accelerators minimise inefficiencies, they come with the penalty of a fixed functionality after fabrication, which can soon make them outdated due to the rapid algorithmic advances of deep learning.

In this context, we target FPGAs to exploit both their customisation and reconfiguration capabilities, aiming for a balance between high performance and flexibility, and focused on the latency-driven mapping of CNN and LSTM inference on FPGA-based platforms. With reference to Fig. 2, Sec. III-A does not modify the algorithmic layer and concentrates on automated CNN-to-FPGA development tools and latency optimisation. Sec. III-B presents a model-hardware co-design method operating across all layers of the stack for the latency-optimised mapping of LSTM inference on FPGAs.

A. Latency-Optimised CNN Inference on FPGAs

To enable the hardware acceleration of latency-critical tasks, a latency-driven methodology of mapping CNNs to FPGAs was proposed in [4]. The CNN-to-FPGA methodology comprises a customisable hardware architecture, explicitly optimised for low-latency CNN inference, together with an automated flow for customising it based on the CNN workload and the resources of the target FPGA. Three main latency-first strategies drove the architectural design: *i) fine-grained customisability*, *ii) latency-centric design space exploration* and *iii) batchless execution*.

At the core of the architecture lies a pipeline of coarse building blocks, supporting typical CNN operations *e.g.* convolutional, pooling and ReLU blocks. The pipeline operates in a streaming manner with each block executing whenever data become available at its input. The architecture is compile-time configurable with respect to the number and type of blocks, the parallelism within each block and the connectivity among them. To generate an instance of the accelerator, the target CNN is first partitioned into subgraphs that will be executed by the accelerator in a time-sharing manner. Based on the CNN partitioning, the proposed flow derives a single, flexible hardware design that i) is capable of executing the layer patterns that are present in any of the subgraphs and ii) is run-time flexible with respect to its datapath so that no latency-costly bitstream-level FPGA reconfiguration is required. Flexibility with respect to datapath is introduced by means of run-time configurable interconnections among the instantiatied building blocks. This flexibility allows the architecture to process the workloads of different subgraphs by forming the appropriate datapath based on the current subgraph's index, without the need for the latency-costly reconfiguration of the FPGA fabric.

To tune the parameters of the architecture, the design space exploration is treated as an optimisation problem. In this respect, a latency-centric optimiser was developed aiming to determine the configuration of the architecture that minimises latency given a target CNN-FPGA pair. The optimiser searches over different partitionings of the CNN, topologies of the coarse pipeline and resource allocation among building blocks, and by employing a latency-driven objective function, guides the exploration towards latency-optimised designs. Following this approach, we demonstrated that the generated latency-optimised accelerators yield more than 5× latency gains over throughput-optimised designs for computationally expensive networks such as VGG, ResNet and DenseNet. Furthermore, compared to highly optimised implementations on NVIDIA's embedded TX1 GPU, the generated designs achieve an average latency improvement of more than 3× under a 5-watt power budget and 30% higher power efficiency in GOp/s/W when both platforms operate at their peak TDP [13].

B. Approximate LSTMs for Latency-Constrained Inference

Autonomous systems, such as self-driving cars, tend to be inherently complex, comprising of numerous streamlined sub-systems such as data pre-processing, localisation and mapping, navigation, obstacle avoidance, emergency reaction

and control [14]. Deep learning models are making their way in many of these sub-systems demonstrating state-of-the-art accuracy in a wide span of machine vision applications [15]. However, this advancement of machine learning algorithms comes at the cost of increased computational and memory demands that cannot be accommodated at the edge, within the resource-constrained environment of an autonomous agent.

Additionally, hard latency constraints are imposed on such mission-critical tasks to achieve real-time performance that would provide adequate reaction time and guarantee functional safety. In this respect, approximation techniques able to extract the best-possible estimate of the output of such systems complying to a pre-specified time budget can be employed to enable the autonomous agent to optimise its overall operation within a low-latency envelope.

At the moment, Long Short-Term Memory networks (LSTMs) [16] form the dominant recurrent neural network (RNN) model, capable of recognising temporal dependencies in sequential data such as video streams. This property has made LSTMs a prominent model for processing sensor data streams in various applications related to autonomous systems. In the field of autonomous driving for example, LSTMs are employed for extracting temporal features from the on-board camera's video streams that are exploited to develop a consistent driving policy [17].

In [5], we have introduced a novel approximate computing scheme for LSTMs that enables their efficient deployment in time-constrained environments by relaxing their demanding memory (and compute) requirements, exploiting their inherent redundancy. The proposed scheme enables us to restructure the computations lying at the core of an LSTM's workload (consisting of multiple matrix-vector multiplications) in order to perform the most information-rich computations first, by means of an SVD-based low-rank approximation of each weight matrix. This allows us to exploit the trade-off between latency and accuracy to yield the best-possible estimate of the final output at any time instant. Alongside, based on an importance criterion of each trained weight, a structured pruning of the model's weight matrices is employed to further tunably reduce the computational demands and memory footprint during inference, following the compute-aware model-hardware co-design approach of Fig. 2.

At the hardware level, the proposed approximation computing scheme is coupled with a parametrised custom architecture that enables the optimised hardware mapping of a given LSTM model on a target FPGA, tailored to the available time-to-decision budget. Having developed an analytical performance model that captures the attainable performance of different architectural configurations, we conduct design space exploration to co-optimise the LSTM approximation and the hardware design of the underlying architecture. This approach automatically configures varying levels of parallelism corresponding to different architectural dimensions of the hardware implementation, as well as approximation scheme parameters including the number of iterative approximation steps and the desired level of sparsity introduced by pruning.

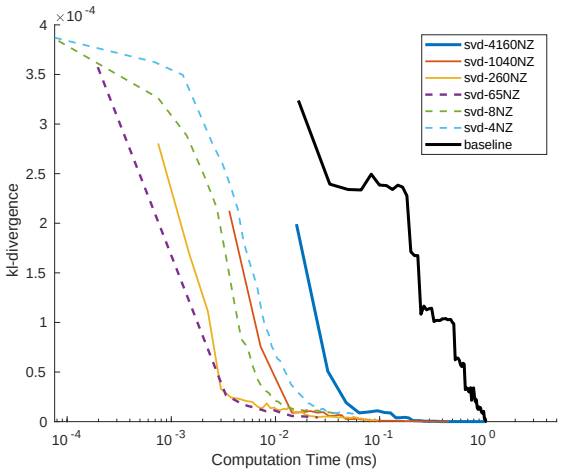

Fig. 4. Quality-of-result (lower is better on the y-axis) over computation time illustration for various instances of our approximate LSTM inference methodology, applied on the autonomous driving LSTM model of [19], a faithful implementation of which also acts as a baseline (black line). The proposed approach demonstrates notably higher quality approximations of the model's output, across the computation time axis.

Autonomous Driving through Approximate LSTMs. In [18], the proposed methodology has been applied on the driving model of [19] that directly predicts the feasibility of discrete actions from a visual sensor's data steam (move forward, stop, turn left, turn right). As a metric of accuracy we examine the KL-Divergence between the predicted probability distribution after each approximation step and the final output of the base model. As illustrated in Fig. 4, all the different configurations of the proposed methodology converge to a prediction similar to the desired, much faster than the examined baseline comprising of a faithful mapping of the LSTM into an optimised tiled matrix-vector multiplication architecture. Moreover, the proposed methodology offers a better trade-off between performance and accuracy, since for any given time budget it provides a prediction closer to the base model's final output, compared to the partial solution generated by the baseline in the same latency envelope.

IV. High-throughput Visual Perception

Although latency is the dominant optimisation objective on many real-time autonomous system applications, some perception tasks also rely on the capability of the system to cope with high rates of input data. In visual Simultaneous Localisation and Mapping (SLAM), for example, effective tracking of a mobile agent's position and orientation (localisation) requires processing input images in an exceedingly high frame-rate to avoid skipping frames that would potentially result to tracking failure under agile motion of the robot platform [20, 21]. CNNs, and other deep learning models, have recently started to emerge in visual SLAM applications in order to augment the progressively constructed map, for example, with semantic information [22].

Exploiting the inherent redundancy of deep learning models to achieve high performance by employing low-precision arithmetic has been widely studied in the literature [23]. Most

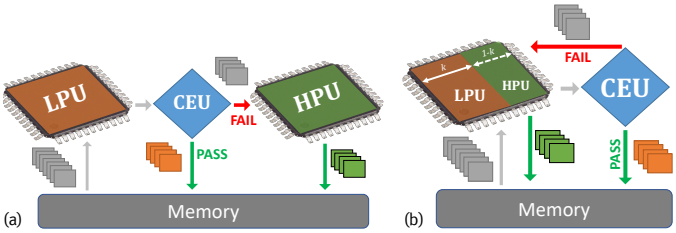

Fig. 5. CascadeCNN's architecture consisting of a low- and a high-precision unit, separated by a confidence evaluator acting on LPU's predictions: (a) Optimised for high-throughput, employing large batches and full device reconfiguration; (b) Throughput-latency co-optimised via resource sharing.

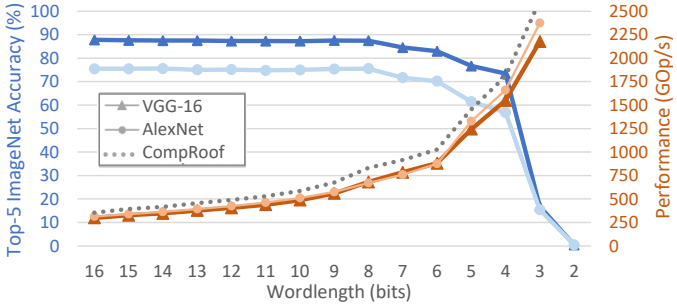

Fig. 6. Trade-off between precision, accuracy and performance of our novel CNN architecture (employed by both computational units of CascadeCNN), implemented on ZC706 FPGA. On both CNN models, CascadeCNN selects a 4-bit representation for the LPU and [6,8]-bit representation for the HPU depending on the user-specified error tolerance.

existing works explore the performance-accuracy trade-off by adopting the minimum wordlength that satisfies a pre-specified –typically inconsequential– error tolerance. Conversely to this approach, we have recently introduced CascadeCNN [7, 8], a novel automated toolflow that pushes the performance of precision-quantised CNNs by generating a high-throughput cascade of CNN classifiers.

An overview of CascadeCNN's architecture is depicted in Fig. 5a. The first stage of this cascade, called the Low-precision Unit (LPU), employs excessively low-precision arithmetic units providing rapid classification results at the expense of accuracy degradation. However, exploiting the fact that not all inputs require the same amount of processing, a significant portion of input samples yields a confident classification prediction at this stage. These are identified by a Confidence Evaluation Unit (CEU) based on how "spiky" the probability distribution of the LPU's prediction is. The remainder of the input samples that do not satisfy the CEU's high-confidence criterion are forwarded for re-processing on a High-precision Unit (HPU), to restore the overall application-level accuracy back to the desired level.

The arithmetic precision adopted by the LPU and HPU is tuned based on statistics extracted from a small set of validation samples. In the current setting, the LPU is derived by direct quantisation of the reference model, avoiding the need of parameter fine-tuning. However, the proposed approach is orthogonal to quantisation with retraining, allowing any precision-quantised model to act as CascadeCNN's LPU.

Especially in the case of precision-optimised models, FPGAs form a promising platform for efficient deployment due to their inherent customisability that provides substantial flexibility on the hardware implementation of the system. To accommodate both the low- and the high-precision units of CascadeCNN, a hardware architecture was designed that scales its performance (Fig. 6) with respect to the selected wordlength, by fully exploiting the available FPGA resources.

Performance modelling is conducted to enable design space exploration across various parallel dimensions of the proposed architecture, considering the CNN model, target FPGA device, user-specified error tolerance and application-based optimisation objectives (*i.e.* latency-throughput requirements). In the case of throughput-driven optimisation, each stage of the cascade (LPU and HPU) is independently mapped to an optimised architecture spread across the target FPGA, while full device reconfiguration is employed to alternate the two stages sequentially (Fig. 5a). To effectively amortise the cost of this occasional reconfiguration, batch processing is employed.

Nevertheless, with the latency being severely hurt by reconfiguration, when latency constraints are also imposed by the application, the design space exploration considers a resource-sharing approach between the two cascade stages. In this case, the resulting implementation adopts two concurrent instances of the proposed architecture –each supporting a different precision– on the same device (Fig. 5b).

CascadeCNN has been evaluated on the task of image classification on ImageNet's 2012 validation set. The results showcase that a throughput boost of up to 55% for VGG16 and 48% for AlexNet is achieved by CascadeCNN's designs, compared to optimised single-stage baselines under the same resource budget and error tolerance.

V. Enabling Multi-CNN Systems

As deep learning algorithms become more mature, several multi-CNN robotic systems are currently under development. Systems such as human-sensing robots and inspection drones typically employ either pipelines of neural networks, where one network feeds the next (*e.g.* a face detector followed by an emotion recogniser), or distinct DNNs working on independent tasks (*e.g.* infrastructure inspection and obstacle avoidance). In such settings, DNNs constitute mere components of the overall system and work synergistically to perform the tasks at hand.

Despite the algorithmic advances, from a pragmatic point of view, deploying multiple models on a given compute platform poses a number of challenges. At the workload level, with each DNN targeting a different task, the compute and memory load as well as the latency constraints vary accordingly. Instead of employing a fixed and model-agnostic architecture, this property requires an accelerator design that captures and reflects both the workload characteristics and the performance requirements of each model. Moreover, the resource-constrained setups of autonomous systems require the multiple CNNs to compete for the same pool of resources and hence resource allocation becomes a decisive factor.

To this end, we recently presented f-CNN^{x} [6], a toolflow that addresses the challenge of mapping multiple CNNs on

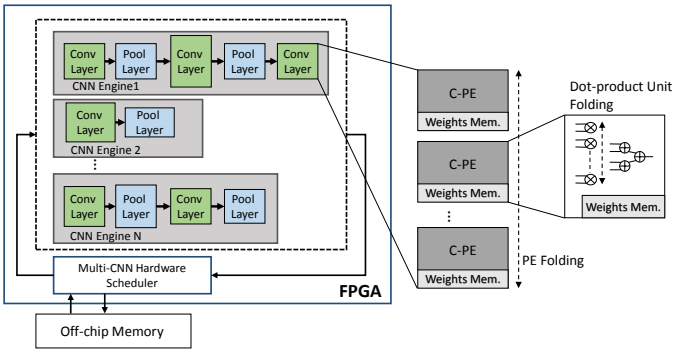

Fig. 7. Parallel architecture for multiple CNNs.

a single FPGA platform while satisfying the latency requirements for each network. f-CNN$^{\text{x}}$ introduces a novel multi-CNN hardware architecture (Fig. 7) consisting of i) a number of heterogeneous CNN engines and ii) a multi-CNN hardware scheduler (`MCNN-HS`). Instead of instantiating a fixed accelerator and relying on scheduling for the sequential execution of the target set of CNNs, the strategy of our framework is to generate one dedicated engine per CNN, customised to its workload and performance needs. This approach enables the concurrent execution of all networks in an efficient manner, with the `MCNN-HS` module allocating the external memory bandwidth to the CNN engines at run time, following a static schedule. To tune the parameters of the architecture, the design space exploration task is cast to a multiobjective optimisation problem aiming to tailor the allocation of both the on-chip resources and external memory bandwidth to the performance needs of the target set of CNNs.

The combination of a highly parametrised architecture that allows fine-grained tuning together with the optimisation of both the architecture and the schedule of the external memory transfers enables f-CNN$^{\text{x}}$ to overcome the limitations of competing parallel architectures, and demonstrates performance gains of up to 6.8$\times$ in performance-per-watt over highly optimised implementations on the NVIDIA Jetson TX1 GPU.

VI. Conclusion

The presented set of works focuses on enabling the embedded deployment of advanced DNNs on-board modern intelligent autonomous systems to enhance their capabilities. From an algorithmic perspective, novel DNN architectures play a key role in improving critical robotic tasks by reaching unprecedented accuracy levels. At the same time, the compute-aware design of DNNs can lead to the generation of hardware-friendly models that are co-optimised for both the task-level accuracy and the compute capabilities of the available processing platform. With latency being at the centre of requirements for mobile robots, custom hardware accelerator designs that are explicitly optimised for low-latency inference and operation under computation-time constraints can be an enabling technology for CNN- and LSTM-based decision-making systems. To provide the computing support for processing high-bandwidth visual sensor data, CascadeCNN exploits mixed-precision arithmetic to boost the throughput of on-board CNN inference. Finally, the f-CNN$^{\text{x}}$ framework paves the way in executing multiple CNNs under stringent latency constraints via FPGA acceleration.

VII. Acknowledgements

The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) is gratefully acknowledged. This work is also supported by EPSRC grant 1507723.

References

[1] S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Toolflows for Mapping Convolutional Neural Networks on FPGAs: A Survey and Future Directions," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 56:1–56:39, Jun. 2018.

[2] A. Kouris and C. Bouganis, "Learning to Fly by MySelf: A Self-Supervised CNN-Based Approach for Autonomous Navigation," in *2018 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct 2018.

[3] C. Kyrkou, G. Plastiras, T. Theocharides, S. I. Venieris, and C. S. Bouganis, "DroNet: Efficient Convolutional Neural Network Detector for Real-Time UAV Applications," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2018, pp. 967–972.

[4] S. I. Venieris and C. S. Bouganis, "Latency-Driven Design for FPGA-based Convolutional Neural Networks," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, Sept 2017.

[5] M. Rizakis, S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Approximate FPGA-based LSTMs under Computation Time Constraints," in *Applied Reconfigurable Computing - 14th International Symposium, ARC 2018, Santorini, Greece, May 2 - 4, 2018, Proceedings*, 2018, pp. 3–15.

[6] S. I. Venieris and C. S. Bouganis, "f-CNN$^{\text{x}}$: A Toolflow for Mapping Multiple Convolutional Neural Networks on FPGAs," in *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2018.

[7] A. Kouris, S. I. Venieris, and C.-S. Bouganis, "CascadeCNN: Pushing the Performance Limits of Quantisation in Convolutional Neural Networks," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 155–1557.

[8] A. Kouris, S. Venieris, and C.-S. Bouganis, "CascadeCNN: Pushing the performance limits of quantisation," in *SysML*, 2018.

[9] A. Loquercio, A. I. Maqueda, C. R. del Blanco, and D. Scaramuzza, "DroNet: Learning to Fly by Driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[10] D. Gandhi, L. Pinto, and A. Gupta, "Learning to Fly by Crashing," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 3948–3955.

[11] R. Hameed *et al.*, "Understanding Sources of Inefficiency in General-purpose Chips," in *Proc. of the 37th Annual Int. Symposium on Computer Architecture*. ACM, 2010.

[12] C. Wu *et al.*, "Machine Learning at Facebook: Understanding Inference at the Edge," in *HPCA*, 2019.

[13] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[14] S. Thrun, "Toward robotic cars," *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.

[15] M. Bojarski *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] L. Chi and Y. Mu, "Learning End-to-End Autonomous Steering Model from Spatial and Temporal Visual Cues," in *Workshop on Visual Analysis in Smart and Connected Communities (VSCC)*. ACM, 2017.

[18] A. Kouris, S. I. Venieris, M. Rizakis, and C.-S. Bouganis, "Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars," *arXiv*, 2019.

[19] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-End Learning of Driving Models from Large-Scale Video Datasets," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[20] K. Boikos and C.-S. Bouganis, "A Scalable FPGA-Based Architecture for Depth Estimation in SLAM," in *Applied Reconfigurable Computing*. Cham: Springer International Publishing, 2019, pp. 181–196.

[21] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?" in *European Conference on Computer Vision (ECCV)*. Springer, 2012.

[22] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks," in *Int. Conf. on Robotics and Automation (ICRA)*, 2017.

[23] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized CNN: A Unified Approach to Accelerate and Compress Convolutional Networks," *IEEE Trans. on Neural Networks and Learning Systems*, no. 99, 2017.