






Automated Search for Deep Neural Network Inference Partitioning on Embedded FPGA

Fabian Kress^(✉), Julian Hoefler, Tim Hotfilter, Iris Walter,
El Mahdi El Annabi, Tanja Harbaum, and Jürgen Becker

Karlsruhe Institute of Technology, Karlsruhe, Germany
{fabian.kress,julian.hoefler,hotfilter,iris.walter,
harbaum,becker}@kit.edu

Abstract. Deep Neural Networks (DNNs) are currently making their way into a broad range of applications. While until recently they were mainly executed on high-performance computers, they are now also increasingly found in hardware platforms of edge applications. In order to meet the constantly changing demands, deployment of embedded Field Programmable Gate Arrays (FPGAs) is particularly suitable. Despite the tremendous advantage of high flexibility, embedded FPGAs are usually resource-constrained as they require more area than comparable Application-Specific Integrated Circuits (ASICs). Consequently, co-execution of a DNN on multiple platforms with dedicated partitioning is beneficial. Typical systems consist of FPGAs and Graphics Processing Units (GPUs). Combining the advantages of these platforms while keeping the communication overhead low is a promising way to meet the increasing requirements.

In this paper, we present an automated approach to efficiently partition DNN inference between an embedded FPGA and a GPU-based central compute platform. Our toolchain focuses on the limited hardware resources available on the embedded FPGA and the link bandwidth required to send intermediate results to the GPU. Thereby, it automatically searches for an optimal partitioning point which maximizes the hardware utilization while ensuring low bus load.

For a low-complexity DNN, we are able to identify optimal partitioning points for three different prototyping platforms. On a Xilinx ZCU104, we achieve a 50% reduction of the required link bandwidth between the FPGA and GPU compared to maximizing the number of layers executed on the embedded FPGA, while hardware utilization on the FPGA is only reduced by 7.88% and 6.38%, respectively, depending on the use of DSPs and BRAMs on the FPGA.

Keywords: Distributed sensor systems · Neural network inference partitioning · Design space exploration · Embedded FPGA

1 Introduction

In the recent decade, Deep Neural Networks (DNNs) became the preferred algorithm for evaluating complex data, like images or radar information. These algorithms show great performance and accuracy, while they usually can be deployed readily. However, DNN inference for complex data can cause high computational complexity, resulting in extensive power consumption. This especially becomes an issue when DNN are used in energy-constrained or safety-critical environments like embedded low-power systems. Hence, the execution of DNN moved from traditional computing devices such as Central Processing Units (CPUs) and Graphics Processing Units (GPUs), further into hardware accelerators designed for fast or energy-efficient DNN execution. These are either implemented in an Application-Specific Integrated Circuit (ASIC) or an embedded Field-Programmable Gate Array (FPGA). The latter provides flexibility regarding runtime reconfiguration or future architecture updates and provides significantly reduced development time. Dedicated accelerators for a given DNN offer a great trade-off between power consumption and performance, but they often lack flexibility to model different kinds of DNNs.

Accelerators for DNNs in autonomous driving or assistive robotics are especially demanding as resource and real-time requirements in those multisensory systems are high. These platforms, such as the humanoid assistive robot ARMAR-6 [2], are usually based on a system architecture as shown in Fig. 1. For visual perception, ARMAR-6 is equipped with a stereo camera and an RGB-D camera. As the bus is highly occupied, images are directly streamed to the compute platform consisting of three PCs, a GPU and an FPGA. The actual data processing not only consists of image processing, e.g., person recognition, human pose estimation, object detection and localization, but speech recognition, force control, task planning, etc. as well. These tasks are distributed among the different devices of the compute platform, realizing DNN-based tasks on the GPU and FPGA, respectively. Time-sensitive applications, e.g., face and gesture recognition, are accelerated on the FPGA [17]. However, acceleration of the whole DNN within the FPGA as dataflow is not feasible for each model. Therefore, we propose the co-execution of DNNs in the distributed system of GPU and FPGA.

The example above shows that there is a need for highly efficient and performant DNN accelerators, which also offer a great flexibility for many different DNN workloads. Over the past years, different optimization strategies for DNN have evolved to make the underlying operations more efficient, such as quantization [8] or pruning [5]. A more recent optimization scheme considers DNNs that are executed on complex System-on-Chips (SoCs) with multiple different domains like CPUs, GPUs or FPGAs. Those systems allow combining the advantages of the various system components to achieve a high overall performance. However, efficient partitioning between the different domains is an emerging challenge. Therefore, we present in this paper our approach to automatically partition a DNN workload between FPGAs and GPUs, which are common domains in novel SoC-architectures. Our toolflow takes a model description from PyTorch and interacts with the FPGA design tools to generate different bitstreams and

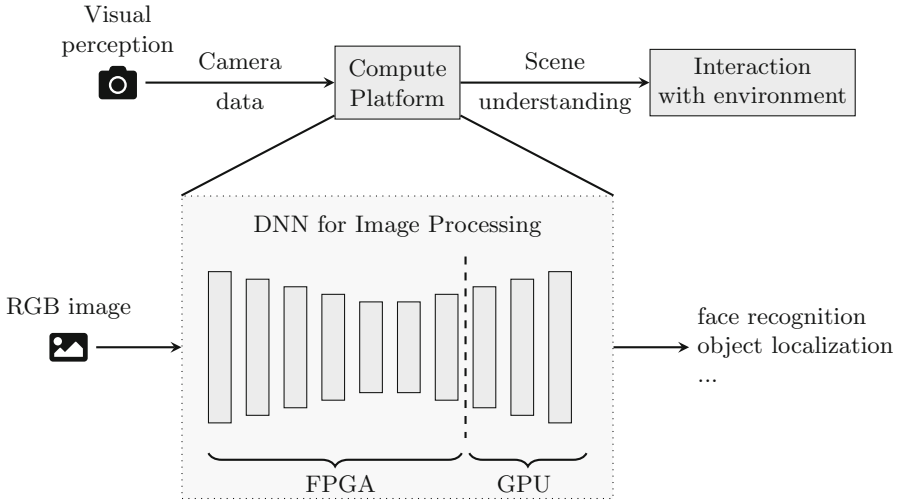


Fig. 1. Conceptual overview. Camera images are captured from the environment and forwarded to the computing platform, which executes a DNN. The scene perception obtained is then used to interact with the environment. Thereby, the DNN inference takes place either on the FPGA, the GPU, or both. For distributed processing, our toolflow determines the optimal partitioning point.

their resource utilization. These are then evaluated considering the given user and FPGA design constraints. Our approach also takes the communication link between the FPGA and the GPU into account, which can be an on-chip solution but also a link between two physically separated platforms. Finally, our tool returns a partitioning point for the DNN that maximizes energy efficiency and performance based on the evaluation results collected before. In summary, our paper makes the following contributions:

- We present our toolflow for determining an optimal partitioning point regarding hardware resource usage and required link bandwidth.
- We apply the toolflow to estimate bandwidth and hardware utilization of quantized DNNs.
- We exemplarily show beneficial partitioning points of quantized MobileNet V1 for different FPGA architectures.

2 Related Work

Distributing DNN inference over multiple compute platforms has been a widely studied topic in recent years. Several publications showed that DNN partitioning is a beneficial approach for edge platforms in terms of latency, memory consumption and link bandwidth utilization [7, 10, 11, 14]. Some of these studies use an adaptive approach to further improve efficiency of the distributed systems by dynamically allocating computational resources depending on the overall system utilization.

Teerapittayanon et al. proposed DDNNs (Distributed Deep Neural Networks) which consider distributed compute hierarchies from cloud to end devices during training [16]. Thereby, they define local exit points within the DNN architecture for each compute platform in the system. According to the presented results, this approach leads to improved accuracy and reduced communication costs in contrast to combining a small DNN on the end device and a large DNN on the central platform or in the cloud.

However, research on distributing inference mostly focuses on DNN partitioning for commercially available off-the-shelf (COTS) platforms such as Tensor Processing Units (TPUs) and GPUs, neglecting evaluation of more energy efficient ASICs or FPGA-based hardware architectures. Since Internet-of-Things (IoT) platforms are often power constrained, a comprehensive hardware/software co-design across multiple platforms is required to allow for larger and more complex DNNs in end devices. In addition, the design space exploration must include an evaluation of link utilization, as distributed systems are severely limited in terms of available bandwidth between computing platforms.

Efficient DNN inference on multi-FPGA architectures has been studied by some works recently [4, 9, 12, 13]. As an example, Zhang et al. propose a mapping approach for large-scale DNNs on asymmetric multi-FPGA platforms considering the required link bandwidth in the system as well as resource allocation to achieve increased performance [18]. The presented mapping problem is solved by dynamic programming for DNN partitioning. Alonso et al. presented Elastic-DF, a framework for resource partitioning in multi-FPGA systems including dynamic mapping of applications to an available accelerator in the FPGA cluster [1]. Thereby, the tool can automatically optimize the performance of a pipelined dataflow DNN inference based on the available hardware resources of each individual FPGA. Although both approaches apply resource- and bandwidth-aware DNN partitioning to increase performance, these target datacenter inference and do not provide any investigation on low-power platforms used in IoT applications.

3 Partitioning Toolflow

Distributed sensor platforms, as found in many applications such as autonomous driving or assistive robotics, often face the problem of limited available bandwidth and limited compute resources in the central computing platform. Especially in safety-critical use cases, minimum latency must be guaranteed. In addition, since DNN topologies are still a major research topic, the hardware architectures also have to provide certain flexibility to cope with varying requirements. Hence, we propose a bandwidth- and resource-aware toolflow for pipelined DNN inference partitioning as shown in Fig. 2. In contrast to state of the art, this approach takes limited resources on the embedded FPGA as well as the limited bandwidth to the central compute platform in low-power embedded systems into account. Thereby, to achieve low latency and high throughput, the hardware implementation on the FPGA is designed in a pipelined manner, i.e. each layer

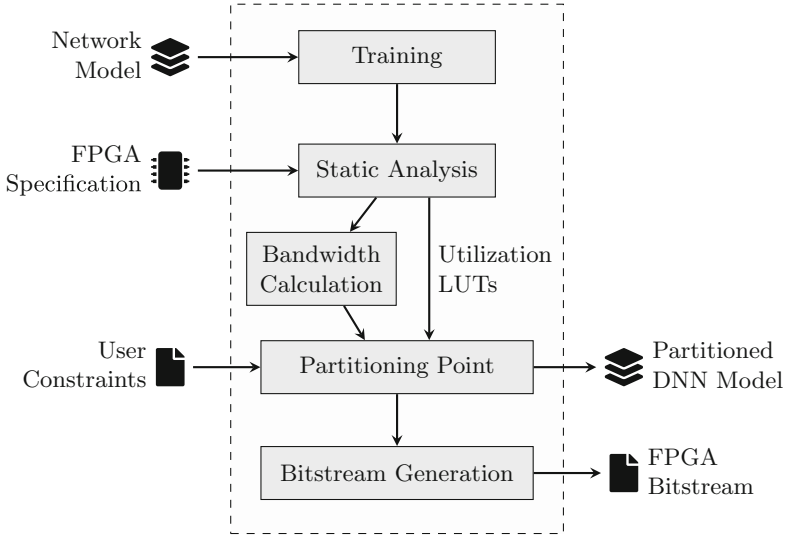


Fig. 2. Overview of our toolflow. As input, we take any given DNN workload, an FPGA specification and user constraints, e.g. available link bandwidth, to determine an optimal partitioning point. The toolflow outputs the partitioned DNN model and generates a bitstream fulfilling the system requirements.

is mapped to a dedicated accelerator. These are not shared between multiple layers of the DNN. Consequently, full hardware implementation on an embedded FPGA would consume a lot of space on the SoC.

3.1 Overview

Our approach offers a toolflow that evaluates any given DNN workload. The DNN is partitioned between an embedded FPGA and a GPU to maximize the performance, while also considering the communication link. Along with the network model, our toolflow also requires an embedded FPGA specification and user constraints as input to distinguish an optimal partitioning point of the DNN regarding resource utilization and required link bandwidth. Especially, the used link bandwidth is an important metric in multi-sensor systems as they can be found in assistive robotics or autonomous driving. In such use cases, the available bandwidth is severely limited by the large amount of data being sent from different nodes on the bus. Hence, the user can set maximum available link utilization to account for other traffic on the bus. Embedded FPGAs on the other hand are often limited in area and thus in available hardware resources. These constraints also have to be taken into account to find an optimal partitioning point of a DNN.

As a whole, our toolflow first optimizes the given DNN by quantizing weights and layer outputs during training to reduce computational complexity and

memory footprint. Based on the resulting quantized DNN, a static analysis is performed taking FPGA specifications like available Block RAM (BRAM) and Digital Signal Processor (DSP) resources into account. In addition, finding an optimal partitioning point of the DNN requires calculation of the required bandwidth. Finally, with the estimated hardware resource utilization, the calculated link bandwidth of each layer, and the given user constraints, our toolflow can determine an optimal partitioning point and generate an appropriate bitstream.

3.2 Training and Static Analysis

Achieving low latency and high throughput on the embedded FPGA is a crucial part which enables usage of such platforms in embedded systems. Hence, we need to automatically optimize a DNN architecture and analyze the resulting model to efficiently map the layers to dedicated hardware accelerators.

Various works in recent years have shown that quantization and pruning lead to a drastic reduction in hardware resource consumption, with only a minimal loss of accuracy [17]. Therefore, in use cases such as assistive robotics which require low power consumption, optimizing DNN is inevitable. Our toolflow makes use of Brevitas [15] to achieve this goal during training. It is based on PyTorch and supports quantization-aware training of DNNs through evaluating reduced precision hardware building blocks at different levels. The resulting optimized DNN is then exported to ONNX file containing custom node types.

Since our approach targets Xilinx FPGAs as test platform, we implement FINN framework as one of the central components our toolflow interacts with for static analysis [3]. The FINN framework provides an end-to-end workflow covering design space exploration based on resource cost estimations and performance predictions, as well as automated code generation for High-level Synthesis (HLS). It takes the ONNX file generated by Brevitas of the DNN as input and provides estimates of hardware resource consumption and performance, among others. In addition, FINN can generate a bitstream for the given FPGA based on Vivado HLS.

3.3 DNN Partitioning

Even though the FINN framework tries to find an optimal implementation of the DNN taking multiple constraints into account, the size of the FPGA is not considered during design space exploration. Hence, if the network model requires more hardware resources than available on the given FPGA, the implementation will fail. Our approach addresses this problem by searching for an optimal partitioning point regarding FPGA usage and required link bandwidth.

To estimate the resources per layer, we use the predictions of the FINN framework, which are automatically generated during HLS. FINN offers different ways to estimate the required hardware resources per layer, before and after IP block generation and also after out-of-context synthesis including hardware optimizations. The latter thereby allows for precise resource estimates at the expense of runtime. Since we aim to find an optimal partitioning point for a

Algorithm 1: Search algorithm to determine an optimal partitioning point, considering hardware resource usage and required link bandwidth.

```

1 function GetOptimalSplitNode;
   Input : hardware resources per layer, layer output size
   Output: Partitioning Point
2 max_layer  $\leftarrow$  last DNN layer fitting on hardware;
3 part_pnt  $\leftarrow$  max_layer;
4 for each layer in [max_layer, first_layer] do
5   | bw_ratio  $\leftarrow$  bandwidth[part_pnt] / bandwidth[layer];
6   | hw_ratio  $\leftarrow$  hw_resources[part_pnt] / hw_resources[layer];
7   | if bw_ratio > 1 and bw_ratio/hw_ratio > threshold_ratio then
8   |   | part_pnt  $\leftarrow$  layer;
9   | end
10  | if stop condition fulfilled then
11  |   | break;
12  | end
13 end
14 return part_pnt;

```

given DNN and thus runtime is not critical, our toolchain takes resource estimates of FINN generated after IP block generation. The link bandwidth can be calculated according to the output feature size of the intermediate layers and the corresponding data bit width. Consequently, this analysis can be neglected with respect to the runtime of the toolflow.

Besides estimating FPGA hardware resources and calculating required link bandwidth, distinguishing an optimal partitioning point of the DNN involves the input of constraints by the designer. This includes the embedded FPGA specifications regarding number of available basic building blocks, the targeted hardware utilization, and the available link bandwidth. Based on these inputs, our approach searches for a suitable partitioning of the DNN, which does not violate any of the given constraints such as area or required link bandwidth. Algorithm 1 presents our approach for finding an optimal partitioning point. First, the algorithm determines a partitioning point in the DNN where the hardware utilization is maximized for a given embedded FPGA platform. Afterwards, the algorithm aims at minimizing the communication overhead while still keeping the hardware utilization as high as possible. To achieve this goal, we set two parameters in advance: The first parameter defines the minimum allowed hardware utilization, which is used as a *stop condition* and should not be undercut. The second parameter defines a *threshold* for the maximum acceptable ratio between optimization of communication overhead and deterioration of hardware utilization. Based on this, the algorithm iterates through the layers starting from the partitioning point determined in the first step of the algorithm and evaluates link bandwidth utilization and hardware resource utilization. Subsequently, these are compared with the current best partitioning point. Only if the bandwidth can be reduced and the threshold value is exceeded, the layer is set as the new

Table 1. Available hardware resources on the evaluated SoC.

Platform	LUTs	FFs	BRAM blocks	DSP slices
ZedBoard	53,200	106,400	280	220
Ultra96-V2	70,560	141,120	432	360
ZCU104	230,400	460,800	624	1,728

partitioning point. Finally, when the stop condition is reached, the algorithm returns the partitioning point with the best ratio.

In summary, our toolflow optimizes the system towards high resource utilization of the embedded FPGA and low link bandwidth. After the partitioning point of the DNN is set, our toolflow splits the DNN model into two sub-models accordingly. Thereby, both are exported to ONNX format based file, which ensures machine learning interoperability. For the embedded FPGA, our toolflow finally generates the bitstream using Xilinx Vivado HLS and Vivado.

4 Evaluation

In this section, we evaluate our toolchain for a DNN on embedded FPGAs. Since we use FINN framework as one of the central components, we exemplarily show the results of inference partitioning for three different Xilinx FPGAs. In order to address the various possible sizes of embedded FPGAs, we evaluate DNN partitioning using the following platforms: ZedBoard, Avnet Ultra96-V2 and Xilinx Zynq UltraScale+ MPSoC ZCU104. The available hardware resources on each SoC are listed in Table 1.

The system we use for the evaluation of our toolflow consists of an Intel Core i7-8565U, a quad-core SoC, running Ubuntu 18.04. To ensure the correct functioning of the FINN framework, we use a Docker container generated from the Dockerfile provided by Xilinx for this purpose. Finally, Vivado 2020.1 is used for HLS to determine the required hardware resources.

4.1 Workload

Low-complexity DNNs are required in embedded systems that need to provide low latency and power consumption. Several DNNs architectures have been proposed in recent years fulfilling these properties, such as MobileNet V1 [6]. Therefore, we select it as an exemplary workload and identify optimal partitioning points for a distributed system combining GPUs and embedded FPGAs. To achieve lower hardware resource utilization, we use a pretrained and quantized model with 4-bit weights and activations for the evaluation. The MobileNet V1 was originally proposed in 2017 and achieves an accuracy of 70.6% on the ImageNet dataset while only using 569 million multiply-accumulate operations and 4.2 million trainable parameters. This is achieved by introducing depthwise-separable convolution blocks, where each block consists of a depthwise convolution followed by a convolution with 1×1 kernels. In total, MobileNet V1 uses

13 depthwise-separable convolution blocks, preceded by a standard 3×3 convolution and followed by a fully-connected layer and softmax for classification.

4.2 Results

Executing DNN inference on an FPGA requires to map layers to one or more basic building blocks depending on the layer type. For MobileNet V1, FINN converts the DNN into 86 layers that can be directly translated to the components available in its hardware library. Without considering the runtime for training the DNN, it takes about 81.6 min on our system in dual-core mode from loading the ONNX file in FINN to finishing HLS. As expected, the IP block generation of each building block takes the most time, about 79 min, which is almost 97% of the whole runtime. However, since this step only needs to be performed once for a quantized DNN model and our evaluation was performed on a low-performance SoC, the runtime is still within an acceptable range.

The results of the IP block generation and the output size of each layer are shown in Fig. 3. It can be seen, that the output size tends to decrease towards the last layer. However, the DNN requires large hardware accelerators, especially towards the last layers, which significantly increases the demands on the available resources of the embedded FPGA. Consequently, the identification of an optimal partitioning point depends in particular on the deployed hardware platform and the defined user constraints.

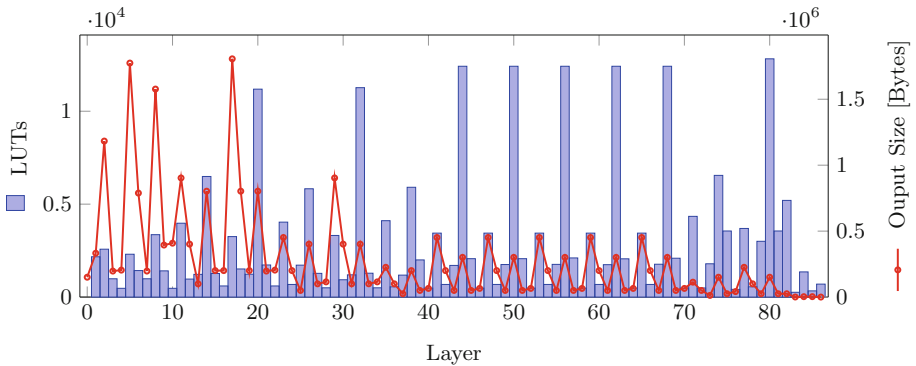


Fig. 3. Resource utilization of each translated MobileNet V1 layer taking LUTs as an example and the corresponding output size in bytes. In this configuration, only LUTs and FFs are used to implement the building blocks in the FPGA.

In the following, we apply our toolflow to find an optimal partitioning point of MobileNet V1 on the three aforementioned exemplary FPGA platforms. Since we want to analyze the impact of DNN inference partitioning independently of the IP cores present on the platforms, we will also evaluate the hardware resource

Table 2. Toolflow evaluation results. Depending on the size of the SoC and whether all resources or only LUTs and FFs are used, the maximum number of layers (max. Layer) that can be implemented on the platform varies. It can be seen that reducing the number of layers executed on the FPGA can significantly reduce the required link bandwidth (BW Red.) since the optimal partitioning point (Part. Point) does not always match the max. layer. Thereby, the hardware utilization reduction (HW Red.) is small.

Platform	Only LUTs/FFs				All resources			
	max	Part	BW Red.	HW Red.	max	Part	BW Red.	HW Red.
	Layer	Point	[%]	[%]	Layer	Point	[%]	[%]
ZedBoard	22	21	6.78	1.19	19	19	0	0
Ultra96-V2	31	25	50	18.73	25	25	0	0
ZCU104	79	73	50	7.88	79	73	50	6.38

consumption of architectures containing only Look-Up Tables (LUTs) and Flip-Flops (FFs). We set the stop condition to 70% minimum hardware resource utilization to allow for low DNN inference latency and the threshold ratio to 1. The results of our exploration are shown in Table 2.

For the ZedBoard, the algorithm finds layer 21 as the optimal partitioning point when only using LUTs and FFs. Even though many hardware resources could be saved when choosing layer 20 instead, this would lead to an increased link bandwidth requirement as can be seen in Fig. 3. Hence, the selected partitioning layer offers a good trade-off between high throughput and low communication overhead. Similarly for Ultra96-V2, layer 25 is identified as an optimal partitioning point since bandwidth can be reduced by 50% in comparison to layer 31 without offloading too many layers to the GPU. In this case, the required hardware resources on the FPGA are significantly reduced by 18.73%, however, the stop condition is still exceeded.

Compared to the consideration of the optimizations for an implementation solely based on LUTs and FFs, only the evaluation results for the ZCU104 show different partitioning than maximum layer. This is due to the fact that FPGA platforms are usually severely limited in terms of BRAM and DSP resources. Since the maximum number of layers that can be implemented on ZedBoard and Ultra96-V2 is small considering all hardware resources, there is no potential to reduce link bandwidth on these platforms. In contrast, link bandwidth can be reduced by 50% on ZCU104, at the cost of a reasonable reduction in hardware utilization.

5 Conclusion and Future Work

DNN inference partitioning can be very advantageous depending on the neural architecture and the deployed hardware. Our results show that it is also beneficial for embedded FPGAs to outsource workload partly from a central compute node to a platform closer to the sensor since this approach reduces the

required bandwidth while maximizing the hardware utilization of the embedded FPGA. The latter can thereby result in lower DNN inference latency depending on the hardware deployed on the central compute node. Especially in applications using multiple sensors, our approach can propose a bandwidth-aware partitioning to enable parallel execution of several DNN-based applications in a distributed system. Using our toolflow, we were able to identify several advantageous partitioning points depending on the platform deployed and the type of hardware resources used on the embedded FPGA. In the best case, our approach can reduce the required link bandwidth by 50% compared to implementing the maximum possible number of layers in the FPGA.

In the future, we plan to further investigate DNN inference partitioning by evaluating not only the sensor node and the link but also the central compute platform of the embedded system. Depending on the workload, analyzing latency on both partitions increases the design space but allows to investigate DNN partitioning for even more applications where minimum latency or energy consumption is the main optimization goal.

Acknowledgment. This work has been supported by the project “Stay young with robots” (JuBot). The JuBot project was made possible by funding from the Carl Zeiss Foundation. The responsibility for the content of this publication lies with the authors.

References

1. Alonso, T., et al.: Elastic-DF: scaling performance of DNN inference in FPGA clouds through automatic partitioning. *ACM Trans. Reconfigurable Technol. Syst.* **15**(2), 1–34 (2021). <https://doi.org/10.1145/3470567>
2. Asfour, T., et al.: ARMAR-6: a high-performance humanoid for human-robot collaboration in real-world scenarios. *IEEE Robot. Autom. Mag.* **26**(4), 108–121 (2019). <https://doi.org/10.1109/MRA.2019.2941246>
3. Blott, M., et al.: FINN-R: an end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfigurable Technol. Syst.* **11**(3), 1–23 (2018). <https://doi.org/10.1145/3242897>
4. Cheng, Q., Wen, M., Shen, J., Wang, D., Zhang, C.: Towards a deep-pipelined architecture for accelerating deep GCN on a Multi-FPGA platform. In: Qiu, M. (ed.) *ICA3PP 2020*. LNCS, vol. 12452, pp. 528–547. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60245-1_36
5. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. [arXiv:1506.02626](https://arxiv.org/abs/1506.02626) [cs], October 2015
6. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications (2017). <https://doi.org/10.48550/ARXIV.1704.04861>, <https://arxiv.org/abs/1704.04861>
7. Hu, C., Bao, W., Wang, D., Liu, F.: Dynamic adaptive DNN surgery for inference acceleration on the edge. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1423–1431 (2019). <https://doi.org/10.1109/INFOCOM.2019.8737614>
8. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Quantized neural networks: training neural networks with low precision weights and activations. [arXiv:1609.07061](https://arxiv.org/abs/1609.07061) [cs], September 2016

9. Jiang, W., et al.: Achieving super-linear speedup across multi-FPGA for real-time DNN inference. *ACM Trans. Embed. Comput. Syst.* **18**(5s), 1–23 (2019). <https://doi.org/10.1145/3358192>
10. Ko, J.H., Na, T., Amir, M.F., Mukhopadhyay, S.: Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In: 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1–6 (2018). <https://doi.org/10.1109/AVSS.2018.8639121>
11. Krefß, F., et al.: Hardware-aware partitioning of convolutional neural network inference for embedded AI applications. In: 2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS), pp. 133–140 (2022). <https://doi.org/10.1109/DCOSS54816.2022.00034>
12. Kwon, D., Hur, S., Jang, H., Nurvitadhi, E., Kim, J.: Scalable multi-FPGA acceleration for large RNNs with full parallelism levels. In: 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6 (2020). <https://doi.org/10.1109/DAC18072.2020.9218528>
13. Mittal, S.: A survey of FPGA-based accelerators for convolutional neural networks. *Neural Comput. Appl.* **32**(4), 1109–1139 (2020). <https://doi.org/10.1007/s00521-018-3761-1>
14. Mohammed, T., Joe-Wong, C., Babbar, R., Francesco, M.D.: Distributed inference acceleration with adaptive DNN partitioning and offloading. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, pp. 854–863 (2020). <https://doi.org/10.1109/INFOCOM41043.2020.9155237>
15. Pappalardo, A.: Xilinx/brevitas (2021). <https://doi.org/10.5281/zenodo.3333552>
16. Teerapittayanon, S., McDanel, B., Kung, H.: Distributed deep neural networks over the cloud, the edge and end devices. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 328–339 (2017). <https://doi.org/10.1109/ICDCS.2017.226>
17. Walter, I., et al.: Embedded face recognition for personalized services in the assistive robotics. In: Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD 2021. CCIS, vol. 1524, pp. 339–350. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-93736-2_26
18. Zhang, W., Zhang, J., Shen, M., Luo, G., Xiao, N.: An efficient mapping approach to large-scale DNNs on multi-FPGA architectures. In: 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1241–1244 (2019). <https://doi.org/10.23919/DATE.2019.8715174>