Enabling Neural Network Edge Computing on a Small Robot Vehicle

Jan Bredereke

Abstract We report on experiences with optimizations that enable neural network edge computing on a small robot vehicle, which serves as a proxy for an autonomous robotic space craft. We realize a visual object recognition task using a neural network on a field-programmable gate array (FPGA) with data processing resources as limited as those of an FPGA suitable for space. We use a quantized neural network nicely matching the properties of an FPGA. The restrictions of the small FPGA require us to sequentialize the processing partially. We employ input frame tiling for this. It allows us to keep the entire neural network on-chip. Furthermore, we split up the visual object recognition task into two stages, using two separate neural networks. The first stage identifies the region of interest approximately, using large and thus few tiles. The second stage looks closely at the single tile containing the region of interest; thus being not that time critical.

1 Introduction

This introduction first motivates our work, then describes our sample application, and finally introduces our solution approach.

1.1 Motivation

Neural networks are often used in data centers with powerful and power consuming special hardware. A current research question is how to also make full use of neural networks at the "edge" of the Cloud. That is, close to the sensors and the actors,

Jan Bredereke
City University of Applied Sciences Bremen · Flughafenallee 10 · 28199 Bremen · Germany ·
e-mail: jan.bredereke@hs-bremen.de

or even autonomously from data and power supply connections. The CPU of a microcontroller has too scarce data processing resources for this. An FPGA can offer more data processing resources, both in absolute numbers and in relation to its power consumption. An FPGA is very well suited for a highly parallel structure such as that of a neural network. In practice, however, many optimization tasks need to be solved before full use of the potential of the FPGA can be made.

Our particular motivation comes from space craft engineering. On-board computers provide especially scarce data processing resources. Access to the ground segment is usually available intermittently only. Due to space radiation, current off-the-shelf processors would fail soon. Therefore, special processors are used. Their chips feature structural widths of at least 65 nm instead of the denser 10 nm or less currently in use elsewhere. These special processors compromise on data processing resources to achieve sufficent robustness. An extremely small number of computers of this kind is made. Therefore, they usually are not made with application specific integrated circuits (ASICs), but with programmable standard hardware (FPGAs). Radiation-hard versions of some FPGAs are available, which have suitably larger structural widths.

There is increasing demand for on-board computing resources. Examples are on-board image processing, e.g. for autonomous rovers on other celestial bodies, or for constellations of nano-satellites, each with narrow bandwidth to the ground segment.

The Internet of Things (IoT) is an area with similar challenges. Cost restrictions lead to scarce data processing resources. Many of these devices are battery powered. This further restricts the data processing resources. Thus, challenges and solutions in space craft engineering might be applicable to the IoT, too, and vice versa.

1.2 The Sample Application

We employ an autonomous model car as a practical robot vehicle, serving as a proxy for an autonomous robotic space craft. The vehicle is equipped with a camera and a system-on-chip (SoC) Xilinx Zynq-7020. The SoC features an Arm CPU and, in particular, an FPGA Artix-7. The data processing resources of the Artix-7 are quite similar to those of an FPGA suitable for space [HSB+20, Chap. 2.1.3]. The SoC is integrated into a PYNQ-Z1 board. In Figure 1, the board can be spotted easily due to its pink colour.

The task of the vehicle is to visually recognize a person in view, to follow it when it moves, and to obey to driving commands given by simple gestures. It shall do all of this autonomously and in real-time.

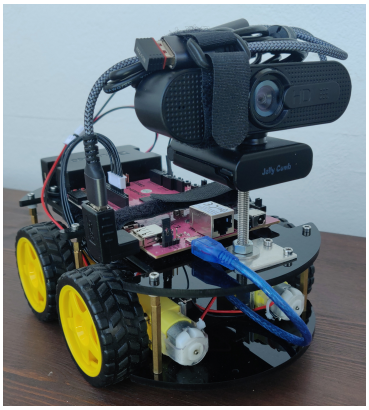

Fig. 1 The sample application vehicle with a camera and an FPGA board. (photo: Felix Müller)

1.3 Approach for Optimizations

We investigate optimizations that enable neural network edge computing on this kind of small robot vehicle. Many ideas are conceivable. Up to now, we tried out the following:

- Use a pre-trained neural network for inference only on the vehicle.
  Rationale: inference is an order of magnitude cheaper than training.
- Use hardware acceleration for the neural network by an FPGA, with the FPGA choice obeying the restrictions of the application area.
  Rationale: a neural network is inherently concurrent, and an FPGA offers massively parallel execution.
- Reduce hardware resources for the neural network by employing a quantized neural network.
  Rationale: a quantized neural network demands far less bits per node, and recent research showed that the resulting loss of precision per node can be more than compensated by a few more nodes [Blo+18; Umu+17].
- Reduce hardware resources for the neural network by splitting the input image into tiles and process them sequentially.
  Rationale: if the task doesn't fit the chip area available at all, making an area/speed trade-off by partially sequentializing the task can help.
- Reduce the number of tiles required, and thus the number of iterations, by separating locating a person in a frame and determining the person's gesture.
  Rationale: A small object of interest requires many tiles for finding it and evaluating its properties in detail. A two-staged approach can cope with larger, less detailed and thus less numerous tiles for finding. The subsequent gesture recognition needs a single tile only. Our restricted-hardware setting requires the sequentialization by tiling; the expensive part here is finding the "region of interest", not evaluating it.

We report on the details and on our experiences in the remainder of this paper.

2 Related Work

The *CHREC Space Processor (CSP)* [Man+18] implements a neural network for visual object recognition on a Xilinx SoC Zynq-7020, like we do. It flew as an experiment on the International Space Station (ISS) in 2017. However, it did not use the FPGA of the SoC for acceleration. The CASPR experiment including the successor *SHREC Space Processor (SSP)* [Rof+20] has just recently begun on Jan. 11, 2022 onboard the ISS.

The *FINN framework* [Umu+17] supports implementing a binarized neural network (BNN) on an FPGA. The *FINN-R framework* [Blo+18] is an improved version. It also supports quantization beyond binary. Both papers report extensively on the efficiency gains of quantizing a neural network.

BinaryEye [JEB18] uses the FINN framework to classify a video stream of handwritten digits at 20.000 frames per second (fps). Each frame is limited to fixed 32×32 pixels, in black/white. BinaryEye uses a Kintex-7 FPGA, more powerful than the Artix-7 used by us.

The *DAC-SDC low power object detection challenge for UAV applications* [Xu+21] found that an FPGA is substantially more energy-efficient than an embedded GPU, with almost the same accuracy, even though with a lower frame rate. The FPGA hardware used by all teams was a PYNQ-Z1 board with a SoC Zynq-7020, the same as ours.

Conclusion: these papers provide valuable work towards our goal, but each achieves partial aspects only.

3 Selection of Suitable Frameworks

Many frameworks exist which promise to support implementing a neural network on an FPGA. When we started our work, we did a little survey on those available at that time. [HSB+20, Chap. 4] Making use of them turned out to be more difficult than expected. In particular, scarce documentation was a problem with several frameworks. Other frameworks did not support our hardware platform, the PYNQ-Z1 board. Two tools did generate code, but it would have required substantial further work to actually make the code run on our board. In this first round, we therefore didn't use any of these frameworks.

Instead, in this first round we manually accelerated the execution of a very simple neural network in a Python environment. The PYNQ-Z1 board provides a user friendly Python development environment. However, execution there is far too slow for our purposes. We manually implemented the neural network on the FPGA and devised a tool chain that feeds data from the Python environment into the FPGA and returns the results back to the Python environment. We used Keras/Tensorflow in the IDE PyCharm to train our neural network, written in Python, in order to obtain the weight parameters for our implementation of the network. We did this offline on standard PC hardware. We then made the inference work on the less powerful PYNQ-Z1 board. We provided the input to our neural network using Python. The PYNQ provides the concept of so-called Python overlays. We wrote the custom driver for such a Python overlay in order to tie the Python code to the AXI bus connecting the CPU and the FPGA of the SoC. We also wrote an IP core in VHDL, consisting of the components of our neural network, suitably connected, interfacing to the AXI bus. We configured the network with the weight parameters from the training. A comparison with a network in Python showed an acceleration of several orders of magnitude. The implementation effort for this first FPGA solution was substantial; however, implementing another network could reuse most of the code except the actual wiring of the network. [HSB+20]

In the second round, we re-evaluated some frameworks. We decided to use the FINN framework [Umu+17], since its documentation had improved considerably at that time. [MKB+21]

We eventually stayed with the FINN framework during our further work, which we describe in the next chapter.

4 Prototype Implementations

This section reports how we implemented the optimizations discussed in Sect. 1.3, and what we learned from this. We applied the optimizations step by step in several rounds. Each subsection reports on one of these iterations.

4.1 Tracking a Moving Traffic Sign

In this round, we approached a non-trivial visual object recognition task, and we added our robot vehicle (shown in Fig. 1) as a sample application. [MKB+21] We used the FINN framework [Umu+17] to implement the neural network for visual object recognition on the FPGA automatically. The results are fed back to the CPU of the SoC, where we implemented a simple tracking algorithm in C/C++: the vehicle turns and follows the object recognized. For simplicity, at this point we used one of the pre-trained neural networks that come with the FINN framework. Therefore, the vehicle tracks and follows a particular traffic sign, of which we moved around a decommissioned life-size example in the open air on a public parking space.

The resulting system was indeed able to successfully perform this driving task in real-time.

The FPGA hardware is by far too restricted to accomodate a neural network that can process an entire video frame. Therefore, we split each frame into several tiles and fed them into the FPGA sequentially. We used tiles of different size classes, in order to transform both small and large regions with an object into tile-filling images. For simplicity in this round, we implemented the tiling using the OpenCV framework on the CPU. Each frame of size 640×480 pixels was cut into 999 regions of 64×64 pixels and 108 regions of 96×96 pixels. These regions were transformed into 1107 tiles of size 32×32 pixels. The resulting frame rate was 2 fps.

Only the actual inference executed on the FPGA, anything else still executed on the CPU of the SoC. So there was ample room for further optimizations.

4.2 Tracking a Moving Person Making Gestures

In our third round, we trained the neural network ourselves, which allowed for a more realistic tracking task. [ACB+21] The task of the vehicle now was to track a

person; additionally, this person can command the vehicle by simple arm gestures (“start”/“stop”/“no specific pose”, compare Fig. 2). We produced a set of 300 images of persons showing these gestures, and we added 120 random images without a person from a Google web search.

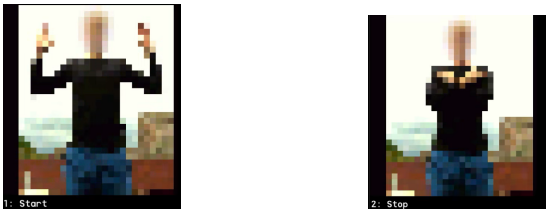

Fig. 2 Arm gestures for commanding the vehicle, at a resolution of 32×32 pixels.

We re-evaluated our choice of a board with a SoC of similar processing capabilities as a radiation-hard FPGA suitable for space. We stuck with our PYNQ-Z1 board, but the Avnet Ultra96-V2 board turned out to be a slightly more powerful alternative still in range. [ACB+21, Chap 3.3]

We re-evaluated our choice of a framework. Many candidates dropped out. This was because they aim at larger FPGAs, have not published their code, provide documentation too scarce, or are not supported anymore. As a result, we stuck with the FINN framework. [ACB+21, Chap. 5]

We considered several publicly available neural network models for visual object recognition. We chose the CNV_1W1A model, because the FINN framework provides additional support for this model. [ACB+21, Chap. 4.5] The model is written using Brevitas, a PyTorch library. Brevitas can export the trained model in the ONNX exchange format, which can be fed directly into the FINN compiler. More details on our tool chain are in our technical report [ACB+21].

We had a first look at alternatives to tiling an image. A Region Proposal Network (RPN) is an interesting idea. A tentative experiment showed the need for additional effort not feasible for us at that time. We also considered a Single Shot Detector (SSD) briefly. [ACB+21, Chap. 6] In hindsight, this would have been feasible only when adding off-chip memory to the FPGA. When the task can be solved on this FPGA by 1000 iterations with a network for a 32×32 pixel image, then no network for a full size frame 640×480 pixel image can fit on this FPGA to do the task in a single shot. Using the FPGA together with off-chip memory while applying some other kind of iteration might help to realize an SSD network. We did not investigate this further, though. The bandwidth of the off-chip memory link would certainly be a relevant parameter.

Our implementation used 70 % of the block RAM and 46 % of the look-up-tables of the FPGA. Unfortunately, our implementation had some bugs then, found only after this round, which prevented a successful evaluation of the detection accuracy in a real-life test.

4.3 Tiling in Hardware

The thesis of Müller [Mül21], one of our collaborators, improves the efficiency of the tiling by realizing it on the FPGA, too, instead of in software on the CPU of the SoC. Müller's solution allows for a highly flexible configuration of the tiling and scaling at run-time. It uses the PYNQ-Z1 board, too.

Müller integrated his tiling implementation with the implemenation of an example CNV neural network from the FINN framework, which was trained with the CIFAR10 data set. All variants of the integrated solution were small enough to fit into the FPGA and its block RAM. Tests proved that the system successfully recognizes differently scaled and differently placed images from the CIFAR10 data set in an input video stream. The hardware solution on the 100 MHz FGPA was up to 4 times faster than a comparable solution in software on the 650 MHz CPU. And Müller identified several aspects of his hardware solution that would allow for further optimization. The development of the solution was more complex than one in software; a large share of this was due to the voluminous AXI4 interface specification.

The integration of this hardware tiling into our robot vehicle system needs yet to be done.

4.4 Tracking a Moving Person and Only Then Looking for Gestures

In a further round, we separated locating a person in a frame and determining the person's gesture, in order to reduce the number of tiles required; furthermore, we improved the quality of our implementation of the robot vehicle system. [ACB+22] We re-trained the CNV neural network from the previous round, now for detecting a person. For this, we used suitable parts of the huge standard COCO data set. We implemented the inference on the same PYNQ-Z1 board as before. Our implementation used 70 % of the block RAM and 44 % of the look-up-tables of the FPGA; this should easily allow to add the above tiling in hardware.

When using 24 tiles only, the system achieved 10–12 fps; it could follow a person in up to 2 m distance. When using 396 tiles, the system could follow a person in nearly 6 m distance; however, the frame rate dropped to 2–3 fps. Significant imperfections of the mechanical drive train let the car lose its target more often in the latter case. An overall optimum appeared to be at 138 tiles and 6 fps.

Because of time restrictions, we could not add the recognition of gestures anymore. For a start, it could be implemented in software, since it is used only once per frame. Thus, it is not that time critical.

5 Summary and Outlook

We reported on experiences with optimizations that enable neural network edge computing on a small robot vehicle.

The basic approach worked well, doing the cheaper inference only on the vehicle, and employing a field-programmable gate array (FPGA) with its massively parallel resources for a neural network. Also, using a quantisized version of a neural network matching the properties of the FPGA well. All of this was expected from recent literature already.

Fitting a realistic visual recognition task into the resources of a small FPGA suitable for space required an area/speed trade-off; that is, processing an input frame in several iterations. We kept the neural network entirely on-chip and avoided the potential memory bandwidth bottleneck of a solution that permanently loads parts of a large neural network into the small FPGA. Instead, we sequentialized processing the input frame by cutting it into several tiles. Tiling is a relatively cheap operation. And Müller proved that tiling can be performed on the very same FPGA, too, thus speeding it up further.

Since we had to sequentialize the processing partially anyway, we split up the visual object recognition task into two stages, using two separate neural networks. The first stage identifies the region of interest approximately, using large and thus few tiles. The second stage looks closely at the single tile containing the region of interest; thus being not that time critical.

A single-shot neural network approach would have required another way of sequentialization; and we suspect that it would have become difficult to avoid a memory bandwidth bottleneck there. Of course, the advantage of our two-staged approach partially hinges on the property of our application that there is only one region of interest in a frame at a time.

Future work should integrate hardware tiling into our robot vehicle system. Furthermore, it should investigate the advantages that a neural network brings which is tailored more specific to the application.

References

[ACB+21] Philipp Altnickel, Tuncer Catalkaya, Jan Bredereke, et al. *Gesten- und Objekterkennung durch schwache FPGAs in autonomen Fahrzeugen mittels neuronaler Netze*. German. Tech. rep. City Univ. of Applied Sciences Bremen, Germany, Sept. 1, 2021. 153 pp.

[ACB+22] Philipp Altnickel, Ferhat Cansu, Jan Bredereke, et al. *Personenerkennung durch schwache FPGAs in autonomen Fahrzeugen mittels Neuronaler Netze*. German. Tech. rep. City Univ. of Applied Sciences Bremen, Germany, Mar. 1, 2022. 57 pp.

[Blo+18] Michaela Blott et al. "FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks". In: *ACM Transactions on Reconfigurable Technology and Systems* 11.3 (Dec. 2018). DOI: 10.1145/3242897.

[HSB+20] Colin von Huth, Marvin Soldin, Jan Bredereke, et al. *Bericht zum Projekt "Neuronale Netze auf strahlungstoleranten FPGAs für die Raum-*

fahrt". German. Tech. rep. City Univ. of Applied Sciences Bremen, Germany, Feb. 14, 2020. 88 pp.

[JEB18] Petar Jokic, Stephane Emery, and Luca Benini. "BinaryEye: A 20 kfps Streaming Camera System on FPGA with Real-Time On-Device Image Recognition Using Binary Neural Networks". In: *2018 IEEE 13th Int'l Symp. on Industrial Embedded Systems (SIES)*. (Graz, Austria). June 6–8, 2018. DOI: 10.1109/SIES.2018.8442108.

[Man+18] Jacob Manning et al. "Machine-Learning Space Applications on SmallSat Platforms with TensorFlow". In: *Proc. of AIAA/USU Conference on Small Satellites (SmallSat)*. (Logan, UT, USA, Aug. 4–9, 2018). 2018.

[MKB+21] Felix Müller, Niklas Krekel, Jan Bredereke, et al. *Applying Binarized Neural Networks on FPGAs to an Autonomous Driving Problem*. Tech. rep. City Univ. of Applied Sciences Bremen, Germany, Mar. 31, 2021. 50 pp.

[Mül21] Felix Müller. "Dynamisches Tiling auf schwachen FPGAs zur Objekterkennung mithilfe kleiner neuronaler Netze". German. Bachelorthesis. City Univ. of Applied Sciences Bremen, Germany, June 23, 2021.

[Rof+20] Seth Roffe et al. "CASPR: Autonomous Sensor Processing Experiment for STP-H7". In: *Small Satellite Conf. 2020*. 2020.

[Umu+17] Umuroglu et al. "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. (Monterey, CA, USA). FPGA '17. ACM, Feb. 22–24, 2017, pp. 65–74. DOI: 10.1145/3020078.3021744.

[Xu+21] Xiaowei Xu et al. "DAC-SDC Low Power Object Detection Challenge for UAV Applications". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.2 (Feb. 2021), pp. 392–403. DOI: 10.1109/TPAMI.2019.2932429.