Diss. ETH No. 27872

Efficient Neural Network Acceleration for Low Power Edge Processing Devices

A thesis submitted to attain the degree of
DOCTOR OF SCIENCES of ETH ZURICH
(Dr. sc. ETH Zurich)

presented by

PETAR JOKIC
MSc ETH EEIT, ETH Zurich
born on May 29th, 1989
citizen of Gondiswil BE, Switzerland

accepted on the recommendation of

Prof. Dr. Luca Benini, examiner
Prof. Dr. Shih-Chii Liu, co-examiner
Mr. Stephane Emery, supervisor

2021

Acknowledgements

I would like to thank Luca Benini and Stephane Emery for offering me the opportunity to conduct research in this fascinating field and pursue a doctorate at CSEM Zurich. The unique positioning of CSEM as a research institute, acting as a bridge between academia and industry, provided me with useful insights from both worlds and helped shaping my research. Luca's enthusiasm for the field was very motivating and his broad know-how helped distilling many of the ideas. I am thankful for his excellent guidance and constructive feedback, sharing his profound knowledge and experience, particularly on the academic side. I would like to thank Stephane for giving me the chance to work in CSEM's System-on-Chip group and for the trust he put in me to work on various internal and industrial projects while offering me the great freedom to pursue my own ideas. The direct contact with industrial customers helped to understand their needs to identify useful research focus areas and new ideas. Special thanks go to Michele Magno, who supervised my Master thesis at ETH and motivated me to pursue a doctorate. His excitement for smart devices has been a great motivation for many of my fellow students. He also introduced me to the art of writing academic publications and helped me submitting my first paper during the Master thesis. I would also like to thank Shih-Chii Liu, the co-examiner of this thesis, for her interest in this work.

There are many people that I had the pleasure to work with throughout the journey of the doctorate. I would like to thank the whole IIS team at ETH Zurich, especially the former colleagues Renzo Andri and Lukas Cavigelli, who helped me getting to know the basics of being a PhD candidate. Unfortunately, the Corona pandemic

limited the exchange a little in the final years, but in return helped to focus on the started projects. The work at CSEM enabled me to meet many experts from a diverse set of fields across the various sites of the company in Switzerland. I could benefit from their vast industrial experience and would like to thank all my colleagues for the many interesting and fruitful discussions throughout the years. Especially, I would like to thank Bernhard Schaffer, Erfan Azarkhish, Komail Badami, Andrea Bonetti, Regis Cattenoz, Virginie Moser, Engin Turetken, and Pascal Nussbaum for their close collaboration in various related projects. But also many other colleagues in the Zurich and Neuchatel offices have always been great technical supporters and social contacts, namely Stephane Devise, Hans-Rudolf Graf, Pierre-Francois Ruedi, Andrea Dunbar, Themis Mavrogordatos, Benjamin Sporrer, Jian Deng, Yingyun Zha, Loic Zhand, Jean-Luc Nagel, Claude Arm, Amrith Sukumaran, and Matthias Krieger. On the administrative side, I would like to thank Barbara Jörg-Clavien for her kind assistance with the bureaucratic burdens, as well as Daniel Sigg for always finding a solution to our IT problems.

Finally, I thank my family and all friends for their support during these years. I am particularly grateful for the support of my parents, Marianne and Jovan, who enabled my university career and motivated me throughout this journey.

Abstract

Within the past decade, neural networks have become the state-of-the-art algorithms for many computer vision applications, utilizing machine learning approaches to model the underlying complex data analysis tasks. The proliferation of miniaturized and smart sensing devices, exploiting neural networks in battery-powered wearables and internet-of-things (IoT) applications, introduced the need for energy-efficient machine learning hardware accelerators. These systems can analyze data directly on board, so-called edge processing, enabling to preserve privacy and reduce latency as well as processing energy, compared to external data analysis using cloud computing.

This thesis investigates multiple optimization approaches to improve the efficiency of edge processing devices and expand their range of applications. It first provides an overview and quantitative comparison of existing optimization techniques, illustrating their broad range, from low-level hardware optimizations up to high-level algorithm co-design and mapping improvements. We then evaluate edge processing capabilities on high-speed cameras, enabling to reduce power consumption by 3×, and provide a tool for automated mapping of trained networks to ease the efficient implementation on cameras with programmable logic. We further propose an efficient memory allocation technique for convolutional neural network (CNN) accelerators, enabling memory savings of up to 48.8% compared to traditional mapping. To equip even tiny edge processing devices with machine learning-based data analysis capabilities, the strict power constraints imposed by their limited battery capacities must be fulfilled. Thus, we present a new hardware accelerator architecture that supports hierarchical processing, facilitating such sub-mW power budgets. Its

dual-engine accelerator enables scalable performance, which is demonstrated in multiple scenarios, reporting 0.41mW power consumption for running face detection and recognition. Exploiting the manufactured chip, we develop an end-to-end face recognition application on a battery-less credit card-sized platform, demonstrating self-sustainable machine learning edge processing using solar energy harvesting.

Zusammenfassung

Neuronale Netzwerke haben sich im vergangenen Jahrzehnt als die leistungsstärksten Algorithmen für computergestützte Bildanalysen etabliert. Maschinelles Lernen ermöglicht es dabei, diese komplexen Analysen automatisch zu modellieren. Die schnelle Verbreitung von miniaturisierten und intelligenten batteriebetriebenen Geräten, welche Neuronale Netzwerke für Anwendungen in Wearables und dem Internet der Dinge (IoT) benutzen, hat die Nachfrage nach energieeffizienten Hardware-Beschleunigern stark erhöht. Diese ermöglichen es, Daten direkt auf dem Gerät auszuwerten, sogenanntes Edge Processing, und somit gleichzeitig die Privatsphäre zu schützen und sowohl die Verarbeitungszeit als auch den Energiebedarf gegenüber externer Berechnung in Cloud Servern zu reduzieren.

Diese Dissertation präsentiert eine Reihe von Optimierungsansätzen, um die Effizienz von Edge Processing-Geräten zu verbessern und dadurch deren Einsatzgebiet zu erweitern. Zuerst wird eine Übersicht mit einem quantitativen Vergleich von existierenden Methoden präsentiert, welche das breite Spektrum von Optimierungen aufzeigt: Diese reichen von hardwarenahen Anpassungen bis zu Verbesserungen durch koordinierte Hardware/Software-Entwicklung. Wir evaluieren den Einsatz von Edge Processing in Hochgeschwindigkeitskameras und zeigen auf, dass der Leistungsverbrauch dadurch um das 3-fache gesenkt werden kann. Ausserdem entwickeln wir ein Programm, welches die effiziente Implementierung von trainierten Netzwerken in Kameras mit programmierbarer Logik automatisiert und damit den Zugang für Laien erleichtert. Zudem präsentieren wir eine effiziente Speicherverwaltungs-Methode für Hardware-Beschleuniger von sogenannten Convolutional Neural Networks (CNNs), welche

oft für Bildanalysen verwendet werden. Diese Methode erlaubt es, den Speicherbedarf gegenüber herkömmlicher Speicherverwaltung um bis zu 48.8% zu senken. Die Rechenleistung von miniaturisierten Edge Processing-Geräten ist stark durch ihre Batteriekapazität eingeschränkt. Um dennoch Datenanalysen basierend auf maschinellem Lernen in solche Geräte einbetten zu können, haben wir eine neue Hardware-Beschleuniger-Architektur entwickelt. Diese ermöglicht eine hierarchische Berechnung der Algorithmen und kann dadurch in Geräten mit einem Leistungsverbrauch von weniger als 1mW eingesetzt werden. Der integrierte Zweikomponenten-Beschleuniger erlaubt es, die Rechenleistung zu skalieren, was in verschiedenen Szenarien demonstriert wird und es ermöglicht den Leistungsverbrauch für die Berechnung von Gesichtsdetektierung und -erkennung auf 0.41mW zu reduzieren. Zudem benutzen wir den entwickelten Chip, um auf einer batterielosen und kreditkartengrossen Plattform eine Gesichtserkennungs-Anwendung zu implementieren. Das Edge Processing System wird dabei lediglich durch ein kleines Solarmodul betrieben und demonstriert damit energieautarken Betrieb von Neuronalen Netzwerken.

Contents

Acknowledgements **iii**

Abstract **v**

Zusammenfassung **vii**

1 Introduction **1**
1.1 Thesis Overview 5
1.2 Contributions and Publications 9

2 Neural Network Accelerator Construction Kit **13**
2.1 Introduction 14
2.2 Related Work 15
2.3 Construction Kit Foreword 17
2.3.1 System-Level View 18
2.3.2 Meaningful Performance Indicators and Metrics 19
2.3.3 Neural Network Notations 20
2.4 Hardware Architectures 20
2.4.1 Temporal Architecture 21
2.4.2 Spatial Architecture 22
2.4.3 In-Memory Computing 22
2.5 Technology 24
2.5.1 Semiconductor Process Technology 24
2.5.2 Power Management 26
2.5.3 Memory 28
2.6 Dataflow and Control Optimizations 34

2.6.1 Dataflow and Blocking 34
2.6.2 Compiler 35
2.6.3 Early Data Reduction 36
2.6.4 Selective Execution and Early Abortion 37
2.7 Data Handling Optimizations 39
2.7.1 Efficient Memory Utilization 39
2.7.2 Data Compression 39
2.8 Computation Optimizations 40
2.8.1 Optimized Convolution Implementations 40
2.8.2 Sparsity Exploitation and Pruning 42
2.8.3 Data Reuse 43
2.8.4 Hardware/Software Co-Optimization 44
2.8.5 Approximate Computing 48
2.8.6 Non-Conventional Arithmetic 49
2.8.7 Mixed Signal Arithmetic 50
2.8.8 Arithmetic Implementations 51
2.9 Quantitative Comparison 52
2.10 Conclusion 53

3 FPGA-Based Binary Neural Network Implementation 57
3.1 Introduction 58
3.2 Related Work 61
3.3 Implementation on FPGA-Based High-Speed Camera 63
3.3.1 High-Speed Camera System 66
3.3.2 BNN Image Classifier 66
3.4 Experiments and Results 68
3.5 Conclusion 72

4 Automated Neural Network Mapping on FPGA-Based Cameras 75
4.1 Introduction 76
4.2 Background 78
4.2.1 Quantized Neural Networks 78
4.2.2 Streaming versus Layer-Wise Processing 80
4.3 Related Work 80
4.4 Neural Network Mapping Framework 84
4.4.1 Network Extraction 87
4.4.2 Mapping to Hardware Functions 87

4.4.3 Allocating and Balancing Resources 88
4.4.4 HLS Compilation to Hardware Representation 91
4.5 Streaming Processing Architecture 91
4.5.1 Sliding Window Generator 94
4.5.2 Processing Element 98
4.5.3 Precision 98
4.5.4 Portability 100
4.6 Experiments and Results 101
4.6.1 640x640 Pixel Optical Character Detection and Recognition 101
4.6.2 640x640 Pixel Face Detection 102
4.6.3 Result Discussion 103
4.7 Extensions and Limitations 104
4.8 Conclusion 106

5 Improving Memory Utilization for Convolutional Neural Network Accelerators 107
5.1 Introduction 108
5.2 Improving CNN Memory Utilization 110
5.2.1 CNN Data Access Pattern 110
5.2.2 Model for Optimized Memory Utilization . . . 112
5.3 Experiments and Results 114
5.4 Related Work 117
5.4.1 Conclusion 118

6 Dual-Engine Machine Learning Inference System-on-Chip for Sub-mW Face-Analysis at the Edge 121
6.1 Introduction 122
6.2 System-on-Chip 124
6.3 BDT and CNN Machine Learning Accelerators 124
6.3.1 Binary Decision Tree Accelerator 125
6.3.2 Convolutional Neural Network Accelerator . . . 126
6.4 Scalable Dual-Accelerator Operation 127
6.5 Implementation Results 131
6.6 Neural Network Mapping Flow 131
6.7 Conclusion 135

7 Battery-Less Face Recognition at the Extreme Edge 139
7.1 Introduction 140
7.2 System Implementation 142
7.2.1 Image Acquisition 142
7.2.2 Control and ML Processing 143
7.2.3 Display 144
7.2.4 Power Management and Energy Harvesting . . 145
7.3 Computer Vision Application 146
7.3.1 ML Algorithm 146
7.3.2 Task Scheduling 148
7.4 Experiments and Results 148
7.5 Conclusion 151

8 Summary and Conclusion 153
8.1 Overview of the Main Results 154
8.2 Outlook 156

A Notations and Acronyms 161
Operators 161
Acronyms 162

Bibliography 165

Curriculum Vitae 207

Chapter 1

Introduction

Nearly ten years ago, neural networks (NNs) have surpassed human-level performance in various image classification tasks like object recognition [1] and face identification [2]. This generated a strong momentum for machine learning-based algorithms, leading to a widespread proliferation in various fields, covering, among others, medical image analysis [3], speech recognition [4], economic trend analysis, autonomous driving, and face verification [5].

In contrast to labor-intensive manual algorithm development, requiring expert knowledge of the field, machine learning (ML) enables laymen to train accurate algorithms, given that sufficient training data and user-friendly training frameworks are accessible [6]. The availability of large, labeled, and public data sets, like ImageNet [7] or Google's speech command dataset [8], and effective training tools, like Caffe [9] or TensorFlow [10], further boosted the research activities, creating a positive feedback loop through a growing community that further advocated the use of ML.

More recently, the ML trend has reached internet of things (IoT) and edge platforms in the consumer electronics market, a multi-billion USD business [11], related to extreme technology investments. IoT devices are becoming smarter by leveraging ML capabilities onboard the device, so-called edge processing. This helps better preserving privacy by avoiding transmitting sensitive data through the Internet [12], as it is required for server-based (cloud) processing. Additionally,

this offers lower latency and power consumption because costly communication activities can be reduced to a minimum. Thus, ML-based edge processing devices are targeting more intelligence in a smaller form factor with longer battery lifetimes, enabling a new era of smart devices. Fig. 1.1 illustrates the power regions for cloud- to edge-processing systems, indicating the extreme efficiency requirements for operating complex ML tasks, as they are currently supported by mobile platforms, within the mW-power region. While graphics processing units (GPUs) cover the cloud-processing region, optimized application-specific integrated circuits (ASICs) are usually employed for the power-constrained mobile and edge domains.

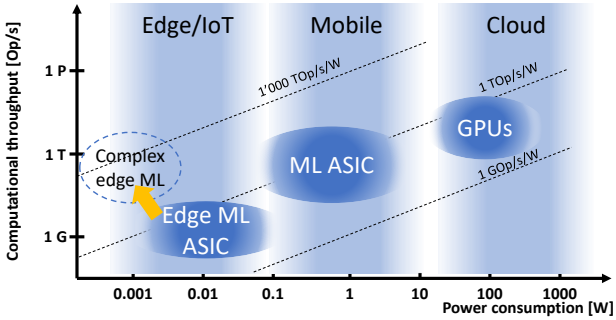

Figure 1.1: Power and processing performance of ML hardware for cloud to edge applications.

Neural Networks

Biological NNs had already been researched for multiple decades when artificial NNs have gained attention [13], attempting to mimic the underlying biological structures to develop powerful data analysis algorithms. The simplest structures consist of neurons that receive input activation signals $a_{i,n}$ through connections from (N_{in}) neighboring neurons, which are weighed with a scaling factor $w_{n,k}$, accumulated and passed through an activation function f_{act}, as shown in Equation 1.1. This results in a vector dot product $a \cdot w$, or a set of multiply and

accumulate (MAC) operations, which is often used with a rectified linear (unit) (ReLU) as activation function.

$$a_{o,n} = f_{act}\left(\sum_{k=1}^{N_{in}} a_{i,k} \cdot w_{n,k}\right) \mid f_{act,ReLU}(x) = max(0,x) \qquad (1.1)$$

To represent complex analysis functions, it became clear that thousands of neurons with millions of connections were required, usually grouped into layers, that are only connected to their neighbors, forming so-called multi-layer perceptrons [13], shown in Fig. 1.2 a). The intermediate layers, connecting the input to the output layer, extract features (thus, called *feature maps*), requiring large memories to store their activations (features). Inter-layer connectivity differs across layer-types and determines the computational effort as well as the number of weight parameters: in fully-connected (FC) layers, each neuron of a layer receives inputs from all neuron outputs in the previous layer, while convolutional neural networks (CNNs) limit the inputs to a small connected region on the previous layer and share their weights with neighboring neurons, largely reducing the number of connections and parameters. Fig. 1.2 b) visualizes the structure of a generic CNN layer, as it is common for image processing, with an input feature map *in* of size $X_{in} \cdot Y_{in} \cdot C_{in}$ being convolved with C_{out} kernels k of size $K_x \cdot K_y \cdot C_{in}$ and thus resulting in an output feature map *out* of size $X_{out} \cdot Y_{out} \cdot C_{out}$. Thus, FC layers can be considered a subset of CNNs, with kernel sizes equal to the input size.

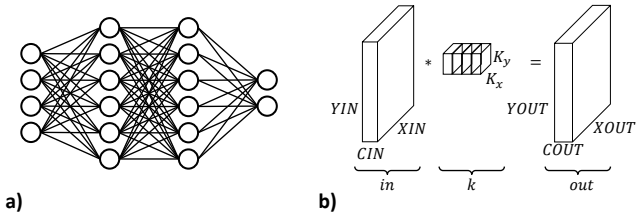

Figure 1.2: Visualization of typical NN layer structures: a) multi-layer perceptron, b) CNN.

Many other layer configurations based on these simple ones have been proposed in the following, introducing connections that allow bypassing a set of layers in residual networks (ResNets) [14], adding recurrent connections that enable to keep a state over time in recurrent neural networks (RNNs) [15], or various optimizations of the underlying computations, like depth-wise separable convolutions in MobileNets [16].

Because all networks rely on large feature maps and data-intensive computations, their sudden success over the past decade can be partially attributed to the appearance of powerful compute hardware that enabled timely training iterations and low-latency inference for exploring deeper and more complex networks, allowing to reach unprecedented algorithmic accuracy.

Edge processing

As illustrated above, executing NNs is computation- and memory-intensive, challenging the processing hardware. Storing data requires large (and costly) chip area for instantiating memories or external storage with significantly higher access energy. The ever increasing network complexity, which mainly drives accuracy improvements, is reflected in more arithmetic operations, demanding higher computational throughput to sustain the established algorithm inference rates required by the applications.

Powerful GPUs and similar server-grade hardware provide highly parallelized arithmetic, making them well suited for running large networks at high throughput. This comes at the cost of high power consumption and bulky dimensions, strongly limiting their deployment for miniaturized and energy-constrained edge devices. Cloud processing provides large servers with virtually unlimited power budgets, offering energy-intensive computation capabilities. To boost the limited computational capacity of IoT devices, cloud-offloading was proposed [17], transmitting the raw data through a communication network for processing in cloud servers. However, the high power consumption of communication interfaces can often not be supplied by battery-powered IoT devices, motivating edge computing [18], processing and analyzing sensor input data directly on-board, at the edge (of the network). Efficient hardware accelerators in ASICs

have therefore been intensively investigated in various works [19–21]. However, the strict energy constraints of miniaturized smart devices still limit the deployment of complex ML-based algorithms for such systems, motivating further research towards efficient ML for low power edge processing devices.

ML-based processing is expected to affect many aspects of our daily life in the near future. Some sensory devices already implement some degree of edge processing [22, 23] or demonstrated the feasibility of creating smart tiny sensing systems [24]. With more efficient ML processing becoming available, a wider range of miniaturized smart applications will emerge, striving for battery-less operation and pervasive intelligence.

1.1 Thesis Overview

This thesis investigates and demonstrates multiple techniques for enabling efficient ML inference processing in embedded systems at the edge. It targets smart power-constrained devices, like IoT nodes, smart sensors, and self-sustainable miniaturized processing platforms, that analyze sensory data on board, limited by the power budget of their small batteries or energy harvesting circuits. Motivated by the data-intensive nature of image and video contents, already dominating the worldwide Internet traffic with an estimated 82% share in 2020 [25], we focus on NN-based image analysis. The main challenges discussed in this work are data handling techniques, efficient processing architectures, and mapping strategies to enable machine learning inference in low power edge processing devices. We investigate various optimizations to lower the power consumption of ML applications, striving towards tiny edge processing platforms with extended battery lifetimes. Fig. 1.3 shows an overview of the covered topics along with the chapter numbers and illustrates their impact on the edge ML implementation flow.

Neural networks and other ML-related fields attracted a lot of attention and consequently experienced a surge of related publications discussing the field. The vast amounts of ML hardware architectures and optimization approaches proposed in the literature ask for a structured overview that allows comparing ideas, removing redundancies,

and identifying promising approaches to build on. To make research efforts more efficient and target-oriented, Chapter 2 surveys the field of low power NN accelerators and provides a quantitative comparison of proposed optimization techniques, targeting efficient edge processing. It discusses hardware architectures, technological aspects, dataflow structures, efficient data handling, and various computation approaches. Reported optimization effects range from up to 10'000× memory savings to 33× energy reductions, providing chip designers an overview of design choices for implementing efficient low power NN accelerators.

While video streams dominate the worldwide Internet traffic [25], significant portions carry redundant data with low information content. In the example of simple video-based surveillance applications for home intrusion detection, video data are streamed from private households to processing systems that trigger an alarm if unauthorized people are detected. Considering the low information content, the raw image data transmission from the camera could be replaced by a single flag bit, indicating whether there is an intruder or not. This would reduce the data traffic in the network and additionally improve privacy, as cameras stream out images from private rooms through a public network. Higher frame rates, as they are produced by high-speed cameras in quality control systems, further aggravate the problem by massively increasing the data bandwidth. Chapter 3 investigates the usage of binary neural networks (BNNs) to classify images in real-time onboard a 20kFPS high-speed camera and only transmit the resulting classifications, allowing to reduce the data-rate by 980x. The BNN is instantiated on the camera's field-programmable gate array (FPGA) using high-level synthesis (HLS) tools, allowing to efficiently implement binary MAC operations using programmable (XNOR) logic. Compared to external image processing, the on-board analysis reduces the energy per frame by 3×.

A follow-up work is discussed in Chapter 4, further simplifying the NN deployment on FPGA-based cameras by extending the HLS approach from the previous chapter to an automated end-to-end mapping framework that takes a trained network as input and generates an efficient FPGA implementation for the camera. HLS provides software developers with limited hardware design expertise a tool to develop applications using high-level C++ language. Our end-to-end

framework enables NN application-developers to directly map their trained network onto the device for edge processing, further reducing the required hardware knowledge of the users. To enable energy and memory savings, arbitrary parameter and activation precision are supported. Experimental results demonstrate the functionality of the framework on various CNNs mapped to the FPGA-based camera, reporting throughputs of up to 337GOPS. The automatic mapping is reused in parts within the NN mapping flow for the ML system-on-chip (SoC), presented in Chapter 6.

The large memory footprint required to map NNs on hardware platforms can be observed on the memory-dominated chip floor-plan in many ML accelerators. Excessive (memory) area directly translates into additional manufacturing cost and increases the data access energy, which grows with the memory size and thus influences the overall power consumption. Conversely, the network size (complexity) has a strong influence on its algorithmic performance, generally enabling higher accuracy for larger memory (network) sizes. In Chapter 5 we describe a technique for more efficient memory utilization in CNN accelerators with layer-wise processing, allowing to either reduce the on-chip memory requirements for a given target network or to increase the network size (complexity) for a fixed memory capacity. Evaluated on multiple image processing networks, memory savings of up to 48.8%, compared to standard ping-pong buffering, are achieved. The technique overlaps activation regions of neighboring layers, allowing to overwrite input activations as soon as they are no longer needed, reducing the overall activation memory needs. Through simple adaptions of the memory mapping for activations, this method can be implemented in any programmable accelerator.

In Chapter 6 we present a hierarchical dual-engine ML SoC for low power face analysis at the edge. The end-to-end processing chip is implemented and tested on a 22nm process, achieving state-of-the-art performance for running face detection and recognition at 0.41mW power consumption. Various sensor peripherals, coupled with a 32-bit RISC-V microcontroller, offer flexible operation, while the 1.2MB on-chip static random-access memory (SRAM) provides sufficient memory for face analysis applications, exploiting the CNN mapping technique from Chapter 5. To achieve a low power consumption, it provides hierarchical processing capabilities using a dual-engine

accelerator, supporting binary decision trees (BDTs) and CNNs. This allows to implement simple (face) detection tasks on the dynamically scalable BDT engine, which can then trigger the evaluation of a more complex CNN (e.g. for face recognition). Combined, the SoC offers to process complex CNNs while consuming very low power on average through dynamic complexity scaling.

The low power edge processing capabilities of the presented ML SoC are exploited in Chapter 7, demonstrating the feasibility of implementing ML processing on tiny, self-sustainable, and battery-less platforms. CNN-based user identification, running on the presented ML SoC, is implemented on a credit card-sized platform with solar energy harvesting, enabling self-sustaining operation at 1 frame-per-second. Measurement results report the energy for each execution phase, identifying dominant contributions that can be investigated in future research.

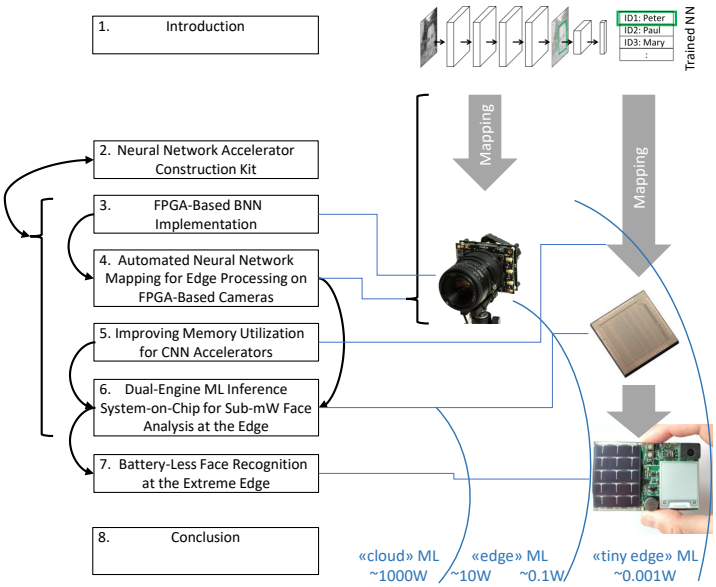

Figure 1.3: Thesis overview.

1.2 Contributions and Publications

As described above, this thesis investigates multiple aspects of efficient ML implementation in edge processing devices, by 1) identifying effective optimization techniques in the literature, 2) proposing quantized implementations and improved mapping strategies, and 3) providing specialized hardware blocks and system designs. The main contributions to this research field are summarized below:

1. A quantitative survey of existing ML accelerator optimizations, allowing to estimate their effects based on five key performance indicators and thus ease the ML accelerator design process.

2. An optimized BNN implementation for FPGA-based edge processing high-speed cameras, enabling early data reduction to reduce communication interface requirements and improve latency.

3. An automated end-to-end NN mapping framework that efficiently implements a trained network on an FPGA-based edge processing camera to simplify NN deployment for non-expert users.

4. A method for improving memory utilization in CNN accelerators which allows minimizing activation memory space.

5. A system-on-chip hardware architecture for hierarchical processing of face analysis tasks using dual-engine BDT/CNN acceleration to enable sub-mW edge processing.

6. A design and an evaluation of a solar-powered battery-less edge ML platform, that demonstrates miniaturized and self-sustainable end-to-end image processing.

The main content of the thesis has been published in the following international conference and journal papers:

[26] **P. Jokic**, E. Azarkhish, R. Cattenoz, E. Turetken, L. Benini, and S. Emery, "A Sub-mW Dual-Engine ML Inference System-on-Chip for Complete End-to-End Face-Analysis at the Edge,"

in *Proceedings of IEEE Symposium on VLSI Circuits*, 2021.
Received the **best demo paper award** at the 2021 Symposia on VLSI Technology and Circuits.

[27] **P. Jokic**, S. Emery, and L. Benini, "Battery-Less Face Recognition at the Extreme Edge," in *Proceedings of the 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2021.
Received the **best student paper award** at the 2021 NEWCAS conference.

[28] **P. Jokic**, S. Emery, and L. Benini, "Improving Memory Utilization in Convolutional Neural Network Accelerators," *IEEE Embedded Systems Letters (ESL)*, 2020.

[29] **P. Jokic**, E. Azarkhish, A. Bonetti, M. Pons, S. Emery, and L. Benini, "A Construction Kit for Efficient Low Power Neural Network Accelerator Designs," submitted to ACM Transactions on Embedded Computing Systems, 2021.

[30] **P. Jokic**, S. Emery, and L. Benini, "BinaryEye: A 20 kfps Streaming Camera System on FPGA with Real-Time On-Device Image Recognition Using Binary Neural Networks," in *Proceedings of the 13th International Symposium on Industrial Embedded Systems (SIES)*, 2018.

[31] **P. Jokic**, S. Emery, and L. Benini, "NN2CAM: Automated Neural Network Mapping for Multi-Precision Edge Processing on FPGA-Based Cameras," submitted to Microprocessors and Microsystems: Embedded Hardware Design (MICPRO), 2021.

Further contributions were made in the field of smart sensing at the edge but are not explicitly covered in this thesis. They originate from work conducted prior to the thesis but have strongly influenced it and are linked to the application domain of low power edge processing:

[32] **P. Jokic**, M. Magno, "Powering smart wearable systems with flexible solar energy harvesting," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[33] **P. Jokic**, G. Salvatore, M. Magno, L. Buthe, G. Troster, and L. Benini, "Self-Sustainable Smart Ring for Long Term Monitoring of Blood Oxygenation," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[34] M. Magno, G. Salvatore, **P. Jokic**, and L. Benini, "Self-Sustainable Smart Ring for Long-Term Monitoring of Blood Oxygenation," *IEEE Access*, 2019.

[35] G. Salvatore, J. Sulzle and F. Dalla Valle, G. Cantarella, F. Robotti, **P. Jokic**, S. Knobelspies, A. Daus, L. Buthe, L. Petti, N. Kirchgessner, R. Hopf, M. Magno, and G. Troster, "Biodegradable and Highly Deformable Temperature Sensors for the Internet of Things," *Advanced Functional Materials*, 2017.

Chapter 2

Neural Network Accelerator Construction Kit

Implementing embedded NN processing at the edge requires efficient hardware acceleration that combines high computational throughput with low power consumption. Driven by the rapid evolution of network architectures and their algorithmic features, accelerator designs constantly have to be adapted to support the improved functionalities. Hardware designers can refer to a myriad of accelerator implementations in the literature to evaluate and compare hardware design choices. However, the sheer number of publications and their diverse optimization directions hinder an effective assessment. Existing surveys therefore provide an overview of these works but are often limited to system-level and benchmark-specific performance metrics, making it difficult to quantitatively compare the individual effects of each utilized optimization technique. This complicates the evaluation of optimizations for new accelerator designs, slowing-down the research progress.

In contrast to previous surveys, this chapter provides a quantitative overview of NN accelerator optimization approaches that have been used in recent works and reports their individual effects on

edge processing performance. The list of optimizations and their quantitative effects are presented as a construction kit, allowing to assess the design choices for each building block individually. Reported optimizations range from up to 10'000× memory savings to 33× energy reductions, providing chip designers an overview of design choices for implementing efficient low power NN accelerators.

2.1 Introduction

NNs allow algorithm developers to implement difficult-to-model tasks, given that sufficient training data is available. However, the computational complexity of these networks is challenging the design of processing hardware. Implemented in miniaturized and battery-powered ML applications, they require high computational throughput while being constrained to limited power budgets, rendering computing efficiency a key design objective for low power ML accelerators.

The rapid progress of ML algorithm research further challenges designers to quickly adopt newly introduced network features, requiring fast development times. While FC networks and CNNs, like the well-known AlexNet [36], have dominated the field during the past decade, ResNets [37], RNNs and derivations like dense CNNs [38] have gained importance, claiming ever-improving algorithmic performance. Edge processing is increasingly used in applications where long battery lifetimes are mandatory, powering smart glasses with object detection [39], face detection [40], or hand-gesture and speech recognition [41]. Other applications employ it in smart cameras with automatic acquisition using scene classification [42], smart doorbells with face recognition [23], or tools that help blind people read texts and recognize people [22], while many more could benefit from edge ML in the future [24, 43]. An overview of ML applications that are feasible on current hardware platforms is summarized in [44], illustrating the challenge of the limited edge processing power budget (<1W).

Existing ML accelerator chips cover the application domain from ultra-low power (ULP) and low-complexity processing, implementing 18uW key-word spotting with 105kB on-chip memory [45], to high-throughput server-grade acceleration, provided by chips like Google

TPUv3 [46] or Graphcore IPU [47], consuming more than 100W. To identify relevant edge ML accelerator designs among the vast number and diversity of publications proposing efficient implementations, quantitative surveys are necessary. However, existing surveys often only provide qualitative comparisons or benchmark implementations on a system-level, obscuring the individual effects of each employed optimization technique. Comparing these optimizations is essential for motivating design choices during the development of new accelerators, currently requiring time-consuming literature research.

This chapter summarizes and, for the first time, quantitatively compares design optimizations of existing NN accelerators for tiny (<10mW) and edge (<1W) processing applications. It is presented as a construction kit, listing optimization options for each building block along with their reported quantitative effects, enabling ASIC designers to evaluate and assess optimizations for new implementations. Fig. 2.1 illustrates the covered building blocks on a generic end-to-end edge processing system and localizes optimizations within the edge ML design flow. The chapter is organized as follows: Section 2.2 presents related surveys that complement this chapter. Section 2.3 introduces basic notations used throughout the chapter, followed by an overview of architectures in Section 2.4, technological optimizations in Section 2.5, dataflow optimizations in Section 2.6, data handling optimizations in Section 2.7, computation optimizations in Section 2.8, and finally the quantitative comparison of all relevant optimizations in Section 2.9.

2.2 Related Work

Various surveys have been conducted to summarize existing NN accelerator designs and implementations, explaining their techniques, and comparing their system-level performance.

To keep track of the vast number of academic and industrial ML hardware accelerators proposed every year, a periodically updated online list [48] is provided, listing the main performance metrics for each chip to compare them in terms of power, throughput, and computational efficiency. A similar survey in paper form was presented

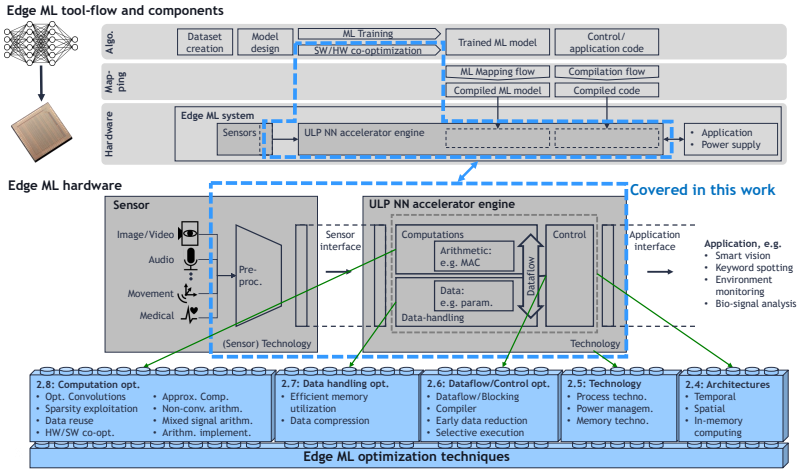

Figure 2.1: Overview of the edge ML tool flow with a summary of the discussed hardware optimizations for NN accelerators (with related section numbers of this chapter).

in 2019 [49] and updated in 2020 [20]. It lists the computational performance and precision of academic research works and commercially available devices. A survey from 2017 claims to cover the past 35 years of works in neuromorphic computing, listing more than 2600 references that accelerated the research field since the 1980s [50]. It covers models, learning approaches, and hardware ranging from analog to digital implementations, covering programmable FPGAs and custom ASICs.

Sze et al. [19] provide a thorough survey on efficient processing of deep neural networks (DNNs), covering historic aspects, common layer types, training frameworks, and popular datasets, extending their previous work [51]. The sections on hardware platforms and energy efficient dataflows are motivated in their preceding work [52], proposing edge processing for extracting meaningful information to reduce the extreme amount of data produced by the ever-increasing

number of sensors in connected devices. A similar summary is presented in [53] and a more FPGA-focused one in [54]. Another well-structured and exhaustive survey on DNN acceleration is presented in [21], covering existing hardware acceleration approaches including software optimizations, and a chapter on the security of DNN approaches and their benchmarking. Survey [55] presents a broad overview of the ML field, focusing on big data, training techniques, and applications. A similarly broad view, additionally covering the transition from modeling biological NNs to implementing artificial NNs in hardware is presented in [56], focusing on novel memories and their use-cases in the field.

In the survey of [57], various ML accelerators and processing blocks are presented and compared to their research group's own works. They list neuromorphic processors, including spiking NN engines, ranging from fully digital to fully analog computations, and discuss possible future directions. Survey [58] also discusses the architectures of selected DNN accelerators and explains their working principles and supported network architectures.

Surveys [12, 59] present an extended view on edge intelligence, covering pure edge processing and combinations with cloud processing. They discuss related optimization strategies, namely compression and early model exit.

While the listed related works give an excellent overview of existing ML accelerators and optimization techniques, none of them attempted to quantitatively compare used optimization approaches, as covered in this chapter, allowing designers to evaluate and assess optimizations for new implementations. This chapter focuses on (deep) NN inference, noting that the field of ML is much broader, containing other approaches like support vector machines, decision trees, and many others.

2.3 Construction Kit Foreword

ML accelerators often combine multiple optimization techniques, as shown in Table 2.1, summarizing a selection of relevant accelerator chips from the past six years. This complicates the assessment of individual optimizations as their effects are obscured by system-level

Table 2.1: Selected ML accelerator chips from the last 6 years

Work (Year)	**Optimizations**	**Throughput [GOPS]**	**Efficiency [TOPS/W]**
ShiDianNao (2015) [60]	Local data reuse	194 (16b)	0.606
EIE (2016) [61]	Sparsity, weight sharing, compression, zero skipping	102 (16b) (∼3'000*)	0.17 (5.0*)
Envision (2017) [62]	Multi-precision, DVFAS, body biasing	76 (4b)	10
Eyeriss (2017) [63]	Local data reuse, zero compression, zero skipping	84 (16b)	0.166
YodaNN (2018) [64]	Bin. weights, standard-cell mem., voltage scaling	1'500 (1b w., 12b a.)	1,1
UNPU (2018) [65]	Multi-precision, LUT-based bit-serial MAC (1-16b)	346/7372 (16/1b w., 16b a.)	3.1/50.6
Eyeriss v2 (2019) [66]	Local data reuse, sparsity, compressed computing	202 (8b)	0.963

*including skipped operations

benchmarking. Thus, we only add individual optimizations to our comparison and only if sufficient quantitative information is reported. We further focus on five performance indicators, namely 1) energy/ power, 2) area (cost), 3) memory size, 4) computational throughput, and 5) impact on algorithmic accuracy. In the following sections, we briefly discuss the importance of a system-level view for meaningful optimization evaluations and introduce some basic performance indicators and notations that are used throughout the chapter.

2.3.1 System-Level View

Edge ML devices process sensory data onboard, communicating with sensors (and memories) for subsequent analysis in an ML engine. Regardless of this fact, publications on low-power ML accelerator designs often neglect the impact of such off-chip communication on system-level power consumption and performance. Analyzing system-level

power-breakdown helps identifying power-dominating sub-systems, allowing to optimize them based on the Pareto principle. Fig. 2.2 shows the power breakdown of four mobile system implementations: two smartphone analyses were taken from a smartphone battery usage review [67], showing dominating communication power, while two IoT nodes were evaluated in a visual presence detection system [68] and an always-on face recognition system [69], reporting dominating sensor and processing power. While core optimizations would only enable marginal system improvements in the first three systems, the fourth example shows a typical edge ML IoT node with processing-dominated power distribution, enabling significant system improvements through core optimizations.

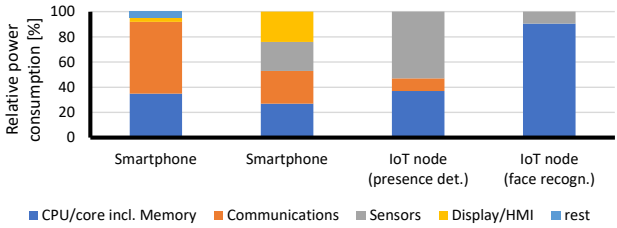

Figure 2.2: Power break-down of mobile/edge systems (smartphones [67], presence detection IoT node [68], and face recognition IoT node [69]).

2.3.2 Meaningful Performance Indicators and Metrics

To identify useful optimization strategies, relevant performance indicators and benchmarks must be chosen. Parameter quantization, for example, can easily reduce memory size but also heavily impacts the accuracy [70, 71]. Similarly, the throughput is subject to large variations across different workloads as pointed out in [49], reporting 20×

lower throughput than stated in the datasheet. Thus, application-relevant benchmarks are indispensable. For a fair cross-device comparison, standard ML benchmarks have been created for smartphones [72], for general-purpose devices (MLPerf) [73], and are now adopted to edge devices (TinyMLPerf) [74]. This is similar to microcontroller benchmarks (e.g. CoreMark [75]). Roofline models [76] are used to visualize the architectural boundaries of a system's operating points, namely the memory bandwidth and the peak computational throughput, as illustrated in Fig. 2.3. Depending on the operating point, the system operates in a memory- or a computation-bound region. NN accelerator performance metrics are often limited to the peak throughput, in tera operations per second (TOPS), and the computational power efficiency (TOPS/W). Fig. 2.4 illustrates that these metrics can be misleading in edge applications, operating in the low power region where extrapolating the efficiency becomes inaccurate (idle power).

2.3.3 Neural Network Notations

This chapter uses the previously introduced notation for NN layer dimensions, specifying input activations *in* ($X_{in} \cdot Y_{in} \cdot C_{in}$), output activations *out* ($X_{out} \cdot Y_{out} \cdot C_{out}$), and C_{out} kernels k containing $K_x \cdot K_y \cdot C_{in}$ parameters. Additionally, we note that other layer types can be described using the same notation: e.g. ResNets have additional bypass inputs in_{by} of the same dimension as *in* and are added pointwise. Depth-wise separable CNNs split kernels in the C_{in} dimension, yielding $C_{out} = C_{in}$, avoiding cross-channel links.

2.4 Hardware Architectures

Two main architectural concepts are used in today's NN accelerators [51]: temporal and spatial architectures. This chapter discusses them and adds emerging in-memory computing as a third architecture, offering a distinct data flow.

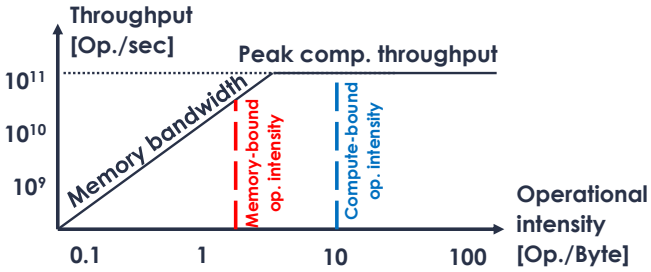

Figure 2.3: Roofline plot showing two operating points, constrained by the available memory bandwidth and the computational throughput.

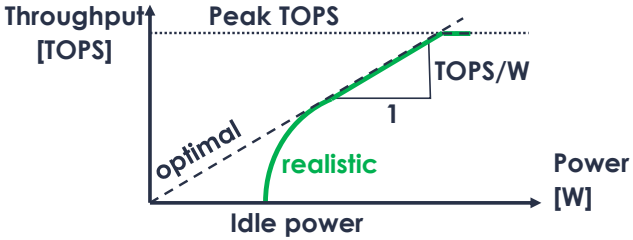

Figure 2.4: NN processing throughput versus power consumption.

2.4.1 Temporal Architecture

Temporal architectures comprise, among others, the widespread central processing units (CPUs) and GPUs, featuring a central control unit that schedules tasks and distributes computations across arithmetic logic units (ALUs). Data is moved from the memory to the ALU and back, offering a high parallelization potential using single instruction, multi-data (SIMD) instructions for vector processing. This is based on the traditional Von Neuman architecture [77], characterized

through a generic single instruction, single data (SISD) processing scheme that stores both instructions and data in the memory. It typically features a control unit that reads instructions and data from the memory, an ALU to perform the operations, and a data bus to communicate across blocks and through a peripheral interface. Multi-issue processors [78], e.g. super-scalar processors, extend this concept by enabling multiple parallel operations through a single instruction, reducing the control overhead, to increase the operational intensity (see Fig. 2.3).

2.4.2 Spatial Architecture

In contrast to temporal architectures, spatial architectures allow their arithmetic units, often called processing elements (PEs), to move data between neighboring PEs, allowing to reduce memory accesses by employing local buffers. Systolic arrays are pipelined 2D spatial architectures that enable data reuse across neighboring PEs. This can be exploited for implementing efficient general matrix multiplication (GEMM) through contraction ("systole" in old Greek) of computations, reusing results from adjacent nodes, and thus minimizing memory accesses. This is exploited in many prominent accelerators like Google TPU [79] and Eyeriss [80]. Replacing single-PE systolic arrays with tensor-PEs, each one computing an entire matrix multiplication per cycle, further allows reducing area and power in a 16nm process by 2.1× and 1.4×, respectively [81]. Increased intra-PE data reuse and fewer pipeline buffer registers enable this improvement but require efficient load distribution to avoid low PE utilization. SCALE-Sim [82] provides a simulation tool to evaluate such design parameters, comparing different dataflows and arrays.

2.4.3 In-Memory Computing

The motivation behind in-memory computing, or compute-in-memory (CIM), is the data-intensive nature of NN inference, requiring high memory bandwidths which result in memory-dominated performance [83], the so-called memory wall [84]. To mitigate this, computations can be directly performed in the memory, where high data access rates are available at a much lower power cost. While the computational

efficiency can be largely increased with this approach, it increases the overhead in the memory and limits the flexibility. Combined with analog memory types, the efficiency can be further improved by computing directly in the analog domain. Analog CIM computes matrix multiplications by breaking them down into vector dot products [85], multiplying analog input voltages (activations) with non-volatile memory (NVM) cell conductances (weights) and accumulating the resulting column current (MAC result). Special design considerations to achieve high accuracy inference along with high computational efficiency are discussed in [83]. SRAM-based CIM is presented in [86], using a 55nm 8T 3.8kb SRAM macro with support for 1-4b input activation and 1-5b weights. The 130nm circuit in [87] demonstrates simple down-sampled MNIST computations, reporting 13× system energy reduction for 1b weight and 5b activation precision compared to a traditional (out-of-memory) computation. CIM integrated into the analog SRAM periphery is evaluated in [88] on a simulated 65nm process, reporting 4.9× system energy efficiency and 2.4× throughput improvement compared to digital processing. A 384kb SRAM-based 8b precision CIM in 28nm [89] is demonstrated to have 28% area overhead compared to a pure SRAM array while achieving up to 22.75TOPS/W throughput. NVM-based CIM implementations are summarized in [85]. Their previous survey [90] provides a comprehensive list of the utilized emerging NVMs, illustrating that NVMs can increase bit density and reduce leakage compared to SRAM and possibly store multiple bits per cell. In [91] an MRAM-based 54×108 CIM crossbar is presented in a 180nm process (MRAM on top of complementary metal-oxide-semiconductor (CMOS) circuit). Over-lifetime variations of up to 4.2% and device-to-device variations of 4.5% have been reported, requiring special considerations (e.g. [92]). Among many other RRAM works, a 1Mb multibit CIM on a 55nm process [93] and a 128×64b CIM array [94] on a 90nm process are presented. NVM-based CIM is an active field of research with various directions, for example 3D CIM architectures [95], reporting up to 28.6× higher energy efficiency compared to 2D chips. A CIM benchmarking tool [96] further reports advantages of NVM- over SRAM-based CIM implementations in a 32nm process, while 7nm SRAM CIM still outperforms any NVM-based work in throughput and both area and energy efficiency. Note that also DRAM has been used

for CIM [97], however, targeting high-performance server applications which goes beyond the scope of this chapter.

2.5 Technology

Integrated circuits (ICs) for NN accelerators are strongly influenced by the underlying semiconductor technology. This chapter introduces common process technologies and related power optimization techniques, followed by an overview of memories, which are often linked to process technologies.

2.5.1 Semiconductor Process Technology

Annual improvements in semiconductor manufacturing technology have been a major driving force in the chip industry as indicated by (the now saturating) Moore's law [98, 99], empirically predicting the doubling of component density on chips every 1-2 years. However, the smaller process nodes increased the static power consumption, resulting in the end of Dennard scaling [100]. This related heuristic scaling trend factor describes the annual shrinking of the minimum feature size in silicon chips and related power savings, culminating in the so-called power wall [101], limiting process improvements because the increased power density has approached the physical limits of silicon-based circuits over the past few years. This process scaling enables significant power reductions [102], as it yields lower supply voltages and smaller switching capacitances. However, improved process technologies are often linked to a significant cost increase and might not support all features of older technologies (e.g. special memory types, photodiodes, etc.). Multi-die solutions can mitigate this problem, exploiting the properties of multiple processes across the dies, each one optimized for a specific target like reduced cost, specialized memory support, or high logic density [24,103]. Combining multi-process solutions with 3D die stacking additionally provides short communication paths and increased densities, as shown in the 8-die-stacked NN accelerator with 96MB of memory [104]. The following sub-sections give a brief introduction of the main semiconductor process technologies used today as they will be referred to throughout the

chapter. We limit the scope to CMOS technology, which dominates digital designs and refer the interested reader to the annual IEEE white paper on future directions of semiconductor technologies (e.g. the 2020 update [105]) for a look into possible future directions.

Bulk

Bulk technology is based on standard silicon wafers and mainly evolves through spatial scaling. However, these annual scaling improvements are slowing down due to increasing difficulties with electrostatic and short-channel effects [106]. Deeply depleted channel (DDC) technology improves bulk technology by introducing multiple vertical doping regions in the channel, forming a threshold setting region and a bias-controllable screening region [107]. This enables reduced supply voltages, low leakage, low process variation as well as improved body biasing characteristics, allowing to dynamically adjust the threshold voltage of the transistor.

FinFET

The introduction of 3D gates in so-called FinFETs improved on the performance of bulk CMOS by enabling lower supply voltages, and thus reduced power consumption, as shown for 22nm tri-gate FinFET [106] and later 14nm FinFET [108]. However, the improved performance comes at a higher production cost due to the complex 3-dimensional structures, requiring more fabrication masks than the bulk technology [109–111].

FD-SOI

Fully depleted silicon-on-insulator (FD-SOI) technology employs very thin insulating layers in the substrate of the transistors, reducing leakage. In [110] a 22nm FD-SOI technology is presented, achieving on par power and performance efficiency compared to 16/14nm FinFET technology while being lower cost due to the use of planar processes, requiring fewer masks. Thus, FD-SOI is more suitable for low-end mobile and IoT applications where cost is an important factor. A detailed comparison between FD-SOI and FinFET is provided in [109],

reporting superior performance of FinFETs in terms of power, delay, and density, for which FD-SOI can compensate through body-biasing. Body biasing allows to dynamically adjust the threshold voltage, boosting speed with a forward body bias or reducing leakage using reverse body biasing (higher threshold).

Specialized Processes

Recent advances in materials and manufacturing technologies have enabled the integration of novel memory technologies (both volatile and non-volatile) close to the processing logic. They offer advantageous characteristics that go beyond transistor density scaling. However, most of them require special process technologies, making them more difficult to integrate within the widely available processes. More details on novel memories can be found in Section 2.5.3.

2.5.2 Power Management

The power consumption of ICs [101] can be decomposed into dynamic and static power. Dynamic power is described in Equation 2.1, taking the circuit switching operations into account. Variable U is the supply voltage, C the switching capacitance, α the switching activity and f the frequency of the circuit. Static power, often also called leakage power, is described in Equation 2.2, reflecting the current consumption I_{leak} when no switching takes place.

$$P_{dynamic} \sim \frac{1}{2} \cdot U^2 \cdot C \cdot \alpha \cdot f \tag{2.1}$$

$$P_{static} \sim U \cdot I_{leak} \tag{2.2}$$

Based on these equations, various optimizations have been proposed to reduce the overall power consumption, using the supply voltage and the frequency as control knobs. The most prominent techniques are listed in the following section, namely sub- and near-threshold operation, adaptive body biasing (ABB), and dynamic voltage and frequency scaling (DVFS). If the application permits, duty cycling (pausing the operations to reduce the switching activity α) and power gating (to reduce static current I_{leak}) can be used to further reduce the power consumption.

Sub- and Near-Threshold Operation

Sub- and near-threshold operation exploits the supply voltage knob, operating transistors below or close to their threshold voltage, respectively, to reduce power consumption [112]. This enables to reduce dynamic power quadratically and static power linearly, as shown in Equations 2.1 and 2.2. However, these savings come at the cost of slower transistor operation, limiting the application frequency. In embedded IoT applications, energy is usually more important than power consumption, as it directly determines the lifetime of the battery. Thus, the minimum energy point (MEP) is identified for each application by adjusting the supply voltage such that the total energy for a specific workload is minimal. Sub- and near-threshold operation extends the supported supply voltage range, reaching the MEP for a large set of workload scenarios. Lowering the voltage implies higher sensitivity to process variations, which must be carefully evaluated during the design phase to ensure robustness under all operating conditions. Special layout considerations and compensation techniques (see ABB and DVFS) can reduce the effects. The 180nm sub-threshold standard cell library developed in [112] demonstrates an extended 0.4-1.0V supply voltage, reducing power by 5× compared to a standard low power library at 1.0V. Their follow-up work presents 1kb sub-threshold SRAM [113] and a 32b microcontroller [114] design, achieving 0.84-3.2nW (3.8x) power scalability for 0.27-0.6V voltage scaling and 7× power scalability for 0.37-1.8V voltage scaling, respectively.

ABB

ABB [115] adjusts the bias voltage of the transistor body to control its threshold voltage, influencing its speed and power consumption (as discussed above). FD-SOI and DDC can fully exploit body biasing, while the effect in bulk technology largely depends on the design parameters. Adapting the body bias to the operating point enables to reduce the adverse effect of sub-threshold operation on timing across process and temperature variations (e.g. worst-case corner distance from 21× to 0.2× [116]). This allows to implement a wide range of timing-clean operating points from fast to slow and low power. Note

that increasing the speed through a strong forward body bias also increases the leakage. Publications on ABB report 30× frequency and 20× leakage scaling on a RISC microcontroller core and SRAM [115].

DVFS

Dynamic voltage, frequency, (and accuracy) scaling (DVF(A)S) allows to trade-off speed against power consumption through supply voltage scaling. The basic principle was already evaluated in the 1990s, allowing CPUs to lower the frequency and voltage for low-intensity tasks, reporting power reductions of 1.05-4× [117], and 9.2× [118]. DVAS [119] extends the principle to dynamic accuracy scaling through adjustable arithmetic bit-widths, allowing for lower supply voltages due to shortened critical paths. Demonstrated on a simulated 40nm 16bit array multiplier, DVAS achieves 11.7× energy reduction for scaling down to 8b precision, noting that the critical path length is reduced by 40%. Pipelined architectures require bypassable pipeline registers to evenly distribute the path length reductions, adding area and energy overhead. On a 16b multiplier, 11.1× energy reduction is reported for 8b operation at 8% energy overhead. Envision [62] combines accuracy scaling with DVFS on a 28nm process, running at constant throughput while scaling precision from $1 \cdot 16b$ operations to $4 \cdot 4b$ operations at a quarter of the 16b frequency. Combined with body biasing to reduce leakage at low frequencies, power is reduced by 25x.

2.5.3 Memory

Data handling is dominating today's accelerator power consumption and area, requiring careful selection of the memory type and the access strategies. Memory access energy is subject to large variations across memory types and sizes, as summarized for a standard 45nm SRAM [102, 120] in Table 2.2. DRAM is reported to have 200× higher access energy than a small 8kB SRAM (for 64b word width) [102]. Fig. 2.5 shows an overview of the existing memory types, distinguishing storage (e.g. non-volatile hard disk) and memory (e.g. RAM) due to their significant difference in access times, density, and power consumption [121]. Systems with long idle/sleep phases might

not be dominated by access power but (idle) leakage power. In that case, rare accesses to (power-intensive) non-volatile memories could be cheaper than retaining data over a long period of time, constantly consuming leakage power.

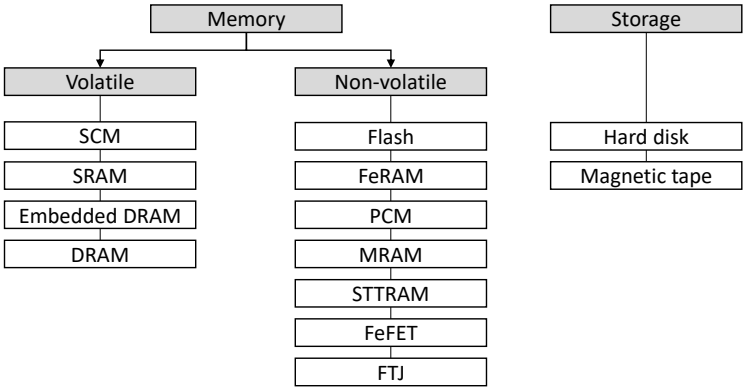

Figure 2.5: Overview of memory technologies.

The range of available memory technologies is rapidly increasing, being an active area of research. A recent overview of novel memories used in neuromorphic computing [122] notes that the memory technologies mostly differ in their writing speed, while the reading process is dominated by the sensing circuit interfacing the memory cells. Survey [121] provides an overview of recent non-volatile memories, focusing on PCM, STTRAM, RRAM, and FeFET. We summarize the working principles and extend it with the dominantly used SRAM and standard cell memories as well as quantitative optimization results from the literature.

PCM

Phase change memory (PCM) [121] is based on the heat-induced reversible phase transition of chalcogenides. It can switch between the low-resistance crystalline phase and the high-resistance amorphous phase, implementing a bipolar switch. The relatively large switching

Table 2.2: Energy per memory access in 45nm SRAM

Size(kB)	**Access energy per 16b word [pJ]**			
	64b	**128b**	**256b**	**512b**
1	1.2	0.93	0.69	0.57
2	1.54	1.37	0.91	0.68
4	2.11	1.68	1.34	0.9
8	3.19	2.71	2.21	1.33
16	4.36	3.57	2.66	2.19
32	5.82	4.8	3.52	2.64
64	8.1	7.51	5.79	4.67
128	11.66	11.5	8.46	6.15
256	15.6	15.51	13.09	8.99
512	23.37	23.24	17.93	15.76
1024	36.32	32.81	28.88	25.22

current requires powerful access circuits, dominating the size. Speed is limited by the transition from amorphous to crystalline phase while the reverse process dominates the power consumption. Endurance and speed are estimated around 1G cycles and <100ns, respectively [121]. TSMC presents a 1Mb PCM memory array in 40nm [123], reporting 300uA write current at 100ns write speed, achieving >200k cycles endurance.

STTRAM

Spin-transfer-torque RAM (STTRAM) [121] is based on a magnetic tunnel junction (MTJ) with two ferromagnetic layers separated by an ultra-thin tunnel oxide layer. It uses the spin-transfer torque of spin-polarized electrons to change the resistance of the memory element, enabling non-volatile states. The access circuit dominates its size due to the relatively high writing currents but is still smaller than SRAM. STTRAM is demonstrated on 7-8Mb arrays (industrialized 1Gb cluster [124]) in 22-28nm [125, 126], reporting densities up to 10.6Mb/mm^2 and write endurances of >1M-10G cycles.

RRAM

Resistive RAM (RRAM) [121] stores information by modulating the resistance of a metal oxide and is therefore often called a memristor. A similar approach is conductive-bridge RAM (CBRAM), that forms a conductive metallic bridge in the on-state and interrupts it for the off-state. Endurances of 1M-1G cycles have been reported. However, there are tradeoffs between speed, power, and endurance. Various RRAM implementations in 14-40nm have been shown [93, 127–130], reporting 0.9-244.8Mb/mm^2 density at supply voltages down to 0.7V for 1Mb-32Gb sizes.

FeFET

Ferroelectric FET (FeFET) [121] employs a ferroelectric gate dielectricum that allows changing the resistance of a FET in a non-volatile fashion. This principle is similar to the Flash technology, which uses a floating gate instead. While the power consumption is low due to the low leakage through the gate oxide, the switching speed is high ($\sim$20ns). However, its endurance is relatively low, at 10k-100k cycles. Its similarity with standard CMOS transistors makes FeFET compatible with many standard processes. In [131], a 10Mb FeFET memory array is implemented in a 22nm process, reporting 200k cycles endurance at 2.5-4.5V supply voltage. Their previous work [132] presents a 64×64b array in the same 22nm process and compares it to a 6T SRAM array of the same size, reporting 74x lower static power and >5.3× lower area for FeFET cells (without peripherals). Writing is 10× more energy-intensive and 10× slower while reading costs 1.6× more energy but is 1.5× faster. A similar, but less mature, type of ferroelectric memory is ferroelectric tunnel junction (FTJ). The tunneling resistance of its ferroelectric layer between two metal electrodes can be adjusted through ferroelectric polarization reversal [121].

SRAM

Static random-access memory (SRAM) is the most often used on-chip memory as it can be easily implemented along with digital circuits. It features higher memory density than standard-cell memory

as foundries use "layout pushed rules" to optimize SRAM bit-cell area beyond standard layout rules. Standard SRAM bit-cells use 6 transistors (6T), but many larger cell structures have been proposed for power reductions. For low leakage operation, a 7-transistor SRAM is presented in [24], reducing area by 18% and 50% compared to standard low leakage 8- and 10-transistor designs, respectively, while achieving similar performance and leakage power. Power optimizations using non-uniform memory hierarchy in SRAM are presented in a 40nm NN accelerator [133], allowing up to 60% power savings when accessing the smallest instead of the largest level (32× smaller memory). The 67.5kB memory is split into 4 levels (1.5, 6, 12, and 48kB). SRAM access energy is shown to increase nearly linearly with the memory size above 100kB [120] as shown in Table 2.2. Low leakage SRAM sleep-mode retention is presented on a 55nm process [134], reporting 26× lower retention power for a 16kB memory. Leakage is reduced by optimizing the design rules (increasing area by 2.7x) and the process corner. An additional sleep controller with a charge pump for the retention voltage allows to power-gate the rest of the chip. Leakage is also reduced in [103], using 180nm low leakage 10T SRAM along with a 65nm 8T SRAM for dense scratchpad memory. The low-leakage memory consumes 4242× less standby power while being 11.3× larger in area. To optimize the data access for 2D structures like in CNNs, a transpose SRAM has been proposed [135], allowing to selectively read a row vector or a column vector of data in parallel, reducing power consumption by 47%.

SCM

Standard-cell memory (SCM) is implemented directly in digital logic using flip-flops or latches. This allows exploiting voltage scaling capabilities and avoids dependencies on vendor-specific memory generators. A dedicated placement strategy for SCM (instead of standard logic place-and-route (P&R)) is presented in [136], reporting area and power savings on a 28nm process. Their experiments on 256b-32kb SCM macros show area reductions of >35% compared to standard P&R with an access energy reduction of up to 65% for reading and 50% for writing. SCM macro sizes of up to 1kb are shown to be smaller than SRAM, but already 2-3× larger for 4kb macros (larger

area of D-latch cell compared to 6T SRAM bit-cell starts dominating). A BNN accelerator [137] with a hybrid memory consisting of 456kB SRAM and 8kB SCM demonstrates power savings of SCM compared to SRAM. It reduces the supply voltage to 0.4V when SCM is used, while SRAM requires 0.6V, leading to 3× energy savings in 22nm post-layout measurements.

DRAM

Dynamic RAM (DRAM) is a volatile high-density memory that stores information as a capacitor charge, which is periodically refreshed. DRAM is usually not compatible with standard logic processes, requiring separate DRAM dies. However, the hybrid memory cube (HMC) architecture [138] proposes high-density DRAM access to standard logic processes through 3D stacking of DRAM dies on top of a logic die using through-silicon vias. HMC is implemented in TETRIS [139], providing DRAM access to a 45nm processing die. 3D DRAM is shown to consume 3.5-4.2× more energy compared to on-chip 256kB SRAM, but 1.5× less than a planar baseline DRAM at 4.1× higher throughput. Embedded DRAM (eDRAM) [140] is a CMOS-compatible derivative of DRAM, targeting high-density volatile memory. A 4-transistor 8kb eDRAM array is presented in 28nm [140], reporting 17% lower area and 23% lower static power consumption compared to 6T SRAM at equal voltage. The same author also evaluates a 2T design [141], showing slightly lower retention power than low power SRAM cells but 2.5-3.8× lower size for a simulated 28nm 4kb array.

Flash

Flash memory [142] dominates NVM technology on the market, being embedded in most commercial chips where data retention during off-state is required. It can be implemented using standard logic process flows by adding a few additional masking layers, e.g. 3 extra masks on a 65nm process [143].

2.6 Dataflow and Control Optimizations

NNs have a relatively simple structure, but their efficient implementation on hardware accelerators often complicates the dataflow. Efficient parallelization requires smart workload distribution among processing elements while ensuring coherent algorithmic functionality. Thus, this chapter discusses algorithm blocking optimizations, efficient scheduling, selective execution, and early data reduction.

2.6.1 Dataflow and Blocking

NNs accelerators have optimized memory allocation and access patterns for efficient computation, splitting a task into smaller blocks that fit on the available resources. The utilized blocking (scheduling) strategy impacts the data access order and thus possible data reuse within the computation blocks. Higher reuse can reduce the number of memory accesses, decreasing the total power consumption [120].

Layer-Wise Processing

Traditionally, NNs are processed layer-by-layer, also called layer-wise or layer-first approach. This enables the reuse of layer parameters (e.g. convolution weights), as they are repeatedly used across the layer. However, this also implies that at least one complete layer is always buffered in memory.

Depth-First Processing

The increasing size of feature maps (e.g. higher resolution images) requires large activation buffers to be allocated for layer-wise processing. However, networks can be processed in a "depth-first" streaming fashion instead [144], allowing each layer to buffer only a minimum set of input activations that are needed for computing the next set of output activations. Up to 200× memory bandwidth reduction or alternatively up to 10'000× memory space reduction is reported for this approach. In a follow-up work [145] depth-first processing is implemented on an FPGA and benchmarked on five models reporting throughput increase of 0.91-1.27× and memory bandwidth reduction

of 3.9-81x. A similar work on "fused layers" [146] observes that each output of a convolution layer only depends on a small region of input values. Tracking these dependencies back through multiple layers results in a pyramid-shaped region. The paper proposes to compute the entire pyramid until the final output while only storing the computed intermediate features instead of buffering each complete. The additional cost for buffering intermediate results locally is traded-off against external memory accesses, achieving up to 95% reduction in off-chip memory traffic for running VGGNet-E.

Loop Ordering and Optimization

NNs can generally be described using nested loops, with the outermost one looping through the layers of the network. The ordering of these loops influences the possible parallelisms and the required memory size. To process larger networks on limited on-chip memory resources, loop tiling is used: the workload of each layer can be split into overlapping tiles, which are processed sequentially. Parallelization and data reuse can be increased by unrolling parts of the sequential loops and thus parallelizing their computations. Unrolling the entire kernel computation is shown in [147], achieving unprecedented power efficiency at the cost of a larger area and limited layer size support. Various tools have been proposed to optimize loop ordering [120, 148–150], improving energy efficiency and memory size.

2.6.2 Compiler

Compilers and NN mapping tools translate the algorithmic representation of a trained network into machine code that is executed on the processing hardware as shown in Fig. 2.1. They can optimize dataflow strategies as listed in Section 2.6.1. Commonly implemented microcontrollers provide specialized libraries to make NN processing more efficient. For example, ARM provides the CMSIS-NN library for its Cortex-M microcontrollers [151], supporting CNN, FC, and pooling layers, as well as 8b or 16b fixed point precision. Evaluated on a network running the CIFAR-10 task, a 4.6× improvement in throughput and 4.9× in energy efficiency is reported, compared to a digital signal processor (DSP)-functions-limited baseline code. A

biomedical signal analysis application [152] reports energy savings of 41.6% for enabling 8 core processing instead of single-core, noting that the overhead of multi-core execution is fully compensated by efficiency improvements above a certain throughput. Block and memory power gating shows 16.8% energy savings but must be traded off against restart time and related energy and storage implication.

2.6.3 Early Data Reduction

The high cost of data access during NN inference motivated to reduce the data flow from the sensor, condensing information early and thus minimizing costly data transfers. One approach is to process the first layer(s) of a NN in the sensor itself. In [153] diffraction gratings above the pixels are used to optically detect Gabor filter-like patterns, as they are often found in the first layer of NNs. Evaluated on the MNIST and CIFAR-10 tasks, sensor communication bandwidth reductions of 10× are reported, with moderate accuracy impacts of -0.1% and -4.6%, respectively. However, the first layer of the employed LeNet-5 only accounts for 3.8% of all operations, rendering the computation reduction negligible. Another study implements the first CNN layer in the optical domain using a controllable (grayscale) mask [154]. All filters (output channels) are displayed in the same plane, allowing the image sensor behind to capture all convolution results in parallel, forwarding them to the last layers implemented in the digital domain. Evaluated on MNIST and EMNIST tasks, operation reductions of 250× and 460× are reported while accuracy drops by 0.4% and 1.7%, respectively. The analog nature of most sensor signals requires analog-to-digital converters (ADCs) which can be exploited by implementing matrix multiplication directly in the ADC [155]. An algorithmic reformulation is shown to implement a simple classification task using boosted linear classifiers, embedded in a single matrix transformation. The matrix multiplication is implemented in the feedback path of the SAR ADC, reporting 13× and 29× energy savings compared to a support-vector machine (SVM)-based implementation with similar accuracy for ECG arrhythmia detection and 160x120 pixel gender classification tasks. RedEye [156] implements an image sensor with analog on-die CNN processing capabilities on a simulated 180nm process. It uses SAR ADCs and

tunable capacitors to implement weighted summation to mimic MAC operations, reporting 73% system energy reductions for running the first 1-5layers of an 8bit GoogleNet on the ImageNet task. Early data reduction is also implemented in distributed computing [157, 158], splitting the DNN computation across the edge and the cloud to reduce costly data communications, latency and preserve privacy by keeping raw data at the edge. Furthermore, application-specific early data reduction mechanisms exist: visual attention [39] is shown to reduce the object recognition workload in smart glasses by limiting the analyzed region to the detected eye-gaze direction.

2.6.4 Selective Execution and Early Abortion

This section presents techniques to dynamically adapt the network complexity to the input, enabling "simple" inputs to be analyzed with a fraction of the network capacity without decreasing the accuracy for complex ones. Average latency and power consumption can thus be reduced if simple inputs dominate the execution. Fig. 2.6 illustrates the techniques, covering a) hierarchically scalable effort [62, 159], b) early exiting [160, 161], and c) selective execution [162].

Hierarchically Scalable Effort

Identifying early exits during training enables 2-6× latency reduction on MNIST and CIFAR-10 tasks [159]. A scalable-effort approach [163] proposes to use a chain of networks with increasing complexity, allowing simple inputs to complete processing with smaller networks than more complex ones. Evaluated on various classification tasks, they achieve 1.2-9.8× average reduction of operations per benchmark. The Envision NN accelerator [62] demonstrates this on a face recognition task, hierarchically increasing the network complexity, starting with a 12MOP network for presence detection (6.4mW, active 98% of the time), followed by a network recognizing the owner, a set of 10 identities, 100 identities, and finally 5760 different identities (77mW, 0.01% of time).

Early Exiting

Adding special output classifiers after every few layers allows terminating a NN execution early if classifiers report high confidence [160]. The trained network is analyzed after each layer to estimate a gain metric, quantifying the ratio between the reduced number of operations and increased overhead due to the added classifier. Benchmarked on two 6- and 8-layer networks with 1 and 2 early exits, respectively, 1.73× and 1.91× reduction in number of operations are reported at iso accuracy. The same technique is used in the 12-class keyword spotting accelerator [161], reporting 69% of the inputs exiting early, reducing the average power consumption by 22% compared to always executing the complete network.

Selective Execution

Selective execution [162] enables different execution paths that can be selected depending on the input provided: an embedded selector network decides which branch to execute, providing less complex network branches for simpler inputs to reduce the average number of computations.

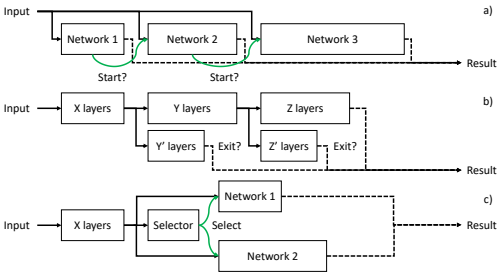

Figure 2.6: Selective execution and early exiting approaches: a) scalable-effort execution, b) early exiting, and c) selective execution.

2.7 Data Handling Optimizations

The data-intensive nature of NNs challenges the memories and related access energy efficiencies. In many systems, more than 50% of the total power consumption is related to memories and data handling [102]. This chapter discusses optimizations for efficient memory utilization and compression, reducing the amount of data to be accessed. Related sections cover the efficient reuse of data across computation elements (Section 2.8.3) and the reduction of data through computational optimizations (Section 2.8).

2.7.1 Efficient Memory Utilization

Efficient memory utilization reduces the required memory space, saving power, chip area, and thus IC cost. The traditionally used buffering scheme for layer-wise NN processing is often called ping-pong buffering [149], following a double buffing approach to allow simultaneous reading of input activations and writing to output activations. It maps the activations of subsequent layers to two disjunctive memory regions, which must therefore allocate at least the maximum sum of any two subsequent layers. By allowing the activation memory regions to overlap during layer-wise processing, memory savings of up to 50% compared to standard ping-pong double buffering can be achieved [28]. The extent of savings depends on the layer dimensions and increases for large layers with small kernel sizes.

2.7.2 Data Compression

Compression reduces the memory footprint of data content and can be adopted in NN accelerator designs. Run-length compression (RLC) of zero values is used in Eyeriss [63] to reduce the memory footprint and bandwidth. It encodes the number of zero entries in 5b, followed by the next non-zero value, reporting 1.2-1.9× reduced memory accesses for AlexNet. Eyeriss v2 [66] uses a "compressed sparse column format" for both weights and activations, allowing to skip sparse operations directly in the compressed form, reducing memory bandwidth and energy. Compared to RLC, it simplifies addressing of sliding window striding. Loss-less Huffman coding [164], is shown to reduce weight

memory by 20-30%. It employs variable-length codewords, providing smaller bit-widths for more common values, reducing the overall memory. An edge ML Huffman-coding direct memory access (DMA) is shown to reduce data bandwidth by up to 5.8× [62]. Weight sharing is used in [164], replacing weights with table indices, referencing a limited number of physically stored values. The upper bound of memory savings is defined by the weight bit-width $N_{width,w}$ and the number of table entries N_{values} as shown in Equation 2.3. The 45nm accelerator EIE [61] reports 8× energy savings through weight sharing (4b indices referring to 16 16b weight values).

$$N_{weightshare,max} = N_{width,w}/log_2 N_{values} \tag{2.3}$$

2.8 Computation Optimizations

NNs contain millions of MAC operations, requiring fast and efficient accelerator designs. This chapter discusses computation optimizations ranging from operation reductions (optimized convolution operations, sparsity, or data reuse) to arithmetic simplifications (quantization, approximate computing, energy-quality scaling, or non-conventional arithmetic) and circuit optimizations (mixed-signal arithmetic, non-conventional arithmetic).

2.8.1 Optimized Convolution Implementations

The dominance of convolution operations in many network architectures [165, 166] motivated optimized convolution implementations, aiming for similar algorithmic behavior while reducing computational complexity and resources.

Separable Convolutions

Separable convolutions are based on a separable filter approximation, splitting higher dimensional kernels (e.g. a $k_x \cdot k_y$ 2D convolution) into multiple lower-dimensional ones (e.g. 2 1D convolutions of $k_x \cdot 1$ and $1 \cdot k_y$), significantly reducing the number of operations. A 2D approach is used in [135], replacing a $5 \cdot 5$ convolution layer with a horizontal and a vertical $5 \cdot 1$ and $1 \cdot 5$ layer, reducing the number

of operations by 4x. Evaluated on an LFW face recognition NN, the total number of operations is reduced by 1.7x, while accuracy was decreased by 1%. To optimize parallel data access for the vertical direction, a transpose SRAM (T-SRAM) is proposed, enabling both row and column vector readout, reducing power by 47%. Depth-wise separable convolutions (DSC) separate the kernel only in the depth dimension, convolving inputs in the spatial directions, followed by a pointwise (across depth) convolution that combines the filtered inputs to an output. In contrast to standard convolutions that combine the filtering and output generation, DSCs enable memory savings and reduce MAC operations. MobileNets [16] use such DSCs, reporting computation and parameter size reductions according to Equation 2.4. Evaluated for the ImageNet task, parameters can be reduced by 7× while the number of MAC operations is reduced by 8.5× at an accuracy drop of just 1%.

$$reduction = \frac{1}{c_{out}} + \frac{1}{k_x \cdot k_y} \tag{2.4}$$

Frequency Domain Computation of CNN (FDC)

Transforming a convolution operation into the Fourier domain results in a point-wise multiplication as shown in Equation 2.5 [167]. This property can be exploited to reduce the number of operations for computing CNNs. While the forward and backward transformations, using fast Fourier transformation (FFT), increase the computational effort and memory needs, the operation count can be significantly reduced for large input and kernel sizes, achieving 1.75-5.3× faster computation at iso-accuracy for 3x3 and 11x11 kernels in experiments on various layer sizes [167]. However, this method requires large memories for the FFT and the subsequent matrix multiplication. A recent study [168] builds on the FFT-based approach, additionally exploiting tiling, result-reuse, and symmetry of real-valued FFTs, roughly cutting the number of operations and Fourier outputs in half. It demonstrates 0.96-1.74× increase in throughput compared to the pure FFT-base approach on 9x9-3x3 kernel sizes.

$$f * g = F^{-1}\{F\{f\} \cdot F\{g\}\} \tag{2.5}$$

Winograd Algorithm

For highly parallelized convolution computations, the sliding window operation can be flattened to convert it into a large point-wise matrix multiplication. The overlapping windows create significant redundancy with neighboring data, allowing to combine certain kernel-weights offline, which reduces the number of multiplications at the cost of more additions [166]. This so-called Winograd convolution achieves 1.48-2.26 speedup for computing 3x3 convolutions.

Strassen Algorithm

Large matrix multiplications, as used in NNs with large kernels, can be efficiently computed using the recursive Strassen algorithm, reducing the number of operations [169]. Evaluated on AlexNet, operations are reduced by 47%: while the dominating 3x3 and 5x5 convolutions are reduced in terms of operations, the 11x11 convolution in the first layer suffers from an increase of 18% compared to standard multiplication.

2.8.2 Sparsity Exploitation and Pruning

Le Cun et al. [170] observed more than 3 decades ago that a significant number (75%) of NN parameters can be removed without affecting its algorithmic accuracy. A recent exhaustive survey [171] provides explanations to this phenomenon and estimates 10-100× model size reduction for various networks. It focuses on sparsification methods that set parts of a network to zero (pruning), while keeping its complexity constant. They reduce a network's size to its minimum required complexity by creating a (too) high dimensional representation to improve the training, knowing that the network can be reduced again through pruning. Previous works, report weight sparsity ratios of up to 99.996% for a LeNet-5 network with 99.3% accuracy on the MNIST task [172]. We refer the interested reader to the literature for more details on the main reduction techniques: model down-sizing through neural architecture search [173], operator factorization [174], quantization (Section 2.8.4)), compression (Section 2.7.2), parameter sharing [175], and sparsification [171]. A survey on hardware acceleration of compressed models can be found

in [176]. To create more sparse models, a 3-step approach is proposed [177], first learning the importance of each connection, then dropping low-weight ones, and finally fine-tuning the training. Evaluated on AlexNet and VGG16 running the ImageNet task, they report 9× and 13× parameter reduction at iso-accuracy. Energy-aware pruning [178] optimizes the pruning strategy to achieve a minimum energy cost. Observing that layers that are pruned in an early stage, tend to have larger sparsity, they start pruning energy-intensive layers first, estimating their cost based on the number of computations and memory accesses. Evaluated on AlexNet running the ImageNet task, reports 3.7× energy reduction and 11× weight reduction at <1% accuracy drop. Envision [62] exploits sparsity by skipping sparse memory accesses and MAC operations using a sparsity map (1b entries per value). It reports 1.6× system energy saving for 30-60% activation sparsity. Similarly, NullHop [179] exploits sparsity by skipping sparse computations using a sparsity map combined with non-zero value list compression, achieving up to 3.68× throughput increase in 28nm synthesis results. Zero-value skipping logic using a zero-free neuron array format is evaluated on a 65nm accelerator [180], reporting 1.37× average throughput increase for various networks at the cost of 25% memory increase (for zero sparsity). A special form of temporal sparsity is proposed in CBinfer [181].

2.8.3 Data Reuse

Memory data that is used multiple times within a short period of time can be buffered in a local cache memory to allow cheaper and faster access. Special focus is set on data that has been fetched from energy-expensive external memory. Three main structures of such data reuse have been studied extensively [51] and are summarized below, namely row-stationary (RS), weight stationary (WS), and output stationary (OS) approaches. While a WS setup minimizes movements of weights, as it is used in CIM architectures, OS keeps the partial sums for computing each output feature local, and RS approaches combine weight and activation data reuse. Eyeriss [63, 80] implements an RS approach to maximize on-chip data reuse and reduce costly external memory accesses. The 65nm chip can buffer one activations row (up to 224 16b values) and one weight row (up to 12 16b values),

increasing energy efficiency by 1.4-2.5x. A weight stationary approach is presented in [182], reducing the weight memory accesses by moving activations along cached weights during convolution computations. OS processing is used in ShiDianNao [60], computing one output per processing element in its 8x8 array (64 outputs of the same output channel). This avoids moving partial sums to the memory and allows sharing inputs with neighboring PEs, reducing the memory bandwidth by up to 10x. Compared to their prior work DianNao [183], featuring no local data reuse, it allows to reduce power consumption by more than 1.66x, while increasing throughput by 1.87x. Another output stationary approach is used in Hyperdrive [184], keeping the feature maps stored entirely on-chip and streaming in weights. This is motivated by the use of binary weights, consuming 16× less memory bandwidth than the 16b float activations. Eyeriss v2 [66] implements a flexible network-on-chip (NoC) that can be reconfigured into 4 main reuse modes, enabling high activation and weight reuse in convolution layers while the dense layer mode can maximize activations broadcast because weights cannot be reused. Evaluated on MobileNet, they report a throughput increase of 5.6x. Temporal data reuse is proposed in CBinfer [181], demonstrating that CNN-based CV applications with a static field of view can reuse large portions of each CNN layer and only compute those features that changed over time. They first detect changes in the input, generating a temporal sparsity map to update the connected output features for which the inputs exceed a calibrated threshold and buffer the new feature map. Evaluated on a 5-layer CNN for 10-class scene segmentation, this achieves an average speed-up of 8.6x.

2.8.4 Hardware/Software Co-Optimization

A recent publication proposes less artificial intelligence [185], suggesting that today's networks are too high dimensional and thus prone to overfitting limited datasets, providing some intuition why high sparsity and quantization are viable optimization strategies. Today's NN algorithms exploit this observation, being developed with the challenges of resource-constrained edge ML hardware in mind. Thus, optimized algorithms like MobileNets [16] or SqueezeNets [186], reducing complexity through separable convolutions and kernel size

minimization, have been introduced. These co-optimization strategies are covered in this section.

Complexity Scaling

The growing number of network architectures created a large design space, shifting the design strategies from hand-crafted architectures to (semi) automatic neural architecture search (NAS), identifying optimal trade-offs between design constraints like accuracy and computational complexity. Frameworks like adaDeep [187] provide multi-dimensional model selection strategies to find the optimal model scaling strategies for a specific model and use-case. Dynamic complexity scaling is proposed by once-for-all networks [188], which are trained once but can then be deployed in various down-scaled complexities (in depth, width, kernel size, and resolution) without re-training. This allows deploying them on platforms ranging from high-performance cloud servers to low-power edge devices, with low accuracy degradations for the reduced-size versions.

Energy-Quality-Scaling

Energy-quality (EQ) scalable systems [189] introduce a quality metric that describes a network's complexity. This allows identifying the knobs for power-scaling through (acceptable) quality degradations. The presented framework for EQ-scalable systems and EQ architectures [189] helps identifying applications where sensing and/or processing quality degradation is acceptable, for example in noise-resilient applications like computer vision. The list of quality knobs for dynamically adjusting EQ scaling encompasses arithmetic precision, bit error rates, sampling rates, algorithmic complexity, and more. Follow-up works exploit this concept for implementing ULP voice activity detection [190] or always-on computer vision system [191]. Voice activity detection is shown to support EQ scaling [190], achieving 3.5× lower energy for 2% accuracy degradation using decision trees on a 28nm chip. Joint voltage and EQ scaling applied to a traditional computer vision task [191] is shown to achieve 3× lower energy through EQ scaling and 3.4× lower energy through VDD scaling on a 40nm process.

Quantization and Reduced Precision

NNs can tolerate significant parameter and activation quantization with negligible effects on accuracy [70]. To reduce the effect of reduced precision, quantization-aware training techniques are used (e.g. straight-through estimators [192]) to keep the model derivable for back-propagation. Quantization works especially well in networks that are limited in training data, while the accuracy degradation increases for smaller (complexity-limited) networks, which cannot compensate for the loss of information [193]. Lowering precision reduces memory size, simplifies computational arithmetic, and lowers the power consumption, which also motivated Google to add 8bit support in their TPUs [79]. The viability of fully binary NNs (BNNs) [194], limiting activation and weight precision to 1bit, finally demonstrates the full range of quantization possibilities. BNNs have considerable accuracy degradations but allow to process multiplications using simple XNOR logic, reducing area and power needs. Survey [71] provides an in-depth analysis of quantization schemes with a special section on sub-8bit quantization and a short overview of quantization-optimized hardware. Weight sharing is another form of quantization, limiting the number of supported values, as discussed in Section 2.7.2. Energy and area savings of reduced precision arithmetic have motivated quantization optimizations. The energy for additions and multiplications are evaluated on a 45nm process [102], reporting 14× and 20× MAC energy reduction for 8b int compared to 32b int and 32b float, respectively, as summarized in Table 2.3. A 45nm overview [195] reports power and area increase with bit-width for adders (linear increase) and multipliers (quadratic increase), as shown in Table 2.4. Comparing a MAC-combination, total area and power are reduced by 13× and 10.8x, respectively, for 32b to 8b, and by 3.6× and 3.6x, respectively, for 16b to 8b. Similarly, this was shown on multipliers for a 65nm process [183] as illustrated in Table 2.5.

The 65nm accelerator in [183], achieves 6.1× reduction in area and 7.33× in power consumption for implementing 16b fixed-point multipliers instead of baseline 32b floating-point arithmetic while maintaining comparable accuracy. Envision [62] exploits 1-16b dynamic precision scaling combined with DVFS, reporting reductions in energy per MAC operation (relative to 16b) of >5× for 8b and >50× for 4b

Table 2.3: 45nm MAC energy per operation for different precision

Precision	**Int8**	**Int32**	**Float16**	**Float32**
Addition energy [pJ]	0.03	0.1	0.4	0.9
Multiplication energy [pJ]	0.2	3.1	1.1	3.7
Total energy/MAC [pJ]	0.23	3.2	1.5	4.6

Table 2.4: 45nm adders and multipliers for different precision

Precision		**Int8**	**Int16**	**Int32**
Adder	Area [μm^2]	212	322	1117
	Power [μW]	753	2235	4819
Multiplier	Area [μm^2]	1038	4209	15126
	Power [μW]	2830	10816	34034

Table 2.5: 65nm multipliers for different precision

Precision		**Int16**	**Float32**
Multiplier	Area [μm^2]	1309	7998
	Power [μW]	577	4230

precision using sub-word parallel computations in a 28nm process. UNPU [196] employs non-linear quantization support by replacing 16b multipliers by 2x4-bit lookup tables, indexing 16 16b activations and 16 16b weights, with the result of each combination stored in the table. They report 79% power reduction and 93% lower latency while the area is reduced by 1.3× compared to instantiating a 16b multiplier. A review on scalable precision MAC architectures [197] compares recent 2-8b scalable implementations, discussing spatial and temporal MAC architectures and benchmarking them on a 28nm process using a data-gated 8b-input MAC as a baseline. Throughput is roughly increased quadratically for cutting precision by a factor of 2, reaching up to 14.5× for 2b precision. Area is increased 1.1-4.4× for parallel precision-scalable designs while bit-serial implementations can reduce the area by up to 40%. The overhead for scalability-support increases the energy per operation in full precision modes by up to 52% for single-level, up to 94% for dual-level scalability, and up to 14× for multi-cycle bit-serial MACs. The energy per operation only reduces linearly with precision in the baseline but decreases supra-linearly for the scalable MACs, achieving up to 4× lower energy at 2b precision (overhead compensated at 6-4b precision for parallel and at 2b for bit-serial MACs). An XNOR-based 22nm BNN accelerator [137] exploits the reduced precision by utilizing SCM instead of SRAM, enabling lower power consumption. Similarly, the 65nm binary-weight (12b activation) CNN accelerator YodaNN [64], reports improved performance using voltage-scalable SCM. Combining binary weights (instead of 12b) with SCM (replacing SRAM) allows them to reduce the power by 11.6x. Cross-layer bit-width optimization [198], shows more than 20% parameter size reduction compared to homogeneous bit-width fixed-point quantization at iso-accuracy on the CIFAR-10 task. Furthermore, knowledge distillation can be used for low-precision quantization [199], improving the accuracy of a highly quantized model using "distilled" knowledge from a larger (higher precision) teacher network during training.

2.8.5 Approximate Computing

Approximate computing trades power consumption, speed, and area off against arithmetic accuracy [200]. Approximation approaches are

either based on voltage over-scaling (VOS) below the technology's threshold voltage or on functional modifications ranging from algorithm- to circuit-level [201]. It differs from energy-quality-scaling (Section 2.8.4), due to its circuit-based scaling approach. A 2020 survey [202] on approximate computing for DNNs reports power, delay, and area numbers from synthesized approximate adders, multipliers, and dividers using a 28nm process at 1V. It reports up to 69% energy savings (power-delay product) for an image sharpening task while for JPEC compression only 20% savings are achieved. An earlier survey [201] reports an approximate integer data format and related arithmetic operation implementations in 45nm [195], limiting values and computation precisions to a dynamically selected range of most significant non-zero bits. This achieves 55-65% power reduction compared to accurate computations at <0.5% accuracy drop in KNN and SVM tasks. Approximate computing using 2- and 3-bit adder designs [203] reports further power and accuracy improvements. VOS introduces bit errors due to missed timing constraints and other unwanted effects but reduces power consumption as shown on a 28nm CNN accelerator [204]. Reducing the SRAM voltage from nominal 1.0V to 0.51V enables 3.12× memory and 2.13× system power reduction for running a 9-layer fully binary CNN at <1% accuracy drop. They report stronger effects on accuracy from weight errors than from activation errors, enabling further activation memory voltage scaling.

2.8.6 Non-Conventional Arithmetic

To further optimize NN computations, non-conventional computer arithmetic has been surveyed [205], comparing currently used CMOS technology and alternative emerging technologies for implementing computer arithmetic. Also alternative number systems are evaluated, for example, a logarithmic system on a 65nm process [206], showing 3× higher energy per addition compared to floating-point, but 1.5× lower for multiplications, 17× lower for divisions, and 38× lower for square root computations.

Spiking Arithmetic

Biological neurons in the human brain function with spike-based signaling and computing, inspiring researchers to rethink the traditional level-based arithmetic in ICs [207]. Neuromorphic spiking arithmetic is employed in IBM's 28nm TrueNorth [208], implementing a total of 1 million digital spiking neurons, and Intel Loihi [209], implementing 131k neurons in a 14nm process. Due to the significantly different computing paradigm, which cannot be directly compared with other optimization approaches, we refer the reader to the specific literature for more details [207, 210, 211].

Hyperdimensional Computing

Hyperdimensional computing is another brain-inspired computing approach that encodes information in very high-dimensional binary vectors with thousands of entries, called hypervectors. The similarity of information contained in hypervectors is encoded in a distance metric, making them robust against bit errors and thus suitable for VOS. Tasks like image recognition are performed by comparing the distance of an input vector (e.g. features of an image) with a known reference vector. We refer to the specific literature [205, 212] for more details as this goes beyond scope of this chapter.

Quantum computing

While still being in its infancy, quantum computing [213] promises extremely powerful computing capabilities, enabling unprecedented throughputs which could allow further acceleration of NN computations in the future.

2.8.7 Mixed Signal Arithmetic

The analog nature of most sensor signals and power advantages of computing in the analog domain motivated mixed signal arithmetic for computing DNN [214, 215]. Survey [57] compares a set of analog DNN accelerator architectures, reporting 40-80% lower area, as well as 70% and 40% reduced power compared to digital implementations for 130nm and 65nm designs, respectively. This shows that analog

circuits do not scale equally well with reduced process nodes as their digital counterparts. Other properties, like the intrinsic computation parallelism from Kirchhoff's law, can compensate for this reduced advantage in smaller node sizes.

Analog implementations usually require peripheral circuits that can diminish analog computation advantages at increasing design efforts, as illustrated in [216, 217], implementing the same accelerator in a 28nm process but using analog or digital MAC-accumulation circuits, respectively. Evaluated on a fully binary CIFAR-10 network at iso-accuracy, the energy per inference dropped from 14.4uJ (digital) down to 3.79uJ (analog), which is a system-level energy improvement of 3.8x, while the energy for the underlying MAC computation dropped by nearly 12.9× (>3× more). The 28nm analog-domain computations (8b dot product) are presented in [218], reporting nearly 75% energy spent on ADC and control logic.

Bong et al. [219] implement a hierarchical analog-digital hybrid binary decision tree engine on a 65nm image sensor, running 60% of the algorithm in the analog domain, reducing the inference energy by 39% compared to digital computation.

A 130nm 32x32 analog MAC array multiplying a DAC-converted vector with a 32x32 matrix is presented in [220]. Compared to a multi-core processor baseline, power and area are reduced by 71% and 43%, increasing throughput by 10.3x.

The CIM approaches discussed in Section 2.4.3 exploit the advantages of mixed signal processing, keeping the data in memory to avoid losses from data movements and digitalization losses. An RRAM-based analog crossbar [221], compares performance to equivalent implementations with digital RRAM-usage and SRAM-based memory. It reports 270× energy and 540× latency improvements over digital RRAM, and 430× energy and 34× latency improvements over SRAM implementations, showing latency issues for digital RRAM.

2.8.8 Arithmetic Implementations

NN training is usually executed with 32bit floating point precision, which can be significantly lowered during inference. Each layer has a specific sensitivity to quantization and can thus be implemented with adapted (minimal) bit-widths [222]. Selecting the arithmetic precision

and the data types allows to optimize implementations, increasing throughput and energy efficiency (as shown in Section 2.8.4). Motivated by the finding that the required precision varies across DNN layers [222], a 65nm bit-serial DNN engine is implemented based on the 16bit DaDianNao architecture [223]. Evaluated on 9 common DNNs with per-layer-minimized precision, it reports 1.2x-4.76× (2.0× on average) increased energy efficiency at iso-accuracy compared to the baseline accelerator. Variable-precision bit-serial MAC can also be implemented using look-up tables [65], reporting energy savings of 23%, 27%, 41% and 54% with respect to standard fixed-point MAC for 16-, 8-, 4-, and 1-bit weight precision, respectively (16b activation). A similar, Booth-Wallace multiplier-based multi-precision implementation was shown in [62], supporting 16b multiplications, that can be split into 4·4b multiplications.

2.9 Quantitative Comparison

This section summarizes the quantitative effects of the discussed edge ML accelerator optimizations. Table 2.6 - Table 2.8 list each optimization approach with a brief description of the implementation setup that was used to demonstrate the effect in the referenced publication. Five performance indicators quantify each technique: the memory usage impacts the (often dominating) energy for data handling, the throughput determines the processing latency, the chip area directly impacts manufacturing cost, and the power/energy reductions translate into longer battery lifetimes. However, optimizations might influence the algorithmic accuracy, which is therefore listed in the fifth impact column. While most works either do not influence the accuracy or report their optimization performance at iso-accuracy levels, some approaches significantly deteriorate algorithmic performance. Reduced accuracy must be carefully traded-off against performance improvements of the optimization, which might be complicated by non-comparable benchmarks used across different publications.

Architectural optimizations (Section 2.4) report up to 13× power savings with increased throughput using CIM. Note that currently, most implementations only support small networks. Power management (Section 2.5.2) allows for significant leakage and dynamic power

reductions, but negatively impacts the throughput due to the reduced operating frequency. Optimizing the memory offers reduced access energy and mainly lower static power while affecting the required area. Optimized placement and low-leakage SRAM types allow trading area off against power. Various dataflow and data handling options (Sections 2.6-2.7) offer improved throughput or reduced memory requirements (up to 10'000× lower size). Computation improvements (Section 2.8) report higher throughput and efficiency using quantization but must be traded off against accuracy deteriorations.

As an example, the designer of a new edge ML accelerator, requiring to run low-complexity ML tasks at very low power consumption, could identify DVFAS [62] in Table 2.6 as useful optimization, reporting 25× power reduction at constant throughput. The reported accuracy impact must be considered but might be acceptable for simple tasks. To further improve the power consumption, SCM can be chosen instead of standard SRAM, adding another 2-3× reduction in (memory) power consumption [137]. If the selected network tolerates sparsity, the approaches exploiting weight and activation sparsity from Table 2.7 might be another option to drastically decrease the power further. However, replacing digital processing with a mixed-signal implementation from Table 2.8 has lower expected power gains and probably comes at the cost of an increased area, as reported in the literature [216].

2.10 Conclusion

This chapter presented a quantitative survey of design optimization strategies for low power NN accelerators. It evaluated each approach based on five key performance indicators, namely memory reduction, throughput increase, area reduction, power savings, and algorithmic impact. The compiled list of optimizations allows comparing these quantitative performance measures across all other approaches, enabling accelerator designers to estimate their impact during the design process. Reported optimizations range from 10'000× memory reduction to 33× energy savings, and illustrate the wide range of proposed optimization techniques.

Table 2.6: Overview of optimization strategies

Field (Sec.)	Optimization approach	Reported impact					Reference work		Remarks
		Memory usage reduction	Throughput increase	Area reduction	Power/ energy reduction	Algorithmic	Work (Year)	Implementation performance	
Architecture (Sec. 2.4)	Replace systolic array PEs with tensor PEs	-	-	2.1×	1.4× system	-	[81] 2020	16nm accelerator running various nets (e.g. ResNet-50, MobileNetV1)	
	Replace MAC with CIM-MAC	-	increased	unknown	13× system	impacted (quantization)	[87] 2017	128x128 CIM array in 130nm running MNIST task (1b w. , 5b act.)	Tiny network presented only
	Replace MAC with SRAM CIM-MAC	-	2.4×	unknown	4.9× system	negligible	[88] 2018	LeNet5 network running MNIST task on simulated 65nm accelerator	
	Add 8b CIM to pure SRAM	-	increased (none before)	0.78× (28% larger)	unknown	-	[89] 2021	384kB SRAM with 8b CIM in 28nm process	
Power management (Sec. 2.5.2)	Sub-threshold operation	-	0.23× (4.4× slower)	0.53× (1.87× larger)	5× power	-	[112] 2013	180nm sub-threshold standard cell compared to standard 180nm library	
	Sub-threshold operation	-	reduced	-	7× power	-	[114] 2016	180nm MCU: 13kHz @ 0.48V vs. 25MHz @ 1.8V	
	Sub-threshold operation	-	reduced	-	3.8× power	-	[113] 2015	180nm 1kb SRAM @ March test: 530Hz @ 0.27V vs. 200kHz @ 0.6V	
	ABB	-	Reduced (30× lower)	-	20× leakage	-	[115] 2019	55nm RISC MCU and SRAM	
	DVFS	-	-	-	1.05-4×	-	[117] 1994	CPU @ varying intensity using DVFS	
	DVFS	-	-	-	9.2×	-	[118] 1996	CPU @ 1.5V and 1/10 frequency DVFS vs. CPU @ 3V and 90% idle	
	DVFAS	-	- (constant)	-	25×	Impacted (quantization)	[62] 2017	28nm accelerator running 4x4b with DVFAS (@0.65-1.1V, bias: 0.2-1.2V) vs. 1x16b @ constant throughput	
	DVAS	-	Beneficial	0.85-0.9× (<15% more)	11.1x(16b->8b) 0.8-0.9× (@16b)	Detrimental	[119] 2015	40nm 16b Baugh-Wooley multiplier with DVAS vs. gating unused bits	E. overhead due to added logic
Memory (Sec. 2.6.3)	Reduce memory size for lower access energy	impacted	-	with memory	Linear with memory size	-	[120] 2016	45nm SRAM access energy for	
	Replace 6T SRAM with FeFET	-	1.5× reading 0.1× writing	>5.3×	74× stat. power, 0.6×/0.1× read/write	-	[132] 2019	22nm 64x64b array: FeFET vs. 6T SRAM (FeFET reading/writing: @4V/1V)	Read/ write energy increases
	Reduce SRAM area: low leak. 7T vs. 8/10T	-	similar	18% 50%	similar	-	[24] 2019	8kB SRAM in 180nm using 7T vs. 8T and 10T low leakage SRAM	
	Non-uniform SRAM instead of uniform	-	-	0.98 (2% larger)	60% (1.5kB vs. 48kB access)	-	[133] 2017	4-level SRAM memory (1.5-48kB) in 40nm at 0.65V and 3.9MHz	
	ULP SRAM sleep mode	-	Detrimental	0.37× (2.7× larger)	26×	-	[134] 2020	16kB SRAM with charge pump for low power retention in 55nm	Slow process corner is used
	Reduce SRAM leakage 180nm/10T vs. 65nm/8T	-	unknown	0.09× (11.3× lager)	4242× retention	-	[103] 2013	Low leakage 180nm 10T SRAM vs. 65nm 8T SRAM	
	SRAM: transpose vs. row-only	-	-	Detrimental (add. logic)	1.9×	.	[135] 2018	65nm accelerator with T-SRAM vs. SRAM for 5x1 & 1x5 sep. conv.	
	Replace SRAM with SCM	-	similar	2-3× (<1kb), <1/3× (>1kb)	>2×	-	[136] 2016	8kb SCM vs. SRAM in 28nm using dedicated SCM placement	
	Opt. placement for SCM	-	similar	>35%	Read: ≤65% Write: ≤50%	-	[136] 2016	256b-32kb SCM macros in 28nm: special placement vs. standard	
	Replace SRAM with SCM	-	-	Not reported	2-3×	-	[137] 2018	BNN accelerator in 22nm using SRAM@0.6V or SCM@0.4V	
	3D HMC DRAM instead of 2D DRAM	-	4.1×	Beneficial (3D-stacked)	1.5×	-	[139] 2017	HMC DRAM dies on 45nm logic die running various networks	
	Replace SRAM with eDRAM (4T)	-	-	17%	23%	-	[140] 2018	28nm 8kb eDRAM array vs. SRAM (both@0.7V and room temperature)	
	Replace ULP SRAM with eDRAM (2T)	-	unknown	2.5-3.8×	Same leakage	-	[141] 2019	28nm 4kb eDRAM memory array at 0.4Vvs. SRAM	Simulation results only

Table 2.7: Overview of optimization strategies

Field (Sec.)	Optimization approach	Reported impact					Reference work		Remarks
		Memory usage reduction	Throughput increase	Area reduction	Power/ energy reduction	Algorithmic	Work (Year)	Implementation performance	
Dataflow & Control (Sec. 2.6)	Depth-first instead of layer-wise processing	Size:≤ $10^4\times$ BW:≤ 200×		Linked to memory	-	-	[144] 2019	Memory reduction for UHD-res. networks: e.g. SRGAN: 19'633×[1], MobileNetV2: >20x1	Limits number of supported layers
	Depth-first instead of layer-wise processing	On-chip:0.8×, Ext.: 20×	(0.94x)	-	Beneficial (less ext. memory)	-	[146] 2016	VGGNet-E on FPGA	AlexNet: only 28% ext. reduc.
	Early data reduction: opt. conv. in 1st layer	2.5–10× (MNIST/ EMNIST)	250-460× MNIST/ EMNIST)[2]	-	Linked to throughput	-0.4% /-1.7% (MNIST/ EMNIST)	[154] 2020	Optical convolution used in first layer in front of 4 dense layers on MNIST and EMNIST task	Only works on first layer.
	Early data reduction: analog in-sensor proc. of 1st conv. layers	-	-	-	73% sys., 85% sens., 45% proc.	Detrimental	[156] 2016	Simulated 180nm SAR ADCs and tunable capacitors running first 1-5layers of 8b GoogleNet ConvNet	
	Add hierarchically scalable effort	Detrimental (more nets)[3]	Beneficial	-	~10×	-	[62] 2017	28nm DNN accelerator running 12MOP – 30.8GOP networks	
	Add hierarchically scalable effort	Detrimental (more nets)[3]	1.2-9.8× (avg.)	-	Linked to throughput	-	[163] 2015	Hier. scaled SVM, NN, and dec. tree on MNIST and other tasks	Overhead for difficult inputs
	Introduce early exit for conditional execution	Detrimental (new layers)[3]	1.73-1.91×	-	Linked to throughput	Slightly increased	[160] 2016	6- and 8-layer NN for MNIST task with 1 and 2 early exits	
	Introduce early exit for conditional execution	Detrimental (new layers)[3]	-	-	1.22×	Slightly reduced	[161] 2020	12-class Google speech command task with early exit on 22nm acc.	
Data handling (Sec. 2.7)	Efficient CNN memory mapping	Up to 2×	-	- (with memory)	-	-	[28] 2020	Act. (total) memory reduction e.g. DMCNN-VD: 48.8% (48.2%)	
	Compression using Huffman coding	1.2-1-3× for weights	-	- (with memory)	-	-	[164] 2016	Encode weights using Huffman coding in AlexNet and VGG16	
	Compression using Huffman coding	Up to 5.8×	-	- (with memory)	-	-	[62] 2017	28nm DNN accelerator running face recognition CNNs	
	Compression using RLC coding	1.2-1.9×	-	- (with memory)	-	-	[63] 2016	65nm process running AlexNet	
	Weight sharing	<4× for weights	-	-	8×	Similar	[61] 2016	45nm accelerator with weight sharing (16 entries of 16b values)	
Computation (Sec. 2.8.1)	Replace CNN with depth-wise sep. conv.	>7×	8.5×	Linked to memory	-	1%	[16] 2017	MobileNet vs. pure CNN on ImageNet task	
	Separable convolution instead of 5x5 (1x5, 5x1)	Beneficial (less param.)	4× (1.7× on total network)	-	-	-1%	[135] 2018	5-layer CNN for LFW face recog. with sep. conv. vs. normal conv.	
	FFT-based convolution	Detrimental (more buffer)[3]	1.75-5.3×	- (with memory)	Linked to throughput	- (iso-accuracy)	[167] 2014	Computing 3x3 – 11x11 CNN kernels via FFT vs. conventional	Smaller kernels profit less
	Opti. FFT-conv.: fine-grained FFT	Beneficial	0.93-1.74×	-	Linked to throughput	-	[168] 2020	Fine-grained FFT-based conv. vs. pure FFT conv. (9x9–3x3 kernels)	
	Use Winograd fast convolution	-	1.48-2.26×	-	-	Negligible normally	[166] 2016	Winograd 3x3 convolution on VGG E network (batch size 64-1)	Accuracy drop for large kern.
	Use Strassen algo. for matrix mult. in CNNs	-	24-47%	-	-	-	[169] 2014	Strassen algo. on AlexNet conv. layers 2-5 (5x5, 3x3) on CPU	More eff. for large matrix
Computation (Sec. 2.8.2)	Sparsity exploitation + pruning	9-13× for weights	Beneficial	-	-	- (iso-accuracy)	[177] 2015	Network pruning on AlexNet and VGG16 for ImageNet	
	Sparsity exploit.: skip memory access + MAC op.	-	Beneficial	-	1.6×	-	[62] 2017	28nm DNN accelerator with 1b sparsitiy map (16b activations)	Assumes 30-60% sparsity
	Sparsity exploit. energy-aware pruning	11× weights	unknown	- (with memory)	3.7×	<1%	[178] 2017	AlexNet on ImageNet task	
	Skipping sparse operations	Beneficial (sparsity)	3.68×	-	-	Impacted (quantization)	[179] 2019	28nm CNN acc. with sparsity map and NZVL compression	Simulation only
	Add zero-value skipping logic	Beneficial (1.25× more)	1.37×	Similar (adds logic)	Beneficial	-	[180] 2016	65nm acc. with zero-skipping logic and zero-free data encoding	

[1] at equal memory bandwidth (lower for lower memory bandwidths) [2] with respect to largest network with similar performance
[3] higher memory usage due to increased number of networks and layers to be executed or larger data to be buffered

Table 2.8: Overview of optimization strategies

Field (Sec.)	Optimization approach	Reported impact					Reference work		Remarks
		Memory usage reduction	Throughput increase	Area reduction	Power/ energy reduction	Algorithmic	Work (Year)	Implementation performance	
Computation (Sec. 2.8.3)	Add data reuse: row-stationary processing	Detrimental (local mem.)[3]	Beneficial	Detrimental	1-4-2.5×	-	[63] 2017	65nm 168 PE row-stationary accelerator running AlexNet	Data in SRAM and ext. DRAM
	Add data reuse: output stationary	Detrimental (local mem.)[3]	1.87×	0.3× (5.5× larger)	1.66×	-	[60] 2015	65nm 64PE OS acc, with vs. without reuse on various CNNs	Savings depend on network
	Add data reuse: NoC for flexible reuse modes	Detrimental (local mem.)[3]	5.6×	Detrimental	1.8×	-	[66] 2019	65nm NN accelerator with 192 PEs running MobileNet	
	Data reuse: change-based temporal sparsity	Detrimental (local mem.)[3]	8.6×	- (with memory)	-	-	[181] 2017	On 5-layer CNN for 10-class scene segmentation	Entire net stored in memory
Computation (Sec. 2.8.4)	Reduce precision int32/float32 to int8	Beneficial	Unknown	Unknown	14× (int32), 20× (float32)	Impacted	[102] 2014	45nm arithmetic compared	
	Reduce precision int32/int16 to int8	Beneficial	Unknown	13× (int32), 3.6× (int16)	10.8× (int32), 3.6× (int16)	Impacted	[195] 2017	45nm arithmetic compared	
	Reduced precision (16b→8/4b)	Up to 4×	Up to 4×	-	>5× (16b->8b), >50× (16b→4b)	Accuracy impacted	[62] 2017	28nm acc. with sub-word parallel 1-16b MAC precision	
	Replace 16b multiplier with 4bx4b LUT	-	14.3×	1.3×	4.7×	Impacted (quantization)	[196] 2019	65nm acc running ImageNet task (quantized vs. 32b baseline)	LUT: 256 pre-comp. 16b results
	Replace MAC with scalable precision MAC (8b→8/4/2b)	-	Up to 14.5× (2b)	>0.22× par.), <1.4× (ser.)	up to 4× (2b)	Accuracy impacted	[197] 2019	28nm scalable prec. MAC designs (parallel and serial) vs. 8b MAC	2-14× power overhead @ 8b
	Reduce weight precision 12b→1b, replace SRAM→SCM	Weights: 12×	similar	1.2×	11.6×	Impacted	[64] 2018	Binary weight accelerator in 65nm with SCM	
	Energy-quality scaling	-	-	-	3.5×	2% reduction	[190] 2020	Voice activity detection using decision trees on 28nm chip	
	Energy-quality scaling: using det. thresh., feature/mem. size	-	-	-	3×	Negligible	[191] 2020	40nm image feature detection	
Computation (Sec. 2.8.5)	Approx. computing	-	-	Beneficial	20%-69% (pow. delay product)	Accuracy impacted	[202] 2020	28nm approx. mult./adders run. image sharpening, JPEG compr.	
	Approx. computing	-	-	Beneficial	55% (KNN)-65% (SVM)	Accuracy impacted	[195] 2017	45nm approx. comp. engine vs. accurate 32bit implementation	
	Approx. computing: SRAM VOS	-	0.095× (10.5× slower)	-	3.12×	Impacted (<1% lower)	[204] 2018	28nm fully binary CNN acc. running 9-layer CNN	
Computation (Sec. 2.8.7)	Replace digital with mixed-signal arithmetic	-	1.58×	0.31× (3.2× larger)	3.8× (system), 12.9× (MAC)	- (iso-accuracy)	[216] 2018	28nm fully binary CNN acc. with analog MAC vs. digital [191]	Analog periph. reduce gain >3×
	Replace float arithmetic with log.	-	unknown	unknown	0.33, 1.5, 17× (add, mult, div)		[206] 2016	65nm logarithmic arithmetic system vs. conventional float	
	Replace digital with mixed-signal arithmetic	-	-	-	1.39×	Impacted	[219] 2017	65nm mixed analog/digital comp. vs. pure digital face detection	60% of algo. in analog domain
	Replace digital with mixed-signal arithmetic	-	10.3×	1.7×	3.4× power	-	[220] 2011	130nm 32x32 analog MAC array for matrix-vector multiplication	Input/ output in digital domain
	Replace dig. SRAM MAC with analog RRAM- CIM	-	34×	11×	430×	Deteriorate significantly	[221] 2018	Simulated accelerator design in 16nm, running MNIST task	
Computation (Sec. 2.8.8)	Replace digital with mixed-signal arithmetic	Beneficial	1.3-5.3	0.95×	1.2-4.76× (2.0× average)	- (at iso-accuracy)	[222] 2017	65nm var. perc. bit-serial acc. vs. 16b baseline	Evaluated on 9 common DNNs
	Replace MAC with var. prec. LUT-based bit-serial MAC	-	-	-	1.23-1.54× (16b-1b weight)	-	[65] 2019	65nm NN acc. with var. precision MAC vs. fixed 16b MAC	Activations: 16b

[3] higher memory usage due to increased number of networks and layers to be executed or larger data to be buffered

Chapter 3

FPGA-Based Binary Neural Network Implementation

Streaming high-speed cameras pose a major challenge to distributed cyber-physical and IoT systems, because large data volumes need to be transferred under stringent real-time constraints. The previously introduced edge processing concept can mitigate this data flood by extracting relevant information from acquired image data directly on-device with low latency.

This chapter presents the advantages of edge processing on an FPGA-based 20kFPS streaming camera system, which can classify regions of interest (ROI) within a frame using a BNN. Executed in real-time streaming mode, the classification enables substantial data reductions and related energy savings. BNNs are shown to enable energy-efficient image classifications for on-device processing. Demonstrated in a case study with a simple real-time BNN classifier, our system achieves 19.28 us latency at 0.52 W power consumption and reduces the data transfer by 980×. We compare external image processing with this result, showing 3× energy savings, and discuss the used HDL/HLS design flow for efficient BNN implementation.

3.1 Introduction

Communication-intensive applications like video surveillance and computer vision-driven automation are largely limited by the bandwidth of their communication interfaces. The emergence of IoT concepts, which aim at connecting large numbers of devices and allowing them to communicate with each other, further exacerbates the problem by increasing the number of data streams. The reason for still transferring unprocessed data from peripheral devices to computers and servers mainly lies in the higher computational capacities of these cloud computing resources. Some tasks though, especially in computer vision applications, require low latency and feature high data volumes that cannot be sustained on standard communication links.

Edge processing [18] is an alternative to moving the data processing part of a system closer to the source or the edge of a network. This allows large data volumes to be processed, and relevant information to be extracted, before being transmitted over communication links. Additionally, this approach decreases latency, which is important for real-time tasks [224]. Looking at the use case of a streaming camera, edge processing means that sampled images are analyzed directly onboard the device. Beside image or video compression algorithms, also image classifiers based on NNs can be used for this task [225]: The hundreds (or even thousands) of pixel values in each image can then be reduced to less than a dozen bytes, containing the image class.

As a motivating example, consider a 1 megapixel camera with 10-bit resolution streaming out a video at 2kFPS. Such a device continuously generates 26.2 Gbit/s of data. Neither a current Wi-Fi interface, achieving throughputs of up to 1.7Gbit/s [226], nor a state-of-the-art wired USB 3.1 communication link, with a claimed theoretical bandwidth of 10 Gbit/s, is capable of supporting such raw data rates. With on-board 1000-class image recognition, like it used in ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) based on the ImageNet dataset [7], the data stream could be reduced to 20 kbit/s (10 bits per frame), allowing many cameras to be transmitting data simultaneously.

NNs, or more precisely CNNs [227], are often used algorithms for such image recognition tasks. But because of their computational complexity, NNs are most frequently processed on powerful servers

in the cloud, requiring the images to be uploaded through a communication network. As previously shown, this is not feasible for high-speed cameras and therefore requires local image classification. On size and power-limited devices, mobile GPUs and specialized image processing chips can run image analysis at low frame rates but do not sustain high-speed image classification [228]. The recently introduced BNNs though, significantly reduce the computational complexity of networks, reaching high frame rates when implemented on FPGAs [228].

Increasingly larger and more complex NN architectures are being proposed for accurate image classifiers, which sometimes even surpass human performance in some image recognition tasks [1]. Deep Neural Networks (DNNs) shifted the trend from shallow networks with very few layers to deeper ones with tens or up to hundreds of layers [36, 37, 229] which need to be stored on external dynamic random-access memory (DRAM) due to their large size. Internal on-chip memory would be much faster and more power-efficient but usually does not support the size of such larger DNNs.

Thus, classifying images with a NN is a computationally intensive task with a substantial memory footprint, challenging the used hardware architectures. High-performance CPUs, GPUs, and FPGAs with fast DRAM memory are often used architectures for computing large size networks but consume vast amounts of power [230]. One of the main performance limitations in these systems is the extensive external DRAM memory access [231], often referred to as "memory bottleneck". In many real-time applications, this memory access dominates the power consumption [61], which is a big challenge for energy-constrained mobile systems.

To mitigate the memory and power limitations, different approaches have been proposed. On one hand, application-specific hardware is being developed, providing energy-efficient processing elements [61]. On the other hand, quantized networks are tackling the problem by reducing the size of the network through parameter quantization [232]. Thus, parameters can be represented with lower precision numbers, reducing both the required storage space as well as the bit widths of computational elements. The recently introduced BNNs exploit this advantage by limiting weights and activations to 1-bit values [232,233]. These modifications substantially lower the hardware requirements

compared to standard 32-bit architectures: Memory needs are reduced by 32× and the dominating MAC operations can be simplified to logic XNOR operations with appended bit-counting [233]. Simpler computations increase the throughput as each operation consumes less time, while lower memory needs allow parameters to be stored on-chip, reducing external memory accesses. Thus, the low memory footprint directly reduces the problematic effects of memory bottlenecks. Summarized, BNNs provide higher computational efficiencies and thus enable low-power ML applications.

While CPUs and GPUs were designed to efficiently process 32-bit arithmetic operations, their instruction sets do not natively support low-precision arithmetic [228]. FPGAs and ASICs can handle such operations more efficiently as they provide arbitrary bit widths for arithmetic and logic operations (such as XNOR). Moreover, the reduced size of learned weights in BNNs allow meaningful networks to be fully stored on internal block RAM (BRAM), avoiding expensive external DRAM transfers [228], as it would be required for standalone CPUs. BNNs implemented on FPGAs are therefore fast and power-efficient solutions for on-board processing. ASICs have similar advantages as FPGAs and can even be designed more efficiently but are in most cases not affordable due to their time-intensive and very costly manufacturing, while being less flexible than FPGAs. However, one of the main drawbacks of FPGAs is the time-intensive firmware development in HDL. Therefore FPGA manufacturers introduced HLS tools, which allow faster coding in higher-level languages with automatic compilation into HDL code [234]. This step substantially reduces the development time and adds efficient throughput optimization capabilities to the design process.

This chapter presents a complete implementation of a real-time image recognition system integrated in an FPGA-based 20kFPS high-speed camera. To the best of our knowledge, this is the first such system, capable of processing all data on-board at full frame rate for performing BNN-based image recognition. We demonstrate that even for multi-kFPS cameras, a completely configurable engine for image acquisition, ROI extraction, and BNN inference can be implemented on a single FPGA while achieving low power consumption and satisfying the high frame-rate requirements. The real-time performance enables edge processing, enormously reducing the communication data

traffic and ensuring privacy as raw images do not leave the device. We present a complete case study of the deployment of our engine on a real high-speed camera. Experiments demonstrate 980× on-board data reduction, leading to 3× energy savings compared to raw data streaming, showing the applicability of BNNs for edge processing in high-speed camera applications. The rest of this chapter is organized as follows. Section 3.2 discusses related work and in Section 3.3 we present the implemented system as well as our case study. Experimental results are presented in Section 3.4.

3.2 Related Work

The motivation of edge processing is well described by Shi et al. [18], illustrating the vast amounts of data being produced by IoT nodes, making efficient on-board processing inevitable. Their overview of edge processing applications suggests performing video analytics close to the data source, as we propose in this chapter. Ananthanarayanan et al. [224] elaborate on large-scale video analytics in huge networks and conclude that real-time video processing might even be "the killer application" for edge processing. Soro et al. [225] provided an extensive survey of the challenges in such visual sensor networks in 2009, already naming resource constraints as the biggest problem for large camera networks. One of the discussed solutions is again on-board video processing. This proposal is supported by Mohan et al. [233], estimating the number of cameras in the world to reach 13 billion by 2030 and thus pushing the video traffic portion in relation to the overall internet traffic to more than 83% in 2020.

The problem of large data bandwidths gets worse for high frame rates and further increases if multiple high-speed cameras are to be used in a network, as for example Noda et al. describe in their vehicle-tracking application [235]. By extracting position features from sampled images and only transmitting this information, they manage to transmit 1kFPS video streams from multiple cameras through a simple Ethernet link. Stevanovic et al. show that on-board image processing is further useful for low-latency applications, like image-based triggering for high-speed X-ray imaging [236]. Their image analysis extracts features that are used to self-trigger video acquisitions, making it

possible to record activities with a very low latency, which external processing cannot support due to long transmission delays.

In many vision applications, traditional image compression techniques are used for data reduction [226]. They are based on efficient representations of the contained information or the exploitation of limitations in human visual perception. Embedded image compression for high-speed image sensing is presented by Mosqueron et al. [237] but only reaches compression ratios of 1:30, which is much smaller than what can be achieved with image classification. Nevertheless, traditional compression could be used as an orthogonal optimization, for pre-processing the data in order to keep the classifier as simple as possible.

For more complex real-time image recognition tasks, NN-based inference engines have been presented. Cavigelli et al. [238] show real-time scene labeling based on CNNs, achieving state-of-the-art performance on an embedded GPU. In their more recent work [181] they additionally exploit frame-to-frame redundancies in static camera settings, which further improve their power efficiency. Also, BNNs have been shown to perform real-time image recognition, as Mazare et al. [239] present in their object-orientation analysis system, based on a small BNN, implemented on an FPGA. Fast CNN-based object trackers, such as YOLOv2 [240], manage to reach a few tens of frames per second on embedded GPUs. However, all listed approaches reach frame rates that are orders of magnitude too low for high-speed cameras, producing tens of kilo-frames per second.

Implementing reduced precision NNs for image recognition tasks has been proposed in many recent papers. This field of research mainly started to expand after Courbariaux et al. [232, 237] successfully presented BNNs achieving near state-of-the-art accuracy in image recognition tasks. Since then, different approaches for improving quantized NNs have been published. Nurvitadhi et al. evaluate the applicability of BNNs on different hardware architectures, namely FPGA, CPU, GPU and ASIC [228]. On FPGAs, BNNs can be implemented in on-chip RAM while other architectures require external DRAM access, consuming 172× more energy than the floating-point multiplication of the fetched number [228]. For networks with many memory accesses, DRAM dominates the power consumption and thus creates a big disadvantage. Their results confirm the superior performance on

FPGA and ASIC implementations. Nevertheless, quantized NNs have also been optimized on mobile CPU architectures [241], improving the inference time compared to a full-precision implementation by more than 3x. Many FPGA implementations have been presented [230, 233, 234, 242–244], showing highly energy-efficient performance reaching continuous computational efficiencies of 400GOPS/W and more, as shown by Umuroglu et al. [233]. Baskin et al. [244] present a BNN implementation that scales better for larger networks, allowing more complex systems to be implemented on FPGAs. Other work focuses more on multi-bit quantization, like ternary NNs [245], which achieve slightly higher accuracies than BNNs at the cost of higher computational and memory costs. The trade-off between accuracy and network quantization has inspired Mishra et al. [246] to propose wider networks for compensating accuracy loss due to quantization. By operating in the region between the two extremes, namely BNNs and full-precision networks, they can balance accuracy versus computational cost.

3.3 Implementation on FPGA-Based High-Speed Camera

In this section, we discuss the proposed FPGA-based BinaryEye streaming camera with on-board image recognition. Instead of streaming all information through the communication interface, BinaryEye reduces the time and energy-intensive data communications by introducing (on-device) edge processing. Classifying an image reduces the data volume from hundreds of bytes, representing each pixel value, to a single classification result. Our camera was originally designed as a streaming system, buffering sampled images in DRAM and transferring them through a USB interface. This version serves as a reference for experimentally benchmarking our proposed on-board processing implementation. Due to the low computational requirements and the small memory footprint of BNNs, we were able to implement edge processing within the existing FPGA and without using power-intensive DRAM. Fig. 3.1 shows the main building blocks of the system architecture. The data flow starts at the image sensor, where pixels are

streamed out as soon as an image gets triggered by the camera control block (through a USB command). A line buffer sorts the pixels and forwards complete lines. In the original streaming application, all data is directly transferred to the USB interface (indicated with a dotted line in Fig. 3.1). Because the USB interface might not be able to extract the image data at the speed they are being produced, a DRAM image buffer is necessary. BinaryEye instead extracts the relevant pixels from the line buffer and binarizes the reconstructed image as soon as all lines are available. Binarization of the image is necessary for binary-input classifiers and needs to be processed online. Thus, we implemented a thresholding function for quantization, which suffices for simple applications: If a pixel is darker than a certain threshold, it is labeled black, otherwise, it is set to white, mapping it to the background. The threshold value continuously needs to be adapted to varying illumination levels, which was achieved by using the average pixel value as a threshold. Fig. 3.2 shows examples of sampled images and their respective binarized representations. As soon as an image finishes the binarization step, the BNN classifier is triggered. Upon completing a classification, the result is buffered and transferred via USB. All camera-specific functions, like trigger-scheduling or image sensor configuration, are implemented inside the camera control block. The proposed design offers simple extension possibilities to extract multiple ROIs and process them at the same time by instantiating multiple parallel classifier paths (if the FPGA resources allow it). To present the feasibility in a case study, we implemented and tested a handwritten digit classifier (based on the MNIST dataset), connected to the output stream of the high-speed image sensor. Such a setup could for example be used in a mail sorting system, where handwritten digits on postcards or letters need to be analyzed at a very high speed. Recognized numbers can be combined to postcodes which in turn tell the system where a certain card needs to be directed. Due to the required high throughput for handling thousands of letters per hour, the letters need to move through the system at the highest possible speed, making high-speed image processing inevitable. Especially the processing latency plays a crucial role as the output (e.g., the postcode) is required for controlling the letter-distribution mechanism, which needs to keep up with the pace of the conveyor belt. Transmitting the same image data to a

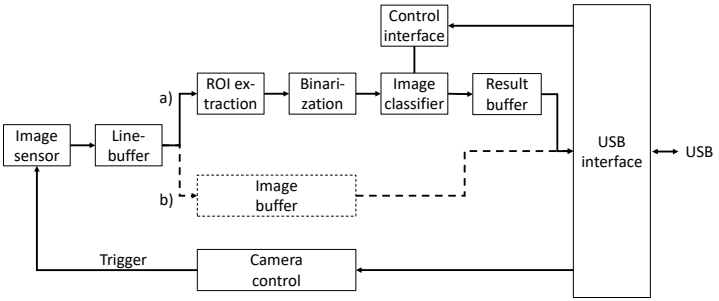

Figure 3.1: System overview showing the data flow. Path a) presents the implemented version with an on-board classifier. Path b) directly transfers images to USB.

centralized classification engine would increase the latency and thus limit the maximum throughput. The implemented system therefore recognizes handwritten digits presented to the camera onboard the device and outputs the classification results. In our demonstration we used an ROI in the top right corner of the receptive field, measuring 32×32 pixels, which is the smallest camera-ROI possible for extracting 28×28 pixel images (as required for benchmarking with the MNIST dataset). Each raw 28×28 pixel image contains 784 pixels with a resolution of 10 bits each, summing up to 980 bytes of data. Image classification enables to represent the image using a single byte for encoding the class information, resulting in a data compression ratio of 980:1. The 10 different classes in a handwritten digit classifier could even be encoded with 4 bits, increasing the compression ratio by 2x. In contrast, the precision of the pixels could only be reduced if the lighting conditions are well matched with the exposure time of the camera. This would require an additional control circuit, increasing the system overhead.

3.3.1 High-Speed Camera System

The presented camera system integrates a custom-made 1 mega-pixel high-speed image sensor. It features a global shutter, allowing smear-free image acquisition of up to 20,000 grayscale frames per second (at a 32×32 pixel resolution). In order to transfer image data at such a high speed, the sensor is directly connected to FPGA peripherals making parallel data readouts of the 10 bit pixel values possible. An integrated ROI functionality enables the camera to process and output a limited region of the whole 1280×1024 pixel image. This reduces the data volume to be transferred, making faster frame rates possible and decreasing the peripheral power consumption. Fig. 3.3 shows the whole system, integrated into a device consisting of the image sensor, power supplies, a USB 2.0 communication interface, and an FPGA, containing the firmware. A Xilinx Kintex-7 XC7K325 FPGA with 326,080 logic cells and 16 Mbit block RAM is implemented in this system.

3.3.2 BNN Image Classifier

As discussed in Section 3.2, different BNN architectures have been proposed to implement image classifiers in FPGAs. Our work does not focus on the BNN classifier itself but its integration in an edge processing system. Thus, we chose to employ an existing handwritten digit classifier for demonstrating our edge processing system. Umuroglu et al. presented a framework for efficiently mapping BNNs on FPGAs (FINN) by converting the required computations into FPGA-friendly operations [233,242]. They made an implementation of this framework publicly available[1], providing HLS code for reproducing a number of example classifiers on a Xilinx PYNQ board. Even though they targeted an ARM-based Zynq platform, we found their published results very promising for adapting it to a pure-FPGA implementation (the processor only interfaces the classifier). Thus, we based our case study implementation on their MNIST classifier "LFC max", which is a fully-binarized three-layer FC network with 1024 neurons per layer. Its input is a binary 28×28 pixel image and it outputs a 10-bit result, each bit representing one of the ten possible digits.

[1] https://github.com/Xilinx/BNN-PYNQ

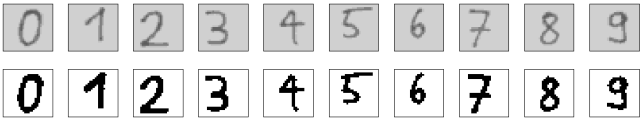

Figure 3.2: Examples of 28x28 pixel images from the camera (top row) and after binarization (bottom row).

Figure 3.3: Image of the presented camera system with a mounted objective.

All network parameters are stored in on-board BRAM, avoiding any external memory access. The provided FINN example is based on a Xilinx Vivado HLS project, which compiles the MNIST classifier, written in C++, into HDL code for the FPGA logic and software code for running the processor. We extracted the compiled HDL code, implementing the BNN, and designed the needed interfaces for embedding the classifier into the existing FPGA camera firmware. As depicted in Fig. 3.1, all network parameters can be loaded via USB through a control interface. This allows network parameter updates to be easily transferred without having to re-program the

device or even changing the HDL code. For the final integration, all HDL components, including the compiled classifier, can be synthesized in Xilinx Vivado and loaded onto the FPGA. This design concept makes the classifier easily replaceable without having to write HDL code and thus takes the quickly evolving nature of (neural) network architectures into account. All that needs to be done for updating the network is compiling the new classifier, replacing the old one in the HDL project, and synthesizing the new implementation, as depicted in Fig. 3.4. Which networks are possible to be implemented on the FPGA mainly depends on the available memory size and the number of logic cells on the FPGA. Our implementation only utilized around one quarter of the available BRAM resources as shown in Table 3.1. According to the resource utilizations presented in the FINN paper, we could also fit their convolutional CIFAR-10 and SVHN classifiers. In fact, much larger BNNs could be implemented on the used FPGA, implementing more demanding tasks for other edge processing applications.

3.4 Experiments and Results

To evaluate the proposed BinaryEye system on the example of the implemented MNIST case study, we compare its performance with other quantized MNIST implementations. Moreover, we present the power-advantages and the latency benefits of on-board image recognition for edge processing. As this work does not focus on the NN itself, but mainly on its implementation for building a streaming edge processing node, classification accuracy plays a secondary role. But we would like to emphasize that even though full-precision networks are often performing slightly better in the benchmarks than their binarized versions, BNNs have been shown to achieve comparable results at much lower computational costs. This is not limited to simple datasets like MNIST, but also applies to more complex networks as for example CIFAR-10 object recognition [232, 233]. Table 3.3 summarizes the performance of state-of-the-art (SoA) MNIST BNN classifiers [230, 233]. For this benchmark, the accuracy measured in this work is less than 1.5% below the highest reported classification rate of 99.79% (using a full-precision network) [247]. Because our classifier network

Table 3.1: Resource utilization of the FPGA

Implementation	**Resource utilization**		
	LUT	FF	BRAM
Original camera	48k (24%)	81k (20%)	14 (3%)
BinaryEye	88k (43%)	115k (28%)	124 (28%)

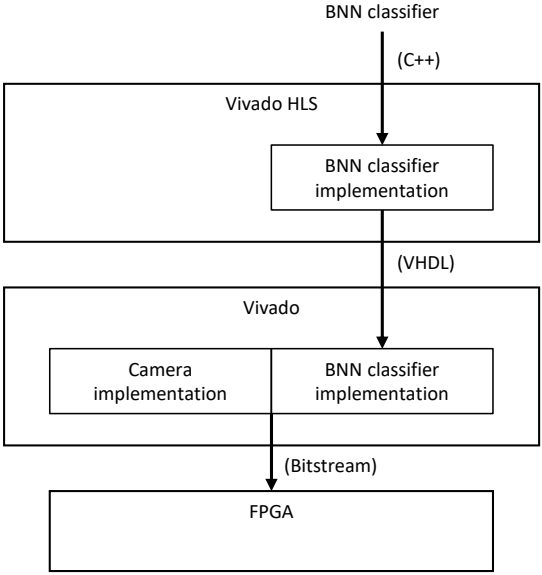

Figure 3.4: Design-flow for implementing a high-level-coded BNN (C++) on the camera FPGA using HLS. Updating the BNN only requires the design to be re-compiled in Vivado HLS and re-synthesized in Vivado.

is an exact reproduction of the FINN implementation by Umuroglu et al. [233], the accuracy of our system is identical to their benchmark performance. But unlike their purely computational implementation of the BNN, our system receives real-world data from an image sensor. Therefore BinaryEye is limited by the camera speed of 20kFPS, while

their fastest classifier implementation reaches 1,561kFPS at 2.44 μs latency. Apart from the lower input frame rate due to the camera, we also clocked the classifier at a lower speed, namely 100 MHz, which is sufficient and more power-efficient for 20kFPS classifications. They clocked the classifier at 200 MHz and thus consumed more than 16× more power. Additionally, we did not use any of the speed optimizations available for this classifier, namely pipelining or adaptions of the network folding parameters to available FPGA resources, which further increased our inference time to 18.49 us. While such optimizations would significantly improve the inference speed due to a higher degree of parallelism, they would negatively impact the power consumption and are therefore not implemented. Our inference time allows 2 ROIs to be classified within the period of a frame, while 78 ROIs per frame would be possible by optimizing the implementation to the published 1,561 kFPS version. Based on the current resource utilization, we expect this version to also be feasible on our FPGA, but at the cost of the mentioned power disadvantages. The unutilized FPGA resources could also be used to instantiate multiple classifiers in parallel, each one for different ROI selections. This would theoretically multiply the image throughput by the number of parallel classification-lines. We expect 3 parallel classification lines to fit inside our FPGA, making a total of 6 ROI classifications possible at 20kFPS.

With a 4× larger network than ours, Liang et al. [230] show a comparable accuracy on the MNIST benchmark but consume considerably more power. For this small network, their computational performance is still lower than what Umuroglu et al. [233] report, but achieves almost 4× higher values on the CIFAR-10 benchmark, making it promising for larger networks. Alemdar et al. [248] achieve similar performance by using a ternary FC network, reaching only slightly lower accuracies but more than 20× higher frame rates. By further reducing the classification accuracy they even achieve a lower power consumption than we present in this work. Compared to our reference classifier from the FINN paper [233], their maximum frame rates are almost 6× lower, making FINN more promising for possible frame rate improvements in the camera. In terms of power consumption, our 20kFPS implementation consumes slightly less than Umuroglu et al. [233] measured on their 12.2kFPS classifier. Even though our

frame rate is higher, the powerful ARM processors on their Zynq FPGA cause some extra consumption. We avoid this by implementing the complete system on FPGA without any processor. In order to quantify the energy savings of edge processing, we compare the power consumption of BinaryEye with the consumption of a pure streaming implementation, which directly transmits all images without any pre-processing (the original camera implementation). Therefore we synthesized the FPGA firmware once with the embedded classifier and once without it (see the dotted path in Fig. 3.1). By evaluating both implementations on the same camera hardware for the task of recording a limited number of 10,000 images at 20kFPS, the energy savings could be determined. The power consumption was measured during the image acquisition and classification phase as well as during the data transfer period. Table 3.2 summarizes the results and shows that edge processing reduces the energy consumption for the given task by a factor of more than 3x. This can be explained by the 980× reduction of the data volume and the associated data transfer time. Although the additional classifier functionalities on BinaryEye are slightly increasing the power consumption compared to the original system, the savings in transmission time still reduce the total energy consumption. This reduction ratio does not depend on the number of recorded images as the energy consumption of each implementation linearly depends on the number of images, making the ratio a constant. Apart from this improvement, it needs to be noted that the system without a classifier could not support continuous image streaming at this frame rate. The image transfer time (1.24 s) is longer than the acquisition time (0.5 s), causing the image buffer to overflow eventually. A faster data interface (e.g. USB 3.1) would reduce the transfer time but still consume more power and possibly limit the maximum framerate. Additionally, the pure streaming implementation outsources the processing power to an external device. This portion could be included in the calculation and would further increase the power consumption. While the reduced power consumption of the edge processing system might not be a big advantage in stationary implementations, its low latency is of paramount importance for real-time applications. The short latency of BinaryEye allows the system to react to an input image within 19.28 μs. If the image gets processed externally, the latency becomes the sum of the transfer time to the external processing

Table 3.2: Comparison between image-streaming and on-board image classification (this work)

Measurement (10k images)	**Original**	**BinaryEye**
Image acquisition: average power [W]	13.26	13.78
Image acquisition: time [s]	0.5	0.5
Data transfer: power [W]	11.96	12.22
Data transfer: time [s]	1.24	0.00024
Data volume to be transferred [kB]	9,800	10
Total energy (acquisition + transfer) [Ws]	21.46	6.89
Relative energy consumption	100%	32%

unit, the actual processing time, and the communication time back to the device. From the time measurements in the original camera implementation, it can be estimated that transferring an image to a computer amounts to 124 μs. The same time will be required to send it back to the camera. If, for example, the detection of a certain object has to trigger a video acquisition sequence, at least 248 us latency need to be taken into account. This makes BinaryEye at least 12$\times$ faster than the camera with external image processing, even without taking the external processing time into account. In applications like mail sorting, the reduced latency would allow letters to be processed quicker as the sorting mechanism would get its directing command faster, increasing the throughput of the machine.

3.5 Conclusion

Binarized neural networks for image classification offer significant computational benefits and memory savings compared to full-precision implementations of the same network. This chapter presented a high-speed streaming camera with real-time on-board classification for edge processing, demonstrating these advantages on a real system. On-board processing of the image, as opposed to external (cloud) computing, enabled the camera to reduce the transmission data and thus achieved high frame rates in streaming mode without congesting its

Table 3.3: Comparison to prior work

Implementation	**Network**	**Platform**	**Net size**	**MNIST Benchmark**			**Power [W]**		**Perf.**
			[Mbit/ MOP]	**Accuracy**	**Inference [μs]/fps**	**Latency [μs]**	**Chip**	**Wall**	**[GOPS/s]**
This work	FC binary	XC7K325T	2.9/ 5.8	98.40%	18.49 / 10k	19.28	0.52	12.7	58
This work	FC binary	XC7K325T	2.9/ 5.8	98.40%	18.49 / 20k	19.28	0.52	13.8	116
Umuroglu et al. [15]	FC binary	ZC706	2.9/ 5.8	98.40%	- / 1561k	2.44	8.8	22.6	9085.67
Umuroglu et al. [15]	FC binary	ZC706	2.9/ 5.8	98.40%	- / 12.2k	282	0.8	7.9	70.76
Alemdar et al. [32]	FC ternary	XC7K160T	-	97.76%	- / 255'102	5.37	0.32	-	-
Alemdar et al. [32]	FC ternary	XC7K160T	-	98.33%	- / 255'102	15.6	2.76	-	-
Liang et al. [11]	FC binary	5SGSD8	10.1/20.2	98.24%	3.39 / -	-	-	26.2	5905.4

communication interface. The pure FPGA implementation made it possible to efficiently exploit the simplified computations as well as the low memory footprint of BNNs, allowing the whole network to be stored in power-efficient on-chip BRAM.

Our system achieved real-time image recognition at 20kFPS with 19.28 us latency. The implemented case study of a hand-written digit classifier demonstrated near-state-of-the-art accuracy at 0.52 W power consumption. Classifying images on-board the camera reduced the data volume to be transmitted by 980x, enabling power saving of 3× compared to full-data streaming. Freeing capacity on the communication channel additionally enabled multiple other cameras to be attached to the same network, making applications with interacting high-speed cameras possible. These results indicated that BNNs could help edge processing to be implemented in other FPGA-based systems. This is supported by various works that evaluate BNNs for more complex classifiers and report comparable accuracies as their full-precision counterparts (e.g., CIFAR-10 classifier [230, 233]), making BNN-based classifiers a promising solution for many other applications that go beyond the presented case study.

The configurability with high-level programming tools, like Vivado HLS, simplifies the deployment of FPGAs and their adaption to more complex applications. Combined with the benefits from efficient handling of BNNs, FPGAs were shown to offer simple implementations of edge processing systems.

Chapter 4

Automated Neural Network Mapping on FPGA-Based Cameras

The FPGA-based camera in Chapter 3 demonstrated low latency and yet highly configurable edge processing capabilities, advocating the advantages of processing NNs on FPGA platforms. While HLS tools simplify the hardware design process, the mapping of trained networks still requires expert knowledge for implementing the dataflow and correctly allocating data in the memory.

To further streamline the implementation process, this chapter presents an automated deployment framework for DNN acceleration at the edge on FPGA-based cameras. The tool automatically converts an arbitrary-sized and quantized trained network into an efficient streaming-processing DNN accelerator block that is instantiated within a generic adapter block in the FPGA. In contrast to prior work, the accelerator is purely built from programmable logic and thus also supports end-to-end processing on FPGAs without on-chip microprocessors. Its automatic translation from trained Caffe networks supports arbitrary layer-wise fixed-point precision for both weights and activations, an efficient XNOR implementation for fully binary layers, as well as a balancing mechanism for effective resource

allocation. To demonstrate the performance of the system we employ this tool to implement two CNN edge processing networks on an FPGA-based high-speed camera with various precision settings, showing computational throughputs of up to 337GOPS in low-latency streaming mode (no batching), running entirely on the camera.

4.1 Introduction

A wide range of deep NNs are being used for detecting and classifying objects [249] in imaging applications but also serve many other domains that go far beyond the field of computer vision. The list of different network architectures is nearly as long as the number of applications itself, each one optimized for its specific purpose, the hardware it is running on, the available power, and the required computational throughput. For efficiently deploying each specific NN, a diverse set of strategies has been followed, such that some networks rely on accurate high-precision computations while others are optimized for fast, but lower-precision, arithmetic [250]. Thus, it is crucial that the embedded processing hardware and its mapping tool are flexible and support the optimal parameter options as well as a developer-friendly deployment.

DNNs consist of billions of multiply-and-accumulate (MAC) operations that must be computed at low latency to meet the throughput requirements of the machine learning (ML) inference application. While such a workload is usually too demanding for general-purpose CPUs, the highly parallelized architectures of GPUs offer sufficient computational throughput for processing small- to medium-sized networks in real-time [251]. Thus, most of today's ML frameworks for training and inference natively support GPUs for accelerating their processing. Cloud computing solutions are employed in some compute-intensive use-cases, outsourcing the large and power-hungry processing engines to servers with virtually unlimited resources. While this provides mobile systems with access to cheap and powerful computational resources, it introduces energy-intensive (raw) data communications between the device and the cloud, adding significant latency to the processing.

However, for applications like (image-based) video self-triggering, low processing latency is crucial as it was shown in [252, 253]. These

implementations achieve low-latency video triggering through simple on-board image comparison or temporal change detection but could benefit from more sophisticated ML-based triggering algorithms. Using on-board algorithm processing capabilities at the edge (of the connected network), so-called edge processing, low latency can be achieved while avoiding power-intensive communications with the cloud. Additionally, edge processing mitigates privacy concerns that might arise for transferring (sensitive) data through a (public) network.

The general-purpose architectures of CPUs and GPUs enable the user to quickly adapt to new NN topologies, which might require novel data types, connection types, and differ in their layer dimensions. However, each specific network can be implemented more efficiently using ASICs, which can avoid unnecessary overhead, implementing only those hardware resources that are required for a specific target application. This minimizes both the memory footprint and the power consumption for processing the network, making them suitable for edge processing on low-power and low-cost devices. Based on the fast evolution of NNs over the past few years and the long development time for ASICs, we note that the SoA network types might already have significantly changed by the time the ASIC arrives on the market.

The resulting need for more flexible hardware accelerators opened the door for FPGA-based devices, providing highly configurable hardware blocks that can quickly be reconfigured to optimize the allocation of computational resources to new network architectures. The complex development flow of FPGAs using HDL can be streamlined with HLS tools like Xilinx Vivado HLS [254]. A range of domain-specific NN mapping tools have further been presented to simplify the implementation of trained NNs on FPGA-based devices, supporting features like parameter quantization and computation scheduling. However, these tools often rely on vendor-specific hardware blocks and are limited to a subset of the available FPGA chips (or series), as they require hard CPU cores integrated within the FPGA fabric [250, 255]. Other tools are restricted in their supported dimensions (e.g. 3x3 convolutions only [256]) or the supported numerical precision (e.g. binary precision only [250] or floating-point only [255]). Additionally, most of these tools rely on batching to achieve high utilization at the expense of latency.

This chapter presents NN2CAM, a flexible and automated framework for mapping NNs as highly parallelized low-latency, streaming implementations onto pure-logic FPGA-based camera platforms for edge processing. Our target platform is a FastEye [257] high-speed camera, shown in Fig. 4.1. The tool automatically extracts the network architecture and its parameters from a trained network, supporting layer-wise configurable fixed-point precisions for weights and activations with an efficient XNOR-implementation for fully binary layers. Each network is implemented as a streaming architecture, reducing the memory needs and enabling parallel computations on all layers simultaneously. A resource-balancing technique ensures that the available processing hardware resources are effectively distributed among the layers to maximize the throughput. In Section 4.2 we summarize quantized networks and their processing approaches. Section 4.3 provides an overview of existing SoA frameworks for mapping networks onto FPGAs and in Section 4.4 we present our proposed framework. The implemented streaming architecture is described in section 4.5, followed by experimental results demonstrating the performance on the high-speed camera platform in Section 4.6.

4.2 Background

Neural networks exist in many forms, differing in their dimensions, layer types, and data precisions, which can be optimized for each specific application. To deploy a NN onto a hardware platform, the mapping tool must understand these abstract parameters and be able to create an (efficient) mapping between the network description and the available hardware resources. In the following subsections, we summarize the relevant terminology and principles for interpreting and mapping NN models.

4.2.1 Quantized Neural Networks

As shown in the previous Chapter, BNNs provide beneficial properties for efficient hardware implementation, reducing the memory and the logic resources compared to full-precision arithmetic. However, the precision requirements of activation and parameter values can differ

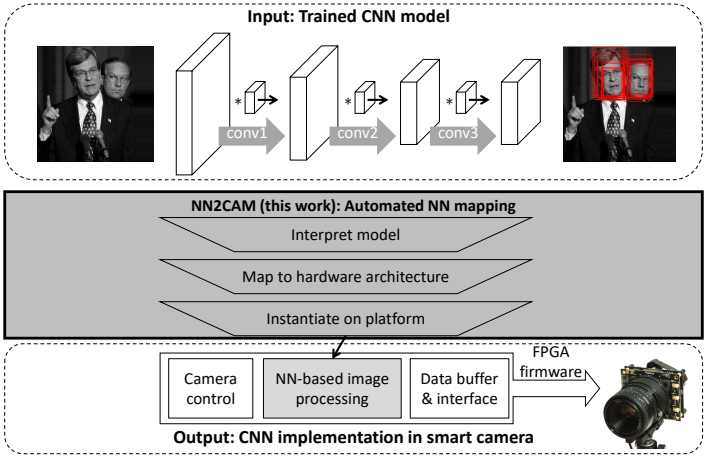

Figure 4.1: Overview of the NN2CAM framework, mapping a trained CNN model onto a smart FPGA-based camera.

across various applications and even across different layers within a single network. While most of the powerful general-purpose processing devices operate with 32 bit floating-point numbers to accurately represent a wide range of values, power- and size-restricted platforms are often limited to smaller and fixed-point precisions ranging from 16 bit down to 1 bit values. The BNNs in Chapter 3 show up to 32× lower memory requirements, while the arithmetic operations require much smaller hardware resources and the related power per operation can be significantly improved [102]. Section 2.8.4 further elaborates on the hardware effects of various quantization levels, reporting substantial benefits also for intermediate sub-8bit quantization levels, which motivates supporting these precisions in our work.

Quantizing NNs to smaller precisions requires careful mapping as quantization errors can quickly propagate through deep networks, accumulating to significant sources of accuracy deteriorations. Research in this field has shown that aggressive quantization can be achieved with negligible impact on accuracy if quantization-aware training is

employed [258, 259]. Such resiliency to quantization errors has been shown for low bit precisions ranging from a few bits [193] down to binary representations [194].

4.2.2 Streaming versus Layer-Wise Processing

Due to the layered structure of NNs, layers can be computed sequentially, or layer-wise, always completing one layer before starting the next one, as shown in Fig. 4.2 a). Once a layer is finished, the memory of its input layer is no longer needed and can be overwritten (freeing that memory space). This has the advantage that only a maximum of two subsequent activation layers must be kept in memory instead of all of them [28], reducing the memory needs. However, computation parallelism is limited to a single layer. In contrast, streaming processing, also called depth-first execution [144], computes a small region of each layer in parallel, only buffering small fractions of the layers in memory. Parallelization of computations can thus be extended to many layers and the required activation memory can largely be reduced as only a small portion of each layer must be stored at any point in time. This principle is depicted in Fig. 4.2 b), showing the entire layer (activations) with dashed lines and the part that is stored in memory with solid lines. Each layer continuously buffers incoming activations computed by its preceding layer and processes them as soon as a complete kernel-sized window is available. Data that is no longer needed in following sliding window positions can be overwritten, making the buffer a first-in first-out (FIFO) block. The results from each convolution are streamed out to the next layer, operating in the same FIFO manner.

4.3 Related Work

Existing frameworks allow to map specific types of neural networks onto FPGA platforms, configuring optimized hardware accelerators for the networks. These tools are specialized for a certain field of application and thus only support a specific set of features. FPGA-based NN accelerators have been surveyed in various works, listing

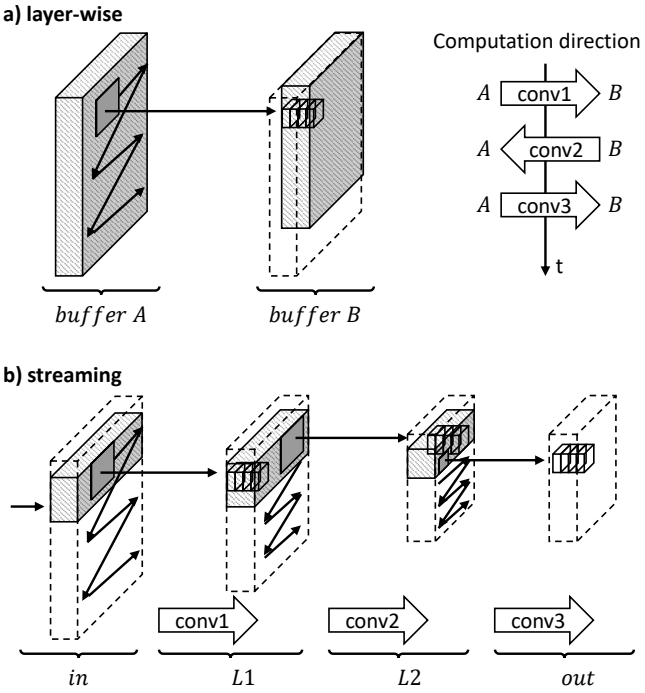

Figure 4.2: Visualization of layer-wise CNN processing (a) and comparison with streaming implementation (b). The grey box marks the window position that is currently being computed (convolved with the kernel) while the striped box indicates data that is buffered in memory.

their optimization strategies and implementations [54,260,261]. Table 4.1 provides a summary of current SoA frameworks.

Caffeine [262] is one of the first frameworks that presented accelerating the inference of networks using FPGAs, executing computations offloaded from a connected host PC. The software library compiles layer settings and parameters that are then synthesized using the Xilinx SDAccel tool and loaded onto the FPGA, computing the

assigned network portions layer-by-layer. A similar implementation, but with a pre-synthesized accelerator that can be configured during run-time, is shown in [262], implementing 3x3 convolution layers using the Winograd algorithm. DLAU [255] is another early NN accelerator architecture for FPGAs, implementing matrix multiplication, addition, and activation completely in logic with three pipelined processing blocks for tile-wise processing. An on-board CPU configures the processing block with parameters and feeds the tiled matrix data. Their 32 MAC units demonstrate a speed-up of 36.1× compared to a 2.3 GHz Intel Core2 processor for computing an FC network at 200MHz.

The introduction of BNNs in 2016 [194], enabled MAC computations to be implemented with simple logic, as demonstrated in the FINN framework [250]. Combined with a streaming architecture, their experiments reported unprecedented throughputs thanks to the highly parallelizable logic implementation of binary MAC operations and the small memory footprint of their weights, allowing to map the entire network in on-chip memory. Additionally, FINN provides a standalone processing platform as the on-chip CPU core of the utilized Zynq FPGA is used for control purposes. The initial work consists of three pre-defined networks and a simple compiler that generates HLS code for synthesis using the Xilinx Vivado HLS tool. Recently, significant extensions to the training and quantization-flow of the compiler [263] as well as extensions to other network types were published. A similar HLS approach is taken in PipeCNN [264], using Altera's (now Intel) OpenCL flow for high-level synthesis to implement their layer-wise accelerator. Angel-Eye [265] is another framework targeting Zynq-based implementations of CNNs. Their framework consists of a Caffe-compatible layer-wise quantizer to generate fixed-point weights, focusing on 8 bit representations, and a compiler that maps the entire network as a standalone engine onto the CPU and the FPGA logic of a Zynq SoC. The CPU controls the FPGA-based CNN accelerator, feeding it with input data as well as with instructions. Another OpenCL-based framework is presented in [266] and their follow-up work [267], reporting a CNN mapping flow for 16 bit fixed point implementations on FPGAs using a latency-driven optimization strategy.

The automated CNN mapping toolchain MALOC [268] implements the entire network on the chip as a pipelined implementation. To enable larger networks, for which intermediate layers cannot be buffered simultaneously, it employs a tiling mechanism that determines which parts of each intermediate layer shall be buffered. Another framework with an automated mapping toolchain is presented in [165], exploiting the efficient implementation of convolution computations using FFT and Winograd transformations. These two implementations report performance for batch processing to keep the utilization in their computation blocks high. Because this is not possible for single-frame operation, where inter-layer dependencies can stall subsequent layer processing, the performance is expected to significantly decrease for single-frame operation, as it was shown in the performance of [250].

Xilinx' Vitis AI tool provides a deep learning processing unit (DPU) [269], which is an accelerator engine for the Xilinx Zynq platforms, allowing to choose from 8 pre-configured (pre-placed-and-routed) architectures with varying degrees of parallelism. It contains a compiler that maps the network to an efficient instruction set and performs data reuse optimization as well as instruction scheduling. Additionally, it provides an optimizer tool that enables model compression. Vitis AI is part of Xilinx' unified development environment that combines high-level synthesis with multi-processor support and a set of APIs.

NEURAghe [270] is a hardware/software framework for Zynq platforms, exploiting a tight interaction between network-optimized software, running on the hard CPU, and a 16 bit fixed point accelerator implemented on the programmable logic. Another recent work [271] implements a dedicated CNN accelerator for YOLOv2 object detectors using Intel's OpenCL FPGA framework. This accelerator achieves unprecedented throughputs, but only supports 8 bit implementations of YOLO and no other networks. Tridgell et al. [272] follow a network unrolling approach for ternary networks and additionally exploit sparsity in convolutional layers (using the knowledge of zero weights in the kernels), achieving up to 2.5TOPS. Due to the unrolling of the network, a high degree of parallelization is achieved but also extreme numbers of FPGA resources are required, which is why it was implemented on a cloud AWS platform and only tested on small input images. A similar unrolling strategy was used in LogicNets [273],

encoding the entire network using look-up tables (LUTs). Using a new network co-design strategy they can keep the resource utilization low and thus achieve throughputs of more than 8TOPS on very small low-precision networks. This is unfortunately limited to specialized applications with very small network sizes like they are used in data communication analytics.

Based on the performance and the flexibility of the supported network topologies, [165, 250, 269] can be considered the state-of-the-art quantized neural network mapping frameworks for FPGA-based hardware acceleration. While some of the other frameworks report up to 3× higher (best in class) throughput results, they are limited in their flexibility in terms of supported quantization levels and supported networks.

Despite the variety of existing frameworks, none of them support the target FastEye camera edge processing use case, requiring flexible and automated NN mapping onto pure-logic FPGA platforms for standalone end-to-end processing, supporting arbitrary-sized CNNs to enable a wide range of applications. While [262, 262] are designed for Caffe-acceleration on PCs and [262] only supports 3x3 convolutions, all other Xilinx-FPGA-compatible frameworks are limited to Zynq platforms with CPU cores, do not feature standalone processing and only cover limited ranges of precision options. Our work supports the required features for the FastEye camera while achieving 337.8GOPS, which is comparable to the highest reported performance among the state-of-the-art frameworks.

4.4 Neural Network Mapping Framework

Deploying a trained NN onto an FPGA platform for on-board inference requires system developers to cross multiple layers of abstraction, from a high-level network description down to a low-level hardware design. Machine learning training tools, like Caffe [9], encode the trained network in a set of files containing the architecture description and the learned parameters. These files form the basis for translating the NN into a representation that can be synthesized for the target FPGA platform. This section describes our NN2CAM framework that automatically maps a trained Caffe-formatted image analysis CNN on

Table 4.1: Frameworks for mapping neural networks on FPGA platforms

Framework (year)	**Supported networks**	**Supported precisions**	**Control (Execution)**	**Supported HW platforms**	**Performance (Platform)**
Caffeine (2016) [262]	Same as Caffe	Float, fixed-point	PC via PCIe (layer-wise)	Xilinx FPGA	1.46 GOPS peak @ 8bit fixed point (Xilinx KU060 FPGA)
DLAU (2017) [255]	FC shown	Float	CPU core (layer-wise)	Xilinx Zynq only	6.4 GOPS peak @ float (Xilinx Zynq 7Z020 SoC)
Caffeinated FPGAs (2016) [256]	Only 3×3 HW conv.	Float	PC via PCIe (layer-wise)	Xilinx FPGA	50 GFLOPS on 3x3 convolution only @ 32bit float (Xilinx 7VX690T FPGA)
FINN (2018) [250]	FC, CNN	Fixed-point, binary	CPU core (streaming)	Xilinx Zynq only	397.5GOPS[1] single frame, 2'465 GOPS batch processing, @ binary CNN (Xilinx Zynq ZC706 FPGA SoC)
Angel-Eye (2017) [265]	FC, CNN	Fixed-point	CPU core (layer-wise)	Xilinx Zynq only	137 GOPS, 14.2GOPS/W @ 8bit fixed point (Xilinx Zynq 7Z045)
PipeCNN (2017) [264]	FC, CNN	Fixed-point	PC via PCIe (layer-wise)	Intel FPGA SoC	179 GOPS @ 8bit fixed point (Intel Arria-10 GX1150)
fpgaConvNet (2017) [266]	FC, CNN	Fixed-point (16b)	CPU core (streaming)	Xilinx Zynq only	162 GOPS @ 16bit fixed point (Xilinx Zynq 7Z045)
Dedicated YOLOv2 acc. (2020) [271]	YOLOv2	Fixed-point (8b)	CPU core (streaming)	Intel FPGA SoC	566 GOPS @ 8bit fixed point (Intel Arria-10 GX1150)
MALOC (2018) [268]	FC, CNN	Fixed-point (16b)	CPU core (streaming)	Xilinx Zynq, Virtex-7	80.35 GOPS2 @ 16bit fixed point (Xilinx Zynq 7Z020 FPGA SoC)
Eval. Fast Algo. for CNNs on FPGAs (2020) [165]	FC, CNN, others	Fixed-point (16b)	CPU core (layer-wise)	Xilinx Zynq only	201.1 GOPS[2] @ 16bit fiexed point (Xilinx Zynq ZC706 FPGA SoC)
Vitis DPU (2020) [269]	FC, CNN, others	Fixed-point	CPU core (layer-wise)	Xilinx Zynq only	230 GOPS peak @ 8bit fixed point (Xilinx Zynq 7Z020 FPGA SoC)
NEURAghe (2018) [270]	FC, CNN	Fixed-point (16b)	CPU core (layer-wise)	Xilinx Zynq only	169 GOPS @ 16bit fixed point (Xilinx Zynq 7Z045)
Unrolling TNN (2018) [272]	FC, CNN	Ternary weight, 16b activations	PC via PCIe (streaming)	Amazon AWS F1	2500 GOPS @ ternary weight (Amazon AWS F1)
NN2CAM (this work)	**FC, CNN**	**Fixed-point & binary**	**Pure logic (streaming)**	**Xilinx FPGA**	**337.8 GOPS average @ binary (Xilinx XC7K325)**

[1]Computed from their CNN size and the latency, [2]Performance for batch processing only (for a fair comparison, we show their medium-sized FPGA only)

an FPGA-based camera using a streaming processing architecture as shown in Fig. 4.3. Starting from the trained network, the architecture gets extracted using Python scripts, determining all dimensions and layer settings as well as extracting and annotating the trained parameters. After mapping the layer operations to HW-implementable functions (e.g. convolution operation) and allocating the required HW resources to efficiently compute the network, the resulting (high-level) representation gets compiled into a hardware description. From there, the usual FPGA synthesis tools are used to synthesize the system to a bitstream. Our target system is a FastEye high-speed camera, in which the CNN-based image analysis block is instantiated by the proposed framework, automatically mapping arbitrary CNN to it. However, the mapping flow is generic and can be used for other applications where similar DNN-based image processing on the edge is required.

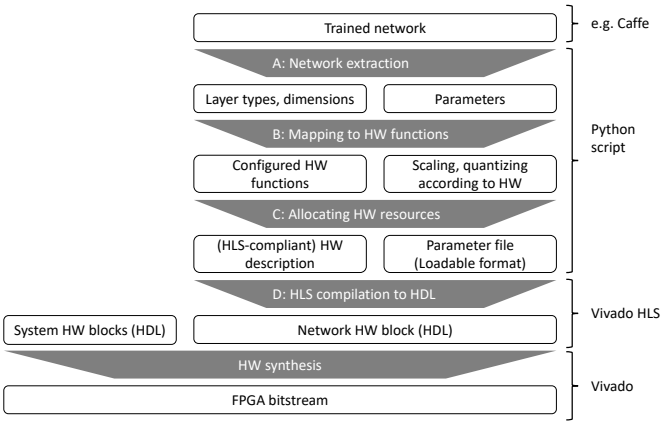

Figure 4.3: Framework tool-flow from trained network description to bitstream.

4.4.1 Network Extraction

Caffe represents its trained models with 1) a textual prototxt-format file containing the architecture skeleton that describes each layer with its type, dimensions, and connectivity as well as 2) a binary caffemodel-format file that stores all learned parameters like weights and biases. To convert the network into a framework-interpretable format, the architecture description is first parsed, extracting all layer dimensions, which then allows arranging the parameters into a mathematical matrix representation. In this form, the network is essentially a series of 2-dimensional convolution operations on 3-dimensional matrices with known convolution kernels.

4.4.2 Mapping to Hardware Functions

In the next step, the individual layers are mapped to the available prototype hardware functions. Our tool currently supports the following layer types: 1) FC layers, 2) CNN layers and 3) average-pooling layers which are all implemented as high-level C++ functions that are compatible with Xilinx Vivado HLS. All network components like layer types, processing elements, and data handling mechanisms are implemented in this generic C++ format within a set of library files, enabling simple high-level extensions for future layer types. The compilation script utilizes these functions and parametrizes them during the instantiation. Most of these library functions are based on the FINN framework [249], which targets implementing BNNs on Xilinx Zynq FPGAs. As FINN focuses on BNNs, it does not directly support arbitrary fixed-point precision networks nor implement input activation padding or ReLU activation functions, such that some of the libraries had to be extended. We list the main extensions to the original FINN library:

- Extended with arbitrary fixed-point precision.
- Extended sliding window function with side padding and fixed-point precision support.
- Added a function for ReLU activation support.

- Extended MAC implementation with fixed-point precision support.
- Mapping tool: The FINN framework did not feature a mapping tool but only supported a set of pre-configured networks.

Considering the processing of a CNN layer, input activations from a sliding window position need to be multiplied and accumulated with a learned weight kernel. The MAC operations of each window position perform a dot product between the two matrices, which can be broken down into a sequence of MAC operations. These matrix operations are repeated for each window position on the input feature map. A data path implementing these functions can thus be split into two stages: 1) the sliding window generation that buffers the input data to provide the correct input activations (window) and 2) the MAC processing that performs the actual computation of the data. As explained earlier, FC layers can be interpreted as a sub-set of CNN layers where the kernel has the same size as the input layer, making the sliding window operation needless. Average pooling layers are also similar to CNN layers but work in a channel-wise manner with a homogeneous weight kernel that averages over the input activations of each input channel separately. To avoid multiplications, the input activations can be summed up and the sum divided by the number of kernel elements per channel. This framework assumes that the parameters of the trained network have already been quantized during training and thus directly converts extracted parameters to the fixed-point precision that is specified in the model.

4.4.3 Allocating and Balancing Resources

Mapping the network to hardware-representable functions generates a functionally complete representation. However, due to the arbitrary size of each layer, the layer processing time can vary significantly across the layers, creating an imbalanced data flow. This negatively impacts the temporal processing element utilization in some layers, leading to a low overall throughput because each layer in the streaming architecture is waiting for data from its previous layer, stalling the following ones if insufficient input activations are supplied. Thus, the processing throughput needs to be balanced across all layers by

allocating MAC processing units accordingly. Fig. 4.4 visualizes this problem, comparing each layer's processing workload in terms of MAC operations per output to achieve a continuous data flow in all layers. The assumed 3-layer network shows higher computational loads in the first layers, decreasing towards the output.

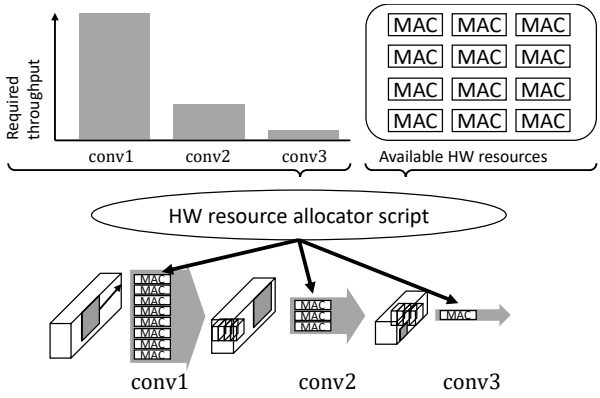

Figure 4.4: Computational hardware resource allocation to ensure a continuous data processing stream across all layers.

The streaming architecture employed in this framework supports two kinds of processing parallelism within each layer, which logically arrange the assigned MAC units:

1. Output channel parallelism: Multiple output channels are being processed at the same time, each one in a separate PE. Input activations can thus be reused across all PEs and each PE independently accumulates the results of its multiplications. The number of PEs is therefore limited by the number of output channels in the layer. Additionally, each PE is independently accessing parameters from the memory, requiring a dedicated BRAM-macro instance per PE.

2. Input parallelism: Multiple input activations in each PE are being processed in parallel. We refer to this as single instruction, multiple data (SIMD) parallelism, as it was called in the FINN

framework [250]. To simplify the implementation, the number of parallel input computations ($\#SIMD$) is restricted to integer divisors of the number of elements in the kernel.

Because the available parallelism and the throughput needs are highly network-dependent, it is necessary to balance the computational resources for each network individually, taking both the network and the available resources into account. By distributing the available hardware resources of the FPGA to the layers according to the balanced distribution of required throughput, the throughput can be maximized. A balance on a specific layer is achieved when the time for computing all output channels of a certain x/y position equals the time for computing and buffering sufficient input activations to execute the next sliding window position as depicted in Fig. 4.6. We compute this balance by quantifying the average rate of input activations, shown in Equation 4.1, required to compute the average rate of output activations, shown in Equation 4.2. Computing the ratio R_i of these rates for each layer i in Equation 4.3, allows to determine the balanced distribution of MAC units across the layers. This is achieved by ensuring that the input activation rate of each layer matches the output activation rate of the previous one. Equation 4.4 formulates the resulting MAC distribution across layers D_{MAC} by computing the normalized product of all R_i for each of the layers. From Equations 4.2 and 4.4 follows, that the number of MAC units for a layer i can be found by multiplying the available number of processing elements (e.g. DSPs) with D_{MAC} (i). Binary MAC units do not consume any DSP elements, making the search for the available number of processing elements an iterative process that depends on available logic elements.

$$in_{rate} = \frac{\#in_{(s_y\ rows)}}{t_{(1\ output\ row)}} = \frac{s_y \cdot X_{in} \cdot C_{in}}{\frac{X_{in}}{s_x}\left(\frac{C_{out}}{\#PE} \cdot \frac{t_{(1\ kernel)}}{\#SIMD}\right)} \tag{4.1}$$

$$out_{rate} = \frac{\#out_{(1\ output\ pos.)}}{t_{(1\ output\ pos.)}} = \frac{\#PE}{\left(\frac{t_{(1\ kernel)}}{\#SIMD}\right)} = \frac{\#PE \cdot \#SIMD}{t_{(1\ kernel)}} \tag{4.2}$$

$$R_i = \frac{out_{rate}}{in_{rate}} = \frac{C_{out}}{s_y \cdot s_x \cdot C_{in}} \tag{4.3}$$

$$D_{MAC}(i) = \frac{\prod_{k \leq i} R_k}{\sum_i \left(\prod_{k \leq i} R_k \right)} \tag{4.4}$$

Our framework takes the number of available DSP resources as well as the number of BRAM instances into account. Resources already used by the hosting application (the image acquisition data path in our camera application), are deducted prior to the distribution.

4.4.4 HLS Compilation to Hardware Representation

The last step of the mapping process consists of translating the internal high-level representation into a hardware description. This framework first compiles the mapped network into an intermediate C++-based high-level representation that is compatible with Xilinx Vivado HLS. Thus, it can be automatically synthesized into HDL using Vivado HLS, bypassing the otherwise necessary HDL know-how. The synthesized block features two standard advanced extensible interfaces (AXI4), allowing to automatically map the block using an adapter that contains all utility functions to load the input image and the parameters through the AXI interfaces. As shown in Fig. 4.7, this includes an image preparation block, a parameter loading, and control block, as well as a result buffering block. These utilities simplify the instantiation of the accelerator in any FPGA design, only requiring two interfaces: 1) an image line input and 2) a result (output layer) interface.

4.5 Streaming Processing Architecture

This chapter describes the streaming processing architecture implemented by the proposed framework. As shown in Section 4.2.2, streaming architectures impose processing dependencies between activations of neighboring layers. Each activation in a specific x/y-position of a layer depends on activations from a limited (kernel-sized) input region on its preceding layer as shown in Fig. 4.5. Thus, the output activation computation can only be completed once all input activations

are available, implying that the following layer's computations will be stalled if the input activations are not computed fast enough. Fig. 4.6 a) visualizes this problem on an example CNN, showing that the computations in the first convolution layer are stalling the processing of the second layer. To efficiently process MAC operations in parallel across all layers, a balanced distribution of MAC units among the layers and pipelining between the layer computations is used, allowing each layer to compute new activations while the following layer is still busy processing the previous ones.

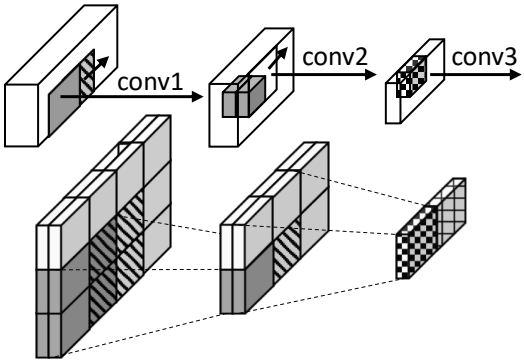

Figure 4.5: Visualization of connected regions across convolutional layers of an example CNN. All convolutions have a stride of 1, indicated with the two neighboring window positions (gray, striped).

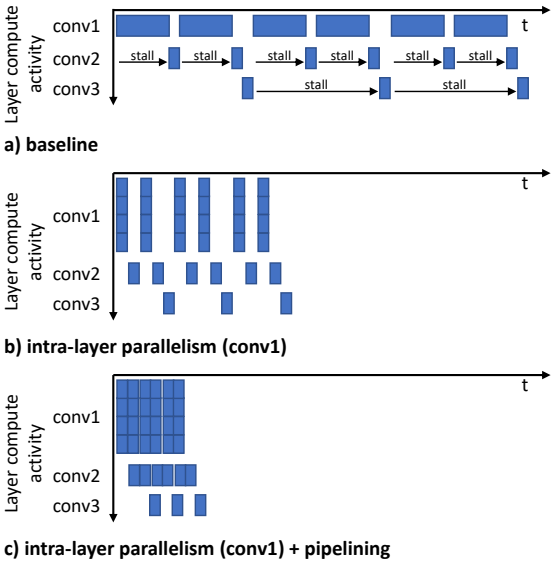

Figure 4.6: Layer-wise computation activities for different parallelism options, indicating stalled computations.

This is achieved by using a systolic streaming architecture, instantiating separated processing blocks for every network layer, allowing layer-pipelining and layer-individual MAC parallelization. Through careful balancing of the computation resources, MAC operations can be efficiently parallelized across all layers, allowing each layer to independently compute new activations, while the following layer is still busy processing the previous ones. Summarized, the system features two types of computational parallelism:

1. Intra-layer parallelism: Each layer processing block can parallelize its computations independently as discussed in Section 4.4.3. This allows the computational power to be optimized for the layer dimensions and the input requirements of the following layer, as shown for the first layer in Fig. 4.6 b).

2. Inter-layer parallelism: Layer-pipelining allows each layer to optimize its compute utilization by virtually removing the dependency between layers through time-shifted processing, as shown in Fig. 4.6 c).

Fig. 4.8 shows the block diagram of the accelerator block in which the NN is instantiated by the framework. The input image from the (camera) application is stored in a buffer that is accessible by the accelerator. Two standard AXI4 ports communicate with the accelerator, the first one accesses the data while the second one is used for controlling and parameter loading, as shown in the system diagram in Fig. 4.7.

Image data is directly accessed from the buffer and converted into a data stream, passing through the network layers. Each layer contains a FIFO that buffers data until they are requested by the subsequent layer for processing and locally stores weights and biases. Parameters are loaded through the control interface during the initialization of the accelerator, while results from the last layer are streamed out through the data interface and buffered in an accelerator-external result buffer. Once the network inference is complete, the application can access the results from this buffer and trigger the next execution. A more detailed look into the block reveals the NN-specific mathematical operations, namely the MAC units that multiply-and-accumulate inputs with weights, and a sliding window input generator that generates the correct input activation access pattern for 2D convolution. The control block contains all necessary utilities and state machines for controlling parameter loading and system state control. Intra-layer parallelism is provided through the instantiation of multiple PEs, allowing to process multiple output channels in parallel, reusing the same input data. Each of these PEs can be configured to additionally process multiple inputs of its kernel computation in a SIMD fashion, fetching and processing multiple activations in parallel.

4.5.1 Sliding Window Generator

The sliding window generator implements the activation access pattern for computing the 2-dimensional convolution in CNNs. It outputs the activations of the kernel-sized input window, which is slid over

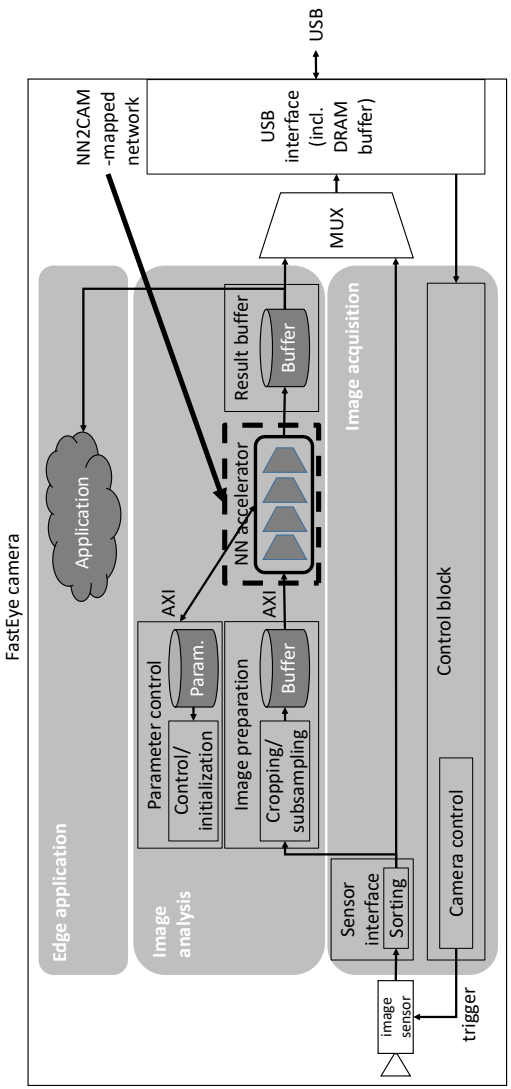

Figure 4.7: Block diagram of the implemented FastEye camera system with automatically mapped NN accelerator block.

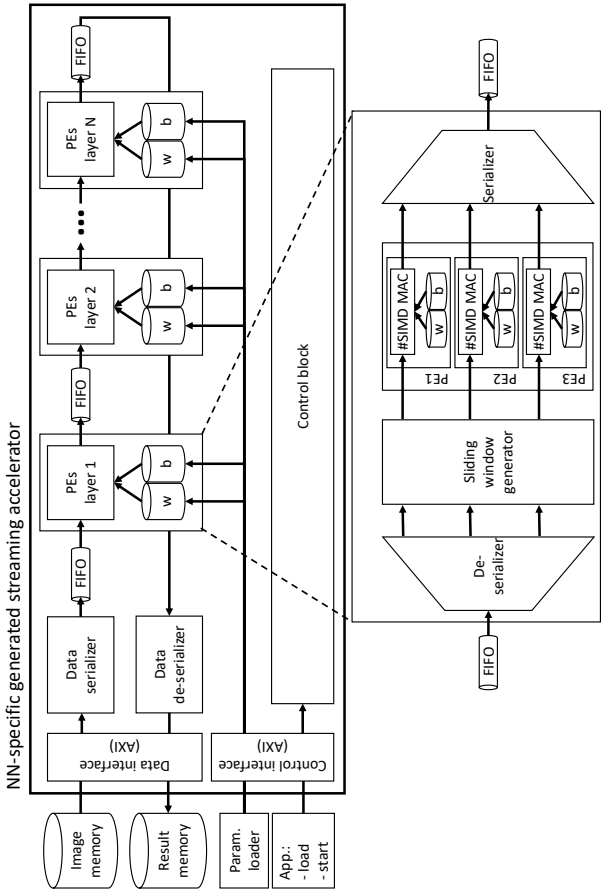

Figure 4.8: Data flow of an implemented streaming architecture with data streams connecting the layer processing blocks. Below, a more detailed view of a layer processing block is shown, consisting of a stream interpreter that feeds an optional sliding window generator, followed by MAC processing units (with local weight and bias memories) and a data serializer that generates the output stream.

the input feature map as illustrated in Fig. 4.2 a). For each window position, the sequence of activations is synchronized with the sequence of kernel weights in the parameter memory, such that the processing block receives the corresponding activations and weights for performing the MAC operations. As the input window is only moved by a few input elements in x or y direction (defined by the stride parameters s_x and s_y), most of the window data is reused across neighboring windows. To avoid transferring the same input data multiple times between subsequent layers, a buffer is instantiated in the generator block, storing all input activations needed for computing 2 output lines (kernel height + vertical stride). This allows generating activations for the subsequent computation of the first output line, while the remaining buffer is continuously being filled, such that the next output row can be started directly after the current row is completed. Figure 4.9 visualizes this concept.

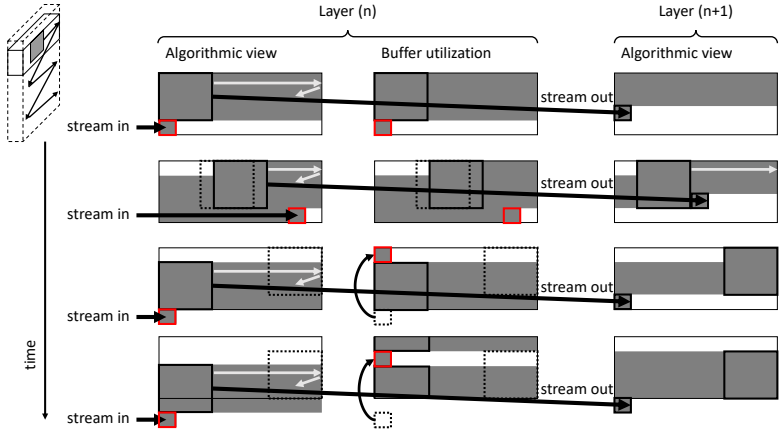

Figure 4.9: Temporal evolution of activation buffer content across two subsequent layers.

4.5.2 Processing Element

The sliding window generator ensures an identical data ordering in the input activation stream as in the kernel weight sequence within the memory. This simplifies the task of the PEs to plain multiplication and accumulation of the incoming data in the order it arrives. Solely the bias addition and the activation at the end of each kernel computation is controlled within the PE, using a simple MAC counter.

The data flow of a layer is visualized in Fig. 4.10, showing the data handling from the input activation stream, passing through the input buffer of the sliding window data generation, the processing elements, and finally ending up in the output stream that continues to the next layer. All data streams are ordered channel-first, streaming all input channels of an x/y position, before moving to the neighboring x/y position in the x-direction. All PEs receive the same input activation data but compute them for a different output channel (output channel parallelism). In every cycle, each PE computes $\#SIMD$ input channels, such that computing a single output element requires $KERNEL_{SIZE}/\#SIMD$ cycles (intra-kernel parallelism). If a layer features more output channels than allocated PEs, all output channels of an x/y-position are completed first, $\#PE$ at a time, before moving to the next x/y-position on the output layer. This creates the correct output data ordering again, allowing the next layer to be processed in the same manner. Each PE's parameter memory is mapped to match this activations sequence with every logical memory address containing $\#SIMD$ weights, making the weights of a single kernel to appear in a sequence, followed by the next kernel to be processed by the PE (C_{out} modulo $\#PE$).

4.5.3 Precision

To exploit the reduced hardware requirements and power demands of quantized networks, this work supports arbitrary activation and weight precisions down to fully binary implementations for BNNs. BNNs simplify the dominating multiply-accumulate (MAC) operations to plain XNOR logic with appended bit-counting (popcount) [250]. This is highly advantageous for FPGAs as the memory needs

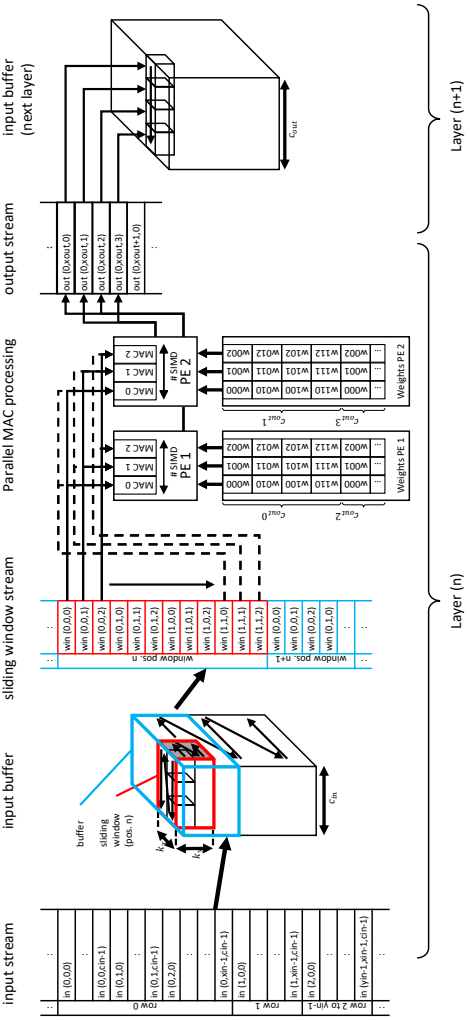

Figure 4.10: Visualization of data-stream order and parallelism for computing an example 2x2 convolution on a 3-channel input/4-channel output layer. Biasing and the output activation functions are omitted for simplicity.

are reduced and XNOR multipliers with popcount adders can be efficiently implemented in logic, minimizing the use of limited hardware resources like DSP blocks. Non-binary but low precision fixed-point arithmetic still requires DSP blocks but can massively reduce the memory requirements for FIFO buffers and parameter storage as well as related glue logic.

The framework allows precision settings to be applied for each layer separately, enabling networks with low precision feature extraction in their first layers to be implemented along with higher precision classifiers in their last layers. Support for higher precision computations is often necessary for the first and last layers to achieve sufficient accuracy in highly quantized networks [274]. Fixed-point precision computations are synthesized into DSP-based computation blocks, while fully binary computations are automatically mapped to resource-saving XNOR implementations. Whenever a value is converted into a fixed-point representation of different precision, e.g. from a higher precision MAC result or from a binary (low precision) input into a fixed-point activation, the fractional data are directly quantized through truncation (with a saturation mechanism to ensure that the minimum/maximum representable value is not exceeded). This fixed-point data type is provided by Vivado HLS (type *ap_fixed*).

4.5.4 Portability

The presented mapping framework can be used on a wide variety of FPGA models, including the Kintex-7 FPGA on our target platform, as family-specific resources like hard CPU cores are avoided. Most other frameworks require such cores for control purposes, limiting the FPGA selection to Zynq-based platforms (for Xilinx-compatible tools).

Due to the use of vendor-specific HLS libraries, and thus also intellectual property (IP) blocks, HLS-compiled blocks can only be used across FPGAs of the same vendor. Other HLS tools have similar design flows and could possibly be used in the same fashion as presented here (e.g. Intel high-level synthesis compiler [275]).

4.6 Experiments and Results

We evaluate the performance of our approach by employing the proposed framework for implementing character detection and face detection networks on our target platform, an FPGA-based FastEye high-speed camera [257]. Table 4.2 reports the FPGA resource utilization as well as the performance and the power consumption of our experiments.

The FastEye platform features a 1 megapixel high-speed image sensor that is directly connected to a Xilinx XC7K325 FPGA. Its wide parallel interface to the sensor enables fast data extraction and pixel sorting, preparing image data for transmission through a USB interface. Fig. 4.7 shows the connections to the sensor and the USB controller. A control block configures the image sensor and forwards commands sent via USB to the other blocks. In the original camera implementation, the acquired and sorted image data is directly sent through the USB interface, allowing a host computer to access them.

The following sections describe the use-cases implemented with the presented NN2CAM framework and report the FastEye FPGA resource utilization as well as the measured power consumption. The accelerator runs at 100MHz and reads the buffered image, cropped to 28x28 - 640x640 pixels, from a block random-access memory (RAM) memory (128 BRAM blocks).

4.6.1 640x640 Pixel Optical Character Detection and Recognition

Many industrial quality control applications require number and text recognition to be performed to identify objects using various kinds of labels. The position of these labels is usually variable, making it necessary to analyze a larger field of view (e.g. 640x640 pixels) at the frame rate required to keep up with the speed of conveyor belts and other movements induced during manufacturing. More detailed analyses (e.g. high accuracy network analysis with higher precision arithmetic) can then be performed on identified regions of interest. Alternative systems with cloud- or GPU-accelerated processing require costly high-end infrastructure like high-speed video communications and network connections.

Optical character recognition (OCR) based on the MNIST dataset, containing small 28x28 pixel images of hand-written digits, is often used as an example to show a basic functionality of a NN. However, this dataset is not very useful for real-world applications, such that we augment the dataset to account for illumination change and resilience to background noise. Using this augmented MNIST-based OCR (M-OCR) dataset, we train two 5-layer CNNs, similar to LeNet-5, for classifying digits within the acquired images as shown in Fig. 4.11. The network output dimension is 1x1x11-78x78x11, for input dimensions 28x28-640x640, respectively, with each output channel representing one of the 10 possible numbers or the undefined class. The CNN is quantized to either 16 bits or to fully binary precision, while the output layer is always 16 bit wide. Using the binary implementation, the FPGA resource utilization is substantially reduced compared to the 16 bit implementation of the same network, allowing to implement a higher degree of parallelism, achieving 17-76× higher throughputs. It must be noted that the large buffers for the input image and the results already amount to 128 BRAM blocks, for the largest resolution. This limits the maximum implementable parallelism in the 16 bit implementation (higher parallelism requires more BRAM blocks as each PE requires a separate BRAM), limiting the throughput and thus keeping the latency higher. Fig. 4.12 illustrates this in a comparison between throughput and image size. The throughput of binary implementations increases with the image size as the latency of small images is dominated by the filling of computing pipelines, leading to a low compute utilization at the beginning. In contrast, the throughput of 16 bit implementations rapidly drops to a low level for image sizes above 28x28 as parallelism becomes limited by the lack of available BRAM resources (consumed by the increasing FIFO buffers). Reducing input and result buffers by directly feeding the image data to the accelerator (without buffering it) could reduce this limitation in the future.

4.6.2 640x640 Pixel Face Detection

To demonstrate that also more complex tasks, like multi-scale face detection on 640x640 pixel images, can be implemented using this

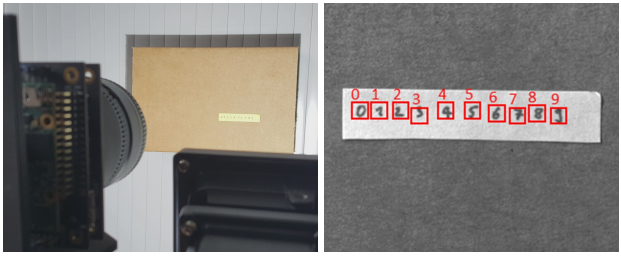

Figure 4.11: 640x640 pixel M-OCR edge processing on FastEye high-speed camera.

system, we trained and mapped a 9-layer CNN, achieving 92.3% accuracy on the FDDB benchmark, onto the camera. The network is quantized to 16 bit activations and features binary weights in its first 7 layers (the last 2 layers have 16 bit weights). Even though its input size is equal to the large OCR network, the combination of binary weight precision and a coarser detection granularity reduces the required BRAM resources and enables frame rates of more than 40FPS.

4.6.3 Result Discussion

Even though our framework implements the complete end-to-end system, it achieves comparable or higher throughput performance in its binary use cases compared to the non-standalone prior work in Table 4.1. The binary implementation of the first use case achieves on-par throughput compared to FINN's fully-binary CNN accelerator [250] for batch size 1. Their experimental setup using stored test images additionally allows frames to be processed in batch mode, which enables 6× higher throughput due to better resource utilization. In our real setup, where camera images must be processed frame-by-frame, this operating mode is not possible. The same reduction in throughput for single frame operation is expected for [165, 268], while they additionally only support 16 bit fixed-point arithmetic. However, these two frameworks achieve higher throughput for 16 bit

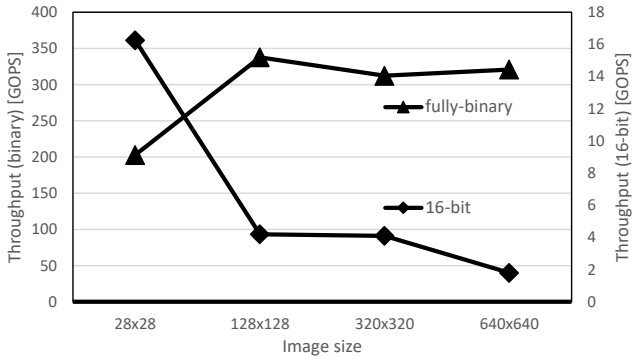

Figure 4.12: Throughput for different image sizes and arithmetic precisions.

implementations. Compared to the 8 bit precision implementations [264, 265, 269], NN2CAM achieves 1.5-2.5× higher throughput for binary implementations but up to 14× lower throughput for 16 bit implementations. However, [264] only supports a set of predefined networks, making it less flexible. While [262] achieves 3× higher throughput than our 16 bit implementations, it is limited to 3x3 convolutions and does not support standalone operation. On the lower end of the SoA performance comparison, [255, 262] report peak performances in the range of our lowest average performance but more than 2 orders of magnitude lower than the fully-binary use cases presented here.

4.7 Extensions and Limitations

The presented framework supports CNN and FC networks with arbitrary precision and the standard ReLU activation. Due to the HLS-based implementation, the library can be extended using high-level C++ functions, which allow adding new activation functions or layer types without having to write HDL code. The framework is mainly constrained by the available resources of the target FPGA

Table 4.2: Experimental results

Implementation	Network				Comp. performance		FPGA utilization				Sys. Power [W]
	Architect.	Size (MOP)	Precision	Accuracy	Latency [ms]	Throughput [GOPS]	LUT	FF	DSP	BRAM	
Camera only	-	-		-	-	-	28.6k (14%)	82.4k (20%)	8 (1%)	12 (3%)	12.6
28x28 OCR	5-layer CNN	1.5	Fully binary	92.5% M-OCR	0.007	203	167.9k (82%)	196.5k (48%)	36 (4%)	328.k (74%)	13.2
128x128 OCR	5-layer CNN	147	Fully binary	92.5% M-OCR	0.458	321	187.9k (92%)	203.4k (50%)	71 (8%)	331.5 (74%)	13.1
320x320 OCR	5-layer CNN	1054.3	Fully binary	92.5% M-OCR	3.375	312.4	156.9k (77%)	177.1k (44%)	79 (9%)	361.5 (81%)	13.2
640x640 OCR	5-layer CNN	4406.6	Fully binary	92.5% M-OCR	13.311	337.8	188.1k (92%)	203.6k (50%)	80 (10%)	432.5 (97%)	13.1
28x28 OCR	5-layer CNN	1.5	16-bit	98.9% M-OCR	0.092	16.3	99.6k (49%)	152,6k (37%)	101 (12%)	435 (98%)	13.5
128x128 OCR	5-layer CNN	147	16-bit	98.9% M-OCR	35.001	4.2	100.1k (49%)	152.9k (38%)	136 (16%)	444 (100%)	13.5
320x320 OCR	5-layer CNN	1054.3	16-bit	98.9% M-OCR	257.146	4.1	96.4k (47%)	148.6k (36%)	100 (12%)	443 (100%)	13.5
640x640 OCR	5-layer CNN	4406.6	16-bit	98.9% M-OCR	2498.11	1.8	96.6k (47%)	148.5 (36%)	93 (11%)	436.5 (98%)	13.5
640x640 face det.	9-layer CNN	420	Mixed 1/16-bit	92.3% FDDB	28.027	15	112.2k (55%)	203.0k (50%)	470 (56%)	417 (94%)	13.2

platform (e.g. available memory and number of DSP units), limiting the maximum network size and throughput.

We envision the support for additional resource-efficient low-precision arithmetic operators (e.g. for ternary weight MAC), that can be implemented using similar resource-saving logic as for the presented fully binary layers. Additionally, future versions could implement residual layers and separable convolutions, allowing MobileNets to be automatically mapped using this framework.

4.8 Conclusion

This chapter presented NN2CAM, an end-to-end framework for automatically mapping trained quantized neural networks onto FPGA-based edge processing devices and reported experimental results on an FPGA-based high-speed camera system. The experiments showcased its support for arbitrary fixed-point precisions with an efficient XNOR-implementation for fully binary layers as well as a computational resource balancing mechanism for effective parallelization within each layer and across the entire network. Utilizing the proposed framework for implementing a customized network architecture on the camera simplifies the development process to compiling the trained network into an IP block and instantiating it in the camera firmware. In contrast to other implementations, this framework does not require powerful CPU cores in the FPGA and can thus be implemented on a wide range of FPGA platforms.

Our edge processing experiments implemented on the camera show computational throughputs of up to 337GOPS, which is SoA performance for single-frame inference, and provide the flexibility of running networks at various arithmetic precisions on the example of implementations with fully binary, 16 bit, and mixed precisions.

Chapter 5

Improving Memory Utilization for Convolutional Neural Network Accelerators

While the accuracy of convolutional neural networks has achieved vast improvements by introducing larger and deeper network architectures, the memory footprint for storing their parameters and activations has significantly increased. This trend especially challenges power- and resource-limited accelerator designs, which are often restricted to store all network data in on-chip memory to avoid interfacing energy-hungry external memories. Maximizing the network size that fits on a given accelerator thus requires maximizing its memory utilization.

While the traditionally used *ping-pong* buffering technique is mapping subsequent activation layers to disjunctive memory regions, we propose a mapping method that allows these regions to overlap and thus utilize the memory more efficiently. This chapter presents the mathematical model to compute the maximum activations memory overlap, and thus the lower bound of on-chip memory needed, to perform layer-wise processing of convolutional neural networks on

memory-limited accelerators. Our experiments with various real-world object detector networks show that the proposed mapping technique can decrease the activations memory by up to 32.9%, reducing the overall memory for the entire network by up to 23.9% compared to traditional *ping-pong* buffering. For higher resolution de-noising networks, we achieve activation memory savings of 48.8%. Additionally, we implement a face detector network on an FPGA-based camera to validate these memory savings on a complete end-to-end system.

5.1 Introduction

Performing inference with a CNN is a highly data-intensive task in which input activations, starting with an input image, get convolved with kernels consisting of learned weights, summed up with bias parameters and fed through an activation function. The layered structure of CNNs allows them to be processed sequentially, layer by layer. This is beneficial in terms of memory requirements because a maximum of only two subsequent activation layers, as opposed to all of them, have to be stored at any point in time: the inputs from the preceding layer are needed to be convolved with the kernels, while the results of these computations (output activations) must be buffered to serve as inputs for processing the following layer. Because consecutive layers output their results alternatingly into one of two activation memory sections this pattern is called *ping-pong* processing. The network parameters (weights and biases) are reused at every inference of the network and should thus be kept in local memory to avoid costly data reloading from external memory. To succeed in storing all network data on-chip for layer-wise CNN processing, the memory must be large enough to store the constant parameters and the largest pair of successive input and output activations as shown in Fig. 5.1 (a). With the traditionally used *ping-pong* buffering technique these activations are alternatingly mapped to disjunctive memory regions, such that the worst-case pair of activations amounts to the maximum sum of any two subsequent layers [149]. For CNN accelerators on resource-limited platforms, like FPGAs [276, 277], this constraint largely limits the maximum network size that can be processed.

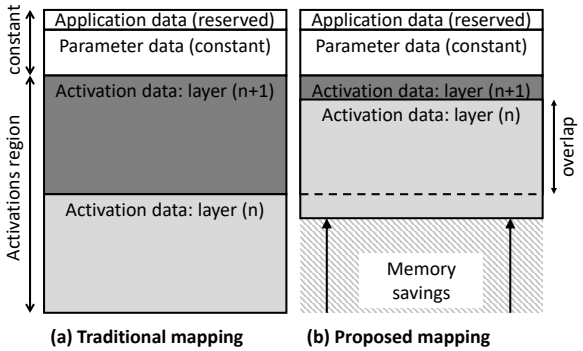

Figure 5.1: Memory allocation of the traditional (a) and the proposed (b) activations mapping approach, visualizing the introduced overlap.

On-chip SRAMs dominate today's CNN accelerator designs (e.g. around 1 mm^2 for 1 MB in 22 nm technology [278]). Reducing memory size will therefore clearly reduce chip area and thus largely influence the chip cost. Additionally, large memories increase the static power consumption due to leakage, and also their energy-per-access is heavily impacted by size, surpassing the energy consumed by the processing of the fetched data by a factor of more than 25× [102, 120]. Thus, it is essential to minimize the on-chip memory size to the targeted networks' needs.

The following sections present our CNN memory mapping method that allows activation regions of subsequent layers to overlap, as shown in Fig. 5.1 (b), and thus utilize the memory more efficiently than the traditionally used *ping-pong* buffering technique. It consists of a mathematical model for computing the maximum overlap and thus its lower bound of on-chip memory needed to perform layer-wise processing of convolutional neural networks. This is especially attractive for newer networks where the memory is dominated by activations. The resulting memory size can be used to determine the minimum memory requirements for a new accelerator design or to optimize a given network to efficiently utilize the memory resources of an existing accelerator. Our experiments show activation memory savings of

up to 32.9% for real-world object detector CNNs and up to 48.8% for high-resolution de-noising CNNs when compared to traditional *ping-pong* buffering.

5.2 Improving CNN Memory Utilization

The traditional *ping-pong* mapping of activations ensures that the outputs of a layer do not overwrite any of its input activations, because they might still be needed for pending computations. But allocating two separate regions for this reason is too pessimistic, unnecessarily restricting the allowed network size, keeping the memory utilization low, and thus the power consumption as well as cost high. In the following sections, we show how the data access pattern of CNNs can be exploited to improve the memory utilization of accelerators.

5.2.1 CNN Data Access Pattern

To determine the memory requirements for computing an entire CNN inference, we need to understand the data access pattern of this process in detail. Fig. 5.2 visualizes the computational structure of a CNN layer, convolving an input feature map *in* (of size $X_{in} \cdot Y_{in} \cdot C_{in}$) with a weight kernel k (C_{out} kernels of size $K_x \cdot K_y \cdot C_{in}$), producing an output feature map *out* (of size $X_{out} \cdot Y_{out} \cdot C_{out}$). To convolve the whole input feature map, input activations are accessed in a sliding window operation, moving the kernel-sized window across the x/y plane. This operation can be represented with the 6 nested loops shown in Algorithm 1. The window moves in strides of s_x and s_y in x- and y-direction, respectively. Inputs can be padded with P_x and P_y zero-pixels on each side in x- and y-direction.

Input data are stored in memory in the depth-first order: all C_{in} input channels of an input pixel are followed by all entries of the neighboring input pixels in the x-direction. At the end of a row, the following rows in the y-direction are appended. This simplifies the addressing scheme and keeps the number of cycles between data reuses low by following the window pattern. Minimizing this so-called reuse distance [148] is important as it allows a specific memory entry to be overwritten as soon as possible, freeing space for new data. Because

output feature maps will serve as inputs in the next layer, they must have the same memory order as the input feature maps.

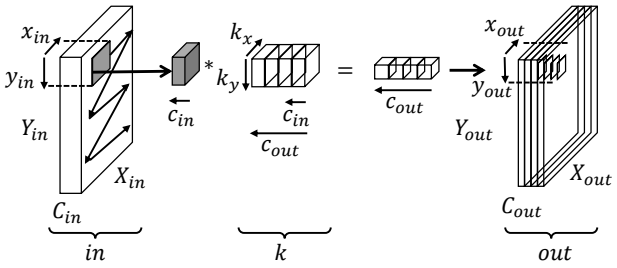

Figure 5.2: Visualization of the 2D convolution operation in a CNN layer with relevant dimensions of the input and output activations as well as the kernel.

Algorithm 1 Computation loops of a CNN layer i (omitting the accumulation reset, the input padding and the activation function at the end of each c_{out} loop)

1: **for all** y_{out} in 0 to $Y_{out}(i)$ **do**
2: **for all** x_{out} in 0 to $X_{out}(i)$ **do**
3: **for all** c_{out} in 0 to $C_{out}(i)$ **do**
4: **for all** k_y in 0 to $K_y(i)$ **do**
5: **for all** k_x in 0 to $K_x(i)$ **do**
6: **for all** c_{in} in 0 to $C_{in}(i)$ **do**
7: $y_{in} = y_{out} \cdot s_y(i) - P_y(i)$;
8: $x_{in} = x_{out} \cdot s_x(i) - P_x(i)$;
9: $out(y_{out}, x_{out}, c_{out})$ += ...
10: $in(y_{in} + k_y, x_{in} + k_x, c_{in}) \cdot k(k_y, k_x, c_{in}, c_{out})$;
11: **end for**
12: **end for**
13: **end for**
14: **end for**
15: **end for**
16: **end for**

5.2.2 Model for Optimized Memory Utilization

To optimize the memory utilization in CNN processors we propose a memory mapping method that allows activation memory regions of consecutive layers to be overlapping. Fig. 5.2 shows a simplified memory map that compares the *traditional* memory allocation (a) with our proposed approach (b). While (a) is keeping each layer's activations in separate disjunctive regions, (b) allows the activation regions to be partially overlapping, resulting in large memory savings.

If two subsequent layers have overlapping memory regions, the allocation method must avoid that output activations are overwriting data from the preceding layer that is still needed for pending computations. This constraint can be ensured by (negatively) offsetting the output write pointer in such a way, that the input reading pointer will never be reached during the computation of any layer in the network. This concept is depicted in Fig. 5.3. At the beginning of each layer computation (here denoted as t=0), the distance between the input activations read pointer p_r and the output activations write pointer p_w, is set to an optimized offset. As the sliding window for computing the convolution operation moves on, pointer p_w writes results and increments in direction of p_r. Pointer p_r moves accordingly on the input activations region, away from p_w. The underlying idea of this memory mapping is the locality of the convolution operation: the activations for each window position are only read from a small, connected region of the input layer which itself is slid over the inputs in a continuous fashion. Because the corresponding memory data is ordered in the same way as they appear in this sliding operation, most parts of the processed input data will never be used again and can thus be overwritten by resulting output activations.

The maximum overlap of the two activation regions is found by mathematically describing the pointer positions and optimizing their relative offset distance at the beginning of each layer such that the total memory is minimal while constraining the write pointer to be smaller than the read pointer, avoiding any overwriting of still needed data. To meet this constraint, both pointer positions must be known for every point in time. They can be calculated from their starting points and velocities, derived from the network characteristics. We model the pointer positions (addresses) as a function of time, assuming

one MAC operation per clock cycle (t) and that activation data is stored in the depth-first order described above. The sliding window follows this pattern and only moves on once all outputs for a certain window position are computed.

Equation 5.1 represents the velocity of pointer p_w, which advances 1 position per calculation of a single kernel convolution (or $C_{in} \cdot K_x \cdot K_y$ MAC operations). The address only increments once the full kernel is computed, which can be mathematically represented by rounding down the integral of its speed over time as shown in Equation 5.2. The formula for p_r takes the padding of the input layer, stride width, and the behavior during sliding window movements into account. It is sufficient to look at the lowest address of the input window (which would collide with p_w at the earliest), simplifying the formula of its average velocity to Equation 5.3. In the resulting p_r Equation 5.4, the y-direction stride is implemented with rounding operations, causing the pointer to skip some rows when moving in the y-direction. The minimum memory M_{min,l_n} required for mapping the activations of two subsequent CNN layers to the memory can then be determined from Equation 5.2 and 5.4, as shown in Equation 5.5. It is given by the sum of the input activations space M_{l_n} and the minimum offset difference $(p_{r0} - p_{w0})$ for which $p_r(t)$ is larger than $p_w(t)$ throughout the entire computation of a layer l_n (during the interval T_{l_n}). From Equation 5.5, the lower bound of activations memory required for computing the entire CNN, M_{min}, can be derived by finding the minimum memory size that supports all layers of the network, as shown in Equation 5.6.

$$v_{p_w} = \frac{1}{C_{in} \cdot K_x \cdot K_y} \tag{5.1}$$

$$p_w(t) = \lfloor v_{p_w} \cdot t \rfloor + p_{w0} \tag{5.2}$$

$$\begin{aligned} v_{p_r} = & \frac{s_x \cdot C_{in}}{C_{out} \cdot (C_{in} \cdot K_x \cdot K_y)} + \dots \\ & + \frac{(s_y - 1) \cdot C_{in} \cdot X_{in}}{\left(\left\lfloor \frac{2 \cdot P_x + X_{in} - K_x}{s_x} \right\rfloor + 1\right) \cdot C_{out} \cdot (C_{in} \cdot K_x \cdot K_y)} \end{aligned} \tag{5.3}$$

$$p_r(t) = max(0, \left\lfloor \frac{1}{C_{out} \cdot (C_{in} \cdot K_x \cdot K_y)} \cdot t \right\rfloor \cdot (s_x \cdot C_{in}) + \dots$$

$$
\begin{aligned}
&+ \left\lfloor \frac{1}{\left(\left\lfloor \frac{2 \cdot P_x + X_{in} - K_x}{s_x} \right\rfloor + 1\right) \cdot C_{out} \cdot (C_{in} \cdot K_x \cdot K_y)} \cdot t \right\rfloor \cdot \ldots \\
&\cdot((s_y - 1) \cdot C_{in} \cdot X_{in}) - C_{in} \cdot X_{in} \cdot P_y - \ldots \\
&- \left\lceil \frac{1}{\left(\left\lfloor \frac{2 \cdot P_x + X_{in} - K_x}{s_x} \right\rfloor + 1\right) \cdot C_{out} \cdot (C_{in} \cdot K_x \cdot K_y)} \cdot t \right\rceil \cdot \ldots \\
&\cdot max(0, \left(\left(\left\lfloor \frac{2 \cdot P_x + X_{in} - K_x}{s_x} \right\rfloor + 1\right) \cdot s_x - X_{in}\right) \cdot \ldots \\
&\cdot(s_x \cdot C_{in}))) + p_{r0} \qquad (5.4)
\end{aligned}
$$

$$
\begin{aligned}
M_{min,l_n} &= min_{p_r(t) > p_w(t) | t \in T_{l_n}} \left(M_{l_n} + (p_{r0} - p_{w0})\right) | \ldots \\
T_{l_n} &= \left[0, \left(\left\lfloor \frac{2 \cdot P_x + X_{in} - K_x}{s_x} \right\rfloor + 1\right) \cdot \ldots \right. \\
&\left. \cdot \left(\left\lfloor \frac{2 \cdot P_y + Y_{in} - K_y}{s_y} \right\rfloor + 1\right) \cdot C_{out} - 1\right] \qquad (5.5) \\
M_{min} &= max_{l_n \in L} \left(M_{min,l_n}\right) \qquad (5.6)
\end{aligned}
$$

The worst-case scenario for memories in layer-wise CNN accelerators is a network with two maximum-sized layers back-to-back, requiring an activations buffer of twice the maximum layer size when using the traditional *ping-pong* buffering. For the same scenario, our method can reduce the memory needs by almost 50% if each input pixel creates equally many output pixels, keeping pointers at a constant short offset. This represents the theoretical upper savings limit of the proposed technique. Our model assumes one datum per memory word but can be easily transferred to multiple data entries per word by linearly scaling down the speed of each pointer accordingly. To support residual layers, M_{l_n} must additionally include the activations of the parallel bypass connections.

5.3 Experiments and Results

We evaluate the memory savings of the presented method on four real-world CNNs: 9-layer DLIB face detector [279], 12-layer YOLO lite [280], 20-layer DMCNN-VD 3x3 [281] and 12-layer MobileNetv2 [14]. The first three have an input resolution of 640x640, while the input

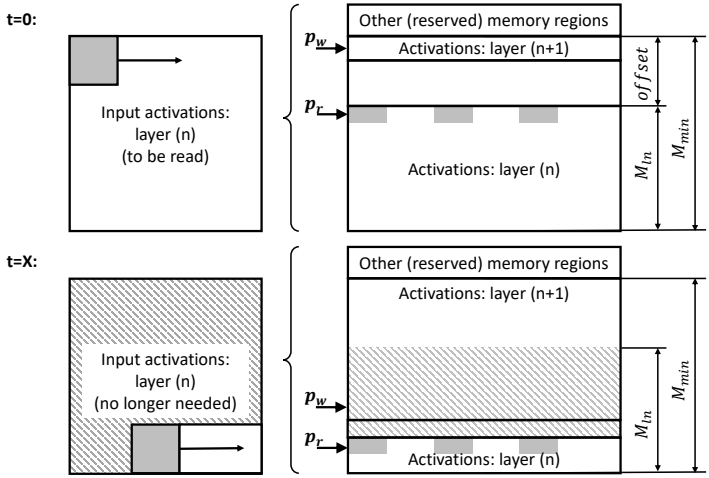

Figure 5.3: Simplified memory map with current pointer positions at two different points in time. The left figures show the current position of the sliding window while the right figures visualize the memory content (including window data).

of MobileNetv2 is 224x224x3. Table 5.1 presents the memory savings of our proposed method compared to traditional ping-pong buffering. Our approach is saving between 19.6% and 48.8% of activations memory in the evaluated networks, achieving total memory savings (including parameters) of 6.2% to 48.2%. The lowest overall savings are found in MobileNetv2, where parameters dominate the memory due to the deep architecture and the small image size. It must be noted that we compute MobileNetv2 in a strictly layer-wise manner, while [14] suggests that operations of some intermediate layers could be concatenated without buffering the respective layers entirely. Many recent networks feature small kernels and larger images, increasing the dominance of activations in memory and thus memory savings. This can be seen in the 20-layer DMCNN-VD with small 3x3 kernels, yielding 48.2% total memory savings. We note that our technique

still offers significant (32.9%) activation memory savings for smaller networks, such as DLIB.

To validate the memory savings in a real application, we employ our method for implementing a DLIB face detector CNN [279] on a configurable FastEye camera [257]. This hosts a 1-megapixel image sensor and a Xilinx XC7K325T FPGA. We extend the existing datapath, implementing the sensor readout and a USB interface, with a simple CNN processing state machine and a BRAM consisting of 36 kbit blocks for parameters and activations. The image from the sensor gets cropped to 640x640 pixels and stored on the BRAM. Triggered by the image, the state machine processes the network layer-wise as described in Algorithm 1. Without any processing optimizations, the maximum CNN inference rate is 0.5 frames per second at 100 MHz clock frequency. Weights and activations are quantized to 16 bits. The output of the last CNN layer gets transferred via USB to a computer for post-processing of the resulting bounding boxes. Two different system configurations are implemented, a) with the standard ping-pong mapping, and b) with our proposed memory mapping technique, differing only in the BRAM size and the address generation. Both FPGA implementations successfully perform on-camera face detection on acquired images. Table 5.2 states the utilization report of the initial FPGA firmware and the two CNN-extended versions. Comparing the resources added to the initial camera firmware, the proposed memory mapping (b) shows memory savings of 23% and power savings of 20% with respect to the standard memory allocation (a). The number of used flip-flops (FFs), LUTs, and DSPs in the FPGA rests almost constant. This confirms the theoretical savings (23.9%), differing by only 0.9%, which is due to the limited block granularity of the memory macro.

Fig. 5.4 illustrates the optimal pointer positions for the DLIB CNN. In layer 0 (a), the read pointer is around 4× faster than the write pointer due to its input-to-output activations compression ratio of 4×. Thus, the read/write pointers are the closest at the beginning of the layer computation. The opposite applies to layer 1 (b), where the output is 4× larger than the input, making the pointers approach towards the end. The jumps in layer 4 are due to the vertical stride of this layer, jumping over some input rows for each output row. Another feature that can be seen in these plots is the influence of the padding

Table 5.1: Results of the evaluated networks

CNN network				Mem. savings: activations only (total network)
Network name	Parameters [#words]	Activations [#words]		
		Standard	This work	
DLIB face det. [279]	229.8k	614.4k	412.2k	32.9% (23.9%)
YOLO Lite [280]	443.0k	16.4M	13.1M	19.9% (18.7%)
MobileNetv2 [14]	3.3M	1.5M	1.2M	19.6% (6.2%)
DMCNN-VD [281]	668.2k	53.7M	27.5M	48.8% (48.2%)

Table 5.2: FPGA utilization report and power measurements of the camera

	LUT	FF	DSP	BRAM	Power
Cam only	28.6k (14%)	82.4k (20%)	8 (1%)	12 (3%)	12.61 W
a) Cam + CNN	52.9k (26%)	98.3k (24%)	13 (2%)	428 (96%)	13.20 W
b) Cam + CNN	52.5k (26%)	98.3k (24%)	13 (2%)	332 (75%)	13.08 W
Savings: b vs. a	2%	0%	0%	23%	20%

and the kernels: while the write pointer continuously increases, the input pointer waits during padding (because the padding pixels are not stored in the memory) and lags behind (same input window is read multiple times for each output position).

5.4 Related Work

Stoutchinin et al. [148] present an optimal model search approach that outputs optimized CNN loop order, tiling, and buffer size parameters to reduce access to external memories. They achieve memory bandwidth reductions of up to 14× compared to previous implementations. Yang et al. [120] propose an analytical approach to model data locality in CNNs to find the optimal blocking strategy that maximizes the energy efficiency of an accelerator. Other works like [149] focus on optimizing data movements between internal and external memory while using traditional ping-pong buffering for on-chip memory. These

approaches either do not consider cases with all activations stored on-chip or base their models on the inefficient ping-pong buffering. In contrast, we provide a more efficient activations mapping that can be used on any platform and only requires the adaption of the addressing scheme. While we focus on standard convolutions, networks with separable convolutions [14] allow intermediate layers to be stored only partially, reducing the memory footprint of intermediate layers with many channels.

5.4.1 Conclusion

This work presented the mathematical model of the lower memory bound for buffering activations in layer-wise convolutional neural network accelerators using overlapping activation regions. We show that the mapping method derived from this model can utilize the memory more efficiently than the standard ping-pong buffering method. This allows reducing the required on-chip memory size of new accelerator designs or to map larger networks to existing resource-limited implementations. Experimental results on real-world CNNs show that the activations memory space can be reduced by up to 48.8% and the overall network memory needs by up to 48.2%.

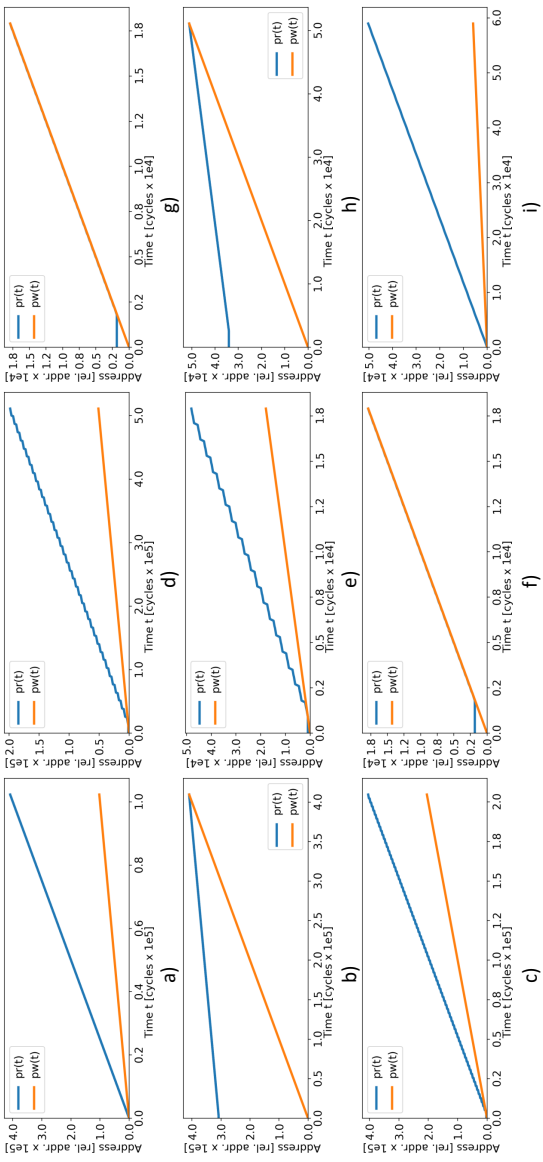

Figure 5.4: Read and write pointer positions over time for each layer of the DLIB face detector CNN. Subfigures (a)-(i): layers 0-8.

Chapter 6

Dual-Engine Machine Learning Inference System-on-Chip for Sub-mW Face-Analysis at the Edge

Always-on ML applications are increasingly deployed to strictly size and energy-constrained smart IoT platforms, requiring high computational efficiency in the sub-mW power domain. Many applications do not require the full complexity of the algorithms to be always active and can therefore be computed hierarchically with increasing computational complexity [62]. This allows complementing purely computation-based optimizations, adding complexity-scaling approaches as an orthogonal optimization strategy.

This chapter presents an SoC that enables hierarchical processing of face analysis tasks under multiple sub-mW operating scenarios using two tightly coupled ML accelerators, as shown in Fig. 6.1. A dynamically scalable BDT engine enables ultra-low power face detection (FD), allowing to trigger a more complex analysis using a

multi-precision CNN engine for subsequent face recognition (FR). The 22nm SoC can therefore dynamically trade-off image analysis depth, FPS, accuracy, and power consumption. It implements complete end-to-end edge processing, enabling always-on FD and FR within the tight 1mW power budget of a 55mm diameter indoor solar panel. The SoC achieves >2× improvement in energy efficiency at iso-accuracy and iso-FPS over SoA systems.

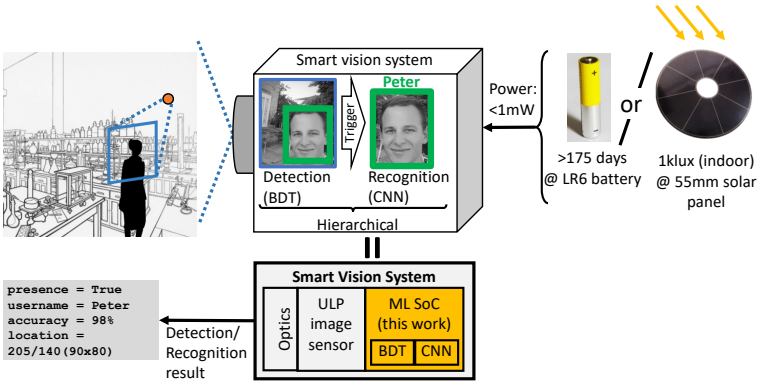

Figure 6.1: Smart vision system for ULP face analysis at the edge using dual-accelerator SoC.

6.1 Introduction

ML models are computation and memory-intensive algorithms. To run them on miniaturized IoT platforms with limited battery capacity, energy-efficient processing engines are required, ensuring long battery lifetimes. State-of-the-art edge processing designs achieve this by restricting their flexibility to either 1) efficient, but application-limited, accelerator types [219] or 2) minimal on-chip memory size (<500kB) [62, 68, 282]. These restrictions confine their scope to simple use-cases with a static trade-off between power consumption and inference rate (FPS).

In contrast, the performance of this SoC can be dynamically adapted to different application requirements as shown in Fig. 6.2, ranging from low power BDT-based FD to CNN-based FR. The BDT-based FD is scalable through 4 orthogonal parameters: 1) in-plane rotation, 2) supported range of face sizes, 3) detection granularity, and 4) classifier depth. CNNs only allow such scaling during training time (e.g. adjusting the network complexity to a specified range of supported rotation angles). The hierarchical approach exploits the BDT as an always-on engine, while the more power-hungry CNN is only triggered when needed. Both steps could be combined in a single algorithm that is evaluated for every frame, but we exploit the causality of the task, knowing that face recognition is only meaningful if there is a face to be evaluated.

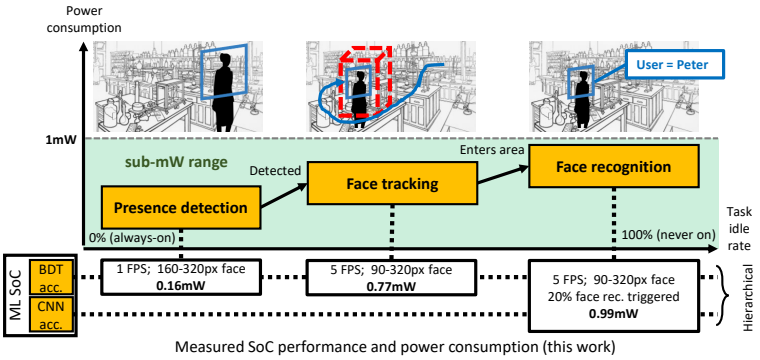

Figure 6.2: Multi-scenario face analysis illustrating power/complexity scalability of ML SoC.

Contributions: This work has been a joint effort of myself, Erfan Azarkhish, Regis Cattenoz, and Engin Turetken with inputs from Luca Benini and Stephane Emery. My contributions have been focused on the CNN accelerator architecture as well as the system and application design, whereas Engin Turetken has focused on the BDT accelerator, Erfan Azarkhish has contributed to the system and

back-end implementation, and Regis Cattenoz has helped with the peripherals.

6.2 System-on-Chip

Fig. 6.3 shows the architecture of the ML SoC, comprising two accelerators, a 32-bit RISC-V microcontroller core, and peripherals to directly interface external sensors. The Octo-SPI interface provides up to 180MB/sec transfer rate, reducing pin toggling power by 41- 92% with respect to single-line SPI. Two word-interleaved multi-bank SRAM memories, L1 (64kB, 16 banks) and L2 (1MB, 8 banks), distribute sequential accesses over the banks, and provide uniform addressing with power-gating options for simple applications. Interconnection trees with pseudo-least-recently granted arbitration provide fair access to master and slave ports. The two accelerators share their memory ports in a time-multiplexed fashion, enabling them to directly access L1 and L2.

Embedded in a smart vision system, as shown at the top of Fig. 6.3, the SoC provides an automatic booting process that allows copying memory content from an external non-volatile memory (through single-line SPI) to its on-chip SRAM. This allows loading execution code and ML parameters after startup, avoiding further external components to program the system. Images from a connected sensor can directly be loaded through DMA transfers into the memory, from where the accelerators can access it for analysis. The SoC is controlled by the 39.8kGE RISC-V core, which achieves 3.2 CoreMark/MHz at 2.23μW/MHz. It provides the flexibility to implement complete application functions, including data pre- and post-processing using direct access to the SRAM memories, enabling standalone end-to-end processing with a single chip.

6.3 BDT and CNN Machine Learning Accelerators

The dual-engine ML accelerator contains the BDT and CNN engines, which can run independently but share the memory interface using a

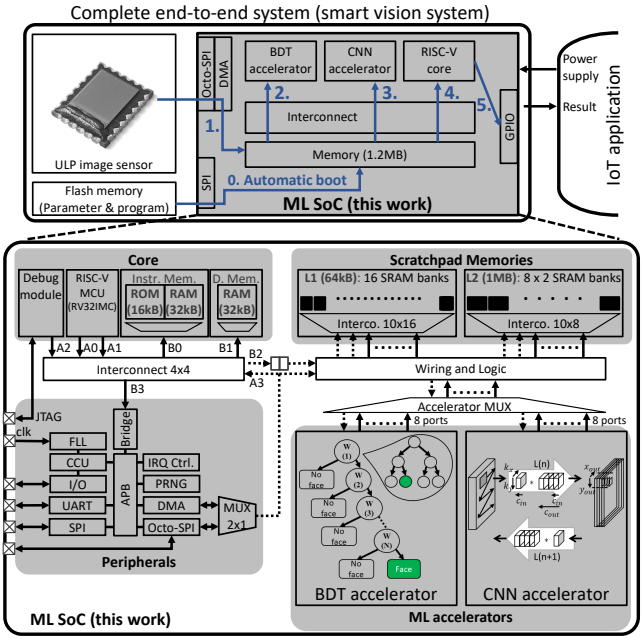

Figure 6.3: SoC architecture (bottom) with BDT and CNN accelerators, RISC-V core, 10-port memories, and peripherals to image sensor and ext. memory, embedded in a smart vision system (top) for edge ML.

multiplexer. In the following sub-sections, we present the architecture of each accelerator.

6.3.1 Binary Decision Tree Accelerator

Fig. 6.4 shows the BDT accelerator, implementing 8 threads of an AdaBoost-based [283] hierarchical classifier, composed of 416 cascaded weak classifiers (WCs), using nested hardware loops in finite-state machines (FSMs). The RISC-V core computes a list of bounding box (BB) windows to be evaluated for a given window scale and orientation and offloads them to the BDT engine. The scheduler distributes the

BBs across the execution units and aggregates the results. For each BB, the same classifier cascade is traversed. As soon as a negative classification is registered, the tree is aborted, reducing the execution time for simple scenes. Two WC execution modes are available, as shown in Fig. 6.4: the *COMPACT* and the LUT-based *FAST* mode (13% more memory, but 41% fewer execution cycles required). A configurable depth-threshold allows dynamically changing from *FAST* to *COMPACT* scheme, trading-off latency versus memory needs.

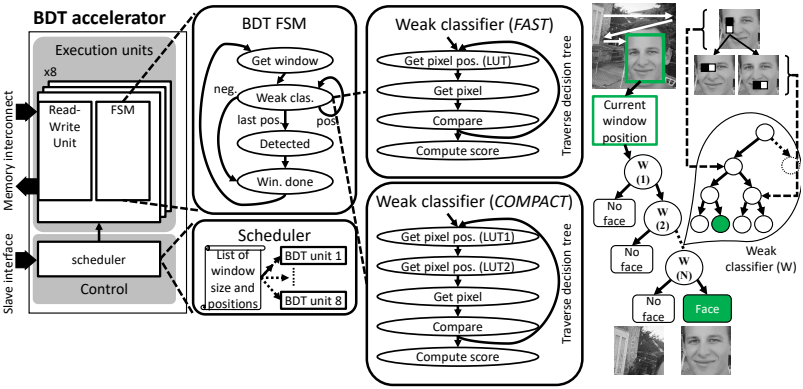

Figure 6.4: BDT accelerator block (left) with programmable FSMs to dynamically trade-off performance and power versus accuracy. The depicted decision tree (right) illustrates the cascade of weak classifiers to be traversed for analyzing a selected window on the input image.

6.3.2 Convolutional Neural Network Accelerator

The architecture of the multi-precision layer-wise CNN accelerator is shown in Fig. 6.5. For each layer, it configures its blocks according to layer definitions, read from a pre-compiled linked list in the memory, allowing to implement arbitrary CNNs. The control FSM requests data through 8 independent and pipelined memory interfaces. Four programmable address generators determine what data (activation,

weight, bias, or scaling factor) to read and where to write back the results, allowing to generate the sliding window pattern of the convolution. The FIFO registers decouple memory access from the 8 (output-stationary) PEs, which perform 16 parallel 16bit fixed-point multiply-accumulate operations per cycle. Their weight precision is configurable to 16bit or 1bit, supporting CNNs with variable per-layer quantization. Input activations are shared across the PEs, enabling up to 8× activation data reuse, and thus reducing memory accesses.

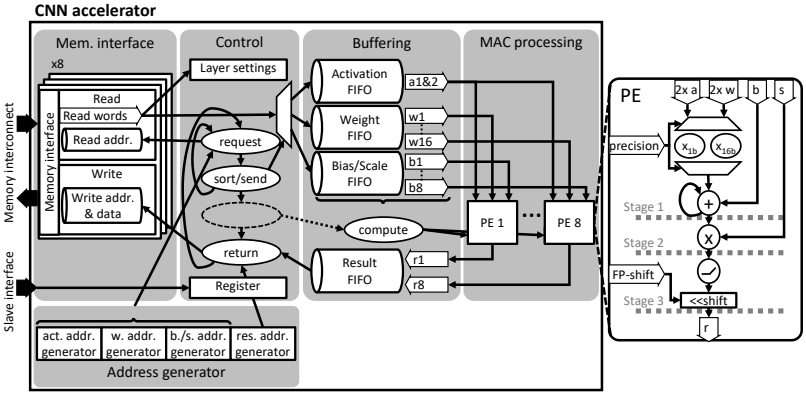

Figure 6.5: Multi-precision CNN accelerator block with 8 memory ports and 8 processing elements containing a total of 16 parallel 1/16bit weight MAC units.

6.4 Scalable Dual-Accelerator Operation

Fig. 6.6 shows the ML processing scalability, reaching a ratio of up to 1'142'000× minimum-to-maximum FD/FR inference energy. The BDT enables 1900× and 21× execution time scaling by adapting the minimum window size (30 to 320pixel) and the maximum rotation angle (360° to 30°), respectively, as shown in Fig. 6.7. Lowering

the CNN weight precision from 16bit to 1bit, reduces energy per operation by 61% and memory needs by up to 83%, as shown in Fig. 6.8. However, data access energy dominates (65-70%) both accelerators. Fig. 6.9 further illustrates the operating points of the BDT and the CNN engine for different configurations. While both the BDT execution and the 16bit CNN inference are memory-bound due to their data-intensive characteristics, the binary-weight CNN mode can largely reduce its memory needs, rendering it computation-bound. This enables the CNN to achieve 0.32-1.07 TOPS/W efficiency including memory power.

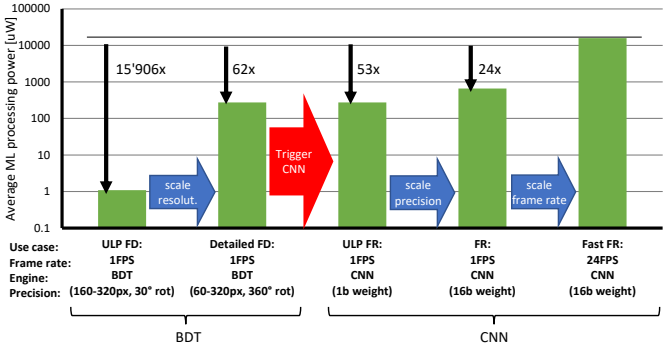

Figure 6.6: Measured ML processing energy versus performance scaling capabilities of the SoC.

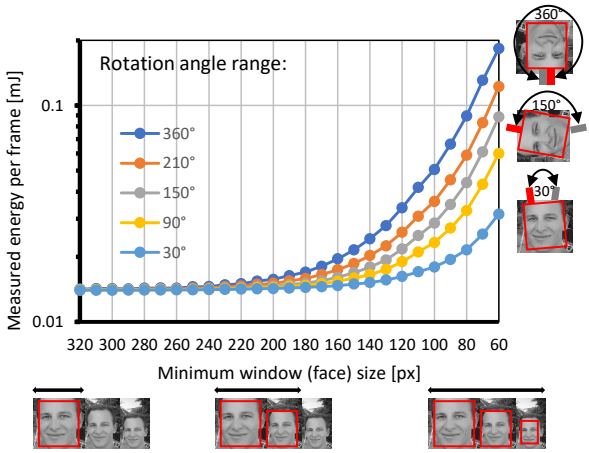

Figure 6.7: BDT energy scalability through minimum window size and supported rotation angle selection.

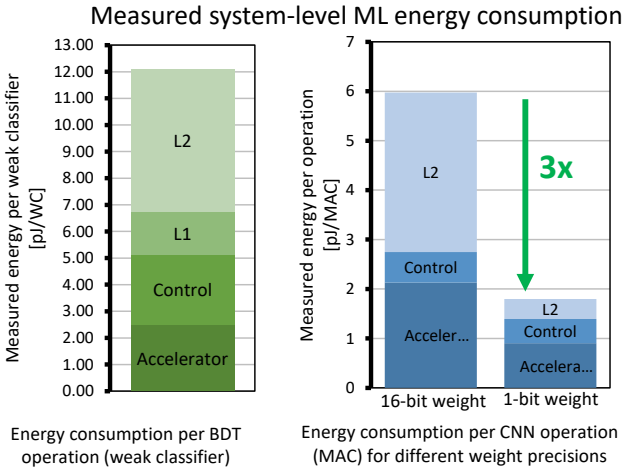

Figure 6.8: Measured ML processing energy per operation of the BDT and CNN accelerators.

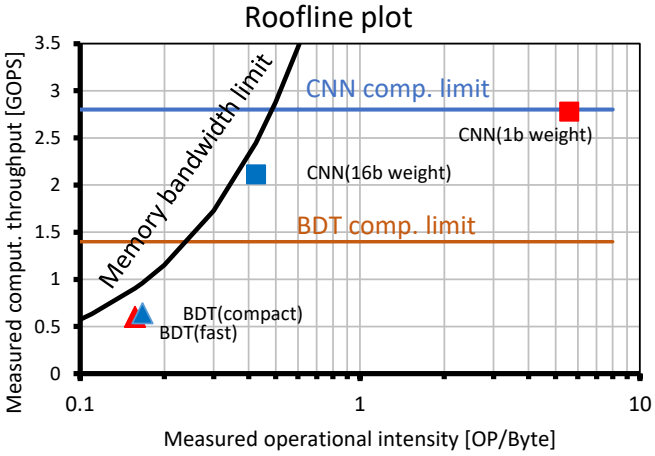

Figure 6.9: Roofline plot, illustrating the operating points of the BDT and CNN accelerators.

6.5 Implementation Results

The SoC is characterized in multiple sub-mW application scenarios using an in-house 320x320 pixel image sensor [284]. Measurement results are summarized in Table 6.1, reporting the power consumption along with the activity and complexity settings of the BDT and CNN engines. The executed FD BDT achieves 98% accuracy at <6% false-positive rate on the FDDB benchmark, triggering upon detection a 6-layer (212MOP) 1bit weight CNN for FR, achieving 96% on the LFW dataset [285]. This 22nm CMOS SoC runs up to 180MHz@0.65V (220MHz@0.65V with 0.8V forward body biasing (FBB)). It achieves a measured peak computational throughput of 1.44GOPS and 5.76GOPS for BDT and CNN inference, respectively. Its main characteristics are summarized in Table 6.3. Fig. 6.10 show the manufactured $3.4mm^2$ chip with an overlay of the main blocks.

Combining both accelerators offers algorithms dynamic power scaling options, achieving 0.41mW at 1 FPS FD and FR, totaling 0.64mW for the smart vision system, which is >2× improvement in energy efficiency over SoA systems (see Table 6.2) at iso-accuracy and iso-FPS [219]. The test board used for the measurements is depicted in Fig. 6.11. It features the ULP image sensor, a 2MB Flash memory for automatic booting, as well as communication interfaces for programming and debugging.

6.6 Neural Network Mapping Flow

Algorithm developers can rely on a set of wide-spread NN tools, which can be considered a de facto standard, for designing and training networks. However, to map a trained network onto a processing system and its accelerators, the network has to be translated into a device-specific representation, which might require developers to understand the detailed working principle of the hardware and perform a complicated memory mapping for each network architecture. We avoid this by providing a tool-flow for automatically generating the required on-chip memory content, which the CNN accelerator directly accesses to execute the trained network. Fig. 6.12 illustrates the mapping flow, starting from a trained network and resulting in an

Table 6.1: Measurement results of the smart vision system scenarios

Scenario	**ULP FD**	**ULP FD and FR**	**360° FD**	**Full dist. FD**	**FD and FR upon trig.**	**Max. FPS FD and FR**
Frame rate [1/s]	1	1	5	5	5	20
Min. detected face size [px]	160	160	90	60	120	60
Covered face angle [°]	30	30	360	30	90	360
FD (BDT acc.) enabled [%]	100%	100%	100%	100%	100%	100%
FR (CNN acc.) triggered [%]	0%	100%	0%	0%	20%	100%
Measured power SoC [mW]	0.16	0.41	0.7	0.85	0.99	12.77
Measured power ext. (incl. sensor) [mW]	0.23	0.23	1.12	1.12	1.12	4.46
Measured power total [mW]	0.39	0.64	1.89	1.97	2.11	17.23

Table 6.2: Comparison with state-of-the-art designs

	ISSCC 2017 [219]	**ISSCC 2017 [62]**	**ISSCC 2018 [282]**	**VLSI 2020 [68]**	**This work**
Application	FD[1] + FR[2]	FR[2] only	FR[2] only	Presence det.	FD[1] + FR[2]
Algorithm	Viola-Jones, CNN	CNN	DNN	FC	BDT + CNN
Algorithm complexity	Single-angle, limited-scale	Single-angle, single-scale	Single-angle, single-scale	N/A	Multi-angle, multi-scale
Hierarchical processing	Yes	Yes	No	Yes	Yes
Image size [px]	320×240	N.A.	48×48	224×224	320×320
Technology	65nm	28nm	65nm	28nm	22nm
Core area (mm^2)	16	1.87	16	4.5	2.1
Supply voltage (V)	0.46-1.0	1	0.63-1.1	0.48-0.9	0.65 (+/-10%)
Memory (kB)	164	152	256	464	1176
Off-chip memory	N/A	Yes	Yes	No	No
Frequency	50MHz @ 0.46V 100MHz @ 1.0V	200MHz @ 1.0V	5MHz @ 0.63V 200MHz @ 1.1V	25MHz @ 0.48V 350MHz @ 0.9V	180MHz @ 0.65V (no bias) 220MHz @ 0.65V (FBB)
Energy per pixel (power @1FPS/ #pixels)	8.1nJ/ px @0.46V, FD+FR	103.4nJ /px @1V, FR	N/A	6.3nJ/ px @ 0.9V SC[3]	4.0nJ/ px @0.65V, FD+FR

[1]Face detection(FD) [2]Face recognition [3]Scene classification @ 0.2FPS (scaled-up to 24h)

Table 6.3: Summary table of implemented ML SoC

System	Technology	GF 22nm FDX
	Die area	1.85mm × 1.85mm (3.42mm^2)
	Core area	1.42mm × 1.48mm (2.10mm^2)
	Total SRAM	1.2MB
	Core supply voltage	0.65V
	SRAM supply voltage	0.8V
	I/O supply voltage	1.8V
	Microprocessor	RV32IMC (without divider) 3.2 CoreMark/MHz (-O3), 2.23µW/MHz
	Programmable FLL range	125-265MHz
	Frequency	180MHz (220MHz @ 0.8V forward bias)
CNN	Supported layer types	CNN, FC, Avg. pooling, ReLU
	Weight precision	1/16 bit
	Activation precision	16 bit
	Bias/Scaling precision	16 bit
	Peak throughput	5.12-5.76GOPS* @ 180MHz
	Measured energy efficiency	0.32-1.07 TOPS/W*
BDT	Arithmetic precision	32 bit
	Peak throughput	1.44GOPS** @ 180MHz

*Each MAC computation is considered as 2 operations. The range indicates the performance difference of the two precision options (1/16 bit weights) and includes the memory access power. **Each BDT comparison computation is considered a 1 operation.

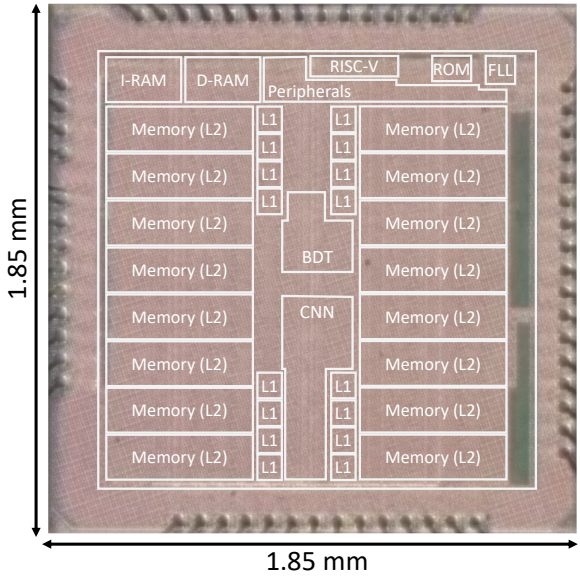

Figure 6.10: Die micrograph of the manufactured chip with an overlay of the main blocks.

executable ELF file. For booting from an external Flash memory, a BIN file can be exported alternatively. The architecture parsing and parameter extraction steps, currently supporting Caffe networks, are based on the work from Chapter 4. It automatically combines convolutions, biasing, normalization, scaling, and activation layers as they are computed as a single layer within the accelerator. The microcontroller application and the generated memory for the NN are then compiled through a standard RISC-V compilation flow.

6.7 Conclusion

This work presented a novel 22nm FDX ML inference architecture for sub-mW face analysis applications at the edge. Its dual-engine

setup with a flexible RISC-V microcontroller demonstrated hierarchical complexity scaling, running simple detection tasks on the dynamically scalable low power BDT accelerator, while offering on-demand CNN acceleration for more complex (recognition) tasks. Combined, the SoC can run face detection and recognition at 0.41mW power consumption, enabling a long battery lifetime for face analysis applications at the edge.

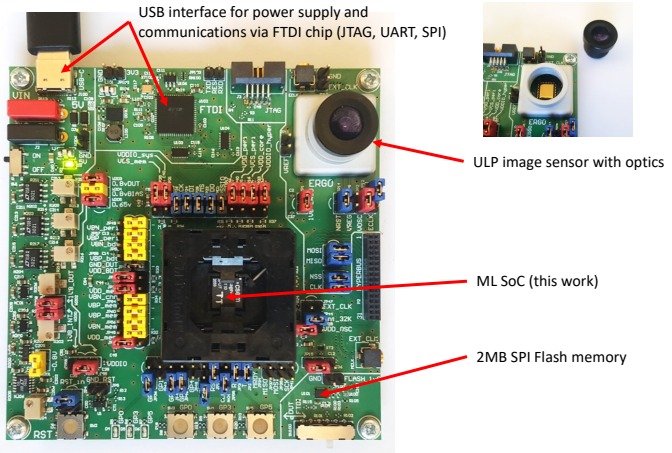

Figure 6.11: Photograph of test and verification board with ULP image sensor and external Flash memory.

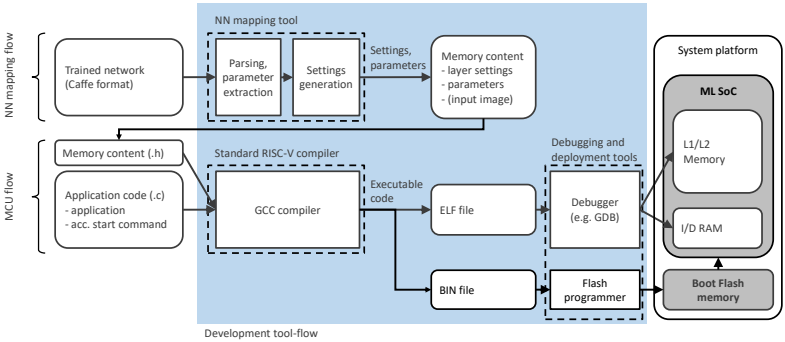

Figure 6.12: Neural network mapping flow.

Chapter 7

Battery-Less Face Recognition at the Extreme Edge

ML-based face recognition systems are commonly used in mobile platforms to assist the camera systems, unlock the device, or analyze the facial expressions of its users. The computational complexity of the underlying algorithms and the power consumption of the entire imaging system largely limit the deployment to powerful mobile processing systems with large rechargeable batteries. However, these computer vision capabilities would also be useful in miniaturized wearables and low-power IoT applications with stringent battery-size limitations.

In this chapter, we assess the feasibility of such a computer vision edge processing system on a battery-less credit card-sized demonstrator using an ultra-low power image sensor and the ML system-on-chip presented in Chapter 6. The system achieves self-sustainable operation using solar energy harvesting with a small on-board solar cell. This enables continuous 1 frame-per-second battery-less imaging and face recognition in indoor lighting conditions.

7.1 Introduction

The majority of today's ML applications are used in environments with virtually unlimited power access (e.g. in a server). However, battery-powered mobile platforms, commonly used in IoT applications [17], are becoming smarter and thus interesting targets for implementing ML algorithms. Mobile platforms employ ML in a rapidly growing number of applications, supporting tasks like visual object detection [249], audio key-word spotting [4], radio signal analysis [286], and many others across a wide range of domains. Most of these algorithms are based on small- to medium-sized NNs, requiring processing systems with high computational throughput and large memory resources. As shown in Chapter 1, the initial solution of offloading the computation to the cloud was replaced by edge computing approaches [18], processing and analyzing sensor input data directly on-board to reduce the power consumption and improve privacy.

Truly mobile applications, with battery lifetimes of more than a month without recharging, require edge ML platforms with a sub-mW power consumption. For instance, to enable a 1-month lifetime using a CR2032 coin cell battery, the system power consumption must be restricted to less than 1mW [287]. This energy limitation can be mitigated by employing on-board energy harvesting, constantly extracting power from the environment. Various harvesting approaches have been evaluated, exploiting solar [103], kinetic [288], vibration [289], or even radio frequency power [290]. The source is continuously tracked to extract the maximum available power, as shown in [291] and [292]. Mobile ML implementations for computer vision (CV) applications have been shown in various previous works and industrial devices. However, they all feature a very limited battery lifetime. Xiaomi's 2-megapixel AI doorbell [23] features face identification and movement detection with up to 60 days of operation using a large 3000mAh secondary battery. The Google Clips camera [42] claims to recognize people in an always-on operation, autonomously deciding when to capture photos and of whom. Its battery allows only 3 hours of operation. Orcam MyEye 2 [22] is a smart camera for blind people that can recognize people and read text out loud. It features a 13 megapixel image sensor and a 350mAh secondary battery for up to 2 hours of operation. EdgeEye [293] presents an end-to-end people

counting system using a 185MOP CNN. Image sampling and processing consume 17.5mJ with an idle power consumption of 430uW. In [135], a combination of a custom 320x240 pixel image sensor and a CNN processing chip is used to perform face detection and recognition, consuming an average of 0.62mW core power, however excluding the power for image transfer and external components. The computation and memory-intensive CV algorithms are challenging the tight power budget of these battery-powered edge processing applications. Thus, they either use low frame rates or accept short battery lifetimes. For embedding real-time CNN-based user identification in a miniaturized "extreme edge" application as illustrated in Fig. 7.1, both powerful CNN processing capabilities and a long lifetime are required.

This work presents a credit card-sized battery-less computer vision platform (Fig. 7.2) capable of performing edge computing for face recognition using an on-board image sensor and a ULP ML SoC while being powered solely by a small solar panel. With an average power consumption of 0.68mW for acquiring images and running face recognition at 1 FPS, 1klux indoor lighting provides sufficient harvesting power to enable self-sustainable operation. The chapter is structured as follows: Section 7.2 describes the system components, followed by the description of the face recognition application in Section 7.3. Experimental results, including power measurements, are reported in Section 7.4.

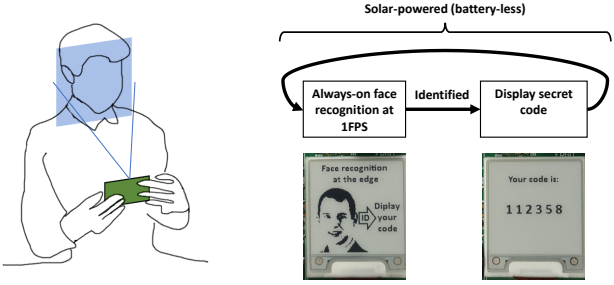

Figure 7.1: Credit card-sized identification using face recognition at the edge.

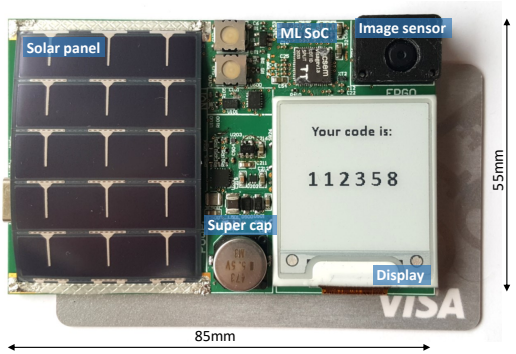

Figure 7.2: Photograph of the computer vision platform with its solar panel (left), display (bottom right), and image sensor below the flat optics (top right) in front of a credit card.

7.2 System Implementation

Fig. 7.3 shows the block diagram of the edge processing platform including the communication interfaces between the sub-systems. It consists of four functional blocks: a) Imaging, b) control and ML-based image processing, c) displaying, as well as d) power management and energy harvesting. The platform, illustrated in Fig. 7.2, is implemented on a printed circuit board with the size of a credit card (55x85mm). Solar harvesting and image processing can be well combined as both operations require lit environments, where image sensing is possible.

7.2.1 Image Acquisition

Operating on a limited power budget requires sufficient image quality in every acquired frame. Iterative exposure time approximation by sampling at different settings should therefore be avoided. Thus, we use an ERGO320 image sensor [284], featuring a high dynamic range of 120dB, enabling high contrast images under strong illumination variations. A miniaturized lens with 2.8mm focal length enables a slim demonstrator (<7mm thick) while providing a wide field-of-view

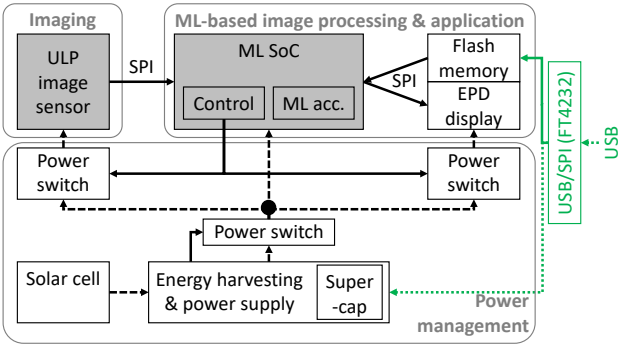

Figure 7.3: System block diagram with power (dashed) and data (solid) interfaces. USB-related circuits (green) are only active during programming.

(107°). The sensor requires a 1.8V supply voltage and communicates through a slave SPI interface.

7.2.2 Control and ML Processing

The heart of the system is our ML SoC, presented in Chapter 6, that orchestrates the sub-systems and provides ML processing capabilities. This work mainly utilizes its RISC-V microcontroller with various sensor interfaces, the 1MB SRAM, and the efficient multi-precision CNN accelerator, supporting 1bit and 16bit weights with 16bit activations. Its automatic booting option is used to start from an external SPI memory, as shown in Fig. 7.3. The SoC directly interfaces the image sensor, the display, and the Flash memory through SPI. The non-volatile 2MB Flash memory (MX25R1635) can additionally be accessed through USB (via a USB/SPI converter), allowing to update the program code and the ML algorithm parameters from a connected PC. Digital outputs are used to control multiple power switches to power-gate unused sub-systems (image sensor, Flash memory, display), minimizing idle currents. The memory-intensive nature of object detection CNNs, requiring millions of operations to be performed for analyzing a single image [249], and the related energy

required to process and move their data [102], strongly influenced the system design. Especially accessing data from external memories is very costly, as the energy for booting indicates, requiring system optimizations to ensure minimal data movements. Therefore, we selected an ML SoC that provides sufficient on-chip memory for face recognition algorithms. Its CNN accelerator with 16 parallel MAC units runs independently of the microcontroller and supports 16bit and 1bit weights (layer-wise configurable), reducing the memory needs for parameters by up to 16×.

The trained (and quantized using the method of [258]) network is compiled in the SoC tool-flow (see Section 6.6), mapping all parameters and layer settings to the on-chip SRAM, which is loaded from the external Flash memory during the booting process. To start the CNN execution, the microcontroller sets the start flag in the CNN register and gets notified by an interrupt upon completion.

7.2.3 Display

A 152x152 pixel E2154CS electronic paper display (EPD) is used to display application results. EPDs feature high contrast, making them well readable even in bright outdoor environments. Other display types, like LCD or LED matrix, require a continuous power supply during operation, while EPDs retain the last image displayed and only require power for updating the content. Fig. 7.4 shows the measured power consumption for a display update. After powering the EPD, its on-board controller is configured, and the new image data is transferred via SPI (busy signal high). Then, the DC/DC converter of the display is started, driving the busy signal low until the power supply is stable. Upon sending the image update command, the EPD starts its internal refresh process, updating the pixels, followed by switching off the DC/DC converter again. The entire update lasts 2625ms and requires 22.8mJ per image refresh. Comparable low-power LCDs [294] consume up to 1000× less power but require a constant power supply and a power-intensive backlight for similar readability.

7.2.4 Power Management and Energy Harvesting

We utilize a BQ25570 solar energy harvester to power the platform. It performs maximum power point (MPP) tracking to efficiently operate the solar cell at its light intensity-dependent MPP. The extracted power is buffered in a capacitor (>3V) and output as a stable 1.8V supply voltage. We use a miniaturized, 5.5mm high, 47mF super-cap instead of a battery, allowing to bridge short energy peaks (e.g. for booting from Flash memory). The buffered energy in the capacitor can be estimated using the formula of the stored energy in Equation 7.1, resulting in 211.5mJ, which is roughly 10× more than a display update requires.

$$E = \frac{1}{2} \cdot C \cdot U^2 = \frac{1}{2} \cdot 47mF \cdot (3V)^2 = 211.5mJ \tag{7.1}$$

The platform is powered by a flexible PowerFilm SP3-37 solar panel but supports other panel types (>100mV, <400mW). Fig. 7.5 shows the characteristics of the employed panel under indoor and outdoor lighting conditions, achieving a maximum output power of 1.05mW and 54.49mW, respectively.

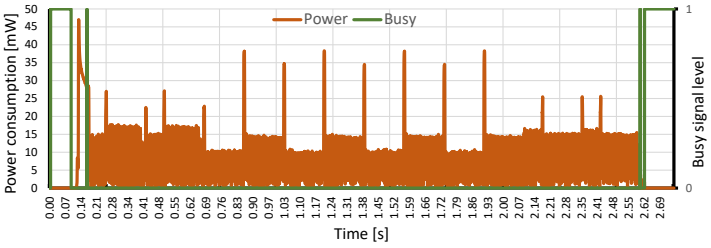

Figure 7.4: Measured EPD display power consumption during a display update.

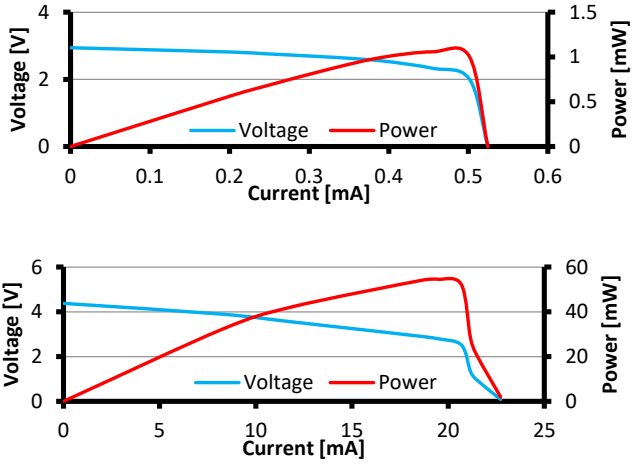

Figure 7.5: SP3-37 characteristics for a) outdoor (20klux) and b) indoor (1klux).

7.3 Computer Vision Application

We demonstrate end-to-end ML edge processing of an always-on face recognition application using solar energy harvesting. The credit card-sized application aims at identifying a specific user to display a secret code (e.g. pin code of the card) upon positive recognition as shown in Fig. 7.1. In the future, the code could be transmitted through RFID.

7.3.1 ML Algorithm

We employ a 6-layer CNN with binary weights for recognizing a face on the input image. Table 7.1 shows its architecture, inspired by [295], totaling 106MMAC operations and 392kB parameters. It takes a 128x128 pixel gray-scale image as input and produces a 512-element output vector using kernel sizes of 3x3-5x5 and 48-256 channels. The face similarity is determined by the Euclidean distance between the output vector and the previously determined reference vector of the

Table 7.1: CNN architecture

Layer	Input size	Input ch.	Output ch.	Kernel size	Stride	#MAC	Param. [kB]
CONV1	128	1	48	5	2	4.9M	0.2
CONV2	64	48	96	3	2	42.5M	5.2
CONV3	32	96	128	3	2	28.3M	13.8
CONV4	16	128	256	3	2	18.9M	36.9
CONV5	8	256	256	3	2	9.4M	73.8
CONV6	4	256	512	4	4	2.1M	262.2
Total						**106.1M**	**392.1**

face to be identified. We define a positive identification as a distance below a specific threshold. Evaluated on the LFW dataset [285], the CNN achieves 96% accuracy in simulation. During layer-wise CNN inference, all network parameters must be stored in memory along with intermediate layer results. To minimize the memory required for buffering activations, we employ the mapping strategy presented in Chapter 5 ([28]). Overlapping the memory space of each layer's input and output activations reduces the activation memory by 32.2% with respect to standard double buffering as shown in Fig. 7.6.

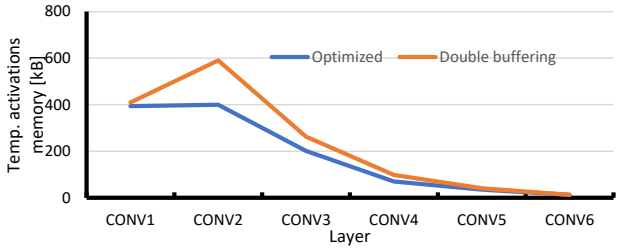

Figure 7.6: Layer-wise activations memory space for evaluated mapping strategies.

7.3.2 Task Scheduling

Self-sustainable operation imposes two power constraints that must be fulfilled at all times: 1) the average power of all executed operations must be lower than the average harvested power and 2) any power peak surpassing the average harvested power must be compensated by the available capacitive load. The flow chart in Fig. 7.7 illustrates the main operations of the application along with the independent energy harvesting. Its timing ensures that both power constraints are fulfilled. When the 1.8V supply voltage is stable, the harvester enables the main power switch. This starts the power supplies and the 32kHz clock generator while keeping the SoC reset during the oscillator startup-time. Releasing the reset automatically starts the booting process in the SoC, loading the program and CNN parameters into the SRAM. The microcontroller then enables the internal 180MHz clock and starts the application. Always-on face recognition is implemented as periodic image sampling with subsequent CNN analysis. Each cycle starts by triggering an image in the sensor through the 20MHz SPI interface. A sensor interrupt starts the SPI image transfer to the SoC SRAM via direct memory access. The CNN accelerator then processes the network layer-by-layer starting from the down-scaled image. An internal interrupt indicates the inference end, waking up the sleeping microcontroller to analyze the output vector. If the face is identified, the display is updated to show the code, followed by the idle screen, after a 60 seconds delay, as shown in Fig. 7.1. Otherwise, the periodic task restarts at 1 FPS.

7.4 Experiments and Results

We measure the solar harvesting and power consumption of the entire system in all operation phases. Fig. 7.8 shows the breakdown of energy per operation across the main sub-systems. Idle energy is measured over 1 second for reference. Image transfer and CNN processing have a similar energy cost while acquiring an image is roughly 3× cheaper. Booting from the Flash memory and updating the display are very intensive, consuming 7.45mJ and 23.87mJ, respectively. The ML SoC consumes 70% of the power during operation. Fig. 7.9 compares

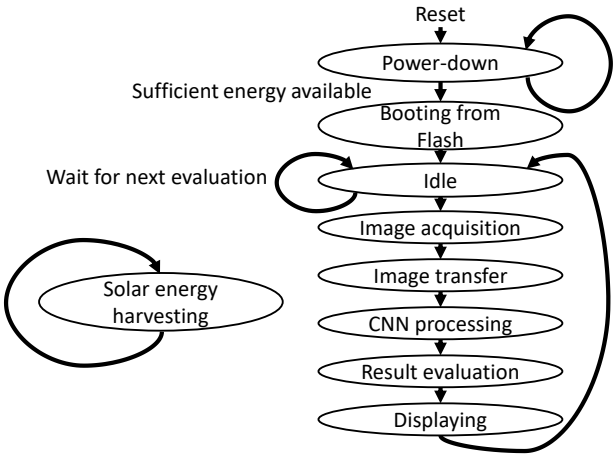

Figure 7.7: Operation flow chart with continuous solar energy harvesting.

the harvested indoor solar power of 0.94mW (90% harvesting efficiency) and the 0.68mW power consumption at 1 FPS, verifying self-sustainable operation. Table 7.2 summarizes the operation phases, highlighting the average power consumption. Note the theoretical values for the operation at the maximum frame rates. Booting and display updating exceed the average harvested power and thus are duty-cycled by appending an idle state phase of 7.5 seconds and 24 seconds, respectively, resulting in an average power consumption below the harvested power of 0.94mW. Outdoor lighting delivers >50× higher harvesting power, which would allow maximum frame rates of 9.6 FPS with a static display and 0.4 FPS with per-frame display update as shown in Table 7.2. However, this requires monitoring the harvested energy and adapting the frame rate accordingly, which the current version of the platform does not support.

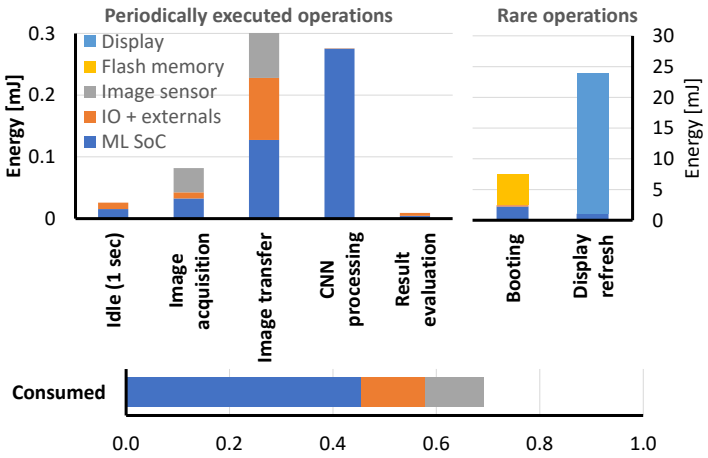

Figure 7.8: System energy per operation (top) and power consumption (bot.) for 1 FPS imaging and CNN processing with break-down across sub-systems.

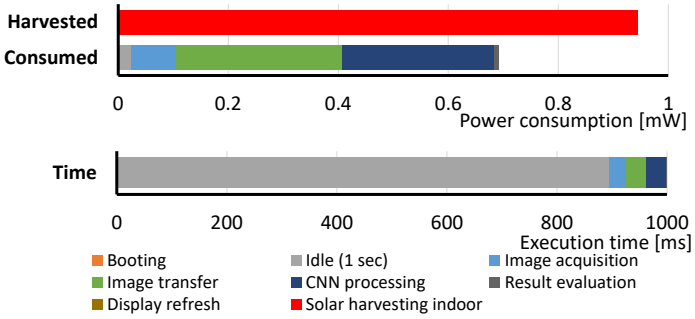

Figure 7.9: System power and execution time break-down across operations for 1 FPS imaging and CNN processing using indoor solar harvesting.

Table 7.2: Operation phases

Scenario	**Booting:** **- boot** **- 7.5s idle**	**Recognition:** **- imaging** **- CNN**	**Displaying:** **- disp. update** **- 24s idle**	**Recognition (@ max FPS):** **- imaging** **- CNN**	**Recogn. & disp. (@ max FPS):** **- imaging** **- CNN** **- disp. update**
FPS [1/s]	N/A	1	N/A	9.6	0.4
Exec. time [ms]	8'365.3	1000.0	26'625.4	1'000.0	1'000.0
Idle time	89.7%	89.6%	90.1%	0%	0%
Av power [mW]	0.91	0.68	0.92	6.33	9.08

*Theoretical scenario to show maximum frame rates possible.

7.5 Conclusion

We presented a credit card-sized and battery-less platform for end-to-end computer vision processing at the edge using solar energy harvesting. The platform allows acquiring images with its on-board image sensor and perform CNN-based face recognition at 1FPS while consuming 0.68mW. This enables self-sustainable operation in indoor lighting conditions and demonstrates the feasibility of implementing miniaturized and battery-less ML applications at the extreme edge.

Chapter 8

Summary and Conclusion

NN-based data analysis is no longer confined to powerful stationary computers but increasingly deployed to miniaturized mobile and near-sensor edge processing systems. The strict size constraints of such systems result in limited battery capacities with tight power budgets, requiring efficient hardware accelerators. To achieve maximum performance, accelerator designs are optimized across multiple fields ranging from improved circuit design to software optimizations.

In this thesis, we have presented multiple approaches to improve the efficiency of NN processing at the edge to enable support for more complex applications and further increase the battery lifetime, moving towards self-sustainable operation. First, we have provided a survey of existing NN accelerators and their optimization techniques, allowing to quantitatively compare and trade-off their effects during the design process (Chapter 2). This is increasingly important due to the myriad of research papers published over the past decade, complicating an objective evaluation. We have further investigated edge processing on high-speed cameras (Chapter 3), demonstrating efficient BNN implementations on reconfigurable FPGA platforms, and presented an automated mapping tool to simplify deployment on FPGAs for non-expert users (Chapter 4). The need for efficient memory utilization

was reported in the FPGA implementations. Therefore, we have developed a technique for improving the memory utilization in layer-wise CNN accelerators, enabling memory savings through memory mapping optimizations (Chapter 5). In Chapter 6 we have presented a new system-on-chip architecture for low power edge ML using hierarchical dual-engine acceleration. We have further presented a battery-less solar-powered platform that exploits the efficient system-on-chip to demonstrate self-sustainable edge ML applications (Chapter 7). We conclude that designing efficient edge ML systems requires optimizations across all design levels, from hardware implementations to efficient network co-design and mapping strategies.

8.1 Overview of the Main Results

This section summarizes the main results of the thesis:

Quantitative Survey of Optimization Techniques

The success of NN-based analysis has led to an explosion of publications in the ML accelerators field, making it difficult to maintain a clear overview of proposed optimization approaches. Additionally, many works combine multiple optimization techniques obfuscating their respective effects and thus complicating a fair comparison. We have surveyed hundreds of research papers in the field and extracted their employed optimization techniques to provide a quantitative summary of existing optimizations and their effects. Each approach is quantified using five key performance indicators, namely memory usage reduction, throughput increase, area reduction, energy/power reduction, as well as accuracy impact. This enables to trade-off improvements and disadvantages across various performance indicators, allowing to estimate the effect of each approach during the design process of new accelerator architectures. The survey describes optimizations with performance improvements ranging from up to 10'000× memory reduction to 33× energy savings.

Edge ML for FPGA-Based Cameras

To reduce the extreme output data rate generated by high-speed cameras, we have investigated edge processing for FPGA-based cameras, decreasing the communication bandwidth by 980× and system energy per frame by 3× through on-board image analysis capabilities. Processing images close to the sensor has been shown to enable new applications like self-triggering, which would not be possible with the communication latency imposed by external processing. Our implementation has shown that high quantization levels down to fully binary precision lead to significant power and memory savings, enabling complete on-chip edge processing.

Hardware Mapping

The computational efficiency of edge processing systems originates from optimizations across various fields, including system control and mapping approaches. Hardware mapping also forms the interface between algorithm and hardware developers, making it especially challenging for FPGA implementations, where the hardware configuration is an additional part of the mapping. Recent advances in high-level synthesis tools have simplified FPGA hardware development, making configurations accessible to software developers with no hardware coding background. To further advocate edge ML for FPGA-based cameras, we have provided an automated mapping framework that translates a trained network into an efficient FPGA hardware implementation. It supports arbitrary layer and kernel dimensions, layer-wise configurable fixed-point precision, and a throughput-balancing mechanism to effectively distribute the available resources across the layers. For fully-binary implementations, resource-efficient XNOR-based arithmetic is instantiated.

We have further investigated the efficient memory mapping of activations in layer-wise CNN processing, where they often dominate the memory due to increasing input (image) sizes. Our proposed mapping technique improves the standard ping-pong buffering by allowing neighboring activation regions to overlap, enabling memory savings of up to 48.8%.

Custom Hierarchical Dual-Engine ML ASIC

Face detection and recognition tasks are increasingly deployed to security and retail customer analysis applications, where battery-powered operation is required. We have presented an energy-efficient system-on-chip that ensures a long battery lifetime for such face analysis tasks. Its highly scalable dual-engine architecture can run simple detections at very low power consumption using a binary decision tree accelerator, which can trigger the more complex CNN accelerator for detailed analyses. It features multiple optimization approaches like multi-precision CNN support, dynamically scalable evaluation complexity as well as end-to-end processing capabilities, enabling it to run face detection and recognition at 0.41mW.

Towards Battery-Less Edge ML

The improving energy efficiency of edge ML accelerators have enabled the deployment of increasingly complex ML applications in domains that have traditionally been restricted to very simple analyses due to stringent battery lifetime requirements. By exploiting the efficient edge processing capabilities of our presented ML ASIC, we have demonstrated the feasibility of implementing battery-less face identification on a new credit card-sized computer vision platform. Its small solar panel enables self-sustainable operation under indoor lighting conditions, running face recognition at 1 frame-per-second.

8.2 Outlook

This section discusses promising directions for future research in the field of efficient NN inference at the edge.

Processing Closer to the (Sensor) Data

While NN inference accelerators have received a lot of attention over the past years, system-level aspects like sensor communications have often been neglected. In Chapter 7 we have shown the power breakdown of our implemented edge ML system, reporting a dominant

power contribution from the sensor communications. This significantly decreases the impact of processing efficiency improvements and thus requests for a system-level analysis to identify effective optimization potentials across the entire system. Early data reduction approaches have shown that processing closer to the sensor can help reducing data movements and related energy costs. Increasing the ratio of processing performed on the sensor side additionally helps reducing the memory size of connected processing chips and allows them to remain in a low-power idle mode for a longer period of time. Sensor-based pre-processing could identify useful data, only triggering the processor for detailed analyses. However, sensors often rely on larger process nodes, making them less suitable for implementing digital processing (low memory density, higher supply voltage). Various chip stacking technologies have been proposed, allowing to tightly connect multiple dies (manufactured in different process technologies), allowing to move the digital processing closer to the data source. Compute-in-memory techniques could further improve this approach, possibly being integrated into the sensor memory for extreme edge processing.

Flexibility for Efficiency

Hardware flexibility is a key factor for supporting a diverse set of applications. It additionally allows adapting the analysis complexity to the application's current needs, which might change over time and enable power savings during low-complexity phases. Various such complexity scaling options have been discussed throughout the thesis, requiring the hardware to support various arithmetic precisions, arbitrary network dimensions, or approximate computing options. Another promising approach is event-based processing, adapting the computational workload to the (temporal) input activity. Constant input data result in reduced switching activities in the processing hardware, leading to lower dynamic power consumption. This enables to flexibly adapt the complexity to temporal information content. Challenges in the training of such algorithms, especially for more complex tasks, are still being investigated in this active field of research.

Co-Design

We have shown that algorithmic network optimizations, like reducing weight precision, enable significant memory and power savings in hardware accelerators. Other algorithmic optimizations, with similar hardware advantages in mind, have been proposed to trade off accuracy against energy efficiency. Such algorithm/hardware co-design is highly application-dependent, making it time-consuming to find the optimum trade-off for each use case.

NAS was recently proposed to find optimal network (and accelerator) architectures for a specific task, evaluating a set of controllable hyper-parameters based on target accuracy and complexity metrics [187, 296]. While some NAS frameworks already include simple (computation) power estimates in their evaluation, detailed hardware implementation models could be included in the search process, further enabling to optimize data reuse, memory utilization, and hierarchical processing options. To enable flexible designs, the search can be extended across multiple operating points with different precision options and power management settings. The extreme hardware simulation and benchmarking complexity currently limits such evaluations, requiring more efficient simulation techniques to become available.

Recent research has identified that currently used networks are generally too large, often overfitting to the training data [185], and thus propose *less artificial intelligence*. Improved training techniques are expected to help networks generalize better beyond the training datasets, which could lead to smaller minimum required network complexities, further improving the efficiency of edge ML applications.

Appendix A

Notations and Acronyms

Operators

$\ll$	much less than
$<$	less than
$\leq$	less than or equal to
$\gg$	much greater than
$>$	greater than
$\geq$	greater than or equal to
$\|\cdot\|$	ℓ^2-norm or Euclidean norm, i.e., $\sqrt{\sum_i^n \|x_i\|^2}$ for $\mathbf{x} \in \mathbb{C}^n$
$\log_2$	base-2 logarithm
$\log_{10}$	base-10 logarithm
$(\cdot)^{-1}$	inverse function
$\lceil\cdot\rceil$	ceil: smallest integer value equal to or larger as argument
$\lfloor\cdot\rfloor$	floor: largest integer value equal to or smaller as argument
$\in$	is member of
$*$	convolution operator

$a+=b$	increment and assign: $a \leftarrow a+b$

Acronyms

ABB	adaptive body biasing
ADC	analog-to-digital converter
ALU	arithmetic logic unit
ASIC	application-specific integrated circuit
BB	bounding box
BDT	binary decision tree
BNN	binary neural network
BRAM	block RAM
CIM	compute-in-memory
CMOS	complementary metal-oxide-semiconductor
CNN	convolutional neural network
CPU	central processing unit
DMA	direct memory access
DNN	deep neural network
DSP	digital signal processor
FC	fully-connected
FD	face detection
FF	flip-flop
FIFO	first-in first-out
FPGA	field-programmable gate array
FPS	frames-per-second
FR	face recognition
FSM	finite-state machine
GE	gate equivalent (reference unit area of a two-input NAND gate)

GOPS	billion operations per second (1 MAC is usually considered as 2 operations)
GPU	graphics processing unit
HDL	hardware description language
HLS	high-level synthesis
IC	integrated circuit
IoT	internet of things
IP	intellectual property
KNN	k-nearest neighbors algorithm
LUT	look-up table
MAC	multiply and accumulate
ML	machine learning
NAS	neural architecture search
NN	neural network
NVM	non-volatile memory
PE	processing element
RAM	random-access memory
ResNet	residual network
RNN	recurrent neural network
SAR	successive approximation register
SoA	state-of-the-art
SoC	system-on-chip
SPI	serial peripheral interface
SRAM	static random-access memory
SVM	support-vector machine

ULP	ultra-low power
USB	universal serial bus
WC	weak classifier

Bibliography

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1026–1034.

[2] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[3] M. Jermyn, J. Desroches, J. Mercier, M.-A. Tremblay, K. St-Arnaud, M.-C. Guiot, K. Petrecca, and F. Leblond, "Neural networks improve brain cancer detection with raman spectroscopy in the presence of operating room light artifacts," *Journal of biomedical optics*, vol. 21, no. 9, p. 94002, September 2016.

[4] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv*, vol. arXiv:1711.07128, 2017. [Online]. Available: http://arxiv.org/abs/1711.07128

[5] M. Wang and W. Deng, "Deep face recognition: A survey," *arXiv*, vol. arXiv:1804.06655, 2018. [Online]. Available: http://arxiv.org/abs/1804.06655

[6] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[7] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[8] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv*, vol. arXiv:1804.03209, 2018. [Online]. Available: http://arxiv.org/abs/1804.03209

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 675–678. [Online]. Available: https://doi.org/10.1145/2647868.2654889

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[11] G. V. Research, "Artificial intelligence market size, share & trends analysis report," 2021, (Accessed 16.06.2021). [Online]. Available: https://www.grandviewresearch.com/industry-analysis/artificial-intelligence-ai-market

[12] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[13] G. Nagy, "Neural networks-then and now," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 316–318, 1991.

[14] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *arXiv*, vol. arXiv:1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[15] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *arXiv*, vol. arXiv:1808.03314, 2018. [Online]. Available: http://arxiv.org/abs/1808.03314

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv*, vol. arXiv:1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[17] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X13000241

[18] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[19] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*. Synthesis Lectures on Computer Architecture, 2020.

[20] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey of machine learning accelerators," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–12.

[21] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225 134–225 180, 2020.

[22] Orcam, "Orcam myeye 2 specifications," 2020, (Accessed 09.02.2021). [Online]. Available: https://www.orcam.com/en/myeye2/specification/

[23] Xiaomi, "Xiaomo ai door bell overview," 2019, (Accessed 09.02.2021). [Online]. Available: https://www.xiaomitoday.com/2019/10/29/xiaomi-xiaomo-mijia-ai-face-identifcation-1080p-door-bell/

[24] S. Oh, M. Cho, X. Wu, Y. Kim, L. Chuo, W. Lim, P. Pannuto, S. Bang, K. Yang, H. Kim, D. Sylvester, and D. Blaauw, "Iot2 — the internet of tiny things: Realizing mm-scale sensors through 3d die stacking," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 686–691.

[25] Cisco, "Visual networking index (vni) complete forecast," 2016, (Accessed 09.02.2021). [Online]. Available: https://www.cisco.com/c/dam/m/en_us/solutions/service-provider/vni-forecast-highlights/pdf/Global_2020_Forecast_Highlights.pdf

[26] P. Jokic, E. Azarkhish, R. Cattenoz, E. Turetken, L. Benini, and S. Emery, "A sub-mw dual-engine ml inference system-on-chip for complete end-to-end face-analysis at the edge," in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2.

[27] P. Jokic, S. Emery, and L. Benini, "Battery-less face recognition at the extreme edge," in *19th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2021.

[28] P. Jokic, S. Emery, and L. Benini, "Improving memory utilization in convolutional neural network accelerators," *IEEE Embedded Systems Letters*, pp. 1–1, 2020.

[29] P. Jokic, E. Azarkhish, A. Bonetti, M. Pons, S. Emery, and L. Benini, "A construction kit for efficient low power neural network accelerator designs," *arXiv*, vol. arXiv:2106.12810, 2021.

[30] P. Jokic, S. Emery, and L. 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 International Symposium on Industrial Embedded Systems (SIES)*, June 2018.

[31] P. Jokic, S. Emery, and L. Benini, "NN2CAM: Automated neural network mapping for multi-precision edge processing on fpga-based cameras," *arXiv*, vol. arXiv:2106.12840, 2021.

[32] P. Jokic and M. Magno, "Powering smart wearable systems with flexible solar energy harvesting," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[33] P. Jokic, G. Salvatore, , M. Magno, L. Buthe, G. Troster, and L. Benini, "Self-sustainable smart ring for long term monitoring of blood oxygenation," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[34] M. Magno, G. A. Salvatore, P. Jokic, and L. Benini, "Self-sustainable smart ring for long-term monitoring of blood oxygenation," *IEEE Access*, vol. 7, pp. 115 400–115 408, 2019.

[35] G. A. Salvatore, J. Sülzle, F. Dalla Valle, G. Cantarella, F. Robotti, P. Jokic, S. Knobelspies, A. Daus, L. Büthe, L. Petti, N. Kirchgessner, R. Hopf, M. Magno, and G. Tröster, "Biodegradable and highly deformable temperature sensors for the internet of things," *Advanced Functional Materials*, vol. 27, no. 35, p. 1702390, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/adfm.201702390

[36] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[38] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *arXiv*, vol. arXiv:1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[39] I. Hong, K. Bong, D. Shin, S. Park, K. Jason Lee, Y. Kim, and H. Yoo, "A 2.71 nj/pixel gaze-activated object recognition system for low-power mobile smart glasses," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 45–55, 2016.

[40] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong, "Draining our glass: An energy and heat characterization of google glass," in *Proceedings of 5th Asia-Pacific Workshop on Systems*, ser. APSys '14. New York, NY, USA: Association for Computing Machinery, 2014. [Online]. Available: https://doi.org/10.1145/2637166.2637230

[41] S. Park, S. Choi, J. Lee, M. Kim, J. Park, and H. Yoo, "14.1 a 126.1mw real-time natural ui/ux processor with embedded deep-learning core for low-power smart glasses," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 254–255.

[42] Google, "Google clips specifications," 2017, (Accessed 21.05.2019). [Online]. Available: https://support.google.com/googleclips/answer/7545447?hl=en

[43] S. Alyamkin, M. Ardi, A. C. Berg, A. Brighton, B. Chen, Y. Chen, H. P. Cheng, Z. Fan, C. Feng, B. Fu, K. Gauen, A. Goel, A. Goncharenko, X. Guo, S. Ha, A. Howard, X. Hu, Y. Huang, D. Kang, J. Kim, J. G. Ko, A. Kondratyev, J. Lee, S. Lee, S. Lee, Z. Li, Z. Liang, J. Liu, X. Liu, Y. Lu, Y. H. Lu, D. Malik, H. H. Nguyen, E. Park, D. Repin, L. Shen, T. Sheng, F. Sun, D. Svitov, G. K. Thiruvathukal, B. Zhang, J. Zhang, X. Zhang, and S. Zhuo, "Low-power computer vision: Status, challenges, and opportunities," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 411–421, 2019.

[44] M. E. Ilas and C. Ilas, "Towards real-time and real-life image classification and detection using cnn: a review of practical

applications requirements, algorithms, hardware and current trends," in *2020 IEEE 26th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, 2020, pp. 225–233.

[45] J. S. P. Giraldo, S. Lauwereins, K. Badami, H. Van Hamme, and M. Verhelst, "18uw soc for near-microphone keyword spotting and speaker verification," in *2019 Symposium on VLSI Circuits*, 2019, pp. C52–C53.

[46] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for google's training chips: Tpuv2 and tpuv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, 2021.

[47] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the graphcore IPU architecture via microbenchmarking," *arXiv*, vol. arXiv:1912.03413, 2019. [Online]. Available: http://arxiv.org/abs/1912.03413

[48] K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, S. Han, Y. Xie, P. Debacker, M. Verhelst, and Y. Wang, "Neural network accelerator comparison," 2020, (Accessed 22.03.2021). [Online]. Available: http://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/

[49] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–9.

[50] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv*, vol. arXiv:1705.06963, 2017. [Online]. Available: http://arxiv.org/abs/1705.06963

[51] V. Sze, Y. Chen, T. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[52] V. Sze, Y. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, 2017, pp. 1–8.

[53] S. Bodiwala and N. Nanavati, "Efficient hardware implementations of deep neural networks: A survey," in *2020 Fourth International Conference on Inventive Systems and Control (ICISC)*, 2020, pp. 31–36.

[54] S. Mittal, "A survey of fpga-based accelerators for convolutional neural networks," *Neural Computing and Applications*, 2018.

[55] V. Gadepally, J. Goodwin, J. Kepner, A. Reuther, H. Reynolds, S. Samsi, J. Su, and D. Martinez, "AI enabling technologies: A survey," *arXiv*, vol. arXiv:1905.03592, 2019. [Online]. Available: http://arxiv.org/abs/1905.03592

[56] J. Tang, F. Yuan, X. Shen, Z. Wang, M. Rao, Y. He, Y. Sun, X. Li, W. Zhang, Y. Li, B. Gao, H. Qian, G. Bi, S. Song, J. J. Yang, and H. Wu, "Bridging biological and artificial neural networks with emerging neuromorphic devices: Fundamentals, progress, and challenges," *Advanced Materials*, vol. 31, no. 49, p. 1902761, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/adma.201902761

[57] K. J. Lee, J. Lee, S. Choi, and H. J. Yoo, "The development of silicon for ai: Different design approaches," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4719–4732, 2020.

[58] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering*, vol. 6, no. 3, pp. 264–274, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809919306356

[59] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Architectures, challenges, and applications," *arXiv*, vol. arXiv:2003.12172, 2020. [Online]. Available: http://arxiv.org/abs/2003.12172

[60] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104.

[61] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 243–254.

[62] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 246–247.

[63] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[64] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Yodann: An architecture for ultralow power binary-weight cnn acceleration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 48–60, 2018.

[65] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "Unpu: An energy-efficient deep neural network accelerator with fully variable weight bit precision," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 173–185, 2019.

[66] Y. Chen, T. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[67] P. K. D. Pramanik, N. Sinhababu, B. Mukherjee, S. Padmanaban, A. Maity, B. K. Upadhyaya, J. B. Holm-Nielsen,

and P. Choudhury, "Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage," *IEEE Access*, vol. 7, pp. 182 113–182 172, 2019.

[68] I. Miro-Panades, B. Tain, J. F. Christmann, D. Coriat, R. Lemaire, C. Jany, B. Martineau, F. Chaix, A. Quelen, E. Pluchart, J. P. Noel, R. Boumchedda, A. Makosiej, M. Montoya, S. Bacles-Min, D. Briand, J. M. Philippe, A. Valentian, F. Heitzmann, E. Beigne, and F. Clermidy, "Samurai: A 1.7mops-36gops adaptive versatile iot node with 15,000× peak-to-idle power reduction, 207ns wake-up time and 1.3tops/w ml efficiency," in *2020 IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[69] J.-H. Kim, C. Kim, K. Kim, and H.-J. Yoo, "An ultra-low-power analog-digital hybrid cnn face recognition processor integrated with a cis for always-on mobile devices," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1–5.

[70] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *arXiv*, vol. arXiv:1604.03168, 2016. [Online]. Available: http://arxiv.org/abs/1604.03168

[71] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," *arXiv*, vol. arXiv:2103.13630, 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[72] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. V. Gool, "AI benchmark: All about deep learning on smartphones in 2019," *arXiv*, vol. arXiv:1910.06663, 2019. [Online]. Available: http://arxiv.org/abs/1910.06663

[73] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B.

Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," *arXiv*, vol. arXiv:1911.02549, 2019. [Online]. Available: http://arxiv.org/abs/1911.02549

[74] C. R. Banbury, V. Janapa Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J.-s. Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, "Benchmarking TinyML Systems: Challenges and Direction," *arXiv e-prints*, p. arXiv:2003.04821, 2020.

[75] EEMBC, "Exploring coremark – a benchmark maximizing simplicity and efficacy," 2009, (Accessed 03.02.2021). [Online]. Available: https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf

[76] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[77] J. von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[78] B. Silc, Jurij and Robic and T. Ungerer, *Multiple-Issue Processors*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 123–219. [Online]. Available: https://doi.org/10.1007/978-3-642-58589-0_4

[79] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan,

H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[80] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 367–379.

[81] Z. Liu, P. N. Whatmough, and M. Mattina, "Systolic tensor array: An efficient structured-sparse gemm accelerator for mobile cnn inference," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 34–37, 2020.

[82] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic CNN accelerator," *arXiv*, vol. arXiv:1811.02883, 2018. [Online]. Available: http://arxiv.org/abs/1811.02883

[83] T.-J. Yang and V. Sze, "Design considerations for efficient deep neural networks on processing-in-memory accelerators," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 22.1.1–22.1.4.

[84] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162. [Online]. Available: https://doi.org/10.1145/977091.977115

[85] S. Yu, X. Sun, X. Peng, and S. Huang, "Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*, 2020, pp. 1–4.

[86] X. Si, J. Chen, Y. Tu, W. Huang, J. Wang, Y. Chiu, W. Wei, S. Wu, X. Sun, R. Liu, S. Yu, R. Liu, C. Hsieh, K. Tang, Q. Li, and M. Chang, "24.5 a twin-8t sram computation-in-memory macro for multiple-bit cnn-based machine learning," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 396–398.

[87] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017.

[88] M. Kang, S. Lim, S. Gonugondla, and N. R. Shanbhag, "An in-memory vlsi architecture for convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 494–505, 2018.

[89] J.-W. Su, Y.-C. Chou, R. Liu, T.-W. Liu, P.-J. Lu, P.-C. Wu, Y.-L. Chung, L.-Y. Hung, J.-S. Ren, T. Pan, S.-H. Li, S.-C. Chang, S.-S. Sheu, W.-C. Lo, C.-I. Wu, X. Si, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, and M.-F. Chang, "16.3 a 28nm 384kb 6t-sram computation-in-memory macro with 8b precision for ai edge chips," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 250–252.

[90] S. Yu, "Neuro-inspired computing with emerging nonvolatile memorys," *Proceedings of the IEEE*, vol. 106, no. 2, pp. 260–285, 2018.

[91] F. Cai, J. Correll, S. H. Lee, Y. Lim, V. Bothra, Z. Zhang, M. Flynn, and W. Lu, "A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations," *Nature Electronics*, vol. 2, p. 1, 07 2019.

[92] S. Dash, Y. Luo, A. Lu, S. Yu, and S. Mukhopadhyay, "Robust processing-in-memory with multi-bit reram using hessian-driven mixed-precision computation," *IEEE Transactions on*

Computer-Aided Design of Integrated Circuits and Systems, pp. 1–1, 2021.

[93] C. Xue, W. Chen, J. Liu, J. Li, W. Lin, W. Lin, J. Wang, W. Wei, T. Chang, T. Chang, T. Huang, H. Kao, S. Wei, Y. Chiu, C. Lee, C. Lo, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang, "24.1 a 1mb multibit reram computing-in-memory macro with 14.6ns parallel mac computing time for cnn based ai edge processors," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2019, pp. 388–390.

[94] S. Yin, Y. Kim, X. Han, H. Barnaby, S. Yu, Y. Luo, W. He, X. Sun, J. Kim, and J. Seo, "Monolithically integrated rram- and cmos-based in-memory computing optimizations for efficient deep learning," *IEEE Micro*, vol. 39, no. 6, pp. 54–63, 2019.

[95] G. Murali, X. Sun, S. Yu, and S. K. Lim, "Heterogeneous mixed-signal monolithic 3-d in-memory computing using resistive ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 2, pp. 386–396, 2021.

[96] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "Dnn+neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 32.5.1–32.5.4.

[97] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo, "Mcdram v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge," *IEEE Access*, vol. 8, pp. 135 223–135 243, 2020.

[98] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff." *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, 2006.

[99] R. S. Williams, "What's next? [the end of moore's law]," *Computing in Science Engineering*, vol. 19, no. 2, pp. 7–13, 2017.

[100] M. Bohr, "A 30 year retrospective on dennard's mosfet scaling paper," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 11–13, 2007.

[101] P. Bose, *Power Wall*. Boston, MA: Springer US, 2011, pp. 1593–1608. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_499

[102] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.

[103] Y. Lee, S. Bang, I. Lee, Y. Kim, G. Kim, M. H. Ghaed, P. Pannuto, P. Dutta, D. Sylvester, and D. Blaauw, "A modular 1 mm^3 die-stacked sensing platform with low power i^2c inter-die communication and multi-modal energy harvesting," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 229–243, 2013.

[104] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, J. Kadomoto, T. Miyata, M. Hamada, T. Kuroda, and M. Motomura, "Quest: A 7.49tops multi-purpose log-quantized dnn inference engine stacked on 96mb 3d sram using inductive-coupling technology in 40nm cmos," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 216–218.

[105] IRDS, "International roadmap for devices and systems: 2020 update," 2020, (Accessed 11.05.2021). [Online]. Available: https://irds.ieee.org/editions/2020

[106] C. Auth, C. Allen, A. Blattner, D. Bergstrom, M. Brazier, M. Bost, M. Buehler, V. Chikarmane, T. Ghani, T. Glassman, R. Grover, W. Han, D. Hanken, M. Hattendorf, P. Hentges, R. Heussner, J. Hicks, D. Ingerly, P. Jain, S. Jaloviar, R. James, D. Jones, J. Jopling, S. Joshi, C. Kenyon, H. Liu, R. McFadden, B. McIntyre, J. Neirynck, C. Parker, L. Pipes, I. Post, S. Pradhan, M. Prince, S. Ramey, T. Reynolds, J. Roesler, J. Sandford, J. Seiple, P. Smith, C. Thomas, D. Towner, T. Troeger, C. Weber, P. Yashar, K. Zawadzki, and K. Mistry, "A 22nm high performance and low-power cmos technology

featuring fully-depleted tri-gate transistors, self-aligned contacts and high density mim capacitors," in *2012 Symposium on VLSI Technology (VLSIT)*, 2012, pp. 131–132.

[107] H. N. Patel, A. Roy, F. B. Yahya, N. Liu, B. Calhoun, K. Kumeno, M. Yasuda, A. Harada, and T. Ema, "A 55nm ultra low leakage deeply depleted channel technology optimized for energy minimization in subthreshold sram and logic," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 45–48.

[108] S. Natarajan, M. Agostinelli, S. Akbar, M. Bost, A. Bowonder, V. Chikarmane, S. Chouksey, A. Dasgupta, K. Fischer, Q. Fu, T. Ghani, M. Giles, S. Govindaraju, R. Grover, W. Han, D. Hanken, E. Haralson, M. Haran, M. Heckscher, R. Heussner, P. Jain, R. James, R. Jhaveri, I. Jin, H. Kam, E. Karl, C. Kenyon, M. Liu, Y. Luo, R. Mehandru, S. Morarka, L. Neiberg, P. Packan, A. Paliwal, C. Parker, P. Patel, R. Patel, C. Pelto, L. Pipes, P. Plekhanov, M. Prince, S. Rajamani, J. Sandford, B. Sell, S. Sivakumar, P. Smith, B. Song, K. Tone, T. Troeger, J. Wiedemer, M. Yang, and K. Zhang, "A 14nm logic technology featuring 2nd-generation finfet, air-gapped interconnects, self-aligned double patterning and a 0.0588 μm^2 sram cell size," in *2014 IEEE International Electron Devices Meeting*, 2014, pp. 3.7.1–3.7.3.

[109] O. Weber, "FDSOI vs FinFET: differentiating device features for ultra low power iot applications," in *2017 IEEE International Conference on IC Design and Technology (ICICDT)*, 2017, pp. 1–3.

[110] R. Carter, J. Mazurier, L. Pirro, J. U. Sachse, P. Baars, J. Faul, C. Grass, G. Grasshoff, P. Javorka, T. Kammler, A. Preusse, S. Nielsen, T. Heller, J. Schmidt, H. Niebojewski, P. Y. Chou, E. Smith, E. Erben, C. Metze, C. Bao, Y. Andee, I. Aydin, S. Morvan, J. Bernard, E. Bourjot, T. Feudel, D. Harame, R. Nelluri, H. . J. Thees, L. M-Meskamp, J. Kluth, R. Mulfinger, M. Rashed, R. Taylor, C. Weintraub, J. Hoentschel, M. Vinet, J. Schaeffer, and B. Rice, "22nm fdsoi technology for emerging

mobile, internet-of-things, and rf applications," in *2016 IEEE International Electron Devices Meeting (IEDM)*, 2016, pp. 2.2.1–2.2.4.

[111] J.-H. Lee, J.-W. Lee, H.-A.-R. Jung, and B.-K. Choi, "Comparison of SOI FinFETs and bulk FinFETs," *ECS Transactions*, vol. 19, no. 4, pp. 101–112, 2009. [Online]. Available: https://doi.org/10.1149/1.3117397

[112] M. Pons, J. Nagel, D. Séverac, M. Morgan, D. Sigg, P. Rüedi, and C. Piguet, "Ultra low-power standard cell design using planar bulk cmos in subthreshold operation," in *2013 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2013, pp. 9–15.

[113] M. Pons, T. Le, C. Arm, D. Séverac, S. Emery, and C. Piguet, "A 1kb single-side read 6t sub-threshold sram in 180 nm with 530 hz frequency 3.1 na total current and 2.4 na leakage at 0.27 v," in *2015 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2015, pp. 1–2.

[114] M. Pons, T. Le, C. Arm, D. Séverac, J. Nagel, M. Morgan, and S. Emery, "Sub-threshold latch-based icyflex2 32-bit processor with wide supply range operation," in *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, 2016, pp. 41–44.

[115] M. Pons, C. T. Müller, D. Ruffieux, J. Nagel, S. Emery, A. Burg, S. Tanahashi, Y. Tanaka, and A. Takeuchi, "A 0.5 v 2.5 uw/mhz microcontroller with analog-assisted adaptive body bias pvt compensation with 3.13nw/kb sram retention in 55nm deeply-depleted channel cmos," in *2019 IEEE Custom Integrated Circuits Conference (CICC)*, 2019, pp. 1–4.

[116] T. C. Müller, J. Nagel, M. Pons, D. Séverac, K. Hashiba, S. Sawada, K. Miyatake, S. Emery, and A. Burg, "Pvt compensation in mie fujitsu 55 nm ddc: A standard-cell library based comparison," in *2017 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2017, pp. 1–2.

[117] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94. USA: USENIX Association, 1994, p. 2–es.

[118] T. D. Burd and R. W. Brodersen, *Processor Design for Portable Systems*. USA: Kluwer Academic Publishers, 1996, p. 203–221.

[119] B. Moons and M. Verhelst, "Dvas: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2015, pp. 237–242.

[120] X. Yang, J. Pu, B. B. Rister, N. Bhagdikar, S. Richardson, S. Kvatinsky, J. Ragan-Kelley, A. Pedram, and M. Horowitz, "A systematic approach to blocking convolutional neural networks," *arXiv*, vol. arXiv:1606.04209, 2016. [Online]. Available: http://arxiv.org/abs/1606.04209

[121] A. Chen, "A review of emerging non-volatile memory (nvm) technologies and applications," *Solid-State Electronics*, vol. 125, pp. 25–38, 2016, extended papers selected from ESSDERC 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0038110116300867

[122] J. Park, "Neuromorphic computing using emerging synaptic devices: A retrospective summary and an outlook," *Electronics*, vol. 9, no. 9, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/9/1414

[123] J. Y. Wu, Y. S. Chen, W. S. Khwa, S. M. Yu, T. Y. Wang, J. C. Tseng, Y. D. Chih, and C. H. Diaz, "A 40nm low-power logic compatible phase change memory technology," in *2018 IEEE International Electron Devices Meeting (IEDM)*, 2018, pp. 27.6.1–27.6.4.

[124] K. Lee, J. H. Bak, Y. J. Kim, C. K. Kim, A. Antonyan, D. H. Chang, S. H. Hwang, G. W. Lee, N. Y. Ji, W. J. Kim, J. H. Lee, B. J. Bae, J. H. Park, I. H. Kim, B. Y. Seo, S. H. Han, Y. Ji, H. T. Jung, S. O. Park, O. I. Kwon, J. W. Kye, Y. D. Kim, S. W.

Pae, Y. J. Song, G. T. Jeong, K. H. Hwang, G. H. Koh, H. K. Kang, and E. S. Jung, "1gbit high density embedded stt-mram in 28nm fdsoi technology," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 2.2.1–2.2.4.

[125] L. Wei, J. G. Alzate, U. Arslan, J. Brockman, N. Das, K. Fischer, T. Ghani, O. Golonzka, P. Hentges, R. Jahan, P. Jain, B. Lin, M. Meterelliyoz, J. O'Donnell, C. Puls, P. Quintero, T. Sahu, M. Sekhar, A. Vangapaty, C. Wiegand, and F. Hamzaoglu, "13.3 a 7mb stt-mram in 22ffl finfet technology with 4ns read sensing time at 0.9v using write-verify-write scheme and offset-cancellation sensing technique," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 214–216.

[126] Y. J. Song, J. H. Lee, S. H. Han, H. C. Shin, K. H. Lee, K. Suh, D. E. Jeong, G. H. Koh, S. C. Oh, J. H. Park, S. O. Park, B. J. Bae, O. I. Kwon, K. H. Hwang, B. Y. Seo, Y. K. Lee, S. H. Hwang, D. S. Lee, Y. Ji, K. C. Park, G. T. Jeong, H. S. Hong, K. P. Lee, H. K. Kang, and E. S. Jung, "Demonstration of highly manufacturable stt-mram embedded in 28nm logic," in *2018 IEEE International Electron Devices Meeting (IEDM)*, 2018, pp. 18.2.1–18.2.4.

[127] T. Liu, T. H. Yan, R. Scheuerlein, Y. Chen, J. K. Lee, G. Balakrishnan, G. Yee, H. Zhang, A. Yap, J. Ouyang, T. Sasaki, S. Addepalli, A. Al-Shamma, C. Chen, M. Gupta, G. Hilton, S. Joshi, A. Kathuria, V. Lai, D. Masiwal, M. Matsumoto, A. Nigam, A. Pai, J. Pakhale, C. H. Siau, X. Wu, R. Yin, L. Peng, J. Y. Kang, S. Huynh, H. Wang, N. Nagel, Y. Tanaka, M. Higashitani, T. Minvielle, C. Gorla, T. Tsukamoto, T. Yamaguchi, M. Okajima, T. Okamura, S. Takase, T. Hara, H. Inoue, L. Fasoli, M. Mofidi, R. Shrivastava, and K. Quader, "A 130.7mm^2 2-layer 32gb reram memory device in 24nm technology," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2013, pp. 210–211.

[128] C. Chou, Z. Lin, P. Tseng, C. Li, C. Chang, W. Chen, Y. Chih, and T. J. Chang, "An n40 256k×44 embedded rram macro with

sl-precharge sa and low-voltage current limiter to improve read and write performance," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 478–480.

[129] P. Jain, U. Arslan, M. Sekhar, B. C. Lin, L. Wei, T. Sahu, J. Alzate-vinasco, A. Vangapaty, M. Meterelliyoz, N. Strutt, A. B. Chen, P. Hentges, P. A. Quintero, C. Connor, O. Golonzka, K. Fischer, and F. Hamzaoglu, "13.2 a 3.6mb 10.1mb/mm^2 embedded non-volatile reram macro in 22nm finfet technology with adaptive forming/set/reset schemes yielding down to 0.5v with sensing time of 5ns at 0.7v," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2019, pp. 212–214.

[130] J. Yang, X. Xue, X. Xu, Q. Wang, H. Jiang, J. Yu, D. Dong, F. Zhang, H. Lv, and M. Liu, "24.2 a 14nm-finfet 1mb embedded 1t1r rram with a 0.022µ m2 cell size using self-adaptive delayed termination and multi-cell reference," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 336–338.

[131] S. Beyer, S. Dünkel, M. Trentzsch, J. Müller, A. Hellmich, D. Utess, J. Paul, D. Kleimaier, J. Pellerin, S. Müller, J. Ocker, A. Benoist, H. Zhou, M. Mennenga, M. Schuster, F. Tassan, M. Noack, A. Pourkeramati, F. Müller, M. Lederer, T. Ali, R. Hoffmann, T. Kämpfe, K. Seidel, H. Mulaosmanovic, E. T. Breyer, T. Mikolajick, and S. Slesazeck, "Fefet: A versatile cmos compatible device with game-changing potential," in *2020 IEEE International Memory Workshop (IMW)*, 2020, pp. 1–4.

[132] D. Reis, K. Ni, W. Chakraborty, X. Yin, M. Trentzsch, S. D. Dünkel, T. Melde, J. Müller, S. Beyer, S. Datta, M. T. Niemier, and X. S. Hu, "Design and analysis of an ultra-dense, low-leakage, and fast fefet-based random access memory array," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, vol. 5, no. 2, pp. 103–112, 2019.

[133] S. Bang, J. Wang, Z. Li, C. Gao, Y. Kim, Q. Dong, Y. P. Chen, L. Fick, X. Sun, R. Dreslinski, T. Mudge, H. S. Kim, D. Blaauw, and D. Sylvester, "14.7 a 288µw programmable

deep-learning processor with 270kb on-chip weight storage using non-uniform memory hierarchy for mobile intelligence," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 250–251.

[134] J. Lee, Y. Kim, M. Cho, M. Yasuda, S. Miyoshi, M. Kawaminami, D. Blaauw, and D. Sylvester, "Aμprocessor layer for mm-scale die-stacked sensing platforms featuring ultra-low power sleep mode at 125°c," in *2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2020, pp. 1–4.

[135] K. Bong, S. Choi, C. Kim, D. Han, and H. J. Yoo, "A low-power convolutional neural network face recognition processor and a cis integrated with always-on face detector," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 1, pp. 115–123, Jan 2018.

[136] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, area, and performance optimization of standard cell memory arrays through controlled placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 4, May 2016. [Online]. Available: https://doi.org/10.1145/2890498

[137] F. Conti, P. Davide Schiavone, and L. Benini, "XNOR Neural Engine: a Hardware Accelerator IP for 21.6 fJ/op Binary Neural Network Inference," *ArXiv*, vol. arXiv:1807.03010, 2018.

[138] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," in *2012 Symposium on VLSI Technology (VLSIT)*, 2012, pp. 87–88.

[139] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 751–764, 2017. [Online]. Available: https://doi.org/10.1145/3093337.3037702

[140] R. Giterman, A. Fish, A. Burg, and A. Teman, "A 4-transistor nmos-only logic-compatible gain-cell embedded dram with over 1.6-ms retention time at 700 mv in 28-nm fd-soi," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 4, pp. 1245–1256, 2018.

[141] R. Giterman, A. Teman, and A. Fish, "A 14.3pw sub-threshold 2t gain-cell edram for ultra-low power iot applications in 28nm fd-soi," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2018, pp. 1–2.

[142] P. Pavan, R. Bez, P. Olivo, and E. Zanoni, "Flash memory cells-an overview," *Proceedings of the IEEE*, vol. 85, no. 8, pp. 1248–1271, 1997.

[143] Cypress, "Sonos flash technology," 2019, (Accessed 21.03.2021). [Online]. Available: https://www.cypress.com/file/123341/download

[144] K. Goetschalckx and M. Verhelst, "Breaking high-resolution cnn bandwidth barriers with enhanced depth-first execution," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 323–331, 2019.

[145] S. Colleman and M. Verhelst, "High-utilization, high-flexibility depth-first cnn coprocessor for image pixel processing on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 461–471, 2021.

[146] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer cnn accelerators," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[147] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini, "Cutie: Beyond petaop/s/w ternary dnn inference acceleration with better-than-binary energy efficiency," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2021.

[148] A. Stoutchinin, F. Conti, and L. Benini, "Optimally scheduling CNN convolutions for efficient memory access," *arXiv*, vol. arXiv:1902.01492, 2019. [Online]. Available: http://arxiv.org/abs/1902.01492

[149] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos, "Memory requirements for convolutional neural network hardware accelerators," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 111–121.

[150] S. Zheng, X. Zhang, D. Ou, S. Tang, L. Liu, S. Wei, and S. Yin, "Efficient scheduling of irregular network structures on cnn accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3408–3419, 2020.

[151] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: efficient neural network kernels for arm cortex-m cpus," *arXiv*, vol. arXiv:1801.06601, 2018. [Online]. Available: http://arxiv.org/abs/1801.06601

[152] E. De Giovanni, F. Montagna, B. W. Denkinger, S. Machetti, M. Peón-Quirós, S. Benatti, D. Rossi, L. Benini, and D. Atienza, "Modular design and optimization of biomedical applications for ultralow power heterogeneous platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3821–3832, 2020.

[153] H. G. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. C. Molnar, "ASP vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels," *arXiv*, vol. arXiv:1605.03621, 2016. [Online]. Available: http://arxiv.org/abs/1605.03621

[154] P. Pad, S. Narduzzi, C. Kundig, E. Turetken, S. A. Bigdeli, and L. A. Dunbar, "Efficient neural vision systems based on convolutional image acquisition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[155] Z. Wang, J. Zhang, and N. Verma, "Realizing low-energy classification systems by implementing matrix multiplication directly within an ADC," *IEEE Trans. Biomed. Circuits*

Syst., vol. 9, no. 6, pp. 825–837, 2015. [Online]. Available: https://doi.org/10.1109/TBCAS.2015.2500101

[156] R. LiKamWa, Y. Hou, Y. Gao, M. Polansky, and L. Zhong, "Redeye: Analog convnet image sensor architecture for continuous mobile vision," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 255–266.

[157] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 328–339.

[158] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[159] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," *arXiv*, vol. arXiv:1709.01686, 2017. [Online]. Available: http://arxiv.org/abs/1709.01686

[160] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 475–480.

[161] P. P. Bernardo, C. Gerum, A. Frischknecht, K. Lübeck, and O. Bringmann, "Ultratrail: A configurable ultralow-power tc-resnet ai accelerator for efficient keyword spotting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4240–4251, 2020.

[162] L. Liu and J. Deng, "Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution," *arXiv*, vol. arXiv:1701.00299, 2017. [Online]. Available: http://arxiv.org/abs/1701.00299

[163] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2744769.2744904

[164] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR*, 2016. [Online]. Available: http://arxiv.org/abs/1510.00149

[165] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 857–870, 2020.

[166] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks." in *CVPR*. IEEE Computer Society, 2016, pp. 4013–4021. [Online]. Available: http://dblp.uni-trier.de/db/conf/cvpr/cvpr2016.html#LavinG16

[167] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through ffts," in *2nd International Conference on Learning Representations, ICLR*, 2014. [Online]. Available: http://arxiv.org/abs/1312.5851

[168] Y. Zhang and X. Li, "Fast convolutional neural networks with fine-grained ffts," in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 255–265. [Online]. Available: https://doi.org/10.1145/3410463.3414642

[169] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *Artificial Neural Networks and Machine Learning – ICANN 2014*, S. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, and

A. E. P. Villa, Eds. Cham: Springer International Publishing, 2014, pp. 281–290.

[170] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *NIPS*, 1989, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage

[171] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks," *arXiv e-prints*, p. arXiv:2102.00554, 2021.

[172] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2498–2507.

[173] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, p. 1997–2017, 2019.

[174] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6655–6659.

[175] B. A. Plummer, N. Dryden, J. Frost, T. Hoefler, and K. Saenko, "Shapeshifter networks: Decoupling layers from parameters for scalable and effective deep learning," *arXiv*, vol. arXiv:2006.10598v1, 2020.

[176] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[177] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, vol. 28. Curran Associates, Inc., 2015. [Online].

Available: https://proceedings.neurips.cc/paper/2015/file/ae0eb3eed39d2bcef4622b2499a05fe6-Paper.pdf

[178] T. Yang, Y. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6071–6079.

[179] A. Aimar, H. Mostafa, E. Calabrese, A. Rios-Navarro, R. Tapiador-Morales, I. Lungu, M. B. Milde, F. Corradi, A. Linares-Barranco, S. Liu, and T. Delbruck, "Nullhop: A flexible convolutional neural network accelerator based on sparse representations of feature maps," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 3, pp. 644–656, 2019.

[180] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 1–13.

[181] L. Cavigelli, P. Degen, and L. Benini, "Cbinfer: Change-based inference for convolutional neural networks on video data," *arXiv*, vol. arXiv:1704.04313, 2017. [Online]. Available: http://arxiv.org/abs/1704.04313

[182] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 696–701.

[183] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *SIGARCH Comput. Archit. News*, vol. 42, no. 1, p. 269–284, 2014. [Online]. Available: https://doi.org/10.1145/2654822.2541967

[184] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "Hyperdrive: A multi-chip systolically scalable binary-weight cnn inference

engine," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 309–322, 2019.

[185] F. H. Sinz, X. Pitkow, J. Reimer, M. Bethge, and A. S. Tolias, "Engineering a less artificial intelligence," *Neuron*, vol. 103, no. 6, pp. 967–979, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627319307408

[186] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *arXiv*, vol. arXiv:1602.07360, 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[187] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 389–400. [Online]. Available: https://doi.org/10.1145/3210240.3210337

[188] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://arxiv.org/pdf/1908.09791.pdf

[189] M. Alioto, V. De, and A. Marongiu, "Energy-quality scalable integrated circuits and systems: Continuing energy scaling in the twilight of moore's law," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 653–678, 2018.

[190] J. H. Teo, S. Cheng, and M. Alioto, "Low-energy voice activity detection via energy-quality scaling from data conversion to machine learning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1378–1388, 2020.

[191] A. Alvarez, G. Ponnusamy, and M. Alioto, "Energy-quality scalable memory-frugal feature extraction for always-on deep

sub-mw distributed vision," *IEEE Access*, vol. 8, pp. 18 951–18 961, 2020.

[192] P. Yin, J. Lyu, S. Zhang, S. J. Osher, Y. Qi, and J. Xin, "Understanding straight-through estimator in training activation quantized neural nets," *arXiv*, vol. arXiv:1903.05662, 2019. [Online]. Available: http://arxiv.org/abs/1903.05662

[193] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," *arXiv*, vol. arXiv:1511.06488, 2015. [Online]. Available: http://arxiv.org/abs/1511.06488

[194] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," *arXiv*, vol. arXiv:1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[195] M. Gao, Q. Wang, A. S. K. Nagendra, and G. Qu, "A novel data format for approximate arithmetic computing," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 390–395.

[196] D. Shin, J. Lee, J. Lee, J. Lee, and H. Yoo, "Dnpu: An energy-efficient deep-learning processor with heterogeneous multi-core architecture," *IEEE Micro*, vol. 38, no. 5, pp. 85–93, 2018.

[197] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 697–711, 2019.

[198] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ser. ICML'16, 2016, p. 2849–2858.

[199] A. K. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv*, vol. arXiv:1711.05852, 2017. [Online]. Available: http://arxiv.org/abs/1711.05852

[200] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.

[201] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, and G. Qu, "Approximate computing for low power and security in the internet of things," *Computer*, vol. 50, no. 6, pp. 27–34, 2017.

[202] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.

[203] S. Tajasob, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi, "Designing efficient imprecise adders using multi-bit approximate building blocks," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3218603.3218638

[204] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann, "Bit error tolerance of a cifar-10 binarized convolutional neural network processor," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[205] L. Sousa, "Nonconventional computer arithmetic circuits, systems and applications," *IEEE Circuits and Systems Magazine*, vol. 21, no. 1, pp. 6–40, 2021.

[206] Y. Popoff, F. Scheidegger, M. Schaffner, M. Gautschi, F. K. Gürkaynak, and L. Benini, "High-efficiency logarithmic number unit design based on an improved cotransformation scheme," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 1387–1392.

[207] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, p. 607—617, 2019. [Online]. Available: https://doi.org/10.1038/s41586-019-1677-2

[208] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[209] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[210] H. Paugam-Moisy and S. M. Bohte, "Computing with Spiking Neuron Networks," in *Handbook of Natural Computing*, G. Rozenberg, T. Back, and J. Kok, Eds. Springer-Verlag, 2012, pp. 335–376. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01587781

[211] H. Jang, O. Simeone, B. Gardner, and A. Gruning, "An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 64–77, 2019.

[212] Kanerva and Pentti, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive Computation*, vol. 1, 06 2009.

[213] P. Kaye, R. Laflamme, and M. Mosca, *An Introduction to Quantum Computing*. USA: Oxford University Press, Inc., 2007.

[214] B. Murmann, "Mixed-signal computing for deep neural network inference," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 1, pp. 3–13, 2021.

[215] W. Haensch, T. Gokmen, and R. Puri, "The next generation of deep learning hardware: Analog computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 108–122, 2019.

[216] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An always-on 3.8uj/86binary cnn processor with all memory on chip in 28nm cmos," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, Feb 2018, pp. 222–224.

[217] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst, "Binareye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28nm CMOS," *arXiv*, vol. arXiv:1804.05554, 2018. [Online]. Available: http://arxiv.org/abs/1804.05554

[218] D. Bankman and B. Murmann, "An 8-bit, 16 input, 3.2 pj/op switched-capacitor dot product circuit in 28-nm fdsoi cmos," in *2016 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2016, pp. 21–24.

[219] K. Bong, S. Choi, C. Kim, S. Kang, Y. Kim, and H. Yoo, "14.6 a 0.62mw ultra-low-power convolutional-neural-network face-recognition processor and a cis integrated with always-on haar-like face detector," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 248–249.

[220] J. Oh, J. Park, G. Kim, S. Lee, and H. Yoo, "A 57mw embedded mixed-mode neuro-fuzzy accelerator for intelligent multi-core processor," in *2011 IEEE International Solid-State Circuits Conference*, 2011, pp. 130–132.

[221] M. J. Marinella, S. Agarwal, A. Hsia, I. Richter, R. Jacobs-Gedrim, J. Niroula, S. J. Plimpton, E. Ipek, and C. D. James, "Multiscale co-design analysis of energy, latency, area, and accuracy of a reram analog neural training accelerator," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 1, pp. 86–101, 2018.

[222] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in

2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2015, pp. 1131–1135.

[223] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," *IEEE Computer Architecture Letters*, vol. 16, no. 1, pp. 80–83, 2017.

[224] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, 2017.

[225] S. Soro and W. R. Heinzelman, "A survey of visual sensor networks." *Adv. in MM*, vol. 2009, pp. 640 386:1–640 386:21, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/amm/amm2009.html#SoroH09

[226] G. Kokkonis, K. E. Psannis, M. Roumeliotis, and D. Schonfeld, "Real-time wireless multisensory smart surveillance with 3d-hevc streams for internet-of-things (iot)," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1044–1062, 2017. [Online]. Available: https://doi.org/10.1007/s11227-016-1769-9

[227] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[228] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, "Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic," in *2016 International Conference on Field-Programmable Technology (FPT)*, 2016, pp. 77–84.

[229] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[230] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural network on fpga," *Neurocomputing*, 2017.

[Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217315655

[231] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 26–35. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847265

[232] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 3123–3131. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969442.2969588

[233] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 65–74. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021744

[234] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 15–24. [Online]. Available: http://doi.acm.org/10.1145/3020078.3021741

[235] A. Noda, M. Hirano, Y. Yamakawa, and M. Ishikawa, "A networked high-speed vision system for vehicle tracking," in *2014 IEEE Sensors Applications Symposium (SAS)*, 2014, pp. 343–348.

[236] U. Stevanovic, M. Caselle, S. Chilingaryan, A. Herth, A. Kopmann, M. Vogelgesang, M. Balzer, and M. Weber, "High-speed camera with embedded fpga processing," in *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing*, 2012.

[237] R. Mosqueron, J. Dubois, and M. Paindavoine, "Embedded image processing/compression for high-speed cmos sensor," in *2006 14th European Signal Processing Conference*, 2006.

[238] L. Cavigelli, M. Magno, and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[239] A. Mazare, L. M. Ionescu, A. I. Lita, and G. Serban, "Real time system for image acquisition and pattern recognition using boolean neural network," in *2015 38th International Spring Seminar on Electronics Technology (ISSE)*, 2015, pp. 292–295.

[240] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *arXiv*, vol. arXiv:1612.08242, 2016. [Online]. Available: http://arxiv.org/abs/1612.08242

[241] Y. Umuroglu and M. Jahre, "Streamlined deployment for quantized neural networks," *arXiv*, vol. arXiv:1709.04060, 2017. [Online]. Available: http://arxiv.org/abs/1709.04060

[242] N. J. Fraser, Y. Umuroglu, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Scaling binarized neural networks on reconfigurable logic," in *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, ser. PARMA-DITAM '17. New York, NY, USA: ACM, 2017, pp. 25–30. [Online]. Available: http://doi.acm.org/10.1145/3029580.3029586

[243] D. J. M. Moss, E. Nurvitadhi, J. Sim, A. K. Mishra, D. Marr, S. Subhaschandra, and P. Leong, "High performance binary

neural networks on the xeon+fpga platform," *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, 2017.

[244] C. Baskin, N. Liss, A. Mendelson, and E. Zheltonozhskii, "Streaming architecture for large-scale quantized neural networks on an fpga-based dataflow platform," *arXiv*, vol. arXiv:1708.00052, 2017. [Online]. Available: http://arxiv.org/abs/1708.00052

[245] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell, and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[246] A. K. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: wide reduced-precision networks," *arXiv*, vol. arXiv:1709.01134, 2017. [Online]. Available: http://arxiv.org/abs/1709.01134

[247] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML'13, 2013, pp. III–1058–III–1066. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042817.3043055

[248] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot, "Ternary neural networks for resource-efficient ai applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2547–2554.

[249] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[250] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration

of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, 2018. [Online]. Available: https://doi.org/10.1145/3242897

[251] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 317–324.

[252] U. Stevanovic, M. Caselle, M. Balzer, A. Cecilia, S. Chilingaryan, T. Farago, S. Gasilov, A. Herth, A. Kopmann, M. Vogelgesang, and M. Weber, "Control system and smart camera with image based trigger for fast synchrotron applications," in *2014 19th IEEE-NPSS Real Time Conference*, 2014, pp. 1–4.

[253] Y. Wang, T. Takaki, and I. Ishii, "Intelligent high-frame-rate video recording with imagebased trigger," in *2010 World Automation Congress*, 2010, pp. 1–6.

[254] Xilinx, "Vivado HLS," 2020, (Accessed 29.07.2020). [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[255] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie, and X. Zhou, "Dlau: A scalable deep learning accelerator unit on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513–517, 2017.

[256] R. DiCecco, G. Lacey, J. Vasiljevic, P. Chow, G. Taylor, and S. Areibi, "Caffeinated fpgas: Fpga framework for convolutional neural networks," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec 2016, pp. 265–268.

[257] CSEM, "Fasteye - a 1 mp high-speed camera with multiple roi running at up to 64'000 fps," 2017, (Accessed 21.03.2018). [Online]. Available: https://www.csem.ch/Doc.aspx?id=49356

[258] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations,"

arXiv, vol. arXiv:1609.07061, 2016. [Online]. Available: http://arxiv.org/abs/1609.07061

[259] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *arXiv*, vol. arXiv:1502.02551, 2015. [Online]. Available: http://arxiv.org/abs/1502.02551

[260] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of fpga-based neural network inference accelerators," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 1, 2019. [Online]. Available: https://doi.org/10.1145/3289185

[261] J.-Y. Kim, "Fpga based neural network accelerators," ser. Advances in Computers. Elsevier, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0065245820300899

[262] C. Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016.

[263] Xilinx, "Brevitas," 2020, (Accessed 20.09.2020). [Online]. Available: https://github.com/Xilinx/brevitas

[264] D. Wang, K. Xu, and D. Jiang, "Pipecnn: An opencl-based open-source fpga accelerator for convolution neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*, 2017, pp. 279–282.

[265] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2018.

[266] S. I. Venieris and C. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," in *2016 IEEE*

24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2016, pp. 40–47.

[267] S. I. Venieris and C. Bouganis , "Latency-driven design for fpga-based convolutional neural networks," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8.

[268] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "Maloc: A fully pipelined fpga accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2601–2612, 2018.

[269] Xilinx, "Zynq dpu," 2020, (Accessed 01.08.2020). [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/dpu/v3_2/pg338-dpu.pdf

[270] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo, and L. Benini, "Neuraghe: Exploiting cpu-fpga synergies for efficient and flexible cnn inference acceleration on zynq socs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, 2018. [Online]. Available: https://doi.org/10.1145/3284357

[271] K. Xu, X. Wang, X. Liu, C. Cao, H. Li, H. Peng, and D. Wang, "A dedicated hardware accelerator for real-time acceleration of yolov2," *Journal of Real-Time Image Processing*, 2020.

[272] S. Tridgell, M. Kumm, M. Hardieck, D. Boland, D. Moss, P. Zipf, and P. H. W. Leong, "Unrolling ternary neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 4, 2019. [Online]. Available: https://doi.org/10.1145/3359983

[273] Y. Umuroglu, Y. Akhauri, N. J. Fraser, and M. Blott, "High-throughput dnn inference with logicnets," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2020, pp. 238–238.

[274] Z. He, B. Gong, and D. Fan, "Optimize deep convolutional neural network with ternarized weights and high accuracy," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019, pp. 913–921.

[275] Intel, "Intel hls," 2020, (Accessed 21.08.2020). [Online]. Available: https://www.intel.de/content/www/de/de/software/programmable/quartus-prime/hls-compiler.html

[276] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A resource-limited hardware accelerator for convolutional neural networks in embedded vision applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.

[277] A. A. Gilan, M. Emad, and B. Alizadeh, "Fpga-based implementation of a real-time object recognition system using convolutional neural network," *IEEE Transactions on Circuits and Systems II: Express Briefs*, pp. 1–1, 2019.

[278] E. Karl, Y. Wang, Y. Ng, Z. Guo, F. Hamzaoglu, M. Meterelliyoz, J. Keane, U. Bhattacharya, K. Zhang, K. Mistry, and M. Bohr, "A 4.6 ghz 162 mb sram design in 22 nm tri-gate cmos technology with integrated read and write assist circuitry," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 150–158, 2013.

[279] D. King, "Dlib cnn face detector," 2018, (Accessed 23.08.2019). [Online]. Available: https://github.com/davisking/dlib-models

[280] J. Pedoeem and R. Huang, "YOLO-LITE: A real-time object detection algorithm optimized for non-gpu computers," *arXiv*, vol. arXiv:1811.05588, 2018. [Online]. Available: http://arxiv.org/abs/1811.05588

[281] N. Syu, Y. Chen, and Y. Chuang, "Learning deep convolutional networks for demosaicing," *arXiv*, vol. arXiv:1802.03769, 2018. [Online]. Available: http://arxiv.org/abs/1802.03769

[282] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: A 50.6tops/w unified deep neural network accelerator with

1b-to-16b fully-variable weight bit-precision," in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pp. 218–220.

[283] P. Viola and M. Jones, "Fast and robust classification using asymmetric adaboost and a detector cascade," in *Advances in Neural Information Processing Systems*, vol. 14. MIT Press, 2002. [Online]. Available: https://proceedings.neurips.cc/paper/2001/file/0b1ec366924b26fc98fa7b71a9c249cf-Paper.pdf

[284] CSEM, "Ergo 320 specifications," 2017, (Accessed 20.05.2019). [Online]. Available: https://www.csem.ch/pdf/51766

[285] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database forStudying Face Recognition in Unconstrained Environments," in *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, Marseille, France, Oct. 2008. [Online]. Available: https://hal.inria.fr/inria-00321923

[286] H. Tamura, K. Yanagisawa, A. Shirane, and K. Okada, "Wireless devices identification with light-weight convolutional neural network operating on quadrant iq transition image," in *2020 18th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2020, pp. 106–109.

[287] Renata, "Cr2032 datasheet," 2019, (Accessed 08.05.2021). [Online]. Available: https://www.renata.com/fileadmin/downloads/productsheets/lithium/3V_lithium/CR2032_MFR.pdf

[288] P. Mayer, M. Magno, and L. Benini, "A low power and smart power unit for kinetic self-sustainable wearable devices," in *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, 2020, pp. 1–4.

[289] M. Bedier and D. Galayko, "Multiple energy-shot load interface for electrostatic vibrational energy harvesters," in *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2016, pp. 1–4.

[290] N. Jose, N. John, P. Jain, P. Raja, T. V. Prabhakar, and K. J. Vinoy, "Rf powered integrated system for iot applications," in *2015 IEEE 13th International New Circuits and Systems Conference (NEWCAS)*, 2015, pp. 1–4.

[291] P. Mayer, M. Magno, and L. Benini, "Smart power unit—mw-to-nw power management and control for self-sustainable iot devices," *IEEE Transactions on Power Electronics*, vol. 36, no. 5, pp. 5700–5710, 2021.

[292] J. Deng, J. Nagel, L. Zahnd, M. Pons, D. Ruffieux, C. Arm, P. Persechini, and S. Emery, "Energy-autonomous mcu operating in sub-vt regime with tightly-integrated energy-harvester : A soc for iot smart nodes containing a mcu with minimum-energy point of 2.9pj/cycle and a harvester with output power range from sub-µw to 4.32mw," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2019, pp. 1–4.

[293] S. Benninger, M. Magno, A. Gomez, and L. Benini, "Edgeeye: A long-range energy-efficient vision node for long-term edge computing," in *2019 Tenth International Green and Sustainable Computing Conference (IGSC)*, 2019, pp. 1–8.

[294] Sharp, "Ls013b7dh05 specifications," 2020, (Accessed 10.02.2021). [Online]. Available: https://www.sharpsma.com/products?sharpCategory=Memory%20LCD&p_p_parallel=0&sharpProductRecordId=1725803

[295] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "Sphereface: Deep hypersphere embedding for face recognition," *arXiv*, vol. arXiv:1704.08063, 2017. [Online]. Available: http://arxiv.org/abs/1704.08063

[296] Y. Lin, M. Yang, and S. Han, "NAAS: neural accelerator architecture search," *arXiv*, vol. arXiv:2105.13258, 2021. [Online]. Available: https://arxiv.org/abs/2105.13258

Curriculum Vitae

Petar Jokic was born in Thun (BE), Switzerland, in 1989 and grew up in Spiez (BE), Switzerland. He received the B.Sc. and M.Sc. degrees in electrical engineering and information technology from ETH Zurich, Zurich, Switzerland in 2014 and 2016, respectively. Since 2017 he has been with CSEM (Centre Suisse d'Electronique et de Microtechnique), Zurich, Switzerland, to pursue a PhD degree under the supervision of Prof. Dr. Luca Benini. His main research interests include embedded systems, machine learning, self-sustainable operation, and low-power integrated circuit design.