Diss. ETH No. 30265

Agile and Efficient Inference of Quantized Neural Networks

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES
(Dr. sc. ETH Zurich)

presented by

GEORG MARTIN RUTISHAUSER
MSc ETH EEIT, ETH Zurich
born on 19.05.1993

accepted on the recommendation of

Prof. Dr. Luca Benini, examiner
Prof. Dr. Vijay Janapa Reddi, co-examiner

2024

Acknowledgments

When I started my PhD studies at IIS, I was less excited than afraid; afraid that at some point, I would fall apart, crumbling under what I imagined to be the extreme pressures of the academic environment. But while the journey was not without hardships, this document marks its successful conclusion. This conclusion would have been impossible to reach without the people who helped me get here, whom I would like to thank in this section.

First, I want to express tremendous gratitude to my advisor, Prof. Luca Benini. Thank you for giving me the opportunity to join your group, for always supporting me and my work. Your unique combination of clear-eyed critical thinking with unshakeable optimism and an unparalleled work ethic, together with your wealth of experience and engineering knowledge oceanic in both depth and breadth, will continue to inspire me for the rest of my career and beyond. Thank you also to my co-advisor, Prof. Francesco Conti, for always taking time out of your very busy schedule, for technical matters as well as when I just needed some reassurance. I am also grateful to Prof. V. J. Reddi for agreeing to serve as co-examiner at my defense and taking the time to read and evaluate this thesis.

Thank you also to the staff of IIS: The indefatigable Hansjörg, as well as the DZ staff Beat, Frank, Alfonso and Zerun, without whom our design tasks would be impossible.

A main motivation to start a PhD at IIS despite my apprehensions was that I knew some of my future colleagues, and I (correctly) assumed that those I did not yet know and those who would arrive after me would be cut from a similar cloth. My fellow IIS "inmates": you have been nothing short of incredible, as colleagues and as friends, and I will always remember you. In particular, I want to thank everyone who shared J68.2 with me. Manuel Eggimann, thank you for many great discussions and for always knowing "what's up" (until we didn't want to know anymore). Thank you to Philipp Mayer, the great connoisseur of schnitzels of all kinds, for your very special humour that brightened even the darker hours of my PhD, and for enthusiastically indulging my love of obscure video material from past times. Thank you, Xia, for being my friend and for many great conversations that made good times better and bad times more bearable. To Matteo Spallanzani, I owe a special debt of gratitude for shared suffering – I will always have a spoon for you. Thanks to Hanna Müller for being a great officemate and for introducing me to running, a sport I've come to enjoy a lot. Thank you also to Vlad Niculescu for many entertaining conversations, about our very different taste in cars and a range of other subjects. Finally, thank you to Victor Kartsch for being a cheerful officemate and a great guy to talk to at any time.

Outside of J68.2, I have also made many connections that have enriched my life. I'd like to thank Darja Nonaca and Reinhard Wiesmayr for being great friends. Thanks also to Gianna Paulin, Cristi Cioflan, Sergio Mazzola, Victor Jung, Robert Balas and all my other friends in the digital team of IIS for many great interactions in the office as well as outside of it. Furthermore, I owe a special debt of gratitude to Lukas Cavigelli for introducing me to the world of IIS and scientific work in general when I first started my PhD.

Multiple chapters of my thesis are adapted from publications, none of which would have been possible without my co-authors. My most important collaborator has been Moritz Scherer – thank you for all the nice discussions, both about work-related and other matters. My other co-authors are too many to list, but that does not diminish my gratitude for them, which also extends to all the master, bachelor and semester thesis students that I had the honor to supervise. This thesis would not have been possible without all the support and feedback I had from my colleagues; for this reason, I use the first-person plural *we* throughout.

I further want to thank my parents Joni and Baba, for always supporting me in everything I did, and my friends from outside of ETH, who have represented an invaluable tether to reality outside of academia. Jonas, Lorenz, Luana, Rafael, Silvan, and everybody else who I've been fortunate to have as a personal friend: thank you, the words to capture how important you all are to me elude me.

Abstract

The rapid proliferation of the Internet of Things (IoT) has coincided with a revolution in machine learning, driven by deep learning-based algorithms. At the intersection of these developments, smart sensing systems aim to integrate sensing, deep learning-based interpretation of the captured data and reaction to the results of the processing. As battery-powered IoT sensor nodes operate under stringent power, memory and compute resource constraints, the deployment of deep learning algorithms on these systems requires careful and joint optimization in both the algorithmic and the hardware domain.

Quantized neural networks (QNNs) use low-bitwidth integer formats to represent model parameters and intermediate results. This shrinks their memory and storage footprints, and by taking advantage of specialized hardware for low-precision integer arithmetic, the operational efficiency of the inference can be optimized.

This thesis explores how QNNs can be used to implement highly energy-efficient IoT and edge applications, from both the algorithmic and hardware implementation perspectives. The common element in its key contributions is a focus on end-to-end applications, with the goal of providing an evaluation of the potential of QNNs to improve full-system efficiency.

For efficient evaluation of the impact of different quantization policies on the energy efficiency of end-to-end applications, we first

present a methodology for the automated conversion of full-precision networks to trainable and to deployable, integer-only quantized counterparts.

We then explore the real-world potential of extreme quantization in the form of ternarized neural networks (TNNs). On the example of end-to-end gesture recognition from dynamic vision sensor data, we demonstrate state-of-the-art accuracy and energy efficiency with a TNN-based processing pipeline. Next, we present TNN-specific RISC-V instruction set architecture (ISA) extensions, which offer a 40 % increase in throughput on a real-world benchmark network at a negligible overhead in silicon area and power consumption.

While extreme quantization holds the most potential for energy efficiency gains, it incurs a considerable accuracy penalty on more challenging tasks. To optimize the trade-off between networks' statistical performance and end-to-end inference energy, we present a method for finding latency-optimized layer-wise mixed-precision quantization policies for a given network and hardware target platform, achieving up to 30 % lower end-to-end inference latency at full-precision equivalent classification accuracy on the 1000-class ImageNet dataset on a microcontroller-class platform.

Zusammenfassung

Zeitgleich mit der rasanten Ausbreitung des Internet of Things (IoT) hat die Entwicklung von Deep-Learning-Algorithmen eine Revolution im Feld des maschinellen Lernens ausgelöst. Die Schnittstelle dieser Entwicklungen bilden intelligente Sensorsysteme, welche die gemessenen Daten mit Deep Learning interpretieren und direkt auf die Resultate reagieren. Da IoT-Geräte oft batteriebetrieben sind und über eng begrenzte Rechen- und Speicherkapazität verfügen, müssen die eingesetzten Deep-Learning-Algorithmen und die Hardware-Plattformen auf maximale Energieeffizienz ausgelegt und eng aufeinander abgestimmt sein.

Quantisierte neuronale Netzwerke (QNN) sind Deep-Learning-Modelle, in welchen die Modellparameter und Zwischenresultate statt als Fliesskommazahlen als Ganzzahlen mit geringer Bitbreite dargestellt werden. Dies reduziert den Speicherbedarf und ermöglicht die Benutzung von weniger komplexen Hardware-Recheneinheiten, was wiederum die Energieeffizienz erhöht.

Diese Dissertation beschäftigt sich mit dem Einsatz von QNN zur Effizienzsteigerung von IoT-Anwendungen, wobei sowohl algorithmische Aspekte als auch das Hardware-Design erforscht werden. Das übergreifende Ziel soll dabei sein, das Potenzial von QNN zur Steigerung der Gesamtsystemeffizienz auszuloten.

Zur effizienten Bestimmung des Einflusses unterschiedlicher Quan-

tisierungsmethoden auf die Energieeffizienz kompletter Anwendungen präsentieren wir zuerst eine Methode zur automatischen Quantisierung von neuronalen Netzwerken. Mit dieser Methode können sowohl trainierbare QNN als auch vollständig ganzzahlige Modelle für den Einsatz in IoT-Anwendungen automatisch aus herkömmlichen Fliesskommazahl-Modellen generiert werden.

Als nächstes erforschen wir das Potenzial extremer Quantisierung anhand ternarisierter neuronaler Netze (TNN). Am Beispiel eines kompletten Systems zur Klassifizierung von Gesten zeigen wir, dass mit TNN sowohl statistische Genauigkeit als auch Energieeffizienz auf dem Stand der Forschung und darüber hinaus erreicht werden können. Wir präsentieren weiterhin eine Erweiterung des RISC-V-Instruktionssatzes, welche diesen für die Ausführung von TNN optimiert. Bei einem minimalen Zuwachs an Chipfläche und Leistungsaufnahme vermag diese Erweiterung den Datendurchsatz im Beispiel einer realistischen Anwendung um 40 % zu steigern.

Extreme Quantisierung ermöglicht zwar dramatische Effizienzsteigerungen, kann aber auch grosse Einbussen bei der statistischen Genauigkeit eines Netzwerkes zur Folge haben, insbesondere bei komplexeren Datensätzen. Wir stellen einen Algorithmus vor, welcher die Ausführungszeit eines gegebenen neuronalen Netzes auf einem gegebenen System optimiert, indem jede Netzwerkschicht zu einer eigenen Präzision quantisiert wird, um einen optimalen Kompromiss zwischen der algorithmischen Leistungsfähigkeit und der Energieeffizienz des gesamten Systems zu erreichen. Wir zeigen, dass die Ausführungszeit eines Netzes auf einer Mikrokontroller-Plattform mit diesem Algorithmus bei gegenüber dem ursprünglichen Fliesskommazahl-Modell unveränderter statistischer Genauigkeit im ImageNet-Bildklassifizierungsdatensatz mit 1000 Klassen um bis zu 30 % reduziert werden kann.

Contents

Chapter 1

Introduction

1.1 Motivation

The proliferation of small-scale sensing systems such as IoT sensor nodes and wearable devices has coincided with a revolution in data processing. Machine learning (ML) algorithms such as convolutional neural networks (CNNs) and transformer networks have dramatically improved the state of the art in fields such as computer vision, natural language processing or biomedical signal processing [1]–[4]. As most applications involving the collection of data demand for their interpretation and reaction to the result, these developments are naturally symbiotic. However, the approach of transmitting raw data to cloud servers to be processed with powerful and compute-intensive ML models does not scale to the enormous amounts of data collected and conflicts with the constraints of battery-powered, sensor-driven applications requiring real-time performance: wireless communication is power-intensive and incurs communication latency, and the transmission of raw sensor data raises privacy and security concerns.

This conflict is addressed by the emerging research field of *edge AI*,

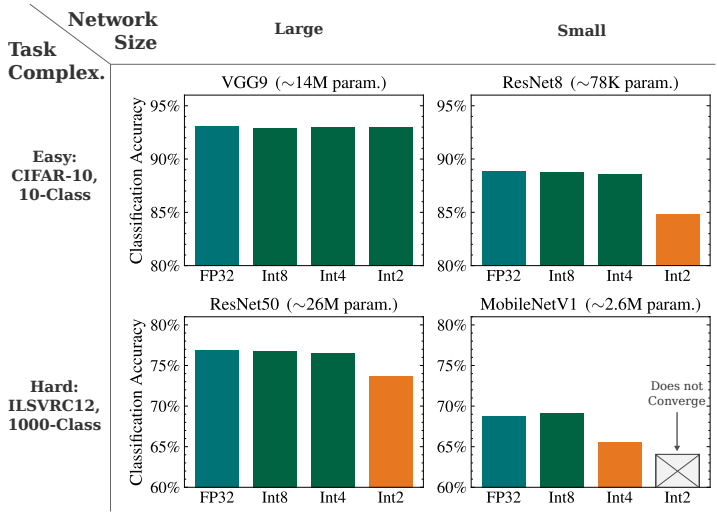


Figure 1.1 – Comparing the impact of integer quantization to different precisions (bitwidths) depending on network complexity and task difficulty. ResNet50 results are taken from [5], all other results are ours.

which aims to enable efficient and accurate ML-based processing of sensor data directly on the edge nodes where they are collected. These systems, typically built around Microcontrollers (MCUs), operate under stringent power, memory, and compute resource constraints. A key algorithmic technique in edge AI is *quantization* of neural networks, where parameters and intermediate activations are represented in low-bitwidth data formats. Smaller data elements directly reduce a given model's memory and storage footprints, enabling the deployment of quantized models on memory-constrained platforms. Lowering operand precision also makes arithmetic hardware units smaller and increases their operational efficiency. Together with the increased data density, this enables higher throughput and efficiency figures at constant memory bandwidth and silicon area budgets.

Of course, the benefits of quantization are not unconditional, but subject to a system of non-trivial trade-offs. On one hand, aggressive quantization to low bitwidths increases a model's data density and operational efficiency. On the other hand, a model's

statistical performance decreases with lower precisions – while research in quantization algorithms has come far in narrowing the accuracy gap, quantizing models to 1-bit or 2-bit precision still typically incurs a considerable accuracy penalty. The trade-off landscape is made more complex by the fact that quantization tends to incur a larger accuracy drop on more difficult tasks than on easy tasks. Similarly, small network topologies tuned toward high efficiency in terms of accuracy vs. model size and computational complexity tend to see a larger accuracy drop from quantization than larger, overparametrized models. This is demonstrated in Figure 1.1. A large network, such as VGG9, used to solve the comparatively easy 10-class CIFAR-10 dataset [6], can be quantized to 2-bit precision with no accuracy drop. On the other hand, when solving the more challenging 1000-class ImageNet dataset [7] with the lightweight MobileNetV1, 4-bit quantization already leads to a substantial accuracy drop, and the 2-bit quantized network does not converge. A designer facing the task of choosing the most efficient combination of model and quantization policy for a given edge AI task must navigate these algorithmic aspects, but that is only one half of the challenge. They must also choose a platform on which to implement their solution and confront the fact that the hardware-software infrastructure most likely does not conform to the idealized performance models that are often used in algorithmic exploration techniques. The design challenge thus involves interactions and feedback loops across the entire stack of algorithmic development, software and hardware infrastructure implementation and application development and deployment, making a divide-and-conquer approach certain to yield suboptimal results and likely to fail outright by delivering a worse trade-off than naïve solutions such as homogeneous quantization to 8 bits, which can be applied with no accuracy drop in the vast majority of cases. Due to this complexity, commercial edge AI solutions by MCU vendors consist of hardware and toolchain solutions that focus on optimization for 8-bit quantization, leaving the efficiency potential of more aggressive quantization unused.

The objective of this thesis is to explore methods to leverage that potential to improve the end-to-end, system-level efficiency of edge AI applications. Its contributions, detailed in the next section, span the gamut from full-stack system design, to ISA extensions tailored to aggressively quantized TNN, to an algorithm to find quantization

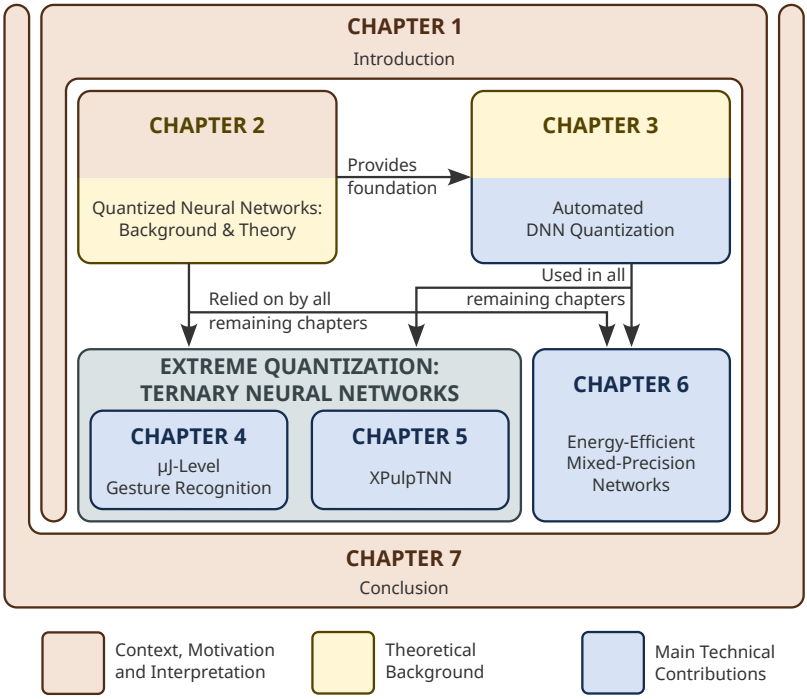


Figure 1.2 – Overview of thesis structure.

policies which optimize the inference latency for a given network and hardware platform. All of these contributions include performance and/or efficiency results for the end-to-end deployment of full networks to real platforms, underlining the focus on end-to-end efficiency.

1.2 Outline

Figure 1.2 provides an overview of how the 7 chapters of this thesis relate to each other. The 5 middle chapters contain the main technical and theoretical contributions and are framed by this introduction and a final conclusory chapter. This introduction provides the motivation for the rest of the thesis and gives an overview of its

content and contributions. The conclusion summarizes and relates the thesis' contributions and gives an outlook on potential future research directions. The rest of the thesis is structured as follows:

1.2.1 End-to-End Quantization: From Theory to Deployed Networks

In the years before and during the writing of this thesis, research in the field of deep neural network (DNN) quantization and hardware for QNN inference has attracted significant interest. Chapter 2 first outlines the basic principles of integer DNN quantization, describing how to obtain trainable, *fake-quantized (FQ)* QNNs from full-precision networks and how to convert these FQ networks to integer-only QNNs. It then gives an overview of the state of the art in the field of DNN quantization algorithms and the hardware platforms used for QNN inference, with a focus on the techniques and hardware relevant to the later chapters of this thesis. To readily apply and evaluate those algorithms in the context of edge AI application development, an end-to-end pipeline is needed to automate the conversion of existing, full-precision DNNs to deployable, integer-only networks. In Chapter 3, we present such a pipeline, which we have integrated in the open-source *QuantLab* [8] Python framework. This pipeline is used in all later chapters to generate, train and export QNNs.

1.2.2 Extreme Quantization with TNNs

Quantization to extremely low precision achieves the highest rates of model compression and allows the use of extremely lightweight hardware for computation. The most aggressive quantization schemes are binary and ternary quantization, where the parameters and intermediate activations can take only two or three different values. In Chapters 4 and 5, we explore the potential of ternary quantization to enable efficient edge AI applications. In Chapter 4, we present an end-to-end gesture recognition pipeline built around a TNN to classify video sequences captured by a dynamic vision sensor (DVS) camera. Our classification network achieves state-of-the-art classification accuracy on an 11-class gesture recognition dataset. It can be run either on a

dedicated TNN accelerator or on a cluster of RISC-V-based general-purpose cores. We implement the gesture recognition pipeline on a physical chip and present real-world power measurements for both approaches, offering a realistic efficiency comparison between inference on an application-specific accelerator and on general-purpose cores. We find that while using the accelerator provides the lowest system-level power consumption, the efficiency gap between the two is orders of magnitude lower than the difference in pure operational efficiency at peak throughput. With both approaches, we achieve a state-of-the-art continuous power consumption of $< 10\,\mathrm{mW}$.

The RISC-V cores we deploy the gesture recognition TNN to are already optimized for integer-bitwidths QNNs, but do not feature TNN-specific optimizations. Furthermore, the inclusion of application-specific accelerators represents a major commitment of silicon area and implementation effort in the system design process. This commitment may not be acceptable when developing an edge platform under stringent cost constraints, where silicon area investments must be justified by returns in the form of versatile functionality. Having documented the efficiency potential of ISA-based processing cores at the system level in Chapter 4, we explore how to improve it further in Chapter 5. We present XpulpTNN, a lightweight extension to the RISC-V ISA consisting of a set of specialized operations for ternary data. XpulpTNN significantly improves the throughput and efficiency of TNN execution at a minimal impact on silicon area and power consumption in general applications.

1.2.3 Mixed-Precision Neural Networks for End-to-End Efficiency

As we demonstrate in Chapters 4 and 5, aggressive quantization to a single low precision has great potential to increase energy efficiency. However, when tackling more difficult tasks with efficient network architectures, extreme quantization tends to incur inacceptable accuracy losses. With *mixed-precision quantization*, different layers of a network are quantized to different precisions. The mixed-precision paradigm opens a vast design space for a given network architecture, allowing for a much finer-grained trade-off exploration than homogeneous quantization to a single precision. As the number

of possible quantization policies is exponential in the number of layers in a network, a brute-force search is out of the question. In Chapter 6, we present *Free Bits*, a directed search algorithm to find latency-optimized precision configurations of a given network for a specific hardware platform. We evaluate our algorithm on multiple variations of a RISC-V-based multi-core MCU platform with two efficient DNN architectures. Our results demonstrate that Free Bits can reliably find Pareto-optimal precision configurations for various latency targets, achieving up to 30 % reduction in measured latency compared to an 8-bit network at full-precision equivalent classification accuracy on the challenging 1000-class ImageNet dataset.

1.3 Contributions

Chapters 1, 3 and 7 are new and unpublished content which I wrote autonomously. The remaining chapters are adapted from publications of which I was the main author (see Section 1.3.1). While I was responsible for the majority of concept formulation, manuscript writing, technical work and experiments, these chapters could not have been realized without the help and feedback from my co-authors. The key technical contributions of this thesis are summarized below.

1. A method for the automated conversion of full-precision networks to their trainable quantized counterparts and for the generation of integer-only deployable QNNs. (Chapter 3)
2. Design, implementation of and evaluation end-to-end processing pipeline consisting of a ternarized CNN architecture for gesture recognition from DVS camera data and preprocessing steps. (Sections 4.4 and 4.6.1)
3. Hardware design, implementation and testing of a DVS camera interface peripheral implementing those preprocessing steps. (Section 4.5.1)
4. Deployment of the complete gesture recognition pipeline to the Kraken SoC and evaluation of the in-silicon power consumption and energy efficiency. (Sections 4.5.2 and 4.6.2)
5. Design and hardware implementation of the XpulpTNN extensions to the RISC-V ISA to optimize the efficiency of

execution of TNNs on general-purpose cores. (Section 5.3)

6. Development of full-stack software support consisting of compiler support, optimized compute kernels and support for automated TNN deployment and evaluation of performance, power consumption and energy efficiency on the complete hardware-software framework. (Sections 5.3.3 and 5.4)
7. Algorithmic design and evaluation of the Free Bits method for finding mixed-precision configurations latency-optimized for specific hardware platforms. (Chapter 6)

1.3.1 List of Publications

[9] G. Rutishauser, M. Scherer, T. Fischer, L. Benini, "*7μJ/Inference End-to-End Gesture Recognition from Dynamic Vision Sensor Data Using Ternarized Hybrid Convolutional Neural Networks*", Future Generation Computer Systems, July 2023.

[10] G. Rutishauser, M. Scherer, T. Fischer, L. Benini, "*Ternarized TCN for μJ/Inference Gesture Recognition from DVS Event Frames*", in Proc. IEEE DATE, 2022.

[11] G. Rutishauser, F. Conti, L. Benini, "*Free Bits: Latency Optimization of Mixed-Precision Quantized Neural Networks on the Edge*", in Proc. IEEE AICAS, 2023.

[12] G. Rutishauser, J. Mihali, M. Scherer, L. Benini, "*xTern: Energy-Efficient Ternary Neural Network Inference on RISC-V-Based Edge Systems*", accepted for publication at IEEE ASAP, 2024.

The following publications are not directly included in this thesis, either because their topics are not sufficiently aligned with the thesis' research focus, or because my contribution to them was of a secondary nature.

[13] L. Cavigelli, G. Rutishauser, L. Benini, "*EBPC: Extended Bit-Plane Compression for Deep Neural Network Inference and Training Accelerators*", IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), 2019.

[14] M. Scherer, G. Rutishauser, L. Cavigelli, 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 (TCAD), 2022.

[15] M. Scherer, A. Di Mauro, G. Rutishauser, T. Fischer, L. Benini, "*A 1036 TOp/s/W, 12.2 mW, 2.72 µJ/Inference All Digital TNN Accelerator in 22 nm FDX Technology for TinyML Applications*", in Proc. IEEE COOL CHIPS, 2022.

[16] G. Rutishauser, R. Hunziker, A. Di Mauro, S. Bian, L. Benini, M. Magno, "*ColibriES: a Milliwatts RISC-V Based Embedded System Leveraging Neuromorphic and Neural Networks Hardware Accelerators for Low-Latency Closed-loop Control Applications*", in Proc. IEEE ISCAS, 2023.

[17] F. Conti, G. Paulin, A. Garofalo, D. Rossi, A. Di Mauro, G. Rutishauser, G. Ottavi, M. Eggimann, H. Okuhara, L. Benini, "*Marsellus: A Heterogeneous RISC-V AI-IoT End-Node SoC With 2–8 b DNN Acceleration and 30 %-Boost Adaptive Body Biasing*", IEEE Journal of Solid-State Circuits, 2024.

[18] A. Nadalini, G. Rutishauser, A. Burrello, N. Bruschi, A. Garofalo, L. Benini, F. Conti, D. Rossi, "*A 3 TOPS/W RISC-V Parallel Cluster for Inference of Fine-Grain Mixed-Precision Quantized Neural Networks*", in Proc. IEEE ISVLSI, 2023.

[8] M. Spallanzani, G. Rutishauser, M. Scherer, A. Burrello, F. Conti, L. Benini, "*QuantLab: A Modular Framework for Training and Deploying Mixed-Precision NNs*", Poster at TinyML Summit, 2022.

Chapter 2

Quantized Neural Networks: Theory and Applications

In this chapter, we provide an introduction to QNNs. We start with a brief overview of approaches to make DNNs more resource efficient. We then introduce the basic theory and fundamental principles of QNNs, before giving an overview of the state of the art in quantization algorithms and hardware for QNN inference.

2.1 Introduction

The breakthrough success of the AlexNet [19] CNN in the ImageNet challenge [7] was followed by the rapid introduction of novel CNN architectures [20]–[22] which proceeded to improve the state of the art in image classification, and were soon successfully applied to other applications such as face recognition [23], [24], content recommendation [25], [26] and speech recognition [27], [28]. As the focus in these initial developments lay on statistical performance rather than efficiency, the size and computational volume of early models was considerable: AlexNet (2012) has around 62M parameters

and required about 380M multiply-accumulate (MAC) operations to compute, and VGG19, proposed in 2014, has 144M parameters and a compute load of 9.8 GMACs per inference. DNN training and evaluation in a research context is most commonly carried out on platforms based on powerful graphics processing units (GPUs), which are capable of handling such processing loads. However, with the increasing demand for DNN processing on embedded systems and edge devices, the large parameter counts and high computational intensity of these early networks were quickly identified as a prohibitive bottleneck to their widespread deployment and application. With DNNs rapidly becoming an intensely researched topic, researchers soon proposed different approaches to address these issues:

Pruning and Codebook-Based Compression *Pruning*, also called *sparsification*, refers to techniques to enforce parameter sparsity (i.e., the proportion of parameters whose value is zero) in a DNN model. Many techniques for sparsification have been proposed; refer to [29] for a comprehensive overview. The simplest approach to pruning consists of setting weights with a magnitude below a certain threshold to zero and, optionally, re-training the sparsified network [30], [31]. A sparse network's parameters can be stored in a sparse representation format such as compressed sparse column (CSC) or compressed sparse row (CSR), leading to a reduction in model size approximately proportional to the degree of sparsity. The remaining nonzero weights can be compressed by clustering them to k centroid values using, e.g., the k-means algorithm. As there are only k discrete values that a parameter can take, a single parameter can be represented as an index in the vector containing the centroids, requiring $\lceil \log_2 k \rceil$ bits of storage space. [32] combines these two steps with Huffman coding of the compressed parameter representation [33] to achieve a reduction in model size by up to 98 % from the original 32-bit floating-point model. While sparsity- and codebook-based compression methods can greatly reduce the size of the stored model, their efficient implementation in application-specific inference hardware is not straightforward: the sparse representation of parameters must be decoded efficiently and sparse computation requires specially designed hardware (as proposed, e.g., in [34], [35]).

Efficient Network Topologies Rather than decrease the resource footprint of a given network architecture, an alternative – and largely orthogonal – approach is to develop DNN architectures which optimize the trade-off between computational and storage resource load and statistical performance. MobileNets [36] were among the first architectures explicitly targeting resource efficiency for mobile and embedded applications. Their key innovation was the depthwise separable (DWS) convolution layer, which replaces regular convolutional layers. By decomposing a convolution with kernel size k into a channel-wise ("depthwise") convolution with kernel size k and a "pointwise" convolution, i.e., a regular convolution with kernel size 1, the parameter count and number of operations is decreased while preserving the receptive field. As shown in [36], this substitution incurs only a moderate drop in statistical performance, providing a favorable trade-off between complexity and accuracy. The second iteration of MobileNets [37] further improved efficiency by using inverted bottleneck blocks with residual connections as building blocks. To scale the network capacity and performance, the input resolution and convolutional layers' channel counts are scaled.

While versions 1 and 2 of MobileNet are parametrizable in their size, the number of layers and their parametrization (excluding the number of input and output channels) are fixed. To find more efficient networks at a given complexity or resource budget, later works proposed more sophisticated approaches. Network architecture search (NAS) algorithms offer a way to search for architectures under consideration of a complexity or efficiency metric; for a comprehensive survey on the topic, we refer the interested reader to [38]. FBNet [39] and MNASNet [40] use NAS to jointly optimize the task loss and inference latency, using layer-wise latency measurements and estimations, respectively, as the quantity to be optimized. MobileNetV3 [41] combines the platform-aware NAS of [42] followed by iterative simplification of the found network with optimized operators and manual adjustments of the network structure. EfficientNets [43] address the question of how to scale up a given, efficient network architecture while maintaining a good accuracy-cost trade-off. Starting from a small baseline architecture found with NAS, the input resolution, each layer's channel count and the number of layers in the network are scaled jointly to obtain a larger, more accurate network.

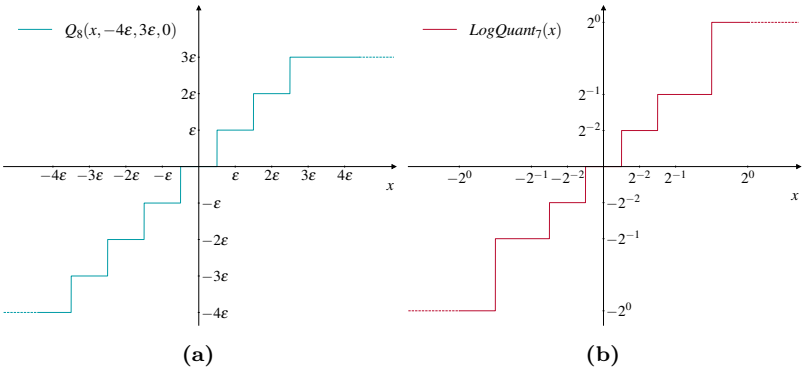


Figure 2.1 – Different 3-bit quantizer functions: (a) shows a linear quantizer with 8 levels with step size ε, while (b) shows a quantizer with 7 levels and logarithmically increasing step sizes.

2.2 A Primer on QNNs

QNNs broadly refers to neural networks whose weights and/or activations are constrained to a discrete set of values. The previously discussed methods for reducing DNNs' resource footprint all operate on full-precision, floating-point models, modifying their structure, parameters and/or representations. The data format used to represent the raw inputs to a computation step, i.e., parameters and intermediate activations, is not altered. Consequently, the elementary MAC operations which dominate the computational load of DNNs must be executed on large, energy-hungry FPUs. Parameters (or, in the case of clustering-based model compression, parameter centroids) and intermediate activations are likewise stored as full-precision floating-point values, contributing to the model size and memory footprint. Furthermore, the specialized hardware required to capitalize on sparsity- and codebook-based compression techniques tends to impose restrictions on the topology of supported models, the order in which computations are performed and the data formats used.

As illustrated in Figure 2.2, low-bitwidth (8 or fewer bits) integer arithmetic units exhibit orders of magnitude lower silicon area and energy consumption than their floating-point counterparts. By

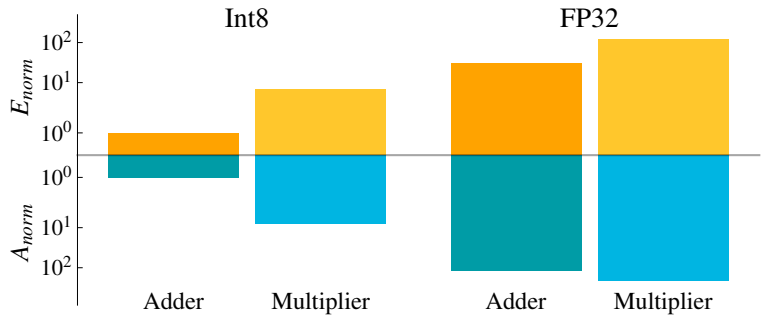


Figure 2.2 – Comparison of normalized energy consumption (E_{norm}) and area (A_{norm}) requirements (in 45 nm process) of 8-bit integer (Int8) and 32-bit floating-point (FP32) adders and multipliers. The figures are normalized to those of an Int8 adder. FP32 arithmetic units have orders of magnitude higher energy and area requirements compared to those of Int8, illustrating the energy-saving potential of integer-quantized DNNs. Data adapted from [44], [45].

representing a model's weights and/or activations in low-precision data formats, such as low-bitwidth integers, QNNs enable a reduction of model size and memory footprint proportional to the reduction in bitwidth. By executing the model with low-precision integer arithmetic, the complexity of the required hardware is reduced, reducing the energy consumed in the execution of a network and facilitating the deployment of QNNs on resource-constrained, battery-powered embedded systems. Different approaches to quantization have been suggested: the quantized values may be spaced logarithmically [46], [47] or may be low-precision floating-point numbers [48], [49]. As clustering-based model compression results in weights being represented by a small number of indices, it may be considered a form of quantization too. In the following explanations, we will focus on *uniform quantization*. In uniform quantization, the quantization levels lie on an evenly spaced grid, with adjacent levels separated by the quantization *step size* ε. Uniform quantization to 2^b levels maps directly to b-bit integer arithmetic, making them very hardware-friendly: the only required hardware are integer MAC units in the appropriate bitwidths, with no further implications on the processing order, network topology, etc.

QNNs have become a key technique both in the high-performance domain to enable more efficient large-scale inference [50], [51] and in the edge domain, where toolchain [52], [53] and hardware [54], [55] support for QNN deployment have become commonplace in commercial offerings. In the remainder of this chapter, we will give an introduction to the working principles of QNNs, the quantization algorithms used to create them and the inference hardware used to run them.

2.2.1 Basic Principles of QNNs

We will consider CNNs consisting of convolutional and linear layers with a fully feed-forward structure (i.e., no recurrent or residual structures - the quantization of residual branches is covered in Chapter 3). Without loss of generality, we consider the l-th layer of the network, with input $\mathbf{X}^l = \mathbf{Y}^{l-1} \in \mathbb{R}^{N_i^l \times H^l \times W^l}$ and output $\mathbf{Y}^l = \mathbf{X}^{l+1} \in \mathbb{R}^{N_o \times H_l \times W_l}$, to be a convolutional layer, optionally followed by a batch normalization [56] layer, and fed into a non-linear activation function σ:

$$\mathbf{Y}_c^l = \sigma\left(BN\left(\mathbf{X}^l * \mathbf{W}_c^l + B_c^l\right)\right) \quad \forall\ 0 \leq c < N_o, \tag{2.1}$$

where $*$ is the 2-dimensional multi-channel convolution operator, $\mathbf{W}^l = \left[\mathbf{W}_0^l, \cdots, \mathbf{W}_{N_o-1}^l\right] \in \mathbb{R}^{N_o \times N_i \times k_y \times k_x}$ are the layer's weights and $\mathbf{B}^l = \left[B_0^l, \cdots, B_{N_o-1}^l\right] \in \mathbb{R}^{N_o}$ is a vector of channel-wise bias parameters. The batch normalization operation on a tensor $\mathbf{T}$ with N_T channels and spatial dimensions $\mathbf{D}_T$ is defined as follows:

$$\mathbf{T} \in \mathbb{R}^{N_T \times \mathbf{D}_T} = [\mathbf{T}_0, \cdots, \mathbf{T}_{N_T-1}]$$
$$BN\left(\mathbf{T}_c\right) = \frac{\mathbf{T}_c - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}\gamma_c + \beta_c, \quad \forall 0 \leq c < N_T,$$

where $\boldsymbol{\mu} = [\mu_0, \cdots, \mu_{N_T-1}]$ and $\boldsymbol{\sigma} = [\sigma_0, \cdots \sigma_{N_T-1}]$ are vectors containing the channel-wise minibatch mean and variance, respectively, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned scale and shift parameters and ϵ is a small constant to ensure numerical stability. For inference, batch normalization (BN) layers can be folded together with the bias into a single affine transform:

$$BN\left(\mathbf{T}_c + B_c\right) = \tilde{\gamma}_c \mathbf{T}_c + \hat{\beta}_c \quad \forall 0 \leq c < N_T,$$
$$\text{with } \tilde{\boldsymbol{\gamma}} = \frac{\boldsymbol{\gamma}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}}, \quad \tilde{\boldsymbol{\beta}} = \tilde{\boldsymbol{\gamma}}\left(\mathbf{B} - \boldsymbol{\mu}\right) + \boldsymbol{\beta}$$

For the ease of illustration, we will assume that our network uses the rectified linear unit (ReLU) activation function:

$$\sigma(x) = \mathrm{ReLU}(x) = \max(x, 0)$$

Note that modern networks feature many other operators and topological structures, such as residual connections [22], concatenations [57] and squeeze-excite layers [41], [58]. This chapter is intended as an introduction to QNNs, a procedure for the automatic quantization of models containing more advanced operators is presented in Chapter 3. Given a full-precision model consisting of such layers, we define two classes of QNNs that we can derive from it:

Fake-quantized networks' parameters and activations take values in a discrete subset of the real numbers, on the same scale as those of the full-precision network. This means that they can not directly be executed on integer-only hardware, and parameters are still floating-point values, albeit from a discrete set. Intuitively, a fake-quantized network approximates its full-precision counterpart, with the precision of the approximation determined by the granularity of the quantization.

True-quantized (or *integerized*) networks' parameters and activations are integers, i.e., a true-quantized network can be computed with integer arithmetic, without further conversion. As such, true-quantized networks are the final output of a quantization pipeline.

In general, the conversion process from a full-precision to a true-quantized model is performed in two steps: from full-precision to fake-quantized, and from fake-quantized to integerized. In the following, we will describe the general procedure for such a conversion, before giving an overview of quantization algorithms designed to optimize the resulting QNN.

2.2.2 Fake-Quantized Networks

In a *fake-quantized* network, weights and intermediate activations are discretized to finite subsets of $\mathbb{R}$. Furthermore, some layers may be

	Signed Int.		Unsigned Int.	
Even n	$c^{lo} < 0,$	$c^{hi} = \frac{n-2}{n} c^{lo}$	$c^{lo} = 0,$	$c^{hi} > 0$
Odd n	$c^{lo} < 0,$	$c^{hi} = -c^{lo}$	$c^{lo} = 0,$	$c^{hi} > 0$

TABLE 2.1
CONSTRAINTS ON CLIPPING BOUNDS c^{hi} AND c^{lo} DEPENDING ON SIGNEDNESS OF THE INTEGER REPRESENTATION AND EVENNESS OF THE NUMBER OF QUANTIZATION LEVELS n.

left in full floating-point precision, e.g., BN layers. A full-precision value $t \in \mathbb{R}$ is converted to its n-level quantized version $\hat{t}$ by the uniform quantizer function Q_n:

$$\hat{t} = Q_n\left(t, c^{lo}, c^{hi}, z\right) = \left\lfloor clip\left(t - z, c^{lo}, c^{hi}\right)/\varepsilon \right\rceil \times \varepsilon + z, \tag{2.2}$$

where c^{lo} and c^{hi} are lower and upper clipping bounds, $\varepsilon = \frac{c^{hi} - c^{lo}}{n-1}$ is the quantization step size and z is a parameter to shift the zero point of the integer representation. Note that in order for the network to be integerizable, z must be a multiple of ε. The clipping function $clip\left(\cdot, c^{lo}, c^{hi}\right)$ is defined as:

$$clip\left(x, c^{lo}, c^{hi}\right) = \min\left(\max\left(x, c^{lo}\right), c^{hi}\right)$$

Intuitively, $Q_n\left(\cdot, c^{lo}, c^{hi}, z\right)$ maps its shifted argument to the closest value on an n-point grid between c^{lo} and c^{hi} whose points are spaced by ε. To simplify the conversion to a true-quantized network, we further constrain c^{lo}, c^{hi} and z. First, we set $z = 0$; while shifting the argument before mapping it to the integer domain enables more faithful quantization of tensors distributed with a non-zero mean, our own experiments and results from literature have shown that good statistical accuracy can be achieved without a shift of the zero point [5], [59]. For simplified notation, we omit $z = 0$ when applying Equation (2.2). Next, we constrain c^{lo} and c^{hi} as shown in Table 2.1, depending on the signedness of the integer representation and the evenness of the number of quantization levels n. For $n = 2^b$, this allows the conversion of a fake-quantized tensor to its (signed or unsigned) b-bit integer counterpart by a simple division by ε. Note also that in the case of unsigned quantization ($c^{lo} = 0$), the ReLU activation is implicitly performed in

the quantization operation. Each layer l's weights and activations can be quantized to a different number of levels n_W^l and n_Y^l. For efficient integer inference, the complete activation tensor $\hat{\mathbf{Y}}^l$ must be quantized to the same step size ε_Y^l, while weights can be quantized channel-wise, with distinct clipping bounds $c_{W,c}^{lo,l}$ and $c_{W,c}^{hi,l}$ for each output channel c. Theoretically, weights can also be quantized to different channel-wise precisions (an approach explored in [60]), but we will assume a single number of levels n_W^l in the following. We will also omit the layer index l in the interest of compact notation, noting that $\hat{\mathbf{Y}}^{l-1} = \hat{\mathbf{X}}^l$. In a fake-quantized network, the computation of layer l thus becomes

$$\hat{\mathbf{Y}}_c = Q_{n_Y}\left(\sigma\left(BN\left(\hat{\mathbf{X}} * \hat{\mathbf{W}}_c + B_c\right)\right), 0, c_Y^{hi}\right) \tag{2.3}$$

$$= Q_{n_Y}\left(\hat{\mathbf{X}} * \hat{\mathbf{W}}_c\tilde{\gamma}_c + \tilde{\beta}_c, 0, c_Y^{hi}\right) \quad \forall 0 \leq c < N_o, \text{ where} \tag{2.4}$$

$$\hat{\mathbf{W}}_c = Q_{n_W}\left(\mathbf{W}, c_{W,c}^{lo}, c_{W,c}^{hi}\right) \quad \forall 0 \leq c < N_o \tag{2.5}$$

2.2.3 Integerized Networks

For deployment, all weight and activation tensors should be represented as integers. Given the l-th layer's weight tensor $\hat{\mathbf{W}}$, quantized channel-wise to step sizes $\boldsymbol{\varepsilon}_W = [\varepsilon_{W,0}, \cdots, \varepsilon_{W,N_o-1}]$, its integer representation $\hat{\mathbf{W}}_{INT}$ can be obtained by division by $\boldsymbol{\varepsilon}_W$:

$$\hat{\mathbf{W}}_{INT,c} = \left\lfloor clip\left(\mathbf{W}_c, c_{W,c}^{lo}, c_{W,c}^{hi}\right)/\varepsilon_{W,c}\right\rceil \tag{2.6}$$

$$= \hat{\mathbf{W}}_c/\varepsilon_{W,c} \quad \forall 0 \leq c < N_o \tag{2.7}$$

Likewise for the integer representation of the l-th activation tensor $\mathbf{Y}$, quantized to n_Y levels with step size ε_Y:

$$\hat{\mathbf{Y}}_{INT} = \left\lfloor clip\left(\mathbf{Y}, 0, c_Y^{hi}\right)/\varepsilon_Y\right\rceil = \hat{\mathbf{Y}}/\varepsilon_Y. \tag{2.8}$$

In an integer-only network, the input to layer l is the integer tensor $\hat{\mathbf{X}}^l_{INT} = \hat{\mathbf{Y}}^{l-1}_{INT}$. Combining (2.1), (2.7), (2.8) and (2.2), we arrive at

$$\hat{\mathbf{Y}}_{INT} = Q_{n_Y}\left(BN\left(\hat{\mathbf{W}} * \hat{\mathbf{X}} + \mathbf{B}\right), 0, c_Y^{hi}\right)/\varepsilon_Y \tag{2.9}$$

$$= Q_{n_Y}\left(BN\left(\hat{\mathbf{W}}_{INT} * \hat{\mathbf{X}}_{INT}\boldsymbol{\varepsilon}_W\varepsilon_X + \mathbf{B}\right), 0, c_Y^{hi}\right)/\varepsilon_Y \tag{2.10}$$

$$= \left\lfloor clip\left(\hat{\mathbf{W}}_{INT} * \hat{\mathbf{X}}_{INT}\tilde{\boldsymbol{\gamma}}\boldsymbol{\varepsilon}_W\varepsilon_Y + \tilde{\boldsymbol{\beta}}, 0, c_Y^{hi}\right)/\varepsilon_Y\right\rfloor \tag{2.11}$$

$$= \left\lfloor clip\left(\hat{\mathbf{W}}_{INT} * \hat{\mathbf{X}}_{INT}\tilde{\boldsymbol{\gamma}}\frac{\boldsymbol{\varepsilon}_W\varepsilon_Y}{\varepsilon_Y} + \tilde{\boldsymbol{\beta}}/\varepsilon_Y, 0, n_Y - 1\right) + 1/2\right\rfloor \tag{2.12}$$

In (2.12), the convolution is performed with integer activations and weights. The multiplication by $\frac{\varepsilon_W\varepsilon_X}{\varepsilon_Y}$ corresponds to a rescaling of the convolution's integer output from a step size of $\varepsilon_W\varepsilon_Y$ to ε_Y. This factor, $\tilde{\boldsymbol{\beta}}$, $\tilde{\boldsymbol{\gamma}}$, and the summand $1/2$ used to substitute the rounding operation by flooring are all not integer. To perform the equivalent computation with integer arithmetic only, the parameters of the combined affine transform can be shifted by D bits and rounded to integers $\hat{\boldsymbol{\gamma}}$ and $\hat{\boldsymbol{\beta}}$:

$$\hat{\mathbf{Y}}_{INT} \approx clip\left(\left\lfloor\hat{\boldsymbol{\gamma}}\hat{\mathbf{X}}_{INT} * \hat{\mathbf{W}}_{INT} + \hat{\boldsymbol{\beta}}\right\rfloor/2^D, 0, n_Y\right), \tag{2.13}$$

$$\hat{\boldsymbol{\gamma}} = \left\lfloor 2^D\tilde{\boldsymbol{\gamma}}\frac{\varepsilon_W\varepsilon_X}{\varepsilon_Y}\right\rfloor \tag{2.14}$$

$$\hat{\boldsymbol{\beta}} = \left\lfloor\frac{2^D}{\varepsilon_Y}\left(\tilde{\boldsymbol{\gamma}}\left(\mathbf{B} - \boldsymbol{\mu}\right)\boldsymbol{\varepsilon}_W\varepsilon_Y + \boldsymbol{\beta} + \varepsilon_Y/2\right)\right\rfloor \tag{2.15}$$

(2.13) is the final, integer-only quantized equivalent of (2.1). The convolution operation is performed on low-bitwidth operands, with the intermediate result accumulated into wider (e.g., 32-bit) integers. The affine transformation parametrized by $\hat{\boldsymbol{\gamma}}$ and $\hat{\boldsymbol{\beta}}$ can be implemented with a single integer multiplication-addition sequence, while the floored division by 2^D is equivalent to an arithmetic right-shift, which avoids the need for expensive integer divisions. The combined affine transform and shifting operation amounts to a fixed-point implementation of the merged batch normalization and requantization operation. D corresponds to the number of fractional bits used to represent $\hat{\boldsymbol{\gamma}}$ and $\hat{\boldsymbol{\beta}}$.

2.3 Quantization Algorithms

With the procedures shown above, a given full-precision network can be converted to an integer-only model. In addition to the model weights, the clipping bounds c^{lo} and c^{hi} need to be defined for every tensor to be quantized. The objective of model quantization algorithms is to determine a suitable parametrization for the conversion. Quantization algorithms can be roughly divided into two classes:

Post-training quantization (PTQ) algorithms optimize only the parameters of the quantization procedure. The parameters of the original model are not modified or retrained, and the amount of data used to optimize the quantization parameters is generally small compared to the size of the training dataset.

Quantization-aware training (QAT) algorithms optimize both the base model's weights and the quantization parameters. They retrain (or *fine-tune*) the model parameters on the full training dataset while taking the quantization into account. QAT generally achieves superior statistical accuracy compared to PTQ, but requires access to the full training dataset and involves greater computational effort, as the complete model is re-trained.

2.3.1 Clipping Bound Initialization

For algorithms of both classes, the first step is to initialize the clipping bounds c^{lo} and c^{hi} for each quantized tensor. In literature, several methods to perform this initialization have been proposed. All of them can be applied in PTQ (in this context, the process is often called *calibration*) and in QAT. In the following, $\mathbf{T}$ refers to an unquantized tensor and $\hat{\mathbf{T}} = Q_n\left(\mathbf{T}, c^{lo}, c^{hi}\right)$ is its counterpart fake-quantized to $n = 2^b$ levels, where $b \in \mathbb{Z}, b > 1$. As activation clipping bounds are usually calibrated with a number of batches N_B of a calibration dataset, we denote the instance of $\mathbf{T}$ produced by the i-th batch of inputs as $\mathbf{T}_i$, $i = 1, \ldots, N_B$. In the case of weight tensors, the procedures for finding clipping bounds are equivalent setting $N_B = 1$, as the weights are quantized based on their values at a fixed point in time. $c_s^{\{lo,hi\}}$ denote clipping bounds for quasi-symmetric (signed) quantization, and

$c_u^{\{lo,hi\}}$ denote clipping bounds for unsigned quantization. The min and max operators are used to denote the maximum element of their argument when operating on a tensor, and the absolute value operator $|\cdot|$ operates on each element of its argument.

Maximum

The simplest way to initialize clipping bounds is to use the maximum and minimum values observed during calibration:

$$c_s^{lo} = -\max_i\left(\max\left(|\mathbf{T}_i|\right)\right),\ c_s^{hi} = -\frac{n-2}{n}c_s^{lo}$$
$$c_u^{lo} = 0, \qquad c_u^{hi} = \max\left(0, \max_i\left(\max\left(\mathbf{T}_i\right)\right)\right)$$

Mean and Standard Deviation

Clipping bounds can be initialized based on the mean and standard deviation of the observed values. As storing all observed tensors $\mathbf{T}_i$ to calculate the mean is impractical, an exponential moving average (EMA) with bias correction is used to track a moving mean. Past values are weighted with a factor β and the current observation is weighted with a factor of $1-\beta$. The EMA of an observed quantity $\mathbf{T}_i$ at time step k is denoted with $\mathrm{EMA}_k\left(\mathbf{T}_i\right)$ and calculate it as follows:

$$\mathrm{EMA}\left(\mathbf{V}_i\right) = \begin{cases} \mathbf{0}, & \text{if } i = 0 \\ \frac{\beta\mathbf{T}_{i-1}+(1-\beta)\mathbf{T}_i}{1-\beta^i}, & \text{otherwise} \end{cases}$$

We denote the mean value of a tensor $\mathbf{T}$ with $\mu\left(\mathbf{T}\right)$ and its standard deviation with $\sigma\left(\mathbf{V}\right)$. Calculating the clipping bounds can be done by setting them to the mean of the observed tensor plus a distance of n_σ standard deviations. For signed tensors, the mean is taken of the absolute value. Unsigned quantization is usually applied to ReLU-activated tensors; if that is the case, we assume that $\mathbf{T}$ is ReLU-activated already. For simplicity of notation, we denote the k-th EMA value of the mean and average of a tensor observation series $\{\mathbf{T}_i\}$ as $\bar{\mu}_k\left(\mathbf{T}\right)$ and $\bar{\sigma}_k\left(\mathbf{T}\right)$, respectively.

$$c_s^{lo} = -\bar{\mu}_{N_B}\left(|\mathbf{T}|\right) - n_\sigma\bar{\sigma}_{N_B}\left(\mathbf{T}\right),\ c_s^{hi} = -\frac{n-2}{n}c_s^{lo}$$
$$c_u^{lo} = 0, \qquad c_u^{hi} = \bar{\mu}_{N_B}\left(\mathbf{T}\right) + n_\sigma\bar{\sigma}_{N_B}\left(\mathbf{T}\right)$$

Mean Square Error

An intuitive measurement of the quality of an approximation is its mean square error (MSE) respective to the exact value. Accordingly, the clipping bounds for a tensor $\mathbf{T}$ can be chosen to minimize the MSE between its unquantized and quantized versions. We denote the MSE between $\mathbf{T}$ and its quantized version $\hat{\mathbf{T}}$ (i.e., the quantization error) as $\Delta_Q\left(\hat{\mathbf{T}}\right) \triangleq \|\mathbf{T} - \hat{\mathbf{T}}\|_2$. The analytical expression the MSE-minimizing clipping bounds is the following:

$$c^{lo}, c^{hi} = \mathrm{EMA}_{N_B}\left(\arg\min_{c^{lo},c^{hi}} \Delta_Q\left(\hat{\mathbf{T}}\right)\right)$$

The arg min operation is assumed to be performed under the respective constraints for signed or unsigned clipping bounds, namely $c_s^{hi} = -\frac{n-2}{n}c_s^{lo}$ and $c_u^{lo} = 0$, respectively. This expression cannot generally be solved analytically, so a grid search can be used to find an approximate solution.

2.3.2 Quantization-Aware Training

QAT methods train the fake-quantized model's parameters directly using stochastic gradient descent (SGD).

STE-based QAT methods The majority of QAT algorithms are fundamentally based on the straight-through estimator (STE), introduced in [61]. In STE-based QAT methods, the forward pass is computed with the fake-quantized model, applying (2.2) to quantize tensors. As the derivative of the quantizer (2.2) is undefined, the STE defines it as the identity:

$$\frac{dQ_n\left(x, c^{lo}, c^{hi}, z\right)}{dx} = \begin{cases} 1, & c^{lo} \leq x - z \leq c^{hi} \\ 0, & \text{otherwise} \end{cases}$$

As the STE makes it possible to optimize model weights using conventional SGD-based training, STE-based methods differ in their treatment of quantization clipping bounds c^{lo} and c^{hi}. This includes

both the initialization of the clipping bounds at the beginning of quantized training and their adjustment over the course of training.

As giving an exhaustive overview of QAT algorithms is out of the scope of this chapter, we will introduce those algorithms we consider the most representative and relevant according to the criteria of i) novelty and impact at the time of publication, ii) statistical performance of the resulting QNNs, and iii) significance to the rest of this thesis.

BinaryConnect (BC) [62], one of the first published implementations of QNNs, proposed a simple, STE-based approach for converting a DNN to a binary weight network (BWN). In the forward pass, weights are replaced by their sign, while in the backward pass, the full-precision weights are updated. This corresponds to setting each quantized weight element $\hat{w}$

$$\hat{w} = Q_2\left(w, 0, 2, 1\right) = \operatorname{sgn}\left(w\right)$$

and applying the STE. As well as decreasing a network's resource footprint, the authors found that BC acts as a regularizer. In this context, they proposed a stochastic binarization of the full-precision weights:

$$\hat{w} = \begin{cases} 1 & \text{with probability } p = \sigma(w) \\ -1 & \text{with probability } p = 1 - \sigma(w), \end{cases}$$

where $\sigma\left(\cdot\right)$ denotes the sigmoid function. Using the stochastic version of BC, the weights are binarized only during the training forward pass, with the full-precision weights used for inference, making this version useful only for regularization purposes. Compared to contemporary regularization methods such as Dropout [63], Maxout [64] and DropConnect [65], BC performed well on the simple MNIST, SVHN and CIFAR-10 datasets. Even the deterministic version, using binary weights for inference, resulted in better accuracies than the unregularized, full-precision baseline - a fact that can likely be attributed to the easy tasks BC was evaluated on.

BinaryNets [66] extended this approach to activations, introducing binarized neural networks (BNNs). It relies on batch normalization to scale the integer results of binary convolutions and benchmarking a GPU implementation of the binary convolution kernel. In hardware,

the binary values ± 1 are represented as single bits, i.e., $\{0, 1\}$. The dot-product of two n-element binary vectors $\mathbf{v}_1, \mathbf{v}_2$ can be computed efficiently from their hardware representations $\mathbf{v}_1^H, \mathbf{v}_2^H$ using bit-wise operations:

$$\mathbf{v}_1 \cdot \mathbf{v}_2 = \left\| \overline{\mathbf{v}_1^H \oplus \mathbf{v}_2^H} \right\|_0 - \left\| \mathbf{v}_1^H \oplus \mathbf{v}_2^H \right\|_0 = n - 2 \left\| \mathbf{v}_1^H \oplus \mathbf{v}_2^H \right\|_0 ,$$

where $\oplus$ denotes the bit-wise XOR operation. The L_0-norm is equivalent to a population count operation, which many ISAs implement natively. The accuracy drop on the same three ten-class tasks that BC was evaluated on was less than half a percentage point from full-precision and binary-weight networks of identical architectures. In [67], the authors extended the concept to multi-bit quantization, composing the multiplication of a k-bit operand by an l-bit operand of $k \times l$ binary multiply-shift-additions. XNOR-Net [68] introduces scaling of the binarized values: Each output channel's binarized weights are scaled by the average L_1 norm. Activation scaling is performed on a finer granularity than in (2.2), with a separate scaling factor applied for each kernel window. While this gives more representational power to the resulting QNN, these scale factors have to be computed from the integer results of the binary convolution at runtime, which limits XNOR-Net's suitability for deployment to platforms with tightly limited memory and compute capabilities. XNOR-Net was the first QNN applied to the large-scale ImageNet [7], with a binarized version of AlexNet reaching a classification accuracy 12 percentage points lower than the full-precision version.

Parametrized activation clipping (PACT) [69] determines the upper clipping bounds for unsigned activations by learning them directly via gradient descent and applying L_2 regularization to it. Weight clipping bounds are determined by a method named statistics-aware weight binning (SAWB). SAWB uses the mean weight magnitude and mean squared weight in a statistical model obtained by linear least-squares regression of the clipping bounds on different statistical distributions. Combining PACT and SAWB to quantize both weights and activations to two-bit precision, [69] reports a classification accuracy for AlexNet within 0.1 percentage points of the full-precision version, while the accuracy drop for ResNet18 is reported to be 3.4 percentage points.

Trained quantization thresholds (TQT) [59] and learned step size

quantization (LSQ) [5] are the last STE-based we describe here. They are closely related, with both methods learning the quantization clipping bounds of both weights and activations by gradient descent. To do so, they define a gradient of (2.2) with respect to ε as

$$\frac{dQ_n\left(x, c^{lo}, c^{hi}, z\right)}{d\varepsilon} = \begin{cases} \left(\hat{x} - x\right)/\varepsilon & \text{if } c^{lo} < x - z < c^{hi} \\ c^{lo}/\varepsilon & \text{if } x - z \leq c^{lo} \\ c^{hi}/\varepsilon & \text{if } x - z \geq c^{hi}, \end{cases}$$

where the first conditional equality is obtained by applying the STE to (2.2), setting the derivative of the rounding operation to unity. This allows the learning of the step size during the fine-tuning process, which determines the clipping bounds (at a fixed offset of $z = 0$). While LSQ trains the step size directly, TQT learns the logarithm of the quantized range to improve numerical stability. Additionally, the original TQT procedure constrains ε to integer powers of two, i.e. $\varepsilon = 2^{-f},\ f \in \mathbb{Z}$. This results in fixed-point quantization: when quantizing unsigned numbers to b bits, the quantized representation will be a fixed-point number with $b - f$ integer bits and f fractional bits. For signed numbers, the number of integer bits is $b - f - 1$. In [5], the authors report ResNets of different sizes with weights and activations quantized to 3 bits reaching full-precision accuracy. For TQT, only combinations of 4-bit and 8-bit weight quantization and 8-bit activation quantization are reported, with quantized ResNets achieving accuracies less than 1 percentage point below their full-precision equivalents. While LSQ's reported statistical performance is better than that of TQT, this discrepancy may be due to the restriction to fixed-point numbers. Furthermore, the authors of [59] only retrained networks with TQT for a maximum of 5 epochs, while in [5], networks are fine-tuned for 90 epochs for all precisions except 8 bits. It should be noted that while the fixed-point restriction of TQT is intended to simplify the rescaling by the factor $\frac{\varepsilon_W^l \varepsilon_X^l}{\varepsilon_Y^l}$ to a simple bit-shift, the presence of batch-norm layers introduces a non-power-of-two factor into the multiplication, eliminating the advantage of this constraint. Similarly, both TQT and LSQ quantize weight tensors to a single step size, which has no resource advantage compared to channel-wise quantization in networks with batch normalization layers, which scale each channel separately regardless of quantization. In Chapters 4 and 6,

we use a version of TQT with channel-wise weight quantization and arbitrary scaling factors to achieve state-of-the-art statistical accuracy with QNNs solving the 11-class DVS gesture dataset [70] and the 1000-class ImageNet classification task, respectively.

QAT without STE The algorithms mentioned so far all make use of the STE, retraining and quantizing a model's weights simultaneously. The incremental network quantization (INQ) algorithm [47] is a weight quantization algorithm that does not rely on the STE. Instead, a model's weights are quantized incrementally over the course of the retraining. Starting from a full-precision model, a fraction of its weights are quantized at the start of the procedure. The quantized weights are frozen and are not updated anymore for the rest of the training, while the weights that remain unquantized are retrained for a few epochs. This procedure is repeated for multiple iterations, with the fraction of quantized weights increasing at each iteration, until in the last step, all weights are quantized. INQ uses logarithmically spaced weights, i.e., $\hat{w} \in \{\pm 2^{n_1}, \cdots, \pm 2^{n_2}, 0\}\ \forall \hat{w} \in \hat{\mathbf{W}}$. This allows MAC operations to be implemented as bit-shifts. With a negligible accuracy drop for weights quantized to 3-bit precision, INQ outperformed previous weight quantization schemes. However, it does not offer a method for the quantization of intermediate activations and STE-based algorithms published in the subsequent years offer better performance while offering comprehensive frameworks for the quantization of both weights and activations.

2.3.3 PTQ

By jointly optimizing quantization parameters and weights, QAT allows for aggressive quantization at minimal accuracy drop from full-precision networks. However, the retraining procedure requires access to the full training dataset and is computationally intensive, as well as requiring the generation of trainable fake-quantized models (see also Chapter 3). To address these shortcomings, PTQ algorithms perform network quantization without modifying the trained full-precision weights. PTQ algorithms determine clipping bounds for each tensor to be quantized based on a limited amount of data. This section will give a brief introduction to the basic working principle of

PTQ before describing some examples of advanced PTQ approaches and comparing the statistical performance of state-of-the-art PTQ algorithms with QAT.

The basic approach to PTQ is to apply the quantizer (2.2) to the unmodified weights of a model and to intermediate activations. To do so, the clipping bounds c^{lo} and c^{hi} must be determined. This can be achieved using any of the methods shown in Section 2.3.1. Statistics of the trained weights can be obtained without running the model, but the activation statistics needed to determine activation clipping bounds depend on the input data the model is executed on. An obvious choice for such a calibration dataset is a subset of the training dataset used to train the full-precision model, an approach adopted by commercially supported tools such as TensorFlow Lite and TensorFlow Lite Micro [71], [72] or Vitis AI [73] by Xilinx. Improvements on this basic model of PTQ have been proposed in two main areas: the amount and nature of data needed for calibration, and techniques for finding the quantized weights.

The requirement for even a small subset of the original training dataset can be a disadvantage. In a scenario where the dataset is not available because it is not public and/or cannot be shared due to privacy concerns, e.g., with health data, methods to quantize pre-trained model without any of the training data are desirable. [74] and ZeroQ [75] use the statistical information about the intermediate activations contained in BN layers' parameters to generate a synthetic input dataset which produces intermediate activations with similar activation statistics. This is achieved by applying loss functions which penalize the difference in intermediate activations' distributions from the stored statistics and backpropagating onto a white noise input to modify it into a more representative input image. Both [74] and ZeroQ achieve a negligible classification accuracy drop relative to full-precision baselines with 8-bit weights and activations on the ImageNet dataset when quantizing various models, including the compact MobileNetV2. [76] removes the dependence on the presence of BN layers in the network, obtaining results competitive with ZeroQ. This is achieved by directly generating labeled input images from one-hot classifier output vectors using a process inspired by visualization techniques such as Inceptionism [77].

Other works have focused on improving the results for lower-

bitwidth quantization by finding quantized parameters that better approximate the behavior of the original full-precision model than those obtained by rounding to the nearest quantized value. With adaptive rounding (AdaRound), [78] proposed learning an element-wise rounding policy for weights. Using the L_2 distance between the output of the full-precision and weight-quantized versions of each layer as a target loss, a tensor that determines whether each weight value is rounded up or down is learned. Using this approach to quantize networks to 4-bit weight and 8-bit activation precision, [78] reported accuracy drops of less than 1 percentage point for ResNets and 1.5 percentage points on MobileNetV2. BRECQ [79] adopts the AdaRound concept, but learns the rounding policy for sequences of multiple layers at a time (e.g., the residual blocks ResNet is composed of) and reported favorable results at even lower precisions. Quantizing MobileNetV2's weights and activations to 4 bits, the authors report an accuracy drop by 6 percentage points relative to the full-precision model. Nevertheless, on low-bitwidth quantization of compact nets QAT methods still incur a significantly lower accuracy penalty (1.1 percentage points in [80] and 2.7 percentage points in [11] on MobileNetV2 for ImageNet). By using SGD to learn the rounding policy, AdaRound and BRECQ arguably blur the lines between QAT and PTQ. While those methods leave the parameters unmodified, AdaQuant [81] applies the principle of layer approximation introduced in AdaRound to directly fine-tune the parameters. Rather than a rounding policy, AdaQuant directly learns the parameters to minimize the L_2 distance between the quantized and full-precision layer outputs. This results in equivalent or better accuracy compared to AdaRound for homogeneously quantized 4-bit ResNets, despite AdaRound using higher-precision 8-bit activations. The overlap in available results for AdaQuant and BRECQ is restricted to 4-bit ResNet18 and ResNet50, where the two approaches report equivalent accuracy.

2.4 Hardware for QNN Inference

To realize QNNs' potential to improve inference energy efficiency, they must be executed on suitable hardware. The main feature of inference platforms for QNNs are arithmetic units which operate on

the reduced-precision data formats used in the QNN to be executed. As with quantization algorithms, hardware for QNN inference is a vast field whose comprehensive coverage exceeds the scope of this thesis, and we will focus on those aspects which are most relevant to the later chapters of this thesis. For a more thorough introduction to the topic, we refer the interested reader to dedicated surveys. [82] gives a general introduction to the principles and processing approaches of dedicated DNN accelerators. [83]–[86] provide snapshots and development analysis of the landscape of both academic and commercial machine learning accelerators in the years 2019, 2020, 2021 and 2022. In [87], the authors present a survey of field-programmable gate array (FPGA)-based DNN acceleration and [88] provides a survey of CNN accelerators, focusing on application-specific integrated circuits (ASIC) designs. Finally, [89] visualizes the efficiency and throughput of a large number of published DNN accelerator designs and can be filtered according to criteria such as operand precision, hardware platform, etc. This allows users to easily compare different classes of accelerators. In this section, we attempt to give a brief overview of the landscape of inference hardware for QNNs. As this thesis emphasizes full-system efficiency, we pay particular attention to the potential of quantization to improve end-to-end energy efficiency in edge scenarios. Processors used to perform QNN inference can be roughly grouped into four categories:

GPUs Manycore processors originally conceived to compute highly parallel algorithms in computer graphics, GPUs have become the main class of processors used for training and inference of DNNs. GPU architecture development has historically been focused on full-precision (FP) performance. As QNNs have gained traction and were shown to achieve statistical performance on par with full-precision models, GPU manufacturers have implemented support for low-precision integer arithmetic. Namely, Nvidia's desktop and server line of GPUs implement tensor cores targeted at deep learning (DL) inference starting with the Volta generation of chips, released in 2017. Apart from 16-bit FP arithmetic, newer generations of tensor cores implement low-precision integer arithmetic. The second generation, introduced in 2018's Turing family of chips, added support

for 8-bit and 4-bit integer arithmetic. Nvidia's embedded GPUs are based on the same microarchitectures as the desktop and server models and have incorporated tensor cores with low-precision integer arithmetic support since 2019's Xavier generation of SoCs. Engineered with a thermal design power (TDP) between 10 W and 65 W, these systems are not suitable for edge applications, but their programmable nature, extensive software infrastructure and hardware support for low-precision integer arithmetic enable the application of high-capacity QNNs in scenarios with a higher power budget, such as large unmanned aerial vehicles (UAVs).

FPGAs FPGAs consist of elementary hardware primitives such as LUTs and flip-flops, embedded in a reconfigurable fabric. By programming an FPGA, the primitives are connected together in a specific way, allowing the implementation of arbitrary digital logic. QNN inference acceleration on FPGAs has been widely researched, and major FPGA vendors are offering parametrizable accelerator solutions that can be implemented on their platforms [73], [90]. However, the flexibility of FPGAs comes at the cost of efficiency: Comparing FPGA to ASIC accelerators in [89]'s visualization, FPGA implementations are roughly 1-2 orders of magnitude less efficient. For edge systems, only the smallest FPGAs with the lowest power consumption would be suitable – these devices are usually too small to implement useful ML accelerators. For this reason, we will not cover FPGA-based inference hardware in more detail and refer the interested reader to the surveys listed at the beginning of this section.

Microprocessor Cores Traditional microprocessor ISAs do not offer native support for low-precision integer arithmetic. Executing QNNs on such architectures thus incurs a runtime overhead for the expansion of packed low-bitwidth operands into a wider, natively supported format. Nevertheless, QNNs' reduced memory and storage requirements can make deployment to such platforms attractive, as demonstrated in [91]. BNNs represent a special case, as the binary MAC operations used in BNNs can be efficiently implemented with bitwise XOR and population count instructions supported by many mainstream ISAs. This makes BNN deployment on off-the-shelf MCU

platforms possible [68], [92]–[94].

To increase inference throughput on non-BNN QNNs, single instruction multiple data (SIMD) instructions that treat each operand register as a packed vector of sub-word elements are commonly used. The ARM ISAs used in many commercially available 32-bit MCUs optionally implement various extensions that support instructions. The ARM DSP extension found in Cortex-M4 and Cortex-M7 processors implements 16-bit MAC instructions. This raises the upper bound on throughput in MAC/cycle by a factor of 4 compared to separate 32-bit multiplications and additions, but does not eliminate the overhead of having to expand 8-bit elements into 16-bit half-words. The latest generation of Cortex-M processors remedies this with the Helium vector extension, which implements SIMD instructions operating on 128-bit vector registers, performing up to 16 8-bit MACs with a single instruction [55].

Academic researchers have presented various extensions to the open RISC-V ISA [95]. The parallel ultra-low power (PULP) family of open-source RISC-V cores, which we will use in several later chapters of this thesis, has introduced several generations of extensions targeting general signal processing tasks and QNN inference in particular. The RI5CY core [96] (now maintained by the OpenHW group under the name CV32E40P[1]) introduced the XpulpV2 extension. XpulpV2 is comprised of lightweight additions to the base RISC-V integer ISA to increase throughput and efficiency on general signal-processing tasks. Post-increment memory access operations and hardware loops reduce bookkeeping overhead for most regular (loop-based) algorithms, while SIMD MAC instructions for half-word and byte operands are particularly useful for QNN inference. The XpulpNN extension [97] explicitly targets QNN inference with SIMD instructions operating on 2-bit and 4-bit data. Finally, in Chapter 5, we propose XpulpTNN, a lightweight extension to RISC-V to speed up TNNs inference with minimal impact on core area and power consumption.

A different approach, adopted by Sparq [98] and Quark [99], is to base custom extensions for sub-byte arithmetic on RISC-V's vector extension. Quark introduces instructions to accelerate bit-serial integer arithmetic, while Sparq's extensions improve the performance

[1] `https://github.com/openhwgroup/cv32e40p`

of ULPPACK [100], a method to perform MAC operations on packed sub-word elements with a regular word-width multiplier.

All of these extensions have in common that they add relatively little implementation overhead to systems built around cores that implement the base ISA. At the same time, they substantially improve throughput and energy efficiency of QNN inference. This makes them attractive to include when designing a system targeting edge AI applications.

Dedicated Accelerators The most efficient platforms for QNN inference are application-specific accelerators. They achieve efficiency through specificity, as they can be designed from the ground up around the constraints of the target application. An extreme example of this is presented in [101], where a BNN architecture with constant parameters is described in RTL and directly optimized with Electronic Design Automation (EDA) tools to obtain a highly efficient – but singularly specific – hardwired BNN circuit. More commonly, QNN accelerators offer some level of flexibility by supporting configurable layer types and variable parameters. Still, the ability to tailor the data path to the specific type of QNN targeted results in dedicated accelerators having both the highest efficiency at a given precision as well as seeing the largest efficiency gain from precision reduction. Evaluated in isolation, configurable BNN and TNN accelerators can achieve efficiencies in the hundreds of TOp/J by using flattened data paths that do not store any intermediate results, eliminating the efficiency overheads inherent to partial computation [14], [102].

However, an operationally efficient accelerator architecture does not automatically yield a practically efficient edge AI system. Most accelerators are optimized for energy efficiency at maximum throughput. In real applications, there may be substantial idle time between inferences – the faster an accelerator performs the inference, the more its idle power consumption will contribute to the average system power. Furthermore, an accelerator alone does not make a versatile edge AI system. Sensor data needs to be collected, prepared and transferred to the inference engine before it can process them. After an inference, the results must be interpreted and, potentially, reacted to. All of these processes can be a bottleneck to the overall energy efficiency of a system. We explore these challenges in Chapter 4 by

designing a complete, TNN-based sensor-to-label classification pipeline, running inference both on a state-of-the-art TNN accelerator and an 8-core cluster of general-purpose RISC-V cores. Notably, the efficiency of the accelerator-based pipeline is two orders of magnitude higher during network execution, but the full-system power consumption in continous operation is only 27 % lower.

2.5 Conclusion

This chapter is intended as the foundation on which the remainder of the thesis builds. To this end, we have given a comprehensive introduction to the principles of integer DNN quantization. With an overview of both the landscape of quantization algorithms and the hardware platforms used for integer QNN inference, we have shown how these principles are applied to create performant QNNs, and how those models can be used to increase the efficiency of DNN inference.

In Chapter 3, we build on the mathematical concepts introduced in Section 2.2 to construct an end-to-end quantization flow to automatically create trainable and integerizable QNN from full-precision models. In all remaining chapters, we use this flow to create the QNN used for our evaluations. In general, all following chapters rely on the theoretical background introduced here.

Chapter 3

Automated Quantization of Deep Neural Networks

3.1 Motivation and Problem Statement

In Chapter 2, we have introduced the concept of QAT and shown how FQ networks can be trained with state-of-the-art quantization algorithms and then integerized for deployment on edge platforms. In practice, users will start from a FP network description, commonly written in Python using frameworks such as PyTorch [103] or TensorFlow [104]. QAT algorithms are usually implemented as extensions to these frameworks and made available by the authors or third parties as a set of quantized layer classes, from which users can assemble QNNs. Manually rewriting the network description in terms of quantized layers is a cumbersome and error-prone process, as is manually converting a trained FQ network to its integerized version. Furthermore, directly replacing each full-precision layer with its quantized counterpart does not generally result in a graph that has an integer equivalent; performing the necessary adjustments to the FQ network manually places a further burden on the designer

and introduces another potential source of errors. Automation of these processes enables researchers and application developers to efficiently apply QAT in diverse settings, explore the impact of different quantization policies and enable rapid turnaround from FP networks to deployable, integer-only QNNs. Ideally, a flow for *automated quantization* should offer the following properties:

Automation: The conversion from FP to FQ and from FQ to integerized networks should be easy to use, requiring minimal adjustments to support different network topologies.

Fidelity: The converted FQ network should have minimal numerical discrepancy from the final FP net. This allows the QAT process to adapt trainable parameters optimally to the quantization and ensures that statistical performance is identical between the FQ and integerized networks.

Flexibility: The user should have complete design freedom to control the quantization process. For example, quantizing individual layers to different precisions or combining different algorithms should be possible.

Modularity: The flow should be extendable, such that support for novel topologies and layer types can be added without major modifications to existing code.

In this chapter, we present a flow that fulfills these criteria. We extend the open-source, PyTorch-based *QuantLab* [8] QAT and experiment management framework with the following features:

- An automated conversion flow from standard floating-point PyTorch modules to integerizable FQ networks, and
- an automated conversion procedure to generate integerized networks from trained FQ networks. The integer-only networks can be exported to precision-annotated graphs in the Open Neural Network Exchange (ONNX) format to be processed by platform-specific deployment tools.

These additions make QuantLab a complete, end-to-end quantization solution. Starting from a full-precision PyTorch network, users can

convert this network to a FQ network that has a direct integer equivalent. The basic quantization policy is specified in terms of layer replacement rules, allowing any level of granularity to be used – from individual settings for each layer to a common policy for all layers of the same type. The FQ network can be trained in QuantLab using a range of state-of-the-art QAT algorithms, and its performance will reflect that of the final integerized version. After training, the FQ network can be automatically converted to an integer-only inference graph. The integer network can be evaluated to ensure the correctness of the conversion, and exported to an ONNX graph to be mapped to a target platform by a deployment tool such as DORY [105]. An overview of the complete QuantLab flow is shown in Figure 3.1. Thanks to QuantLab's modular structure, support for novel topological features and layer types can be added easily.

3.2 Problem Setup

Ultimately, our goal is to quantize a full-precision network architecture, train it using a QAT algorithm and convert it to an equivalent integer-only network that can be exported and deployed to a given platform that only supports integer arithmetic. The procedures we describe can be applied in principle to arbitrary network architectures and hardware platforms, but for the sake of clarity, we will constrain the setting to one relevant to the later chapters of this thesis.

Network Architectures In our examples, we will consider CNNs in the style of ResNets [22] and MobileNetV2 [37]. They contain convolutional, batch normalization and ReLU activation layers arranged into residual blocks that also perform an element-wise addition of activations. These architectures can be implemented as QNN with just four types of FQ operators: quantized convolution layers, quantized fully-connected layers, and quantized activations in signed and unsigned form. The linear operators have quantized parameters, while quantized activations perform an unconditional clipping and requantization of the input to a single step size ε_{out}, i.e., the output step size is independent of the input step size.

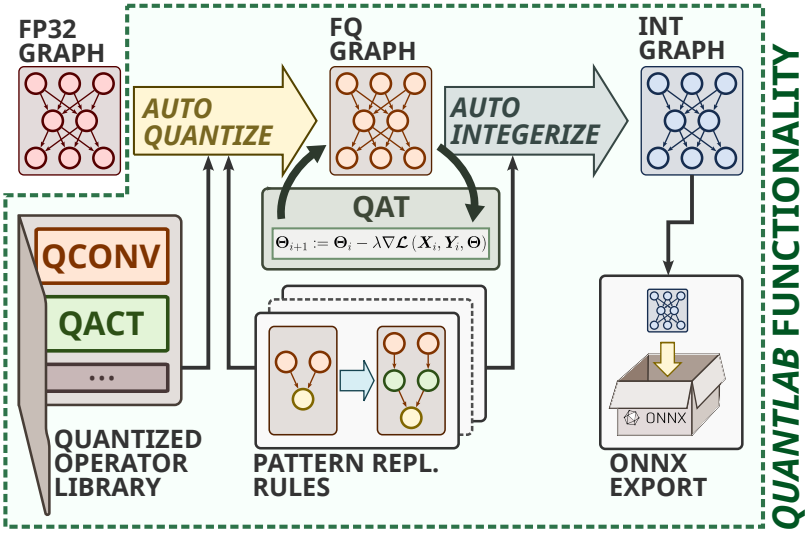


Figure 3.1 – Overview of the flow for automated quantization of neural networks.

Target platform The target platform is characterized by the integer *kernels* it supports. Each kernel is defined by the data formats (bit width and signedness) of its inputs, internally kept intermediate results (if any) and its outputs; each kernel can be thought of as implementing a subgraph of an integerized network. The platform we consider for our examples supports only integer arithmetic. It has kernels to compute convolutional and fully-connected layers, as well as element-wise additions. All kernels operate exclusively on low-bitwidth (i.e., $b \leq 8$) inputs and can perform integer channel norm (ICN) requantization on the output tensors. Intermediate results are accumulated into 32-bit integers before they are requantized to low-bitwidth integers.

3.3 Automated Generation of FQ Networks

The first step in a QAT-based quantization flow consists of implementing the desired topology with quantized operators that support the chosen QAT algorithm. As our ultimate purpose is to obtain an integer-only network for inference, the FQ network should correspond as closely as possible to the final integerized network. This ensures that all effects of the final quantization are modeled during training and that the statistical performance of the FQ network accurately reflects that of the deployed integer network. We achieve this goal with a two-step procedure. First, we perform a *layer-wise replacement*, replacing full-precision operators with their fake-quantized counterparts. In the second step, the resulting graph is searched for patterns that do not have an integer equivalent, and all instances of such patterns are replaced with an integerizable version. Figure 3.2 shows an example of this process: in the first step, the full-precision convolution and activation layers are replaced with quantized versions. In the second step, two non-integerizable patterns are made integerizable: a quantized activation is inserted between two subsequent convolution layers, and two quantized activations with equal step sizes are inserted before the inputs to the element-wise addition.

3.3.1 Requirements for Integerization

In the following, we describe necessary – but not generally sufficient – conditions for a FQ network to be *directly integerizable* for a given target platform. By directly integerizable, we refer to networks that can be integerized with ICN activations as described in Section 2.2. The only mismatches between intermediate activations of the FQ net (after scaling by the respective quantization step size ε) and the integer activations we allow for are those resulting from the fixed-point approximations introduced by the ICN activations. The intuitive fact that the complete computational graph of a network must be mappable to the kernels available on the target platform establishes a first requirement for integerizability:

Operator-Kernel Compatibility There must exist a partition of the network's computational graph into non-overlapping subgraphs, with each subgraph corresponding to a kernel supported by the target platform. However, the target kernels are the building blocks of *integer-only* graphs, whereas we are interested in assessing whether a given *fake-quantized* graph is directly integerizable. Thus, a more accurate formulation of this requirement is the following: First, for each subgraph in the partitioning, there must exist an integerization rule that is applicable to it. Second, the integerized subgraph produced by the rule must map to a kernel supported by the target platform. For a common example that violates this rule, consider our example system, which supports only 8-bit convolutions. For this system, a sequence of two 8-bit convolutional layers with no activation between them – a structure that occurs, for example, in MobileNetV2 [37] – would violate this requirement, as the output of the first convolution is not representable with 8-bit numbers anymore.

Identically Quantized Inputs to Combination Operators We summarize the operations of element-wise addition (found, e.g., in the residual blocks introduced by ResNet [22]) and concatenations under the term *combination operators*. Combination operators must have inputs which are quantized to the same step size. Formally, if the fake-quantized activation tensors $\mathbf{a_1}$ and $\mathbf{a_2}$, quantized to step sizes ε_1 and ε_2, respectively, are added together, we impose:

$$\varepsilon_1 \stackrel{!}{=} \varepsilon_2 \tag{3.1}$$

To understand why this condition is necessary, consider the case $\varepsilon_1 \neq \varepsilon_2$. The integer representations are given as $\mathbf{A_1} = \mathbf{a_1}/\varepsilon_1$ and $\mathbf{A_2} = \mathbf{a_2}/\varepsilon_2$. Clearly, no step size ε_{sum} can be generally determined such that $(\mathbf{A_1} + \mathbf{A_2})\varepsilon_{sum} = \mathbf{a_1} + \mathbf{a_2}$. This means that the sum of the integer activations would not have a unique correspondence to a fake-quantized tensor. The same argument applies to concatenations if one considers that the concatenated tensor is fed into a convolutional layer, which forms a weighted sum of the differently quantized values, leading to the same problem.

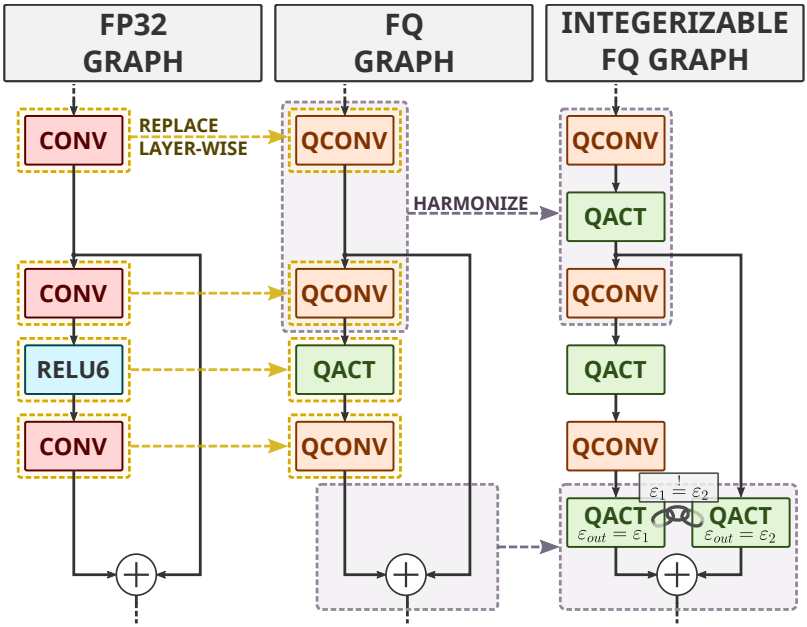


Figure 3.2 – Automated conversion of full-precision networks to their trainable and integerizable fake-quantized counterparts.

3.3.2 Layer-Wise Replacement

In the first step of the conversion procedure, layers that have a direct quantized equivalent are replaced. In the specific case of CNNs, this means replacing convolutions and fully-connected layers with equivalent layers that have fake-quantized parameters. Analogously, activation functions are replaced with quantizing activations.BN layers are left in full precision, as they are folded into the ICN activations in the integerization step.

3.3.3 Harmonization

The network graph resulting from the layer-wise replacement step will be directly integerizable in the case of simple, feed-forward networks that consist only of convolution-normalization-activation layer stacks.

Structures such as residual blocks, introduced in ResNets [22], are not directly integerizable (see above) and require special attention. To ensure that the final network graph meets the requirements listed in Section 3.3.1, we apply a *harmonization* procedure to the graph produced by the layer-wise replacement step. The harmonization procedure is based on *pattern replacement*: The network graph is searched for subgraphs (patterns) which violate the requirements. Each such pattern is associated with a rule that specifies how to create an integerizable replacement pattern. When an instance of a pattern is found, the rule is applied to it to generate a replacement. The found pattern is removed from the graph and the replacement pattern inserted in its place. Commonly, the modifications specified by the rules amount to the insertion of quantizer layers. It is important to note that harmonization is a heuristic procedure: non-integerizable patterns that are not in the library of pattern-rule pairs are not detected and, consequently, not replaced. To support new topologies containing subgraphs that are not integerizable after layer-wise replacements, a user must specify the patterns they contain and formulate the associated replacement rules. In the remainder of this thesis, we will use CNNs in the VGG style (Chapters 4 and 5) and the MobileNetV1/V2 architectures Chapter 6. The examples of harmonization rules we present here are sufficient to make these architectures, as well as ResNets, fully integerizable.

The first harmonization rule ensures that the inputs to linear and pooling operators are always quantized. This is achieved by detecting layer sequences where the output of a linear operator (i.e., a convolution or fully-connected layer, or a BN layer following such a linear operator) is the input to another linear operator or a pooling layer. The replacement pattern consists of an identical layer stack, with an additional signed quantization operator inserted after the first linear operator or BN layer.

A second harmonization rule ensures that element-wise addition nodes are correctly integerizable. This is done by replacing adder nodes with quantized addition layers. The quantized addition layer passes each input through a quantized activation to requantize both input tensors to an identical step size ε. When using QAT algorithms that learn ε (e.g, LSQ [5]), a gradient descent step may cause the two quantizers' step sizes to diverge. To be able to use these algorithms, the

shared step size is set to the larger of the two quantizers' step sizes after each gradient update. If one or both of the inputs are already outputs of a quantized activation layer, those layers are removed from the graph to avoid unnecessary requantization steps. The signedness of each input quantizer is derived automatically from the signedness of the respective adder input. These two steps are sufficient to make ResNet- and MobileNet-style networks directly integerizable. These network families serve as inspiration for many practical edge applications of CNNs [106], [107] and their building blocks are used in various NAS algorithms [39], [108]. Different topological features may require additional quantized layers and/or harmonization passes, both of which can easily be added to the existing framework.

3.4 Automated Integerization of FQ networks

The FQ network produced by the harmonization step can be trained using any of the QAT algorithms supported by QuantLab. As it directly corresponds to an integerized inference graph, its statistical performance will be representative of the final integer-only network. We generate the integerized network in two steps: *epsilon propagation* and *integerization*.

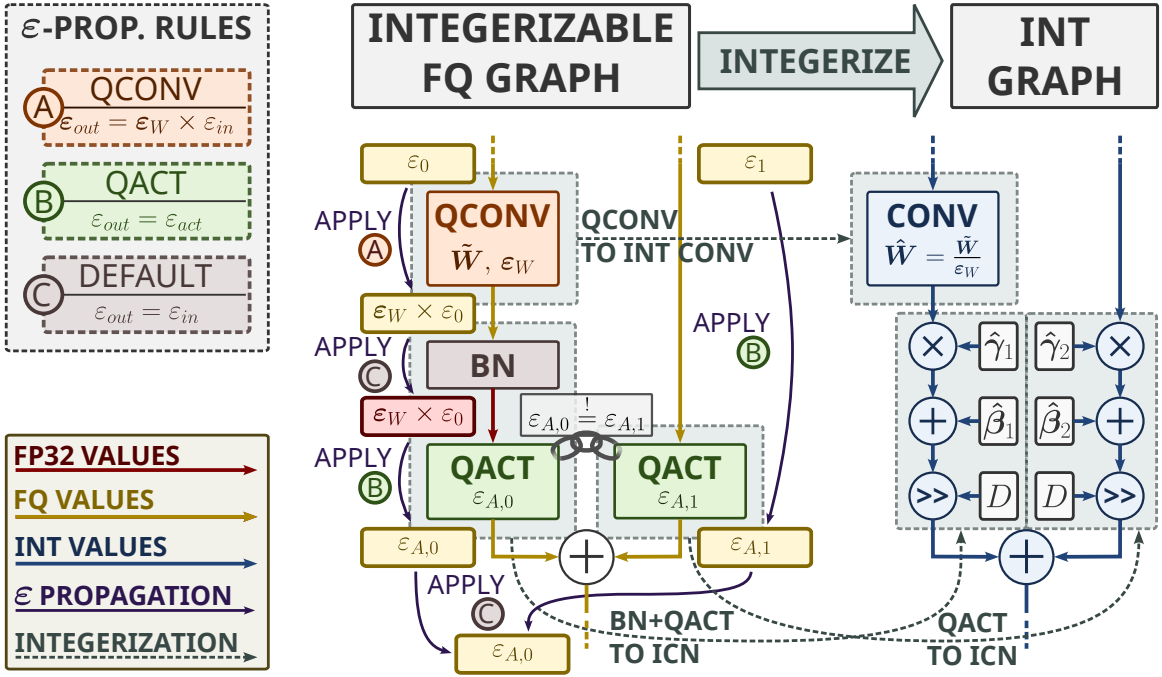


Figure 3.3 – Automated conversion of fake-quantized networks to deployable, integerized QNNs. In the first step, every tensor is annotated with its quantization step size (ε). With this information, the network can be integerized, resulting in a graph that can be executed with only integer arithmetic.

3.4.1 Epsilon propagation

To convert fake-quantized layers to their integer counterparts, we use the procedure described in Section 2.2.3:, specifically Equations (2.7) and (2.13). This requires knowledge of the input and output quantization step sizes in every layer (see Equations (2.14) and (2.15)). The epsilon propagation step annotates the network graph with this information at every node. For every quantized layer type T, an *epsilon conversion* function $f_c^T(\boldsymbol{\varepsilon}_{in}, l)$ specifies how the output quantization step size $\boldsymbol{\varepsilon_{out}}$ depends on the input step sizes $\boldsymbol{\varepsilon}_{in}$ and the layer instance l. Layers for which no explicit epsilon conversion function is specified are assumed to have an identity conversion, i.e., the layer's output is quantized identically to its input. By feeding an input epsilon $\varepsilon_{in,0}$ into the network and propagating the result of the conversion function for each node along the network graph, each node can be annotated with its input and output step sizes. $\varepsilon_{in,0}$ is determined by the scaling of the training dataset relative to the integer inputs that the network will receive when deployed. For example, if the network is trained on raw integer sensor data, the scale in training and inference will be the same, i.e., $\varepsilon_{in,0} = 1$. If the training dataset consists of 8-bit integer sensor data scaled to the range $(-1, 1)$, we would set $\varepsilon_{in,0} = \frac{1-(-1)}{2^8-1} = \frac{2}{255} \approx 0.0078$. A pseudocode description of the epsilon propagation algorithm is given in Algorithm 1.

3.4.2 Integerization

Once the input and output quantization step sizes are known at every layer in the network, it can be integerized. This is again performed as a pattern replacement pass. Convolutional and fully-connected layers are integerized by scaling their weights with a factor of $1/\boldsymbol{\varepsilon}_W$. Sequences of BN and quantized activation layers are converted to ICN operations as described in Equation (2.13). The final graph consists only of integer operations. The export procedure produces an ONNX graph, with each operator node annotated with the input and output precisions. This ONNX file can be consumed by a deployment tool and mapped to the target platform. Figure 3.3 shows both steps of the integerization procedure on an example subgraph.

Network	Precision[a]	Accuracy[b] FP	FQ	Int.[b,c]
ResNet18	8/8 (NH)	69.8 %	69.2 %	**69.2 %**/**69.1 %**
	8/8 (H)		69.4 %	**69.5 %**
	4/4 (NH)		68.6 %	**68.2 %**/**68.3 %**
	4/4 (H)		68.3 %	**68.3 %**
MobileNetV2	8/8 (NH)	71.9 %	71.7 %	**68.7 %**/**62.1 %**
	8/8 (H)		71.5 %	**71.4 %**
	4/4 (NH)		69.3 %	**68.1 %**/**51.7 %**
	4/4 (H)		68.8 %	**69.0 %**

[a] **H** and **NH** indicate whether quantized networks were trained with (**H**) or without (**NH**) harmonization prior to training.

[b] All accuracies are for top-1, single-crop classification on the ImageNet 2012 validation dataset.

[c] For networks trained without harmonization (**NH**), we report two accuracy numbers for the integerized network. The first is obtained by choosing clipping bounds to minimize the MSE between the quantized and unquantized tensors observed from feeding the calibration dataset through the network. For the second, we set the clipping bounds to the maximum magnitude of the observed activations. The color of the accuracy figures indicates the severity of the accuracy drop from integerization: green indicates a negligible drop, orange a non-negligible drop and red indicates an unacceptable drop (> 5 pp.).

TABLE 3.1
EFFECT OF HARMONIZATION ON INTEGER NETWORK ACCURACY.

3.5 Experimental Results

To determine the impact of our harmonization procedure, we consider the statistical accuracy of ResNet18 and MobileNetV2 on the 1000-class ImageNet dataset [7]. We trained each architecture in 4-bit and 8-bit precision, both with and without harmonization before training. After training, the non-harmonized networks were made integerizable by the same harmonization procedure used on the harmonized networks before training. To calibrate the clipping bounds of the newly inserted activation layers, we fed a random subset of the validation dataset through the network and determined two sets of clipping bounds for each layer: the first is found by taking the highest-magnitude value

seen in the respective layer, while the second is found by tracking an EMA across all batches of the clipping bounds minimizing the MSE between the quantized and unquantized tensors. In the 4-bit quantized networks, all linear layers are quantized to 4-bit precision, including the first and last layer. Activations inserted before adder nodes by the harmonization passes are always quantized to 8 bits; all other activation layers – including activations inserted after adder nodes – are quantized to the respective network precision. The complete QAT hyperparameters are shown in Table A.1. The results of our evaluations are shown in Table 3.1. Three aspects of the results are notable. First, in ResNet18, the networks trained in harmonized form are not considerably more accurate after integerization than those harmonized after training. This can be attributed to the fact that in this architecture, the post-training harmonization only inserts activations before adder nodes. In the original network, each adder node is already followed by a ReLU layer which is replaced with a quantized activation in the layer-wise conversion step. Thus, the input activations to the following linear layer are quantized identically between the unharmonized and harmonized nets. Second, in MobileNetV2, harmonization before training has a much larger impact: while the fake-quantized accuracy of unharmonized networks is equivalent to those harmonized before training, integerization leads to a considerable accuracy drop. We attribute this to the fact that in MobileNetV2, the harmonization pass inserts additional 4-bit or 8-bit activations directly before convolutional layers. This changes the quantization step size of the inputs to those layers, whose weights were trained to process much finer-grained data. Finally, in MobileNetV2, the calibration method used to find the clipping bounds in post-training harmonization has a major impact on the final accuracy, which grows with decreasing precision of the inserted activations. When using the maximum magnitude of the observed activations to determine clipping bounds, statistical outliers give rise to clipping bounds with a large magnitude, leading to very coarse quantization granularity. This results in large quantization errors and leads to unacceptable accuracy drops of 17.5 pp. (4-bit quantization) and 7.5 pp. (8-bit quantization). In contrast, using the MSE criterion results in smaller-magnitude bounds that clip extreme outliers, reducing the accuracy drop from integerization to 1.2 pp. (4-bit quantization) and

3 pp. (8-bit quantization). We note that in our experiments with harmonization before training, the initialization method of activation clipping bounds did not have a large impact on final accuracy, as the clipping bounds are adjusted to a suitable magnitude by the TQT algorithm during training.

In conclusion, the benefits of harmonization before training for inference accuracy depend highly on the network architecture. In MobileNetV2, where harmonization introduces additional quantizer layers whose output is consumed by linear operators, a considerable accuracy drop of 1.2 pp. to 3 pp. from integerization is eliminated by training the harmonized network. Harmonization before training also makes the quantization process more robust to suboptimal clipping bound initialization, as clipping bounds can be learned by the QAT algorithm. Finally, we note that for topologies containing, e.g., residual blocks, some form of harmonization is required in every case if the network is to be deployed to an integer-only platform – an unharmonized network will contain structures that are not directly integerizable. With this in mind, harmonization before network training offers only benefits, as the complexity cost of implementing it must be paid in any case for successful deployment.

3.6 Conclusion

In this chapter, we have presented an open-source automated quantization flow for QAT. In the first step, our flow automates the conversion of standard, full-precision networks to trainable FQ networks. An essential step of this conversion is an automated harmonization process, which ensures that the generated FQ graph is integerizable by replacing non-integerizable subgraphs with their integerizable versions. This ensures that the statistical performance of the FQ network is representative of the final integer-only network and allows QAT algorithms to adapt network and quantization parameters to the full extent of quantization is modeled during training,

After QAT, the FQ network is automatically converted to an integer-only inference graph and can be exported to a precision-annotated ONNX graph. By providing an end-to-end solution to quantize existing topologies, our flow makes QAT accessible to application

developers targeting edge platforms that support only low-bitwidth integer arithmetic.

In experimental evaluations, we show that performing network harmonization before QAT eliminates accuracy drops of 1.2 pp. to 3.0 pp. that occur when unharmonized FQ MobileNetV2 networks are integerized.

Algorithm 1 Epsilon Propagation

Input:

L: DNN represented as dictionary $\{i : (\texttt{inputs}, \texttt{m}, \texttt{users})\}$, mapping a layer's (topologically sorted) index i to its `input` indices, a layer object `m` containing layer information (operator type, parameters etc.) and a list of `user` indices indicating which layers use its output.

C: Dictionary $\{\texttt{op} : \texttt{convert\_eps}(\texttt{eps\_in}, \texttt{m})\}$, mapping an n-input operator type `op` to an epsilon-conversion function `convert_eps` taking a list of input epsilon vectors $\texttt{eps\_in} = [\boldsymbol{\varepsilon}_{in,1}, \ldots, \boldsymbol{\varepsilon}_{in,n}]$ and the layer object `m`.

ε_0: Step size of the quantized input to the network.

Output:

ε_{ann}: Dictionary $\{i : (\boldsymbol{\varepsilon}_{in} = [\boldsymbol{\varepsilon_{in,1}}, \ldots, \boldsymbol{\varepsilon_{in,n}}], \boldsymbol{\varepsilon_{out}})\}$ mapping layer indices i to the input step size vectors $\boldsymbol{\varepsilon}_{in}$ for each of the layer's n inputs and the output step size vector $\boldsymbol{\varepsilon}_{in}$

$\varepsilon_{ann}[0].\boldsymbol{\varepsilon}_{in} \leftarrow [\varepsilon_o]$ ▷ Annotate first layer's input step size
for all $(i, \texttt{l} \in L)$ **do** ▷ Iterate over all layers
  $\boldsymbol{\varepsilon}_{in} \leftarrow \varepsilon_{ann}[i].\boldsymbol{\varepsilon}_{in}$ ▷ Get layer's input step size
  if $\texttt{l.op} \in C$ **then** ▷ If a conversion function is registered...
    $\boldsymbol{\varepsilon}_{out} \leftarrow C[\texttt{l.op}](\boldsymbol{\varepsilon}_{in}, \texttt{l.m})$ ▷ ...use it to compute the $\boldsymbol{\varepsilon}_{out}$.
  else
    $\boldsymbol{\varepsilon}_{out} \leftarrow \boldsymbol{\varepsilon}_{in}[0]$ ▷ Else, propagate the first input's $\boldsymbol{\varepsilon}_{in}$
  end if
  $\varepsilon_{ann}[i].\boldsymbol{\varepsilon}_{out} \leftarrow \boldsymbol{\varepsilon}_{out}$
  for all $i_u \in \texttt{l.users}$ **do**
    $\varepsilon_{ann}[i_u].\boldsymbol{\varepsilon}_{in}.\texttt{append}(\boldsymbol{\varepsilon_{out}})$ ▷ Prop. $\boldsymbol{\varepsilon}_{out}$ to all users' inputs
  end for
end for

Chapter 4

Efficient End-to-End Gesture Recognition with Ternary Neural Networks

4.1 Introduction

In this chapter, we apply aggressively quantized QNNs in the context of a complete system built around an MCU. In a practical application scenario, such a systems is typically battery-powered and the key performance characteristic – apart from the latency and accuracy with which it solves the task it is designed for – is the battery lifetime. Thus, energy efficiency becomes a system-level design goal; e.g., efficient QNN inference on its own is not useful if the system has a very high idle power consumption. Thus, to optimize both power consumption and performance, the entire pipeline from the sensor to the reaction to the results of processing must be considered. An efficient end-to-end solution consists of a low-power sensor that minimizes redundant data generation, a tightly integrated sensor-processor interface to avoid communication overhead (e.g., from dedicated logic circuits for sensor

readout) and an efficiently implemented processing algorithm executed with low latency.

In the context of visual sensing, DVS cameras minimize sensing and communication energy by only capturing and outputting events describing the location and polarity of pixel-wise brightness changes exceeding a certain threshold. Accordingly, DVS communicate at a data rate proportional to the activity in the captured scene, with a corresponding reduction in communication energy when operating in static or low-activity environments.

Spiking neural networks (SNNs), a class of brain-inspired neural networks, operate directly on a stream of input events and extend the principle of energy-activity proportionality to the processing domain. As such, they can be viewed as the natural processing paradigm for event-based vision systems and the combination of the two principles has attracted significant research. However, SNNs are still an emerging technology: they lag behind classical ML models such as CNNs both in task accuracy and in hardware support, and have not yet proven their suitability for low-power applications in the edge computing domain [109], [110]. Aggregating the DVS event stream into video frames provides an alternative to SNN-based processing and allows the use of more established, highly optimized machine learning models such as aggressively QNNs [10], [111], [112].

In this chapter, we follow the latter approach and present a frame-based end-to-end processing pipeline for DVS event data implemented on the RISC-V-based Kraken SoC. Thanks to a dedicated on-chip DVS camera interface implementing configurable event frame aggregation, data transfer, and processing overheads are minimized. By classifying the accumulated frames with a 2-stage TNN, either on the integrated CUTIE accelerator [14] or on a PULP cluster of 8 RISC-V cores, we show that frame-based processing of DVS event data has the potential to enable ultra-low-power gesture recognition in real-time on edge computing nodes.While the CUTIE accelerator offers maximal efficiency at very low latencies, it is a large design with a silicon footprint of $3\,\mathrm{mm}^2$ in a 22 nm process and imposes restriction on the supported network architectures, such as the restriction to TNNs, the number of channels per layer or the maximum kernel size. The fully software-programmable PULP cluster has no such restrictions and can efficiently perform a wide range of compute tasks. An efficient

implementation of our network on the cluster is thus an attractive option for area-constrained systems requiring maximal flexibility and a point of reference for applications requiring network architectures not compatible with CUTIE.

The contributions of this chapter are the following:

- We present a detailed exploration of the impact of architectural parameters and training algorithms on the performance of ternarized hybrid temporal convolutional networks (TCNs) for gesture recognition from DVS event frames.
- Motivated by ReLU activations' superior performance, we present a procedure to convert networks using 3-level unsigned (ReLU) activations to networks with signed ternary activations commonly supported by TNN accelerators.
- We evaluate the effect of the different parameters in generating event frames from raw DVS camera output on the network's statistical accuracy and latency, achieving state-of-the-art classification accuracy of up to 97.9 % on the DVS128 gesture dataset.
- We map the proposed network architecture on both the CUTIE TNN accelerator [14] and a PULP cluster of 8 RISC-V cores with native hardware support for low-precision arithmetic. We report in-silicon measurements of inference energy for both targets, demonstrating continuous end-to-end inference at a power consumption of 4.7 mW and an inference energy of 7 µJ at a latency of 0.9 ms on CUTIE. On the PULP cluster, continuous inference consumes 6.4 mW with a classification energy of 0.41 mJ at a latency of 17.8 ms. The energy per CUTIE inference represents an improvement on the state of the art for embedded end-to-end gesture recognition by a factor of 67×. The energy per inference for cluster-mapped networks is still competitive with the state of the art, demonstrating the viability of software-based processing of DVS events within a sub-10 mW power budget.

4.2 Related Work

As this chapter presents a pipeline for power-efficient end-to-end gesture recognition from DVS data, we first focus our review of the related literature on works that perform gesture recognition on embedded, low-power devices with various types of sensors before giving a more detailed overview of works that employ event cameras as the primary sensor. The interested reader may refer to [113] for a wider perspective on deep learning-based techniques for device-free wireless sensing tasks such as person localization, gesture recognition, or fall detection.

	[114]		[115]	[70]	[116]		**This work**
Sensor	SRR	DVS128 [117]/sEMG (Myo)		DVS128	DVS128		DVS132S [118][a]
P_{sensor}	95 mW		>23 mW[b]	23 mW	23 mW		250 µW
Dataset	Custom		Custom	DVS128	DVS128		DVS128
Model	CNN/transformer		SNN	SNN	SNN		Ternarized CNN/TCN
End-to-end?	✓		✗	✓	✗		✓
Processor	GAP8	ODIN + MorphIC	Loihi	TrueNorth	Loihi	CUTIE	PULP Cluster
P_{idle}	-	-	29 mW [119]	134.4 mW	29 mW [119]	4.7 mW	3.6 mW
P_{cont}	>7.1 mW[c]	-	33.2 mW [d]	178.8 mW	-	**4.7 mW**	6.4 mW
t_{inf}	9 ms	19.5 ms	7.8 ms	-	11 ms	**0.9 ms**	17.8 ms
E_{inf}	0.47 mJ	37 µJ [e]	1.1 mJ [e]	18.8 mJ	-	**7 µJ**	0.41 mJ
Accuracy	77.15 %	89.4 %	96.0 %	94.6 %	90.5 %	**97.7 %**	**97.7 %**

[a] Network accuracy was evaluated on the DVS128 gesture dataset collected with a different sensor.
[b] Power consumption of the Myo armband is not specified.
[c] We assume 15 inf/s and zero idle power consumption as a lower bound for continuous power consumption.
[d] Calculated from idle power, inference energy and inference latency.
[e] Only dynamic energy figures reported.

TABLE 4.1
COMPARISON OF THE RESULTS WITH STATE OF THE ART EMBEDDED GESTURE RECOGNITION IMPLEMENTATIONS

4.2.1 Embedded Gesture Recognition

Gesture recognition based on electromyography (EMG) has attracted significant research interest. A common approach is to attach an EMG electrode array to the forearm and classify the signals using a DNN. [120] combines a custom analog frontend for 8-channel EMG signal acquisition and analog-to-digital conversion with an ARM Cortex-M4 microcontroller running a support vector machine (SVM) to classify the input. The complete system achieves an average classification accuracy of 89.7% on four user-specific datasets of 7 gestures in a power envelope of 29.7 mW. [121] proposes a larger system based on a 32-channel wireless sensing armband and CNN-based classification on the NVidia Jetson Nano platform, achieving 98.5% classification accuracy on eight gesture classes collected from a single user with 5 ms of inference latency at a total power consumption of 3.1 W. At tens-of-mW power envelopes, other EMG-based embedded gesture recognition systems [122]–[124] achieve comparable results to [120]. The drawback of EMG-based gesture recognition is that it is not suitable for ad-hoc human-machine interaction for two reasons. First, the models are commonly user-specific, requiring retraining for different users. Second, placing EMG electrodes requires careful setup and preparation. While systems supporting on-device learning based on hyperdimensional computing [124] have addressed the former issue, the latter is inherent to EMG interfaces.

A different sensing approach is to use miniaturized radar sensors. Google pioneered this technique with SOLI [125], using custom radar ASICs in combination with off-the-shelf (embedded and desktop-scale) processing platforms to achieve up to 92.1% classification accuracy on a four-class hand gesture dataset collected from five users with a random forest-based classifier. [126] uses SOLI hardware to collect an 11-class hand gesture dataset, achieving 94.2% classification accuracy with a CNN-based classifier run on desktop hardware. The radar approach was subsequently refined and miniaturized in [107], using embedded short-range radar sensors produced by Acconeer and performing classification with a hybrid TCN on the RISC-V PULP-based GAP8 processor. The authors report 81.5% classification accuracy at a continuous inference power of 21 mW and a classification energy of 4.52 mJ, bringing radar-based gesture recognition into the edge

computing space. [114] improves on this work by using a transformer-based classifier, reducing latency and inference energy to 9 ms and 0.47 mJ, respectively. While this shows that radar data processing can be performed under strict power constraints, radar sensors are generally power-hungry: Google's SOLI radar consumes 300 mW per chip and the sensors used in [107] use 90 mW each. This means that sensing power dominates full-system power consumption and makes radar-based gesture recognition unsuitable for ultra-low-power applications requiring power envelopes of <10 mW.

Other sensing techniques used for gesture recognition include sensor gloves [127], [128], ultrasonic echolocation [129], [130] and camera-based visual sensing [131], [132]. However, these approaches have not found any application to ultra-low-power embedded systems so far.

4.2.2 DVS Gesture Recognition

In recent years, gesture recognition from DVS event data has seen considerable research interest. The most common approach is to process the event stream using SNNs. In [70], the authors introduced the 11-class DVS128 full-body gesture dataset, which has become the de-facto standard for DVS gesture recognition. They implement an end-to-end gesture recognition system on the NS1e development board based on IBM's TrueNorth neuromorphic processor. The event stream is preprocessed with a cascade of temporal filters and classified with a 16-layer convolutional SNN at 1 ms timesteps. This pipeline achieves a classification accuracy of 91.8%, The application of a sliding-window filter to the output further increases accuracy to 94.6%. TrueNorth's power consumption is measured at 178.8 mW, making this system unsuitable for edge computing applications.

Similarly, [116] proposes an SNN converted from a 5-layer CNN to classify the event stream from the DVS128 dataset. In notable contrast to [70], events are accumulated over much longer timeframes (300 ms for the best-performing network) which are divided into multiple input channels. This processing approach is equivalent to feeding event frames to a SNN. Classification accuracy is reported as 90.5% at a processing latency of 15 ms. While the authors do not provide information on power consumption, [119] reports a single Loihi chip's idle power consumption as 29 mW, providing a lower bound. It should

also be noted that the acquisition and preprocessing of the event stream are not performed on Loihi.

[115] performs sensor fusion on DVS and EMG data, using dual-branched SNN models with a joint classifier layer to perform hand gesture recognition on a custom 5-class dataset. The model mapped to Loihi, consisting of a spiking CNN handling DVS data and a spiking multi-layer perceptron (MLP) to process EMG data, achieves a classification accuracy of 96% at an inference latency of 7.8 ms. The authors also map a model consisting of two spiking MLPs to a pair of research SNN accelerators, ODIN [133] and MorphIC [134]. Like the Loihi model, this smaller model operates on a 200 ms window of input data and achieves 89% classification accuracy at a latency of 19.5 ms. The authors report only the dynamic energy consumption of both networks, which is measured at 1.1 mJ for Loihi and calculated (using detailed power models) at 37 µJ for ODIN+MorphIC. As with [116], the preprocessing step is not accounted for in these figures and is performed offline.

Other works are purely algorithmic and do not include power or energy figures on embedded platforms. [135] proposes a spiking convolutional recurrent neural network, achieving 90.3% validation accuracy on the DVS128 dataset. [136]–[138] propose different methods of training SNNs, achieving validation accuracies of up to 97.6%. [112] proposes a method to aggregate events into decimal pixel values to encode time information, processing these event frames with a large 3D CNN. This approach achieves a near-perfect validation accuracy of 99.6% on the DVS128 dataset, the highest result reported in the literature.

In conclusion, our literature study reveals that accessible end-to-end gesture recognition on extreme-edge devices is an unsolved problem: existing edge solutions use sensing technologies that are either impractical to use (e.g., EMG) or too power-hungry to fit the constraints of the IoT. DVS cameras have the potential to close this gap but have not yet been used to their full potential in ultra-low-power sensor nodes. We propose to close this gap with an event frame-based processing pipeline based on a hybrid TNN mapped to the Kraken SoC, which integrates the sensor interface with efficient processing units (namely, the CUTIE accelerator and 8-core PULP cluster) to perform both inference and general-purpose processing tasks. To the best of

our knowledge, our system outperforms all embedded solutions in both classification accuracy (97.7% on DVS128) and efficiency (7 µJ/inf., 4.7 mW continuous inference power) while providing the versatility of a fully-featured MCU, enabling always-on gesture detection on battery-powered sensor nodes at the extreme edge. Table 4.1 compares our system to the state of the art in embedded gesture recognition systems.

4.3 Background

In this section, we give a brief overview of the operating principle of DVS cameras, their advantages and drawbacks compared to conventional cameras, and the approaches adopted to process DVS data. We then provide background on the training and inference of QNN and a short description of the architecture of the CUTIE TNN accelerator.

4.3.1 DVS Cameras and Processing DVS Data

DVS, first proposed in 1991 [139], are an emerging class of visual sensors that detect and transmit information on brightness changes in the captured scene. In contrast to conventional cameras, which produce a stream of *frames* at fixed time intervals, DVS cameras emit a stream of *events* describing the location and polarity of an individual pixel's change in brightness. An event can be described as a tuple (t, x, y, p), where t denotes the time at which the event is produced, (x, y) are the coordinates of the pixel which produced the event and the polarity $p \in \{-1, 1\}$ indicates whether brightness at the respective pixel increased or decreased by a specific threshold since the last event emitted.

An advantage of DVS cameras over conventional image sensors is that they enable *energy-proportional sensing*: As only localized brightness changes are sensed and transmitted, the pixel array's activity and transmission data rate (and, with them, the respective power consumptions) are proportional to the activity level in the captured scene. In contrast, conventional cameras capture and transmit frames at a constant data rate, leading to constant and activity-independent power consumption. A second advantage lies in their low latency (typically < 200 µs) due to their asynchronous nature: Each pixel in

a DVS array operates independently, emitting information about a change in the captured scene as soon as it occurs. Sensing latency is thus limited only by the readout circuitry and the physical interface used to transmit the event stream. DVS pixels also typically have a high maximum event rate per pixel of >300 e/s, making DVS cameras suitable for high-speed applications. Due to the logarithmic brightness response of DVS pixels, DVS cameras also support high dynamic ranges of > 120 dB [117], [140], [141], enabling their application in a wide range of lighting conditions.

Compared to shutter-based cameras, DVS cameras have some limitations. The first is a lack of color representation; while there are some research works exploring this [142], commercially available DVS cameras are exclusively monochrome. Availability and price are further obstacles to widespread adoption of DVS technology; while camera modules for embedded applications are mass-market products, DVS camera availability is very limited and prices range in the equivalent of thousands of dollars.

The processing of DVS data presents a significant challenge, and [111] divides approaches into two broad categories. The first class aggregates event data into frame-like groups. An example of this is time surfaces, where the value of each pixel describes how recently the last event at that pixel occurred. The most basic approach in this class is that of *event frames*, which we also adopt. Event frames are constructed by dividing the event stream into fixed time intervals, aggregating the events occurring during each frame interval into pixel values for each spatial location in the frame [10], [112], [143]. These approaches enable processing with conventional image processing algorithms such as DNNs, but omit some of the information in the event stream. E.g., in periodically sampled time surfaces, there is no information about how many events on a given pixel occurred during the frame interval.

The second category of processing algorithms for DVS data operates on individual events. This category's most prevalent algorithmic paradigm is that of SNNs. A class of artificial neural networks (ANNs) inspired by the working principle of the human brain, SNNs, consist of one or multiple layers of neurons that behave according to a biologically-motivated model. In the most commonly utilized model, the leaky integrate-and-fire (LIF) neuron, each neuron in the network is connected to neurons in the previous layer and takes discrete spikes

as its input. The input spikes are weighted with parameters specific to their source and added to the neuron's internal membrane potential, which decays exponentially independent of input spiking activity. When a neuron's membrane potential reaches a parametrizable threshold, it fires and propagates a spike to the neurons of the next layer to which it is connected. Upon firing, the membrane potential resets to a rest potential. SNNs are fundamentally asynchronous and time-continuous, but their dynamics can be discretized in time, a fundamental step for their implementation in digital circuits [109], [136]. As DVS events can be directly mapped to input spikes, SNNs are a natural fit for processing DVS data and this pairing has accordingly attracted widespread research attention. Examples of their successful application include optical flow estimation [144], [145] and image classification [136], [146], [147] on event datasets converted from traditional datasets consisting of static images [148]–[150]. Gesture recognition, the application we target, has also been implemented with SNNs, achieving accuracies of up to 94.6% on the DVS128 dataset [70].

However, on all the above tasks, the accuracy of SNN-based methods does not match that of traditional DNNs [109], [112], [136], [151]. Furthermore, the integration of flexible SNN accelerators suitable for executing complex networks into systems capable of end-to-end processing is not well-established. So far, it has been restricted to large-scale designs such as Intel's Loihi family of systems [152], [153] or IBM's TrueNorth platform [154].

4.3.2 Training and Inference of Aggressively Quantized Neural Networks

Extreme Quantization - Binarized and Ternarized Networks

The techniques introduced in Section 2.2 can be applied to achieve quantization to any desired precision, with lower-precision quantization generally leading to more significant degradation of task accuracy. The most extreme forms of QNNs are BNNs [66], [68] and TNNs [155], [156]. In BNNs, all weights and activations are quantized to values in $\{-1, 1\}$. TNNs can be considered an extension of BNNs, with tensor elements taking values in the set $\{-1, 0, 1\}$. Consequently, arithmetic operations on binary and ternary values can be implemented with

simple hardware: Computing the dot product of two N-element binary vectors can be achieved by N XNOR gates and an N-bit popcount unit [68]. The dot product of two N-element ternary vectors can be computed using N ternary multipliers and 2 N-element popcount units. This property makes BNNs and TNNs highly attractive targets for acceleration. While binary and ternary quantization schemes allow for the design of simple arithmetic units which improves inference energy efficiency considerably [14], statistical inference accuracy is typically degraded, even when employing QAT. Significant research has been conducted to combat this negative impact on accuracy and significant advances have been reported [157], [158]; however, low-bitwidth quantization typically still carries a non-negligible penalty on statistical accuracy. As such, binary and ternary neural networks are best suited for applications where the penalty on accuracy is acceptable, such as in low-complexity tasks or always-on sensing or wake-up trigger applications. In this chapter, we apply the INQ [47] and TQT [59] QAT algorithms to the training of binary and ternary neural networks, comparing the resulting networks' accuracy and showing that TNNs are ideally suited for ultra-efficient gesture recognition from DVS data as they achieve minimal accuracy drop while allowing inference to be run on simple, highly efficient hardware. In particular, we deploy our gesture recognition TNNs to the CUTIE accelerator, which we introduce in the following.

The CUTIE TNN Accelerator

The Completely Unrolled Ternary Inference Engine (CUTIE) is an accelerator for TNNs, introduced in [14]. In contrast to most neural network accelerator architectures, CUTIE uses a compute architecture that is fully parallel in the computation of each output pixel by processing elements termed output channel compute unit (OCU). Each OCU computes the sum of dot products corresponding to a single filter in a fully unrolled manner and in a single cycle. To parallelize the computation of each pixel, CUTIE uses one OCU per output channel. Each OCU has a dedicated latch-based weight buffer that holds all filter weights. In order to reuse input kernel windows optimally, CUTIE features a 3-line buffer, which dispatches full 3×3 kernel windows.

CUTIE's maximally parallel design not only increases arithmetic

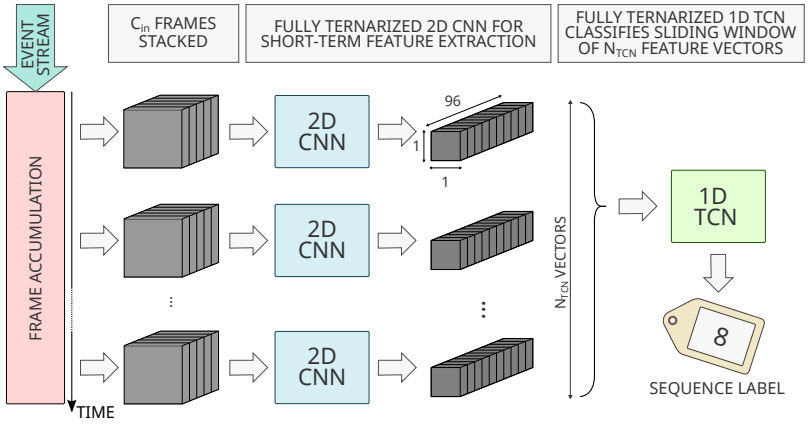


Figure 4.1 – Overview of the proposed processing pipeline.

optimization by forming large adder trees over the sum of dot products in each OCU, but also minimizes data movement since every pixel in the input feature map and every weight in every filter is only loaded once per layer. These characteristics make CUTIE highly efficient for neural network inference at the edge while enabling throughput at rates exceeding 1000 inf./s. Kraken features an updated version of CUTIE, described and evaluated in detail in [159]. The main improvements from the original version are support for causal, dilated 1D convolutions and the addition of an intermediate buffer for a window of up to 24 CNN output vectors, enabling the execution of ternarized TCNs. The CUTIE accelerator in Kraken features 96 OCUs, computing one output pixel with up to 96 input/output channels per cycle and resulting in a peak throughput of 56 TOp/s when clocked at 54 MHz for maximum efficiency in a 22 nm implementation.

4.4 Event Frame Processing Pipeline

4.4.1 Overview

The class of cameras we target produce events consisting of three values each: Two spatial coordinates (x_e, y_e) describing the location of the event on the sensor grid, and a polarity $p_e \in \{-1, 1\}$ to indicate a decrease or increase in brightness, respectively. The first stage of our proposed pipeline performs data preparation, aggregating events detected by the DVS camera into 2-dimensional frames. The resulting frames are natively ternary: Pixels where at least one event occurred during a frame interval take the value of the most recent event's polarity, while pixels with no activity take the value 0. The prepared data is then processed by a two-stage hybrid CNN inspired by [107]. In the first processing stage, C_{in} sequential frames are fed into a fully ternarized 2D CNN. By processing multiple frames per inference, the 2D CNN can analyze short-term temporal dependencies, which are encoded into ternary feature vectors by the last layer.

In the third and final stage, a fully ternarized TCN analyzes the longer-term temporal dependencies by performing inference on a sliding window covering N_{TCN} feature vectors. Its output is a vector of class scores $V_p \in \mathbb{Z}^{N_c}$, where N_c is the number of classes. For the DVS128 dataset, $N_c = 11$. Finally, the class label of the sequence is given as $Cls = \arg\max_i (V_p)$.

Figure 4.1 shows a diagram of the proposed processing pipeline. The frame extraction process is shown in Figure 4.2 and described in more detail in Section 4.4.5.

4.4.2 Network Design

The ternarized hybrid CNN/TCN adopts a simple feed-forward architecture, i.e., there are no residual branches in the network. All layers but the first have N_{ch} output channels. The two networks' topologies are listed in Table 4.2. The final, fully ternarized network consists only of convolutional layers with no bias, ternary activations (4.2) and pooling layers. Consider a layer stack consisting of a convolutional layer with N_i/N_o input/output channels respectively and an optional pooling layer and denote the ternary values as

	Layer	Out Ch.	Out Res.	Pad	Dilation
2D CNN	Input	C_{in}	64×64	N/A	N/A
	C2D (3×3)	32	64×64	S	1
	MP (2×2)	32	32×32	V	1
	C2D (3×3)	N_{ch}	32×32	S	1
	MP (2×2)	N_{ch}	16×16	V	1
	C2D (3×3)	N_{ch}	16×16	S	1
	MP (2×2)	N_{ch}	8×8	V	1
	C2D (3×3)	N_{ch}	8×8	S	1
	MP (2×2)	N_{ch}	4×4	V	1
	C2D (3×3)	N_{ch}	2×2	V	1
	MP (2×2)	N_{ch}	1×1	V	1
1D TCN	Input	N_{ch}	N_{TCN}	N/A	N/A
	C1D (2)	N_{ch}	N_{TCN}	CS	1
	C1D (2)	N_{ch}	N_{TCN}	CS	2
	C1D (2)	N_{ch}	N_{TCN}	CS	4
	C1D (N_{TCN})	11	1	V	1

TABLE 4.2
TOPOLOGY OF THE PROPOSED HYBRID CNN ARCHITECTURE, CONSISTING OF A 2D CNN FOR SHORT-TERM FEATURE EXTRACTION AND A 1D TCN TO CAPTURE LONGER-TERM TEMPORAL DEPENDENCIES. **C{1,2}D**: 1/2-DIMENSIONAL CONVOLUTION, **MP**: MAX-POOLING. **(C)S/V** INDICATE (**c**AUSAL) **s**AME/**v**ALID PADDING, RESPECTIVELY.

$\mathcal{T} \triangleq \{-1, 0, 1\}$. The layer stack takes as input a tensor $\mathbf{X} \in \mathcal{T}^{N_i \times H_X \times W_X}$, where H_X/W_X are the spatial dimensions. $\mathbf{X}$ is convolved with the convolutional weights $\mathbf{W} \in \mathcal{T}^{N_o \times N_i \times k \times k}$ and pooled, to yield pre-activations $\mathbf{Z} \in \mathbb{Z}^{N_o \times H_Y \times W_Y}$, shown in Equation 4.1. $\mathbf{Z}$ is then mapped to ternary activations $\mathbf{Y} \in \mathcal{T}^{N_o \times H_Y \times W_X}$ by channel-wise thresholding as shown in Equation (4.2) with $\mathbf{t}^{lo}$ and $\mathbf{t}^{hi},\ \mathbf{t}^{lo}, \mathbf{t}^{hi} \in \mathbb{Z}^{N_o}$.

$$\mathbf{Z} = pool\left(\mathbf{X} * \mathbf{W}\right) \tag{4.1}$$

$$y_{i,x,y} = \begin{cases} -1, & z_{i,x,y} < t_i^{lo} \\ 0, & t_i^{lo} \leq z_{i,x,y} < t_i^{hi} \\ 1, & z_{i,x,y} \geq t_i^{hi} \end{cases} \tag{4.2}$$

4.4.3 Quantization-Aware Training

To train the ternarized networks we evaluate, we perform QAT using two algorithms: INQ [47] and TQT [59]. To determine the impact of the activation function used to train the network, we compare hard hyperbolic tangent (HtanH)-activated networks to ReLU-based ones. Note that both types of FQ networks are ultimately converted to fully ternarized networks as described in Section 4.4.2. QAT is performed in 4 steps (3 in the case of HtanH activations):

1. Training full-precision network to convergence,
2. quantization of activations,
3. (incremental) weight quantization, and, if ReLU activations were used,
4. fine-tuning to compensate for non-zero padding.

For all training algorithm and activation function combinations, we train networks with BN layers inserted after the convolutional layers to allow for channel-wise scaling. After full-precision training has converged, HtanH and ReLU activations of layer l are replaced with symmetrical and asymmetrical 3-level step functions (4.3) and (4.4), respectively.

$$A^l_{symm}(x) = \begin{cases} \varepsilon^l_A, & x \geq \varepsilon^l_A/2 \\ 0, & -\varepsilon^l_A/2 \leq x < \varepsilon^l_A/2 \\ -\varepsilon^l_A, & x < -\varepsilon^l_A/2 \end{cases} \tag{4.3}$$

$$\begin{aligned} A^l_{asymm}(x) &= \begin{cases} 2\varepsilon^l_A, & x \geq 3/2\varepsilon^l_A \\ \varepsilon^l_A, & \varepsilon^l_A/2 \leq x < 3/2\varepsilon^l_A \\ 0, & x < \varepsilon^l_A/2 \end{cases} \\ &= A^l_{symm}(x - \varepsilon^l_A) + \varepsilon^l_A \end{aligned} \tag{4.4}$$

The network is retrained for a few epochs before weight quantization is applied. When using INQ, weight and activation quantization step sizes are kept constant at 1, i.e., $\varepsilon_W = \varepsilon_A = 1$. Weights are frozen to the nearest values in $\mathcal{T}$ incrementally in order of decreasing magnitude, and the unmodified STE is used to propagate gradients through quantized activations. After each freezing step, the network's

remaining learnable parameters (free weights and BN parameters) are retrained to convergence. When using TQT, both weight and activation clipping bounds are learned, with weight clipping bounds of each layer learned individually for each output channel. When quantizing weights with TQT, fake-quantization is applied to all weights simultaneously. The detailed training parameters are listed in Section 4.6.1.

4.4.4 Network Ternarization

To be deployed on CUTIE, FQ networks must be mapped to fully ternarized models containing only ternary parameters and thresholding activation functions of the form (4.2). For FQ networks using asymmetrical (ReLU) activations (4.4), this requires the replacement of those activations by symmetrical ones (4.3). (4.4) is equivalent to (4.3) shifted by ε_A^l in both the argument and the output. The shift of the argument translates to shifted thresholds, while the shift in the output is equivalent to a bias of ε_A^l being added to the next layer's input, which can be propagated through the convolution operation to an equivalent output bias. However, this transformation would require padding the edge regions of the input feature map to layer $l+1$ with $-\varepsilon_A^l$ (FQ)/-1 (integerized), which CUTIE does not support. We compensate for this by fine-tuning the network after fake-quantization with padding values of $+\varepsilon_A^l$, which corresponds to the zero-padding applied by CUTIE on the transformed network.

To generate the fully integerized network, all floating-point parameters of an activation-convolution-BN(-pooling)-activation layer stack are folded into channel-wise integer thresholds of the ternary activation layer, terminating the stack. Consider the l-th stack with (FQ) convolutional weights $\mathbf{W}^l = \left[\mathbf{W}_0^l, ..., \mathbf{W}_{N_o-1}^l\right]$ and bias $\mathbf{B}^l$, input and output activation quanta ε_A^{l-1} and ε_A^l (as the input data is already ternarized, $\varepsilon_A^0 = 1$), channel-wise weight quanta $\boldsymbol{\varepsilon}_W^l$ and N_O^l output channels. We fold the convolutional bias and the BN parameters the into channel-wise affine transformation parameters $\hat{\boldsymbol{\gamma}}^l$ and $\hat{\boldsymbol{\beta}}^l$, shown in Equation (4.5). From these, we can compute the integer threshold vectors $\hat{\mathbf{t}}^{lo,l}$ and $\hat{\mathbf{t}}^{hi,l}$ for layer l as in Equation (4.6). Because negative entries of $\hat{\boldsymbol{\gamma}}$ flip the inequalities in (4.2), we sign-invert the corresponding output channels' weights and flip their thresholds before integerizing them, respectively (4.7, 4.8). In the final network, $\mathbf{W}^l$

is replaced with its integerized counterpart $\tilde{\mathbf{W}}^l$ and BN-activation sequences are replaced with a single channel-wise integer thresholding activation of the form (4.2).

$$
\begin{aligned}
1_R &= \begin{cases} 1 & \text{Net uses ReLU activations} \\ 0 & \text{Otherwise} \end{cases} \\
\hat{B}_c^l &= \begin{cases} B_c^l & l = 1 \\ B_c^l + 1_R \varepsilon_A^{l-1} \sum_i W_{c,i}^l & l > 1 \end{cases} \quad \forall\ 0 \le c < N_o \\
\hat{\boldsymbol{\beta}}^l &= \boldsymbol{\gamma}^l \frac{(\hat{\mathbf{B}}^l - \boldsymbol{\mu}^l)}{\boldsymbol{\sigma}^l} + \boldsymbol{\beta}^l, \qquad \hat{\boldsymbol{\gamma}}^l = \frac{\boldsymbol{\gamma}^l}{\boldsymbol{\sigma}^l} && (4.5) \\
\hat{\mathbf{t}}^{lo} &= \frac{(1_R - 0.5)\varepsilon_A^l - \hat{\boldsymbol{\beta}}}{\hat{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_W^l \varepsilon_A^{l-1}}, \qquad \hat{\mathbf{t}}^{hi} = \frac{(1_R + 0.5)\varepsilon_A^l - \hat{\boldsymbol{\beta}}}{\hat{\boldsymbol{\gamma}} \boldsymbol{\varepsilon}_W^l \varepsilon_A^{l-1}} && (4.6) \\
\tilde{\mathbf{W}}_c^l &= \left(\mathbf{W}_c^l - 2\mathbf{W}_c 1_{\gamma_c < 0}\right) / \varepsilon_{W,c}^l \qquad \forall\ 0 \le c < N_o && (4.7) \\
t_c^{lo,hi} &= \left\lceil \hat{t}_c^{lo,hi} - 2\hat{t}_c^{lo,hi} 1_{\gamma_c < 0} \right\rceil \qquad \forall\ 0 \le c < N_o && (4.8)
\end{aligned}
$$

Integerization for deployment on the PULP cluster is more straightforward and follows the procedure laid out in Section 2.2.3. The ternarized weights are mapped to 2-bit integers, and activation operations are mapped to requantization layers (also called *Integer Channel Normalization* in [91]). The l-th requantization layer folds ε scaling and BN into a single channel-wise affine transformation with parameters $\tilde{\boldsymbol{\gamma}}^l$ and $\tilde{\boldsymbol{\beta}}^l$ followed by an arithmetic right shift by a layer-wise parameter D_l, effectively performing fixed-point arithmetic with integer operations. The result is finally clipped to the appropriate integer range, implementing the nonlinear activation function. Equation (4.9) shows the ternary requantization operation for layer l (assuming a ReLU activation) and Equations (2.14, 2.15) show how $\hat{\boldsymbol{\gamma}}^l$ and $\hat{\boldsymbol{\beta}}^l$ are computed. Note that by adding 0.5 to the folded bias term, the flooring operation in (4.9) is converted into a rounding operation.

$$
RQ^l(\boldsymbol{X}) = clip\left(\lfloor\left(\left(\hat{\boldsymbol{\gamma}}\boldsymbol{X} + \hat{\boldsymbol{\beta}}\right) >> D_l\right)\rfloor, 0, 2\right) \tag{4.9}
$$

While thresholding-based activations are completely equivalent to their fake-quantized equivalents, the requantizing activation (4.9)

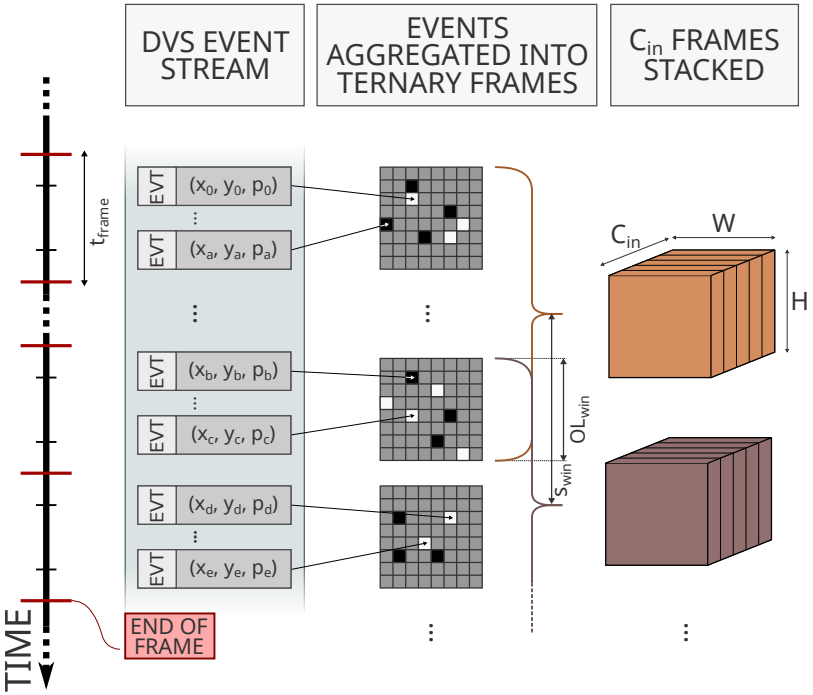


Figure 4.2 – Frame aggregation for processing by the 2D CNN. All events occurring during a time window of length t_{frame} are assigned to the same frame. The 2D CNN takes C_{in} frames as input, and two successive stacks of C_{in} frames overlap by OL_{win} frames

.

introduces some numerical mismatches due to the rounding of $\hat{\boldsymbol{\gamma}}$ and $\hat{\boldsymbol{\beta}}$. Higher values of D decrease the discrepancy, so D is chosen as large as possible while avoiding integer overflows resulting from the multiplication with $\mathbf{X}$ in (4.9). In our implementation, we choose $D = 19$ to achieve equivalent integerized classification accuracy to that of the FQ network.

4.4.5 Data Preparation

Unlike previous works [70], [116], which employ fully event-based processing schemes and rely on SNNs to classify event data directly, our proposed approach first aggregates the event stream into ternary 2D images. This incurs some overhead for data preparation (frames need to be assembled and buffered) and decouples the processing energy from the density of the event stream. In practice, however, the power required for data preparation is vanishingly small in comparison to the idle power of the running system (see Section 4.6.2). The lack of fine-grained energy-activity proportionality is more than compensated by the unparalleled efficiency of TNN-based processing on CUTIE, which is indeed owed to the regularity of DNNs. Finally, activity-energy proportionality is still supported in a coarse-grain manner by only triggering inference when the input event density detected by the DVS peripheral surpasses a programmable threshold.

Parameters in data generation

The process of generating frame data from the event stream, illustrated in Figure 4.2 has several degrees of freedom:

- Frame rate FPS, or, equivalently, frame time $t_{frame} = \frac{1}{FPS}$
- CNN input window size C_{in}; this parameter dictates the number of input channels to the CNN's first layer
- temporal CNN input window stride s_{win}, which gives rise to the input window overlap $OL_{win} = C_{in} - s_{win}$, and
- downsampling factor D - from our experimental observations, we choose $D = 2$, i.e., the height and width of the original frame are both halved, and the resulting frame has 1/4 the resolution of the raw camera data.

While all of these parameters must be fixed for network training, the DVS peripheral performing the frame aggregation has been designed to leave them configurable at runtime at negligible hardware overhead. The receptive time interval t_p of the full hybrid network is given by the number of input vectors to the TCN N_{TCN}, the number of frames

encoded in a single vector C_{in} and the stride of the CNN input windows s_{win} as $t_p = C_{in} + (N_{TCN} - 1)s_{win}$. Once running, the latency l of the system to react is determined by the window stride and the frame rate, as well as the processing time t_{inf}: $l = s_{win}/FPS + t_{inf} \approx s_{win}/FPS$. When running networks on CUTIE, we can approximate $t_{inf} \approx 0$ due to CUTIE's very high throughput compared to the frame time (see Section 4.6.2).

4.5 SoC Architecture: Kraken

We deploy the trained hybrid TCNs on the *Kraken* PULP SoC [160]. Kraken has three main processing domains, each individually power-gateable: The *SoC domain*, the *cluster domain* and the *accelerator domain*.

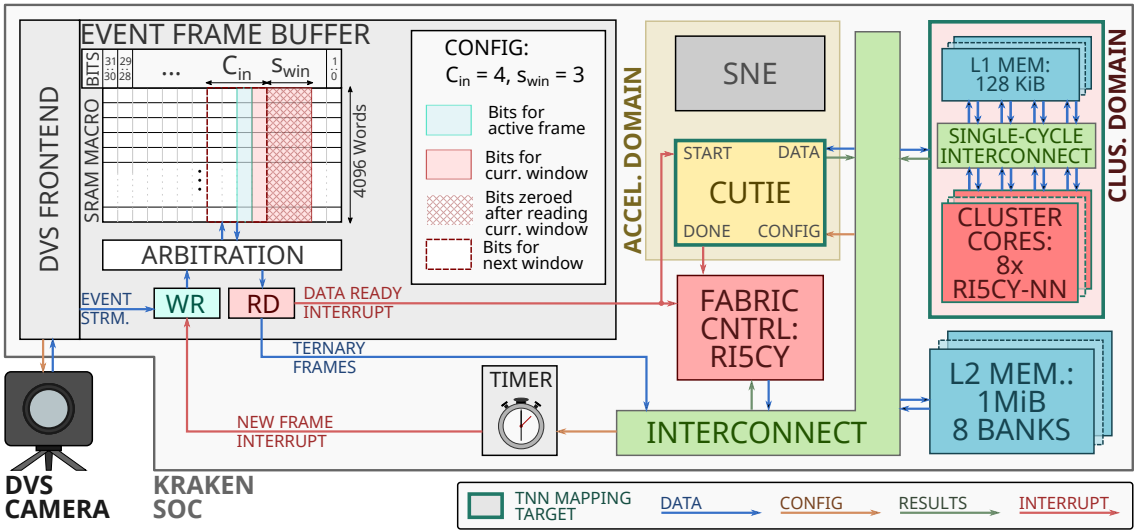


Figure 4.3 – Overview of Kraken SoC with detailed illustration of frame buffer operation. Ternary frames aggregated from DVS events are stored in a frame buffer. After s_{win} frames have been received, C_{in} frames are written to CUTIE's internal memories or L2 memory. From there, they can be processed by CUTIE or the PULP cluster (green outlines indicate processing units to which the TNN can be mapped). When running the TNN on CUTIE, the only interaction required from the fabric controller (FC) core is readout and interpretation of the class scores.

The SoC domain contains a single RI5CY core, implementing the XpulpV2 extension [96] to the RISC-V ISA. This core is designated as the *FC* and is responsible for orchestrating system operation and managing peripherals. The SoC domain contains 1 MiB of SRAM-based L2 memory, which serves as the main working memory of Kraken. A wide range of peripherals for off-chip communication, including a DVS camera interface (described in detail in Section 4.5.1), are also located in the SoC domain and are managed by the *µDMA* engine [161], which allows operation of peripherals with minimal involvement of the FC core. Off-chip peripherals are powered by the same supply rail as the FC and the L2 memory but are clocked by a separate frequency-locked loop (FLL) clock generator.

The cluster domain contains a PULP cluster of 8 RI5CY cores to perform compute-intensive processing tasks. Additionally to the XpulpV2 extension, the cluster cores implement the XpulpNN extension [97]. XpulpNN offers optimized hardware support for low-precision arithmetic via custom MAC-and-load instructions. These instructions combine low-precision MAC operations on packed 2-bit, 4-bit, or 8-bit operands with memory access and pointer update operations. This approach greatly mitigates the von Neumann bottleneck of load-store ISAs for matrix multiplication kernels, enabling the efficient execution of low-precision neural network layers. To minimize memory access overhead, the cluster has 128 KiB of L1 tightly-coupled data memory (TCDM), which is divided into 16 interleaved banks and provides single-cycle access to temporary data.

Finally, the accelerator domain contains two application-specific accelerators: CUTIE, targeting TNN inference and SNE [162], an all-digital SNN accelerator. Each accelerator can be clock-gated and power-gated individually. For Kraken, CUTIE has been modified from its first-generation architecture and configuration by changing the maximum number of input and output channels to 96, the inclusion of an additional memory bank to store ternary CNN output vectors and the capability to process those vectors with TCN layers by allowing for 1-D kernels with configurable dilation and causal padding.

4.5.1 DVS Interface

With the DVS interface (DVSI), Kraken possesses a versatile peripheral unit to interface with DVS cameras such as the DVS132S [118]. Its task is to read event data from the camera unit and write it to a configurable memory address. Event readout can be triggered by writing to a configuration register or by an on-chip timer/counter peripheral. As the DVS peripheral is implemented as a µDMA peripheral, it can operate without any intervention from the FC. The peripheral can write event data to the configured location in two different formats. For processing by algorithms operating on discrete events (e.g., SNNs), it can encode each event in a 32-bit word and write event data to consecutive memory addresses. Alternatively, the peripheral can buffer up to 15 event frames in the format required by our proposed TCN. For this purpose, it contains an SRAM macro of 4096×32 bits, which is used as a frame buffer. Each word corresponds to one pixel of a 64×64 feature map, with each channel's value occupying 2 bits in a word for NHWC ordering. An incoming event at location (y, x) is written to bits $[2c_{curr} + 1, 2c_{curr}]$ at address $64y + x$, where c_{curr} is a wrapping counter ranging from 0 to 15, indexing the bits at which the currently active frame is stored. As a new frame interval starts, c_{curr} is incremented. The frame buffer is configurable in C_{in} and s_{win}. After s_{win} frames have been written into the buffer, the readout logic streams out the most recent C_{in} frames by reading each word, circularly shifting it to the right by $2c_{curr}$ bits and masking bits $[31 : 2C_{in}]$ of the resulting word to zero before writing it to the configured memory location. After reading a word, the s_{win} timesteps no longer used in the next input window are zeroed to avoid contamination of future frames by old events. The selective writing of events to bit indices and zeroing of stale data in single write operations is made possible by bit-selection signals exposed by the memory macros used in Kraken.

As the DVS peripheral can be configured to write to any address in Kraken's memory range, it can transfer input data directly to CUTIE's internal activation memories. After an input window has been streamed out, the peripheral raises an interrupt line which is connected both to the FC and CUTIE, allowing autonomous operation of the entire processing pipeline by directly triggering CUTIE's inference without redundant data transfers or other intervention by the FC

core. After CUTIE has finished running the TNN on the input data, it raises an interrupt line and the FC can process and interpret the network's output.

4.5.2 Mapping TNNs on Kraken

The two processing domains we target for the deployment of our networks are the PULP cluster and CUTIE. In both cases, networks are executed sequentially in a layer-by-layer fashion. To efficiently use the cluster's computational resources, the data on which the cores operate must be stored in the high-bandwidth L1 memory. However, Kraken's L1 memory is too small to hold the inputs, outputs, and weights, necessitating *tiled* execution of layers: the complete input, weight, and output buffers for a layer are held in L2 memory and the layer is divided into smaller execution units by tiling the input, output, and weight tensors along the input channel, output channel or spatial dimensions. For the execution of each tile, the corresponding inputs and weights are transferred by direct memory access (DMA) from L2 to L1 memory and the cluster computes the partial output, which is then transferred back to L2. Double buffering is used to hide the latency of the DMA transfers.

In contrast, CUTIE specializes in efficient processing of TNNs by implementing dedicated activation and weight memories, which offer sufficient memory to avoid tiling of the network. In order to maximize energy efficiency of the system, inferences are executed on the accelerator in a self-contained fashion. To prepare for network execution, weights are written to CUTIE's internal weight memory and parameters for each layer (e.g., kernel size, input size, stride, etc.) are configured, both via CUTIE's external data interface. A ternary input tensor can then be written to CUTIE's activation memory. Inference is triggered by writing to a configuration register or if CUTIE's start interrupt line is raised. CUTIE executes networks autonomously on its internal memories and raises an interrupt when inference has concluded, allowing the host system to read and interpret the results.

Parameter	Value
Epochs	50
Batch Size	128
Opt.	Adam
LR_0	0.01
LR decr. [a]	Cosine Annealing [163]
$E_{Q,Wt}/E_{Q,Act}$ [b]	0/8
Act. clip init. [c]	Const. 6.0

[a] We perform one cycle of cosine annealing over 50 epochs.
[b] Activations and weights are quantized starting from the specified epoch
[c] Const. x: clipping bounds initialized to x

Table 4.3
Hyperparameters for QAT with TQT

4.6 Results

In this section, we present the results of our algorithmic evaluations as well as power measurements conducted on the Kraken system. First, we show the impact of data preparation, architectural, and training parameters on the network's classification accuracy. Then, we show power consumption and latency measurements for networks mapped both to CUTIE and to Kraken's PULP cluster.

4.6.1 Network Design, Data Preparation and QAT Algorithms

We evaluate the impact of network design, training, and data preparation parameters on statistical accuracy from multiple aspects. The hyperparameters for QAT with TQT are shown in Table 4.3, the training schedule for QAT with INQ is shown in Figure 4.4.

Receptive Time Interval t_p

The receptive time interval t_p is tied to the network's reaction time to changes in the input: For larger t_p, it will take longer for the new inputs to propagate through the network and saturate the TCN's input

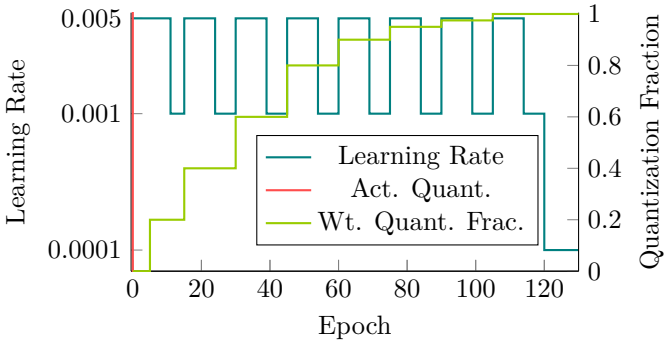


Figure 4.4 – Quantization and learning rate schedules used to ternarize full-precision networks with the INQ algorithm. The left y-axis measures the learning rate, the right y-axis measures the fraction of weights quantized. Hyperparameters not shown in this plot are identical to those used for QAT with TQT.

window. To minimize the system's reaction time, it is thus desirable to keep t_p as short as possible. However, a very short t_p limits the temporal context of the network's input, which may negatively impact classification accuracy. In this context, we consider a configuration *Pareto-optimal* if there is no network that achieves higher accuracy with a shorter t_p, and the set of Pareto-optimal networks forms a *Pareto front*. As described in Section 4.4.5, t_p is influenced both by network design (C_{in}, N_{TCN}) and data preparation parameters (s_{win}, FPS). Setting $N_{ch} = 96$, we trained 47 networks with parameter combinations resulting in t_p ranging from 42 ms to 1433 ms. We trained full-precision networks on the dataset split used by most previous works (users 1-23 in the training set and users 24-29 as the validation set). Of the full-precision networks forming the Pareto front, we also trained ternarized versions using the TQT algorithm and ReLU activations using 4-fold cross-validation (CV). The results are shown in Figure 4.5 for both the standard dataset split and with 4-fold CV, selecting different users as the validation set. From Figure 4.5, we can observe multiple points of interest. First, at $t_p = 300$ ms, a ternarized network already reaches 96.86 % validation accuracy on the standard dataset split, only 0.74 percentage points below the maximum observed accuracy of 97.7 %.

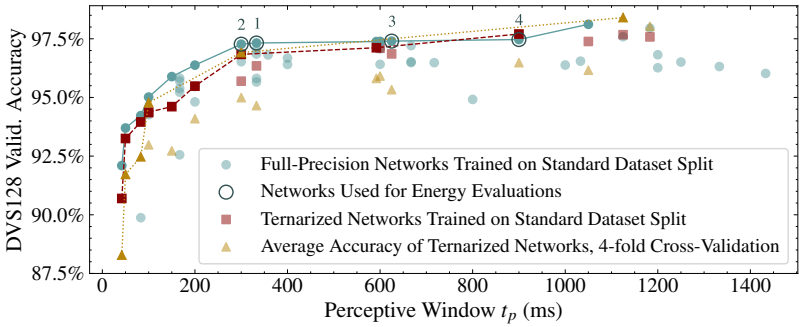


Figure 4.5 – Validation accuracy vs. t_p for various network architecture and dataset generation parametrizations. Data points connected by lines form the Pareto front of the respective experiments.

The accuracy drop from ternarization is below 1.5 percentage points in all evaluated networks, with an average drop of 0.6 percentage points. Quantization even results in increased accuracy in some cases, confirming that the problem is well-suited to be solved by TNNs. Lastly, the average statistical accuracy when using 4-fold CV is lower by 1.55 percentage points than with the standard dataset split, indicating that the standard split is easier, with the training set representing the validation set accurately.

Network Design, Dataset Generation and QAT Algorithms

As a fully exhaustive network design, training, and dataset generation parameter search would be infeasible, we present ablation results over those parameters and design choices we found to have the largest impact on statistical accuracy. Figure 4.6 shows the impact of varying FPS while maintaining $t_p = 900\,\text{ms}$ by changing C_{in}. We observed that increasing framerates significantly improves classification accuracy only up to 60 FPS, after which it stagnates between 97.3% and 97.9%. Next, we sought to determine the impact of the network's TCN stage on statistical accuracy by training two of the Pareto-optimal networks shown in Figure 4.5 ($t_p = 300\,\text{ms}$ and $t_p = 900\,\text{ms}$) without the TCN. Instead, the N_{TCN} outputs of the CNN are fed directly into a linear classifier layer. Table 4.4 shows that the inclusion of the TCN

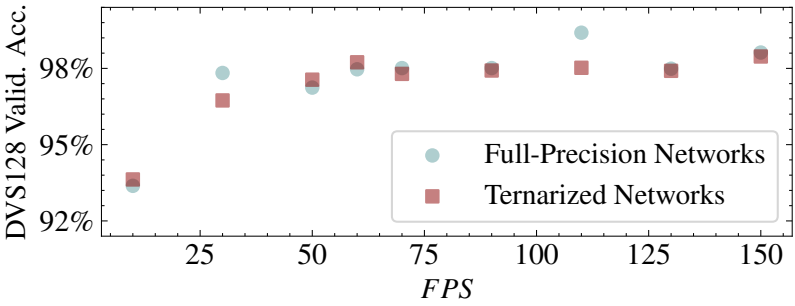


Figure 4.6 – Validation accuracy vs. FPS. Apart from FPS and C_{in}, the network's parametrization corresponds to network 4 from Table 4.6 with C_{in} chosen so that $t_p = 900\,\text{ms}$, i.e., $C_{in} = FPS/10$. The highest accuracy is achieved at $150FPS$ with 97.9 %

stage improves classification accuracy by 1.5 and 1.4 percentage points respectively. Finally, we evaluated the impact of the choice of QAT algorithm and activation function on accuracy. As Table 4.4 shows, the combination of ReLU activations and TQT yields the highest classification accuracy. Furthermore, ReLU activations and TQT outperform HtanH and INQ individually. To determine the impact of the number of quantization levels on accuracy, we also trained BNNs with equivalent architectures using ReLU activations and the TQT algorithm. Since our DVS event frame representation is ternary, the first layer of each BNN processes ternary inputs but uses binary weights. As Table 4.4 shows, these networks still perform well but achieve lower statistical accuracies by 1.0 and 0.7 percentage points than their ternary counterparts.

4.6.2 End-to-End Gesture Recognition on the Kraken SoC

We evaluated the real-world energy consumption of our processing pipeline on the Kraken SoC with the four network parametrizations shown in Table 4.6. On both CUTIE and the PULP cluster, we run inference under realistic operating conditions by matching the inference rates to the evaluated networks' training and accounting for the power

Net (t_p)	n_{lvls}	Act.	QAT	TCN?	Acc. Quant. (FP)
Net 1 (300 ms)	3	ReLU	TQT	✗	95.3 % (96.9 %)
	3	ReLU	TQT	✓	**96.8**% (97.3%)
	3	ReLU	INQ	✓	93.9% (97.3%)
	3	HtanH	TQT	✓	95.5% (95.5%)
	3	HtanH	INQ	✓	93.8% (95.5%)
	2	ReLU	TQT	✓	95.8 % (97.3 %)
Net 4 (900 ms)	3	ReLU	TQT	✗	96.3% (96.9%)
	3	ReLU	TQT	✓	**97.7**% (97.5%)
	3	ReLU	INQ	✓	97.2% (97.5%)
	3	HtanH	TQT	✓	96.4% (96.1%)
	3	HtanH	INQ	✓	95.5% (96.1%)
	2	ReLU	TQT	✓	97.0 % (97.5 %)

TABLE 4.4
IMPACT OF QAT ALGORITHM, QUANTIZATION LEVELS AND THE TCN STAGE ON CLASSIFICATION ACCURACY. CLASSIFICATION ACCURACY IS SPECIFIED FOR TERNARIZED NETWORKS TRAINED ON THE STANDARD DATASET SPLIT WITH FULL-PRECISION ACCURACY IN PARENTHESES. n_{lvls} DENOTES THE NUMBER OF QUANTIZATION LEVELS, WITH 2 YIELDING A BNN AND 3 A TNN. FOR THE FULL PARAMETRIZATION OF THE NETWORKS, SEE TABLE 4.6.

consumption of idle domains. Our results show that the CUTIE-based implementation of our TCN network requires only 7 µJ, 58.5 × less than a comparable implementation on the state-of-the-art RISC-V cluster using efficient, specialized ISA extensions.

Experimental Setup

To map the network to the PULP cluster, we use DORY [105], an open-source DNN mapping utility targeting PULP systems, to generate test applications. The network layers are mapped to 2-bit kernels from the PULP-NN library [164] which take advantage of the XpulpNN ISA extension. As the DVSI's operation does not result in a measurable increase in power consumption, our test applications store input samples in L2 memory. An on-chip timer generates interrupts to the FC at the rate appropriate for each tested network, upon which the FC triggers inference either on the cluster or on CUTIE. As CUTIE operates on data from its internal activation memory, the FC transfers

	FC	Cluster	CUTIE
f_{clk} (MHz)	40	115	15
V_{DD} (V)	0.55	0.55	0.5
P_{idle} (mW)	2.0	1.6	2.7
P_{inf} (mW)	2.0	21.1	5.3
t_{inf} (ms)	N/A	17.8	0.9

TABLE 4.5
OPERATING CONDITIONS FOR POWER MEASUREMENTS AND POWER CONSUMPTION OF THE DIFFERENT POWER DOMAINS ON THE KRAKEN SOC

input activations before triggering the computation, which is slower than direct transfer by the DVSI and results in pessimistic latency and energy estimations. Table 4.5 details the operating conditions for our experiments, the power consumption of each domain during each phase of inference, and the inference latency t_{inf}. While the tested networks differ in the values of both N_{TCN} and C_{in}, this has no measurable impact on inference latency. Due to CUTIE's fully unrolled architecture, the number of input channels does not influence latency. For cluster networks, the number of input channels must be padded to a multiple of 16 to comply with the constraints of XpulpNN instructions. The differences in computation load caused by varying N_{TCN} are so small as to be unmeasurable. During idle phases, the processing units (PULP cluster or CUTIE) are clock-gated, but not power-gated. We also correct our power numbers for the leakage current drawn by oversized power gates for the accelerators in our design by measuring the leakage current with both accelerators power-gated and subtracting it from our current measurements. This makes our reported results equivalent to those of a design without any power-gating features. As the FC, cluster and accelerator domains have separate supply rails, unused domains can be turned off by switching off their external power supplies and we calculate the total power consumption as the summed power draw of the used domains.

Net	t_p (ms)	FPS	C_{in}	N_{TCN}	inf./s	Acc.	E_{inf} (mJ)		P_{cont} (mW)		E_{win} (mJ)	
							Cluster	CUTIE	Cluster	CUTIE	Cluster	CUTIE
1	333	60	4	5	15	96.3%	0.41	**0.007**	8.9	4.68	**2.74**	**1.26**
2	**300**	120	4	9	30	96.8%			14.1	4.74	4.08	1.27
3	625	120	15	5	8	96.9%			**6.4**	4.68	3.59	2.35
4	900	60	6	9	10	**97.7**%			7.1	4.69	6.02	3.75

TABLE 4.6
KEY POWER AND ENERGY FIGURES FOR 4 PARETO-OPTIMAL NETWORKS FROM FIGURE 4.5. THE NETWORKS WERE MAPPED TO BOTH THE PULP CLUSTER AND CUTIIE.

Power Measurements

Table 4.6 reports the key figures of merit of the four networks we evaluated on the Kraken SoC. The inference energy E_{inf} is given as the total energy consumed by the FC and CUTIE or the PULP cluster for a single prediction update, i.e., a single inference of the complete network. A complete inference on the PULP cluster takes 17.8 ms and consumes less than 500 µJ. Running the network on CUTIE reduces these figures further to 0.9 ms and 7 µJ, including the inefficient transfer of input data by the FC. This represents an improvement of 5× over the closest result reported in literature while accounting for the complete power consumption of the processing system including data transfer, rather than only the processing cores. Compared to previous end-to-end systems, our approach achieves a 67× lower inference energy. During inference, the operational efficiency of the system is 363 GOp/J when mapping the network to the cluster and 21 TOp/J for CUTIE-mapped networks. Figure 4.7 shows the power traces from repeated inferences on two of the evaluated networks.

To evaluate the efficiency of our processing pipeline in real-world applications, we consider two scenarios. The first is that of always-on inference at the rate for which the network was trained (inf./s in Table 4.6). In this scenario, the metric of interest is P_{cont}, the processing system's average power consumption while performing continuous inference. Table 4.6 shows that the cluster and CUTIE implementations behave very differently in this respect: While P_{cont} strongly correlates with the inference rate for cluster-mapped networks, it is dominated by the idle power consumption for CUTIE-mapped networks. With higher inference rates, the efficiency advantage of CUTIE-mapped networks grows. While gesture recognition does not benefit from high inference rates, CUTIE's extremely high throughput makes it optimal for low-latency applications (e.g., perception pipelines for fast-flying nano-drones). Higher inference rates would allow it to amortize its leakage power consumption by increasing the utilization of its extremely efficient datapath.

The second scenario is event-triggered inference: The system is put to sleep until it is awakened by, e.g., increased event activity (which can be detected by Kraken's DVS peripheral). Upon awakening, a complete window of N_{TCN} inferences must be completed to produce a

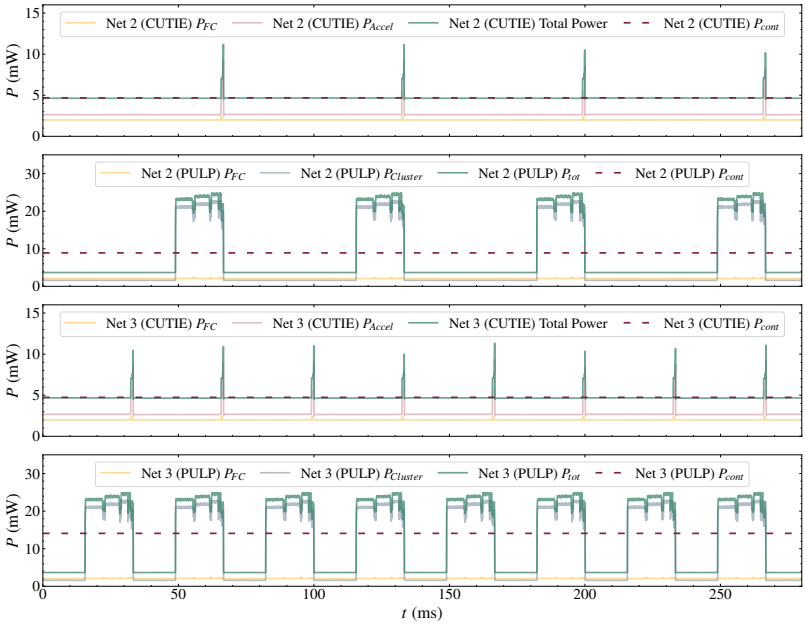


Figure 4.7 – Power traces for networks 2 and 3. Measurements for CUTIE implementations are shown in the first and third rows, cluster-mapped implementations are shown in the second and fourth rows. The differences in peak power for different inferences on CUTIE are a result of our power analyzer's temporal resolution, which is lower than the duration of the shortest current spikes.

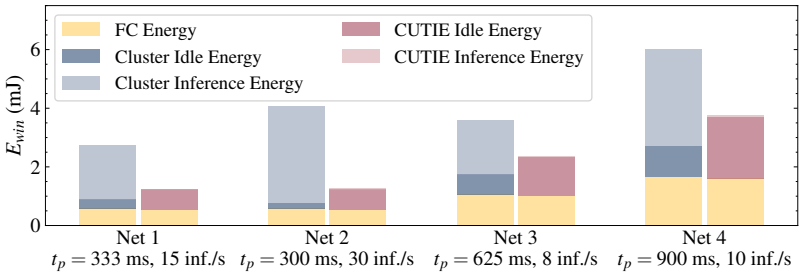


Figure 4.8 – Breakdown of E_{win} into contributions from FC, idle and inference energies. CUTIE-mapped nets exhibit very low inference energy with larger contributions from idle time, while cluster-mapped networks see large contributions from inference.

reliable prediction. We calculate the energy required to compute this prediction as $E_{win} = N_{TCN}E_{inf} + (N_{TCN} - 1)(\frac{1}{(inf/s)} - t_{inf}) \times P_{idle}$. Consequently, shorter t_p reduces the idle energy contribution and lower inference rates reduce the computation energy contribution. As with P_{cont}, idle energy dominates CUTIE networks' E_{win} and CUTIE performs best on networks with short t_p, while cluster networks' energy consumption rises sharply with N_{TCN} at similar t_p, as seen on networks 1 and 2. Figure 4.8 visualizes the breakdown of E_{win} for the four evaluated nets mapped to CUTIE and the PULP cluster.

To determine the individual contributions of the components of our processing pipeline to the overall energy efficiency, we implemented the event-frame conversion in software. The inference energy breakdown of Network 1 mapped to the cluster and to CUTIE, using the software frame buffer and the hardware buffer in the DVSI peripheral, is shown in Figure 4.9. CUTIE's inference energy is two orders of magnitude lower and the total inference energy is reduced again by almost 2× by using the hardware frame buffer.

4.7 Conclusion

In this chapter, we have presented an end-to-end pipeline for frame-based gesture recognition from DVS camera data, implemented on the Kraken SoC. A dedicated on-chip peripheral aggregates the event

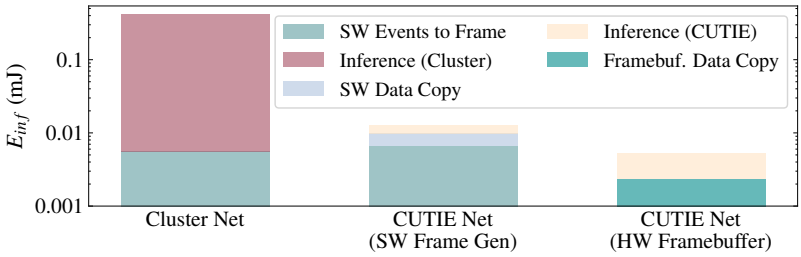


Figure 4.9 – Breakdown of inference energy for network 1 mapped to the PULP cluster as well as to CUTIE, using both a software-based event-to-frame mapping and the hardware frame buffer in the DVS peripheral. Note the logarithmic y-axis, causing the equal energy consumption of CUTIE inference to appear different visually.

stream from the DVS camera into ternary event frames. We classify the event frames with a fully ternarized hybrid TCN mapped either to the CUTIE accelerator or to the 8-core PULP cluster. To the best of our knowledge, our network sets a new state of the art for embedded implementations with 97.7 % validation accuracy on the DVS128 gesture dataset, and the most accurate network we trained achieves 97.9 %. On the CUTIE accelerator, we achieve a classification energy of 7 µJ, 67× lower than the previous state of the art for end-to-end gesture recognition at an inference latency of 0.9 ms. We further show that our approach can perform competitively even on software-programmable RISC-V cores with ISA extensions for sub-byte arithmetic. The cluster implementation exhibits an inference energy of 0.41 mJ at a latency of 17.8 ms for the same network running on Kraken's PULP cluster. With a continuous classification power consumption of 4.7 mW (CUTIE)/6.4 mW (cluster) for all involved processing components at 96.6 % classification accuracy, our implementation highlights the added value of complete integration of sensor interface, preprocessing and efficient compute units. Kraken's programmability and flexibility allow for the implementation of a variety of processing scenarios (e.g., split execution of mixed-precision networks between CUTIE and the PULP cluster) to be explored in the future.

Chapter 5

XpulpTNN: Efficient Ternary Neural Network Inference on RISC-V-Based Edge Systems

5.1 Introduction

In Chapter 4, we have demonstrated that aggressively quantized neural networks, in particular TNNs, can enable ultra-low-power end-to-end applications on the edge. By mapping networks to a dedicated hardware accelerator such as CUTIE, throughput, latency and efficiency can be optimized beyond what is possible on even the most efficient general-purpose processing cores. The specificity of such accelerators to a single type of computation allow them to eliminate energy overheads and drives their efficiency potential. At the same time, the inclusion of such an accelerator in a system represents a substantial commitment of silicon area and implementation effort. For example, the CUTIE accelerator targeted in Chapter 4 occupies around 1/3 of the silicon area of the complete Kraken system [160]. As affordability is a core requirement for many edge systems, such overheads may not

be acceptable for system designers. nIn such cases, implementing hardware support for low- and mixed-precision arithmetic within general-purpose processing cores is an attractive alternative to single-purpose accelerators: while the operational efficiency is generally lower, the implementation overhead is greatly decreased, and the efficiency gains over the unmodified system are substantial. Consequently, several ISA extensions for low- and mixed precision integer arithmetic have been proposed and used to optimize the accuracy-energy trade-off. In Chapter 4, we used the XpulpTNN extension to the RISC-V ISA to execute the proposed TNN as a 2-bit network. While this approach already achieves state-of-the-art inference efficiency, it is not tailored to the unique properties of TNNs and provides no advantage over regular 2-bit QNNs.

To overcome this limitation, we aim at achieving efficient TNN inference on RISC-V processing cores with minimal hardware overhead, enabled by a lightweight extension of the RISC-V ISA. Specifically, we present the following contributions in this chapter:

- We propose XpulpTNN, a lightweight extension to the RISC-V ISA designed to enable efficient inference of TNNs. We further enable the development of end-to-end applications using XpulpTNN with an end-to-end software stack consisting of GCC compiler support, an optimized kernel library and an automated deployment flow.

- We implement the proposed instructions in an open-source RISC-V core targeted at energy-efficient edge AI applications and construct an 8-core compute cluster based on the modified architecture. We show that our modifications result in a negligible area overhead of $< 1\,\%$ compared to the baseline cluster with a negligible impact on the power consumption of 8-bit applications.

- We conduct detailed performance and energy efficiency evaluations of the XpulpTNN hardware and software stack. We show that our kernels increase throughput by $67\,\%$ on ternary convolutions compared to state-of-the-art 2-bit kernels. In post-layout simulations of the XpulpTNN system, we demonstrate a

marginal increase in power consumption of only 5.2 %, leading to an energy efficiency gain of 57 %.

- We evaluate XpulpTNN's impact on inference energy efficiency on two end-to-end applications. In image classification on the CIFAR-10 dataset, we demonstrate that XpulpTNN enables the deployment of TNNs that achieve up to 1.6 pp. higher classification accuracy at equal inference latency compared to 2-bit QNNs. In an 11-class gesture recognition task, we demonstrate a reduction of inference energy by 33 % at a negligible accuracy drop when comparing TNN inference using XpulpTNN to an optimized 2-bit QNN running on an equivalent system without our extension.

5.2 Background

5.2.1 Ternary Neural Networks

Ternary neural networks are QNNs where all activations and weights take values in the set $\mathcal{T} \triangleq \{-1, 0, 1\}$. A typical convolutional TNN is composed of a series of layer stacks, each consisting of a convolutional layer followed by an element-wise non-linear activation, with an optional pooling layer between convolution and activation to decrease the spatial dimension of the output feature maps. The activation layer maps the convolution/pooling layer's integer output $\mathbf{Z} \in \mathbb{Z}^{N_o \times H \times W}$ to ternary activations $\mathbf{Y} \in \mathcal{T}^{N_O \times H \times W}$ by the channel-wise thresholding function $\sigma(\cdot)$:

$$y_{i,x,y} = \sigma\left(z_{i,x,y}\right) = \begin{cases} -1, & z_{i,x,y} < t_i^{lo} \\ 0, & t_i^{lo} \leq z_{i,x,y} < t_i^{hi} \\ 1, & z_{i,x,y} \geq t_i^{hi}, \end{cases} \tag{5.1}$$

where $\mathbf{t}^{lo}, \mathbf{t}^{hi} \in \mathrm{Z}^{N_o}$ are vectors of lower and upper integer thresholds, respectively.

At first glance, TNNs are at a disadvantage in terms of network size and efficiency when compared to two-bit QNNs and BNNs: Each ternary value requires two bits of storage but only encodes

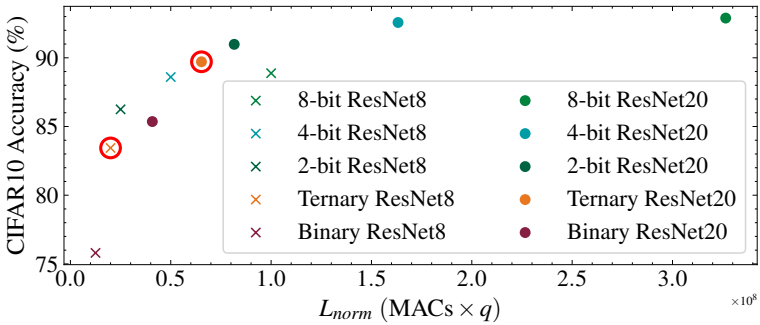


Figure 5.1 – Comparison of accuracy vs normalized load L_{norm} of ResNet8 and ResNet20, quantized to different precisions, on CIFAR10. Networks were trained with the TQT algorithm [59], with the first and last layers quantized to 8-bit precision. TNNs are highlighted in red.

$\log_2 3 \approx 1.585$ bits of information. However, by assigning each possible sequence of 5 ternary values a distinct 8-bit string, the storage space required is reduced to 1.6 bit per value, close to the theoretical optimum. In the context of dedicated accelerators, TNNs represent a particularly attractive operating point. In [14], the authors show that a well-designed ternary accelerator can achieve better inference efficiency than an equivalent BNN accelerator on the same network topology. Ternary compression partially amortizes the memory overhead over a binary design, and the addition of a zero value allows for sparsity, which directly translates to reduced switching activity in an unrolled datapath, improving energy efficiency. At the same time, classification accuracy is improved over BNNs due to the increased representational capacity of TNNs. In contrast to application-specific accelerators, ISA-based processing cores operate on fixed-width data words. This constraint on datapath design means that the 25 % increase in data density from ternary compression is crucial to maximizing the achievable performance when processing ternary data. In this paper, we choose the encoding proposed in [165] for its low-complexity hardware implementation.

Comparison to Other QNNs To achieve an optimal accuracy-efficiency trade-off for a given application, a QNN must be chosen among all model architectures *and* quantization policies supported by the target hardware architecture; Figure 5.1 illustrates such a trade-off curve for two lightweight networks, ResNet8 (from the MLPerf Tiny Benchmark [166]) and ResNet20, on the CIFAR-10 [6] dataset. We define the *normalized load* L_{norm} posed by a q-bit QNN as the number of MAC operations multiplied by q, assuming SIMD execution and inverse linear scaling of throughput with bitwidth. We set $q = 1.6$ for TNNs, assuming the use of the compression scheme described earlier. Figure 5.1 exemplifies that smaller networks like ResNet8, quantized to higher bit widths, may dominate larger BNNs in terms of accuracy and inference latency. In contrast, the ResNet20 TNN extends the Pareto front, offering an attractive operating point and demonstrating the advantages of TNNs.

5.2.2 QNN Inference on MCU-Class Platforms

To reap QNNs' efficiency potential, an inference platform must provide hardware support for their execution. Application-specific accelerators achieve the highest efficiency: BNN [102], [167] and TNN [14], [168] accelerators report efficiencies of up to 1 POp/J. Many accelerator designs for BNNs, TNNs and other low-bitwidth QNNs have been proposed, but there are fewer documented end-to-end applications deployed to such accelerators embedded in MCU-class systems. Examples include face recognition [169] and gesture recognition [9]. While they report superior energy efficiency figures, the accelerators implemented in these systems occupy a significant proportion of the total silicon area. Such a commitment of silicon resources may not be affordable for low-cost edge MCUs – indeed, most commercial edge systems only have ISA-based reduced instruction set computer (RISC) processing cores. Recent work has shown that BNNs can be efficiently executed with XOR and population count instructions [68], found in most ISAs targeting embedded and edge systems. Multiple authors have adopted this approach to implement BNN kernel libraries [92], [93] and applications such as BNN-based keyword spotting [94] on off-the-shelf MCUs without BNN-specific hardware.

In contrast to BNNs, the efficient execution of QNNs in general and TNNs in particular on ISA-based processing cores poses significant challenges. Mainstream ISAs have limited support for arithmetic on sub-word data types, severely limiting the potential efficiency gains, as data packing and unpacking has to be implemented in software. In commercial MCUs based on the ARMv8.1 ISA, native support for sub-word arithmetic has been addressed for 8-bit and 16-bit operands by packed-SIMD MAC instructions [55].

While 8-bit QNNs achieve full-precision equivalent accuracy with the use of quantization-aware training algorithms [5], [69], sub-byte and mixed-precision quantization enable even higher efficiency and a finer-grained trade-off between inference energy and statistical accuracy. Multiple works have proposed extensions to the open RISC-V ISA targeted at QNN inference. Quark [99] and BISDU [170] propose extensions accelerating the bit-serial computation of sub-byte arithmetic operations, targeting minimum silicon area overhead. As we aim for maximum throughput at the best possible efficiency, we base the present work on XpulpNN [97]. XpulpNN extends the RISC-V ISA by adding support for 4-bit and 2-bit data types through packed-SIMD instructions. Furthermore, it mitigates the von Neumann bottleneck by fusing computation and data access into MAC-and-load (M&L) operations. Input activations and weights are read from an additional 2-port register file (RF), the NN-RF. Combining a SIMD MAC operation with the loading of the next activation or weight word and the update of the corresponding pointer enables optimized kernels to eliminate most explicit loads and pointer arithmetic, with the utilization of arithmetic units reaching up to 94 %. The XpulpNN ISA extension further encompasses the XpulpV2 ISA, which implements instructions to decrease memory management and control flow overhead, such as post-increment load and stores and hardware loops, further increasing QNN inference efficiency.

5.3 The RISC-V XpulpTNN ISA extension

In the following, we describe the XpulpTNN ISA extension, its integration into an 8-core cluster of high-performance RISC-V cores, and the software infrastructure to enable its use in end-to-end edge AI applications.

5.3.1 Instructions

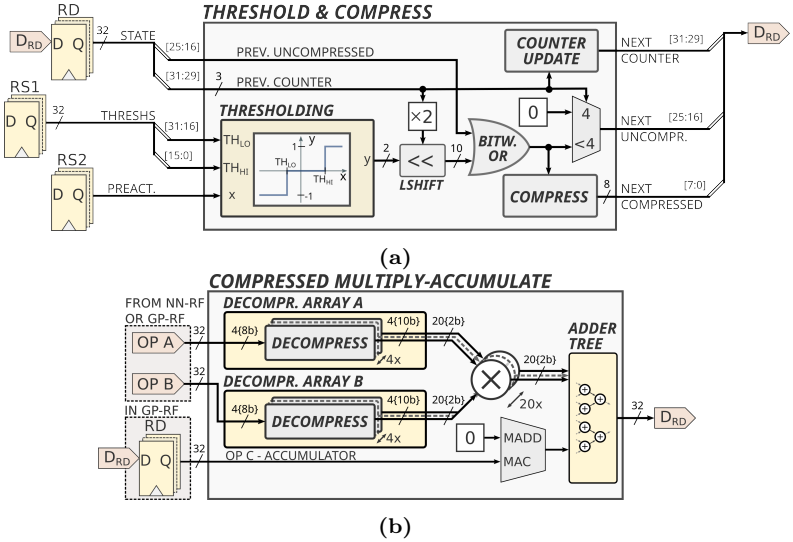


Figure 5.2 – Schematic of the hardware for the threshold-compress (`thrc`, shown in a) and compressed MAC (`smlsdotsp.t, sdotsp.t`, shown in b) instructions.

XpulpTNN extends the 32-bit RISC-V ISA. It is a very compact extension, consisting of three types of instructions which cover all operations required to execute the different layers of a TNN. All XpulpTNN instructions consume or produce ternary data elements, using the compression scheme proposed in [165] to represent 5

31 25	24 20	19 15	14 12	11 7	6 0	
1111100	IMM	rs1	100	rd	1110111	smlsdotsp.t
1011101	rs2	rs1	100	rd	1010111	sdotsp.t
1001101	rs1	rs1	100	rd	1010111	dotsp.t
0010001	rs2	rs1	100	rd	1010111	min.t
0011001	rs2	rs1	100	rd	1010111	max.t
0000100	rs2	rs1	110	rd	0110011	thrc

(a) Encoding of XpulpTNN instructions

31	29	28	26	25 16	15 8	7 0	
c		0		$x_{uncompr}$	0	x_{compr}	rd
t^{lo}					t^{hi}		rs1

(b) Encoding of `thrc` status (`rd`) and threshold (`rs1`) registers

Figure 5.3 – Encoding of instructions and input/output registers of XpulpTNN instructions

ternary elements (trits) with one byte or 20 trits with a 32-bit word. TNN layers can be mapped to kernels using three types of XpulpTNN instructions: multiply-add (MADD) instructions, element-wise comparison instructions and the threshold-and-compress instruction. All instructions are only implemented for signed ternary elements, as networks with unsigned activations can be converted to signed as shown in [9]. Figure 5.3a shows the encoding of the instructions introduced by XpulpTNN.

MADD Instructions The dot products in linear operators such as convolutions are implemented with MADD instructions, which perform 20-way packed-SIMD MAC operations on two input words, producing a 32-bit integer result. XpulpTNN implements three instructions of this class, all using the same hardware unit to perform the multiply-add operation, shown in Figure 5.2b. `dotsp.t` performs a pure MADD of `rs1` and `rs2` (operands A and B in Figure 5.2b), storing the result in `rd`. `sdotsp.t` is the analogous MAC operation, adding the MADD result to the contents of `rd`. `mlsdotsp.t` is a MAC-and-load instruction

extending XpulpNN. The MADD input operands are taken from the NN-RF, and `rd` is used as an accumulator. Its operation and encoding is implemented analogously to the 2-bit, 4-bit and 8-bit M&L instructions implemented by XpulpNN, which are described in detail in [97].

Element-wise Comparison Instructions Element-wise comparison instructions take two compressed 20-element input words, comparing each element between both and storing a compressed word of the smaller (`min.t`) or larger (`max.t`) element at each position into the destination register. The `max.t` instruction can be used to implement max-pooling layers efficiently.

Threshold-and-Compress Instruction Activation layers in TNNs are implemented with thresholding operations, which can be mapped to XpulpTNN's *threshold-and-compress* instruction (`thrc`). It takes three registers as inputs: `rs1` contains a 32-bit integer number, which is compared with two 16-bit integer thresholds stored in `rs2` to produce a ternary result. As the ternary compression encodes five trits into an 8-bit value and thresholding only produces one trit, the instruction is designed in a stateful manner. `rd` serves both as an input and output register, holding the instruction's state. The state consists of 3 items: 10 bits containing up to 5 packed, uncompressed 2-bit trits, 8 bits containing the compressed representation of those trits, and a 3-bit counter indicating how many trits have been processed already. When the instruction is executed at a counter value of c, the thresholding result is extended to a 10-bit value and left-shifted by $2c$ bits. The shifted value is then merged with the previous uncompressed values by a bitwise OR operation, and the result is fed into a ternary compression unit. The compressed byte is stored in the updated status register, along with the next counter value and the uncompressed vector, which is reset to all-zeros after five elements have been processed. Figure 5.2a shows the hardware implementation of the threshold-and-compress instruction.

5.3.2 The XpulpTNN System

To evaluate the performance and efficiency of XpulpTNN, we integrated the ISA extension into the open-source RI5CY-NN core, which

implements the base RV32IMC ISA and the XpulpV2 [96] and XpulpNN [97] extensions. This core represents the state of the art for efficient QNN inference [171], [172]. It possesses the hardware infrastructure to support M&L instructions, making it both a highly optimized comparison baseline and an ideal starting point for implementing the proposed extensions. We use the XpulpTNN-enabled RI5CY core to assemble a fully-featured, high-performance, low-power SoC on which we perform our evaluations. The system has two main processing domains: the *SoC domain* and the *cluster domain*.

A RI5CY core manages system operation in the SoC domain, called the *fabric controller (FC)*. The system's main program and data memory, termed L2 memory, is also located in the SoC domain and consists of 1 MiB of SRAM, divided into eight banks. The system's main interconnect links the FC, on-chip peripherals, L2 memory, and the cluster domain.

Compute-intensive parallelizable tasks are offloaded to the cluster domain. It contains a PULP cluster of 8 RI5CY cores with 128 KiB of L1 scratchpad TCDM in 16 banks, connected with a single-cycle logarithmic interconnect to minimize data access latency. Cluster cores execute program code stored in L2 memory, which is located in a different clock domain and accessible via a 64-bit AXI4 port through a clock domain crossing (CDC). A shared instruction cache of 4 KiB minimizes stalls caused by instruction fetching through this CDC. The cluster is programmed in the single program, multiple data (SPMD) model, i.e., all cores execute the same program, using the core index to control program flow.

5.3.3 XpulpTNN Software Support

To make the deployment of TNNs to XpulpTNN-enabled RISC-V systems accessible to application developers, we have implemented an end-to-end deployment pipeline. It consists of compiler support for the new instructions, a library of performance-optimized, parallelized kernels leveraging XpulpTNN, and a mapping tool that takes an ONNX representation of a TNN and generates a C application which executes the network.

GCC Compiler Support All XpulpTNN instructions can be inferred from pure C code by calling built-in functions. This removes the need for inline assembly code and enables GCC's full range of optimizations during the compilation process. The modified version of GCC supporting XpulpTNN is available open source[1].

Kernel Support We have implemented a set of optimized ternary layer kernels leveraging XpulpTNN. They implement four layers, all operating on compressed ternary inputs: 2-dimensional convolutions, 1-dimensional dilated convolutions, max-pooling, and fully-connected layers. The fully connected kernel is intended for classifier layers that compute class scores and produces integer outputs, while the other kernels use the `thrc` instruction to produce compressed ternary outputs. The number of ternary input and output channels is restricted to multiples of 5, as the ternary compression encodes blocks of 5 elements in one byte.

As convolutional layers constitute most of the workload in DNNs, a well-optimized convolution kernel is crucial to end-to-end efficiency. Our ternary convolutional kernels take inspiration from the open-source PULP-NN library [164] and decompose convolutions into a data reordering step (*im2col*) and a matrix multiplication step (*matmul*). The matrix multiplication kernels use the `smlsdotsp` M&L instructions to perform the dot product operations, and the integer results must then be mapped back to quantized values in an additional step merging activation, batch normalization, and requantization [91]. The instructions introduced by XpulpTNN increase efficiency by optimizing both steps. The 25 % increase in data density afforded by the ternary compression directly translates to a corresponding increase of throughput in the hot loop that calculates the integer dot products. In integer-bitwidth QNNs, the activation-requantization step consists of an affine transformation followed by an arithmetic shift and data packing, taking up a significant share of the total kernel execution time. XpulpTNN's `thrc` instruction performs the complete process in a single instruction, minimizing this overhead. Combined, these improvements lead to an increase in throughput over PULP-NN's 2-bit kernels by much more than the 25 % that the increased data density

[1] https://github.com/da-gazzi/pulp-tnn-gnu-toolchain

provides, as detailed in Section 5.4.3.

5.3.4 Deployment Pipeline

For the seamless deployment of full networks to XpulpTNN-enabled systems, we have integrated support for TNNs and our XpulpTNN kernels into the open-source DORY [105] tool. DORY takes precision-annotated ONNX graphs as input and produces compilable C applications executes the network on the target platform. Multi-level memory hierarchies are supported by tiling layers that do not fully fit into the scratchpad memory. DORY uses integer linear programming (ILP) to optimize the tiling policy and inserts the appropriate DMA driver calls for data transfer between the different levels of the memory hierarchy.

5.4 Results and Discussion

In this section, we evaluate the efficiency and performance of XpulpTNN. We first present hardware results collected on a full backend layout of the XpulpTNN-enabled PULP cluster. This implementation is used to evaluate the silicon area overhead and compare power consumption in post-layout simulations to the baseline XpulpNN cluster. We then evaluate the performance of the ternary kernels on a comprehensive benchmark suite and compare the efficiency to that of 2-bit kernels. Finally, we compare TNNs mapped to the XpulpTNN system with 2-bit networks executed on the baseline system on two end-to-end benchmark applications to evaluate XpulpTNN's impact on the trade-off between accuracy and inference latency and energy.

5.4.1 Experimental Setup

For our hardware evaluations, we synthesized the 8-core PULP cluster described in Section 5.3.2 in the GlobalFoundries 22 nm FDX process, using libraries for the typical corner with $V_{DD} = 0.8\,\text{V}$. We used Synopsys Design Compiler 2019.3 and constrained the core clock to a period $t_{clk} = 1.5\,\text{ns}$. We used the synthesized netlist to perform

		XPulpNN	**XpulpTNN** (this work)
Core	A_{core} [a]	163.5 kGE	168.4 kGE (+3.0 %)
	$P_{MM,8b}$ [b]	4.1 mW	4.2 mW(+2.4 %
	$P_{Conv,LP}$ [c]	4.0 mW	4.3 mW (+7.8 %)
Cluster	A_{clus} [a]	3.29 MGE	3.32 MGE (+0.9 %)
	Density	70.2 %	71.0 % (+0.8 pp.)
	f_{clk} [b]	500 MHz (376 MHz)	500 MHz (389 MHz)
	$P_{MM,8b}$ [c]	58.1 mW	58.9 mW(+1.4 %)
	$P_{Conv,LP}$ [c]	57.7 mW	60.7 mW (+5.2 %)
	$\text{Eff}_{Conv,LP}$ [c]	383.9 GOp/J	603.3 GOp/J **(+ 57.1 %)**

[a] Obtained from synthesized netlists. One gate equivalent (GE) in 22 nm FDX is 0.199 µm^2, the size of a NAND2 gate.
[b] The first number is obtained at HV operating conditions ($V_{DD} = 0.72\,\text{V}, T = 25\,^\circ\text{C}$), the number in parentheses is obtained at LV operating conditions ($V_{DD} = 0.65\,\text{V}, T = 25\,^\circ\text{C}$).
[c] Evaluated at LV operating conditions.

TABLE 5.1
HARDWARE FIGURES OF MERIT FOR THE XPULPTNN SYSTEM AND THE BASELINE IMPLEMENTING ONLY XPULPNN.

a full backend layout using Cadence Innovus 21.13. We optimize for two supply voltages, $V_{DD} = 0.72\,\text{V}$ (HV, max. throughput) and $V_{DD} = 0.65\,\text{V}$ (LV, max. efficiency). We target a system clock frequency of $f_{clk} = 500\,\text{MHz}$; the achieved frequencies are listed in Table 5.1. The synthesis and backend layout flow is identical for both cluster versions. Power results were generated by simulating the full system using the post-layout netlists of the two cluster implementations to collect value change dump (VCD) files, which were used to estimate the power consumption in Innovus. We ran performance evaluations on an FPGA port of the complete system, generated from the same SystemVerilog RTL code as the physical implementation.

5.4.2 Hardware Impact

To evaluate the implementation overhead of our extension, we compare the silicon area and timing of the 8-core cluster implementing XpulpTNN to an identically parametrized baseline cluster imple-

menting only the XpulpV2 and XpulpNN extensions. This baseline represents the state of the art in AI-targeted, RISC-V-based MCU-class systems. The overhead we report thus reflects the cost of adding TNN optimizations to a system already intended for inference of QNNs. Table 5.1 reports the standard cell areas $A_{\{core,clus\}}$ after synthesis, the achieved operating frequency, and the power consumption. $P_{conv,LP}$ denotes power consumption of 2-bit convolution on the baseline system and ternary convolution on the XpulpTNN system. $P_{MM,8b}$ is the power consumption during 8-bit matrix multiplication, an indicator of the impact of our modifications on cluster power consumption for programs that do not use the new instructions. We report these figures both for a single core and the complete cluster. The area overhead of XpulpTNN on a single core is already low at 3 %. At 0.9 %, the cluster-level overhead is even lower, as our modifications do not affect the other components of the cluster, such as L1 memory and instruction cache. Timing is not impacted since the critical path in both cluster versions is in the FPU. The negligible area increase and non-existent timing impact mean that the addition of XpulpTNN essentially incurs zero implementation overhead when placing and routing the cluster. This assessment is confirmed by the negligible post-route standard cell density increase of 0.8 percentage points.

5.4.3 Kernel Performance and Efficiency

Figure 5.4 shows the throughput of ternary (using our kernels, optimized for XpulpTNN) and 2-bit (using PULP-NN kernels optimized for XpulpNN) convolutions running on eight cores. N_b is the number of bytes required to store all $N_{\{i,o\}}$ input/output channels of a single pixel, with $N_i = N_o = 5N_b$ for ternary kernels and $N_i = N_o = 4N_b$ for 2-bit kernels. The kernel size is $k \times k = 3 \times 3$, and the input and output feature map sizes are chosen such that inputs, outputs, and weights fit into L1 memory. On average, ternary kernels achieve 67 % higher throughput than 2-bit kernels at equal N_b and resolution. When restricting the comparison to layers where the channels of each pixel fill an integer number of 32-bit words, i.e., $N_b = 4k$, $k \in \mathbb{Z}$, the ternary kernels exhibit 51 % higher throughput than their 2-bit equivalents. Those layers also stand out for their considerably higher throughput, as all dot product calculations can be performed in the

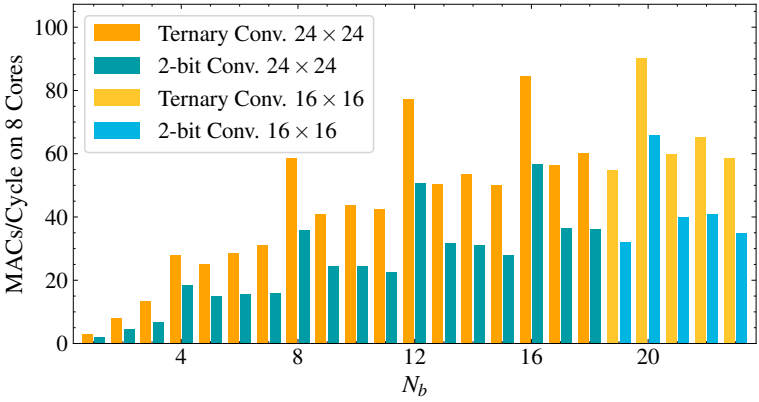


Figure 5.4 – Throughput comparison between ternary convolution kernels and 2-bit kernels from the PULP-NN on the 8-core PULP cluster.

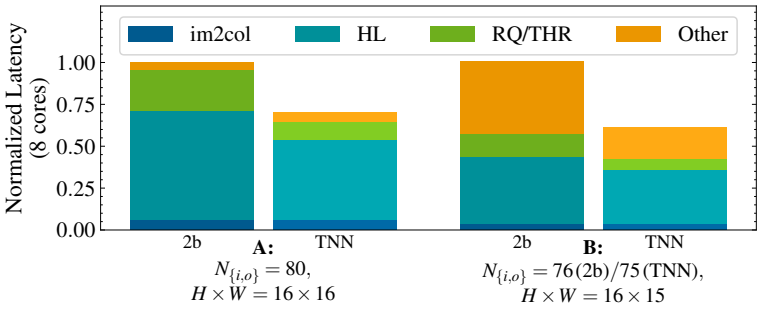


Figure 5.5 – Latency breakdown comparison between 2-bit and ternary 3×3 convolution kernels. Latency is normalized to the 2-bit kernel's latency and is decomposed into *im2col*, *hot loop* (HL), *requantization/thresholding* (RQ/THR) and *Other* components and shown for two test cases.

optimized matrix multiplication hot loop. Figure 5.5 illustrates the nature of the speedup afforded by XpulpTNN. In test case A, all pixels are word-aligned, and all calculations are performed in the optimized hot loop, which accounts for most of the latency in the 2-bit kernel together with the requantization step. The ternary kernels reduce hot loop and requantization latency by 27 % and 55 %, respectively,

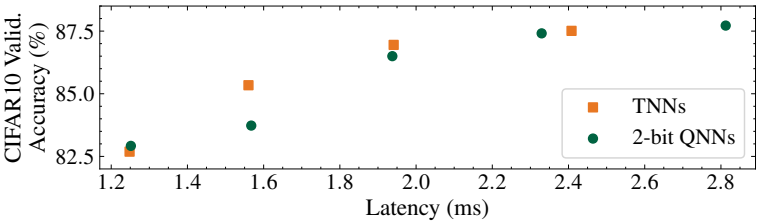


Figure 5.6 – Latency comparison of VGG-like networks quantized to 2-bit and ternary precision.

resulting in an overall latency reduction by 30 %. The latency from the other contributors is roughly equal to the 2-bit kernel. In test case B, pixels are not word-aligned. The ternary kernels' optimized handling of leftover calculations yields a latency reduction of 38 % after scaling the latency to the slightly reduced number of MACs performed. Thanks to this increased throughput and the low power consumption overhead, ternary convolutions on an equal data volume on the XpulpTNN system are 57 % more efficient than 2-bit convolutions on the baseline system, placing TNNs at a very attractive trade-off point of accuracy and energy consumption.

5.4.4 End-to-End Network Inference

To evaluate how XpulpTNN can be used in end-to-end edge applications, we consider two benchmark tasks. The first is 11-class gesture recognition on DVS data from the DVS128 dataset [70], and the second is image classification on the popular CIFAR-10 dataset [6]. On both tasks, we compare optimized 2-bit QNNs and TNNs mapped to the XpulpTNN system using our deployment pipeline.

CIFAR-10 Image Classification We evaluate the trade-off between latency and classification accuracy for 2-bit and ternary networks on the example of a small VGG-like network architecture. The architecture is detailed in Table 5.2. The first and last layers are trained and executed in 8-bit precision, all other layers are in 2-bit or ternary precision. We scale the networks by adjusting the number of

Layer	Outp. Res.	C_{out}	k_{conv}	b_w [a]	b_o [a]
Input	32×32	3	3×3	–	8
C2D(S)-MP	16×16	32	3×3	8	1.6/2
C2D(S)	16×16	N_c	3×3	1.6/2	1.6/2
C2D(S)-MP	8×8	N_c	3×3	1.6/2	1.6/2
C2D(S)	8×8	N_c	3×3	1.6/2	1.6/2
C2D(S)-MP	4×4	N_c	3×3	1.6/2	8
FC	1	10	–	8	32

[a] Weight/output precision in bits; 1.6 denotes ternary precision

TABLE 5.2
ARCHITECTURE OF VGG-LIKE NETWORKS USED IN END-TO-END LATENCY EVALUATION ON CIFAR-10 DATASET. **(C{1/2}D({S/V/C}):** 1/2-DIMENSIONAL CONVOLUTION WITH **S**AME/**V**ALID/**C**AUSAL PADDING. **MP**: MAX-POOLING LAYER. **FC**: FULLY-CONNECTED LAYER. MP LAYERS USE A 2×2 KERNEL, A STRIDE OF 2 AND NO PADDING. THE CHANNEL COUNT OF CONVOLUTIONAL LAYERS N_c IS PARAMETRIZABLE; WE EVALUATE $N_c \in \{32, 48, 64, 80, 96\}$ (2-BIT QNNS) AND $N_c \in \{40, 60, 80, 100\}$ (TNNS).

input and output channels of all layers after the first, which is fixed to 32 output channels. As our ternary kernels do not support this number of input channels, the output of the first layer is zero-padded to 40 channels for the TNNs. We use our deployment flow to map the networks to the XpulpTNN system, using a cluster clock frequency of 300 MHz. Figure 5.6 shows the accuracy-latency trade-off for the 4 different TNN and 5 different 2-bit QNN parametrizations. All evaluated TNNs extend the accuracy-latency Pareto front, with the 60-channel TNN achieving a 1.6 pp. higher validation accuracy than the 48-channel 2-bit network at a marginally lower latency.

DVS Gesture Recognition We adopt the hybrid network architecture proposed in [9], consisting of a 2-dimensional CNN followed by a 1-dimensional TCN. The 2D CNN takes an input image of 64×64 pixels with 4 channels, where each channel represents a DVS event frame. DVS event frames are natively ternary, as the sensor can report a positive/negative event (± 1) or no event at all (0) during a time window; this makes DVS data particularly suited for processing with TNNs. The architecture of the network is detailed in Table 5.3.

	Layer	Outp. Res.	C_{out}	k_{conv}	D	S
2D CNN	Input	64×64	4	–	–	–
	C2D(S)-MP	32×32	$N_{c,1}$	3×3	1	1
	C2D(S)-MP	16×16	80	3×3	1	1
	C2D(S)-MP	8×8	80	3×3	1	1
	C2D(S)-MP	4×4	80	3×3	1	1
	C2D(V)-MP	1×1	80	3×3	1	1
1D TCN	C1D(C)	1×5	80	2	1	1
	C1D(C)	1×5	80	2	2	2
	C1D(C)	1×5	80	2	4	4
	FC	1×1	11	–	–	–

Table 5.3
Architecture of the DVS gesture classification networks used for evaluation. S: Stride, D: Dilation. Other notation and parametrization as in Table 5.2.

For a fair comparison between 2-bit QNNs and TNNs, the layers of 2-bit networks must have multiples of 16 input channels, while TNN layers should have multiples of 20 input channels, so that feature map pixels are word-aligned and kernel performance is optimal (see Figure 5.4). We achieve this by slightly modifying the architecture from [9]. We change all layers but the first to have 80 input and output channels, the smallest common multiple of 16 and 20. As the first layer's large input resolution incurs a high computation cost that scales with the number of its output channels $N_{c,1}$, we set $N_{c,1} = 20$ for the TNN, and evaluate 2 settings ($N_{c,1} = 16$ and $N_{c,1} = 32$, as originally described in [9]) for the 2-bit QNN. Inference latency, energy and validation accuracy results are shown in Table 5.4. Compared to the large baseline 2-bit QNN, the TNN exhibits 37.4 % lower inference latency and an estimated 33 % lower inference latency at a negligible accuracy drop of 0.3 pp.. Compared to the reduced-size 2-bit network, which has approximately 10 % fewer operations, the TNN's latency and inference energy are still lower by 13.5 % and 9 %, respectively, while the validation accuracy is improved by 0.2 pp.

	2-bit QNN		TNN
$N_{c,1}$	32	16	20
DVS128 Acc.	96.5 %	96.0 %	96.2 % (−0.3 pp.)
MACs	47.9M	33.7M	37.2M
t_{inf}	5.7 ms	4.1 ms (−27.7 %)	3.6 ms **(− 37.4 %)**
Θ (MACs/Cyc.)	28.0	27.2 (−2.7 %)	34.8 **(+ 24.3 %)**
$\mathbf{E_{inf,cl}}$[a]	329 µJ	238 µJ (−28 %)	217 µJ **(− 33 %)**

[a] Estimated from post-layout power simulation results

TABLE 5.4
COMPARISON OF END-TO-END INFERENCE PERFORMANCE BETWEEN 2-BIT AND TERNARY NETWORKS ON DVS GESTURE RECOGNITION NETWORKS.

5.5 Conclusion

In this chapter, we have addressed the gap between existing TNN systems, which primarily rely on dedicated accelerators, and popular edge computing systems, which are based on RISC cores with area-efficient ISA extensions. Specifically, we describe the implementation of XpulpTNN, an extension to the RISC-V ISA designed to enable the efficient processing of TNNs, in an open-source RISC-V core targeted at edge AI applications and assemble an 8-core compute cluster from the extended core.

A complete implementation in an IoT-friendly GF 22 nm FD-SOI technology shows that XpulpTNN incurs negligible area overhead of only 3 % and 0.9 % at the core and cluster levels, respectively, with no timing degradation. In post-layout simulations, the cluster's power consumption while running a ternary convolution kernel is only 5.2 % higher than the baseline system running an equivalent 2-bit convolution, while our optimized kernels achieve 67 % higher throughput. This results in 57 % higher energy efficiency. In end-to-end evaluations, we show that TNNs deployed to the XpulpTNN-enabled system offer a competitive trade-off between inference latency/energy and accuracy, achieving up to 1.6 pp. higher CIFAR-10 accuracy than a 2-bit QNN at equal latency. For a gesture recognition application, XpulpTNN enables the deployment of a TNNs that decreases inference latency

and energy by 37 % and 33 % from a 2-bit baseline at a negligible accuracy loss of 0.3 pp.. Overall, our results show that XpulpTNN enables efficient TNN inference on RISC-V cores at negligible overhead to system implementation, posing no barrier to the adoption of our extension in edge computing systems.

Chapter 6

Mixed-Precision Networks for End-to-End Efficiency in Edge Applications

6.1 Motivation

As the previous two chapters have demonstrated, aggressively quantized DNNs such as TNNs can achieve extremely low inference energies at good statistical accuracy levels. However, extreme quantization of small, efficient network topologies applied to challenging tasks results in severe accuracy drops. For example, a 2-bit quantized version of MobileNetV1 (MNv1) for ImageNet failed to converge in our experiments, shown in Figure 1.1. To optimize the latency-energy trade-off of these networks with quantization, a finer-grained approach is thus needed. *Mixed-precision quantization* proposes to quantize different parts (usually at the granularity of individual layers) of the network to different precisions. Ideally, this allows a designer to aggressively quantize those layers where the resulting latency and energy reductions are greatest and the accuracy penalty

smallest. Conversely, layers where low-bitwidth quantization does not substantially decrease inference energy but severely degrades the statistical accuracy would be left in a high precision. The challenge of this approach consists of optimizing the layer-wise quantization policy: the search space of mixed-precision configurations for a given network is exponential in the number of layers and thus intractable. This makes brute-force evaluations impossible, and various works have proposed directed search algorithms for precision configurations. Multiple works have applied differentiable neural architecture search (DNAS) to mixed-precision search. However, these approaches generally rely on a proxy for latency, such as binary operation (BOP) count, to guide the search [173], [174]. On real hardware platforms, latency is highly dependent on factors such as hardware implementation, memory hierarchy, tiling, and kernel implementation, none of which are directly linked to the number of BOPs in a network. In this work, we propose a mixed-precision latency optimization method consisting of a hardware-agnostic differentiable search step based on the Bayesian Bits algorithm [173], followed by a hardware-aware, profiling-based heuristic which both reduces execution latency and improves accuracy by increasing the precision in layers where higher precisions achieve lower latency. To reach a desired target latency, the configurations can be further refined in an optional greedy search step. In evaluations on MNv1 and MobileNetV2 (MNv2), deployed to a cycle-accurate RISC-V multi-core simulator, our approach results in an accuracy-latency trade-off curve that dominates those produced by either differentiable search or greedy heuristics on their own. To our best knowledge, we demonstrate for the first time end-to-end deployment of mixed-precision networks to an MCU-class platform that exhibit not only a reduced memory footprint but also reduced execution latency by up to 29.2% at full-precision equivalent classification accuracy. Our key contributions are the following:

- We present a lightweight method to find latency-optimized mixed-precision quantization configurations for DNNs, consisting of a hardware-agnostic differentiable model search and hardware-aware heuristics, allowing the quick generation of optimized configurations for different platforms,
- we compose an end-to-end flow consisting of precision search,

training, generation of integerized models and deploy the found configurations on a cycle-accurate simulator for high-performance RISC-V MCU systems and

- we analyze the resulting accuracy-latency trade-offs, showing that our approach achieves an end-to-end latency reduction by up to 29.2% vs. 8-bit quantization at full-precision equivalent classification accuracy and finds Pareto-dominant configurations with respect to homogeneous 4-bit quantization.

6.2 Related Work

Approaches to mixed-precision configuration search in literature can be broadly clustered into three categories: *Differentiable search* algorithms which jointly train precision-selection parameters and model parameters, *static precision assignment*, where the precision of each layer is determined off-line based either on heuristic criteria or model statistics and *reinforcement learning (RL)*-based approaches which train a RL agent to assign precisions to each layer.

[173] adopt a differentiable-search approach, which we adapt for the first step of our algorithm. In their Bayesian Bits algorithm, quantization to a given precision is decomposed by first quantizing a tensor to the lowest supported bit-width, and higher precisions are recovered by adding the difference to the next quantization level to the quantized tensor. During training, the addition of the error tensors is controlled by stochastic, continuous-value gates whose distribution is parametrized by trainable parameters. To encourage low-precision quantization, a regularization term is added to the loss, with each precision gate contributing a term proportional to the BOP cost its associated layer incurs in the precision level controlled by the gate. EdMIPS [174] uses a less complex approach, training gate parameters for each available precision which are used to assemble the final tensor by weighting its differently quantized versions with the softmax distribution parametrized by the gates. The regularizer term used is analogous to that of Bayesian Bits, with a term proportional to the expected value of BOPs being added to the classification loss. [175] take a similar approach, learning a single continuous parameter

used to interpolate between bit-widths per layer, and [176] train only weight precisions, using the Gumbel Softmax technique [177] to sample the precision gates' values. All of these algorithms have in common that the objective functions use BOP count (or another quantity that is monotonically related to precision, such as model size) as the target metric. As we show in Section 6.3, BOP count does not necessarily correlate with latency, and there is no trivial adaptation to enable these algorithms to optimize for execution latency directly. Furthermore, differentiable search does not allow for enforcing hard constraints on the configurations during the search.

In contrast, techniques performing static assignment of layer-wise precisions may target any performance metric, including execution latency, but can not directly optimize statistical accuracy at the same time. [91] greedily reduce weight precision to allow a network to fit in on-device storage and decrease activation precision of those layers which would not otherwise fit into device memory. As their target platform is an off-the-shelf MCU without native sub-byte arithmetic support, the low-precision operations incur runtime overhead, and the reduced memory footprint comes at the cost of increased runtime latency. [178] calculate a second-order sensitivity metric for each layer and use ILP to optimize layer-wise precisions for execution latency, model size or BOP count while minimizing the estimated cumulative disturbance due to quantization.

In the category of RL-based algorithms, [179] use deep deterministic policy gradient (DDPG) and restrict the agent's action space to networks that fulfill a user-determined (latency, energy, or model size) constraint, such that the reward function only considers the evaluated network's accuracy drop relative to full-precision accuracy. [180] use a combined reward function incorporating both the state of quantization and the classification accuracy with the proximal policy optimization (PPO) actor-critic algorithm to find weight-only quantization policies.

6.3 Free Bits

Free Bits is a multi-step method to find mixed-precision configurations of DNNs optimized for low latency on a given target hardware platform. In the first step, we employ two variants of the Bayesian Bits algorithm

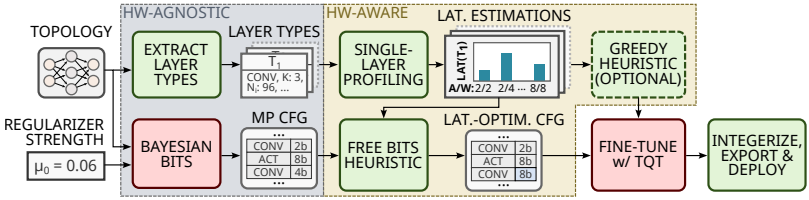


Figure 6.1 – Overview of the proposed algorithm. Mixed-precision configurations (**MP CFGs**) for a given topology are first generated using Bayesian Bits. These configurations are then optimized for the target platform using profiling results for the layer types occurring in the topology by increasing the precisions in layers where this results in lower latency (**LAT.**). Optionally, the resulting configurations can be further refined to meet a latency target using a greedy heuristic. The final configuration is fine-tuned with QAT using the TQT algorithm, after which it can be deployed to the target hardware. Green boxes represent steps with low computational effort while steps with high computational effort (i.e., involving neural network training) are represented by red boxes.

with n_{reg} different regularizer strengths to find reduced-precision configurations of the targeted network architecture. The Bayesian Bits regularizer penalizes high BOP counts and has no concept of latency (and indeed does not support targeting latency optimization directly, see Section 6.3.1). Thus, the resulting configurations will parametrize mixed-precision networks which are not optimized for any particular hardware platform and which may even exhibit higher inference latency than, e.g., an all-8-bit baseline on the target platform.

In the second step, we optimize the mixed-precision configurations found with Bayesian Bits for a specific target platform. To achieve this, we first profile the latency of each unique layer type in the network in all allowed precision configurations on the target platform. This profiling data is used to update the initial configuration by choosing, for every layer in the network, the lowest-latency precision setting that is higher or equal to that of the initial configuration. As the precision increase is expected to improve both latency and statistical accuracy, we name this step the *free bits* heuristic.

The latency of a mixed-precision network resulting from the first two steps is not bounded by a hard constraint, and if no satisfactory configuration is found, we apply a greedy heuristic in the third step

to reach the latency target. When starting from a configuration with latency higher than the target, the heuristic successively decreases the precision of layers where this results in the largest latency reduction until the latency target is met. When starting from a configuration with a latency lower than the target, precision is increased in those layers where the latency penalty is the lowest.

The final configuration is then fine-tuned using a modified version of the TQT algorithm [59] and automatically converted to an integer-only model, which can be fed to a deployment backend for the target platform.

Our method emphasizes versatility and efficiency: Decoupling the differentiable search for low-precision configurations from the platform-specific latency optimization means that the computationally intensive first step only needs to be performed once for every network. From the generic mixed-precision configurations it produces, latency-optimized configurations can be generated for different hardware targets efficiently, as the second and third steps only require profiling data from the target platform and do not directly take into account task accuracy. The total number of Bayesian Bits training runs performed is $2n_{reg}$: For each regularizer strength, both variants of Bayesian Bits are applied. Once the baseline configurations have been found with Bayesian Bits, optimized configurations for different hardware targets and latency constraints can be generated with a single QAT fine-tuning training run.

6.3.1 Differentiable Mixed-Precision Search

To find the initial mixed-precision configurations which are latency-optimized in the second step, we apply two variants of the Bayesian Bits algorithm. Bayesian Bits aims to reduce a network's total BOP count with a regularizer that penalizes each precision gate's contribution to the expected BOP count individually (see also Section 6.2). There are two reasons why the algorithm cannot target latency directly. First, Bayesian Bits treats activation and linear operator layers individually, summing up the regularizer terms for every layer and optimizing each layer's gates independently. However, the latency of a single layer is determined jointly by the precision of input activations and weights and cannot be decomposed into additive contributions from

each precision. Second, the regularizer terms for an individual layer's gates are added together, yielding a monotonously increasing penalty as a layer's expected precision increases. As shown in Figure 6.4, this is not necessarily the case for the execution latency of a layer: Instead, a layer may have lower latency when executed in a higher precision than in a lower one, depending on the hardware platform. In addition to the original Bayesian Bits algorithm, we also employ a modified version where the gate parameters for each precision are shared between a linear operator layer and its input activation, enforcing equal input and weight precisions. This modification accounts for the fact that on our target platforms, the theoretical throughput for a layer with non-equal activation and weight precisions is bounded by the higher of the two precisions.

6.3.2 Free Bits Heuristic

We update the configurations found by the latency-agnostic Bayesian Bits with a target-specific heuristic using profiling data from the hardware target platform. This step relies on two core ideas: First, we assume our target platform executes networks layer-by-layer, which implies $L_{net} \approx \sum_{i=1}^{N} L_i$ for the total execution latency L_{net} of an N-layer network where the i-th layer is executed with latency L_i. This is the case for most systems on which DNNs inference is run. Second, increasing a layer's input activation or weight precision never decreases the network's statistical accuracy. While the regularizing effect of quantization to 8-bit (or higher) precision can have beneficial effects on task accuracy (as is the case with MNv1, see Section 6.4), the literature shows that lower-precision quantization generally leads to an accuracy drop for difficult tasks such as ImageNet [59], [178], [181], [182], justifying this second assumption.

Following the first assumption, we characterize each unique linear operator in the target network as a *layer type*. A layer type is defined as the tuple of all parameters which determine how a computational kernel is invoked, e.g., input dimensions, number of input/output channels, number of channel groups, and kernel size. For each layer type occurring in the network, we measure the execution latency on the target platform for all supported combinations of input and weight bit-widths (b_{in}, b_{wt}). With this estimation of $\{L_i\}$, $i \in [1, N]$, we

Algorithm 2 Profiling-based Heuristic Precision Search

Input:

LD: Latency dictionary mapping layer type c and precisions (b_{in}, b_{wt}) to a measured latency
C_{net}: Dictionary of layer types and precisions representing a mixed-precision network, of the form $\left\{i : \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right\}_{i=1}^{N}$
P_{all}: Set of allowed combinations (b_{in}, b_{wt}) of input and weight precisions

Output:

C'_{net}: Latency-optimized mixed-precision configuration of the input network

function HIGHER($(b_{in,1}, b_{wt,1}), (b_{in,2}, b_{wt,2})$) ▷ Returns True if precision 1 $\geq$ precision 2
  return $(b_{in,1} \geq b_{in,2}) \wedge (b_{wt,1} \geq b_{wt,2})$
end function
$C' \leftarrow C$
for all $i, \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right) \in C_{net}$ **do**
  $lat^i_0 \leftarrow LD\left[\left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right]$ ▷ Initial latency
  $cands \leftarrow \{(b_{in}, b_{wt}) \,|\, (b_{in}, b_{wt}) \in P_{all}, LD\left[\left(t^i, (b_{in}, b_{wt})\right)\right] \leq lat^i_0, \text{HIGHER}((b_{in}, b_{wt}), (b^i_{in}, b^i_{wt}))\}$ ▷ Select lower-or-equal-latency configurations with higher-or-equal precisions
  $best \leftarrow \underset{(b_{in}, b_{wt}) \in cands}{\arg\min} LD[(t^i, (b_{in}, b_{wt}))]$ ▷ Select lowest-latency candidate
  $C'_{net}[i] \leftarrow (t^i, best)$ ▷ Update net configuration
end for

iterate over the layers in the network found by Bayesian Bits and check for every layer if higher-precision configurations of the same layer type with a lower estimated latency exist. If so, we update the layer's precision configuration to that with the lowest latency estimation. By the two assumptions above, the resulting network's execution latency and statistical accuracy will be upper-bounded and lower-bounded, respectively, by those of the configuration found by Bayesian Bits. As it is expected to produce strictly superior configurations in terms of latency and statistical accuracy, we call this procedure the *free bits* heuristic. A pseudocode description of the procedure is shown in Algorithm 2.

6.3.3 Greedy Mixed-Precision Search

The configurations produced by the free bits heuristic can optionally be further refined with a greedy heuristic. Given a latency target L_{tgt} and a network configuration C_0 with estimated execution latency L_0, we aim to modify C_0 to reach a latency lower than, but as close as possible to, L_{tgt}. If $L_0 < L_{tgt}$, we apply the heuristic in the upward direction, seeking to increase the precision of as many layers as possible to maximize the accuracy improvement. Accordingly, we increase the precision of layers where this carries the lowest latency penalty. Conversely, in the downward direction (i.e., $L_0 > L_{tgt}$), we want to decrease the precision of as few layers as possible to minimize accuracy drop and thus choose the layers where a precision reduction results in the largest latency reduction. In the case $L_0 > L_{tgt}$, we additionally try to keep the layer-wise precision reductions applied "small", e.g, we prefer modifying two layers' configurations from 8b/8b to 8b/4b to modifying a single layer from 8b/8b to 8b/2b to achieve the same latency improvement. This is based on the experience that very low bit-widths have a disproportionate impact on classification accuracy. A pseudocode description of the two versions of the greedy heuristic is given in Appendix B.2.

This greedy heuristic can be applied to arbitrary C_0, and, applied to homogeneous-precision baseline configurations, serves as a test to assess the utility of the initial differentiable-search step: If applying this low-cost heuristic directly yields an accuracy-latency curve that is not Pareto-dominated by that resulting from the full algorithm,

the computationally expensive differentiable search serves no useful purpose. The results of this comparison are detailed in Section 6.4.

6.3.4 Quantization-Aware Fine-Tuning and Deployment

Having arrived at a latency-optimized mixed-precision configuration, we perform QAT to fine-tune the network's parameters using a generalized version of the TQT algorithm. Our implementation of TQT differs from the original algorithm only in that we do not force clipping bounds to be exact powers of two. For the conversion of full-precision networks to FQ models, QAT and generation of deployable integer-only models, we use the QuantLab[1] [8] framework, which allows automating large parts of this flow. For integerization, we follow the procedure referred to as ICN by [91] and *dyadic quantization* by [178]. The inputs to element-wise addition nodes which occur in networks with residual connections are quantized to 8 bits, and an equal quantization step size is enforced during the QAT phase. Likewise, the outputs of adder nodes are always quantized to 8-bit precision.

6.4 Results

6.4.1 Experimental Setup

We performed experiments on two network architectures, applying the procedure proposed in Section 6.3 to MNv1 [36] and MNv2 [37]. We used width multipliers of 0.75 for MNv1 and 1.0 for MNv2. The input resolution was 224×224 for both networks. We train our networks on the ILSVRC2012 [7] 1000-class dataset and report top-1 classification accuracies on the validation set.

Differentiable Mixed-Precision Search and QAT Fine-Tuning We applied the two variants of Bayesian Bits described in Section 6.3.1 to the MNv1 and MNv2 network topologies. The hyperparameters for Bayesian Bits training are listed in Table B.2. The configurations

[1] `https://github.com/pulp-platform/quantlab`

produced by our algorithm (as well as those produced by Bayesian Bits in the case of MNv1) were fine-tuned with TQT using the hyperparameters listed in Table B.1. In accordance with the capabilities of our hardware target (see below), the precisions Bayesian Bits can select from are 2, 4, and 8 bits for both weights and activations. We did not use the pruning mechanism of Bayesian Bits, i.e., 2-bit gates are always fully turned on.

Profiling, Deployment and Hardware Targets We use QuantLab's automated integerization flow to generate precision-annotated, integer-only ONNX models, which are consumed by the DORY [105] deployment backend. DORY generates compilable C code leveraging the PULP-NN [164] kernel library, which we run on GVSOC, a cycle-accurate, open-source simulator for multi-core RISC-V systems. The hardware platforms we target are open-source RISC-V MCUs of the PULP family. One core, designated the fabric controller, orchestrates system operation, while compute-intensive tasks are executed on a PULP cluster of 8 RI5CY cores [96]. System memory is split into two parts. A low-bandwidth L2 memory (parametrized to 512 KiB for MNv1 and 640 KiB for MobileNetV2 to accommodate the larger code size) stores program code and data and is used for partial result storage. The cluster cores operate on 64 KiB of high-bandwidth L1 scratchpad memory, optimized for low access contention. This hierarchical memory structure necessitates *tiled* execution of a network's layers with each tile's input, output, and weight data fitting into the L1 scratchpad. Tiling is automatically performed by DORY. If the total size of a layer's inputs, outputs, and weights exceeds the size of the L2 memory, an off-chip HyperRAM memory is used to store intermediate activations. Off-chip memory is also used to store the weights for all network layers. As our target platforms do not implement data caches and are modeled with a deterministic simulator, latency measurements are taken from single runs of the DORY-generated networks.

All cores in the target system implement the base RV32IMF ISA in addition to the custom XpulpV2 extensions. We evaluate our found configurations on three systems, each containing a cluster whose cores have varying degrees of sub-byte arithmetic support. The first system's cluster implements only XpulpV2, which supports

only 8-bit SIMD arithmetic. We refer to this system as *XpulpV2*. The second system implements the XpulpNN [97] extension, which additionally provides support for packed-SIMD sub-byte arithmetic (for 2- and 4-bit data). While natively supporting sub-byte arithmetic, XpulpNN's sub-byte arithmetic instructions require operands to have equal bit-widths. For operands of mismatching precisions, the lower-precision operands must first be unpacked in software to the larger data size. We refer to this version of the system as *XpulpNNv1*. The third system's cluster implements an improved version of XpulpNN which eliminates the runtime overhead from unpacking lower-precision operands by performing it transparently in hardware. We refer to this version of the system as *XpulpNNv2*. To generate the profiling data used by the heuristic steps of our algorithm, we again use DORY to generate and export dummy networks for all layer types in all precision configurations. The dummy networks sandwich the layer to be evaluated between a convolutional layer with 4 input channels and a FC layer with a single output neuron. The generated application is executed on GVSOC and the execution latency of the middle layer is extracted.

6.4.2 Latency-Accuracy Trade-Offs for XpulpNNv1

MobileNetV1 Figure 6.2a shows the latency-accuracy trade-off for MNv1 deployed to a PULP system with the XpulpNNv1 ISA extensions, with the effect of the free bits heuristic indicated. We observe that the original Bayesian Bits algorithm generally does not produce low-latency configurations. This confirms that even on hardware with native sub-byte arithmetic support, low BOP count does not directly translate to low latency. The modified Bayesian bits enforcing symmetric activation and weight precisions produces configurations which outperform the homogeneous-precision 4b/4b baseline, with small or zero latency penalty, which indicates that asymmetric-precision layers are responsible for the poor performance of the configurations found by the original algorithm. With two exceptions, applying the free bits heuristic improves the latency of all configurations substantially while increasing classification accuracy. For the homogeneous-precision 4 b/4 b baseline, the heuristic increases the precision of 12 layers, improving latency and accuracy by 7%

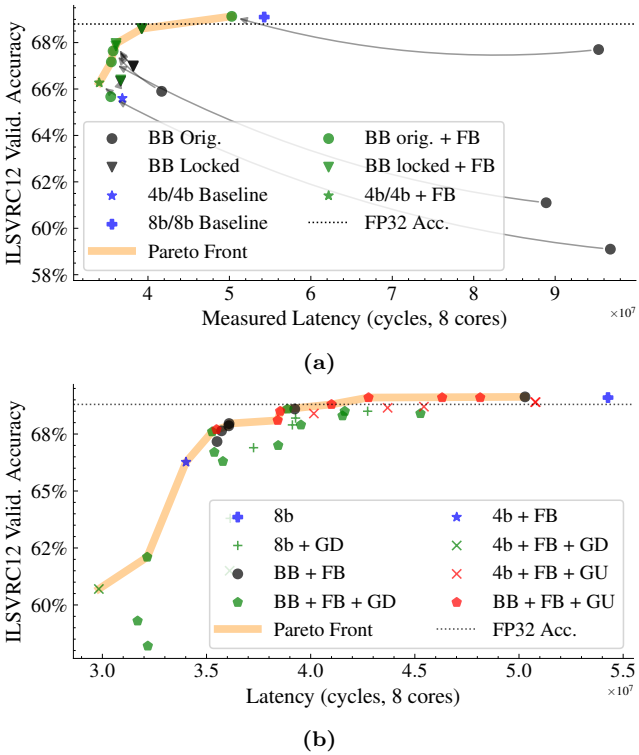


Figure 6.2 – Latency-Accuracy trade-off of mixed-precision MobileNetV1 configurations running on a PULP system with XpulpNNV1 ISA extensions. (a) shows the configurations found by Bayesian Bits before and after applying the free bits heuristic, with grey arrows indicating the effect of applying the heuristic. The graphs reflect the measured end-to-end latency. (b) shows the Pareto-optimal configurations from (a) along with configurations produced by the greedy search described in Section 6.3.3. **BB orig./locked**: Configurations found by the original Bayesian Bits algorithm and the modified version enforcing symmetric activation/weight precisions, respectively. **FB**: Free bits. **GU/GD**: Greedy search in the upward/downward direction.

and 0.7 percentage points, respectively. As symmetric activation and weight precisions are theoretically optimal for XpulpNNv1's hardware implementation of sub-byte arithmetic, this is a non-trivial result. The free bits heuristic lifts the previously uncompetitive configurations found by the original Bayesian Bits algorithm to the Pareto front, leading to accuracy and latency gains of $1.4 - 6.6$ percentage points and $12.3\% - 61.6\%$, respectively. The most accurate configuration matches the 8b/8b baseline in statistical accuracy at 69.1% while reducing execution latency by 7.6%, and the configuration at the Pareto front's knee point improves the execution latency by 27.8% at a classification accuracy within 0.2 percentage points of the 32-bit floating-point baseline of 68.8%.

Figure 6.2b shows the effect of applying the greedy search (see Section 6.3.3) to the configurations produced by Bayesian Bits and the freebie heuristic, as well as the homogeneous-precision baselines. The greedy heuristic produces mostly non-optimal configurations when applied in the downward direction. In contrast, when applied in the upward direction to configurations found by our combined algorithm, it yields Pareto-optimal networks, refining the original Pareto front and finding a configuration that reduces latency by 29.2% at an accuracy drop of only 0.3% from the full-precision baseline. Finally, we note that while the configurations produced by the upward greedy heuristic starting from the 4b/4b baseline are completely dominated by those found by Bayesian Bits and the free bits heuristic, they form a Pareto front which lies within 0.3 percentage points of classification accuracy.

We conclude that i) the differentiable-search step is indeed helpful in finding mixed-precision configurations optimized for low end-to-end latency, ii) the greedy search step applied to these configurations in the upward direction reliably refines the Pareto front, and iii) greedy search which increases layer-wise precisions starting from a low-precision baseline can provide a low-cost alternative to the multi-step procedure.

MobileNetV2 Figure 6.3 shows the latency-accuracy trade-off of MNv2 configurations produced by Bayesian Bits modified with the free bits heuristic running on the XpulpNNv1 system. The baseline 4b/4b configuration contains many asymmetric-precision convolutional layers due to adder node outputs being quantized to 8 bits. This leads

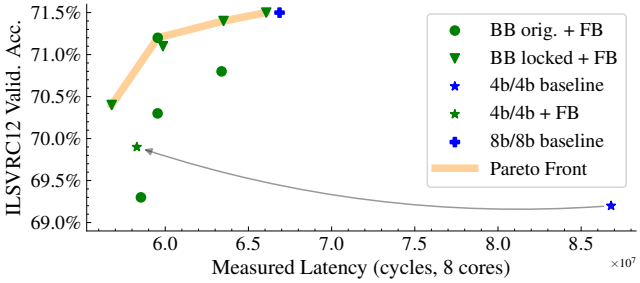


Figure 6.3 – Latency-Accuracy tradeoff of MobileNetV2 configurations optimized for XpulpNNv1. The grey arrow indicates the effect of applying the heuristic to the 4b/4b baseline.

to a latency higher than that of the 8b/8b baseline, which is reduced by the free bits heuristic by 46%. At the same time, classification accuracy is improved by 0.6 percentage points. Nevertheless, the resulting configuration is not Pareto-optimal with respect to those produced by our algorithm. In particular, the locked-precision version of Bayesian Bits, when combined with the free bits heuristic, produces configurations that dominate the optimized 4b/4b baseline and the 8b/8b baseline. The configuration at the Pareto front's knee point reduces execution latency by 10.9% at an accuracy penalty of only 0.3 percentage points relative to the 8b/8b baseline.

6.4.3 Free Bits Across Different Target Platforms

To evaluate the portability of our algorithm, we optimized MNv1 and MNv2 configurations found with Bayesian Bits for the three different PULP systems with different levels of support for sub-byte arithmetic, described in Section 6.4.1. Table 6.1 shows the lowest-latency configurations within 0.5 and 1.5 percentage points of classification accuracy of the 8b/8b baseline. The configurations listed were found with only the first two steps of our algorithm. Notably, our approach achieves latency reductions even on the XpulpV2 system without hardware support for sub-byte arithmetic, which can be attributed to a lower data movement overhead thanks to larger tile sizes. While relative latency reduction and the resulting accuracies are

Acc. Margin	ISA	MobileNetV1		MobileNetV2	
		Lat. vs. 8b	Acc.	Lat. vs. 8b	Acc.
8b Baseline	*all*	+0%	69.1%	+0%	71.5%
0.5 pp.	XPv2	−5.5%	69.3%	−3.4%	71.0%
	XPNNv1	−27.9%	68.6%	−10.9%	71.2%
	XPNNv2	−28.6%	68.6%	−15.3%	71.0%
1.5 pp.	XPv2	−5.5%	69.3%	−6.3%	70.7%
	XPNNv1	−34.4%	67.6%	−15.1%	70.4%
	XPNNv2	−35.1%	67.6%	−15.3%	71.0%
4b + FB	XPv2	−3.5%	67.7%	−7.7%	70.9%
	XPNNv1	−37.1%	66.3%	−12.8%	69.9%
	XPNNv2	−39.8%	66.6%	−25.7%	69.6%
4b Baseline	XPv2	+49.9%	65.6%	+37.0%	69.3%
	XPNNv1	−32.3%	65.6%	+48.9%	69.3%
	XPNNv2	−38.3%	65.6%	−23.4%	69.3%

TABLE 6.1
CONFIGURATIONS WITHIN MARGINS OF 0.5 AND 1.5 PERCENTAGE POINTS (**pp.**) OF 8B/8B CLASSIFICATION ACCURACY FOR PULP SYSTEMS IMPLEMENTING DIFFERENT ISA EXTENSIONS: XPULPV2 (**XPv2**), XPULPNNV1 (**XPNNv1**) AND XPULPNNV2 (**XPNNv2**). THE CONFIGURATIONS LISTED HERE WERE FOUND WITHOUT THE GREEDY HEURISTIC SEARCH STEP. **4b+FB**: TARGET-SPECIFIC FREE BITS HEURISTIC APPLIED TO HOMOGENEOUSLY QUANTIZED 4B/4B NETWORK.

very similar between XpulpNNv1 and XpulpNNv2 on MNv1, significant differences can be observed on MNv2's 4b/4b baselines, both before and after applying the free bits heuristic. On XpulpNNv2, both exhibit significantly lower latency than the 8b/8b baseline, while on XpulpNNv1 the unoptimized baseline is uncompetitive and the optimized configuration is dominated in accuracy and latency by other configurations.

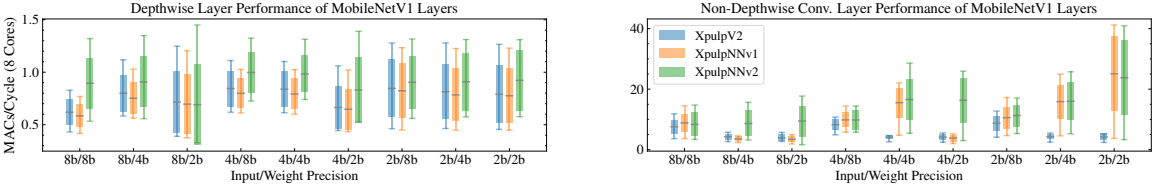
6.4.4 Analysis

To explain the mechanism by which the free bits heuristic reduces latency on different target platforms, it is helpful to consider the results of the layer-wise profiling step, shown in Figure 6.4a. Despite

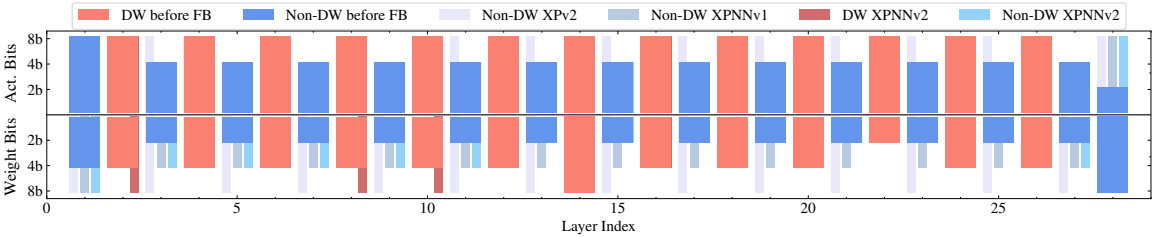
having no native sub-byte arithmetic support, XPv2 sometimes exhibits speedups from reduced precisions due to reduced tiling overheads in the memory-bound depthwise (DW) layers as well as in the early layers of the network, which have the largest activation tensor sizes. Conversely, XPNNv1 supports sub-byte arithmetic, but non-DW layers with lower weight than activation precisions incur overhead from weight unpacking, which results in lower performance. XpulpNNv2 does not see this performance degradation as the unpacking is performed by the hardware with no latency penalty. This explains the action of the free bits heuristic (Figure 6.4b): Activation and weight precisions of non-DW layers are increased to 8 bits on many layers for XPv2 and set to be equal for XPNNv1, while for XPNNv2, many layers are left in asymmetric precision.

6.4.5 Quality of Latency Estimation

As detailed in Sections 6.3.2 and 6.3.3, both the free bits heuristic and the greedy search assume that the network's total latency is roughly equal to the sum of the individual layers' latencies as measured during profiling. For MNv1, this holds, with measured latency on average 10% lower than estimated and all estimations remaining within 15% of the true latency. For MNv2, measured latencies are from 20% to 39% (average: 26%) higher than the estimations. The reason for this lies in MNv2's skip connections, which require some intermediate activations to be buffered. This reduces the space available for input/output activations of the residual layers, leading to smaller tile sizes and increased data movement overhead. Nevertheless, the free bits heuristic reliably reduces MNv2 configurations' latencies, indicating the robustness of the heuristic to tiling-related effects.



(a) Performance of MNv1 layers quantized to different precisions on systems implementing XpulpV2 (**XPv2**), XpulpNNV1 (**XPNNv1**) and XpulpNNV2 (**XPNNv2**). The candlesticks' bodies are centered around the mean throughput and extend to one standard deviation above and below it. The ends of the candlesticks represent the minimum and maximum throughput of all layers in the respective precision combination. Notably, depthwise (**DW**) layers may see a speedup from sub-byte quantization even on XpulpV2, which has no hardware support for sub-byte arithmetic.



(b) Effect of applying the free bits heuristic on a MNv1 configuration found by Bayesian Bits, which are explained by (a): As non-depthwise layers exhibit much lower throughput when a layer's weight precision is lower than that of the input activation, the heuristic increases the weight precision to match that of the activations in those layers.

Figure 6.4 – Profiling results and the effects of the Free Bits heuristic.

6.5 Conclusion

In this chapter, we have presented *Free Bits*, an efficient method to find latency-optimized mixed-precision network configurations for inference on edge devices. Taking advantage of the fact that, depending on the target platform, increasing input or weight precision may lead to lower execution latency, the method optimizes mixed-precision configurations found by the hardware-agnostic Bayesian Bits differentiable search algorithm. To further refine the precision configurations found in this way, a greedy heuristic can be applied. Deploying the MNv1 and MNv2 configurations found with our algorithm on a family of high-performance MCU-class RISC-V platforms, we find that, i) with hardware support for sub-byte arithmetic, MNv1 end-to-end latency can be reduced by 30% while retaining full-precision equivalent accuracy, ii) even without such hardware support, mixed-precision quantization enables a latency reduction of up to 7.7%, and iii) the found configurations offer a superior accuracy-latency trade-off to homogeneous 4-bit and 8-bit quantization.

Chapter 7

Summary and Conclusions

In this concluding chapter, we will review the main insights and results of the thesis, followed by an outlook on potential future research directions in quantized deep learning for edge applications. The potential of QNNs for improving the energy efficiency of neural network inference – generally, and specifically in edge AI applications – was known before this thesis was started, and 8-bit integer quantization has been a key component ini enabling DNN inference on low-power, low-cost, MCU-based platforms. In this thesis, we have explored how to translate the theoretical efficiency advantages of more aggressively quantized DNNs to system-level energy efficiency gains. A major goal of the thesis has been to treat the topic in a holistic manner and to present methods and results that translate well to practical scenarios. This has led us to explore a considerable portion of the QNN development and deployment pipeline. With toolchain support for efficient QNN creation from full-precision networks and a precision configuration search algorithm for arbitrary hardware platforms, we have facilitated the development of QNNs that offer improved energy efficiency. On the hardware and system design side, we have demonstrated the potential of extreme quantization in the form of TNNs with a system

that optimizes the complete sensor-to-inference pipeline by eliminating interface and data transfer bottlenecks. We also showed how lightweight ISA extensions can significantly improve TNN inference overhead on general-purpose cores at negligible silicon area overhead.

In conclusion, we have shown that low-bitwidth and mixed-precision QNNs offer tangible efficiency advantages not only in theory, but also in end-to-end edge AI scenarios. In the following, we will summarize the main results of each part of the thesis.

7.1 Summary of Results

Automated Quantization of Full-Precision Networks

Chapter 3 presented a methodology to automatically convert full-precision DNNs to integer-only QNNs. In the first step, the full-precision network is converted to its fake-quantized counterpart. As part of this conversion, subgraphs that can not be integerized for deployment on the target platform are detected and replaced with integerizable versions in a procedure we call *harmonization*. The harmonized fake-quantized network can then be trained with a variety of QAT algorithms. Finally, the trained network is automatically converted to its integer-only equivalent and exported for deployment to the target platform. In experimental evaluations, we show that on the popular MobileNetV2 [37] architecture, harmonization can prevent substantial accuracy drops of up to 3 pp. resulting from the low-bitwidth quantization of residual adder nodes during integerization.

Gesture Recognition on DVS Data with Ternary Neural Networks

In Chapter 4, we proposed and evaluated a complete sensor-to-classification pipeline for gesture recognition from DVS camera data. The processing strategy most commonly adopted for event-based camera data is to directly process the event stream with a SNN. We take an alternative approach, showing that aggregating the event stream into ternary event frames and processing it with a TNN achieves state-of-the-art classification accuracy, outperforming SNN-based works by

at least 1.7 pp.. We then deploy the complete pipeline to the Kraken SoC, which includes a lightweight DVS camera peripheral that performs the frame aggregation in hardware. The TNN can be run either on the on-chip application-specific CUTIE accelerator or on an 8-core PULP cluster of RISC-V-based microprocessor cores. At 7 µJ, the full-system classification energy when running the TNN on CUTIE is 67× lower than the previous state of the art. The suitability of our approach for realistic application scenarios is underlined by the system power consumption of 4.7 mW (CUTIE)/6.4 mW (cluster) under continuous inference. A key takeaway of Chapter 4 concerns the comparison of application-specific accelerators with general-purpose processing cores in terms of full-system power consumption. The CUTIE TNN accelerator boasts a compute efficiency of hundreds of TOp/J, two orders of magnitude higher than the PULP cluster of RISC-V cores. Nevertheless, running inference on CUTIE only reduces the steady-state system-level power consumption by 27 %. This discrepancy can be attributed to accelerator power consumption during idle phases overshadowing the very short bursts of power consumption during inference.

XpulpTNN: RISC-V ISA Extensions for Efficient TNN Inference

As the previous chapter has shown, ISA-based processing cores are attractive targets for efficient QNN deployment, especially when the silicon area budget does not allow for a dedicated accelerator. In Chapter 5, we present XpulpTNN, a set of instructions that extend the RISC-V ISA with instructions targeted at efficient TNN inference. Additionally, we implement full toolchain and SDK support for XpulpTNN, enabling the development and deployment of TNNs on XpulpTNN-enabled platforms.Compared to optimized 2-bit kernels, XpulpTNN achieves 67 % higher throughput and 57 % higher energy efficiency. These improvements incur a negligible silicon area overhead and do not degrade the longest path. Together with end-to-end software support, XpulpTNN thus enables the efficient execution TNN on resource-constrained edge systems without specialized accelerators.

Free Bits: Finding Latency-Optimized Mixed-Precision Configurations

Aggressively quantized networks such as TNNs exhibit the lowest inference energy in absolute terms. However, extreme quantization leads to unacceptable accuracy drops on more difficult tasks such as 1000-class image classification. While large, inefficient networks can be quantized to very low precisions without a catastrophic impact on their statistical performance, the speed-up from this quantization does not compensate for the inefficient network architecture, and lighter networks quantized to a higher precision offer a better compromise between accuracy and inference energy. Mixed-precision quantization allows the individual quantization of each layer to a different precision to obtain the best trade-off between statistical performance and latency, which is closely proportional to inference energy when considering the power consumption of the complete system. In Chapter 6, we present *Free Bits*, a method to find latency-optimized mixed-precision configurations for a given network topology and target platform. By combining a hardware-agnostic neural architecture search step with a hardware-aware heuristic that increases precision in layers where it reduces the inference latency. The second step of this process is trivial in terms of compute load, allowing users to find latency-optimized configurations for different hardware platforms with only one run of the first, compute-intensive step. By using exact profiling data collected directly on the target platform, the algorithm effectively and directly optimizes the latency for the specific hardware target, inherently taking into account implementation-specific non-idealities that are usually too complex to model analytically. We evaluate Free Bits on the popular, lightweight MobileNetV1/V2 architectures, achieving up to 30 % latency reduction while retaining full-precision-equivalent accuracy.

7.2 Outlook

To conclude the thesis, this section will give a brief outlook on potential directions future research directions. Research into quantized neural networks, particularly CNNs, is in a mature state and the algorithmic

side is well explored by existing work. Similarly, a wide variety of hardware architectures targeting QNN inference have been presented. However, there is a comparative lack of functional systems capable of running end-to-end applications using low- or mixed-precision QNNs. Consequently, the number of such end-to-end applications presented is even lower. To gain a better understanding of the practical potential of quantized deep learning on the edge and its limitations, future research should be conducted with the efficiency of the complete system in mind. That is also the spirit of the following proposals.

Mixed-Precision Acceleration

The mixed-precision capabilities of the system we targeted in Chapter 6 were implemented as ISA extensions in general-purpose cores. This made it a rewarding target for automated optimization, as the dependence between precision and throughput is highly non-monotonic and influenced by so many factors that a handcrafted approach could hardly account for all of them. Nevertheless, a dedicated hardware accelerator with mixed-precision capabilities would achieve higher throughput and energy efficiency. Multiple mixed-precision accelerator designs have been proposed in literature [171], [183]–[185]. However, they suffer from one or multiple shortcomings. The designs have either not been integrated in edge systems, they only operate efficiently on a limited set of kernels or they only support mixed-precision quantization of weights, but not of activations.

To fill this "practicality gap", a promising approach might be the combination of a flexible mixed-precision matrix multiplication accelerator, a powerful data movement and reordering DMA engine and an appropriate number of programmable processing units (which may also the form of RISC-V cores) to perform element-wise and/or non-linear operations. With such a system, the vast majority of current DL models could be supported. Replacing or supplementing the PULP cluster used in multiple chapters throughout this thesis with such a combination could yield a mixed-precision system that is both highly efficient and flexible enough to support the deployment of end-to-end applications.

Low- and Mixed-Precision Transformers

Since their introduction in 2017, transformer models have experienced massive research interest [186]. With the release of transformer-based image and text generation tools such as Dall-E [187] and ChatGPT [188], transformer-based models have also entered the commercial market and the public consciousness, at a scale that CNNs have never reached. The natural language processing (NLP) and generative visual applications that have popularized transformers tend to benefit from very large models with billions to trillions of parameters [189], [190] that are not suitable for edge deployment. However, transformer architectures have also been successfully applied in other fields such as discriminative computer vision tasks [191]. Furthermore, various approaches have been proposed to make transformer inference less resource-intensive [192], [193], and small transformers have been successfully quantized and deployed on edge devices [114], [194]. Quantization to precisions lower than 8 bits has been successfully explored [195], but not in a mobile or edge context. Exploring the limits of low- and mixed-precision quantization on edge-compatible transformers would further lower their resource requirements and enable better trade-offs between inference energy and statistical performance.

Appendix A

Appendix to Chapter 3

A.1 QAT Hyperparameters

	ResNet18	**MobileNetV2**
Epochs	10	13
QAT Algo.	TQT	TQT
Batch Size	240	190
Opt.	SGD	SGD
Momentum	0.9	0.9
Wt. Decay	0.0001	0.00005
LR_0	0.001	0.00075
LR decr. [a]	0.1@4, 0.1@7, 0.3@9	0.1@6, 0.1@9
Wt. Q. start[b]	0	1
Act. Q. start[b]	1	1
Act. clip init.[c]	Perc. 0.5/99.5	Const. 6.0/Max[d]
Wt. clip init.[c]	MSE	Max

[a] a@b indicates that LR is decreased by a factor of a before the start of epoch b, with epochs indexed starting with 1

[b] Activations and weights are quantized starting from specified epoch

[c] Const. x: clipping bounds initialized to x,
Max: clipping bounds initialized to maximum value observed during unquantized training,
Percentile a/b: Clipping bounds set to specified percentile of values observed during unquantized training (upper percentile for unsigned activations, max magnitude of upper and lower percentiles for signed activations),
MSE: Clipping bounds set to minimize MSE between quantized and unquantized values by brute-force optimization

[d] Activations inserted before and after adder nodes have clipping bounds initialized to Max, all others to Const. 6.0

Table A.1
QAT hyperparameters for ResNet and MobileNetV2

Appendix B

Appendix to Chapter 6

B.1 Training Hyperparameters

	MobileNetV1	**MobileNetV2**
Epochs	11	13
Batch Size	256	340
Optimizer	SGD	SGD
Momentum	0.9	0.9
LR_0	0.001	0.00075
LR decr. [a]	4,7	5,8
Quant. start [b]	0	1
Act. clip init. [c]	Const. 6.0	Const. 6.0/Max[d]

[a] LR is decreased by a factor of 0.1 at specified epochs
[b] Activations and weights are quantized starting from specified epoch
[c] Const. x: clipping bounds initialized to x,
Max: clipping bounds initialized to maximum value observed during unquantized training
[d] Activations inserted before and after adder nodes have clipping bounds initialized to Max, all others to Const. 6.0

TABLE B.1
QAT HYPERPARAMETERS FOR MOBILENETV1 AND MOBILENETV2 TRAINED WITH TQT

	MobileNetV1	**MobileNetV2**
Epochs	11	12
Batch Size	256	150
Net Opt.	SGD w/ mom. 0.9	SGD w/ mom. 0.9
$LR_{0,net}$	0.001	0.00075
Prec. Gate Opt.	Adam	Adam
LR_{gate}	0.0001	0.0002
LR decr. [c]	4,7	4,7
Quant. start	0	0
Act. clip init.	Const. 6.0	Const. 6.0
Φ_0 [a]	2.0	2.0
μ_0 [b]	0.01, 0.03, 0.06, 0.12	0.01, 0.03, 0.06, 0.12

[a] Initialization of precision gating parameters
[b] Global regularizer strength
[c] Only network parameters' learning rate is decreased

TABLE B.2
BAYESIAN BITS HYPERPARAMETERS FOR MOBILENETV1 AND MOBILENETV2

B.2 Greedy Latency-Matching Heuristics

Algorithm 3 Greedy Heuristic Precision Search - Latency Reduction

Input:

LD: Latency dictionary mapping layer type t and precisions (b_{in}, b_{wt}) to a measured latency

C_{net}: Dictionary mapping layer indices to layer types and precisions representing a mixed-precision network with latency L_0, of the form $\left\{i : \left(t^i, \left(b^i_{in}, b^i_{wt}\right)\right)\right\}_{i=1}^{N}$ with $\sum_{i=1}^{N} LD[C_{net}[i]] = L_0$

P_{all}: Set of allowed combinations b_{in}, b_{wt} of input and weight precisions

O_p: Ordering of P_{all}, e.g. $\{2\text{b} : 0,\ 4\text{b} : 1,\ 8\text{b} : 2\}$

$L_{tgt} < L_0$: Target latency

Output:

C'_{net}: Latency-optimized mixed-precision configuration of the input network

function DST($(b_{in,1}, b_{wt,1}), (b_{in,2}, b_{wt,2})$) ▷ Returns distance between two layer precisions based on O_p

  return $O_p\left[b_{in,1}\right] - O_p\left[b_{in,2}\right] + O_p\left[b_{wt,1}\right] - O_p\left[b_{wt,2}\right]$

end function

function GET_MOVES(cfg, s_{max}) ▷ Returns configuration with largest latency decrease, decreasing each layer's precision at most by s_{max}

  $moves \leftarrow cfg$ ▷ Initialize with starting configuration

  for all $i, \left(t_i, \left(b^i_{in}, b^i_{wt}\right)\right) \in cfg$ **do** ▷ Iterate over network layers

    $lat^i_0 \leftarrow LD\left[cfg\left[i\right]\right]$ ▷ Initial latency of layer i

    $cands \leftarrow \{(b_{in}, b_{wt}) \mid (b_{in}, b_{wt}) \in P_{all},\ LD\left[(t_i, (b_{in}, b_{wt}))\right] \leq lat^i_0,\ \text{DST}((b_{in}, b_{wt}), \left(b^i_{in}, b^i_{wt}\right)) < s_{max}\}$ ▷ All precisions within s_{max} precision distance with latency $\leq lat^i_0$

    $best \leftarrow \underset{(b_{in}, b_{wt}) \in cands}{\arg\min}\ LD\left[(t_i, (b_{in}, b_{wt}))\right]$ ▷ Select lowest-latency candidate for layer i

    $moves\left[i\right] \leftarrow (t_i, best)$ ▷ Update configuration

  end for

  return $moves$

end function

$C' \leftarrow C$ ▷ Initialize to original net configuration

for all $step_{max} \in [0, ..., 2\left|O_p\right|]$ **do** ▷ Prioritize low-distance modifications

  $moves \leftarrow \text{GET\_MOVES}(C, step_{max})$

  if $\sum_{i=1}^{N} LD\left[moves\left[i\right]\right] < L_{tgt}$ **then** ▷ Modified config. meets target

    while $\sum_{i=1}^{N} LD\left[C'\left[i\right]\right] > L_{tgt}$ **do** ▷ Loop while target not met

      $move_{appl} \leftarrow \underset{i}{\arg\max}\ LD\left[C\left[i\right]\right] - LD\left[moves\left[i\right]\right]$ ▷ Find and apply highest-impact modification

      $C'\left[move_{appl}\right] \leftarrow moves\left[move_{appl}\right]$

      $moves \leftarrow moves \setminus moves\left[move_{appl}\right]$ ▷ Discard applied move

    end while

    return C'

  end if

end for

return Failure ▷ If no configuration was found, declare failure

Algorithm 4 Greedy Heuristic Precision Search - Latency Increase

Input:

LD: As in Algorithm 3
C_{net}: As in Algorithm 3
P_{all}: As in Algorithm 3
$L_{tgt} > L_0$: Target latency

Output:

C'_{net}: Latency-optimized mixed-precision configuration of the input network

function HIGHER($(b_{in,1}, b_{wt,1}), (b_{in,2}, b_{wt,2})$) ▷ True if precision 1 > precision 2
  return $((b_{in,1} > b_{in,2}) \wedge (b_{wt,1} \geq b_{wt,2})) \vee ((b_{in,1} \geq b_{in,2}) \wedge (b_{wt,1} > b_{wt,2}))$
end function
function GET_MOVES(cfg) ▷ Find layer-wise precision increases with smallest latency penalty
  $moves \leftarrow cfg$ ▷ Initialize to input config.
  for all $i, \left(t_i, \left(b^i_{in}, b^i_{wt}\right)\right) \in cfg$ **do** ▷ Loop over network layers
    $cands \leftarrow \{(b_{in}, b_{wt}) \,|\, (b_{in}, b_{wt}) \in P_{all},$ HIGHER$((b_{in}, b_{wt}), \left(b^i_{in}, b^i_{wt}\right))\}$ ▷ Find all higher-or-equal precision configurations
    if $cands \neq \emptyset$ **then**
      $best \leftarrow \underset{(b_{in}, b_{wt}) \in cands}{\arg\min} LD\left[(t_i, (b_{in}, b_{wt}))\right]$ ▷ Lowest-latency move
      $moves\left[i\right] \leftarrow (t_i, best)$ ▷ Modify layer's configuration
    end if
  end for
  return $moves$
end function
$C' \leftarrow C$ ▷ Initialize to original net configuration
while True do ▷ Loop until failure or success
  $moves \leftarrow$ GET_MOVES(C') ▷ Increase precision at minimal latency penalty
  if $moves == C'$ **then**
    return Failure ▷ If no moves are found, the algorithm fails
  end if
  if $\sum_{i=1}^{N} LD\left[moves\left[i\right]\right] < L_{tgt}$ **then** ▷ Moves do not exceed target latency...
    $C' \leftarrow moves$ ▷ ...so apply all of them
  else ▷ Moves can increase latency past the target
    while True do
      $mv_{appl} \leftarrow \arg\min_i LD\left[moves\left[i\right]\right] - LD\left[C'\left[i\right]\right]$ ▷ Lowest-latency move
      if $\sum_{\substack{i=1 \\ i \neq mv_{appl}}}^{N} LD\left[C'\left[i\right]\right] + LD\left[moves\left[mv_{appl}\right]\right] > L_{tgt}$ **then**
        return C' ▷ If the move would exceed target latency, return
      else
        $C'\left[mv_{appl}\right] \leftarrow moves\left[mv_{appl}\right]$ ▷ Otherwise, apply it
      end if
    end while
  end if
end while

Appendix C

Notations and Acronyms

ANN	artificial neural network
ASIC	application-specific integrated circuits
BC	BinaryConnect
BN	batch normalization
BNN	binarized neural network
BOP	binary operation
BWN	binary weight network
CDC	clock domain crossing
CNN	convolutional neural network
CSC	compressed sparse column
CSR	compressed sparse row

CV . . . . . cross-validation

DDPG . . . . . deep deterministic policy gradient

DL . . . . . deep learning

DMA . . . . . direct memory access

DNAS . . . . . differentiable neural architecture search

DNN . . . . . deep neural network

DVS . . . . . dynamic vision sensor

DVSI . . . . . DVS interface

DWS . . . . . depthwise separable

EDA . . . . . Electronic Design Automation

EMA . . . . . exponential moving average

EMG . . . . . electromyography

FC . . . . . fabric controller

FC . . . . . fabric controller

FLL . . . . . frequency-locked loop

FP . . . . . full-precision

FPGA . . . . . field-programmable gate array

FPU . . . . . Floating Point Unit

FQ . . . . . fake-quantized

GE . . . . . gate equivalent

GPU . . . . . graphics processing unit

HtanH . . . . . hard hyperbolic tangent

ICN . . . . . integer channel norm
ILP . . . . . integer linear programming
INQ . . . . . incremental network quantization
IoT . . . . . Internet of Things
ISA . . . . . instruction set architecture

LIF . . . . . leaky integrate-and-fire
LSQ . . . . . learned step size quantization
LUT . . . . . lookup table

M&L . . . . . MAC-and-load
MAC . . . . . multiply-accumulate
MADD . . . . . multiply-add
MCU . . . . . Microcontroller
ML . . . . . machine learning
MLP . . . . . multi-layer perceptron
MNv1 . . . . . MobileNetV1
MNv2 . . . . . MobileNetV2
MSE . . . . . mean square error

NAS . . . . . network architecture search
NLP . . . . . natural language processing

OCU . . . . . output channel compute unit
ONNX . . . . . Open Neural Network Exchange

PACT . . . . . parametrized activation clipping

PPO . . . . . proximal policy optimization

PTQ . . . . . post-training quantization

PULP . . . . . parallel ultra-low power

QAT . . . . . quantization-aware training

QNN . . . . . quantized neural network

ReLU . . . . . rectified linear unit

RF . . . . . register file

RISC . . . . . reduced instruction set computer

RL . . . . . reinforcement learning

SAWB . . . . . statistics-aware weight binning

SGD . . . . . stochastic gradient descent

SIMD . . . . . single instruction multiple data

SNN . . . . . spiking neural network

SoC . . . . . system on chip

SPMD . . . . . single program, multiple data

SRR . . . . . short-range radar

STE . . . . . straight-through estimator

SVM . . . . . support vector machine

TCDM . . . . . tightly-coupled data memory

TCN . . . . . temporal convolutional network

TDP . . . . . thermal design power

TNN . . . . . ternarized neural network

TQT . . . . . trained quantization thresholds

UAV . . . . . unmanned aerial vehicle

VCD . . . . . value change dump

Bibliography

[1] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2022, pp. 11 966–11 976, ISBN: 978-1-6654-6946-3. DOI: 10.1109/CVPR52688.2022.01167.

[2] H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and T. Zhao, "SMART: Robust and Efficient Fine-Tuning for Pre-trained Natural Language Models through Principled Regularized Optimization," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 2177–2190. DOI: 10.18653/v1/2020.acl-main.197.

[3] A. Dubatovka and J. M. Buhmann, "Automatic Detection of Atrial Fibrillation from Single-Lead ECG Using Deep Learning of the Cardiac Cycle," *BME Frontiers*, vol. 2022, pp. 1–12, May 2022, ISSN: 2765-8031. DOI: 10.34133/2022/9813062.

[4] A. Y. Hannun, P. Rajpurkar, M. Haghpanahi, *et al.*, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature Medicine*, vol. 25, no. 1, pp. 65–69, Jan. 2019, ISSN: 1078-8956. DOI: 10.1038/s41591-018-0268-3. arXiv: 1707.01836v1.

[5] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned Step Size Quantization," in *International Conference on Learning Representations*, Feb. 2020, pp. 1–12.

[6] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009, ISSN: 00012475.

[7] O. Russakovsky, J. Deng, H. Su, *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015, ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. arXiv: 1409.0575.

[8] M. Spallanzani, G. Rutishauser, M. Scherer, A. Burrello, F. Conti, and L. Benini, "QuantLab: a Modular Framework for Training and Deploying Mixed-Precision NNs," in *TinyML Summit*, Mar. 2022.

[9] G. Rutishauser, M. Scherer, T. Fischer, and L. Benini, "7 µJ/inference end-to-end gesture recognition from dynamic vision sensor data using ternarized hybrid convolutional neural networks," *Future Generation Computer Systems*, p. 109 231, Jul. 2023, ISSN: 0167739X. DOI: 10.1016/j.future.2023.07.017.

[10] G. Rutishauser, M. Scherer, T. Fischer, and L. Benini, "Ternarized TCN for μJ/Inference Gesture Recognition from DVS Event Frames," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, Mar. 2022, pp. 736–741, ISBN: 978-3-9819263-6-1. DOI: 10.23919/DATE54114.2022.9774592.

[11] G. Rutishauser, F. Conti, and L. Benini, "Free Bits: Latency Optimization of Mixed-Precision Quantized Neural Networks on the Edge," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, IEEE, Jun. 2023, pp. 1–5, ISBN: 979-8-3503-3267-4. DOI: 10.1109/AICAS57966.2023.10168577.

[12] G. Rutishauser, J. Mihali, M. Scherer, and L. Benini, "xTern: Energy-Efficient Ternary Neural Network Inference on RISC-V-Based Edge Systems."

[13] L. Cavigelli, G. Rutishauser, and L. Benini, "EBPC: Extended Bit-Plane Compression for Deep Neural Network Inference and Training Accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 723–734, Dec. 2019, ISSN: 2156-3357. DOI: 10.1109/JETCAS.2019.2950093. arXiv: 1908.11645.

[14] 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*, vol. 41, no. 4, pp. 1020–1033, Apr. 2022, ISSN: 0278-0070. DOI: 10.1109/TCAD.2021.3075420. arXiv: 2011.01713.

[15] M. Scherer, A. Di Mauro, G. Rutishauser, T. Fischer, and L. Benini, "A 1036 TOp/s/W, 12.2 mW, 2.72 μJ/Inference All Digital TNN Accelerator in 22 nm FDX Technology for TinyML Applications," in *2022 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, IEEE, Apr. 2022, pp. 1–3, ISBN: 978-1-6654-1989-5. DOI: 10.1109/COOLCHIPS54332.2022.9772668.

[16] G. Rutishauser, R. Hunziker, A. Di Mauro, S. Bian, L. Benini, and M. Magno, "ColibriES: a Milliwatts RISC-V Based Embedded System Leveraging Neuromorphic and Neural Networks Hardware Accelerators for Low-Latency Closed-loop Control Applications," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, Ed., 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA: IEEE Computer Society Press, 2023, pp. 1–5. DOI: https://doi.org/10.1109/ISCAS46773.2023.10181726.

[17] F. Conti, G. Paulin, A. Garofalo, *et al.*, "Marsellus: A Heterogeneous RISC-V AI-IoT End-Node SoC With 2–8 b DNN Acceleration and 30%-Boost Adaptive Body Biasing," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 1, pp. 128–142, Jan. 2024, ISSN: 0018-9200. DOI: 10.1109/JSSC.2023.3318301.

[18] A. Nadalini, G. Rutishauser, A. Burrello, *et al.*, "A 3 TOPS/W RISC-V Parallel Cluster for Inference of Fine-Grain Mixed-Precision Quantized Neural Networks," in *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, IEEE, Ed., 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA: IEEE Computer Society Press, 2023, pp. 1–6. DOI: https://doi.org/10.1109/ISVLSI59464.2023.10238679.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. arXiv: 1102.0183.

[20] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Computer Society Conference on*

Computer Vision and Pattern Recognition, vol. 07-12-June, 2015, pp. 1–9, ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594.

[21] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *Proceedings - 2015 International Conference on Learning Representations*, 2015, pp. 1–14.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Dec, pp. 770–778, 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

[23] J. Hu, J. Lu, and Y. P. Tan, "Discriminative deep metric learning for face verification in the wild," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, 2014, pp. 1875–1882, ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.242.

[24] G. B. Huang, H. Lee, and E. Learned-Miller, "Learning hierarchical representations for face verification with convolutional deep belief networks," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2518–2525, 2012, ISSN: 10636919. DOI: 10.1109/CVPR.2012.6247968.

[25] A. van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in Neural Information Processing Systems*, C. J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26, Curran Associates, Inc., 2013.

[26] X. Geng, H. Zhang, J. Bian, and T. S. Chua, "Learning image and user features for recommendation in social networks," in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, IEEE, 2015, pp. 4274–4282, ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.486.

[27] T. Sercu, C. Puhrsch, B. Kingsbury, and Y. LeCun, "Very deep multilingual convolutional neural networks for LVCSR," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, vol. 2016-May, IEEE, Mar. 2016, pp. 4955–4959, ISBN: 978-1-4799-9988-0. DOI: 10.1109/ICASSP.2016.7472620.

[28] O. Abdel-Hamid, A. R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014, ISSN: 15587916. DOI: 10.1109/TASLP.2014.2339736.

[29] 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," *J. Mach. Learn. Res.*, vol. 22, no. 1, Jan. 2021, ISSN: 1532-4435.

[30] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," *Advances in Neural Information Processing Systems*, no. Nips, pp. 1387–1395, 2016, ISSN: 10495258. arXiv: 1608.04493.

[31] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 4856–4864, 2016, ISSN: 10636919. DOI: 10.1109/CVPR.2016.525. arXiv: 1512.01891.

[32] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016, pp. 1–14.

[33] D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952, ISSN: 0096-8390. DOI: 10.1109/JRPROC.1952.273898.

[34] A. Aimar, H. Mostafa, E. Calabrese, *et al.*, "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, ISSN: 21622388. DOI: 10.1109/TNNLS.2018.2852335. arXiv: 1706.01406.

[35] A. Parashar, M. Rhu, A. Mukkara, *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *Proceedings - International Symposium on Computer Architecture*, vol. Part F1286, pp. 27–40, 2017, ISSN: 10636897. DOI: 10.1145/3079856.3080254. arXiv: 1708.04485.

[36] A. G. Howard, M. Zhu, B. Chen, *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017. DOI: 10.48550/arXiv.1704.04861. arXiv: 1704.04861.

[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2018, pp. 4510–4520, ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00474.

[38] P. Ren, Y. Xiao, X. Chang, *et al.*, "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions," *ACM Comput. Surv.*, vol. 54, no. 4, May 2021, ISSN: 0360-0300. DOI: 10.1145/3447582.

[39] B. Wu, X. Dai, P. Zhang, *et al.*, "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search," 2018. arXiv: 1812.03443.

[40] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-Aware Neural Architecture Search for Mobile," 2018. DOI: arXiv:1807.11626v1. arXiv: 1807.11626.

[41] A. Howard, M. Sandler, B. Chen, *et al.*, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2019, pp. 1314–1324, ISBN: 978-1-7281-4803-8. DOI: 10.1109/ICCV.2019.00140.

[42] T. J. Yang, A. Howard, B. Chen, *et al.*, "NetAdapt: Platform-aware neural network adaptation for mobile applications," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12, vol. 11214 LNCS, 2018, pp. 289–304, ISBN: 9783030012489. DOI: 10.1007/978-3-030-01249-6_18. arXiv: 1804.03230.

[43] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *36th International Conference on Machine Learning, ICML 2019*, Jun. 2019, pp. 10 691–10 700, ISBN: 9781510886988.

[44] O. Sentieys, "Approximate Computing for DNN," in *CSW 2021 - HiPEAC Computing Systems Week*, Lyon, France, Oct. 2021.

[45] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, vol. 57, IEEE, Feb. 2014, pp. 10–14, ISBN: 978-1-4799-0920-9. DOI: 10.1109/ISSCC.2014.6757323.

[46] D. Przewlocka-Rus, S. S. Sarwar, H. E. Sumbul, Y. Li, and B. De Salvo, *Power-of-Two Quantization for Low Bitwidth and Hardware Compliant Neural Networks*. Association for Computing Machinery, 2022, vol. 1. arXiv: 2203.05025.

[47] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights," in *International Conference on Learning Representations*, 2017.

[48] X. Sun, J. Choi, C. Y. Chen, *et al.*, "Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 1–10.

[49] B. Noune, P. Jones, D. Justus, D. Masters, and C. Luschi, "8-bit Numerical Formats for Deep Neural Networks," pp. 1–30, 2022. arXiv: 2206.02915.

[50] N. P. Jouppi, C. Young, N. Patil, *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings - International Symposium on Computer Architecture*, vol. Part F1286, New York, New York, USA: ACM Press, 2017, pp. 1–12, ISBN: 9781450348928. DOI: 10.1145/3079856.3080246.

[51] N. P. Jouppi, D. H. Yoon, G. Kurian, *et al.*, "A domain-specific supercomputer for training deep neural networks," *Communications of the ACM*, vol. 63, no. 7, pp. 67–78, 2020, ISSN: 15577317. DOI: 10.1145/3360307.

[52] STMicroelectronics, *X-CUBE-AI Data Brief*, Available at https://www.st.com/resource/en/data_brief/x-cube-ai.pdf, Feb. 2023.

[53] NXP Semiconductors, *i.MX Machine Learning User's Guide*, English, version LF6.1.22_2.0.0, Available at https://www.nxp.com/docs/en/user-guide/IMX-MACHINE-LEARNING-UG.pdf, NXP Semiconductors, Aug. 1, 2023, 111 pp.

[54] Arm Ltd., *Arm Ethos-U65 Technical Reference Manual*, English, version Revision r0p0, Available at https://developer.arm.com/documentation/102023/0000/?lang=en, Arm Ltd., Jul. 28, 2023, 125 pp., August 19, 2022.

[55] Arm Ltd., *Armv8-M Architecture Reference Manual*, English, version Version B.t, Available at https://developer.arm.com/documentation/ddi0553/bw/?lang=en, Arm Ltd., Jun. 30, 2022, 2140 pp., September 26, 2022.

[56] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, 2015. arXiv: 1502.03167.

[57] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018, ISSN: 0146-4833. DOI: 10.1109/CVPR.2017.690. arXiv: 1804.02767.

[58] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, "Squeeze-and-Excitation Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 2011–2023, Aug. 2020, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2019.2913372. arXiv: 1709.01507.

[59] S. R. Jain, A. Gural, M. Wu, and C. H. Dick, "Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks," in *Proceedings of Machine Learning and Systems*, 2020, pp. 112–128.

[60] M. Risso, A. Burrello, L. Benini, E. Macii, M. Poncino, and D. J. Pagliari, "Channel-wise Mixed-precision Assignment for DNN Inference on Constrained Edge Nodes," in *2022 IEEE 13th International Green and Sustainable Computing Conference (IGSC)*, IEEE, Oct. 2022, pp. 1–6, ISBN: 978-1-6654-6550-2. DOI: 10.1109/IGSC55832.2022.9969373.

[61] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," pp. 1–12, 2013. arXiv: 1308.3432.

[62] M. Courbariaux, Y. Bengio, and J. P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in Neural Information Processing Systems*, pp. 3123–3131, Nov. 2015, ISSN: 10495258. arXiv: 1511.00363.

[63] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. DOI: 10.5555/2627435.2670313.

[64] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout Networks," in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., vol. 28, Atlanta, Georgia, USA: PMLR, 2013, pp. 1319–1327.

[65] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of Neural Networks using DropConnect," in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., vol. 28, Atlanta, Georgia, USA: PMLR, 2013, pp. 1058–1066.

[66] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29, Curran Associates, Inc., 2016.

[67] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017, ISSN: 1532-4435. DOI: 10.5555/3122009.3242044. arXiv: 1609.07061.

[68] M. Rastegari, V.-n. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9908 LNCS, 2016, pp. 525–542, ISBN: 9783319464923. DOI: 10.1007/978-3-319-46493-0_32. arXiv: arXiv:1603.05279v4.

[69] J. Choi, S. Venkataramani, V. Srinivasan, K. Gopalakrishnan, Z. Wang, and P. Chuang, "Accurate and Efficient 2-bit Quantized Neural Networks," in *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., vol. 1, 2019, pp. 348–359.

[70] A. Amir, B. Taba, D. Berg, *et al.*, "A Low Power, Fully Event-Based Gesture Recognition System," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 27, IEEE, Jul. 2017, pp. 7388–7397, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.781.

[71] Google LLC, *TensorFlow Lite Quantization Specification*, Available at https://www.tensorflow.org/lite/performance/quantization_spec, May 2023.

[72] R. David, J. Duke, A. Jain, *et al.*, "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems," in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 800–811.

[73] Advanced Micro Devices, Inc., *Vitis AI User Guide*, version 3.5, Available at `https://docs.xilinx.com/api/khub/maps/hr1oyp3fbsYqqO85WhA_HQ/attachments/6jTo22b8iWtV9yXFSuFxaA/content`, Sep. 5, 2023, June 29, 2023.

[74] M. Haroush, I. Hubara, E. Hoffer, and D. Soudry, "The Knowledge Within: Methods for Data-Free Model Compression," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8491–8499. DOI: 10.1109/CVPR42600.2020.00852.

[75] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A Novel Zero Shot Quantization Framework," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2020, pp. 13 166–13 175, ISBN: 978-1-7281-7168-5. DOI: 10.1109/CVPR42600.2020.01318.

[76] D. Lee, M. Cho, S. Lee, J. Song, and C. Choi, "Data-free mixed-precision quantization using novel sensitivity metric," 2021. arXiv: 2103.10051.

[77] A. Mordvintsev, C. Olah, and M. Tyka, *Inceptionism: Going Deeper into Neural Networks*, 2015.

[78] M. Nagel, R. A. Amjad, M. van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *37th International Conference on Machine Learning, ICML 2020*, vol. PartF16814, 2020, pp. 7154–7163, ISBN: 9781713821120.

[79] Y. Li, R. Gong, X. Tan, *et al.*, "BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction," in *International Conference on Learning Representations*, 2021, pp. 1–16.

[80] M. Nagel, M. Fournarakis, Y. Bondarenko, and T. Blankevoort, "Overcoming Oscillations in Quantization-Aware Training," in *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, Eds., vol. 162, PMLR, 2022, pp. 16 318–16 330.

[81] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving Post Training Neural Quantization: Layer-wise Calibration and Integer Programming," 2020. arXiv: 2006.10518.

[82] V. Sze, Y.-H. Chen, T.-J. 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, Dec. 2017, ISSN: 0018-9219. DOI: 10.1109/JPROC.2017.2761740. arXiv: arXiv:1703.09039v1.

[83] 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)*, IEEE, Sep. 2019, pp. 1–9, ISBN: 978-1-7281-5020-8. DOI: 10.1109/HPEC.2019.8916327.

[84] 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)*, IEEE, Sep. 2020, pp. 1–12, ISBN: 978-1-7281-9219-2. DOI: 10.1109/HPEC43674.2020.9286149.

[85] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "AI Accelerator Survey and Trends," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, Sep. 2021, pp. 1–9, ISBN: 978-1-6654-2369-4. DOI: 10.1109/HPEC49654.2021.9622867.

[86] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "AI and ML Accelerator Survey and Trends," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, IEEE, Sep. 2022, pp. 1–10, ISBN: 978-1-6654-9786-2. DOI: 10.1109/HPEC55821.2022.9926331.

[87] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A survey of FPGA-based neural network inference accelerators," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 1, pp. 1–26, 2019, ISSN: 19367414. DOI: 10.1145/3289185.

[88] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN Inference on ASICs: A Survey," *Journal of Systems Architecture*, vol. 113, no. September 2020, p. 101 887, Feb. 2021, ISSN: 13837621. DOI: 10.1016/j.sysarc.2020.101887.

[89] K. Guo, L. W., K. Zhong, *et al.*, *Neural Network Accelerator Comparison*, https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/.

[90] Intel Corporation, *Intel FPGA AI Suite IP Reference Manual*, version 2023.3, Available at https://www.intel.com/content/www/us/en/docs/programmable/768974/2023-3/reference-manual.html, Dec. 1, 2023, January 24, 2024.

[91] M. Rusci, A. Capotondi, and L. Benini, "Memory-Driven Mixed Low Precision Quantization for Enabling Deep Network Inference on Microcontrollers," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., vol. 2, 2020, pp. 326–335.

[92] L. Geiger and P. Team, "Larq: An Open-Source Library for Training Binarized Neural Networks," *Journal of Open Source Software*, vol. 5, no. 45, p. 1746, 2020. DOI: 10.21105/joss.01746.

[93] F. Sakr, R. Berta, J. Doyle, H. Younes, A. De Gloria, and F. Bellotti, "Memory Efficient Binary Convolutional Neural Networks on Microcontrollers," *Proceedings - IEEE International Conference on Edge Computing*, vol. 2022-July, pp. 169–177, 2022, ISSN: 27679918. DOI: 10.1109/EDGE55608.2022.00032.

[94] G. Cerutti, L. Cavigelli, R. Andri, M. Magno, E. Farella, and L. Benini, "Sub-mW Keyword Spotting on an MCU: Analog Binary Feature Extraction and Binary Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 5, pp. 2002–2012, 2022, ISSN: 15580806. DOI: 10.1109/TCSI.2022.3142525. arXiv: 2201.03386.

[95] A. Waterman, Y. Lee, R. Avizienis, H. Cook, D. Patterson, and K. Asanovic, "The RISC-V instruction set," in *2013 IEEE Hot Chips 25 Symposium (HCS)*, IEEE, Ed., 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA: IEEE Computer Society Press, 2013, p. 1. DOI: https://doi.org/10.1109/HOTCHIPS.2013.7478332.

[96] M. Gautschi, P. D. Schiavone, A. Traber, *et al.*, "Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2017.2654506.

[97] A. Garofalo, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "XpulpNN: Enabling Energy Efficient and Flexible Inference of Quantized Neural Networks on RISC-V Based IoT End Nodes," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1489–1505, Jul. 2021, ISSN: 2168-6750. DOI: 10.1109/TETC.2021.3072337. arXiv: 2011.14325.

[98] T. Dupuis, Y. Fournier, M. AskariHemmat, *et al.*, "Sparq: A Custom RISC-V Vector Processor for Efficient Sub-Byte Quantized Inference," in *2023 21st IEEE Interregional NEWCAS Conference (NEWCAS)*, IEEE, Ed., 1109 Spring Street, Suite 300, Silver Spring,

MD 20910, USA: IEEE, Jun. 2023, pp. 1–5, ISBN: 979-8-3503-0024-6. DOI: 10.1109/NEWCAS57931.2023.10198172.

[99] M. AskariHemmat, T. Dupuis, Y. Fournier, *et al.*, "Quark: An Integer RISC-V Vector Processor for Sub-Byte Quantized DNN Inference," 2023, ISSN: 02714310. DOI: 10.1109/ISCAS46773.2023.10181985. arXiv: 2302.05996.

[100] J. Won, J. Si, S. Son, T. J. Ham, and J. W. Lee, "ULPPACK: Fast Sub-8-bit Matrix Multiply on Commodity SIMD Hardware," in *Proceedings of Machine Learning and Systems*, D. Marculescu, Y. Chi, and C. Wu, Eds., vol. 4, 2022, pp. 52–63.

[101] M. Rusci, L. Cavigelli, and L. Benini, "Design Automation for Binarized Neural Networks: A Quantum Leap Opportunity?" In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2018-May, IEEE, 2018, pp. 1–5, ISBN: 978-1-5386-4881-0. DOI: 10.1109/ISCAS.2018.8351807.

[102] 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," in *2018 IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, Apr. 2018, pp. 1–4, ISBN: 978-1-5386-2483-8. DOI: 10.1109/CICC.2018.8357071.

[103] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, 2019, ISSN: 10495258. arXiv: 1912.01703.

[104] M. Abadi, A. Agarwal, P. Barham, *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015.

[105] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "DORY: Automatic End-to-End Deployment of Real-World DNNs on Low-Cost IoT MCUs," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1253–1268, Aug. 2021, ISSN: 0018-9340. DOI: 10.1109/TC.2021.3066883. arXiv: 2008.07127.

[106] T. M. Ingolfsson, M. Hersche, X. Wang, N. Kobayashi, L. Cavigelli, and L. Benini, "EEG-TCNet: An Accurate Temporal Convolutional Network for Embedded Motor-Imagery Brain–Machine Interfaces," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, Oct. 2020, pp. 2958–2965, ISBN: 978-1-7281-8526-2. DOI: 10.1109/SMC42975.2020.9283028.

[107] M. Scherer, M. Magno, J. Erb, P. Mayer, M. Eggimann, and L. Benini, "TinyRadarNN: Combining Spatial and Temporal Convolutional Neural Networks for Embedded Gesture Recognition With Short Range Radars," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 336–10 346, Jul. 2021, ISSN: 2327-4662. DOI: 10.1109/JIOT.2021.3067382. arXiv: 2006.16281.

[108] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," in *7th International Conference on Learning Representations, ICLR 2019*, 2019, pp. 1–13.

[109] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, "Spiking Neural Networks: A Survey," *IEEE Access*, vol. 10, pp. 60 738–60 764, 2022, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3179968.

[110] A. Basu, L. Deng, C. Frenkel, and X. Zhang, "Spiking Neural Network Integrated Circuits: A Review of Trends and Future Directions," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*, IEEE, Apr. 2022, pp. 1–8, ISBN: 978-1-6654-0756-4. DOI: 10.1109/CICC53496.2022.9772783.

[111] G. Gallego, T. Delbruck, G. Orchard, *et al.*, "Event-Based Vision: A Survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 1, pp. 154–180, 2022, ISSN: 19393539. DOI: 10.1109/TPAMI.2020.3008413. arXiv: 1904.08405.

[112] S. U. Innocenti, F. Becattini, F. Pernici, and A. Del Bimbo, "Temporal Binary Representation for Event-Based Action Recognition," in *2020 25th International Conference on Pattern Recognition (ICPR)*, IEEE, Jan. 2021, pp. 10 426–10 432, ISBN: 978-1-7281-8808-9. DOI: 10.1109/ICPR48806.2021.9412991.

[113] R. Zhang, X. Jing, S. Wu, C. Jiang, J. Mu, and F. Richard Yu, "Device-Free Wireless Sensing for Human Detection: The Deep Learning Perspective," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2517–2539, Feb. 2021, ISSN: 23274662. DOI: 10.1109/JIOT.2020.3024234.

[114] A. Burrello, M. Scherer, M. Zanghieri, F. Conti, and L. Benini, "A Microcontroller is All You Need: Enabling Transformer Execution on Low-Power IoT Endnodes," in *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, IEEE, Aug. 2021, pp. 1–6, ISBN: 978-1-6654-3156-9. DOI: 10.1109/COINS51742.2021.9524173.

[115] E. Ceolini, C. Frenkel, S. B. Shrestha, *et al.*, "Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing," *Frontiers in Neuroscience*, vol. 14, no. August, pp. 1–15, Aug. 2020, ISSN: 1662-453X. DOI: 10.3389/fnins.2020.00637.

[116] R. Massa, A. Marchisio, M. Martina, and M. Shafique, "An Efficient Spiking Neural Network for Recognizing Gestures with a DVS Camera on the Loihi Neuromorphic Processor," in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Jul. 2020, pp. 1–9, ISBN: 978-1-7281-6926-2. DOI: 10.1109/IJCNN48605.2020.9207109.

[117] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008, ISSN: 0018-9200. DOI: 10.1109/JSSC.2007.914337.

[118] C. Li, L. Longinotti, F. Corradi, and T. Delbruck, "A 132 by 104 10μm-Pixel 250μW 1kefps Dynamic Vision Sensor with Pixel-Parallel Noise and Spatial Redundancy Suppression," in *2019 Symposium on VLSI Circuits*, vol. 2019-June, IEEE, Jun. 2019, pp. C216–C217, ISBN: 978-4-86348-720-8. DOI: 10.23919/VLSIC.2019.8778050.

[119] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking Keyword Spotting Efficiency on Neuromorphic Hardware," in *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, New York, NY, USA: ACM, Mar. 2019, pp. 1–8, ISBN: 9781450361231. DOI: 10.1145/3320288.3320304.

[120] S. Benatti, F. Casamassima, B. Milosevic, *et al.*, "A Versatile Embedded Platform for EMG Acquisition and Gesture Recognition," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 9, no. 5, pp. 620–630, Oct. 2015, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2015.2476555.

[121] S. Tam, M. Boukadoum, A. Campeau-Lecours, and B. Gosselin, "A Fully Embedded Adaptive Real-Time Hand Gesture Classifier Leveraging HD-sEMG and Deep Learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 14, no. 2, pp. 232–243, Apr. 2020, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2019.2955641.

[122] M. Zanghieri, S. Benatti, A. Burrello, V. Kartsch, F. Conti, and L. Benini, "Robust Real-Time Embedded EMG Recognition Framework Using Temporal Convolutional Networks on a Multicore IoT Processor," *IEEE Transactions on Biomedical Circuits and*

Systems, vol. 14, no. 2, pp. 244–256, Apr. 2020, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2019.2959160.

[123] S. Benatti, G. Rovere, J. Bosser, *et al.*, "A sub-10mW real-time implementation for EMG hand gesture recognition based on a multi-core biomedical SoC," in *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, IEEE, Jun. 2017, pp. 139–144, ISBN: 978-1-5090-6707-7. DOI: 10.1109/IWASI.2017.7974234.

[124] S. Benatti, F. Montagna, V. Kartsch, A. Rahimi, D. Rossi, and L. Benini, "Online Learning and Classification of EMG-Based Gestures on a Parallel Ultra-Low Power Platform Using Hyperdimensional Computing," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 3, pp. 516–528, Jun. 2019, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2019.2914476.

[125] J. Lien, N. Gillian, M. E. Karagozler, *et al.*, "Soli," *ACM Transactions on Graphics*, vol. 35, no. 4, pp. 1–19, Jul. 2016, ISSN: 0730-0301. DOI: 10.1145/2897824.2925953.

[126] S. Wang, J. Song, J. Lien, I. Poupyrev, and O. Hilliges, "Interacting with Soli," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, New York, NY, USA: ACM, Oct. 2016, pp. 851–860, ISBN: 9781450341899. DOI: 10.1145/2984511.2984565.

[127] D. W. O. Antillon, C. R. Walker, S. Rosset, and I. A. Anderson, "Glove-Based Hand Gesture Recognition for Diver Communication," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2022, ISSN: 2162-237X. DOI: 10.1109/TNNLS.2022.3161682.

[128] X. Huang, Q. Wang, S. Zang, *et al.*, "Tracing the Motion of Finger Joints for Gesture Recognition via Sewing RGO-Coated Fibers Onto a Textile Glove," *IEEE Sensors Journal*, vol. 19, no. 20, pp. 9504–9511, Oct. 2019, ISSN: 1530-437X. DOI: 10.1109/JSEN.2019.2924797.

[129] F. Zhou, X. Li, and Z. Wang, "Efficient High Cross-User Recognition Rate Ultrasonic Hand Gesture Recognition System," *IEEE Sensors Journal*, vol. 20, no. 22, pp. 13 501–13 510, Nov. 2020, ISSN: 1530-437X. DOI: 10.1109/JSEN.2020.3004252.

[130] Y. Qifan, T. Hao, Z. Xuebing, L. Yin, and Z. Sanfeng, "Dolphin: Ultrasonic-Based Gesture Recognition on Smartphone Platform," in *2014 IEEE 17th International Conference on Computational Science and Engineering*, IEEE, Dec. 2014, pp. 1461–1468, ISBN: 978-1-4799-7981-3. DOI: 10.1109/CSE.2014.273.

[131] H. Lakhotiya, H. S. Pandita, and R. Shankarmani, "Real Time Sign Language Recognition Using Image Classification," in *2021 2nd International Conference for Emerging Technology (INCET)*, IEEE, May 2021, pp. 1–4, ISBN: 978-1-7281-7029-9. DOI: 10.1109/INCET51464.2021.9456432.

[132] S. Hussain, R. Saxena, X. Han, J. A. Khan, and H. Shin, "Hand gesture recognition using deep learning," in *2017 International SoC Design Conference (ISOCC)*, IEEE, Nov. 2017, pp. 48–49, ISBN: 978-1-5386-2285-8. DOI: 10.1109/ISOCC.2017.8368821.

[133] C. Frenkel, M. Lefebvre, J.-D. Legat, and D. Bol, "A 0.086-mm^2 12.7-pJ/SOP 64k-Synapse 256-Neuron Online-Learning Digital Spiking Neuromorphic Processor in 28nm CMOS," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 1, pp. 1–1, 2018, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2018.2880425.

[134] C. Frenkel, J.-D. Legat, and D. Bol, "MorphIC: A 65-nm 738k-Synapse/mm^2 Quad-Core Binary-Weight Digital Neuromorphic Processor With Stochastic Spike-Driven Online Learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 5, pp. 999–1010, Oct. 2019, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2019.2928793.

[135] Y. Xing, G. Di Caterina, and J. Soraghan, "A New Spiking Convolutional Recurrent Neural Network (SCRNN) With Applications to Event-Based Hand Gesture Recognition," *Frontiers in Neuroscience*, vol. 14, no. November, 2020, ISSN: 1662453X. DOI: 10.3389/fnins.2020.590164.

[136] S. B. Shrestha and G. Orchard, "SLAYER: Spike Layer Error Reassignment in Time," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, vol. 2018-Decem, Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 1419–1428. DOI: 10.5555/3326943.3327073.

[137] L. Cordone, B. Miramond, and S. Ferrante, "Learning from Event Cameras with Sparse Spiking Convolutional Neural Networks," in *2021 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Jul. 2021, pp. 1–8, ISBN: 978-1-6654-3900-8. DOI: 10.1109/IJCNN52387.2021.9533514.

[138] W. Fang, Z. Yu, Y. Chen, T. Masquelier, T. Huang, and Y. Tian, "Incorporating Learnable Membrane Time Constant to Enhance Learning of Spiking Neural Networks," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, Oct. 2021,

pp. 2641–2651, ISBN: 978-1-6654-2812-5. DOI: 10.1109/ICCV48922.2021.00266.

[139] M. A. Mahovald and C. Mead, "The Silicon Retina," *Scientific American*, vol. 264, pp. 76–83, 1991.

[140] C. Brandli, R. Berner, Minhao Yang, Shih-Chii Liu, and T. Delbruck, "A 240 × 180 130 dB 3 µs Latency Global Shutter Spatiotemporal Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, Oct. 2014, ISSN: 0018-9200. DOI: 10.1109/JSSC.2014.2342715.

[141] T. Finateu, A. Niwa, D. Matolin, *et al.*, "5.10 A 1280×720 Back-Illuminated Stacked Temporal Contrast Event-Based Vision Sensor with 4.86µm Pixels, 1.066GEPS Readout, Programmable Event-Rate Controller and Compressive Data-Formatting Pipeline," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, IEEE, Feb. 2020, pp. 112–114, ISBN: 978-1-7281-3205-1. DOI: 10.1109/ISSCC19947.2020.9063149.

[142] D. P. Moeys, C. Li, J. N. Martel, *et al.*, "Color temporal contrast sensitivity in dynamic vision sensors," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, May 2017, pp. 1–4, ISBN: 978-1-4673-6853-7. DOI: 10.1109/ISCAS.2017.8050412.

[143] A. Zhu, L. Yuan, K. Chaney, and K. Daniilidis, "EV-FlowNet: Self-Supervised Optical Flow Estimation for Event-based Cameras," in *Robotics: Science and Systems XIV*, Robotics: Science and Systems Foundation, Jun. 2018, ISBN: 978-0-9923747-4-7. DOI: 10.15607/RSS.2018.XIV.062.

[144] F. Paredes-Valles, K. Y. W. Scheper, and G. C. H. E. de Croon, "Unsupervised Learning of a Hierarchical Spiking Neural Network for Optical Flow Estimation: From Events to Global Motion Perception," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 8, pp. 2051–2064, Aug. 2020, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2019.2903179.

[145] G. Haessig, A. Cassidy, R. Alvarez, R. Benosman, and G. Orchard, "Spiking Optical Flow for Event-Based Sensors Using IBM's TrueNorth Neurosynaptic System," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 4, pp. 860–870, Aug. 2018, ISSN: 1932-4545. DOI: 10.1109/TBCAS.2018.2834558.

[146] Y. Li, Y. Guo, S. Zhang, S. Deng, Y. Hai, and S. Gu, "Differentiable Spike: Rethinking Gradient-Descent for Training Spiking Neural Networks," in *Advances in Neural Information Processing Systems*, A. Beygelzimer and Y. Dauphin and P. Liang and J. Wortman Vaughan, Ed., 2021, pp. 1–14.

[147] J. H. Lee, T. Delbruck, and M. Pfeiffer, "Training Deep Spiking Neural Networks Using Backpropagation," *Frontiers in Neuroscience*, vol. 10, no. November, Nov. 2016, ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00508.

[148] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades," *Frontiers in Neuroscience*, vol. 9, no. November, pp. 1–11, Nov. 2015, ISSN: 1662-453X. DOI: 10.3389/fnins.2015.00437.

[149] H. Li, H. Liu, X. Ji, G. Li, and L. Shi, "CIFAR10-DVS: An Event-Stream Dataset for Object Classification," *Frontiers in Neuroscience*, vol. 11, no. May, pp. 1–10, May 2017, ISSN: 1662-453X. DOI: 10.3389/fnins.2017.00309.

[150] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 ×128 1.5% Contrast Sensitivity 0.9% FPN 3 µs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, Mar. 2013, ISSN: 0018-9200. DOI: 10.1109/JSSC.2012.2230553.

[151] A. Kolesnikov, L. Beyer, X. Zhai, *et al.*, "Big Transfer (BiT): General Visual Representation Learning," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 491–507, ISBN: 978-3-030-58558-7. DOI: 10.1007/978-3-030-58558-7_29. arXiv: 1912.11370.

[152] M. Davies, N. Srinivasa, T. H. Lin, *et al.*, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018, ISSN: 02721732. DOI: 10.1109/MM.2018.112130359.

[153] G. Orchard, E. P. Frady, D. B. D. Rubin, *et al.*, "Efficient Neuromorphic Signal Processing with Loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*, IEEE, Oct. 2021, pp. 254–259, ISBN: 978-1-6654-0144-9. DOI: 10.1109/SiPS52927.2021.00053.

[154] M. V. Debole, B. Taba, A. Amir, *et al.*, "TrueNorth: Accelerating From Zero to 64 Million Neurons in 10 Years," *Computer*, vol. 52, no. 5, pp. 20–29, 2019, ISSN: 15580814. DOI: 10.1109/MC.2019.2903009.

[155] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-Net: Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Networks*, vol. 100, pp. 49–58, Apr. 2018, ISSN: 08936080. DOI: 10.1016/j.neunet.2018.01.010. arXiv: 1705.09283.

[156] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Petrot, "Ternary neural networks for resource-efficient AI applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, IEEE, May 2017, pp. 2547–2554, ISBN: 978-1-5090-6182-2. DOI: 10.1109/IJCNN.2017.7966166.

[157] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A Survey of Quantization Methods for Efficient Neural Network Inference," in *Low-Power Computer Vision*, Boca Raton: Chapman and Hall/CRC, Jan. 2022, pp. 291–326, ISBN: 9781003162810. DOI: 10.1201/9781003162810-13. arXiv: arXiv:2103.13630v3.

[158] B. 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, Apr. 2020, ISSN: 0018-9219. DOI: 10.1109/JPROC.2020.2976475.

[159] M. Scherer, A. D. Mauro, T. Fischer, G. Rutishauser, and L. Benini, "TCN-CUTIE: A 1,036-TOp/s/W, 2.72-µJ/Inference, 12.2-mW All-Digital Ternary Accelerator in 22-nm FDX Technology," *IEEE Micro*, vol. 43, no. 1, pp. 42–48, Jan. 2023, ISSN: 0272-1732. DOI: 10.1109/MM.2022.3226630. arXiv: arXiv:2212.00688v1.

[160] A. Di Mauro, M. Scherer, D. Rossi, and L. Benini, "Kraken: A Direct Event/Frame-Based Multi-sensor Fusion SoC for Ultra-Efficient Visual Processing in Nano-UAVs," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, IEEE, Aug. 2022, pp. 1–19, ISBN: 978-1-6654-6028-6. DOI: 10.1109/HCS55958.2022.9895621.

[161] A. Pullini, D. Rossi, G. Haugou, and L. Benini, "μDMA: An autonomous I/O subsystem for IoT end-nodes," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, vol. 1, IEEE, Sep. 2017, pp. 1–8, ISBN: 978-1-5090-6462-5. DOI: 10.1109/PATMOS.2017.8106971.

[162] A. Di Mauro, A. S. Prasad, Z. Huang, M. Spallanzani, F. Conti, and L. Benini, "SNE: an Energy-Proportional Digital Accelerator for Sparse Event-Based Convolutions," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, Mar. 2022, pp. 825–830, ISBN: 978-3-9819263-6-1. DOI: 10.23919/DATE54114.2022.9774552.

[163] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts," in *International Conference on Learning Representations*, 2017, pp. 1–16, ISBN: 978-0-674-02343-7.

[164] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2164, 2020, ISSN: 1364503X. DOI: 10.1098/rsta.2019.0155. arXiv: 1908.11263.

[165] O. Muller, A. Prost-Boucle, A. Bourge, and F. Petrot, "Efficient Decompression of Binary Encoded Balanced Ternary Sequences," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1962–1966, Aug. 2019, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2019.2906678.

[166] C. Banbury, V. J. Reddi, P. Torelli, *et al.*, "MLPerf Tiny Benchmark," *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

[167] R. Andri, G. Karunaratne, L. Cavigelli, and L. Benini, "ChewBaccaNN: A Flexible 223 TOPS/W BNN Accelerator," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, May 2021, pp. 1–5, ISBN: 978-1-7281-9201-7. DOI: 10.1109/ISCAS51556.2021.9401214.

[168] S. Jain, S. K. Gupta, and A. Raghunathan, "TiM-DNN: Ternary In-Memory Accelerator for Deep Neural Networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 7, pp. 1567–1577, Jul. 2020, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2020.2993045. arXiv: 1909.06892.

[169] P. Jokic, S. Emery, and L. Benini, "Battery-Less Face Recognition at the Extreme Edge," in *2021 19th IEEE International New Circuits and Systems Conference (NEWCAS)*, IEEE, Jun. 2021, pp. 1–4, ISBN: 978-1-6654-2429-5. DOI: 10.1109/NEWCAS50681.2021.9462787.

[170] D. Metz, V. Kumar, and M. Själander, "BISDU: A Bit-Serial Dot-Product Unit for Microcontrollers," *ACM Transactions on Embedded Computing Systems*, Jul. 2023, ISSN: 1539-9087. DOI: 10.1145/3608447.

[171] M. Scherer, M. Eggimann, A. D. Mauro, *et al.*, "Siracusa: A Low-Power On-Sensor RISC-V SoC for Extended Reality Visual Processing in 16nm CMOS," in *49th European Solid State Circuits Conference (ESSCIRC)*, IEEE, Sep. 2023, pp. 217–220, ISBN: 979-8-3503-0420-6. DOI: 10.1109/ESSCIRC59616.2023.10268718.

[172] A. Garofalo, G. Ottavi, A. di Mauro, *et al.*, "A 1.15 TOPS/W, 16-Cores Parallel Ultra-Low Power Cluster with 2b-to-32b Fully Flexible Bit-Precision and Vector Lockstep Execution Mode," in *47th European Solid State Circuits Conference (ESSCIRC)*, IEEE, Sep. 2021, pp. 267–270, ISBN: 978-1-6654-3751-6. DOI: 10.1109/ESSCIRC53450.2021.9567767.

[173] M. van Baalen, C. Louizos, M. Nagel, *et al.*, "Bayesian bits: Unifying quantization and pruning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 5741–5752.

[174] Z. Cai and N. Vasconcelos, "Rethinking Differentiable Search for Mixed-Precision Neural Networks," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2020, pp. 2346–2355, ISBN: 978-1-7281-7168-5. DOI: 10.1109/CVPR42600.2020.00242.

[175] M. Nikolic, G. B. Hacene, C. Bannon, *et al.*, "BitPruning: Learning Bitlengths for Aggressive and Accurate Quantization," 2020. arXiv: 2002.03090.

[176] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search," pp. 1–11, 2018. arXiv: 1812.00090.

[177] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization With Gumbel-Softmax," in *International Conference on Learning Representations*, 2017, pp. 1–13.

[178] Z. Yao, Z. Dong, Z. Zheng, *et al.*, "HAWQV3: Dyadic Neural Network Quantization," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., vol. 139, PMLR, 2021, pp. 11 875–11 886.

[179] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2019-June, IEEE, Jun. 2019, pp. 8604–8612, ISBN: 978-1-7281-3293-8. DOI: 10.1109/CVPR.2019.00881.

[180] A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, A. Yazdanbakhsh, and H. Esmaeilzadeh, "ReLeQ : A Reinforcement Learning Approach for Automatic Deep Quantization of Neural Networks," *IEEE Micro*, vol. 40, no. 5, pp. 37–45, Sep. 2020, ISSN: 0272-1732. DOI: 10.1109/MM.2020.3009475.

[181] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," pp. 1–15, 2018. arXiv: 1805.06085.

[182] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A White Paper on Neural Network Quantization," 2021. arXiv: 2106.08295.

[183] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. J. 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, ISSN: 00189200. DOI: 10.1109/JSSC.2018.2865489.

[184] Y. Umuroglu, L. Rasnayake, and M. Sjalander, "BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, IEEE, Aug. 2018, pp. 307–3077, ISBN: 978-1-5386-8517-4. DOI: 10.1109/FPL.2018.00059.

[185] H. Sharma, J. Park, N. Suda, *et al.*, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, Jun. 2018, pp. 764–775, ISBN: 978-1-5386-5984-7. DOI: 10.1109/ISCA.2018.00069.

[186] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017.

[187] A. Ramesh, M. Pavlov, G. Goh, *et al.*, "Zero-Shot Text-to-Image Generation," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., vol. 139, PMLR, 2021, pp. 8821–8831.

[188] OpenAI, *ChatGPT [Large Language Model]*, `https://chat.openai.com/chat`, Jan. 2024.

[189] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, vol. 1, Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 4171–4186, ISBN: 9781950737130. DOI: `10.18653/v1/N19-1423`.

[190] T. Brown, B. Mann, N. Ryder, *et al.*, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 1877–1901.

[191] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *International Conference on Learning Representations*, 2021.

[192] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers Are RNNs: Fast Autoregressive Transformers With Linear Attention," in *Proceedings of the 37th International Conference on Machine Learning*, H. D. III and A. Singh, Eds., PMLR, Jul. 2020, pp. 5156–5165.

[193] S. Luo, S. Li, T. Cai, *et al.*, "Stable, Fast and Accurate: Kernelized Attention with Relative Positional Encoding," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, Eds., vol. 34, Curran Associates, Inc., 2021, pp. 22 795–22 807.

[194] S. Wyatt, D. Elliott, A. Aravamudan, *et al.*, "Environmental Sound Classification with Tiny Transformers in Noisy Edge Environments," in *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, IEEE, Jun. 2021, pp. 309–314, ISBN: 978-1-6654-4431-6. DOI: `10.1109/WF-IoT51360.2021.9596007`.

[195] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate Quantization for Generative Pre-trained Transformers," in *The Eleventh International Conference on Learning Representations*, 2023.

Curriculum Vitæ

Georg Rutishauser was born in Zürich, Switzerland on May 19, 1993 and grew up in nearby Winterthur, where he currently lives. He obtained his BSc and MSc degrees in Electrical Engineering and Information Technology from ETH Zürich in 2016 and 2018, respectively. In 2019, he joined the Digital Circuits and Systems group at the Integrated Systems Laboratory of ETH Zürich to pursue a PhD degree under the supervision of Prof. Luca Benini. His doctoral research focused on the application quantized neural networks for efficient embedded systems and involved circuit design on the component and system levels, as well as algorithmic research. He successfully defended his thesis in May 2024.