Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

INGEGNERIA ELETTRONICA, TELECOMUNICAZIONI E TECNOLOGIE DELL'INFORMAZIONE

Ciclo 35

Settore Concorsuale: 09/E3 - ELETTRONICA

Settore Scientifico Disciplinare: ING-INF/01 - ELETTRONICA

OPTIMIZING AI AT THE EDGE: FROM NETWORK TOPOLOGY DESIGN TO MCU DEPLOYMENT

Presentata da: Alessio Burrello

Coordinatore Dottorato

Aldo Romani

Supervisore

Luca Benini

Co-supervisore

Davide Rossi

Esame finale anno 2023

ALMA MATER STUDIORUM - UNIVERSITY OF BOLOGNA

Optimizing AI at the Edge: from network topology design to MCU deployment

by

Alessio Burrello

A thesis submitted for the degree of
Doctor of Philosophy

in the

Faculty of Engineering
Department of Electrical, Electronic and Information Engineering "G. Marconi (DEI)

December 2022

Acknowledgements

I first would like to thank my supervisor Luca Benini for his support, inputs, and all the help he gave me in these years.

I would like to thank Simone Benatti, Francesco Conti, and Davide Brunelli, who always provided essential suggestions for projects that I did during my Ph.D.

I also thank all the people of the research group in Bologna, in Torino, and in KU Leuven that shared this incredible journey with me and helped me explore many different aspects of research and life.

Finally, I thank my parents and my friends who supported me during these three years. A special thank goes to my brother, Jacopo Burrello, which inspired me in this career since 2018 when we did our first research work together.

Abstract

Optimizing and deploying artificial intelligence on edge devices to remove the necessity of cloud computing systems and sending data over networks is vital for reducing energy consumption and improving privacy. This thesis will describe two essential knobs to optimize the so-called EdgeAI.

The first topic analyzed in the thesis will be Neural Architecture Search (NAS). NAS is quickly becoming the go-to approach to optimize the structure of Deep Learning (DL) models. I will focus on two different tools that I developed, one to optimize the architecture of Temporal Convolutional Networks (TCNs), a convolutional model for time-series processing that has recently emerged, and one to optimize the data precision of tensors inside CNNs. The first NAS proposed explicitly targets the optimization of the most peculiar architectural parameters of TCNs, namely dilation, receptive field, and the number of features in each layer. Note that this is the first NAS that explicitly targets these networks. The second NAS proposed instead focuses on finding the most efficient data format for a target CNN, with the granularity of the layer filter. Note that applying these two NASes in sequence allows an "application designer" to minimize the structure of the neural network employed, minimizing the number of operations or the memory usage of the network.

After that, the second topic described is the optimization of neural network deployment on edge devices. Importantly, exploiting edge platforms' scarce resources is critical for NN efficient execution on MCUs. To do so, I will introduce *DORY* (*Deployment Oriented to memoRY*) – an automatic tool to deploy CNNs on low-cost MCUs. DORY, in different steps, can manage different levels of memory inside the MCU automatically, offload the computation workload (i.e., the different layers of a neural network) to dedicated hardware accelerators, and automatically generates ANSI C code that orchestrates off- and on-chip transfers with the computation phases. On top of this, I will introduce two optimized computation libraries that DORY can exploit to deploy TCNs and Transformers on edge efficiently.

I conclude the thesis with two different applications on bio-signal analysis, i.e., heart rate tracking and sEMG-based gesture recognition. In these two applications, I will show the employment of previously described techniques as fundamental blocks for optimizing the execution of these tasks on edge.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Learning (DL) models are at the core of many time-series processing applications. Notable examples are audio classification [3], bio-signals analysis [4–7] and predictive maintenance [8, 9]. For many years, the state-of-the-art DL models for time-series-based tasks have been Recurrent Neural Networks (RNNs) [10]. Recently, however, new architectures have been proposed as viable alternatives to RNNs, such as Attention-based Transformers and Temporal Convolutional Networks (TCNs) [11]. Transformer networks are based on the attention mechanism, which puts in the relationship every time-point of the input. These networks are quickly becoming state-of-the-art, outclassing all competitors in terms of accuracy also in this field. These results are obtained thanks to models such as BERT [12], and GPT-3 [13], which include hundreds of millions or even billions of parameters. TCNs are uni-dimensional Convolutional Neural Networks (CNNs) specialized for time series, which have been shown to provide accuracy comparable to RNNs, while providing computational advantages, namely higher arithmetic intensity, smaller memory footprint, and more data reuse opportunities [11]. Thanks to these features, TCNs are particularly interesting for edge computing applications, where inference is directly executed on Internet of Things (IoT) edge devices rather than on centralized servers in the cloud.

However, dedicated optimization steps are required to execute both Transformers and TCN networks on edge. On-device inference requires indeed small and efficient DL models compatible with edge nodes' limited memory spaces and tight energy budgets. On the other hand, selecting an efficient model is usually insufficient to meet such tight constraints. This thesis will show two key optimization steps for shrinking, optimizing, and deploying neural network execution on edge devices.

The first step is the optimization of architectural hyperparameters and the decision of the arithmetic precision of the tensors so that the resulting network occupies as

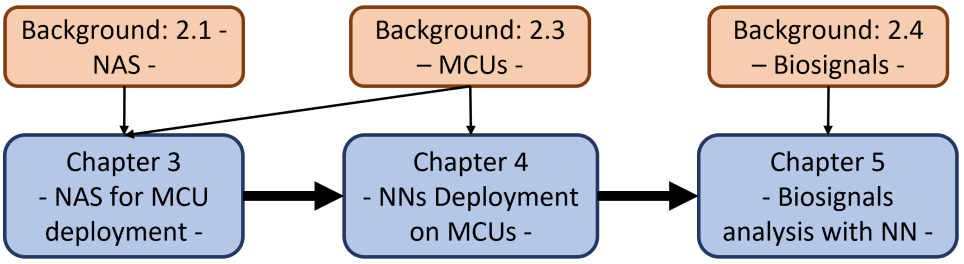

Figure 1.1: Flow of the following thesis. Four background topics introduce the three main chapters of the thesis, i) NAS for efficient deployment on MCUs (Chapter 3), ii) deployment of DNNs on MCUs (Chapter 4), and iii) biosignal analysis with deep neural networks on edge (Chapter 5).

low memory and performs as few operations – with the simplest format – as possible to reach the desired accuracy level. Nowadays, rather than manually, such architectural optimization is increasingly performed with automatic Neural Architecture Search (NAS) tools. A plethora of different NAS approaches has been proposed in the last few years, and several of these works have targeted edge devices [14–18]. However, none of them has focused specifically on models for time-series processing (i.e., TCNs), nor specifically on the data format for weights and intermediate activations. Although there are NAS tools, initially designed for 2D CNNs, that can be easily extended to explore TCNs-related parameters and data format, they do so in a coarse-grain way. Specifically, they create a different copy of all network layers for each setting [18]. This approach results in high memory- and time-consuming searches, requiring hundreds of GPU hours even for relatively simple tasks, which translates into large energy wastes and CO2 emissions. In contrast, in this thesis, I will show two innovative DmaskingNAS, which can perform a fine-grain search of hyperparameters within a single relatively fast training: the first, *Pruning in Time (PiT)*, is tailored explicitly for TCNs. It explores dilation, receptive field, and the number of channels. The second has been designed to explore mixed-precision quantization, i.e. the association of each different tensor of a DNN with a different data format (e.g., int2, int4, int8, etc...). Noteworthy, these tools can also be extended to work on different type of networks, with a small amount of modifications. However, in this thesis I will only describe the main scope of these NASes, and why they have been initially designed.

After applying these optimization steps, the output is a fixed topology with fixed data formats. However, to port it to edge devices without support for any OS, we still need i) to map it onto layer primitives and ii) to manage the memory system of the devices to maximize execution performance. The deployment of DL-based algorithms on IoT devices indeed demands aggressive hardware, software, and algorithmic co-optimization to exploit the scarce resources on these systems to the maximum degree [19]. For instance, on the hardware side, accelerators [20–22] and instruction set

architecture (ISA) extensions [23] that exploit low-precision data formats have been introduced to speed up the computation, lessen the impact of memory constraints, and minimize energy consumption. A typical architecture often couples a conventional MCU with an accelerator [24, 25] and employs multi-level hierarchies of on- and off-chip memories. In some cases, they do away entirely with energy-expensive coherent data caches, exploiting manually managed scratchpad memories instead to maximize area and energy efficiency. Therefore, the main contribution of the thesis towards integrating small DNNs on state-of-the-art MCU nodes will be DORY (Deployment Oriented to memoRY), which automatically generates ANSI C code for a target network. The generated code i) optimally manages data transfers between different memory levels and ii) double-buffer them with an optimized execution of the micro-kernels, e.g., convolutions. While DORY has been initially thought for 2D convolutional neural networks, given the focus on time-series-based application, I will also introduce two new specialized collections of kernels that are integrated into DORY: PULP-NN-1D, a collection of optimized convolutional kernels for 1D convolutions, and TinyFormers, a collection of micro-kernels for the optimization of the sub-nodes inside attention layers. Thanks to these two libraries, I expanded the scope of DORY, bringing to the edge different 1D applications. Note that DORY could be also similarly extended to other architectures and platforms, such as heterogeneous hardware platforms, but it is outside the scope of this work.

In the last chapter of this thesis, "Biosignal analysis with deep neural networks on edge", I demonstrate how the tools and libraries that I introduced are exploited to optimize real-world tasks (I consider two bio-signal based temporal tasks, but these tools can be applied to any application domain). As reported, I used them to optimize and reach cutting-edge results in terms of accuracy and performance on real hot applications. For instance, in heart-rate estimation using photoplethysmographic (PPG) signals, Deep Learning (DL) is increasingly applied. Compared to classical solutions, it allows for a better generalization, despite several problems related to DL. On the other hand, deploying accurate models for heart-rate estimation on wrist-worn devices, typically based on Microcontrollers (MCUs), is far from trivial [26]. I will show that thanks to the tools introduced to optimize the topology and to optimally map networks on MCU resources, we can efficiently deploy such solutions on different platforms, maximizing the battery life of devices.

In the next section, I will detail all the contributions.

1.1 Contributions

In this thesis, I introduce three main important research directions:

- **Neural Architecture Search (NAS) for Efficient Deployment on MCUs**: first, I will describe a NAS algorithm called Pruning in Time (PIT) to optimize Temporal Convolutional Networks, 1D Convolutional Neural Networks characterized by the presence of the *dilation* (a fixed gap between the activations that are convolved with weights). The algorithm targets three crucial hyperparameters, the receptive field, the dilation, and the channels of each layer. I frame their optimization as a *structured weight pruning*, in which additional trainable masking parameters are added to different layers' weights so that their binarized values encode valid settings of the architectural hyperparameters. These masks are then trained with a regularizer to reduce the model complexity as much as possible while preserving accuracy. I consider two different regularizers to explore both the reduction of the number of parameters and of the number of inference operations. These regularizers allow me to enlarge and enrich the collection of Pareto architectures found by this NAS. In Sec. 3.2.5, I will show PIT results on four benchmarks relative to real-world time-series processing tasks where TCNs are commonly employed and for which a deployment on edge devices is relevant: (i) PPG-Based Heart-Rate Monitoring; (ii) ECG-based Arrhythmia Detection; (iii) sEMG-based Hand-Gesture Recognition; (iv) Keyword Spotting. Results show that PIT can find multiple Pareto-optimal architectures starting from a single seed network, achieving 15.9-152× parameter reduction while maintaining the same accuracy of the seed. Furthermore, the approach shown dominates three popular NAS tools developed for computer vision, thanks to the exploration of a larger search space. The results of some relevant networks deployed to two platforms, the multicore GAP8 IoT processor [27], and the single-core STM32H7 MCU [28] are shown to demonstrate the suitability of the NAS for edge deployment. At iso-accuracy, solutions found by PIT reduce energy consumption and latency up to 5.45× on GAP8 and up to 3.83× on the STM32H7, compared to hand-tuned networks. After, I will describe a second NAS algorithm that explores for the first time the potential of a *channel-wise mixed-precision assignment*. Note that it can be used on top of any architectural NAS (e.g., PIT). This NAS explores the space of all possible bit-width associated with each channel of each weight tensor in a Convolutional Neural Network (CNN) while maintaining layer-wise quantization granularity for activations. Again, a lightweight gradient-based search method, which belongs to the family of *Differentiable* NAS (DNAS), is used. With experiments on the four benchmarks of the MLPerf Tiny suite [29] I show that the tool can find a rich front of Pareto-optimal solutions. When deployed on the MPIC [30] RISC-V edge processor, the networks found by this DNAS can reduce memory footprint/energy up to **63%/27%** at iso-accuracy compared to a per-layer mixed-precision search. Finally, I will show an additional improvement to these NASes by reformulating

the problem (shown in Fig. 3.13) of optimization to allow them to find a set of Pareto-optimal architectures in the accuracy vs OPs space, under a fixed model size constraint. Note that when applied to a problem, these tools' scope is to minimize the resource utilization of a device while keeping the highest possible accuracy.

- **Deployment of Deep Neural Networks on MCUs**: After optimizing the topology and the data format, I here explore how to deploy the topologies found on edge devices, such as MCUs. I describe DORY (Deployment Oriented to memoRY), a tool for multi-level memory tiling aiming at fitting realistically sized DNNs on memory-constrained MCUs. Relying on Constraint Programming (CP) optimization, this tool matches on- and off-chip memory hierarchy constraints with DNN geometrical requirements, such as the relationships between input, weight, and output tensor dimensions to create a tiling solution for each network layer. DORY includes a set of heuristics to maximize the performance of the CP solution on PULP platforms using dedicated backend libraries [25, 31, 32]. It also includes a code generator, which uses tiling solutions to produce ANSI C code for the target platforms, with data L3-L2-L1 orchestration implemented as fully pipelined, triple-buffered DMA transfers and integrated calls to the computational backend. Finally, it includes many backends for classical convolutional networks [25], transformers architectures [31], and 1D convolutional networks [32]. I evaluate the performance and energy efficiency of the deployed networks produced by DORY on GWT GAP-8, considering both single layers and end-to-end networks. DORY achieves up to 18.1× better MAC/cycle than the state-of-the-art result on a conventional cache-based microcontroller, the STM32-H743 MCU, in single 2D-convolutional layer execution. Using DORY, end-to-end deployment of 8-bit quantized networks such as *0.5-MobileNet-v1-192*, achieve up to 8.00 MACs/cycle, with a 13.2× improvement compared to the same networks running on the STM32-H743 using the state-of-the-art ST X-CUBE-AI. Furthermore, on a layer-by-layer basis, DORY can achieve up to 2.5× better throughput than the proprietary GWT AutoTiler, on the same GAP-8 platform and up to 27% better performance on full network execution. On 1D-convolutional layers, I obtain up to 17.2 MAC/cycles, with an average performance improvement on single layers of 9.7× compared to the proprietary backend for GAP8 and up to 354× compared to the Cube-AI toolkit on an STM32H7 microcontroller (MCU). On real-world TCNs, DORY demonstrates a throughput of up to 1.11 GMAC/s and an energy efficiency of 21.79 GMAC/s/W. On Attention layers, comparing my novel kernels library for inference operations with implementations based on SoA public libraries, PULP-NN [25], and CMSIS-NN [33], on the STM32H7, the STM32L4,

and GAP8, I obtain a speed-up of 3.4×, 1.8×, and 2.1×, respectively, reaching 0.61, 0.18, and 11.3 MAC/cycle.

- **Biosignal analysis with deep neural networks on edge**: In this last chapter, I leveraged previously introduced tools to improve the accuracy, energy efficiency, and latency of bio-oriented applications. In detail, I will show two main applications. The first is the heart rate (HR) estimation starting from photoplethysmographic signals. The second is the recognition of gestures from surface electromyographic signals. For the HR, first, I leveraged Neural Architecture Search (NAS) tools presented in Chapter 3 to obtain TimePPG, a collection of Pareto-optimal TCN architectures that predict a user's HR based on raw PPG and acceleration data. All TCNs are automatically derived from a single seed architecture [34]. With respect to [35], which only optimized the number of feature maps in each TCN layer, in this work, I extend the search also to consider the *dilation* parameter of convolutional layers, which effectively reduces the model complexity with a limited impact on accuracy. Finally, I applied the NAS presented in Sec. 3.3, to select the best data representation format for the networks' parameters and intermediate input/outputs. Thanks to the hardware-friendly *quantization* used by that tool, I further reduced model size and latency, thus enriching and improving the Pareto frontier. On PPGDalia, the best performing model obtained with the proposed flow, coupled with simple smoothing post-processing, achieves a Mean Absolute Error (MAE) of 4.36 BPM, and includes ≈ 269k trainable parameters. With an additional fine-tuning step, the MAE is further reduced to 3.61 BPM. After quantization to 8bits and deployment on the STM32WB55, the smallest model with a MAE < 8 BPM and the most accurate one consume 1.79 mJ and 47.65 mJ per inference, with a latency of 71.6 ms, and 1.9 s, and an error of 7.73 BPM and 4.41 BPM, respectively. These two models are respectively 32154.3-145.63× smaller and require 3711.1-19.6× fewer operations per inference compared to the previous state-of-the-art DL solution [36], while also significantly improving the HR tracking accuracy. For the gesture recognition task, I introduce a novel DL network topology, the *Bioformer*, which exploits the attention mechanism to reduce computational complexity while achieving state-of-the-art gesture recognition results. My results show a clear advantage of using an initial 1D-convolutional layer to aggregate raw signals in a series of projections to feed the transformer network. I deployed this network using DORY and the attention kernel extensions presented in Sec. 4.4. Testing this new architecture on the Ninapro DB6 dataset, which includes eight grasp gestures from 10 subjects, I achieve 62.34% accuracy, further improved to 65.73% thanks to the inter-subject pre-training. Quantized to

8bits, it occupies as little as 94.2 kB, which is 4.9× lower than previous state-of-the-art CNN, TEMPONet [34, 37], achieving 65.0% on the same task. Deployed on the GAP8 multicore MCU, the same Bioformer only consumes 0.139 mJ per inference, being 8.0× more efficient than TEMPONet.

The rest of the thesis is organized as follows. Chapter 2 introduces the concepts crucial for understanding the thesis's contribution. Chapter 3 and Chapter 4 present tools which are needed to optimize the accuracy and the deployment of Deep Neural Networks (DNNs). Finally, Chapter 5 shows two applications of these tools in the bio-signal processing field. Figure 1.1 shows the dependencies between the different thesis chapters.

1.2 List of Publications

The following articles published in either international journals or conferences are directly discussed in the thesis dissertation:

1. **Burrello, Alessio**; Conti, Francesco; Garofalo, Angelo; Rossi, Davide; Benini, Luca; *Work-in-progress: DORY: Lightweight memory hierarchy management for deep NN inference on IoT endnodes*, 2019 International Conference on Hardware/-Software Codesign and System Synthesis (CODES+ ISSS), 2019.

2. **Burrello, Alessio**; Garofalo, Angelo; Bruschi, Nazareno; Tagliavini, Giuseppe; Rossi, Davide; Conti, Francesco; *Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus*, IEEE Transactions on Computers, 2021.

3. Risso, Matteo; **Burrello, Alessio**; Pagliari, Daniele Jahier; Benatti, Simone; Macii, Enrico; Benini, Luca; Pontino, Massimo; *Robust and energy-efficient PPG-based heart-rate monitoring*, 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021.

4. **Burrello, Alessio**; Dequino, Alberto; Pagliari, Daniele Jahier; Conti, Francesco; Zanghieri, Marcello; Macii, Enrico; Benini, Luca; Poncino, Massimo; *TCN mapping optimization for ultra-low power time-series edge inference*, 2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), 2021.

5. **Burrello, Alessio**; Scherer, Moritz; Zanghieri, Marcello; Conti, Francesco; Benini, Luca; *A microcontroller is all you need: Enabling transformer execution on low-power iot endnodes*, 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), 2021.

6. **Burrello, Alessio**; Pagliari, Daniele Jahier; Risso, Matteo; Benatti, Simone; Macii, Enrico; Benini, Luca; Poncino, Massimo; *Q-ppg: energy-efficient ppg-based heart rate monitoring on wearable devices*, IEEE Transactions on Biomedical Circuits and Systems, 2021.

7. Risso, Matteo; **Burrello, Alessio**; Pagliari, Daniele Jahier; Conti, Francesco; Lamberti, Lorenzo; Macii, Enrico; Benini, Luca; Poncino, Massimo; *Pruning In Time (PIT): A Lightweight Network Architecture Optimizer for Temporal Convolutional Networks*, 2021 58th ACM/IEEE Design Automation Conference (DAC), 2021.

8. **Burrello, Alessio**; Pagliari, Daniele Jahier; Rapa, Pierangelo Maria; Semilia, Matilde; Risso, Matteo; Polonelli, Tommaso; Poncino, Massimo; Benini, Luca; Benatti, Simone; *Embedding temporal convolutional networks for energy-efficient PPG-based heart rate monitoring*, ACM Transactions on Computing for Healthcare (HEALTH), 2022. .

9. **Burrello, Alessio**; Morghet, Francesco Bianco; Scherer, Moritz; Benatti, Simone; Benini, Luca; Macii, Enrico; Poncino, Massimo; Pagliari, Daniele Jahier; *Bioformers: embedding transformers for ultra-low power sEMG-based gesture recognition*, "2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)", 2022.

10. Risso, Matteo; **Burrello, Alessio**; Conti, Francesco; Lamberti, Lorenzo; Chen, Yukai; Benini, Luca; Macii, Enrico; Poncino, Massimo; Pagliari, Daniele Jahier; *Lightweight Neural Architecture Search for Temporal Convolutional Networks at the Edge*, IEEE Transactions on Computers, 2022.

11. Risso, Matteo; **Burrello, Alessio**; Benini, Luca; Macii, Enrico; Poncino, Massimo; Jahier Pagliari, Daniele; *Multi-Complexity-Loss DNAS for Energy-Efficient and Memory-Constrained Deep Neural Networks*, Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design, 2022.

12. Risso, Matteo; **Burrello, Alessio**; Benini, Luca; Macii, Enrico; Poncino, Massimo; Pagliari, Daniele Jahier; *Channel-wise Mixed-precision Assignment for DNN Inference on Constrained Edge Nodes*, arXiv, 2022.

These additional articles discuss topics which are related to the ones discussed in the thesis and can add to the interested readers, further informations in this research area:

1. **Burrello, Alessio**; Schindler, Kaspar; Benini, Luca; Rahimi, Abbas; *One-shot learning for iEEG seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing*, 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2018.

2. **Burrello, Alessio**; Cavigelli, Lukas; Schindler, Kaspar; Benini, Luca; Rahimi, Abbas; *Laelaps: An energy-efficient seizure detection algorithm from long-term human iEEG recordings without false alarms*, "2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)", 2019.

3. **Burrello, Alessio**; Marchioni, Alex; Brunelli, Davide; Benini, Luca; *Embedding principal component analysis for data reduction in structural health monitoring on low-cost iot gateways*, Proceedings of the 16th ACM International Conference on Computing Frontiers, 2019.

4. **Burrello, Alessio**; Schindler, Kaspar; Benini, Luca; Rahimi, Abbas; *Hyperdimensional computing with local binary patterns: One-shot learning of seizure onset and identification of ictogenic brain regions using short-time ieeg recordings*, IEEE Transactions on Biomedical Engineering, 2019.

5. Zanghieri, Marcello; Benatti, Simone; **Burrello, Alessio**; Kartsch, Victor; Conti, Francesco; Benini, Luca; *Robust real-time embedded EMG recognition framework using temporal convolutional networks on a multicore IoT processor*, IEEE transactions on biomedical circuits and systems, 2019.

6. Zanghieri, Marcello; Benatti, Simone; Conti, Francesco; **Burrello, Alessio**; Benini, Luca; *Temporal variability analysis in semg hand grasp recognition using temporal convolutional networks*, 2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2020.

7. **Burrello, Alessio**; Brunelli, Davide; Malavisi, Marzia; Benini, Luca; *Enhancing structural health monitoring with vehicle identification and tracking*, 2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), 2020.

8. **Burrello, Alessio**; Benatti, Simone; Schindler, Kaspar Anton; Benini, Luca; Rahimi, Abbas; *An Ensemble of Hyperdimensional Classifiers: Hardware-Friendly Short-Latency Seizure Detection with Automatic iEEG Electrode Selection*, IEEE journal of biomedical and health informatics, 2020.

9. **Burrello, Alessio**; Marchioni, Alex; Brunelli, Davide; Benatti, Simone; Mangia, Mauro; Benini, Luca; *Embedded streaming principal components analysis for*

network load reduction in structural health monitoring, IEEE Internet of Things Journal, 2020.

10. Daghero, Francesco; **Burrello, Alessio**; Pagliari, Daniele Jahier; Benini, Luca; Macii, Enrico; Poncino, Massimo; *Energy-efficient adaptive machine learning on IoT end-nodes with class-dependent confidence*, "2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)", 2020.

11. Ingolfsson, Thorir Mar; Wang, Xiaying; Hersche, Michael; **Burrello, Alessio**; Cavigelli, Lukas; Benini, Luca; *Ecg-tcn: Wearable cardiac arrhythmia detection with a temporal convolutional network*, 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2021.

12. **Burrello, Alessio**; Pagliari, Daniele Jahier; Bartolini, Andrea; Benini, Luca; Macii, Enrico; Poncino, Massimo; *Predicting hard disk failures in data centers using temporal convolutional neural networks*, European Conference on Parallel Processing, 2020.

13. Moallemi, Amirhossein; **Burrello, Alessio**; Brunelli, Davide; Benini, Luca; *Model-based vs. data-driven approaches for anomaly detection in structural health monitoring: A case study*, 2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), 2021.

14. Daghero, Francesco; Xie, Chen; Pagliari, Daniele Jahier; **Burrello, Alessio**; Castellano, Marco; Gandolfi, Luca; Calimera, Andrea; Macii, Enrico; Poncino, Massimo; *Ultra-compact binary neural networks for human activity recognition on RISC-V processors*, Proceedings of the 18th ACM International Conference on Computing Frontiers, 2021.

15. Ardebili, Mohsen Seyedkazemi; Zanghieri, Marcello; **Burrello, Alessio**; Beneventi, Francesco; Acquaviva, Andrea; Benini, Luca; Bartolini, Andrea; *Prediction of Thermal Hazards in a Real Datacenter Room Using Temporal Convolutional Networks*, "2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)", 2021.

16. Zanatta, Luca; Barchi, Francesco; **Burrello, Alessio**; Bartolini, Andrea; Brunelli, Davide; Acquaviva, Andrea; *Damage Detection in Structural Health Monitoring with Spiking Neural Networks*, 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT), 2021.

17. Barchi, Francesco; Zanatta, Luca; Parisi, Emanuele; **Burrello, Alessio**; Brunelli, Davide; Bartolini, Andrea; Acquaviva, Andrea; *Spiking Neural Network-Based*

Near-Sensor Computing for Damage Detection in Structural Health Monitoring, Future Internet, 2021.

18. Zanghieri, Marcello; Benatti, Simone; **Burrello, Alessio**; Morinigo, Victor Javier Kartsch; Meattini, Roberto; Palli, Gianluca; Melchiorri, Claudio; Benini, Luca; *sEMG-based Regression of Hand Kinematics with Temporal Convolutional Networks on a Low-Power Edge Microcontroller*, 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), 2021.

19. **Burrello, Alessio**; Zanghieri, Marcello; Sarti, Cristian; Ravaglia, Leonardo; Benatti, Simone; Benini, Luca; *Tackling time-variability in sEMG-based gesture recognition with on-device incremental learning and temporal convolutional networks*, 2021 IEEE Sensors Applications Symposium (SAS), 2021.

20. Zanghieri, Marcello; **Burrello, Alessio**; Benatti, Simone; Schindler, Kaspar; Benini, Luca; *Low-latency detection of epileptic seizures from IEEG with temporal convolutional networks on a low-power parallel MCU*, 2021 IEEE Sensors Applications Symposium (SAS), 2021.

21. Borghesi, Andrea; **Burrello, Alessio**; Bartolini, Andrea; *ExaMon-X: a Predictive Maintenance Framework for Automatic Monitoring in Industrial IoT Systems*, IEEE Internet of Things Journal, 2021.

22. Daghero, Francesco; **Burrello, Alessio**; Xie, Chen; Benini, Luca; Calimera, Andrea; Macii, Enrico; Poncino, Massimo; Pagliari, Daniele Jahier; *Adaptive Random Forests for Energy-Efficient Inference on Microcontrollers*, 2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC), 2021.

23. Palossi, Daniele; Zimmerman, Nicky; **Burrello, Alessio**; Conti, Francesco; Müller, Hanna; Gambardella, Luca Maria; Benini, Luca; Giusti, Alessandro; Guzzi, Jérôme; *Fully Onboard AI-powered Human-Drone Pose Estimation on Ultra-low Power Autonomous Flying Nano-UAVs*, IEEE Internet of Things Journal, 2021.

24. **Burrello, Alessio**; Zara, Giovanni; Benini, Luca; Brunelli, Davide; Macii, Enrico; Poncino, Massimo; Pagliari, Daniele Jahier; *Traffic Load Estimation from Structural Health Monitoring sensors using supervised learning*, Sustainable Computing: Informatics and Systems, 2022.

25. Moallemi, Amirhossein; **Burrello, Alessio**; Brunelli, Davide; Benini, Luca; *Exploring Scalable, Distributed Real-Time Anomaly Detection for Bridge Health Monitoring*, IEEE Internet of Things Journal, 2022.

26. Seyedkazemi Ardebili, Mohsen; Zanghieri, Marcello; **Burrello, Alessio**; Beneventi, Francesco; Acquaviva, Andrea; Benini, Luca; Bartolini, Andrea; *Prediction*

of Thermal Hazards in a Real Datacenter Room Using Temporal Convolutional Networks, "Proceedings of the 2021 Design, Automation & Test in Europe (DATE 2021)", 2021.

27. Daghero, Francesco; **Burrello, Alessio**; Xie, Chen; Castellano, Marco; Gandolfi, Luca; Calimera, Andrea; Macii, Enrico; Poncino, Massimo; Pagliari, Daniele Jahier; *Human Activity Recognition on Microcontrollers with Quantized and Adaptive Deep Neural Networks*, ACM Transactions on Embedded Computing Systems (TECS), 2022.

28. **Burrello, Alessio**; Sintoni, Giacomo; Brunelli, Davide; Benini, Luca; *Adversarially-Trained Tiny Autoencoders for Near-Sensor Continuous Structural Health Monitoring*, 2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS), 2022.

Finally, these articles have been done in collaboration with physicians, showing that machine learning could be strictly correlated with hospital applications and can lead to better treatments:

1. Meyer, Lucie S; Wang, Xiao; Sušnik, Eva; Burrello, Jacopo; **Burrello, Alessio**; Castellano, Isabella; Eisenhofer, Graeme; Fallo, Francesco; Kline, Gregory A; Knösel, Thomas; *Immunohistopathology and steroid profiles associated with biochemical outcomes after adrenalectomy for unilateral primary aldosteronism*, Hypertension, 2018.

2. Yang, Yuhong; Burrello, Jacopo; **Burrello, Alessio**; Eisenhofer, Graeme; Peitzsch, Mirko; Tetti, Martina; Knösel, Thomas; Beuschlein, Felix; Lenders, Jacques WM; Mulatero, Paolo; *Classification of microadenomas in patients with primary aldosteronism by steroid profiling*, The Journal of steroid biochemistry and molecular biology, 2019.

3. Burrello, Jacopo; **Burrello, Alessio**; Stowasser, Michael; Nishikawa, Tetsuo; Quinkler, Marcus; Prejbisz, Aleksander; Lenders, Jacques WM; Satoh, Fumitoshi; Mulatero, Paolo; Reincke, Martin; *The primary aldosteronism surgical outcome score for the prediction of clinical outcomes after adrenalectomy for unilateral primary aldosteronism*, Annals of surgery, 2020.

4. Vallelonga, Fabrizio; Di Stefano, Cristina; Merola, Aristide; Romagnolo, Alberto; Sobrero, Gabriele; Milazzo, Valeria; **Burrello, Alessio**; Burrello, Jacopo; Zibetti, Maurizio; Veglio, Franco; *Blood pressure circadian rhythm alterations in alpha-synucleinopathies*, Journal of Neurology, 2019.

5. Vallelonga, Fabrizio; Romagnolo, Alberto; Merola, Aristide; Sobrero, Gabriele; Di Stefano, Cristina; Milazzo, Valeria; Burrello, Jacopo; **Burrello, Alessio**; Zibetti, Maurizio; Milan, Alberto; *Detection of orthostatic hypotension with ambulatory blood pressure monitoring in Parkinson's disease*, Hypertension Research, 2019.

6. Burrello, Jacopo; **Burrello, Alessio**; Pieroni, Jacopo; Sconfienza, Elisa; Forestiero, Vittorio; Rabbia, Paola; Adolf, Christian; Reincke, Martin; Veglio, Franco; Williams, Tracy Ann; *Development and validation of prediction models for subtype diagnosis of patients with primary aldosteronism*, The Journal of Clinical Endocrinology & Metabolism, 2020.

7. Castellani, C; Burrello, J; Fedrigo, M; **Burrello, Alessio**; Bolis, S; Di Silvestre, D; Tona, F; Bottio, T; Biemmi, V; Toscano, G; *Circulating extracellular vesicles as a noninvasive biomarker of rejection in heart transplant*, The Journal of Heart and Lung Transplantation, 2020.

8. Burrello, Jacopo; Bolis, Sara; Balbi, Carolina; **Burrello, Alessio**; Provasi, Elena; Caporali, Elena; Gauthier, Lorenzo Grazioli; Peirone, Andrea; D'Ascenzo, Fabrizio; Monticone, Silvia; *An extracellular vesicle epitope profile is associated with acute myocardial infarction*, Journal of cellular and molecular medicine, 2020.

9. Vacchi, Elena; Burrello, Jacopo; Di Silvestre, Dario; **Burrello, Alessio**; Bolis, Sara; Mauri, Pierluigi; Vassalli, Giuseppe; Cereda, Carlo W; Farina, Cinthia; Barile, Lucio; *Immune profiling of plasma-derived extracellular vesicles identifies Parkinson disease*, Neurology-Neuroimmunology Neuroinflammation, 2020.

10. Burrello, Jacopo; **Burrello, Alessio**; Pieroni, Jacopo; Sconfienza, Elisa; Forestiero, Vittorio; Amongero, Martina; Rossato, Denis; Veglio, Franco; Williams, Tracy A; Monticone, Silvia; *Prediction of hyperaldosteronism subtypes when adrenal vein sampling is unilaterally successful*, European Journal of Endocrinology, 2020.

11. Burrello, Jacopo; **Burrello, Alessio**; Pieroni, Jacopo; Sconfienza, Elisa; Forestiero, Vittorio; Amongero, Martina; Rossato, Denis; Veglio, Franco; Williams, Tracy A; Monticone, Silvia; *Prediction of hyperaldosteronism subtypes when adrenal vein sampling is unilaterally successful*, European Journal of Endocrinology, 2020.

12. Burrello, Jacopo; Amongero, Martina; Buffolo, Fabrizio; Sconfienza, Elisa; Forestiero, Vittorio; **Burrello, Alessio**; Adolf, Christian; Handgriff, Laura; Reincke, Martin; Veglio, Franco; *Development of a prediction score to avoid confirmatory testing in patients with suspected primary aldosteronism*, The Journal of Clinical Endocrinology & Metabolism, 2021.

13. Vacchi, Elena; Burrello, Jacopo; **Burrello, Alessio**; Bolis, Sara; Monticone, Silvia; Barile, Lucio; Kaelin-Lang, Alain; Melli, Giorgia; *Profiling inflammatory extracellular vesicles in plasma and cerebrospinal fluid: an optimized diagnostic model for Parkinson's disease*, Biomedicines, 2021.

14. Balbi, Carolina; Burrello, Jacopo; Bolis, Sara; Lazzarini, Edoardo; Biemmi, Vanessa; Pianezzi, Enea; **Burrello, Alessio**; Caporali, Elena; Grazioli, Lorenzo Gauthier; Martinetti, Gladys; *Circulating extracellular vesicles are endowed with enhanced procoagulant activity in SARS-CoV-2 infection*, EBioMedicine, 2021.

15. Burrello, Jacopo; Bianco, Giovanni; **Burrello, Alessio**; Manno, Concetta; Maulucci, Francesco; Pileggi, Marco; Nannoni, Stefania; Michel, Patrik; Bolis, Sara; Melli, Giorgia; *Extracellular vesicle surface markers as a diagnostic tool in transient ischemic attacks*, Stroke, 2021.

16. Buffolo, Fabrizio; Burrello, Jacopo; **Burrello, Alessio**; Heinrich, Daniel; Adolf, Christian; Müller, Lisa Marie; Chen, Rusi; Forestiero, Vittorio; Sconfienza, Elisa; Tetti, Martina; *Clinical Score and Machine Learning-Based Model to Predict Diagnosis of Primary Aldosteronism in Arterial Hypertension*, Hypertension, 2021

17. Burrello, Jacopo; **Burrello, Alessio**; Vacchi, Elena; Bianco, Giovanni; Caporali, Elena; Amongero, Martina; Airale, Lorenzo; Bolis, Sara; Vassalli, Giuseppe; Cereda, Carlo W; *Supervised and unsupervised learning to define the cardiovascular risk of patients according to an extracellular vesicle molecular signature*, Translational Research, 2022.

18. Vallelonga, Fabrizio; Sobrero, G; Merola, A; Valente, M; Giudici, M; Di Stefano, C; Milazzo, V; Burrello, J; **Burrello, Alessio**; Veglio, F; *Machine learning applied to ambulatory blood pressure monitoring: a new tool to diagnose autonomic failure?*, Journal of Neurology, 2022.

19. Gallone, G; Burrello, J; **Burrello, Alessio**; Iannaccone, M; De Luca, L; Patti, G; Cerrato, E; Venuti, G; De Filippo, O; Mattesini, A; *C25 PREDICTION OF ALL–CAUSE MORTALITY FOLLOWING PERCUTANEOUS CORONARY INTERVENTION IN BIFURCATION LESIONS USING MACHINE LEARNING ALGORITHMS–THE RAIN–ML PREDICTION MODEL*, European Heart Journal Supplements, 2022.

20. Burrello, Jacopo; Caporali, Elena; Gauthier, Lorenzo Grazioli; Pianezzi, Enea; Balbi, Carolina; Rigamonti, Elia; Bolis, Sara; Lazzarini, Edoardo; Biemmi, Vanessa; **Burrello, Alessio**; *Risk stratification of patients with SARS-CoV-2 by tissue factor expression in circulating extracellular vesicles,Vascular Pharmacology*, 2022.

21. Buffolo, Fabrizio; Burrello, Jacopo; **Burrello, Alessio**; Heinrich, Daniel; Adolf, Christian; Muller, Lisa Marie; Chen, Rusi; Forestiero, Vittorio; Sconfienza, Elisa; Tetti, Martina; *DEVELOPMENT OF CLINICAL SCORING SYSTEM AND MACHINE LEARNING-BASED MODEL TO PREDICT DIAGNOSIS OF PRIMARY ALDOSTERONISM IN PATIENTS WITH HYPERTENSION*, Journal of Hypertension, 2022.

22. Burrello, Jacopo; Gallone, Guglielmo; **Burrello, Alessio**; Jahier Pagliari, Daniele; Ploumen, Eline H; Iannaccone, Mario; De Luca, Leonardo; Zocca, Paolo; Patti, Giuseppe; Cerrato, Enrico; *Prediction of All-Cause Mortality Following Percutaneous Coronary Intervention in Bifurcation Lesions Using Machine Learning Algorithms*, Journal of personalized medicine, 2022.

23. Burrello, Jacopo; Monticone, Silvia; **Burrello, Alessio**; Bolis, Sara; Cristalli, Carlotta Pia; Comai, Giorgia; Corradetti, Valeria; Grange, Cristina; Orlando, Giuseppe; Bonafè, Massimiliano; *Identification of a serum and urine extracellular vesicle signature predicting renal outcome after kidney transplant*, Nephrology Dialysis Transplantation, 2022.

Chapter 2

Background

In this chapter, I will introduce the key concepts at the basis of the work. In particular, I first describe Neural Architecture Search (NAS) and all its possible realizations, and then I depict the different deployment platforms that I target in the thesis. Finally, I will give a few details on the target applications on which these vertical flows have been tested.

2.1 Deep Neural Networks

2.1.1 Temporal Convolutional Networks

Temporal Convolutional Networks are 1-dimensional (1D) CNN variants that have recently gained significant traction for efficient time-series processing, obtaining state-of-the-art results in several tasks [35, 38, 39]. With respect to Recurrent Neural Networks (RNNs) and their successive evolutions, such as the Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU), TCNs are less affected by training-time issues, such as vanishing/exploding gradients and the large amount of training memory required by RNNs for long input sequences. Moreover, they also have computational advantages at inference time, since they share the better data locality and arithmetic intensity of standard CNNs, which makes them latency- and energy-efficient [11].

The main building blocks of TCNs are the same ones found in standard CNNs, i.e. Convolutional, Pooling and Fully Connected (FC) layers. However, the convolutional layers of a TCN are characterized by *causality* and *dilation*, two properties that make them suited for temporal inputs.

Causality enforces that the outputs of convolutions do not violate the natural cause-effect ordering of events. In practice, the outputs y_t of a TCN convolution only depend on a finite set of past inputs $x_{[t-F;t]}$, where t is a discrete index. *Dilation* is the mechanism used in TCNs for enlarging the receptive field of convolutions on the time axis, without requiring more trainable parameters and without increasing the number of operations required for inference. It is a fixed step d inserted between the input samples processed by each convolutional filter. Eq. 2.1 summarizes the 1D dilated convolution operation implemented by TCN layers:

$$y_t^m = \sum_{i=0}^{K-1} \sum_{l=0}^{C_{in}-1} x_{ts-d\,i}^l \cdot W_i^{l,m} \;, \forall m \in [0, C_{out}-1], \forall t \in [0, T-1] \tag{2.1}$$

where x and y are the input/output activations, T is the output sequence length, W the array of filter weights, C_{in}/C_{out} the number of input/output channels, K the filter size and s the stride. We also define $F = d \cdot (K-1) + 1$ the *receptive field* of the layer.

2.1.2 Attention & Transformers

I call *attention*, in general, any layer used in machine learning models to emphasize "important" parts of data, imitating the cognitive mechanism with the same name. Here we focus on *multi-head self attention (MHSA)*, introduced by [40]. Sequence-to-sequence Transformers [12] are mostly composed of stacks of MHSAs. Other kinds of Transformers also follow this line [41], possibly integrating convolutional layers. MHSA takes as input a tensor $\mathbf{X}$ of sequential data, with *sequence length* S and organized in E channels usually called *embeddings*; it produces an output of the same shape $S \times E$.

Internally, MHSA uses a set of H mutually independent parallel *heads*, all of which perform an identical set of operations divided into three steps. In the first step, each element of the input sequence $\mathbf{X}$ is projected from the space of embeddings of size E to three separate *projection spaces* each of size P, known as *queries* $\mathbf{Q}$, *keys* $\mathbf{K}$ and *values* $\mathbf{V}$ – using three trainable linear transforms:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_{\text{query}} \qquad \mathbf{K} = \mathbf{X}\mathbf{W}_{\text{key}} \qquad \mathbf{V} = \mathbf{X}\mathbf{W}_{\text{value}} \tag{2.2}$$

where $\mathbf{W}_{\text{query}}$, $\mathbf{W}_{\text{key}}$ and $\mathbf{W}_{\text{value}}$ are all matrices of size $E \times P$. In the second step, $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ are used as inputs to the core attention mechanism, *scaled dot-product attention*, which is defined as

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \doteq \mathbf{A}\mathbf{V} \doteq \underset{\text{over keys}}{\text{SoftMax}}\left(\frac{\mathbf{Q}\mathbf{K}^{\text{T}}}{\sqrt{P}}\right)\mathbf{V} \tag{2.3}$$

Table 2.1: State-of-the-art NAS (Values: ↑= large, ↗= medium, ↓= small).

	Time	Mem.	Search Space	Topology
Reinforcement Learning				
Zoph et al. [14]	↑	↓	↗	Variable*
MNASNET [17]	↑	↓	↑	Variable
NASNET [42]	↑	↓	↗	Variable
MetaQNN [43]	↑	↓	↑	Variable
Evolutionary				
Real et al. [44]	↑	↓	↑	Variable
DifferentiableNAS				
DARTS [45]	↗	↑	↓	Variable
ProxylessNAS [18]	↗	↗	↗	Variable
DmaskingNAS				
FBNetV2 [15]	↓	↓	↑	Fixed
MorphNet [16]	↓	↓	↗	Fixed
S.-Path NAS [46]	↓	↓	↗	Fixed
PIT (this work)	↓	↓	↑	Fixed

* Depth only

where $\mathbf{A}$ is an $S \times S$ matrix called the *attention matrix*. The intuition behind this is that MHSA can learn to understand which elements in the sequence are relevant with respect to one another (through $\mathbf{A}$), and can use this information to gate $\mathbf{V}$. Finally, in the third step, the output of scaled dot-product attention from all heads is projected back to the original embedding space with another linear layer, resulting in an output of size $S \times E$.

MHSAs can be expressed with three input Linear layers, the scaled dot-product attention, and one output Linear layer by properly reorganizing the dimensions at each step.

2.2 Neural Architecture Search

Exploring the network topology is the basis of the most successful applications in computer vision or time-series analysis. For instance, in recent years, several manually designed efficient and compact convolutional neural network architectures for edge devices have been proposed, including early MobileNets [47], ShuffleNets [48], EfficientNet [49], SqueezeNet [50], TEMPONet [34], etc. While these models are very efficient, obtaining them requires a lengthy and time-consuming manual tuning of hyper-parameters, which has to be repeated from scratch when considering a different target task or another deployment target. For example, MobileNets, a class of networks explicitly designed for smartphone devices, are too large to be deployed on extreme-edge devices such as Microcontrollers (MCUs) and do not fit their small on-chip memories.

However, only scaling them, for instance, by changing their number of channels (with the pre-defined width-multiplier [47]) could be detrimental for performance and accuracy.

To solve this issue, many automated or semi-automated methods to optimize neural network architectures, easing the burden of designers, have been proposed. These approaches, generally denoted as *Neural Architecture Search (NAS)* algorithms, explore a design space of different combinations of layers and/or hyper-parameter values, selecting solutions that optimize a cost metric. The latter is often a function of both the accuracy of the network and its computational cost (e.g., number of parameters or inference operations).

Table 2.1 qualitatively compares some of the most relevant works in this field in terms of search time, memory requirements during training (Mem.), search space size, and the possibility to vary the topology (number and type of layers) of the resulting NNs. For Time and Mem., smaller is better, whereas, for Search Space, larger is better.

2.2.1 Reinforcement Learning based NAS

Early NAS tools were based on Reinforcement Learning (RL) [14, 17, 42, 43] or Evolutionary Algorithms (EA) [44]. The network architectures are searched by employing successive iterations and, therefore, subsequent training. These methods sample one or more architectures from the search space at each search iteration. Sampled networks are then trained to converge to evaluate their accuracy (and possibly cost), which is used to drive the following sampling. The repeated training in each iteration is the main drawback of these tools, for which a single search requires 1000s of GPU hours, even on relatively simple tasks. Accordingly, these methods are associated with large search time in Table 2.1. Memory occupation is low and comparable to standard training since each sampled architecture can be trained separately. The search space size is virtually unlimited, and these tools can easily support variable topologies. Notable exceptions are [14], which searches over a fixed convolutional topology of a variable number of layers without varying their type and the connections between them, and [42], which constrains its search space to a set of only 13 different layers per node.

2.2.2 Differentiable NAS

To solve the search time issue of RL and EA methods, more recent *Differentiable* NAS (DNAS) approaches have proposed the so-called *supernets* [45]. Supernets are DNNs that include all possible alternative layers to be considered during the optimization. For instance, a single supernet layer might consist of multiple Convolutional layers

with different kernel sizes, operating in parallel, or even layers such as identity, depth-wise convolutions, or pooling. Therefore, the problem of choosing a specific architecture is then translated into the simpler problem of choosing a *path* in the supernet [45]. To make a parallel with RL-based NAS, sampling each different possible path in the supernet result in the collection of models explored in RL. The choice between the different paths is encoded with binary variables, which are jointly trained with the standard weights of the network using gradient-based learning.

DNAS tools enhance the normal training loss function with an additional differentiable regularization term. Typical cost metrics are the number of parameters and the number of Floating Point Operations (FLOPs) per inference [16]. In mathematical terms, DNAS tools use the following loss function:

$$\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta) \tag{2.4}$$

Where $\mathcal{L}$ is the standard loss function, W is the set of standard trainable weights (e.g., convolutional filters), θ is the set of additional NAS-specific trainable parameters that encode the different paths in the supernet, $\mathcal{R}$ is the regularization loss that measures the cost of the network and λ is a hand-tuned *regularization strength*, used to balance the two loss terms.

While DNAS algorithms are more efficient than early RL/EA-based solutions, training the entire supernet still requires huge computational resources both in terms of training time and memory occupation. This, in turn, translates into a reduction of the explored search space for practical DNASes such as [45], which have to limit the search to a few alternatives per layer to keep the memory occupation under reasonable bounds. Note that the training time increases exponentially while adding additional alternative layers. Therefore, the authors of [18] have proposed ProxylessNAS, an advanced DNAS that reduces the memory requirements, keeping in memory at most two supernet paths for each batch of inputs. In ProxylessNAS, the average weights and the additional parameters encoding supernet paths are trained and updated differently from previous DNASes. First, path parameters are frozen and based on their current value, two sub-architecture of the supernet are stochastically sampled out of all the possible alternatives. Then, the weights of the sampled architectures are updated based on the training set. Second, the weights are frozen, and the architectural parameters are trained on the validation set. This clever strategy allows ProxylessNAS to explore a significantly larger search space than other DNAS tools. On the other hand, while reducing the training time of a single epoch and strongly reducing the memory occupation, ProxylessNAS needs more epochs to explore different paths and train all of them.

2.2.3 Dmasking NAS

As a last evolution of the NAS theory, further going in the direction of efficient and lightweight NAS, researchers explored DMaskingNAS [15], fine-grain NAS [16] and Single-Path NAS [46] approaches. In these solutions, the supernet is replaced by a single, usually large, architecture with a unique path. Optimized architectures are found as modifications of this initial *seed model*, obtained tuning hyper-parameters, such as the number of channels in each layer [16]. The key mechanism that enables this tuning within a normal training loop is the use of *trainable masks*, used to prune parts of the network. Compared to DNAS algorithms, the choice of different layers can not be done, but the parameters of individual layers can be explored. DMaskingNAS tools pursue the same DNAS objective of (2.4), where θ now represents the set of trainable masks. FBNet-V2 [15], for instance, uses a set of dedicated masks, each of which encodes a different number of output channels or a different spatial resolution, and is weighted with a trainable parameter. At the end of the search, the mask coupled with the largest parameter is used to determine the final architectural setting. Similarly, MorphNet [16] exploit as masking parameters the pre-existing multiplicative terms of batch normalization layers [51]. When these parameters assume a value lower than a threshold, the corresponding channels/feature maps from the preceding Convolutional layer are eliminated.

In general, the search space of these approaches is slightly more constrained compared to supernet-based ones. For example, they do not allow the selection between alternative layers (e.g., standard convolution versus depth-wise + point-wise). On the other hand, they have two key advantages. First, they have much lower memory cost and search time while still being able to find high-quality architectures. Crucially, the search time of a DMaskingNAS is comparable to standard network training. Second, some DMaskingNASes (including my proposed works) can explore the search space at a much finer grain. For example, MorphNet [16] can easily select between 1 and 32 output channels in a Convolutional layer with a granularity of 1, starting from a 32-channel seed layer, and eliminating those corresponding to the smallest batch normalization parameters. Obtaining the same result with a standard DNAS would require a very large supernet with 32 parallel convolutional layers. Note that I can also combine the masking and supernet approaches to bypass the limitations of DMaskingNAS.

2.3 Microcontrollers: ARM & RISC-V platforms

Edge devices are becoming pervasive in everyday life. This thesis mainly focuses on the enrichment of these platforms with AI capability. Therefore, in the following

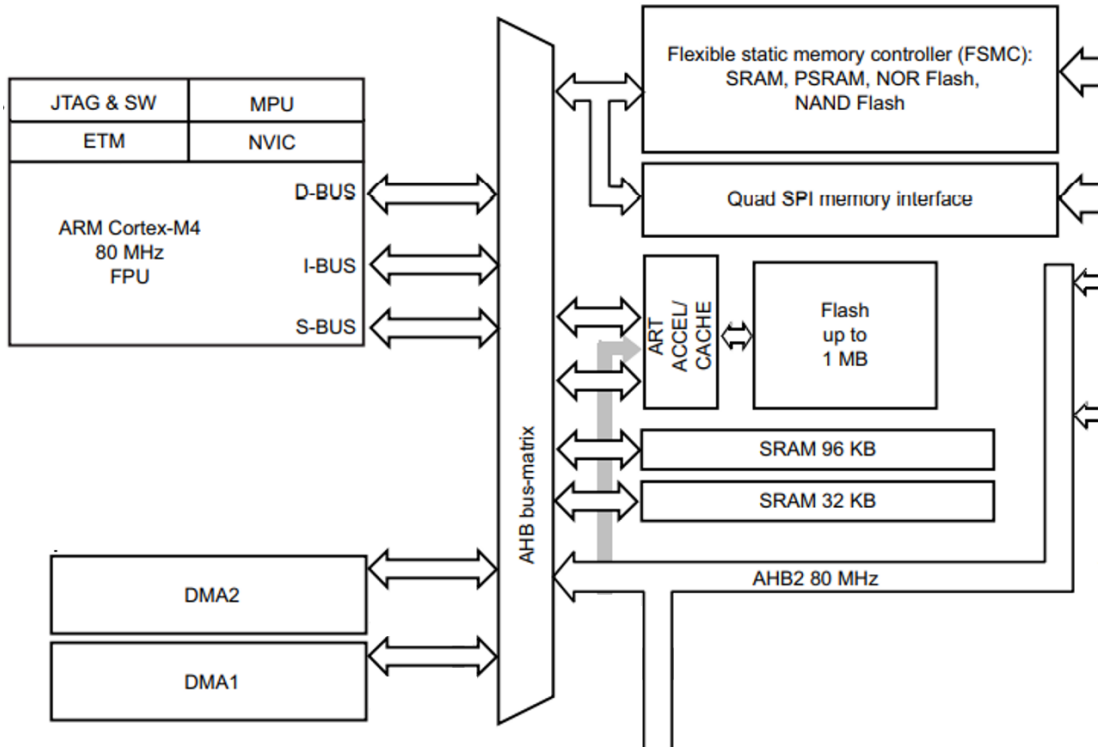

Figure 2.1: STM32L4 block diagram.

paragraphs, I will introduce different MCUs which are suitable or even designed to support AI execution on board.

2.3.1 ARM Platforms

The first MCUs presented are based on ARM cores, either single or multi processors. In the first section, I will describe single-core devices, STM32L4 and STM32H7 families. At the same time, in the second, I will introduce the STM32WB55, a dual-core MCU thought for edge applications with wireless connection capabilities.

2.3.1.1 Single-core: STM32L4 and STM32H7

The STM32H7 and the STM32L4 comprise one core ARM M7 and one core ARM M4, respectively. The two platforms are thought to be either extremely low power (STM32L4) or to maximize the computational capability for general-purpose computing in a thight power envelope (STM32H7). The STM32L4 block-diagram is depicted in Fig. 2.1. The main component is the ARM Cortex M4, a general-purpose processor equipped with FPU for floating point computation. Through the bus system, on-chip SRAM and Flash memories are connected. Additionally, Direct Memory Access peripherals are connected to support the connections of external memories or to move data from the

sensor to the core rapidly. The best operating frequency is 80 MHz, at which the STM32L4 only consumes 10 mW.

The STM32H7 presents a similar structure with some fundamental differences. First, this MCU includes two caches to improve the core performance at the expense of more energy consumption. In particular, the cache system consists of a data cache and an instruction cache to reduce the time to both fetch instructions but also to move the data to the register file. Additionally, the M7 core allows reaching a higher frequency, up to 480 MHz, at a power consumption of 234 mW.

Both these MCUs support a wide range of peripherals, including, for instance, Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C) for communication or Analog-to-digital converters (ADCs) for sensor data connection.

2.3.1.2 Dual-core: STM32WB55

I here refer to the STM32WB55RGV6 System-on-Chip (SoC), a dual-core platform from ST Microelectronics [52]. In the rest of the thesis, I refer to it simply as STM32WB. The SoC architecture includes two fully independent cores, an Arm® Cortex®-M4 core running at 64 MHz (application processor) and an Arm® Cortex®-M0+ core at 32 MHz (network processor), optimized for real-time and low-power execution. Moreover, the SoC also includes a Radio-Frequency (RF) transceiver with a radio stack compliant with Bluetooth Low Energy 5.0 (BLE) standard, including Bluetooth SIG, Mesh profile, and an HCI for proprietary custom solutions. The two cores are thought to run in parallel and manage different functionalities: the M0 core is dedicated to managing the radio-frequency and communication system. In contrast, the M4 core runs the applications. Given the similarity with the STMicroelectronics L4 families of MCUs, the STM32WB series provide similar digital and analog peripherals, suitable for applications requiring both extended battery life and high computational capability. Note that this MCU is thought for edge devices involved in networks of devices, allowing for both a high computaitonal capability on board and a variety of solutions for device connection.

Hwatch In [1], the authors introduced a new platform with a wrist-worn form factor. Its picture, together with a simplified block diagram of the system, is shown in Fig. 2.3, where only the components needed for PPG-based HR monitoring are shown. The board includes the above-mentioned STM32WB55 System-on-Chip (SoC) from ST Microelectronics [52], as main computational unit. The power supply sub-system of the board exploits a TPS63031 from Texas Instruments, a buck-boost DC/DC converter specifically designed to provide stable output voltage also with impulsive and non-reliable

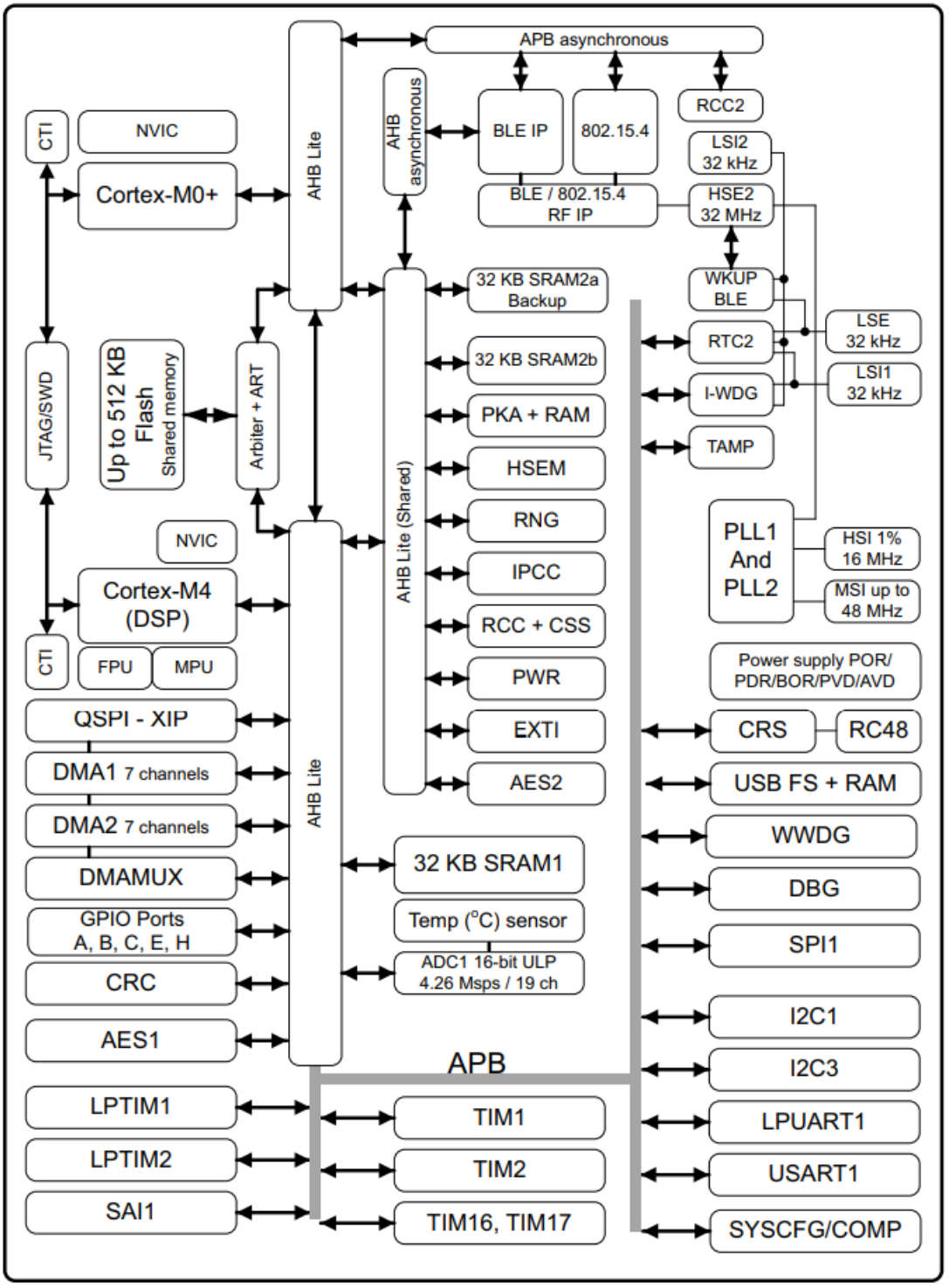

Figure 2.2: STM32WB55 block diagram.

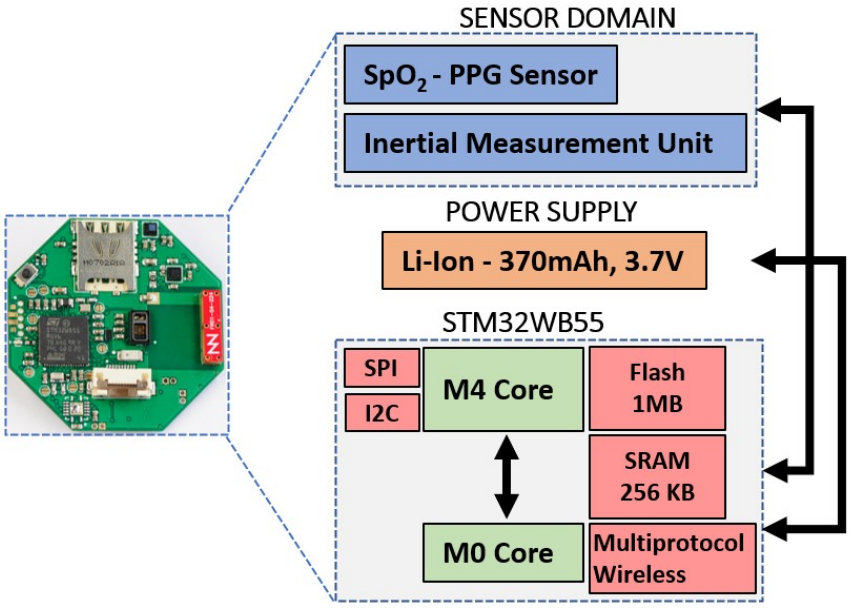

Figure 2.3: Wrist-worn form factor board presented in [1].

Table 2.2: Board components power profile.

Component	State	Current (I)	Power Consumption
Microcontroller			
STM32	Active	7.59 mA	25 mW
STM32	Idle	4.15 mA	13.7 mW
STM32	Stop	2.45 μA	8.1 μW
STM32	BLE*	30 μA	99 μW
STM32	BLE$^{\bowtie}$	2.1 mA	6.9 mW
Sensors			
MAX30101	Active	1100 μA	5.5 mW
MAX30101	Shutdown	0.7 μA	3.5 μW
LSM6DS	Active	9 μA	30 μW
LSM6DS	Shutdown	3 μA	10 μW

* STM32 BLE current advertising (0 dBm; 1 s; 31 B).
$^{\bowtie}$ STM32 BLE connected master (200 B; 100 ms)

power sources, such as energy harvesters and solar panels. The converter reaches 90% efficiency during sensor acquisition and processing modes. The TPS63031 uses a Li-Ion 370 mAh battery as the primary source of power. The other two relevant components of the system of [1] that are used in the bio applications shown in Chapter 5 are two sensors for PPG and acceleration, the MAX30101 [53], and the LSM6DSM [54]. The

former is a low-power pulse oximeter and PPG module, which is thought for extremely low-power signal aquisition from Maxim instruments. The latter is a 6-axes Inertial Measurement Unit (IMU) from STMicroelectronics. Besides its low level of noise, this sensor also includes an additional computational unit, which can be used to build small AI models (random forests of fewer than 8 trees) which use acceleration data as inputs. Additionally, different features from the acceleration can be computed directly on board, unburdening the main computational unit from this task and giving the possibility to trigger interrupts as a result of these AI on-sensor computations. Both the sensors are connected with the MCU, using respectively I2C and SPI digital busses of the STM32WB55. The hardware power consumption of the different components in all the respective working states, measured through a measurement unit Keysight B2900A, is reported in Table 2.2. Compared to the rest of the System-on-Chip, the PPG sensor requires a dedicated 5 V power supply to power up the internal LEDs. This supply voltage is generated using a step-up converter, with an efficiency of 80%.

2.3.2 RISC-V Platforms: PULP & GAP8

Compared to the previously introduced MCUs, in this paragraph, I will talk of two platforms that are based on a different instruction set architecture (ISA), the RISC-V one. Noteworthy, compared to ARM ISA, the RISC-V is open source; additionally, many extensions are possible and are currently public, which allows for specialization in tasks such as digital signal processing. On top of this ISA, a new architectural paradigm has been introduced in the direction of general-purpose MCUs with some specialized hardware parts to accelerate modern applications (e.g., deep learning). Some examples are specialized co-processors (accelerators) and hierarchical memories designed to exploit the pervasive data regularity. Parallel Ultra-Low Power computing is one of these paradigms, which leverages near-threshold computing to achieve high energy efficiency, coupled with parallelism to improve the performance degradation at low-voltage [55]. The PULP paradigm builds upon RISC-V ISA optimizations for DSP, and DNN computing, heterogeneous parallel acceleration, architecturally different compute units dedicated to unrelated tasks, and explicitly managed memory management. Some examples of ISA extensions include SIMD MACs operations at the core of DNN computation and load/store with post-increment. For instance, adding these operations reduces the number of cycles to perform an int8 MAC (the atomic operation repeated multiple times in most of the layers at the basis of every neural network, e.g., convolutions) from 7 to just 1, with a speed-up of 7×. If I only exploit the basic ISA operations, I would need three index updates, two loads, one MAC, and one store to perform a complete

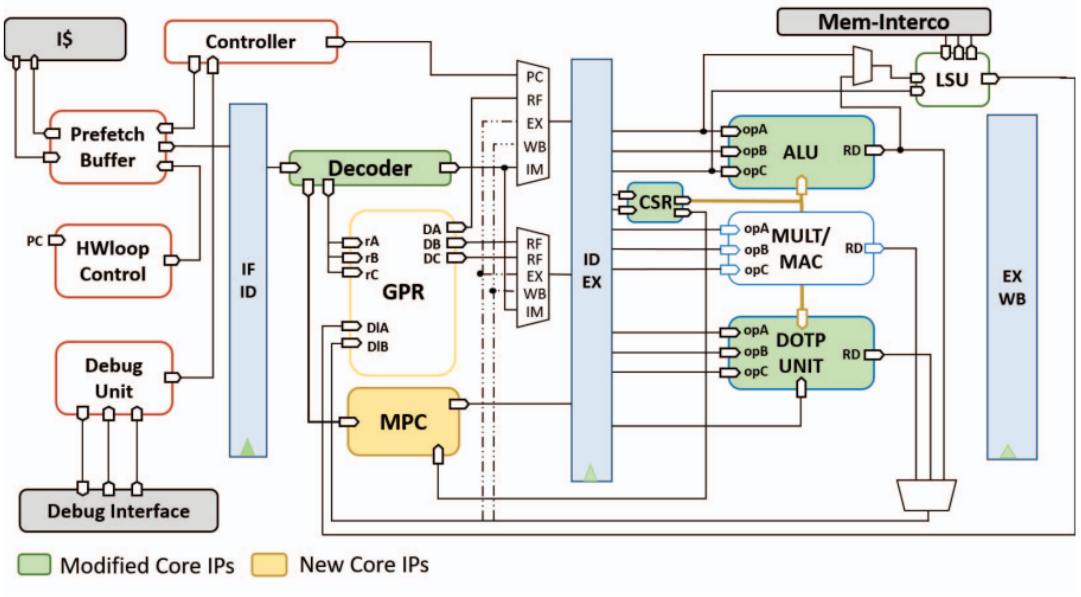

Figure 2.4: MPIC block diagram with mixed-precision dedicated hardware IPs in yellow.

MAC with operands stored in memory. On the other hand, exploiting the new operations introduced in PULP, the index updates are included in the memory operations. Furthermore, the MAC parallelizes the process on 4 int8 data (the bus and the ALU is on 32bits). Therefore, 4 cycles are used to perform 4 MACs in parallel.

Most of the embodiments of the PULP paradigm are centered around a state-of-the-art single-core microcontroller (*Fabric Controller domain*) with a standard set of peripherals. In contrast, they offload the processing-intensive tasks to a software-programmable parallel accelerator composed of N additional cores, standing in its voltage and frequency domain (*cluster domain*).

2.3.2.1 MPIC

The first RISCV core that I present is MPIC, shown in Fig. 2.4. This core could be the building block at the basis of the clusters of different PULP embodiments. Compared to classical RISC-V cores, MPIC includes optimized hardware units for the execution of MAC operations with inputs independently quantized to $p_{w/x} \in \{2, 4, 8\}$ bit. This new hardware block allows executing integer-reduced precision operations (less than 8 bits) faster than 8 bits operations, allowing the efficient deployment of quantized neural networks. The reader can find more details in [30].

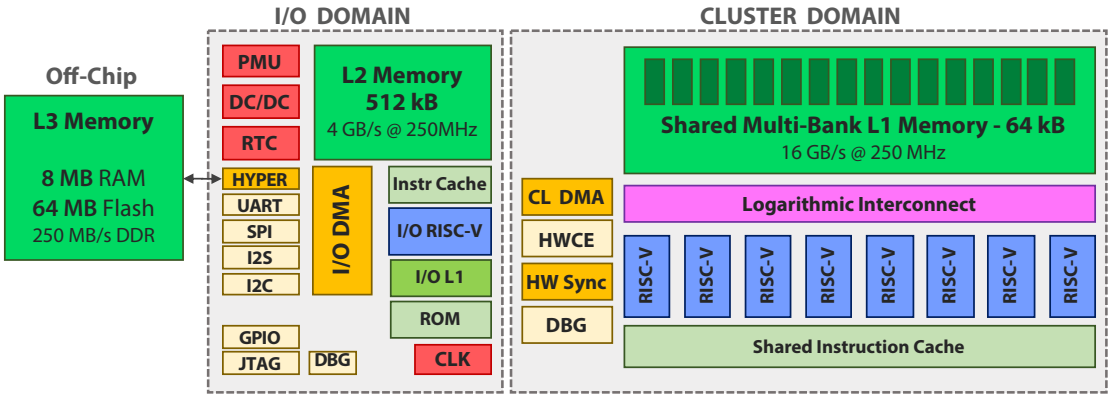

Figure 2.5: GWT GAP-8 MCU block diagram.

2.3.2.2 GAP8

GWT GAP-8 [27] (depicted in Figure 2.5) is a commercial PULP system with 9 extended RISC-V cores (one I/O + an eight-core cluster), which represents one of the most advanced embodiments of the DNN-dedicated MCU trends. The GAP-8 'cluster' comprises eight 4-stage in-order single-issue pipeline RI5CY [56] cores, implementing the RISC-V RV32IMCXpulpV2 Instruction Set Architecture (ISA). XpulpV2 is a domain-specific extension meant for efficient digital signal processing, with hardware loops, post-modified access LD/ST, and SIMD instructions down to 8-bit vector operands.

The cores of the *cluster* share the first level of memory, a 64 kB multi-banked L1 memory Tightly-Coupled Data Memory (TCDM), accessible from the cluster's cores through a high-bandwidth, single-cycle-latency logarithmic interconnect. The L1 features a 2× banking factor and a word-level interleaving scheme to reduce the probability of contention [57]. To manage data transfers between the L1 TCDM memory and a second-level 512 kB of memory (managed as a scratchpad as well) available in the SoC domain, the *cluster DMA* [58] can manage data transfers between L1 and L2 with a bandwidth up to 2 GB/s and a latency of 80 ns at the maximum frequency. On the other hand, to interface the L2 memory with the external world, and in particular, with the Cypress Semiconductor's HyperRAM/HyperFash module [59] available on the GAPuino board, GAP-8 can use an autonomous I/O sub-system called *I/O DMA* [60]. Through the HyperBus interface, the external L3 HyperRAM and/or HyperFlash memory can be connected to the system, enabling a further 64 MB of storage for read-only data on Flash and 8-16 MB for volatile data on DRAM, with a bandwidth up to 200 MB/s.

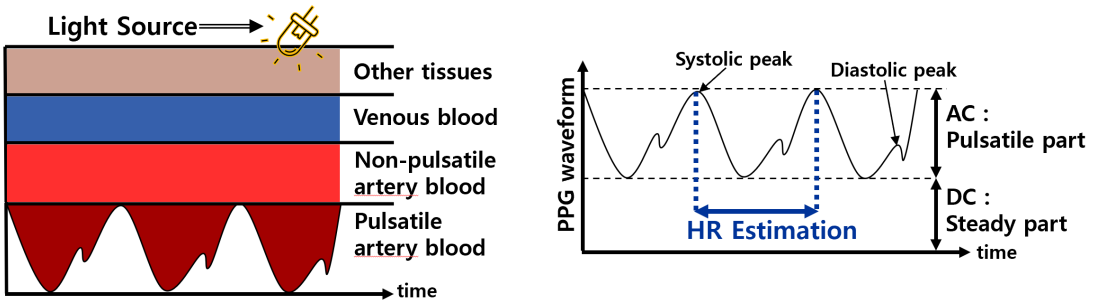

Figure 2.6: Synthtic generated and ideal PPG signal.

2.4 Biosignals: Sensors and processing

I here introduce the two applications on which the tools described in the following sections are applied. The first is the photoplethysmography (PPG) based heart rate (HR) monitoring. Thanks to the flowing of the blood, the beats of the core can also be identified in the remote part of the arterio-venous system, allowing for minimal invasive settings in the monitoring of the heart condition. The second application is surface electromyographic signal (sEMG) based gesture recognition. The scope is to identify the gestures of the arms from sEMG signals, possibly allowing patients to move robotic hands or prostheses for amputees.

2.4.1 Photoplethysmography and PPG-based HR

Photoplethysmography (PPG) is a technique that measures the light absorption variations of blood vessels during cardiac activity [61]. A PPG sensor consists of one or more Light-Emitting Diodes (LEDs) that continuously emit light to the skin and a photodetector (i.e., a photodiode) that measures variations of light intensity caused by blood flow. The blood flow's periodicity can be associated with the heart rate. More specifically, the larger the blood volume variation, the greater the attenuation of the light emitted by the LED, resulting in a lower current output on the photodiode. Therefore, a heartbeat can be associated with each peak in the PPG signal. Figure 2.6 shows how the PPG works.

Indeed, many studies demonstrated that the second derivative of the PPG signal contains essential information for heart rate monitoring [62]. Therefore, the simplicity of wearing a PPG sensor and the low cost contribute to its increasing popularity as an alternative to ECG for HR monitoring [63]. However, one of the significant challenges in employing PPG signals to recognize heartbeats and estimate HR is their considerable dependency on the subject's movements, which negatively affect the measurement quality during daily activities, as first shown in [64]. In particular, the arm movements cause

the so-called Motion Artifacts (MA), which alter the readings of the sensor and strongly impact the performance of the HR estimation. Therefore, they demand ad-hoc algorithms for their remotion/reduction. The standard approach to cope with this problem is to leverage additional inertial measurements (mainly acceleration) that correlate with the PPG signal and can be used to discriminate between real HR peaks and MA-caused ones. For additional details on using PPG for HR estimation, please refer to [65].

2.4.2 Surface Electromyographic Signal

EMG signals [66] used for gesture recognition originate from the electrical activity that occurs during a muscular contraction, ranging from 10 uV to 1 mV. The bandwidth is $\sim$ 2 kHz for standard applications, even though it is possible to acquire EMG data up to $\sim$ 10 kHz in Motor Unit Action Potential Analysis. Conductive plates placed on the skin surface (i.e., electrodes) acquire EMG activity. However, a significant issue of signal acquisition is related to the skin-electrode interface, which is prone to the high variability and can degrade signal quality given the arm's movements. Also, electrode repositioning and user adaptation [34] cause degradation in the signal since they can change from one acquisition session to another or from a day to the successive one. Finally, also motion artifacts and floating ground noise represent causes of signal degradation and variability.

Chapter 3

Neural Architecture Search for Efficient Deployment on MCUs

3.1 Related Works

In literature, different approaches to NAS have been proposed. As previously described, first NASes were based on reinforcement learning, e.g., *NAS-RL* [67] and *MetaQNN* [14], or evolutionary algorithms [44].

DARTS [68] has been the first differentiable NAS algorithm. DARTS is modeled as the optimization of a *supernet*, where many layers are connected through edges, each associated with a trainable weight. Some of them are pruned at the end and only a single path is selected. Successively, many NASes have been designed to explore the huge space of DNNs. In particular, I focus here on two Dmasking NASes which have inspired the work of this thesis.

MorphNet [16] is one of the first Dmasking NAS from 2018 that searches the number of output channels of convolutional layer. In particular, it exploits batch normalization parameters as masks for the weights. These parameters are then thresholded to yield a simplified architecture, with an approach similar to weight pruning. The result is a network with same topology but a reduced number of channels. MorphNet uses specific regularizers to optimize specific metrics such as size, FLOP and latency. Further details can be found in the original MorphNet paper [16].

FBNetV2 [15] also searches for the optimal number of output channels of convolutional layer, but with a method based on exclusive masks. Noteworthy, this approach can be easily extended to other hyper-parameters such as the filter size, increasing the possible search space. For example, if a certain convolutional layer presents N output

channels, then it is possible to build N different masks with different numbers (l) of leading *1s* and *N-l* trailing *0s*, that correspond to variants of the layer with l output channels. Masks are then multiplied with a *Gumbel-Softmax* weight [69], summed together and multiplied with the output of convolution. In this way, a single mask is chosen at the end of the search and applied to the output, obtaining a layer with the selected number of channels.

In the following sections, I will first describe two Dmasking NAS on which I worked to optimize TCNs and the quantization of DNNs. Then, I will briefly explain a technique to combine different optimization directions inside the same NAS algorithm, allowing embedded designers to obtain good deployment points in a lower amount of time.

3.2 Lightweight Neural Architecture Search for Temporal Convolutional Networks at the Edge

In this section, I introduce *Pruning in Time (PIT)*, a new lightweight DmaskingNAS that targets networks that process time series. PIT explores the architectures of convolutional and fully-connected (FC) layers, the two most compute- and memory-expensive operations present in TCNs. For each convolutional layer, PIT jointly explores the *number of channels* (C_{out}), the *receptive field* (F), and the *dilation* (d). Moreover, by tuning both F and d, it also indirectly affects the *filter size* K. Similarly, PIT can also optimize the number of output neurons of FC layers[1].

First, in Section 3.2.1, I give an overview of the search space explored by the tool and of its general working principle. Then, I illustrate the mechanisms used to generate differentiable masks for each considered hyper-parameter in Sections 3.2.1.1-3.2.1.4. Finally, the two cost regularizers used to augment the classic loss are described in Section 3.2.2 and 3.2.3, respectively. Table 3.1 sums up the main mathematical symbols used in this section.

3.2.1 Search Space

As shown in Figure 3.1, PIT's search space encompasses all sub-architectures derived from a seed TCN, with any combinations of the three abovementioned parameters per layer. In particular, PIT can *reduce* C_{out} or F, and to *increase* d compared to the seed, all of which have the effect of reducing the complexity and memory occupation of the layer.

To achieve this objective, each convolutional/FC layer of the seed is modified to become a function $L_n(W^{(n)};\theta^{(n)})$ of its original weights tensor $W^{(n)}$ and of a new set of architectural parameters $\theta^{(n)}$. For a TCN with N layers, the search space of PIT is therefore defined by:

$$\mathcal{S} = \{L_n(W^{(n)};\theta^{(n)})\}_{n=0}^{N-1} \tag{3.1}$$

During the search, the elements of $\theta^{(n)}$ are properly combined to form a *binary mask* $\Theta^{(n)}$. This mask prunes a portion of the layers' weights by multiplying them by 0s. In practice, an architecture $\hat{\mathcal{S}}$ is sampled from $\mathcal{S}$ in each search iteration, by performing the Hadamard product between $W^{(n)}$ and $\Theta^{(n)}$, i.e., $\hat{\mathcal{S}} = \{L_n(W^{(n)} \odot \Theta^{(n)})\}_{n=0}^{N-1}$. This eliminates the portions of $W^{(n)}$ that correspond to 0-valued mask elements: the seed

[1] This can be seen as a corner case of the C_{out} optimization since FC layers are just a particular case of 1D convolutions with $F = K = d = 1$ and C_{out} equal to the number of output neurons. Accordingly, the rest of this section describes PIT's functionality for convolutions.

Table 3.1: List of symbols used in the description of the NAS algorithm.

Symbol	Description
x	Input activations of a convolutional layer
y	Output activations of a convolutional layer
T	Output sequence length of a convolutional layer
C_{in}, C_{out}	Number of input/output channels of a conv. layer
W	Convolutional filter weights
K	Convolution filter size
s	Convolution stride
d	Convolution dilation
F	Convolution receptive field
$\mathcal{L}$	Task-specific loss function
$\mathcal{R}$	Regularization loss function
λ	Regularization strength
$\mathcal{S}$, $\hat{\mathcal{S}}$	Search space and sampled architecture
L_n	Generic convolutional/FC layer
N	Number of convolutional/FC layers
θ, Θ	Generic NAS architectural parameters and corresponding binary mask
α, Θ_A	NAS architectural parameters to optimize C_{out} and corresponding binary mask
β, Θ_B	NAS architectural parameters for F, and corresponding binary mask
γ, Γ, Θ_Γ	NAS architectural parameters for d, intermediate binary mask elements and final binary mask
C_β, C_γ	Transformation matrices to generate Θ_B and Θ_Γ from β and γ.
$k(i)$	Index mapping function used to generate Θ_Γ from Γ

layer pruned produces the same output that I would obtain by having a smaller layer with a lower number of channels or a smaller receptive field. The way in which $\Theta^{(n)}$ is generated from $\theta^{(n)}$ to produce this effect is explained in Sections 3.2.1.1-3.2.1.3.

Noteworthy, having *binary* masks is required to either eliminate slices of $W^{(n)}$ (with value 0) or keep them untouched (with value 1) when sampling an architecture with the Hadamard product. In practice, this corresponds to sampling only *feasible* architectures (with integer C_{out}, F and d). To this end, $\Theta^{(n)}$ is binarized in the forward-pass of search/training, applying an Heaviside step function with a fixed threshold $th = 0.5$.

At the same time, the $\theta^{(n)} \rightarrow \Theta^{(n)}$ transformation has to be differentiable to embed the search into the standard gradient-based training of the network, learning contextually the weights $W^{(n)}$ and the topological parameters $\theta^{(n)}$. To cope with the Heaviside function derivation issues, i.e., derivative equal to 0 almost everywhere and not existent in δ, I follow the approach proposed in BinaryConnect [70], which substitutes the backward pass with the Straight-Through Estimator (STE), and therefore the derivative with the identity.

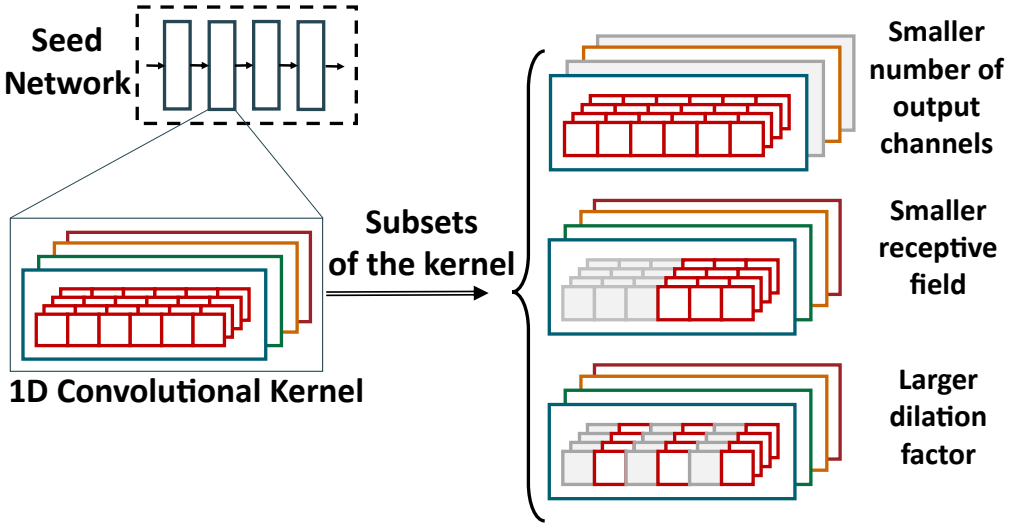

Figure 3.1: Search space of PIT.

For notation simplicity, $\theta^{(n)}$ parameters are divided into three groups: $\alpha^{(n)}$ are the parameters associated with the output channels, $\beta^{(n)}$ tunes the receptive field, and $\gamma^{(n)}$ affects the dilation factor. In PIT, each of these three parameters is used to generate an *independent* binary mask, which can be combined with the other two. The final mask is multiplied with weights to prune the correct portion. Also, having independent masks for C_{out}, F and d, gives PIT the flexibility to work into two different modalities: in full optimization, PIT optimizes the entire network, while in independent mode, the user can define a single (or more) hyper-parameters to optimize.

During full optimization, PIT explores:

$$|\mathcal{S}| \approx \prod_{n=0}^{N-1} (C^{(n)}_{out,seed} \cdot F^{(n)}_{seed} \cdot \lceil \log_2(F^{(n)}_{seed}) \rceil) \tag{3.2}$$

different solutions, where $C_{out,seed}$ and F_{seed} in (3.2) are those of the seed layers. The logarithmic term in (3.2) comes from the fact that I only consider power-of-2 dilation factors, as detailed in Section 3.2.1.3 since they are the most used in state-of-the-art networks. For a relatively small seed with $N = 8$, $F^{(n)}_{seed} = 17$, and $C^{(n)}_{out,seed} = 128\ \forall n$, this corresponds to evaluating $\approx 10^{32}$ architectures in a single training.

3.2.1.1 Channels Search

To explore the number of channels in each convolutional layer, I take inspiration from [16]. In that work, the parameters of batch normalization (BN) layers [51] were transformed into binary masks to prune entire output channels and explore the space of all sub-layers with $C_{out} < C_{out,seed}$. Indeed, when a BN layer follows a convolutional

one, each output channel is obtained as:

$$\tilde{y}_t^m = \gamma^m \cdot y_t^m \tag{3.3}$$

where y_t^m is the output of convolution and γ^m is the multiplicative factor of BN. When the latter is binarized to 0, the entire m-th output channel is effectively pruned. However, requiring the presence of a BN layer after each convolution, although common in modern 2D-CNNs, still limits the applicability of the approach of [16]. Therefore, in PIT, the channel search is decoupled from BN, adding a dedicated α parameter to each output channel to zero-out entire filters from the W tensor of convolutional layers. PIT treats each output channel independently. So, it uses an α array of length $C_{out,seed}$, The layer function is modified to:

$$\tilde{y}_t^m = \sum_{i=0}^{K-1} \sum_{l=0}^{C_{in}-1} x_{ts-di}^l \cdot (\Theta_{A,m} \cdot W_i^{l,m}) \tag{3.4}$$

In practice, each α parameter is multiplied with all the weights of the *same convolutional filter*, i.e., with an entire slice of the weights tensor over the output channels axis. Each filter multiplied with a 0-mask is effectively pruned from the network, reducing the number of channels by 1. Figure 3.2 depicts the application of Θ_A parameters to a simple layer with $C_{out,seed} = 4$.

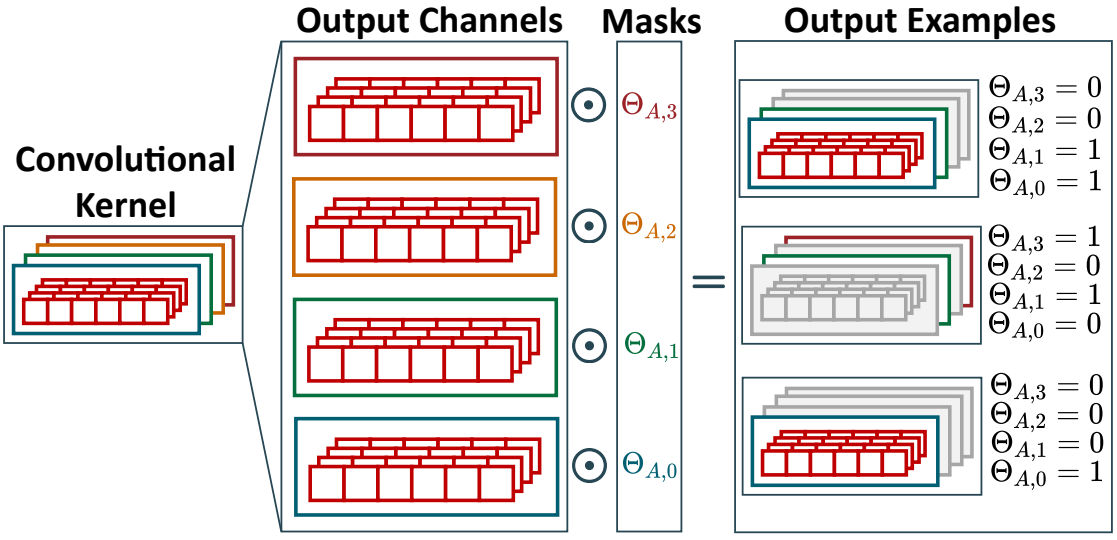

Figure 3.2: Channels search example. Each $\Theta_{A,m} = 0$ zeroes-out the m-th convolutional filter, i.e., a slice of size $K \times C_{in}$ of the weights tensor W.

Noteworthy, besides reducing the number of channels, PIT can also *eliminate* entire layers from the network if the latter includes skip connections. In particular, if all the $\Theta_{A,m}$ of a convolutional layer are zeroed-out, then the inputs only flow through the skip connection, effectively reducing the number of layers in the network by one. If skip

connections are not present, at least one output channel is always kept active to avoid breaking the network connectivity.

3.2.1.2 Receptive Field Search

The second hyperparameter that is explored in PIT is the receptive field F, i.e., the range of input time-steps involved in a convolution. In standard convolutions, the receptive field equals the filter size ($F = K$). However, in the modern TCNs, where the d parameter has been introduced, the general relation becomes: $F = (K-1) \cdot d + 1$. Therefore, since K depends on both F and d, PIT also indirectly optimizes the filter dimension K.

The receptive field is explored using the array of parameters β, with $len(\beta) = \left\lfloor \frac{F}{gr} \right\rfloor$ where F is the seed receptive field and $gr \geq 1$ is the granularity of the exploration. A granularity greater than 1 for the receptive field allows, for instance, to force the network only to use even/odd receptive fields. Differently from the output channels, however, the β needs to be further combined to define the corresponding binary differentiable masks. The reason is that, to "simulate" the effect of a smaller receptive field through masking, it is not sufficient to mask *any* set of time-slices in the weights tensor: on the other hand, this should be i) continuous and, in the case of "causal" convolutions, the slices should be eliminated from the "oldest" part of the weights, i.e., those that are multiplied with input time-steps that are farthest in the past. The following equation derives elements of the binary masks Θ_B from β to reproduce this behaviour:

$$\Theta_{B,i} = \mathcal{H}\left(\sum_{j=1}^{F_{seed}-i} |\beta_{F_{seed}-j}|\right) \tag{3.5}$$

Each $\Theta_{B,i}$ is then multiplied with a time-slice of the W tensor during the forward pass, as shown in Figure 3.3. Therefore, when searching for the receptive field, the equation becomes:

$$y_t^m = \sum_{i=0}^{K-1} \sum_{l=0}^{C_{in}-1} x_{ts-di}^l \cdot (\Theta_{B,di} \odot W_i^{l,m}) \tag{3.6}$$

Thanks to the construction of (3.5), if $i > j$, then $\Theta_{B,i} \leq \Theta_{B,j}$. This ensures that the first weight slices to be pruned are always the leftmost ones, as shown in the example on the right of Figure 3.3. Notably, β_0 is always kept constant and equal to 1, to ensure that even after binarization, at least one time-step will still be convolved with weights.

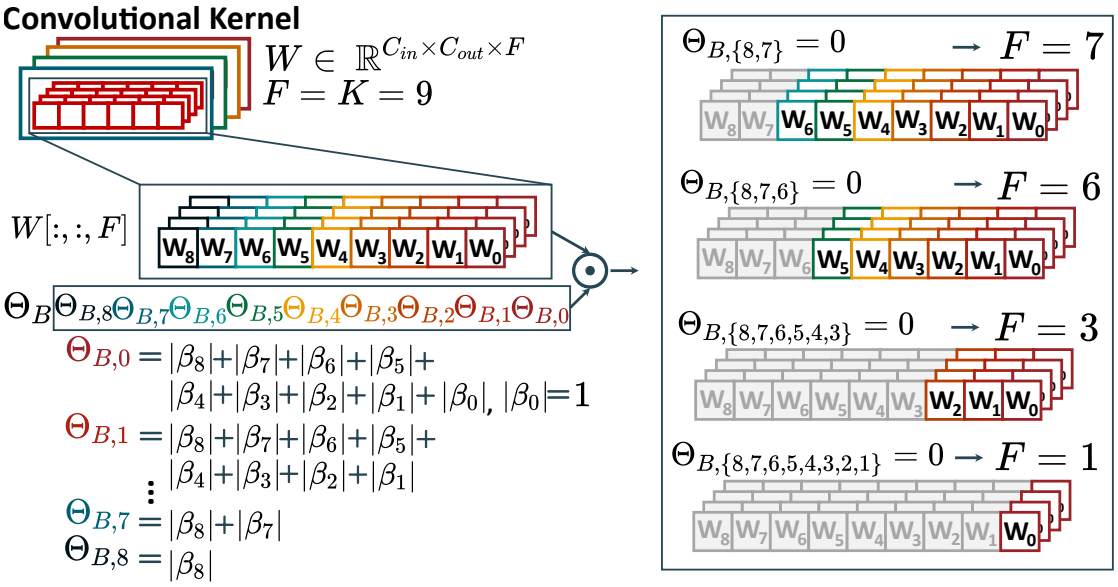

Figure 3.3: Receptive field search example. Each $\Theta_{B,i} = 0$ eliminates the contribution of 1 input time-step from the convolution output, by zeroing out a time-slice of size $C_{out} \times C_{in}$ of the weights tensor W.

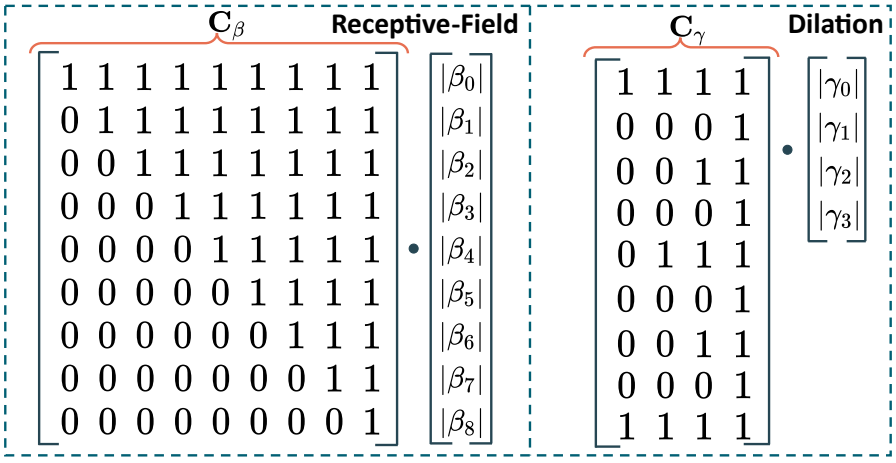

Figure 3.4: Example of conversion between trainable architectural parameters β and γ and corresponding binary masks Θ_B and Θ_Γ, for a layer with $F_{seed} = 9$.

In practice, for efficiency reasons, binary masks are generated using the matrix transformation:

$$\Theta_B = \mathcal{H}\left(C_\beta \cdot |\beta|\right) \tag{3.7}$$

Where C_β is a constant upper triangular matrix of 1s generated once at the beginning of a search, as shown on the left of Figure 3.4.

3.2.1.3 Dilation Search

Lastly, PIT also explores the dilation factor d. Similarly to the receptive field, searching for dilation imposes some constraints on the portions of the weights tensor that

this NAS should prune. In particular, only *regular* dilation factors should be generated, i.e., the time-steps gaps between consecutive convolution inputs are all equal for a given layer. For example, I do not want to obtain a layer that takes as input time-steps t, $t-2$, $t-3$, and $t-8$, corresponding to gaps of 0, 1, and 5 time-steps respectively. Note that this is necessary since most inference libraries would not support layers with "random" time gaps, in particular those for edge devices [71, 72], which only implement regular gaps, since they enable repetitive memory accesses therefore minimizing the additional arithmetic operations needed to compute memory indexing.

Based on these observations, I follow an approach similar to the one described in Section 3.2.1.2. Starting from an array of trainable parameters γ, they are then combined to compose differentiable binary masks. This method only supports power-of-2 dilation factors since i) they are the most common and ii) they simplify the generation of the masks. Thus, I obtain: $len(\gamma) = \lceil \log_2(F_{seed}) \rceil$.

To obtain the elements of Θ_Γ, I pass through an intermediate array Γ, generated similarly to (3.5):

$$\Gamma_i = \mathcal{H}\left(\sum_{j=1}^{len(\gamma)-i} |\gamma_{len(\gamma)-j}|\right) \tag{3.8}$$

Then, the mask is obtained by further reorganizing the Γ_i values into the vector Θ_Γ, of length F_{seed}, as follows:

$$\Theta_{\Gamma,i} = \Gamma_{k(i)}, \text{ with } k(i) = \sum_{p=1}^{len(\gamma)} 1 - \delta(i \bmod 2^p, 0) \tag{3.9}$$

and where $\delta()$ is Kronecker's Delta function. This reorganization ensures that the Γ element with the largest index ($\Gamma_{len(\gamma)-1}$) ends up in all positions corresponding to time-steps that a layer would skip with $d = 2$. Similarly, the element with the second largest index ends up in positions that are skipped when using $d = 4$, and so on. This, combined with the fact that, by construction of (3.8), it holds that $\Gamma_i \leq \Gamma_j$ for $i > j$, ensures that the dilation is *progressively increased*. In this way, each additional binarized Γ_i causes a proportional increase in the dilation, which is doubled.

The obtained Θ_Γ vector is multiplied with the W tensor, exactly as the previous ones. Again, γ_0 is set to 1 to ensure that I never prune the entire convolution, and I always keep the first and last weight of the receptive field, therefore minimizing the number of weights. Figure 3.5 shows an example of how the tensor is generated and its effect on the dilation. In practice, similarly to the receptive field mask, also Θ_Γ is obtained from γ with a simple matrix multiplication:

$$\Theta_\Gamma = \mathcal{H}(C_\gamma \cdot |\gamma|) \tag{3.10}$$

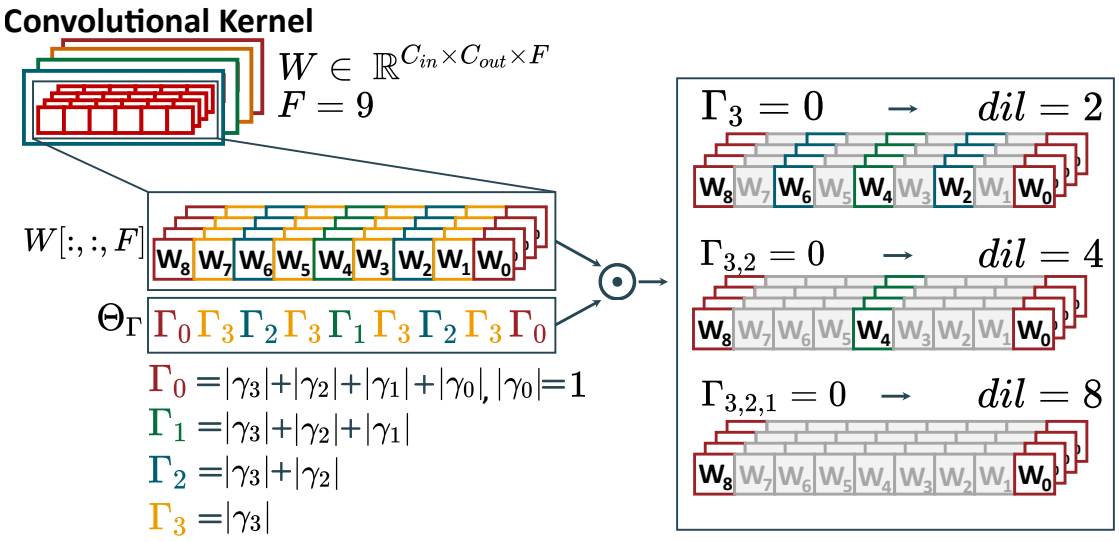

Figure 3.5: Dilation search example. Each $\Gamma_i = 0$ increases d by a factor 2.

where C_γ is a constant matrix of 0s and 1s that can be generated procedurally based on the value of F_{seed}. An example of C_γ is shown on the right of Figure 3.4.

3.2.1.4 Joint Search

To jointly optimize all three hyper-parameters mentioned above, I simultaneously apply all three Θ masks to the weight tensor of a layer. Therefore, the equivalent of the equation for a seed convolutional layer during a joint search is:

$$y_t^m = \sum_{i=0}^{K-1} \sum_{l=0}^{C_{in}-1} x_{ts-i}^l \cdot (\Theta_{B,i} \odot \Theta_{\Gamma,i} \odot (\Theta_{A,m} \cdot W_i^{l,m})) \tag{3.11}$$

In the experiments section, I will show that performing such a joint search yields superior results compared to optimizing the three hyper-parameters sequentially since PIT can take into account the complex interactions among them (especially among F and d).

3.2.2 Regularization

PIT searches for accurate yet low-complexity architectures by combining the task-specific loss function $\mathcal{L}$ with a regularization term $\mathcal{R}$ as introduced in Sec. 2.2. The additional differentiable term proportional to the dimension/operations of the network encodes a prior in the loss landscape that directs the optimization towards low-cost solutions. The two cost metrics considered in this work are the model's number of parameters (or size) and the number of operations (OPs) for an inference. The corresponding two regularizers $\mathcal{R}_{size}$ and $\mathcal{R}_{ops}$ are differentiable functions of the *pre-binarization* masks

$\tilde{\Theta}_A$, $\tilde{\Theta}_B$ and $\tilde{\Theta}_\Gamma$. The latter, in turn, depends on the trainable architectural parameters α, β, and γ. I use pre-binarization masks as in [16], because this yields a smoother loss landscape, improving convergence. Indeed, using floating point continuous values do not lead to any abrupt change in the loss, giving more stability to the training process.

3.2.2.1 Size Regularizer

The Size regularizer $\mathcal{R}_{size}$ estimates, during each forward-pass, the *effective* number of parameters of the network, based on the values of the differentiable binary masks. The number of parameters of a 1D convolutional layer, i.e., the size of weight tensor W, is equal to $C_{in} \times C_{out} \times K$. Accordingly, the size regularizer for a TCN with N convolutional (or FC) layers is defined as:

$$\mathcal{R}_{size} = \sum_{n=0}^{\text{N-1}} (\mathcal{R}_{size}^{(n)}) = \sum_{n=0}^{\text{N-1}} C_{out,eff}^{(n-1)} \cdot C_{out,eff}^{(n)} \cdot K_{eff}^{(n)} \tag{3.12}$$

where:

$$C_{out,eff}^{(n)} = \sum_{i=0}^{C_{out,seed}^{(n)}-1} \tilde{\Theta}_{A,i}^{(n)} \tag{3.13}$$

is the effective number of channels in the n-th layer, and:

$$K_{eff}^{(n)} = \sum_{i=0}^{F_{seed}^{(n)}-1} \frac{\tilde{\Theta}_{B,i}^{(n)}}{F_{seed} - i} \cdot \frac{\tilde{\Theta}_{\Gamma,i}^{(n)}}{len(\gamma) - k(i)} \tag{3.14}$$

is the effective kernel size, which depends both on the total receptive field and on the dilation. For the 1st layer of the network, $C_{out,eff}^{(n-1)}$ is constant and equal to the number of channels of the input signal.

The definitions of (3.13) and (3.14) are continuous relaxations of the number of *active* (non-pruned) channels and time-slices of $W^{(n)}$, respectively. By minimizing R_{size}, PIT is encouraged to reduce the $\tilde{\Theta}$ values, bringing them below the binarization threshold. Depending on the regularization strength λ, PIT balances the corresponding reduction in cost with the accuracy drop caused by the reduced number of parameters included in the different layers.

The denominators in (3.14) are needed to make sure that, when β and γ are equal to 1 (i.e., the initialization value, see Section 3.2.3), $K_{eff}^{(n)}$ corresponds to the real filter size of the seed. In fact, each $\tilde{\Theta}_{B/\Gamma}$ is obtained as sum of a different number of γ (or β) elements. As a result, without normalization, the estimated cost would be higher than the real filter size. For instance, in a layer with $F_{seed} = 5$ and with all β/γ initialized

Algorithm 1

for $i \leftarrow 1, \ldots, \text{Steps}_{\text{wu}}$ **do** #warmup loop
    Update W based on $\nabla_W \mathcal{L}(W)$
end for
while not converged **do** #search loop
    Update W and θ based on $\nabla_{W,\theta}(\mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta))$
end while
for $i \leftarrow 1, \ldots, \text{Steps}_{\text{ft}}$ **do** #fine-tuning loop
    Update W based on $\nabla_W \mathcal{L}(W)$
end for

at 1, without the denominators, I would have $K_{eff} = 33$, which is clearly incorrect. Conversely, with the denominators, I have $K_{eff} = 5 = F_{seed}$, which is correct, since the initialization of $\gamma = 1$ implicitly imposes $d = 1$.

3.2.2.2 OPs Regularizer

The second proposed regularizer R_{ops} estimates the number of operations required to perform inference. Since the number of OPs of a 1D convolutional layer is $T \times C_{in} \times C_{out} \times K$, the regularizer expression is simply:

$$\mathcal{R}_{ops} = \sum_{n=1}^{\text{N}} (\mathcal{R}_{size}^{(n)} \cdot T^{(n)}) \tag{3.15}$$

In practice, when targeting the reduction of the total OPs for inference, the only difference in the regularizer is that the output sequence length weights the cost of each layer. Note that this is not a constant over every layer since T can change over layers such as pooling, strided convolution, etc..

3.2.3 Training Procedure

Algorithm 1 summarizes the three main phases of a PIT architecture search. The first phase consists of Steps_{wu} iterations of warm-up. At this algorithm stage, all θ parameters (i.e., α, β, and γ) are initialized to 1 and frozen. Accordingly, all elements of the binary masks Θ are also binarized to 1. Therefore, warm-up coincides with a normal training of the seed network, where the only objective is minimizing the task loss function $\mathcal{L}$. The number of warm-up iterations is a user-defined parameter. In all experiments, I perform the warm-up to the convergence of the floating point network.

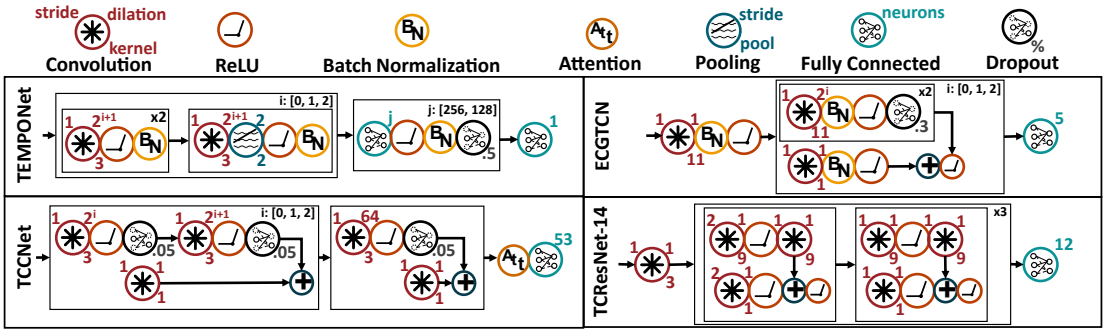

Figure 3.6: Seed network architectures for the four considered benchmarks.

The second phase is where the actual NAS takes place. In the search loop, the model weights W and the architectural parameters θ are optimized simultaneously. Accordingly, the goal of this phase is to minimize the sum of the task-specific loss $\mathcal{L}$ and of the regularization loss, weighted by the regularization strength λ. The duration of the search phase is controlled by an early-stop mechanism that monitors the value of $\mathcal{L}$ on an unseen validation split of the target dataset and stops the search when the latter does not improve for 20 epochs. Note that the higher is λ, the more the minimization of the network is favored.

Finally, in the third and last phase, the θ parameters and corresponding Θ binary masks are frozen to their latest values. This corresponds to sampling from the search space the architecture that PIT determined as optimal during the previous phase. Then, the weights W of the selected network are fine-tuned only considering the task loss $\mathcal{L}$.

To obtain different Pareto points in the accuracy versus cost (size or OPs) space with PIT, Algorithm 1 should be repeated, changing the regularization strength λ. More precisely, the warm-up phase can be performed just once, saving the final weights of the seed network. Overall, Algorithm 1 has a complexity that is comparable to a single TCN training. Moreover, GPU time and memory requirements are greatly reduced with respect to a supernet-based DNAS.

3.2.4 Benchmarks

I test PIT on four edge-relevant real-world benchmarks. The benchmarks include regression as well as classification tasks; the four examples are described in detail in the rest of this section. All the seed networks are depicted in Fig. 3.6.

3.2.4.1 PPG-based Heart-Rate Monitoring

The first benchmark deals with Heart-Rate (HR) monitoring on wrist-worn devices, using Photoplethysmography (PPG) sensors coupled with tri-axial accelerometers to mitigate the effect of motion artifacts [35, 36]. PPG-Dalia [36] dataset is considered since it is the most extensive open-source dataset. The task is formulated as a *regression* of the HR value, whose ground truth is derived with ECG measurements. Both PPG and acceleration are sampled at 32 Hz and organized in 8 s long sliding windows with a time shift of 2 s between successive windows.

The seed network for this task is TEMPONet, a TCN originally proposed in [4] and later used for HR monitoring with state-of-the-art results in [35]. The network architecture is depicted in the top left of Figure 3.6. The network is composed of three *feature extraction* blocks and a final *regressor* module with three FC layers. Each feature extraction block is made of three convolutional layers with BatchNorm and ReLU activation, followed by an average pooling. The FC layers are also followed by BatchNorm and ReLU, and by a dropout layer with 50 % rate. With respect to the original TEMPONet, the seed is obtained doubling the receptive field of all convolutions and setting the dilation to 1.

3.2.4.2 ECG-based Arrhythmia Detection

The second benchmark deals with Electrocardiogram (ECG)-based arrhythmia detection for wearable medical devices. I consider the ECG5000 dataset [73], and the task consists in classifying the ECG signals in 5 classes: Normal, R-on-T Premature Ventricular Contraction, Premature Ventricular Contraction, Supraventricular Premature or Ectopic beat, and Unclassified Beat.

The reference TCN is ECGTCN, shown in the top right of Figure 3.6. Differently from TEMPONet, ECGTCN is based on residual blocks. It has a first convolutional layer that enlarges the number of input channels, followed by three modular blocks, each including two dilated convolutions with ReLU activation, BatchNorm, and 50% dropout. The input and output feature maps of each block are then summed together. When the number of input and output channels differs, the residual path also includes a point-wise convolution (i.e., $K = 1$) to adapt the tensor sizes. The PIT seed is obtained from ECGTCN, setting the dilation of all layers to 1, while keeping the original receptive field.

3.2.4.3 sEMG-based Hand-Gesture Recognition

The third benchmark deals with hand-gesture recognition based on surface electromiography (sEMG) signals. Executing gesture recognition at the edge is a crucial enabler for applications such as complex human-computer interfaces, non-invasive prosthesis control, and rehabilitation. For this task, I target the NinaPro DB1 dataset [74], which includes records of 27 healthy patients monitored with 10 electrodes while performing 52 heterogeneous hand gestures, including basic finger and wrist movements, different hand poses, and grasping.

The seed network is TCCNet, originally proposed in [39], and depicted in the bottom left of Figure 3.6. The architecture includes three feature extraction blocks, each composed of two dilated convolutions with ReLU and dropout (5% rate) and a residual branch with a point-wise convolution. The classifier includes an attention layer of the type described in [75] and a final FC layer with 53 output neurons (52 hand-gestures + 1 unknown class).

3.2.4.4 Keyword Spotting

The last benchmark is keyword spotting (KWS), a key component of speech-based human-machine interfaces (e.g., for smart personal assistants). The standard benchmark for KWS systems, i.e., the Speech Commands v2 dataset [76], which consists of 105829 utterances collected from 2618 speakers, is used. I follow the pre-processing scheme proposed by the MLPerf Tiny industry-standard benchmark suite [77], which produces 12 possible labels, including 10 words and two special classes for "unknown" and "silence".

As the seed, the TCN presented in [38], called TC-ResNet14, whose architecture is shown in the bottom right of Figure 3.6, is used. The main difference with the other reference TCNs is that the original TC-ResNet14 did not use dilation and the modular convolutional blocks alternate plain convolutions with strided convolution with $s = 2$. PIT's seed is obtained by doubling the receptive field in each layer.

3.2.5 Experimental Results

This section discusses the results obtained by PIT on the four benchmarks mentioned above. In Section 3.2.5.1, I present the global results of the NAS search in the accuracy versus the number of parameters and the number of OPs planes. In Section 3.2.5.2, I conduct ablation studies on the PPG benchmarks, and in Section 3.2.5.3 PIT is compared with a state-of-the-art DNAS, ProxylessNAS [18], and with two state-of-the-art

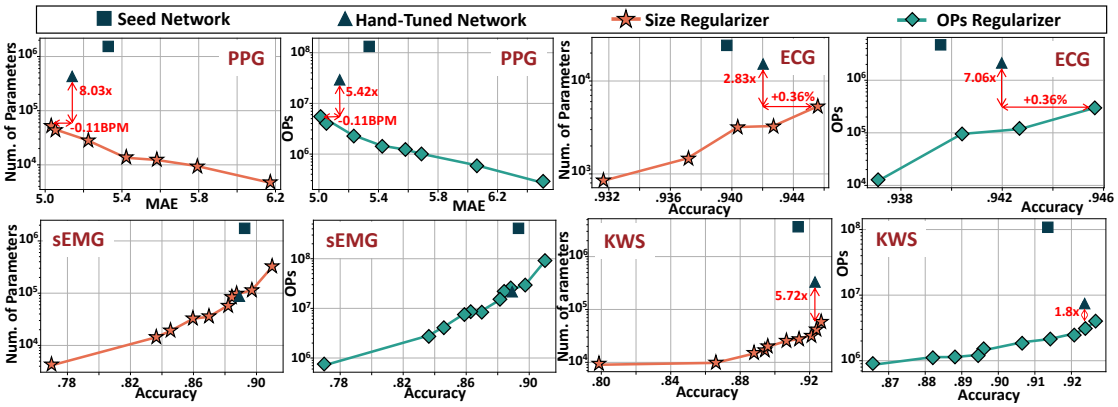

Figure 3.7: Overall PIT Pareto fronts for the four target benchmarks, and comparison with seed and hand-tuned TCNs.

DMaskingNAS approaches, namely, MorphNet [16] and FBNetV2 [15]. Since the code for [15] is not publicly available, it has been re-implemented based on the information provided in the paper. Finally, Section 3.2.5.4 presents the memory, latency, and energy consumption results obtained deploying some of the networks found by PIT on two commercial edge devices, GAP8, and the STM32H7. As an inference software backend, I use the open-source layers library described in the following chapter for GAP8 and the CMSIS-NN library [78] for the STM32H7. All deployed networks are quantized to 8-bit, using PyTorch's built-in quantization algorithm.

3.2.5.1 Search Space Exploration

Figure 3.7 shows the results of applying PIT to the four benchmarks. The graphs report the TCNs accuracy (for classification tasks) or Mean Absolute Error (MAE, for regression tasks) on the x-axis and the number of parameters or OPs per inference on the y-axis. The curves correspond to the outputs of PIT. The different curves' points are obtained by varying the regularization strength λ. I also consider both size and OPs regularizers. Moreover, each plot also reports the metrics of two additional TCNs. Black triangles correspond to the results obtained by the *hand-tuned* state-of-the-art TCNs of Figure 3.6, directly taken from [35, 38, 39, 79], with the original number of channels, receptive fields, and dilation factors. Black squares, instead, indicate the metrics of the PIT *seeds*, i.e., the same networks modified as described in Section 3.2.4 (setting $d = 1$ everywhere, etc.) to enlarge the PIT search space.

The upper-left part of Figure 3.7 reports the results of the PPG-Dalia dataset for the PPG-based HR monitoring task. The HR tracking is the only regression task considered, so the network performance is measured with the MAE, for which lower values are better. As shown by the graphs, starting from a single seed network, PIT can obtain a rich

collection of Pareto-optimal architectures, spanning more than one order of magnitude both in terms of parameters (4.7k-78k) and OPs (0.27M-9.6M). Notably, PIT networks dominate trade-off accuracy vs. OPs, the seed architecture, and the hand-tuned state-of-the-art TEMPONet. In particular, I obtain a similar MAE to the seed TCN (5.38 vs 5.40 BPM), with 120.0× fewer parameters and 96.0× fewer operations. Moreover, PIT also finds a new state-of-the-art deep learning model for this task, achieving an MAE of just 5.03 BPM. It also requires only 53k parameters and 5.1M OPs, improving the best-performing architecture proposed in [35][2] requiring 8.03× and 5.42× fewer parameters and OPs.

The other parts of the figures refer to classification tasks, whose performance is measured in terms of accuracy. The Upper-right pair of charts shows the results obtained on the ECG5000 dataset for Arrhythmia Detection. PIT results span almost one order of magnitude in parameters (0.91k-5.36k) and OPs (50.3k-293.5k). Moreover, both the seed network and the hand-tuned one are Pareto-dominated. The best performing architecture found by the NAS improves the accuracy of the hand-tuned network (+1.03%), reducing both the number of parameters (-64.7%) and the FLOPs (-85.8%).

The lower-left part of Figure 3.7 shows the results obtained for the sEMG-based Hand-Gesture Recognition task on the NinaPro-DB1 dataset. Also, in this case, I found architectures in a wide range of sizes and numbers of OPs. However, while PIT results still dominate the seed, the hand-tuned TCNNet sits on the Pareto front. Indeed, the PIT network nearest to the hand-tuned architecture on the curve achieves a slightly lower accuracy (-0.47%) traded-off with a reduction of size (-3.33%). This result demonstrates the goodness of the original TCNNet proposed in [39] but, at the same time, it shows the excellent quality of the architectures found by PIT, which despite starting from an oversized seed, is still able to produce optimized networks that closely resemble those tuned by experts.

Lastly, the lower-right part of Figure 3.7 shows the two Pareto fronts obtained on the Google Speech Commands dataset for Keyword Spotting. Once again, PIT vastly outperforms both the seed and the hand-tuned TCNs. Specifically, the most accurate PIT architecture slightly improves the accuracy of the hand-tuned network (+0.36%) while significantly reducing both the number of parameters (-82.53%) and FLOPs (-44.53%). Moreover, the Pareto points span 10k-98k parameters and 0.87M-3.98M OPs.

I report in Table 3.2 the range of regularizer strengths λ that I used for the experiments on the four benchmarks to obtain the Pareto frontier analysis. In general, λ should be set so that the two additive terms in the loss ($\mathcal{L}$ and $\lambda\mathcal{R}$) assume comparable

[2]Note that this result is achieved without applying any additional post-processing as described in [35].

Table 3.2: Range of regularizer strength (λ) values for the four benchmarks.

Regularizer	PPG	ECG	sEMG	KWS
$\mathcal{R}_{size}$	1e-7 : 5e-4	5e-7 : 7.5e-3	1e-7 : 5e-6	5e-10 : 1e-5
$\mathcal{R}_{ops}$	1e-8 : 5e-5	5e-8 : 5e-4	5e-10 : 5e-8	1e-10 : 1e-6

values at the beginning of training. With this setting, I ensure that PIT considers both accuracy and inference cost in its search without degenerating to one of the two corner cases in the first few epochs. The corresponding values of λ vary for different tasks, as shown in the table. However, a good rule of thumb, which works for all benchmarks, to identify the order of magnitude of the regularization strength is to start from $\lambda = 1/(\text{Seed Model Size})$. Then, based on the results of a PIT search with this initial value, one can decide to increase/decrease λ to obtain smaller/more accurate TCNs, respectively. By monitoring the loss in the initial epochs, it is also straightforward to detect when the NAS is falling in one of the corner cases (one term much larger than the other) and stop the search immediately without wasting training time.

3.2.5.2 Ablation Studies

This section analyzes the impact of some of the most important PIT parameters. I report the results of this study only for the PPG-based HR monitoring benchmark since the same conclusions also hold for the other datasets.

Hyper-parameters Figure 3.8 analyzes the contribution of different hyper-parameters to the quality of results found by PIT. For this experiment, I used the $\mathcal{R}_{size}$ regularizer while considering solutions in the MAE versus the number of parameters space. Three curves are obtained by running PIT individually on the three different hyperparameters (α, β, and γ), freezing the other ones. This gives i) the results of a search that only optimizes the number of channels in each layer (*Ch-Only*), performed on a TCN with maximal receptive field and $d = 1$, ii) the results of a receptive field-only search (*Rf-Only)*, on a TCN with maximal C_{out} and $d = 1$, and iii) the results of a dilation-only search (*Dil-Only*) on a network with maximal F and C_{out}. The Pareto fronts obtained in each of these 3 conditions by varying the regularization strength λ are shown in the figure, together with the output of a complete search that optimizes all three hyper-parameters simultaneously (*All-in-One*).

The primary source of parameter reduction and performance improvement is the search along the channels dimension since the channels represent a significant source of redundancy in hand-tuned TCN (their number is typically set using common heuristics).

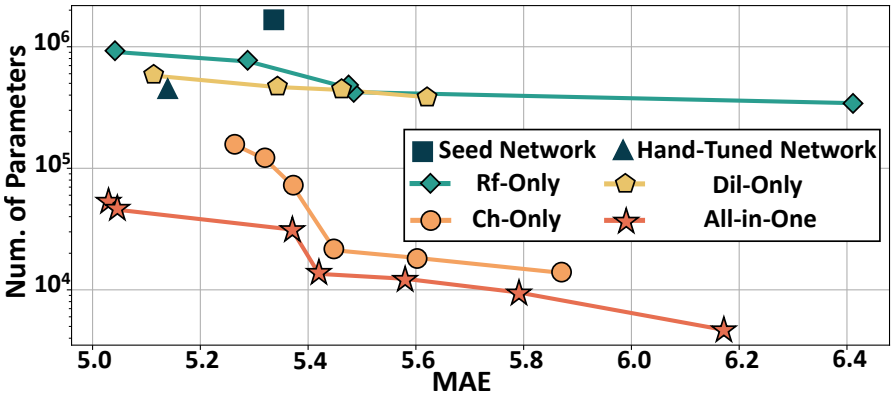

Figure 3.8: Comparison between the results of PIT searches with different combinations of hyper-parameters for PPG-Dalia.

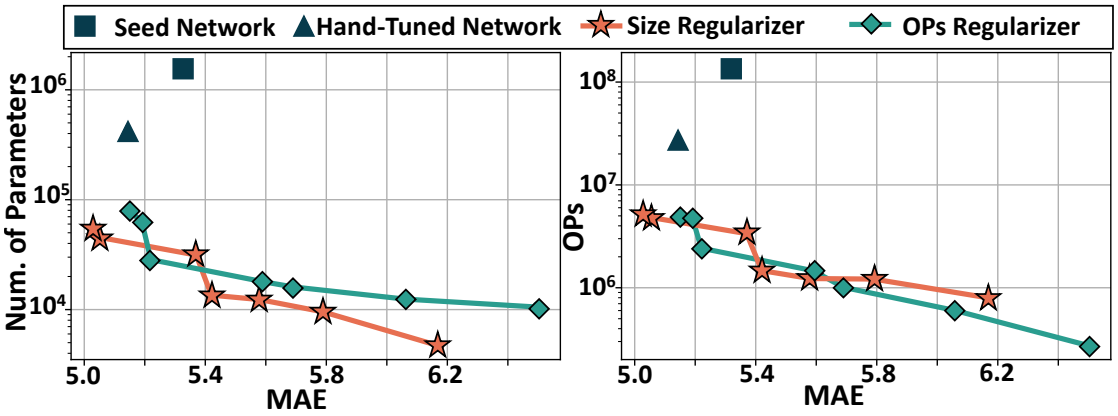

Figure 3.9: Comparison of $\mathcal{R}_{size}$ and $\mathcal{R}_{ops}$ regularizers for PPG-Dalia.

However, Figure 3.8 also shows that optimizing *only* the number of channels is insufficient and that a combined optimization that considers receptive field and dilation can yield Pareto-optimal networks across the entire MAE/parameters range.

Regularizers Note that the PPG-based HR monitoring benchmark on which I show this ablation study is the one for which the distinction between model size and number of OPs is most relevant due to the presence of both strided convolutions and pooling, which strongly impact the time dimension T.

The figure shows that, as expected, the majority of the Pareto points in the MAE versus the number of parameters plane are produced using the R_{size} regularizer, with the few exceptions being local minima. Vice versa, the R_{ops} regularizer tends to generate superior solutions regarding MAE versus the number of OPs.

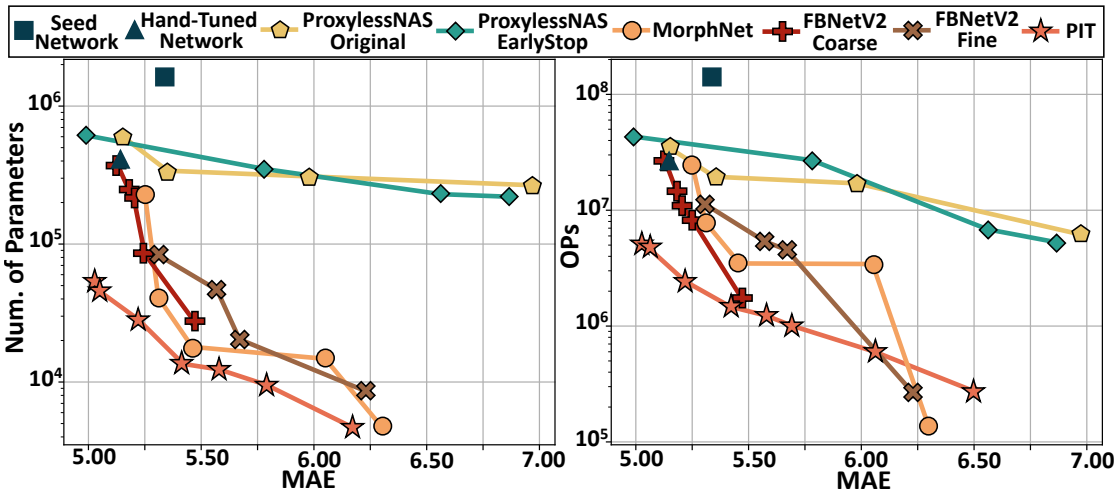

Figure 3.10: Quality of results comparison between PIT and state-of-the-art NAS tools on the PPG-Dalia dataset.

3.2.5.3 Comparison with state-of-the-art NAS tools

Figure 3.10 compares the Pareto fronts obtained with PIT and three state-of-the-art NAS tools, namely ProxylessNAS [18] MorphNet [16] and FBNetV2 [15], on the HR monitoring benchmark. Results show that PIT outperforms all three across the entire design space, except for one MorphNet and one FBNetV2 point, which achieve a low number of operations, although at the cost of a larger MAE. The main reason for the superior results of PIT is the fact that the NAS explores a larger and finer-grain search space compared to the baselines. For MorphNet and FBNetV2, this is partly due to the intrinsic nature of those tools, which are limited to exploring one parameter. For this reason, I set up the remaining hyperparameters to their values inside the seed model for these NAS searches. This different search starting point compared to PIT is the reason why, in the low-size/high-MAE regime, these tools find a single Pareto-optimal point.

For FBNetV2, I considered both a *Coarse* search space and a *Fine* search space. For the first one, I included 4 C_{out} alternatives per layer, uniformly spaced, i.e., $1/4 C_{out,seed}$, $1/2 C_{out,seed}$, $3/4 C_{out,seed}$ and $C_{out,seed}$. The latter instead evaluates all C_{out} values with a granularity of 1. The latter is more similar to PIT, but the former generally achieves superior results. This is because FBNetV2 uses a pre-defined binary mask for each layer variant, combining them through a Gumbel softmax, as explained in Sec. 3.2.1.1. Experimentally, it is shown that with too many masks, the search becomes unstable and yields sub-optimal results. In contrast, PIT does not have this limitation since it uses *independent* trainable masks that keep or eliminate an individual channel.

ProxylessNAS would be virtually able to explore the entire PIT search space, but at the cost of a very long training time, leading to the impossibility of training the

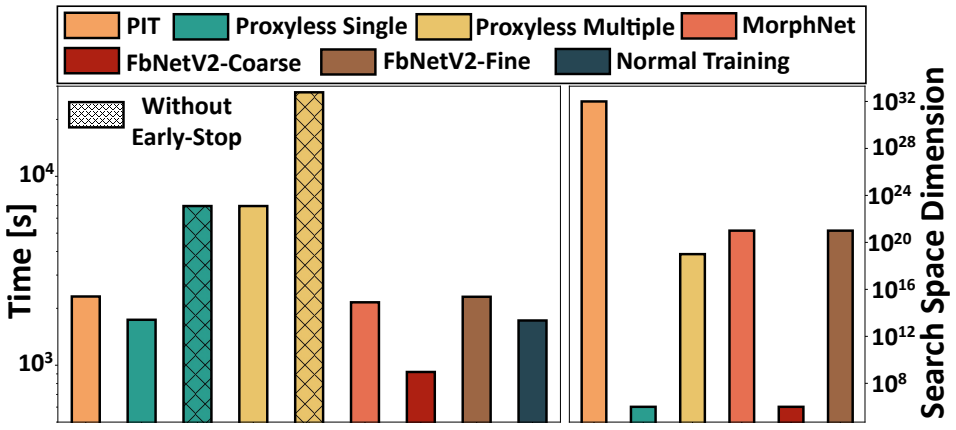

Figure 3.11: Search space and time comparison between PIT and state-of-the-art NAS tools on the PPG-Dalia dataset.

network due to its requirements. In fact, as detailed in Section 3.2, PIT explores C_{out} and F with a granularity of 1, and for d, it considers all possible power-of-2 values. Therefore, each super-net node should include $C_{out,seed} \cdot F_{seed} \cdot \lceil \log_2(F_{seed}) \rceil$ different layers, connected in parallel. With the same parameters used for the example at the end of Section 3.2.1, this would correspond to ≈10000 different versions of *each layer*.

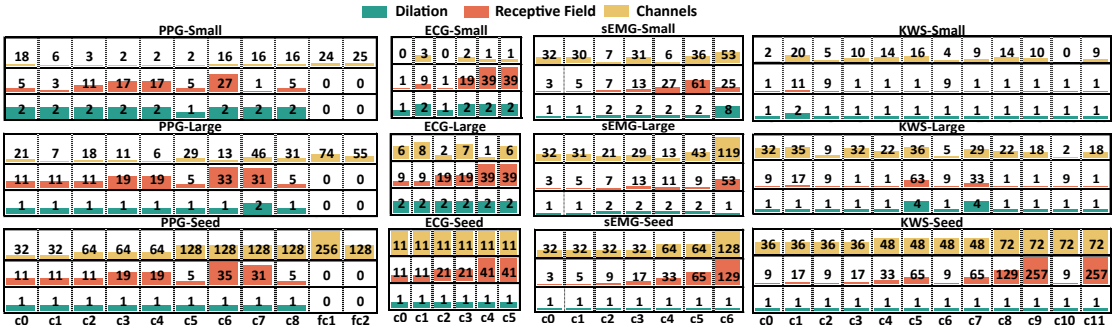

Figure 3.12: Hyperparameters of the deployed PIT architectures and corresponding seed network for the four benchmarks.

Therefore, for ProxylessNAS, I tried to make the comparison with PIT as fair as possible while keeping the search space size similar to the one of the original paper [18]. To do so, I used an iterative procedure. First, multiple ProxylessNAS searches on C_{out}, F, and d separately, keeping the two not-optimized hyper-parameters at the seed values are performed. In each of these searches, I consider 4 variants in each super-net layer, uniformly sampling the PIT search space (in the same way described above for FBNetV2-Coarse). After repeating the same procedure for F and d, for every layer, the two values of each hyper-parameter have been chosen. The 2^3 possible combinations of the latter are used to generate the *combined* search space for ProxylessNAS, which includes 8 layer variants in each super-net node. The Pareto fronts of Figure 3.10 are obtained running ProxylessNAS multiple times on this combined search space, with different regularization strengths. I show both the results from the original paper training procedure (*Original*

Table 3.3: Detailed deployment results for the four benchmarks.

				GAP8		STM32	
Task	TCN	Perf. int8 (float32)	Mem. [kB]	Lat. [ms]	En. [mJ]	Lat. [ms]	En. [mJ]
PPG	HT	5.01 (5.14) BPM	423	23.2	1.2	58.3	13.6
	S	5.71 (6.17) BPM	4.7	1.18	0.06	3.2	0.75
	L	5.01 (5.03) BPM	53.2	4.25	0.22	15.2	3.56
ECG	HT	94.2 (94.2) %	15.2	2.69	0.14	6.66	1.56
	S	92.84 (93.16) %	0.9	0.78	0.04	1.8	0.42
	L	94.13 (94.13) %	5.4	1.26	0.06	2.84	0.66
sEMG	HT	88.89 (88.87) %	88.8	61.0	3.11	291	68.1
	S	86.97 (86.98) %	35.4	39.6	2.02	169	39.5
	L	91.2 (90.99) %	317.8	238	12.1	960	225
KWS	HT	92 (92.31) %	323.4	13.4	0.68	30.7	7.17
	S	87 (86.58) %	9.8	1.40	0.07	2.66	0.62
	L	92.16 (92.64) %	56.5	3.74	0.19	10.6	2.48

curve), which runs for a fixed number of epochs, and from the early-stop mechanism employed for PIT (*EarlyStop* curve).

Further, Figure 3.11 compares the search space dimension and average execution time of PIT, Morphett, and ProxylessNAS. For reference, the execution time of a standard training of the seed network is also reported. All time results refer to the search phase only (without warm-up), and are obtained on a single NVIDIA Titan XP GPU with a batch size of 128. For ProxylessNAS, I report the results of initial single-hyper-parameter searches (*Proxyless-Single*) and the final combined search (*Proxyless-Multiple*), both with and without early-stopping.

The algorithm explores a $10^{26}\times/10^{12}\times$ larger search space than Proxyless-Single/-Multiple. Further, it is only 1.13× slower than Proxyless-Single with early-stopping, and 3.55× faster than the variant without early-stopping, while it is $3.0\times/14.22\times$ faster compared to Proxyless-Multiple with/without the early-stopping training. Compared to MorphNet, PIT explores a $10^{11}\times$ larger search space at the cost of a small 1.07× increase in runtime. FBNetV2-Coarse is the fastest tool, converging in a few search epochs. Whereas offering a 2.5× speedup compared to PIT, the explored search space is 10^{26} smaller. Instead, FBNetV2-Fine explores a 10^{11} smaller space while requiring the same search time of the proposed approach. Lastly, PIT's time-overhead of this approach compared to regular training is only 34%.

3.2.5.4 Embedded Deployment

This section analyzes the results obtained by deploying two TCNs for each target benchmark on the GAP8 IoT processor (running at 100 MHz) and on the STM32H7 MCU (at 480 MHz). For each task, I chose the best performing network in terms of

MAE or accuracy (L) and the smallest network that achieves a MAE drop < 1 BPM, or an accuracy drop $< 5\%$ compared to the best performing one (S). For comparison, I also deploy the baseline hand-tuned architectures (HT). Table 3.3 reports the results in terms of performance (MAE or accuracy, depending on the dataset), memory footprint, inference latency, and energy consumption, while Figure 3.12 shows the hyper-parameters selected by the NAS.

PIT finds competitive solutions for both hardware targets and for all 4 tasks, despite the large difference in complexity among them, testified by the more than two orders of magnitude span in memory, latency, and energy consumption in the results of Table 3.3. For PPG-based HR monitoring, the L/S models achieve a 0/0.70 BPM MAE increase compared to the hand-tuned TEMPONet, respectively, while resulting in a 8.03/90.8× lower memory footprint and a 5.45/19.6× lower latency and energy consumption on GAP8. On the STM32 MCU, the latency and energy reduction of the two PIT outputs is 3.83/18.2×. PIT's L/S models for ECG processing, instead, achieve +0.07%/-1.36% accuracy compared to the hand-tuned ECGNET, with a 2.83/16.8× lower memory footprint, 2.13/3.44× latency and energy reduction on GAP8, and 2.34/3.7× on the STM32. For the sEMG gesture recognition task, the L/S models found by PIT obtain +2.31%/-1.92% accuracy compared to TCCNet. The higher accuracy is paid with 3.57× larger memory footprint and a 3.85× latency on GAP8 (3.33× on STM32H7). The small TCN, instead, results in a 2.51× memory reduction and 1.54× and 1.72× lower latency and energy on the two targets. Lastly, the L/S PIT outputs for KWS obtain +0.16%/-5% accuracy compared to TCResNet-14, with a 5.72/33.1× lower memory footprint, 3.58/9.54× lower energy and latency on GAP8, and 2.9/11.54× lower energy and latency on STM32H7.

To conclude the results, I show in Figure 3.12 the different TCNs deployed to show the high variability of hyper-parameters settings found by PIT among the various benchmarks.

3.3 Channel-wise Mixed-precision Assignment for DNN Inference on Constrained Edge Nodes

In this section, I will introduce a second Dmasking NAS, similarly in structure to the previous one, but with totally different scope. In particular, I show a lightweight DNAS method able to learn an independent precision assignment for each weight tensor *channel* in convolutional (Conv) or fully-connected (FC) layers. In other words, an independent precision is assigned to the weights of *each filter* in Conv layers[3], and to the weights associated with *each output neuron* in FC layers. Reducing the precision of the weights allows me to reduce the memory occupation of the network and, accordingly, its number of operations. In the following, I discuss the details of the method for Convolutional layers only since the extension to FC is straightforward.

Note that the channel-based precision assignment explores a larger and finer-grain solution space than the layer-wise assignment. For instance, considering a MobileNetV1 with a width multiplier of 0.25, the number of solutions grows from 10^{26} (considering also different precisions for the activations) in a layer-wise approach to 10^{74} in ours. However, this allows this NAS to exploit the *different relative importance of extracted features* within a single layer to optimize the model further.

3.3.1 Precision Assignment Optimization Method

Fig. 3.13 summarizes the flow of this approach. For each optimized layer of the target DNN, first the fake-quantization is applied to the activation tensor X at all supported bit-widths $p_x \in P_x$, e.g., $P_x = \{2, 4, 8\}$ bit. I then combine the fake-quantized tensors through a vector of NAS parameters $\delta^{(n)} \in \mathbb{R}^{|P_x|}$, where the n superscript refers to the n-th layer, in a way similar to [80]. Specifically, I first compute $\hat{\delta}^{(n)} = \text{SM}(\delta^{(n)}; \tau)$ where $\text{SM}(x; \tau)$ is the softmax with temperature:

$$SM(x;\tau) = \frac{e^{x_i}/\tau}{\sum_i e^{x_i}/\tau}, \ \forall i \tag{3.16}$$

so that the elements of $\hat{\delta}^{(n)}$ sum to 1. As detailed in Sec. 3.3.2, the temperature τ is progressively annealed during training, driving the softmax to increasingly resemble a non-differentiable *argmax*, which is the function used to select the final precision at the end of the optimization.

[3] By filter, I consider a slice of weights that processes all channels of an input activation patch, and produces a single output channel.

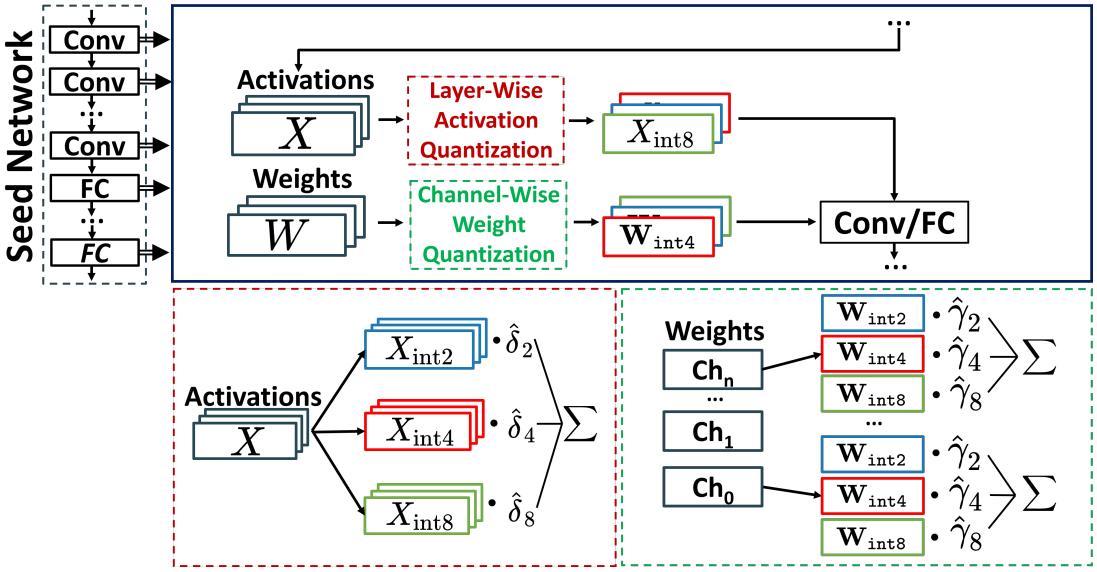

Figure 3.13: Overview of the proposed approach.

Then, the *effective* activation tensor is obtained as the sum of all the activation tensors at the different precisions:

$$\hat{X}^{(n)} = \sum_{p_x \in P_X} \hat{\delta}^{(n)}_{p_x} \cdot X_{p_x} \tag{3.17}$$

where X_{p_x} is the p_x-bit fake-quantized version of the original float tensor X. Similarly to [80], the rationale of (3.17) is that the larger $\hat{\delta}^{(n)}_{p_x}$, the more the final activation becomes similar to the result of a p_x-bit quantization.

To explore the channel-wise assignments of bit-widths $p_w \in P_W$ for weights, which is the main novelty of this NAS, I further associate each layer with a 2D matrix $\gamma^{(n)} \in \mathbb{R}^{C^{(n)}_{out} \times |P_W|}$, where $C^{(n)}_{out}$ is the number of output channels/filters in the layer. The i-th row of the matrix $\gamma^{(n)}_i$ contains the NAS parameters that will determine the bit-width assignment for the i-th channel. Accordingly, each row is applied an independent softmax to obtain $\hat{\gamma}^{(n)}_i = SM(\gamma^{(n)}_i; \tau)$.

Next, the i-th effective weight tensor slice $\hat{W}^{(n)}_i$, i.e., the i-th effective filter, is obtained similarly to (3.17), as:

$$\hat{W}^{(n)}_i = \sum_{p_w \in P_W} \hat{\gamma}^{(n)}_{i,p_w} \cdot W^{(n)}_{i,p_w} \tag{3.18}$$

Lastly, the stack of these slices along the C_{out} dimension is used, together with $\hat{X}^{(n)}$ to produce the layer output:

$$Y^{(n)} = \text{Conv}(\hat{X}^{(n)}, \underset{i}{\text{stack}}(\hat{W}^{(n)}_i)) \tag{3.19}$$

The exposed method uses *weight sharing* to reduce the amount of memory occupied:all fake-quantized weights and activations are obtained from a *single* float tensor and not from different ones, being generated on the fly. A single copy of the weights in the floating point format is therefore trained and stored, minimizing the memory overhead of this method, which is almost identical to the one of [80]. A key difference between this NAS and [80] is instead in the quantization scheme, i.e., the function mapping $X \rightarrow X_{p_x}$ and $W \rightarrow W_{p_w}$. Namely, I replace the original Gaussian quantizer used in [80] with the PaCT method described in [81], since it is fully compatible with most of the hardware libraries for MCUs.

To optimize the precision assignment, I minimize the same loss function of the previously presented PIT

$$\mathcal{L}(W;\theta) = \mathcal{L}_{\mathcal{T}}(W;\theta) + \lambda \mathcal{L}_{\mathcal{R}}(\theta), \tag{3.20}$$

The difference is in the formulation of the complexity regularizer $\mathcal{L}_{\mathcal{R}}$. When the goal is to minimize the *model size*, i.e., the number of parameters weighted by their precisions, the NAS is guided to select smaller bit-widths for weights tensors channels where possible, whereas activations bit-widths have no impact. The regularizer is, therefore, in the form of:

$$\mathcal{L}_{\mathcal{R}}{}^{(n)} = C_{in}^{(n)} K_x^{(n)} K_y^{(n)} \sum_{i=1}^{C_{out}^{(n)}} \sum_{p_w \in P_W} \hat{\gamma}_{i,p_w}^{(n)} \cdot p_w \tag{3.21}$$

where $C_{in}^{(n)}$ $K_x^{(n)}$ and $K_y^{(n)}$ are number of input channels and the horizontal and vertical convolution kernel sizes. In practice, this regularizer computes the *effective number of weight bits* in the layer, multiplying each softmax coefficient $\hat{\gamma}_{i,p_w}^{(n)}$ times the corresponding bit-width p_w.

The second considered regularizer is associated with the *energy consumption* of the network, which depends both on weights and activation precision. The corresponding regularizer is:

$$\mathcal{L}_{\mathcal{R}}{}^{(n)} = \Omega^{(n)} \cdot \sum_{p_x \in P_X} \hat{\delta}_{p_x}^{(n)} \sum_{i=1}^{C_{out}^{(n)}} \sum_{p_w \in P_W} \hat{\gamma}_{i,p_w}^{(n)} \mathrm{C}(p_x, p_w) \tag{3.22}$$

where $\Omega^{(n)}$ is the total number of operations required to produce the n-th layer output, which is independent of the precision assignment and can be computed from well-known formulas for Conv or FC layers. The rest of (3.22) computes the *expected average energy per operation* of the layer. Specifically, $\mathrm{C}(p_x, p_w)$ is a Look-Up Table (LUT) returning an estimate of the energy/OP of a p_x-bit $\times p_w$-bit convolution. The LUT is populated by profiling the target hardware and is necessary because, for most hardware platforms,

Algorithm 2

for $i \leftarrow 1, \ldots,$ Epochs$_{\text{wu}}$ **do** # warmup loop
  Update W based on $\nabla_W \mathcal{L}_\mathcal{T}(W_{p_{max}})$
end for
while not converged **do** # search loop
  if #samples $<$ 20% current epoch **then**
    Update θ based on $\nabla_\theta(\mathcal{L}_\mathcal{T}(W;\theta) + \lambda\mathcal{L}_\mathcal{R}(\theta))$
  else
    Update W based on $\nabla_W(\mathcal{L}_\mathcal{T}(W;\theta))$
  end if
  Anneal temperature τ
end while
for $i \leftarrow 1, \ldots,$ Epochs$_{\text{ft}}$ **do** # fine-tuning loop
  Freeze θ
  Update W based on $\nabla_W \mathcal{L}_\mathcal{T}(W)$
end for

the energy cost of arithmetic operations at sub-byte precision is not linearly proportional to the bit-width. The regularizer weighs each combination of NAS parameters for activations and weights with the cost of the corresponding mixed-precision operation.

The overall regularization loss is obtained as for PIT by summing either (3.21) or (3.22) over all layers.

3.3.2 Training Procedure

Alg. 2 shows the training scheme of this NAS algorithm, which is almost identical to the ones shown in PIT. I will therefore discuss here only the differences between them. In the initial *warmup* phase, instead of training the floating point network, a quantization-aware training at the maximum supported precision p_{max} is performed while keeping NAS parameters frozen. In all the experiments, p_{max} is equal to 8bit. Warmup, as already said, needs to be performed only once, reusing the result for multiple searches.

The second phase represents the core of the optimization. The main difference is that the training dataset is randomly split into 20%-80%. First, θ are trained on the first 20% to minimize (3.20). Then, only the network weights are trained on the remaining 80%. At the end of each epoch, I anneal the softmax temperature τ to choose among alternative bit-widths more "decisively". In all experiments, τ is initially set to 5 and progressively annealed by $e^{-0.0045}$ as in [15].

The *fine-tuning* phase is identical to the PIT training algorithm.

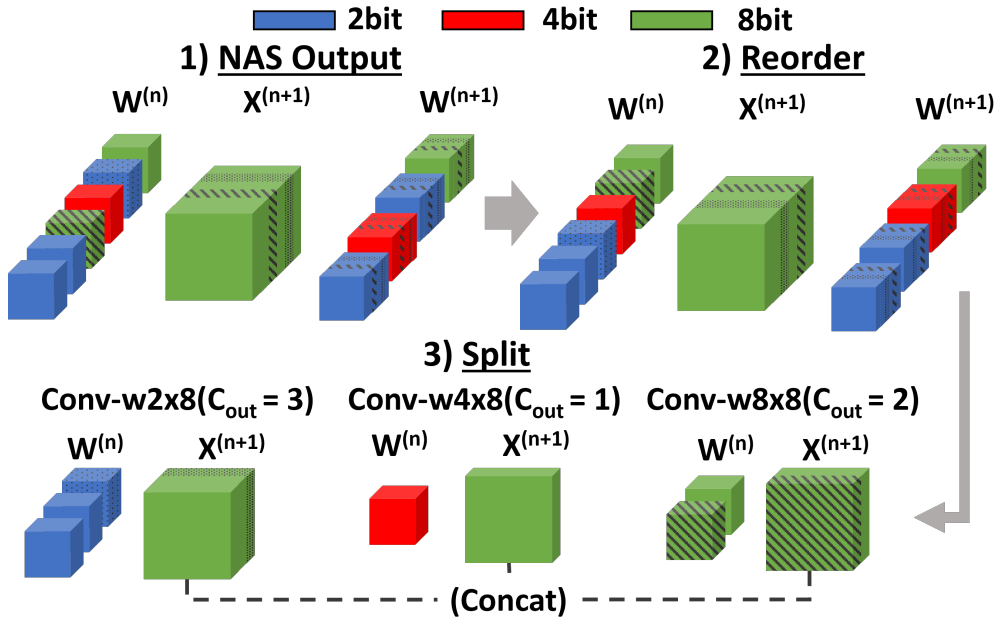

Figure 3.14: Layer re-organization to support channel-wise precision assignment.

3.3.3 Implementation Details

In this section, I show that besides having the potential to improve the *theoretical* compression and efficiency of DNNs compared to standard mixed-precision, this method is also *fully compatible* with existing hardware and software libraries for mixed-precision inference, with minimal overheads.

The top-left part of Fig. 3.14 shows the DNAS output for a generic Conv layer, with all filters assigned 2, 4, or 8-bit, independently from their ordering in memory (e.g., the 3rd and last filter are 8-bit, while the 4th filter is 2bit, etc.). To deploy such a layer, the filters are *reordered* by bit-width (top-right of the figure). Note that by doing so, also the output activations are influenced. Accordingly, to preserve the functionality of the *following* layer, its weight tensor has to be re-organized along the C_{in} axis so that each weight is still multiplied with the correct input activation.

After this re-organization, which is performed offline and does not have run-time overheads, the layer can be *split* into $|P_W|$ separate convolutions working in parallel, each with a fraction of the original output channels. These sub-layers have a single bit-width for W. Hence they are fully compatible with existing mixed-precision hardware and libraries [2, 30]. Moreover, since the activation bit-width is assigned layer-wise, all outputs have the *same* precision. Therefore, they can be concatenated (i.e., stored in adjacent memory locations) after executing the three "parallel" virtual layers created at different precisions. Allowing this simple concatenation is why I do not show the channel-wise precision assignment for activations. In that case, the next layer's filters would have to process an interleaved mix of different precision inputs, which is not

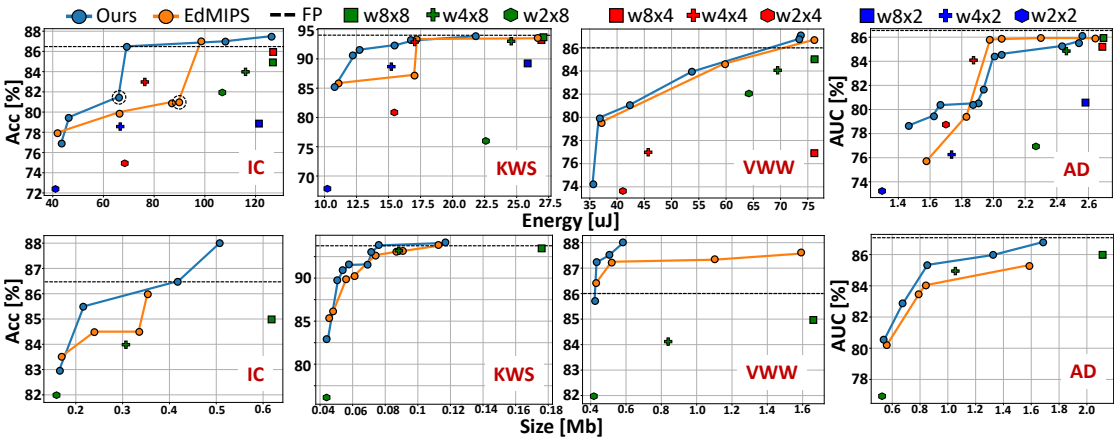

Figure 3.15: Pareto fronts obtained for the four MLPerf Tiny benchmarks, and comparison with EdMIPS and fixed-precision solutions.

currently supported by inference libraries, causing a high overhead in the embedded execution and nullifying the gains obtained by the mixed-precision inference.

3.3.4 Experimental Results

3.3.4.1 Setup

I evaluate this DNAS on the four benchmarks of the MLPerf Tiny suite [29]. Below is a summary of each task, while readers can find more details in [29]. *Image Classification* (IC) targets the CIFAR-10 dataset using a custom ResNet-like CNN with a backbone of 8 convolutional layers. *Keyword Spotting* (KWS) targets the Google Speech Commands (GSC) v2 dataset with a small Depthwise Separable CNN (DS-CNN) originally proposed in [82]. *Visual Wake Word* (VWW) uses the MSCOCO 2014 dataset for a presence detection task, solved with a MobileNetV1 [83] with a width-multiplier of 0.25. Lastly, *Anomaly Detection* (AD) targets the Toy-car subset of the DCASE2020 dataset with a Dense Autoencoder.

The proposed DNAS is implemented in Python 3.9 using PyTorch v1.10.2. The deployment target, in this case, is MPIC [30], which, compared to the previous deployment targets, has optimized hardware units for the execution of MAC operations with inputs independently quantized to $p_{w/x} \in \{2, 4, 8\}$ bit. The LUT implementing the cost function of (3.22) is built using the energy/OP values profiled from the MPIC core running at 250MHz.

3.3.4.2 Search-Space Exploration

Fig. 3.15 shows the results of applying the mixed-precision assignment method to the four MLPerf Tiny benchmarks. The vertical axes of all graphs report the task scores, i.e., the accuracy for IC, KWS, and VWW and the Area Under the ROC Curve (AUC) for AD. The horizontal axes of the first row of graphs report the energy consumption of the models in μJ, whereas those in the second row report the model sizes in Mb.

Each plot compares the mixed-precision solutions found with this tool (blue dots) with the results of EdMIPS [80] (orange dots), the reference DNAS, which optimizes precision per-layer. To have a fair comparison, in which the presented method's advantages are solely due to the channel-wise precision assignment, I run it and EdMIPS with identical training protocols, including the 20/80% alternate θ/W training and the τ annealing. Moreover, I replace the original quantization algorithm of [80] with PaCT [81], since the former would not be compatible with deployment on MPIC.

I also compare against all relevant fixed-precision quantization baselines, i.e., solutions in which all tensors are quantized to the same precision. I only report wNx8 baselines in memory plots since the activations bit-width is not relevant for model size. Lastly, the accuracy of a floating point version of each model is shown as a horizontal dashed line. I do not report the float model energy and size in the figure since MPIC does not have a Floating Point Unit (FPU).

Each Pareto-optimal point reported in the graphs for this method and for [80] refers to a DNN with a different precision assignment. Multiple points are again obtained changing the regularization strength λ in (3.20). I use the regularizer expressions of (3.21) and (3.22) for memory and energy results respectively, both for this tool and for [80]. Noteworthy, for all the benchmarks, neither the float models nor the full 8-bit one outperforms the best mixed-precision assignment due to the well-known overfitting reduction effect of low bitwidth quantization [84].

The left-most part of Fig. 3.15 shows the results obtained on the IC task. The fine-grain mixed-precision approach outperforms all fixed-precision baselines and EdMIPS in terms of energy and model size. Noteworthy, it saves up to 26.4% energy and 35% memory compared to EdMIPS at iso-accuracy while also obtaining a higher maximum accuracy, +0.5%/+1% depending on the regularizer used. Similar results are obtained also for KWS (middle-left part of Fig. 3.15). Again, the proposed method Pareto-dominates, all comparison baselines, finding solutions that save up to 27.2% energy and 15.6% memory for the same accuracy and improve the best score by +4.3% and +0.7% respectively.

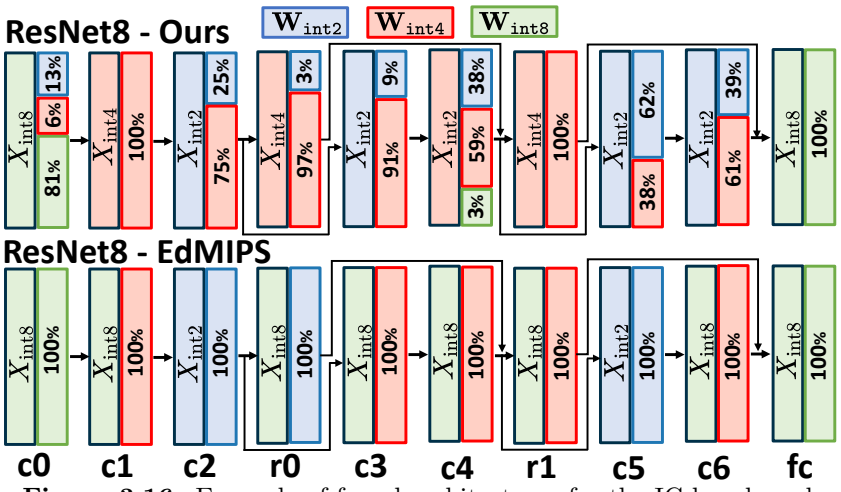

Figure 3.16: Example of found architectures for the IC benchmark.

The middle-right part of Fig. 3.15 reports the results on VWW. On this benchmark, this NAS Pareto dominates fixed-precision networks in terms of memory and energy. Conversely, compared to EdMIPS, this method is particularly beneficial in reducing the memory footprint of the network, with up to 63.4% saving at equal accuracy, and a maximum accuracy improvement is +0.4%. This benefit is due to the reduced precision of many channels in layers that still need a part of high-precision channels. Note that, for this benchmark, the fixed-precision networks with 2bit activations are not shown because their training does not converge.

Lastly, the right-most part of Fig. 3.15 shows the results on AD. As for all other benchmarks, the proposed method obtains superior results regarding AUC versus model size. The memory saving is at most 46.1% for the same AUC compared to EdMIPS. In this case, however, [80] outperforms the proposed NAS in terms of AUC versus energy in the high score regime (up to 21.8% saving), while it is superior for lower AUC values (up to 11.6%). I attribute this result to this NAS's more difficult optimization problem. The AD Autoencoder is composed solely of FC layers with 128 channels (i.e., neurons), except for the bottleneck. With so many channels, the difference between the search space explored by the described method and by [80] explodes. Hence, the gradient-based DNAS optimization likely ends up in a local minimum. Nonetheless, I think that this issue could be solved by tuning the training hyper-parameters precisely for this task. In contrast, I keep all settings identical across benchmarks in these experiments for fairness and reproducibility.

3.3.4.3 Results Analysis

Fig. 3.16 shows an example of the precision assignments determined by this tool and by [80], to deliver to the readers how the precision is assigned to the weights of the different layers. The two architectures are ResNet-8 for the IC task, obtained with

the energy regularizer of (3.22), and correspond to the two circled Pareto points of Fig. 3.15, i.e., those for which the proposed method obtains the most significant energy saving with no accuracy drop. Specifically, the channel-wise model saves 26.4% energy with an accuracy improvement of +0.5%. Each rectangle represents a Conv (cn or rn, where the latter are those in residual branches) or FC layer, with the activation bit-width reported on the left and the fraction of weight channels associated with each precision on the right.

This example shows some interesting insights into the effectiveness of the approach. For instance, readers can notice that EdMIPS quantizes most of the activations with 8bit. In contrast, this proposed method exploits its additional flexibility to *reduce* the activation precision, compensating it with an increase in the bit-width assigned to an often small subset of the weights channels (e.g., only 3% in c4) to obtain the same final accuracy. Eventually, only the first and last layer activations, which notoriously often require higher precision [85] remain at 8bit. Although a single example is shown, similar considerations apply to the results on the other 3 benchmarks.

3.4 Multi-Complexity-Loss DNAS for Energy-Efficient and Memory-Constrained Deep Neural Networks

As previously said and shown for the two illustrated Dmasking NAS, these tools minimize the following function:

$$\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda\mathcal{R}(\theta). \tag{3.23}$$

However, there are two main limitations. First, $\mathcal{R}$ models a *single cost metric*, i.e., either the model size, the number of OPs, or a differentiable approximation of the latency or energy consumption, as a function of the DNN hyper-parameters [16, 18]. Second, the cost is considered an objective to minimize rather than a constraint. While this is appropriate for some metrics (e.g., OPs, latency, or energy), it is sub-optimal when considering memory occupation. Most designers are interested in finding the "best" model (e.g., the most accurate or the best balance between accuracy and energy consumption) that *fits a memory constraint*, given by the target hardware. Doing so with (3.23) requires repeating the DNAS multiple times, sweeping λ, until a model with an appropriate memory footprint is found.

In this section, I will propose a new DNAS formulation that addresses both problems, considering both memory occupation and other cost metrics (OPs, in the experiments) simultaneously, taking the former as a constraint and the latter as an objective. This enables the search for Pareto-Optimal architectures in the Accuracy vs. OPs plane, *around a fixed memory budget*.

Most state-of-the-art DNAS tools [15, 16, 18] sum the task-specific loss $\mathcal{L}$ and the complexity term $\mathcal{R}$, scaled by a strength constant, using the scheme of (3.23). For instance, the two proposed NASes regularize either against the number of parameters or against the number of OPs. However, more recently, UDC [86] proposed a different approach, where the regularization term is not minimized. In contrast, the term $|\mathcal{R}(\theta) - r^*|$ is minimized.

Building upon both these ideas, I propose to enhance the formulations of previous sections with a *two complexity loss terms*, which drive the DNAS towards a desired search space region, while still allowing the exploration of accuracy versus complexity trade-offs. The proposed optimization problem formulation takes the form:

$$\min_{W,\theta} \mathcal{L}(W;\theta) + \lambda|\mathcal{S}(\theta) - s^*| + \mu\mathcal{O}(\theta) \tag{3.24}$$

In the equation, $\mathcal{S}$ models the size (i.e., memory footprint) of the DNN as a function of the architecture parameters θ.

Since, as explained above, memory occupation is usually a constraint that DNNs should respect for edge deployment, rather than a metric to optimize, I follow the approach of [86], minimizing the absolute value difference from a target size s^* which depends on the hardware.

I associate this cost term with a relatively large and *fixed* regularization strength λ, with $\lambda >> \mu$, thus forcing the NAS to find a set of θ^* parameters that yield $\mathcal{S}(\theta^*) \approx s^*$. This allows immediately respecting the memory constraint in each search without a long sweep of λ values. The way λ is calculated for a given seed is detailed in Sec. 3.4.1.

Furthermore, I add a further loss term $\mathcal{O}$ to model additional complexity-related metrics, which is the term already present in my previous formulation. Differently from $\mathcal{S}$, $\mathcal{O}$ is treated as an objective, not a constraint, and its importance is weighted by μ. μ is the main parameter to tune to generate different final architectures starting from a single seed (as in Sec. 3.2 and Sec. 3.3). To summarize, the formulation of (3.24) will produce DNNs with a size around s^*. With a small μ, the DNAS will focus on minimizing $\mathcal{L}$, producing networks that maximize accuracy (for that size constraint). In contrast, large μs will cause to sacrifice the accuracy in exchange for fewer OPs.

Note that the OPs of a convolutional layer are equal to the parameters multiplied by the output feature map size. Since the feature sizes tend to reduce going forward in the network, OPs reduction under a fixed size budget usually is obtained by masking more channels in the *initial* layers of the DNN, and fewer channels in the *final* ones. I will show that this formulation produces precisely this behavior in the following Sec. 3.3.4.

3.4.1 Weighting Losses

For a given target size s^*, the size strength λ is determined with the formula $\lambda = {\mathcal{L}(\theta_{seed})}/{|\mathcal{S}(\theta_{seed}) - s^*|}$, where $\mathcal{S}(\theta_{seed})$ and $\mathcal{L}(\theta_{seed})$ are the model size and task loss of the full seed network after warmup. The rationale is to have similar values for the first two addends of (3.24) at the beginning of a search so that the DNAS does not just shrink the network in the first iteration, ignoring the impact on accuracy completely. I verified that this heuristic works well, but I also noticed that varying λ in a reasonable range ($\pm$ one order of magnitude) does not alter the search results significantly since the term $\mathcal{S}(\theta) - s^*$ is quickly brought close to zero in the search phase. Importantly, this means that λ can be computed in closed form and does not have to be swept.

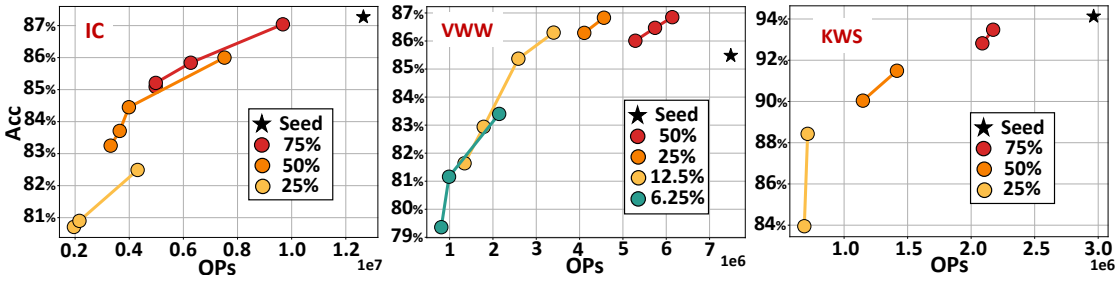

Figure 3.17: Accuracy versus OPs results for different size targets.

3.4.2 Experimental Results

In this section, I show the application of this loss to the PIT algorithm applied only to channels and extended to 2D kernels since I want only to show the benefit of using this newly formulated loss. Noteworthy, it can be applied to all the differentiable NASes, bringing advantages to designers in terms of reduction of searching time and training iterations to find an architecture that fits target hardware.

3.4.2.1 Setup

As benchmarks, I evaluated the proposed loss applied to PIT on three datasets from the previously described MLPerf Tiny Benchmark Suite [77]. As seed networks, I used the same reference architectures shown before. The three considered tasks are Image Classification (IC), Visual Wake Word (VWW), and Keyword Spotting (KWS). I do not consider the last task since the reference DNN is an Autoencoder composed only of Dense layers, for which the model size and number of OPs are directly proportional, leading to no benefits from this new loss formulation.

3.4.2.2 Search-Space Exploration

Fig. 3.17 shows the results obtained by applying the proposed DNAS to the three benchmarks. Each plot reports the found architectures (represented with colored dots) and the seed (denoted with a black star) in the Accuracy versus OPs space. Different colors correspond to different size targets s^*. To validate the approach, I initially set s^* to be respectively 75%, 50%, and 25% of the original size of each seed network. In a real scenario, s^* would depend on the hardware, so this setup simulates targeting three different MCUs with progressively less available memory. Different points are obtained within the curve relative to each s^* target, changing the OPs regularization strength μ.

The leftmost graph shows the results obtained on the IC task. Considering all three memory targets, the DNAS can find networks that span almost one order of magnitude in

OPs 1.96M-9.86M), and ± 6.3% in accuracy. Moreover, under the 75% size constraint, I obtain a network that achieves a negligible accuracy drop compared to the seed (-0.23%), while reducing the number of OPs by 1.3x (12.7M vs. 9.86M).

The center graph reports the Pareto fronts obtained for the VWW task. In this case, I found that the results obtained with the 75% and 50% size constraints are completely outperformed by those obtained with lower memory, which achieves higher accuracy with fewer OPs (only the 50% curve is shown in the graph for clarity). This means that the reference network used for this task is strongly over-parameterized. Therefore, forcing to optimize OPs with a too high s^*, leads to un-balanced architectures which attain the same accuracy as smaller ones (see orange vs. red curves in the mid graph of Fig. 3.17). Thus, I decided to add two additional memory targets (i.e., 12.5% and 6.25% of the seed) to show more insightful trade-offs. The NAS results span almost one order of magnitude in terms of OPs (0.81M-6.14M). Furthermore, many of the found architectures Pareto-dominate the seed, even at 12.5% size (+0.39% accuracy with 2.5× OPs reduction and +0.82% accuracy with 2.2× OPs reduction).

Lastly, the right-most plot in Fig. 3.17 shows the results on the KWS task. In this case, Pareto-fronts are not as rich as for the other two benchmarks due to the peculiarities of the seed network. DS-CNN includes strided convolutions and pooling only in the first and last convolutional layers. Consequently, all intermediate feature map sizes are identical, with OPs and model sizes strongly correlated. Nonetheless, multiple networks for each size constraint are found but with a less favorable trade-off. At most, for the 50% size target, a OPs difference of 1.2× is traded for a small accuracy degradation of 1.45%.

3.4.2.3 Architecture Details

As an example of the architectures found by PIT with this new loss, Fig. 3.18 reports four of the networks generated for the IC benchmark under the 75% and 25% size constraints. The models labeled with "-H" (high-OPs) are obtained with $\mu = 0$, i.e., performing the search with the only constraint of respecting the target size without OPs reduction. Instead, the networks labeled with "-L" (low-OPs) are obtained with $\mu \neq 0$, and in particular, they correspond to the points with the fewest OPs in the 75% and 25% Pareto fronts of the graph in Fig. 3.17. Each rectangle represents a convolutional layer, and numbers inside them correspond to $C^{(n)}_{out,final}/C^{(n)}_{out,seed}$.

These examples demonstrate that this formulation generates meaningful results. First, as expected, a lower target size results in more masked channels, regardless of the OPs regularization strength. Moreover, the "-L" networks have fewer channels in their

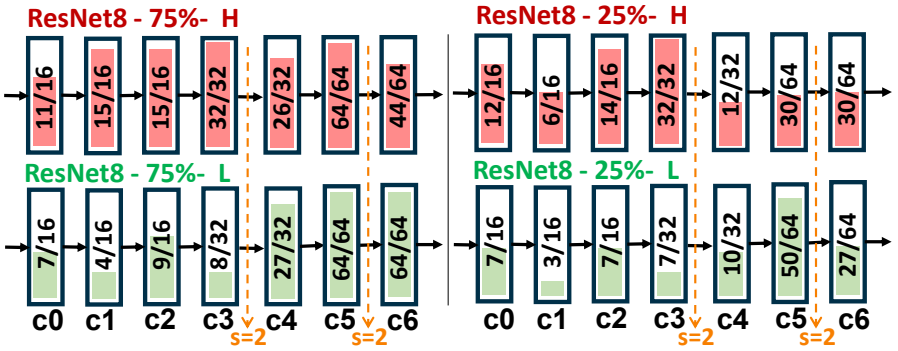

Figure 3.18: Examples of found architectures for the IC benchmark.

initial layers, which contribute more to the total OPs, due to the larger resolution of their input/output feature maps. The considered ResNet8 has two convolutional layers with stride $s = 2$ (c3 and c5), indicated by yellow dashed lines in Fig. 3.18, which reduce the feature map sizes of downstream layers by a factor 4. As evident from the figure, this new loss allows reducing much more aggressively the layers before c3 when μ increases.

Chapter 4

Deployment of Deep Neural Networks on MCUs

4.1 Related Works

After the step of neural architecture search, shown in the previous section, the NN has to be deployed on some hardware platform. In order to optimally deploy them, there are two main challenges that should be faced. First, an appropriate optimized hardware/software library is demanded. Second, the memory hierarchy and the orchestration of the tensors should be tackled to ensure that data needed from the kernels are always in the fastest memory available in the system. In the next three sections, I would describe my holistic tool for DNN and Transformers deployment (DORY) in Sec. 4.2, and two optimized software backend for 1D-CNNs (Sec. 4.3) and Transformers (Sec. 4.4).

Table 4.1: Data flow scheduling and tiling in literature for different computing scales, super computing, ASIC accelerators, and tiny MCUs.

Work	Networks	Optimizations	Output	Open-Source	Precision
Supercomputers					
DMIAYN [87]	Transformers	1) Operator Fusing, 2) Data Layout Exploration	Transformer Primitives	Yes	fp32
DNN Accelerators					
dMazeRunner [88]	CNN, Nested Loops	1) Loop Ordering, 2) Loop Tiling, 3) Memory Movements	Temporal/Spatial Schedule, Loop Tiling	Yes	Flexible
MAESTRO [89]	CNN	1) Mapping & Data Reuse, 2) PEs Design	PEs array, Temporal/Spatial Schedule	Yes	Flexible
Interstellar [90]	CNN, LSTM, MLP	1) Loop Ordering, 2) Loop Tiling, 3) PEs+Mem. Design	PEs + Mem. Array, 7-Loops Ordering and Tiling	Yes	16 bits
Timeloop [91]	CNN	1) Loop Ordering 2) Loop Tiling	Model Scheduling, Latency/Energy Estimation	No	Flexible
Mobile & MCUs					
LCE [92]	BNN	1) Loop Tiling, 2) Vectorization, 3) Parallelization	C++ Runtime Interpreter, C++ Descriptor	Yes	1bit
TFLite Micro [93]	CNN, MLP	1) Hand-configurable Mem., 2) Optimized Backends	C++ Runtime Interpreter, C++ Descriptor	Yes	int8-fp32
Cube-AI [71]	CNN, MLP	1) Mem. Access Opt.	C Optimized Executable	No	int8-fp32
GWT AutoTiler	CNN, MLP	1) Loop Tiling, 2) Mem. Access Opt.	C Optimized Executable	Partially	int8-int16
DORY	CNN, MLP, Transformers	1) Loop Tiling, 2) Mem. Access Opt. 3) Mem. Fragmentation	C Optimized Executable	Yes	int8

4.1.1 Optimized software & ISA for DNN computation

The scope of this "deployment challenge" is to achieve maximal utilization of the computing units, while minimizing the performance and energy penalties associated with data transfers across the memory hierarchy.

A first alternative is to employ hardware-oriented optimization. Application-specific hardware architectures are very useful in accelerating particular layers and, in some cases, entire networks [20–22] – but their lack of flexibility can be a liability in a field such as DL, where every year researchers introduce tens of new topologies and different ways to combine the DNN basic blocks. To provide higher flexibility, in many cases, DNN primitives are implemented in highly optimized software instead of full-hardware blocks. Therefore, several software libraries of DNN kernels have been proposed [2, 25, 33, 94] to maximize the efficiency of DNN execution with DSP-oriented single-instruction multiple-data (SIMD) ISA capabilities [56]. The different software libraries are based on different optimization criteria. For instance, they could optimize data reuse in the spatial dimensions, to be faster on convolutions with larger filters and lower channel connectivity; conversely, they can also optimize channel reuse, often requiring the construction of a flattened data structure ('im2col' buffer) to exploit spatial data reuse partially [33], but increasing the possibility to exploit ISA operations (e.g., ARM Helium) to support and accelerate the pervasive convolutional layers with low-bitwidth linear algebra instructions.

4.1.2 Memory hierarchy management

The other critical challenge in DNN deployment is memory hierarchy management: modern DNNs generate high amounts of weight and activation traffic between different levels of the memory hierarchy, which may constitute a significant bottleneck. In Table 4.1, I report different methods for data flow scheduling and generation that cover three broad classes of devices, namely high-performance computing systems [87], DNN accelerators [88–91], and embedded systems [92, 95]. For what concerns high-performance computing systems, [87] propose new transformer primitives to exploit data reuse and limit data movement by fusing pointwise operators. In this way, all intermediate activations have not to be produced, strongly reducing the memory usage. [88–91] discuss DNN optimization on AI-specialized accelerators based on systolic arrays of processing elements (PEs), with a focus on loop tiling and/or reordering to i) efficiently move the data to fastest memory regions and ii) correctly schedule layers in space and time to maximize PE utilization. The output of these tools can be either an

accelerator model to run a given DNN [89, 90] or the spatial scheduling to maximize PE array utilization on a target accelerator [88, 91].

MCU data flow scheduling tools show similarities to frameworks such as DMazeRunner, as both target the optimization of a dataflow schedule given an externally known architecture. However, the MCU scenario also imposes some additional unique challenges, such as the fact that DNN execution has to be adapted to a general-purpose architecture and the small amount of memory that MCU platforms include. Further, the kernel instructions are heavily influenced by the limited size of the register file, which causes additional load-store operations and thus demand for an optimal loop sizing to avoid register spilling overhead. Major frameworks for DNNs have so far focused either on cloud-scale or specific DNN acceleration, and only recently started to put attention on inference at the edge. Tensorflow Lite (TFLite) from Tensorflow creators is an open-source framework for the deployment of DNN models on top of mobile-class devices and represent a first step forward in this direction. A Tensorflow DNN model is converted into a compressed flat buffer and interpreted on-device. Similarly, Larq Compute Engine (LCE) [92] is a framework targeting the deployment of heavily quantized neural networks on edge mobile devices. LCE leverages optimization techniques such as tiling, vectorization, and multi-threading parallelization to speed-up the execution of inference tasks. Both TFLite and LCE are not suitable for IoT MCU-class devices for two reasons: they require the target device to be capable of booting a full-fledged operating system, and they are not tuned to target low-cost low-power MCUs with less than 1MB of on-chip SRAM and just a few MB of off-chip memory.

4.1.3 DNN-oriented microcontrollers and related tools

Recently, the first generation of low-power neural-network oriented MCUs has been introduced (Sec. 2.3.1 and Sec. 2.3.2). These platforms show an increased complexity in terms of memory hierarchy compared to conventional flat-memory MCUs, with an L1 memory optimized for speed and an L2 optimized for capacity. Programming these DNN-oriented MCUs is typically more complicated with respect to conventional MCUs. Maximizing the exploitation of computational resources is challenging, and scratchpads require manually managed data orchestration and tiling.

To the best of my knowledge, the two most powerful DNN deployment tools available in the state-of-the-art have been proposed by the industry as proprietary, vendor-locked solutions for their own MCUs. X-CUBE-AI [71] from STMicroelectronics is an automatic NN library generator optimized on computation and memory. It converts a pre-trained DNN model from DNN tools such as Tensorflow into a precompiled library for the

ARM Cortex-M cores embedded in STM32 series MCUs. However all STM32 MCUs supported by X-CUBE-AI have a cache. Therefore, it can be argued that they do not directly tackle the memory management problem, relying on extended L1 cache (up to 16 kB) to maximize the performance. On the other hand, GWT designed a tool called AutoTiler, to target the GAP-8 RISC-V based multi-core ultra-low-power microcontroller. The AutoTiler has a wider scope than other strictly AI-focused tools, also targeting the deployment of traditional vision and signal processing algorithms. One of its primary functions is to take a pre-trained DNN and generate code for memory tiling and efficient transfers of weight and activation data between all memory levels (on- and off-chip). The GWT AutoTiler directly tackles the data-movement and tile sizing challenge to optimize memory access, reaching state-of-the-art performance on the execution of many networks. The tool is proprietary, but its backend basic kernels are available as open-source as part of the GAP-8 SDK[1].

The framework that I will introduce in the next Section, DORY, is the first open-source framework to directly tackle the MCU memory hierarchy management challenge, with a comprehensive exploration of data tiling, optimized loop ordering for different layers (i.e., pointwise and depthwise), and a solution for the data fragmentation problem that is critical to deploy residual layers at the edge. DORY consistently outperforms all the illustrated frameworks on all the proposed benchmarks.

[1] https://github.com/GreenWaves-Technologies/gap_sdk

Table 4.2: Symbols used throughout this work.

Input **x** dims (height/width/chan)	h_x / w_x / C_x
Output **y** dims (height/width/chan)	h_y / w_y / C_y
Weight **w** dims (out c/height/width/in c)	C_y / K_h / K_w / C_x
Buffer for tensor q at i-th level of mem. hier.	Li_q
Tiled dimension d_q of a tensor **q**	d_q^t

4.2 DORY: Automatic End-to-End Deployment of Real-World DNNs on Low-Cost IoT MCUs

In this section, I will describe DORY, my tool for the deployment of neural networks on MCUs. The explanation of DORY will be made on top of a node with three memory levels, e.g., the GAP8 platform introduced, and taking into account the presence of an optimized neural network kernel library. In the following section, we will use the general PULP-NN library [96] to explain the general concepts. However, note that DORY is equally able to manage the two specialized libraries PULP-NN-1D and TinyFormers, exposed in the two sections after. Table 4.2 also introduce the notation used throughout this chapter. DORY supports L3-L2 and L2-L1 tiling of both weights and activations. Storage of weights in L3 ($>$ 512 kB) is essential for the deployment of most non-trivial networks such as [47, 83]. On the other hand, activations' tiling is typically necessary only for networks working on high-resolution images with big spatial dimensions, which are rare in the edge computing domain. In the following, I will dig inside the three main offline steps before network deployment.

ONNX decoder: it receives as input a QNN graph using the Open Neural Network Exchange (ONNX format) and parses it to create a DORY-compatible graph.

Layer analyzer: it optimizes and generates code to run the tiling loop, orchestrate layer-wise data movement and call a set of *backend* APIs to execute each layer of the network individually.

Network parser: it merges information from the whole network to infer memory buffer sizes in each hierarchical level and orchestrate the end-to-end network execution. It uses this information to generate an ANSI C file that embodies the whole DNN execution and can be compiled for the target platform.

4.2.1 ONNX Decoder

In the first operation, DORY decodes the input ONNX graph representing an already quantized DNN and reorganizes it into a set of layers. In DORY, a *layer* corresponds to a canonical sequence of operations performed by distinct ONNX graph nodes. Each

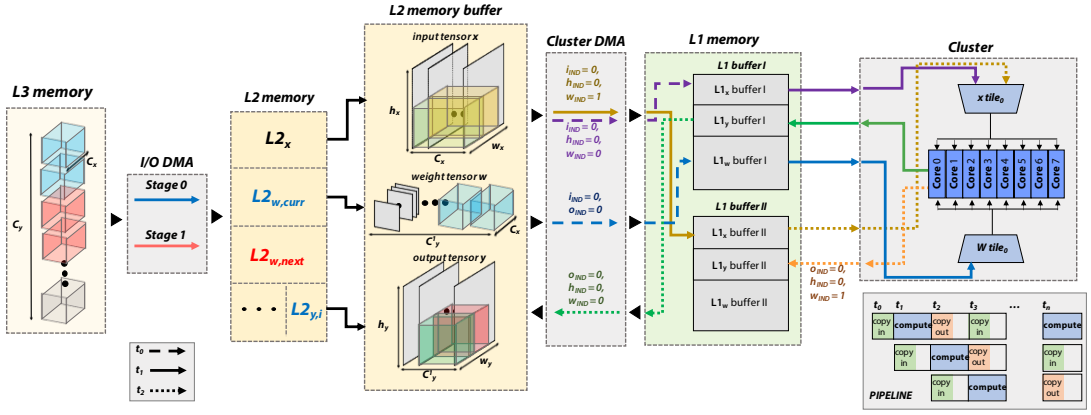

Figure 4.1: DORY L3-L2-L1 layer routine example. On the left, the I/O DMA copies weights tile in case only C_y is L3-tiled. Two different buffers are used for $L2_w$. Then, the Cluster DMA manages L2-L1 communication using double-buffering while the cores compute a kernel on the current tile stored in one of the L1 buffers.

```
1   LTO: for (o = 0; o < C_y^t; o++)
2     LTH: for (h = 0; h < h_y^t; h  ++)
3       LTW: for (w  = 0; w < w_y^t; w ++)
4         LTI: for (i = 0; i < C_x^t; i ++)
5           dma_wait(L1_{x,load});  swap(L1_{x,load}, L1_{x,exec})
6           dma_async(L1_{x,load} <- L2_x[i, w, h])
7           dma_wait(L1_{w,load});  swap(L1_{w,load}, L1_{w,exec})
8           dma_async(L1_{w,load} <- L2_w[i, o])
9           if (o  + h + w  + i > 0)
10                DNN_kernel (L1_{x,exec}, L1_{w,exec}, L1_{y,exec})
11  # from 3° iteration: fully operating pipeline
12          if (o  + h  + w  + i > 1)
13            dma_wait(L1_{y,load})
14            dma_async(L1_{y,load  -> L2_y[ o, w, h])
15            swap(L1_{y,load , L1_{y,exec})
```

Listing 4.1: DORY L2-L1 loop nest implementing the double buffering scheme as represented in right part of Figure 4.1. At each most internal loop iteration, two asynchronous Cluster DMA calls are made to copy the weights and input activation of the next tile into L1 memory, the basic kernel is executed on the current tile, and one other cluster DMA transfer is executed to copy the output back on the L2 memory.

layer includes *i*) a Linear/add/pooling operation, *ii*) an optional Batch-Normalization operation, *iii*) a Quantization/Activation operation. Each DORY layer uses quantized to 8bits inputs, outputs, and weights, while the representation of any temporary data is made on 32-bit data arrays.

4.2.2 Layer Analyzer

After recognizing every single layer, in the first optimization phase, DORY layers are considered separately from each other. Furthermore, partial information from the

previous layer is employed. The layer analyzer includes three submodules: a platform-agnostic *tiling solver*; a set of *heuristics & constraints* optimizing execution over a target-specific backend and limiting the tiling search space; and a *SW-cache generator*.

4.2.2.1 DORY Tiling Solver

In the following discussion, I denote a buffer residing in Li memory as $Li_{\mathbf{t}}$, where $\mathbf{t}$ is the name of the tensor. The Solver relies on a 2-step engine, which solves the L3-L2 tiling constrained problem first, and the L2-L1 one after. These two steps are generic for architecture with three memory levels, as the GAP8 SoC described in Sec. 2.3.2. On the other hand, user can also enable only one step, supporting different architecture topologies. With L3-L2 tiling, I enable storing activations and weights in the L3 off-chip memory instead of the on-chip L2, if available. Compared to tools that do not support L3 tiling for activations, such as Tensorflow Lite Micro, this feature enables the support of significantly larger layers.

In this first step, the Solver verifies whether the layer memory occupation fits the L2 memory input constraint or needs to be stored in L3:

$$L2_{w,next} + L2_{w,curr} + L2_x + L2_y \overset{?}{<} L2 \ . \tag{4.1}$$

If it does not fit in L2 and the L3 memory is available, DORY searches for a proper tiling solution using a five-stage cascaded procedure. At each stage, I try to tile a different selection of buffers to fit the constraint of Eq. 4.1. If possible, the tiler tries to avoid L3-L2 tiling of output activations since it requires a double number of transfers compared to other tensors, i.e., L3 to L2, and vice-versa. Instead, the tiler tries to keep output activations in L2 as much as possible. Each successive stage is more restrictive; therefore, the L3-L2 Tiling Solver is stopped as soon as the constraint is respected to avoid any additional performance loss.

stage 0. `L3-tile x, w, y = OFF, OFF, OFF`. If Eq. 4.1 is directly satisfied, no L3 is required.

stage 1. `L3-tile x = ON`. This solution is selected when the previous layer's output was tiled in L3, so input tiling cannot be avoided. Tiling is performed along the h_x dimension of the input to avoid 2D transfers at the L3-L2 interface. The tiler splits the layer into a series of identical ones that work on a different stripe of the input image.

stage 2. `L3-tile w = ON`. Weight tiling is enabled on the C_y dimension, dividing the layer into a set of smaller layers that work on different channels of the output

image with $C'_y < C_y$. This solution can only be selected when the previous layer's output is already in L2.

stage 3. `L3-tile w , y = OFF, ON`. Weight tiling is disabled while output tiling is enabled: the approach is similar to input tiling but requires doubling the DMA transfers for the tiled tensor across the full network execution.

stage 4. `L3-tile w, y = ON, ON`. The L3 tiling is enabled on both buffers, y, *weights*. This solution is selected when no other solution can fit L2.

After the L3 tiling step, I have either a series of identical smaller nodes that fit L2 or the initial node that fits L2. Then, the DORY solver processes the layer to find a suitable L2-L1 tiling scheme, which requires more effort due to the typically small sizes of L1 memories. Compared to high-end computation engines, with much larger memories, a suboptimal sizing of the tensors for the L1 small MCUs memory can be even more detrimental in terms of performance, as exposed in Section 4.2.6.1. DORY abstracts this as a Constraint Programming (CP) problem, exploiting the CP solver from the open-source Google AI OR-Tools [2] to meet hardware and geometrical constraint (e.g., C^t_y for output and weights must be the same) while maximizing an objective function. The base objective function of the Solver is to maximize L1 memory utilization:

$$\max(L1_x + L1_y + L1_w) \,, \tag{4.2}$$

manipulating the tile dimensions (e.g., C^t_x and C^t_y). The hardware constraint is related to the max L1 buffer dimensions:

$$L1_x + L1_y + L1_w + L1_{backend} < \frac{L1}{2} \,.$$

with $L1_{backend}$, the overhead of the backend kernel, such as the im2col memory occupation of PULP-NN backend [25] or any other support buffer (e.g., the intermediate full-precision accumulators for CHW-based convolutions or the intermediate activations from the sub-nodes of attention layers). Topological and geometrical constraints are due to the relationships between each tensor's characteristic dimensions and other parameters of a layer; for instance, I show the relationship between the input and the output dimensions of an image,

$$h^t_y = \Big(h^t_x - (K_h - 1) + 2 \cdot p\Big).$$

[2] https://developers.google.com/optimization/

4.2.2.2 GAP8-specific Heuristics & Constraints

In this paragraph, I will show how to augment the objective function of Eq. 4.2 with a series of heuristics targeting a specific backend to maximize performance. The heuristics are combined with the objective function of Eq. 4.2 using a set of tweakable parameters:

$$\max\Big(\alpha(L1_x + L1_y + L1_w) + \sum_i \beta_i \mathcal{H}_i\Big) \ . \tag{4.3}$$

Here, I list four heuristics related to a specific combination, i.e., PULP-NN as the backend and GAP8 as the target platform. Note that a different kernel library or a different backend can dramatically change all the heuristics.

- `HIDE_IM2COL`: the PULP-NN library exploits the im2col transformation to re-order data in L1. This buffer is reused for each output pixel; therefore, maximizing the number of output channels optimizes the reuse of input pixels, reducing the overhead to create the im2col:

$$\mathcal{H}_{i2c} = C_y^t$$

- `PAR_BALANCE`[3]: PULP-NN primarily divides workload among cores following the h dimension (i.e., a chunk of rows per core). Therefore, I make it a multiple of cores number (8), to maximizes balance:

$$\mathcal{H}_{par} = (h_y^t - 1) \bmod 8$$

- `MATMUL_W` and `MATMUL_CH`: the innermost loop of PULP-NN is a 4x2 matrix multiplication on 4 output channels and 2 pixels in w direction. Maximizing adherence of a tile to this scheme optimizes performance:

$$\mathcal{H}_{mm\_w} = (w_y^t - 1) \bmod 2 \ ; \ \mathcal{H}_{mm\_ch} = (C_y^t - 1) \bmod 4$$

I will discuss the utilization of these heuristics in Section 4.2.6.1. Additionally, Section 4.2.6.1 describes the impact of applying these heuristics both to the main tiling problem and to the sizing of the layer borders tile.

Finally, I impose one last but vital constraint to enforce an entire computation along the input channel direction:

$$C_x^t = C_x$$

I choose not to tile the C_x dimension to avoid the memory overhead of long-term storage (and, therefore, transfer to $L2$ and $L3$) of 32-bit partially accumulated values produced

[3]The `PAR_BALANCE` constraint is changed to $\mathcal{H}_{par} = (h_y^t \times w_y^t - 1) \bmod 16$ for "pathological" output activations with $h_y < 8$.

by the backend. For the same reason, I do not tile the spatial dimension of filters, i.e., K_h and K_w.

4.2.2.3 DORY SW-cache Generator

The SW-cache Generator automatically generates C code orchestrating the execution of a whole layer given the tiling solution found by the Tiling Solver. It instantiates asynchronous data transfers and calls to the backend kernels without any manual effort. DORY uses a *triple-buffering* approach for the communication between L3-L2 and L2-L1 memories and a *double-buffering* approach when only two levels of memory are available. For triple-buffering, double-buffering is applied simultaneously between L3-L2 and L2-L1 (Figure 4.1), and all data transfers are pipelined and asynchronous. I can almost completely hide the memory transfer overhead with these approaches, as discussed in Section 4.2.5. While the code generator is not platform-agnostic (it uses SIMD specific of GAP8 in this example and should be customized for each hardware platform), the approach I follow can be easily generalized to any computing node with a three-level memory hierarchy.

Listing 4.1 provides DORY's scheduling scheme of L2-L1 layer execution through LTO, LTW, LTH, and LTI loops on output channels, height, width, and input channels tiles, respectively. Loop iteration limits are statically resolved by the *DORY tiling Solver*. Moreover, DORY autonomously controls the complete execution of the layer by managing padding, stride, and overlap for every single tile (e.g., padding > 0 for border tiles whereas padding $= 0$ for internal ones, when the input padding parameter is > 0). Note that using statically resolved parameters maximizes immediate usage, minimizing loads/stores from the stack.

The layer-wise loop nest detailed in Listing 4.1 and Fig. 4.1 is executed in three concurrent pipeline stages: *i*) a new computation starts and fill the output buffer that was not used in the previous cycle; *ii*) the results of the last cycle are stored back in L2; *iii*) a new set of inputs is loaded in L1. At each pipeline cycle, I swap the load and the execution buffer (*swap* operation of Listing 4.1) to enable double buffering.

4.2.3 DORY Hybrid Model

In the HWC data layout, used by CMSIS-NN [33] and PULP-NN [25], pixels referring to channels are contiguous, while spatially adjacent ones are stored with stride > 1. This layout enables constructing very optimized convolutional layers out of a single optimized matrix-multiplication kernel by exploiting the reuse of activations over input channels

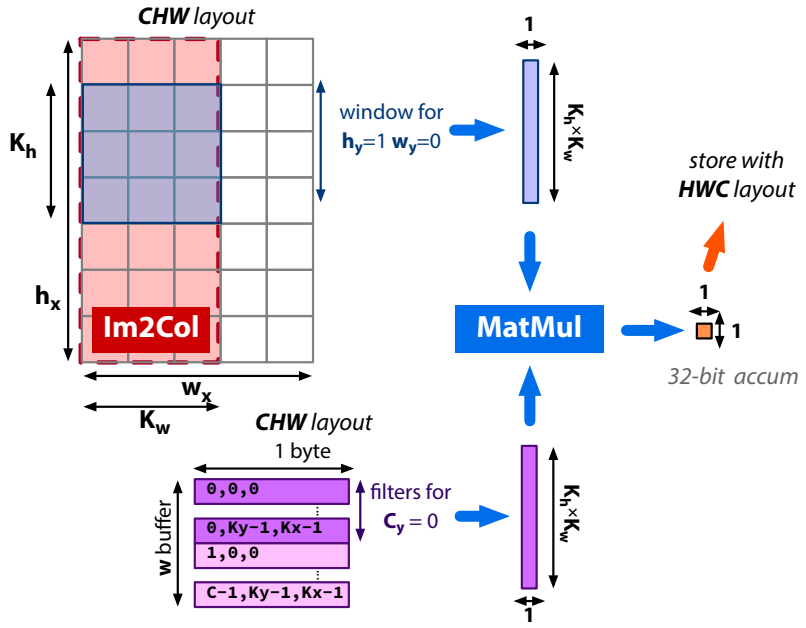

Figure 4.2: Modified execution model for depthwise convolutions: the Im2Col buffer is built using a single channel out of CHW-layout activations; outputs are quantized and stored back using the PULP-NN model.

[25, 33]. In contrast, the CHW layout requires separately handcrafted and optimized kernels for each kernel size/stride configuration. The principal limitation of this approach is the efficient execution of depth-wise convolutions. These do not accumulate over multiple channels; instead, they project each input channel into a single output channel disjointly from other channels, not showing any possibility to exploit channel data reuse.

On the one hand, depth-wise convolutions are the core of many significant "efficient" networks. They are used to reduce the number of operations and the memory occupation while maintaining a high accuracy [47]; on the other hand, they are typically only responsible for 10% or less of the overall operations [47, 83], meaning that directly optimizing for them may be suboptimal. Therefore, I keep the HWC layout for general convolutional layers (and point-wise 1x1 layers). Still, I try to apply a hybrid CHW/HWC layout in depth-wise layers to optimize their performance.

Following this idea, I define new optimizations for existing layers and a new depthwise convolution that consumes and produces activations in HWC layout from L2/L3 memory but reorders them in CHW layout on L1 to maximize the data reuse and, therefore, computational efficiency.

Specifically, multiple strided Cluster DMA transfers are used to marshal data from L2, converting it directly from the HWC to CHW layout, reducing data-reordering overhead. In the new depthwise layer, an Im2Col buffer is constructed simply as a contiguous

```
udma_async(L2_{w,load} <- L3_w[I_0])
udma_wait(L2_{w,load});
LTL: for (i = 0; i < nlayers; i ++)
# number of CNN layers
  udma_wait(L2_{w,load}); swap(L2_{w,load}, L2_{w,exec})
  if  (layer{i+1} fit L2 && is Conv)
      udma_async(L2_{w,load} <- L3_w[I_i])
  Layer{i} (L2_x,[L2_{x2}], [L3_w[I_i]], [L2_{w,exec}], L2_y)
  # [] optional arguments
      swap(L2_y, L2_x)
  if  (layer{i} has residual) # bypass management
      store  (L2_y->L2_{x2})
  if  (layer{i} is Sum)
      delete  (L2_x2)
      Stack_dealloc(L2_y) # stack control
      Stack_alloc(L2_x[I_{i+1}])
```

Listing 4.2: DORY network execution loop.

vertical stripe of width K_w; the innermost loop proceeds along the vertical line by computing a single output pixel per iteration. The output pixels are then quantized and stored in an output buffer using the HWC layout, which can be directly transferred to L2. Figure 4.2 shows the execution model adopted for depthwise convolutions. With this strategy, input data reuse – the only kind available in depth-wise convolutions – can be exploited along the vertical dimension, thanks to the fact that spatially adjacent pixels are contiguous in memory. For parallel execution, multiple cores operate simultaneously on different channels. Due to channel independence, this choice minimizes memory contention and optimizes parallelization performance: the same kernel can be used to compute depth-wise layers of various filter shapes and strides.

4.2.4 Network Parser

After the Layer Analyzer has completed layer-wise tiling, DORY uses the information extracted from all the layers to build a network graph, considering every single layer as a sub-function to be called from it. Listing 4.2 showcases the execution loop of the DNN execution as created by the DORY framework. At each step, three main tasks are concatenated: *i*) I transfer from L3 the weights of the layer. *ii*) a new layer is executed after that all data are in the correct memory level (either L3 or L2); *iii*) input and output buffer offsets are updated after the layer execution.

Similarly to single layers, DORY generates the network-wise code automatically without programmer intervention.

4.2.4.1 Buffer allocation stack & Residual connections

To allocate layer-wise input and output buffers in the L2 memory, I extend the two-stack strategy proposed by Palossi et al. [97], employing an approach based on a single bidirectional stack designed to avoid memory fragmentation and enable the execution of a sequence of differently sized layers. Buffers are allocated/deallocated from the buffer allocation stack. Unlike a classical stack, the buffers can be distributed on both sides. By construction, the bidirectional stack is always smaller than two concurrent stacks growing in the same direction. For example, in a simple case without residual connections, the dimension of this new *bidirectional stack* is

$$D_{stack} = \max_i(L2_{x,i} + L2_{w,i} + L2_{w,i+1} + L2_{x,i+1}) \ ,$$

which is always less or equal than the size of two concurrent stacks $D_{stack,1}$, $D_{stack,2}$ due to the triangle inequality.

Before executing the i-th layer, the allocator manages the weight buffer $L2_{w,i}$ and output buffer $L2_{y,i}$; notice that $L2_{x,i}$ is already allocated as the $L2_{y,j}$ of a previously executed j-th layer (or the input of the network). A special case is associated to residual connections: I associate each $L2_{y,i}$ buffer to different `lifetime` counters. To allocate a buffer in the stack for the i-th layer:

1. one of the two corners of the stack is selected depending on a `begin_end` flag that is switched at each new weight allocation;
2. the allocator deallocates the last $L2_{w,i-2}$ buffer on the corner;
3. the allocator checks if $L2_{y,i-2}$ has its `lifetime` counter set to 0; if so, it is deallocated; note that by constructions, these will always be the most internal buffers of the stack and therefore can be safely deallocated.
4. $L2_{y,i}$, $L2_{w,i}$ are allocated in order in the selected corner (with $L2_{w,i}$ nearest to the pointer);
5. the `lifetime` counter of $L2_{y,i}$ is set to the lifetime of the activation buffer, i.e., the number of layers to be executed before its deallocation.
6. all `lifetime` counters are decreased by 1.

This newly designed buffer allocation stack is naturally suited to execute a network with different branches (i.e., residual connections), solving the memory fragmentation problem produced by allocating and deallocating successive layers.

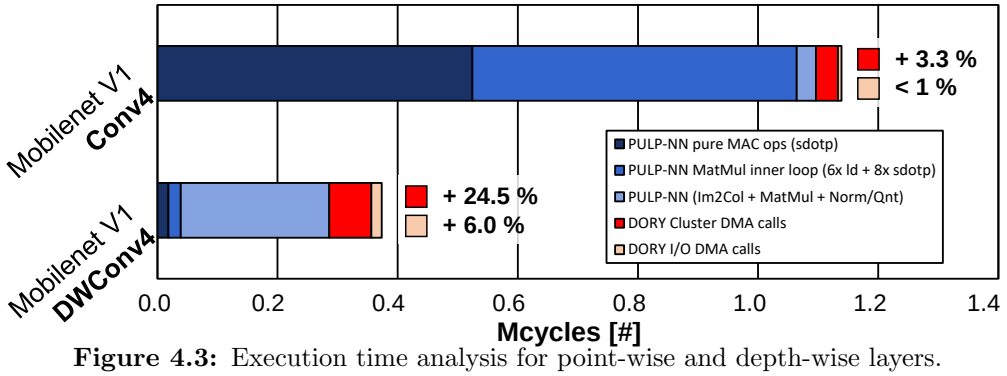

Figure 4.3: Execution time analysis for point-wise and depth-wise layers.

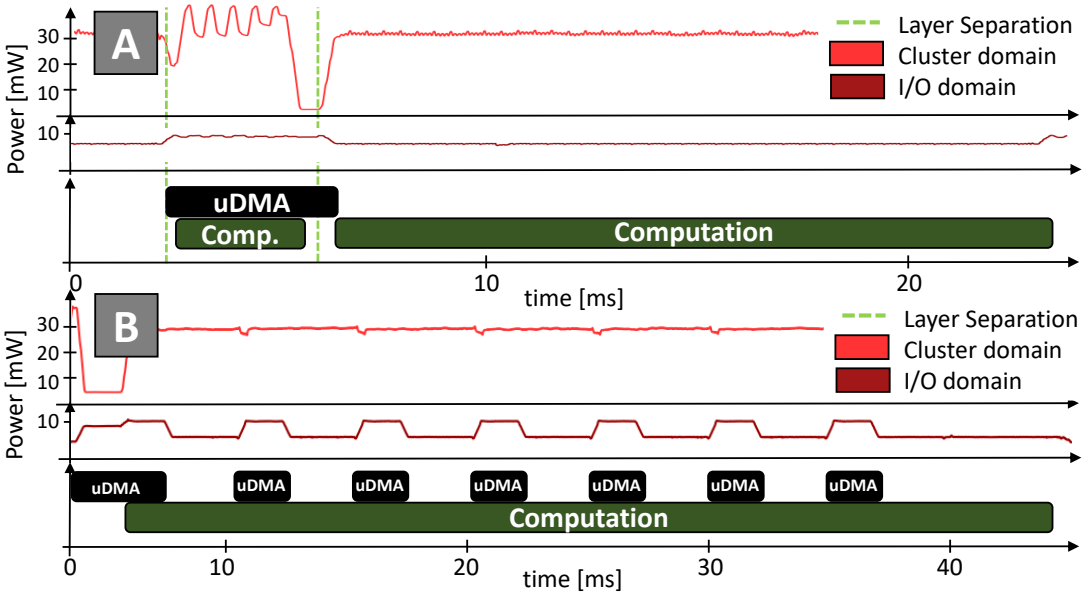

Figure 4.4: In Part. A, the power traces of a point-wise Convolution following a depth-wise one. The I/O DMA causes the CORES to go in IDLE, waiting for the memory transfer to end. In Part. B, an L3-tiled layer is executed and perfectly buffered to hide the memory hierarchy to the computing engine. fr = 100 MHz and V_{DD} = 1V have been used on the GAP8 MCU.

4.2.5 Results

In this Section, I evaluate DORY in terms of energy efficiency and latency on both single layers and complete networks, using GWT GAP-8 as a target platform. I also compare the shown results with those obtained on an STM32-H743 MCU using STM X-CUBE-AI and on the same GAP-8 platform using the proprietary AutoTiler tool. The results on single layers refer to a whole 8-bit QNN layer, with Linear, Batch-Normalization, and Quantization/Activation sub-layers. I set α to 0.5, $\beta_{\texttt{HIDE\_IM2COL}}$ to 10^2, and other β_i to 10^6 in the objective function.

Table 4.3: Average performance and efficiency on 8-bits MobileNet-V1 layers obtained with DORY and other SoA MCU-deployment frameworks.

		Performance (speed-up)		**Efficiency**
		MAC/cycle	GMAC/s	GMAC/s/W
TFLite$_a$ Micro	DwConv	0.064 (0.2×)	0.03 (0.2×)	0.13 (0.2×)
	PwConv	0.056 (0.1×)	0.027 (0.1×)	0.11 (0.1×)
STM[a] CUBE-AI	DwConv	0.39 (1×)	0.19 (1×)	0.8 (1×)
	PwConv	0.71 (1×)	0.34 (1×)	1.46 (1×)
GWT[b] AutoTiler	DwConv	2.16 (5.5×)	0.22 (1.2×)	4.24 (5.3×)
	PwConv	7.87 (11.1×)	0.79 (2.3×)	15.4 (10.6×)
GWT[c] AutoTiler	DwConv	2.16 (5.5×)	0.56 (3.0×)	2.16 (2.7×)
	PwConv	7.87 (11.1×)	2.05 (6.0×)	7.87 (5.4×)
DORY[b]	DwConv	1.14 (2.9×)	0.11 (0.6×)	2.24 (2.8×)
	PwConv	12.86 (18.1×)	1.29 (3.8×)	25.2 (17.3×)
DORY[c]	DwConv	1.14 (2.9×)	0.30 (1.6×)	1.14 (1.4×)
	PwConv	12.86 (18.1×)	3.34 (9.8×)	12.86 (8.8×)

[a] Collected on the STM32H743 @ 480MHz.
[b] Collected on the GWT GAP8 @ (100MHz, 1V).
[c] Collected on the GWT GAP8 @ (260MHz, 1.15V).

4.2.5.1 Single layer performance & SoA comparison

In this Section, I analyze the impact of the DORY optimizer on the execution of an entire layer, including the unavoidable processing overhead to perform I/O DMA and Cluster DMA calls and the data transfer overhead from imperfect pipelining. Fig. 4.3 analyzes all the execution time for a point-wise convolutional layer and depth-wise ones. I will describe several effects. For the point-wise layer, roughly all the time is spent in the innermost loop of MatMul (most of which is pure MAC operations). The rest of the time is due to building the Im2Col buffer, Norm/Qnt, and MatMul loops that cover the SIMD leftover cases (e.g., C_y^t not multiple of vector size 4). In the case of depth-wise layers, this latter class of loops dominates the backend execution time. Regarding the overhead introduced by DORY-generated tiling, the readers can observe that the Cluster DMA does not impair the point-wise convolutional layers since they are compute-bound and efficiently pipelined. Therefore, the cycles needed to transfer the data are totally overlapped with computing cycles. On the other hand, depth-wise layers are small, and the Cluster DMA and I/O DMA overheads are exacerbated. Therefore, the load of the internal tiles and the asynchronous I/O DMA load of the following layer's weights often impact performance. Fig. 4.4 corroborates this conclusion, showing power valleys in the cluster computation while waiting for new tile transfers, i.e., meaning that the cluster is not doing any computation since the transfer time is higher than the computation time.

In Table 4.3, I also compare DORY with three state-of-the-art frameworks for DNN

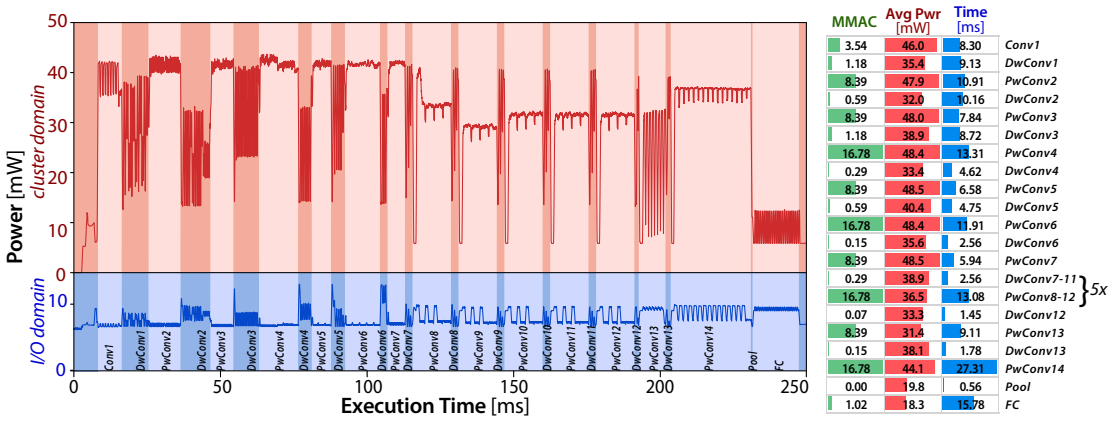

Figure 4.5: In the left part, the 1.0-MobileNet-128 power profile when running on GAP-8 @ $f_{\text{cluster}} = f_{\text{io}} = 100MHz$ and $V_{DD} = 1V$. On the right are MAC operations, average power, and time for each network layer. power was sampled at 64 KHz and then filtered with a moving average of 300 us.

deployment on MCU: TFLite Micro, STM X-CUBE-AI, and GWT AutoTiler. I used convolutional and depth-wise convolutional layers as benchmarks since they constitute the vast majority of computation in modern DNN models. Results obtained on TFLite Micro and STM X-CUBE-AI refer to the STM32H743 microcontroller, while GWT AutoTiler and DORY ones to GWT GAP-8, described in Section 2.3.2. All results refer to 8-bit quantized networks, even if STM32 also supports 32-bit floating point; accuracy is equivalent to that of a non-quantized network.

TFLite Micro has the main advantage of being available on many different ARM and RISC-V MCUs; on the other hand, its performance is severely limited because it uses very general APIs without deep optimizations. State-of-the-art for ARM platforms is indeed X-CUBE-AI, which outperforms it by 6.1× to 12.7×. Nonetheless, layers generated by DORY outperform TFLite Micro and X-CUBE-AI by a margin of 2.9× to 229.6× in terms of MAC/cycle. This significant advantage is due to the architectural benefits of GAP-8 (multi-core acceleration, DSP-enhanced instructions) that DORY can exploit fully, but also to the perfect overlap of memory transfers and computation. In Section 4.2.6.5, I will decouple DORY performance enhancement and architectural benefits to underline the benefits of the DORY framework, deploying layers with DORY both on the STM32H7 and on GAP8 forced to run with a single-core.

When I also compare DORY to GWT AutoTiler in GAP8, DORY is 1.6× faster in point-wise convolutions, while it pays a performance toll in depth-wise convolutions, being 1.9× slower. These differences amount mainly to the different strategies followed by the tools in their respective backends.

4.2.5.2 End-to-end network performance

In this Section, I focus on the performance of DORY in the deployment of two popular image detection networks end-to-end: MobileNet-v1 [83] and MobileNet-v2 [47]. These networks are used as benchmarks for many edge-oriented works [2]. They include many topological characteristics of modern networks: convolution, depth-wise convolution, pooling, fully-connected layers, and residual connections. Here, I focus on the specific configurations with 1.0 width multiplier and 128x128 input frames (1.0-MobileNet-128 and 1.0-MobileNetV2-128, respectively). All the networks were run on GWT GAP-8, verifying all intermediate results as well as the final result of end-to-end runs against a PyTorch-based bit-accurate golden model for QNNs [98], to confirm the correct functionality of the DORY framework and the PULP-NN backend.

4.2.5.3 End-to-end MobileNet-v1 and -v2 & SoA comparison

Table 4.4 showcases a full comparison in terms of energy efficiency (GMAC/s/W), throughput (GMAC/s), latency, and energy per frame. Different variations of the MobileNet-v1 have been compared, with the same topology but a different number of channels or input dimensions. For state-of-the-art, I show the most extensive networks that fit the on-chip/off-chip memory of the STM32H7 and GAP8, respectively (compatible with the ones deployed with DORY). As can be noticed from the Table, DORY on MobileNet-v1 achieves up to 13.19× higher throughput in MAC/cycles than the execution on an STM32H7 (on 0.5-M.V1-192), using the best framework (X-CUBE-AI) currently available. On different operating points, I achieved up to 7.1× throughput (1.78 vs. 0.25 GMAC/s) and 12.6× better energy efficiency, given the different frequencies and power consumption of the two platforms. I want to stress that since the technique that DORY uses is not dependent on the specific target, users can extend it to different platforms (e.g., NXP and STM32 dual-core M0/M4). To do it, the user should adapt the tiling and correct the offloading to improve the cache friendliness of DNN primitives and exploit optimized ISAs. For instance, in Section 4.2.6.5 I show DORY application also to the STM32H7 platform. Compared with GWT-proprietary and partially closed-source AutoTiler run on the same GAP-8 platform, DORY performs on average 20.5% better. As previously discussed, the advantage lies in *1)* the more efficient backend (PULP-NN) and *2)* the heuristics, which guarantee that the tiling solution is optimized for the PULP-NN execution model.

4.2.5.4 In-depth analysis of MobileNet-v1 execution

Fig. 4.5 depicts the power profile of the end-to-end execution of a MobileNet-v1 (1.0 width multiplier, 128 × 128 resolution) on GAP-8, with both the cluster and the fabric controller running at 100 MHz. The power consumption of the cluster domain (including 8 RI5CY cores, the L1, and the Cluster DMA) and the I/O domain (including 1 RI5CY core, the L2, and the I/O DMA) is shown separately in two separate subplots. In the cluster domain, power is dominated by the cores when the computation is in the active phase. Small valleys within a layer are given by (short) waits for the end of a memory transfer where the cores are all idle (e.g., during depthwise layer execution), or by cluster DMA calls where a single core is active. In the I/O domain, DMA consumption spikes can be observed: at the beginning of layers, the weights of the following one are transferred from L3 to L2.

Table 4.4: End-to-end execution of image recognition MobileNet-v1 and MobileNet-v2 on GAP8 and STM32H7 MCUs.

Configuration	**Params**	**Work** MAC	**Cycles**	**Perf** MAC/cyc	**Eff** GMAC/s/W	**Perf** GMAC/s	**Lat** lat. [ms]	**Energy** E [mJ]	**Eff** GMAC/s/W	**Perf** GMAC/s	**Lat** lat. [ms]	**Energy** E [mJ]
DORY @ GAP8												
					Low energy 1V @ 100 MHz				Low latency 1.15V @ 260 MHz			
1.0-M.V1-128	4.2 M	186.4 M	23.3 M	8.00	15.68	0.80	233.11	11.89	7.93	2.08	89.66	23.51
0.5-M.V1-192	1.3 M	110.0 M	16.0 M	6.86	13.46	0.69	160.2	8.17	6.82	1.78	61.62	16.16
0.25-M.V1-128	0.5 M	13.5 M	2.8 M	4.74	9.30	0.47	28.50	1.45	4.69	1.23	10.95	2.87
1.0-M.V2-128	3.47 M	100.1 M	19.0 M	5.27	10.33	0.53	190.03	9.69	5.22	1.37	73.09	19.16
GWT AutoTiler @ GAP8												
					Low energy 1V @ 100 MHz				Low latency 1.15V @ 260 MHz			
1.0-M.V1-128	4.2 M	186.4 M	28.1 M	6.64	13.02	0.66	280.80	14.32	6.58	1.73	108.00	28.32
1.0-M.V2-128	3.47 M	100.1 M	19.7 M	5.07	9.95	0.51	197.38	10.07	5.03	1.32	75.92	19.91
X-CUBE-AI @ STM32H7, solutions fitting 2MB ROM + 512 kB R/W RAM @ 480 MHz [2]												
0.25-M.V1-128	0.5 M	13.5 M	26.0 M	0.52	1.07	0.25	51.14	12.67	n.a.	n.a.	n.a.	n.a.
0.5-M.V1-192	1.37 M	109.5 M	212.3 M	0.52	1.06	0.25	442.27	103.49	n.a.	n.a.	n.a.	n.a.

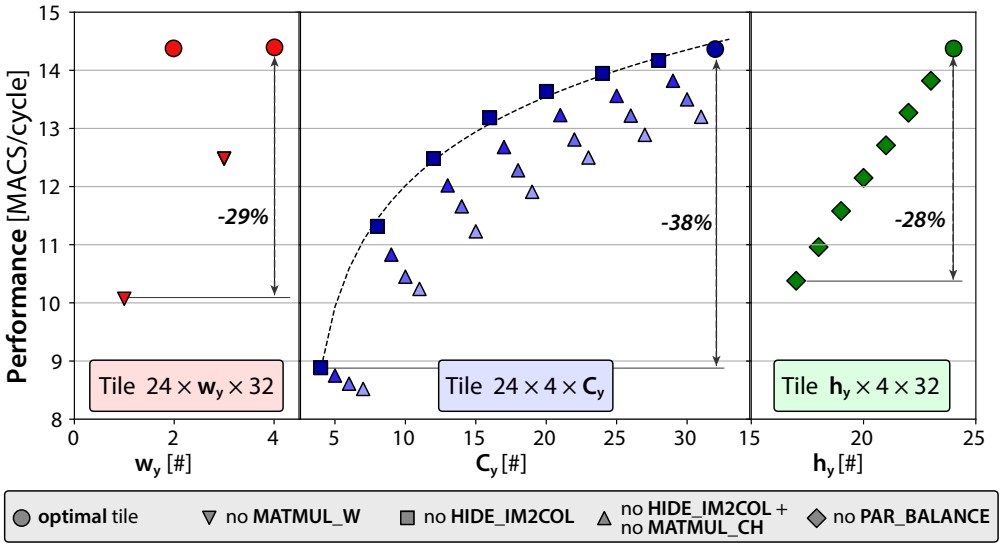

Figure 4.6: example of the effect of heuristic optimizations on convolutional layer performance. In this case, the "optimal" tile has output tensor 24×4×32 (HWC) and weight tensor 32×3×3×32 (CoHWCi). Different optimizations are showed by varying w_y, h_y, and C_y and violating the heuristics of Section 4.2.2.2.

4.2.6 Ablation Study

This section presents a detailed ablation study of each previously described contribution. I separately analyze: *i)* the proposed heuristics; *ii)* the hybrid optimization for depthwise layers; *iii)* voltage and frequency scaling on GAP-8; *iv)* the size of L1 and L2 memories; *v)* the specific GAP-8 architecture compared to standard MCUs.

4.2.6.1 Single tile performance

I analyze the effects of the heuristics shown on the tiling solution's quality-of-results. Moreover, I show the effect of applying these techniques to the border tile, increasing the performance in different configurations. In particular, the size of the tile influences the execution efficiency of the backend layer. As such, a sub-optimal tiling choice can significantly reduce performance in executing a single inner tile. Figure 4.6 exemplifies this phenomenon starting from an "optimal" tile of output tensor $24 \times 4 \times 32$ (HWC) with a $32 \times 3 \times 3 \times 32$ filter (channel out - height - width - channel in, or CoHWCi) and violating progressively each of the imposed heuristics. Violating `MATMUL_W/CH` leads to a maximum performance loss of 29%, violation of `HIDE_IM2COL` to a 38% loss, and violation of `PAR_BALANCE` to a 28% loss in this example layer. Noteworthy, the performance loss is cumulative since each heuristic is written to improve the performance of a different section of the PULP-NN kernel. Indeed, suppose I set all the β_i coefficients into the objective function of Eq. 4.3 to 0 and only focus on maximizing the tile sizes. In that

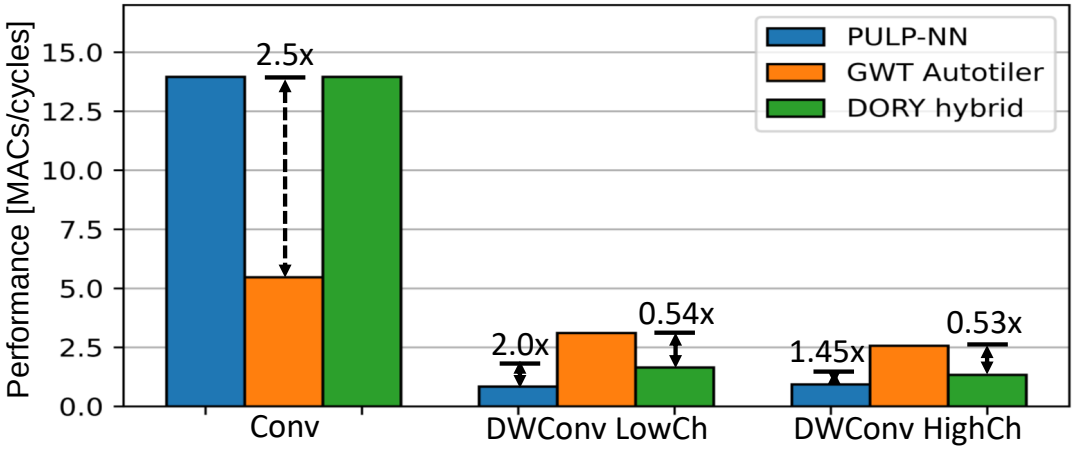

Figure 4.7: Comparison between HWC, CHW, and DORY layers layout. Different kernels are explored.

case, DORY chooses a tiling scheme that achieves only 2.78 MAC/cycles, 80.6% lower than the 14.37 MAC/cycles achieved with the β_i values previously reported.

4.2.6.2 Hybrid optimization for Depthwise layers

Here, I discuss the improvement of the new DORY kernel library (with new depth-wise) over PULP-NN kernels [25] (HWC layout) and Greenwaves' ones (CHW layout). In Fig. 4.7, I show a comparison of regular convolutions and depth-wise ones. On classical convolutions, the new HWC approach is 2.5× faster compared to the CHW layout. As discussed in Section 4.2.3, the DORY library includes an optimized depth-wise layer, reducing the penalty of using the HWC layout in its execution. Using an HWC layout on depth-wise layers can cause up to 3.7× slow down if compared to the CHW one, strongly penalizing the performance for these layers. With my newly designed kernels, I reduce this loss by a factor of 2: this kernel is 1.5×/2.0× faster than the HWC one, reaching 0.54× the performance of the Greenwaves' one. On the Mobilenet-v1-1.0 with resolution 128x128, updating the depth-wise and point-wise kernel from the HWC ones, I gain 1.79 MAC/cycles on the network's overall execution. At a frequency of 100 MHz on both cluster and I/O domains, I improved the 3.0 FPS of the HWC layout, reaching 4.3 FPS thanks to the optimized DORY kernel library.

4.2.6.3 Voltage and frequency scaling

Since the I/O DMA and the cluster are in two different clock domains, the ratio of the two frequencies can significantly impact the bandwidth of both the $L3$-$L2$ and $L2$-$L1$ transfers and the performance and energy efficiency. In Fig. 4.8, I show the

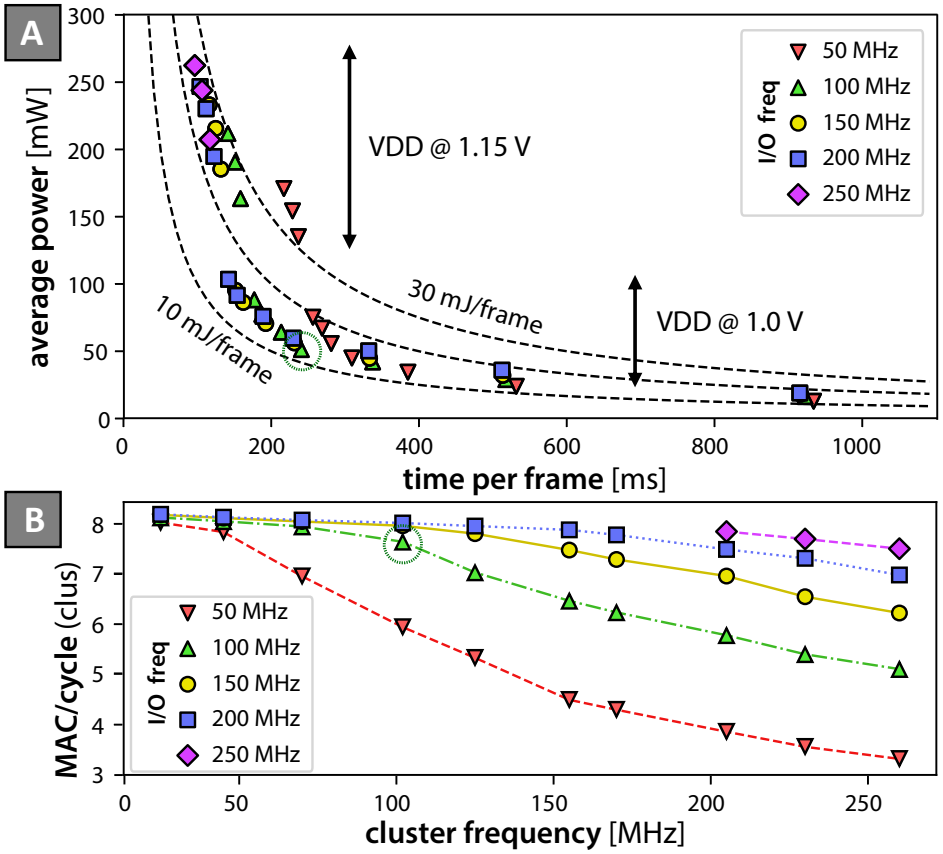

Figure 4.8: Power, latency, and MAC/cycles performance exploration with swiping frequencies. The 1.0-MobileNet-128 is used as a benchmark. CL frequency varies in [25 MHz, 260 MHz], I/O one in [50 MHz ,250 MHz]. A green dashed circle highlights the (100 MHz, 100 MHz) configuration used throughout the paper.

relationships between average power, execution time, and throughput in MAC/cycles, which are strictly related to the two frequencies. In sub-plot A, energy efficiency is also shown as a set of iso-energetic curves. The first significant effect that can be observed in these plots – particularly sub-plot B – is that increasing the fabric controller frequency improves performance. Increasing the fabric controller frequency causes the memory transfers to be faster since the DMA is in the same domain, minimizing the fraction of time in which the system is memory bound. On the other hand, increasing frequencies also raises proportionally average dynamic power, as visible in sub-plot A.

It is also interesting to observe that by using voltage and frequency scaling, it is possible to scale the execution of MobileNet from a minimum latency of 93.9 ms at 24.6 mJ per frame to minimum energy of 12.5 mJ at 244 ms per frame.

4.2.6.4 Memory hierarchy sizing

I also investigate the impact of memory dimensions on the network execution time.

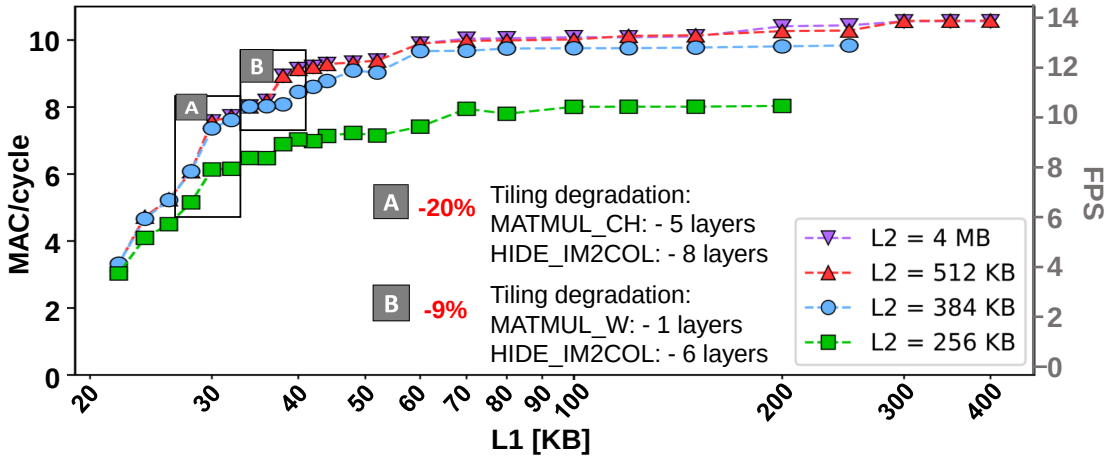

Figure 4.9: MAC/cycles and FPS are explored with different configurations of L1-L2 memories using a 1.0-MobileNet-v1 with resolution 128x128. L2 varies from 256 kB (19/29 layers tiled from L3) to 4 MB (No L3 tiling), whereas L1 varies from 22 kB to 400 kB.

To explore configurations with high dimensions of the memory, I used an FPGA-based emulator, realized with a Xilinx Zynq Ultrascale+ zcu102 since it can host different instantiations of the PULP architecture template with varying memories of various sizes.

Since DORY solves a series of tiling constrained problems, these constraints are relaxed/hardened if I increase/reduce the internal MCU memories' size. Fig. 4.9 depicts MAC/cycles and FPS while sweeping L1 between [22 kB, 400 kB] and L2 in {256 kB, 384 kB, 512 kB, 4 MB}, highlighting different working corners in the tiling problem. L1 memory limits have been chosen since *i)* 22 kB are needed to construct the smaller tile available and store the corresponding im2col buffer, and *ii)* over 400 kB no performance improvements are yet observed. L2 limits are related to chip design: 256 kB is the lowest memory used as on-chip memory on a PULP platform [55], and no current MCUs currently have more than 4 MB as maximum memory.

A first performance gap can be observed between the L2 = 256 kB and L2 = 512 kB configurations: with different L1 dimensions, using half of the memory causes up to 3.2 FPS loss @ 260MHz. Using only half of the L2, 9 out of 29 layers demand the tiling of their activations from the external memory slowing down the execution of the first half of the network since they can not fit the tightened constraint. Readers can observe a relatively constant decrease in performance when reducing L1 memory from 70 kB down to 22 kB with some abrupt performance loss. Two different phenomena can be observed: first, reducing L1 memory requires smaller tiles and hence more iterations, increasing overhead. A second and more severe degradation can be observed when the tiling heuristics of the network layers cannot be maximized anymore. For example, from case **A** of Fig. 4.9, if I reduce L1 memory from 30 kB to 28 kB, the heuristics of 13 layers simultaneously get worse with a corresponding degradation of 20% in the

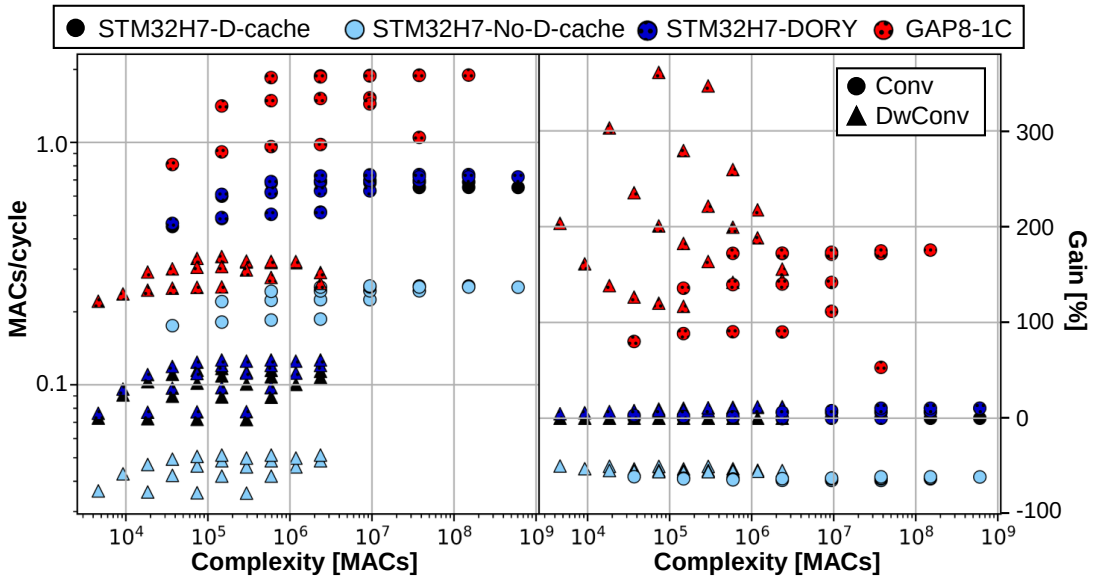

Figure 4.10: On the left, absolute MAC/cycle of DORY framework on STM32H7 and single-core GAP8, compared with default CUBE-AI/TensorFlow Lite for Micro layer backend, CMSIS-NN. On the right, relative gains compared to the fastest CMSIS-NN implementation.

performance. Conversely, from 70 kB to 400 kB of L1 the gain is minimal because all the tiling heuristics are already satisfied.

Overall, thanks to DORY's optimizations (tiling and optimized backend), I see that a 80 kB L1 and 384 kB L2 memory configuration is sufficient to lead to a MAC/cycle degradation of 8% (from 10.57 to 9.74 MAC/cycles) compared to the largest memory configuration of the platform.

4.2.6.5 Single core performance on different architectures

In this section, I explore the impact of architectural and microarchitectural choices on DNN deployment using DORY. To do so, I directly compare the single-core performance obtained on GAP-8 with that achievable on the STM32H743ZI2 MCU in several configurations. DORY is generic enough that it can be easily ported to many different platforms. To show this feature, I not only run "native" layers using the STM32H7 data cache but, as an alternative, I used DORY to manage the DTCM scratchpad on this device manually.

In this experiment, I tested 44 different layers configurations (depthwise and convolutional) spanning six orders of magnitudes of complexity. I show four sets of solutions: for GAP-8, I used DORY and ran on a single core in the cluster; for the STM32H7, I used CMSIS-NN with and without D-Cache enabled. Finally, in the third STM32H7 configuration, I ran using the DTCM scratchpad by combining DORY (for memory

management) with CMSIS-NN. This was possible thanks to the modular architecture of DORY and required only changing the computational backend and adapting the code generator to use the correct DMA hardware abstraction layer calls.

The results are shown in Fig. 4.10. First of all, as expected, performance drops dramatically deactivating the D-Cache on the STM32: I observe a degradation of 58.5 $\pm$ 5.5 % compared to the baseline over all the benchmark layers. More interestingly, the results also show that the software caching mechanism realized by DORY on the DTCM can achieve the same performance as the D-Cache on average, with a slight speedup in some cases: on average, 9.1$\pm$ 2.1 % for depthwise layers and 3.9 $\pm$ 3.8 % for normal convolutions.

On the other hand, single-core execution on GAP-8 shows, on average, a speedup of 2.5$\pm$0.9$\times$ compared to the STM32H7 baseline in terms of cycle/cycle. Since multi-core execution is disabled in this test, the speed-up achieved in GAP8 compared to the STM32H7 is referred mainly to the more specialized architecture, particularly to the DSP extensions extensively exploited by the PULP-NN backend.

4.3 TCN Mapping Optimization for Ultra-Low Power Time-Series Edge Inference

In this section, I will describe a new kernel library for Temporal convolutional networks, which is thought to be plugged inside DORY, to generate end-to-end temporal convolutional networks on the edge.

4.3.1 TCN Kernel Toolkit

I first introduce the main design choices on which this library is based, the kernel implementations, and how they are plugged into DORY. I tested these kernels again on the GAP8 platform.

4.3.1.1 Design Choices

1) Data Layout: Kernel libraries for 2D convolution organize input and output data either as Channel-Height-Width (CHW), i.e., with the spatial dimension as the innermost one or HWC. In the case of 1D convolutions, the equivalent layouts are CT (channel-time) and TC (time-channel). Using TC, inputs relative to subsequent time-steps are separated by $d \times C_{in}$ elements. In particular, for $d = 1$, all convolution inputs are stored contiguously. Therefore, given the presence in many DSP-oriented ISAs of single-cycle loads with pointer increment (e.g., `p.lw` in *XpulpV2*, the ISA of GAP8), I select the TC layout. The chosen data organization is shown in the "x buffer" of Fig. 4.11.

2) Data Gathering: Conceptually, the convolution kernels in this library operate in two phases, which I will call *input data gathering* and *MatMul loop* respectively, similarly to [25, 33]. In the first phase, dedicated buffers in each core are used to prepare the input data needed for the convolution. Differently from CMSIS-NN and PULP-NN, I propose different ways of data re-ordering, either using explicit *im2col* buffers [33] or *indirect* buffers [99]. This design choice is motivated by the fact that the two buffers exploit different trade-offs regarding memory occupation and performance. For instance, the indirect buffers strongly reduce memory occupation on layers with a large C_{in}, allowing them to fit in small on-chip memories. At the same time, im2col leads to high-memory occupation but also better performance. Note that in non-dilated 1D-CNN convolutional layers ($d = 1$), input data is already contiguous, and this phase can be bypassed.

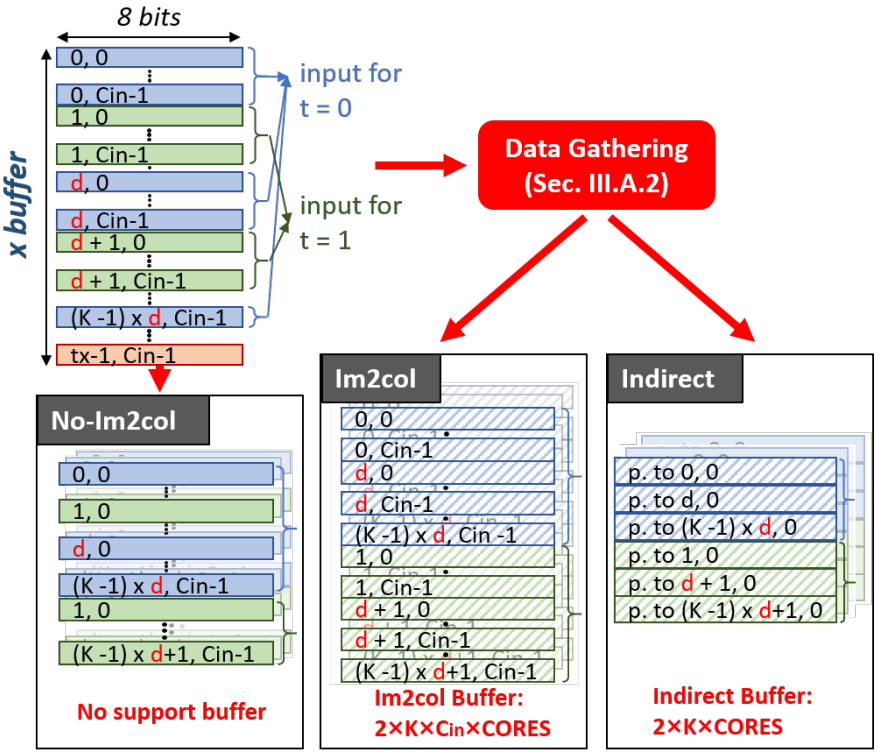

Figure 4.11: Three different input data gathering options are used in the proposed kernels.

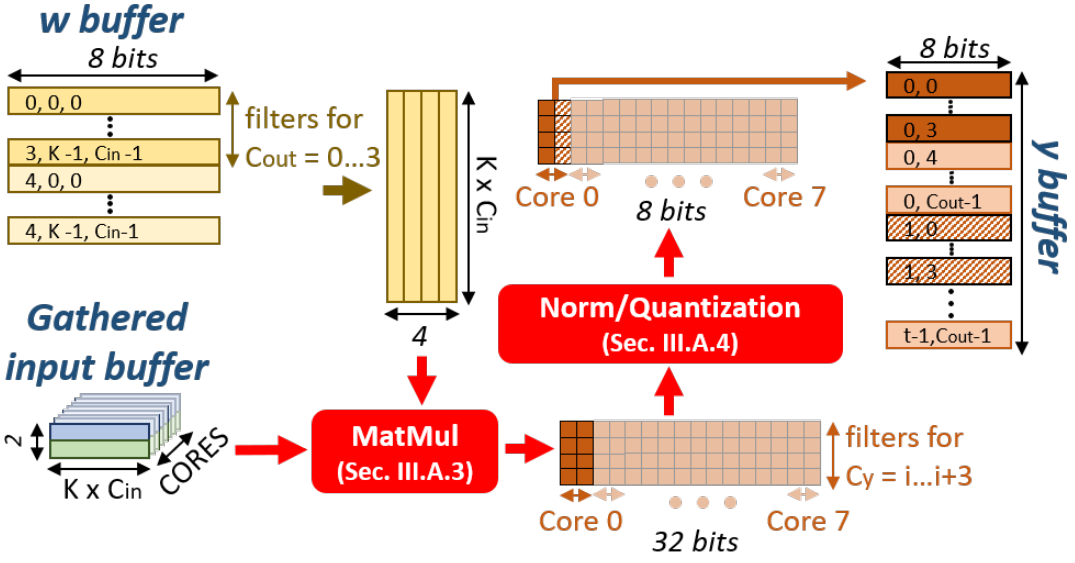

Figure 4.12: MatMul loop, Quantization, and Batch Normalization in the proposed toolkit. Lighter colors represent parallelization over multiple cores.

3) MatMul Loop: After data gathering, convolution reduces to a series of MatMul, as depicted in Fig. 4.12. As an atomic operation, I use a 4×2 unrolled MatMul as in PULP-NN (i.e., the product of 4 sets of weights with two sets of inputs). Indeed, it has been demonstrated that 4×2 unrolling maximizes data reuse in a RISC-V register file with 32 registers. Since 4×2 unrolling requires two sets of inputs, I allocate two im2col/indirect buffers in each core (see Fig. 4.11). Each unrolled MatMuls is further

vectorized using the `pv.sdotsp.b` instruction of the *XpulpV2* ISA, which computes the dot product of 4 contiguously stored 8-bit inputs in parallel.

4) Normalization and Quantization: I "fuse" the quantization and normalization pointwise operations, essential for quantized inference [100], together with the new convolution kernels. In contrast, using *separate* kernels for these operations would result in additional data movement and worsen performance.

5) Parallelization: I split the convolution workload on multiple cores over the time dimension, i.e., each core computes the output features of all channels for an assigned range of time steps. I select time-wise over channel-wise parallelization since it allows cores to produce outputs of their assigned time-steps without exchanging partial data with other cores and to store results on a separate, contiguous memory area. The workload subdivision among cores is shown on the right of Fig. 4.12.

4.3.1.2 1D Convolutional Kernels

The three convolution kernels implemented in this library differ mainly in the data gathering phase, as shown in Figure 4.11.

1) No-im2col Kernel: As explained in Section 4.3.1.1-1, due to the sequential nature of 1D data and the TC layout, data gathering can be bypassed when $d = 1$. Removing this buffering phase positively affects both memory usage and performance. On the other hand, for kernels with $d > 1$, performing the MatMul loop without data gathering would require interleaving the weight vectors with zeros to eliminate the contribution of input time-steps that have to be skipped. The resulting memory occupation increase and performance loss make the *No-im2col* approach feasible only for non-dilated convolutions, where, however, it is optimal for both memory and performance.

To efficiently handle dilation rates higher than 1, data gathering becomes necessary. *2) Im2col kernel:* One approach is to use an im2col buffer [33] (bottom-center of Figure 4.11). This support buffer is a linear array in which all inputs required to produce a given convolution output are copied contiguously. When the convolution stride is smaller than K, data will be replicated in multiple im2col buffers, causing a memory overhead. However, the linear im2col output yields maximal exploitation of the hardware facilities to optimize the MatMul performance (e.g., SIMD operations, single cycle pointer increment, etc.). *3) Indirect kernel:* To minimize the memory footprint of convolution, the im2col buffer can be replaced with an *indirect* buffer for data gathering. Instead of copying all convolution inputs in contiguous memory, this buffer only stores the *pointers* to the first input relative to each time-step involved in the convolution (bottom-right

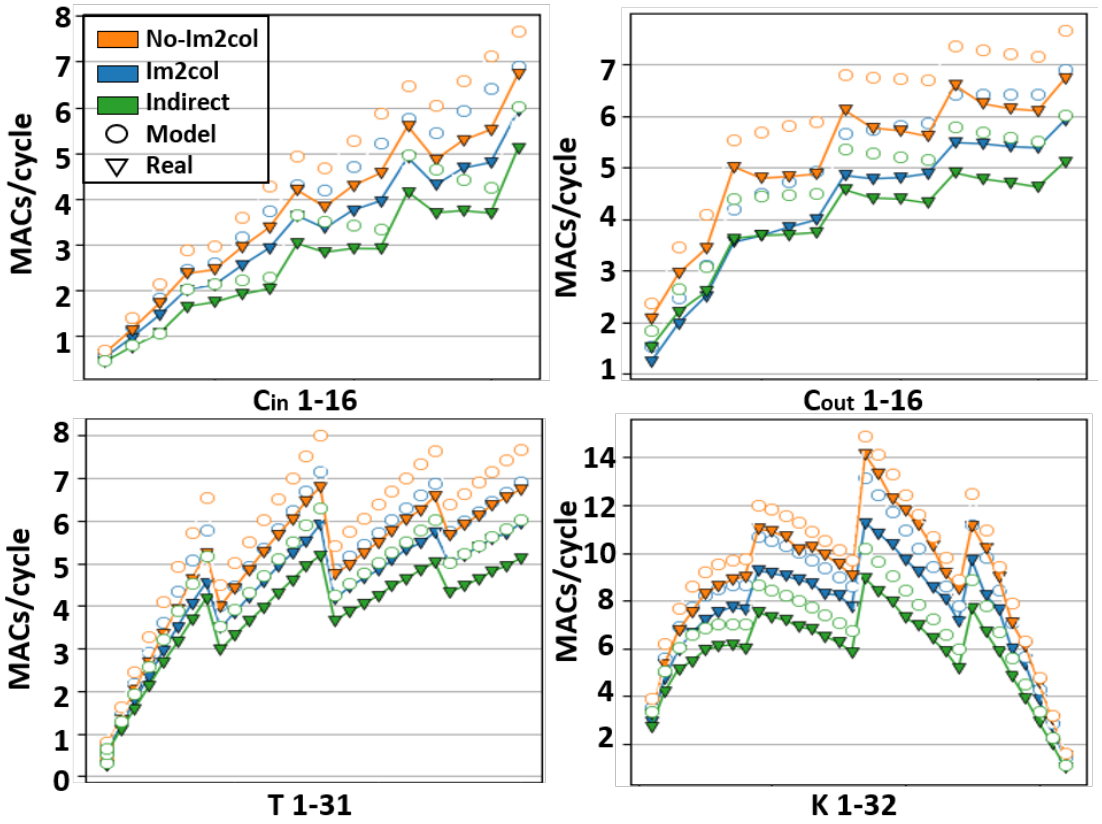

Figure 4.13: Modeling of the three kernels versus various layer parameters.

of Fig. 4.11). Indirect convolution reduces by a factor C_{in} the memory overhead for data gathering (that it becomes negligible compared to buffer memory) but requires an additional loop to cycle through the buffer's addresses in the MatMul section, negatively impacting performance. Note that this is the first edge-oriented backend to include both im2col and indirect convolution kernels.

4.3.1.3 Kernel modeling and selection

In this section, I show how this library is plugged inside DORY. To do so, I modified the baseline optimizer so that it finds appropriate tiling solutions and selects the optimal 1D convolution implementation for a given layer and tiling via an additional *kernel selection* step. Therefore, I computed a detailed model of each kernel's execution cycle based on the target platform's compiled assembly code. Based on the models, the optimizer first determines the best tiling scheme for each of the three alternative implementations using the Constraint Programming (CP) solver of [100] and it then computes the absolute execution cycles to choose among the *no-im2col*, *im2col* and *indirect* kernel implementations. As an example, to model the performance of the *im2col* kernel, the total number of convolutions performed by each core are denoted as as

Core_Iter $= \frac{T}{2N_{cores}}$, where T is the total number of time-steps in the input sequence and the factor 2 comes from the fact that all cores manage 2 time-steps simultaneously. I also call MM_Iter $= \frac{C_{out}}{4}$ the number of iterations on the output channel dimension performed within each convolution, where the factor 4 comes from the 4x2 MatMul loop that simultaneously generates 4 C_{out} elements (Sec. 4.3.1.1).

I then compute the execution cycles for the two main phases (data gathering and MatMul) and for the entire kernel as:

$$\text{Gather\_Cyc} = \max\left(2 \times K \times \alpha, 2 \times K \times C_{in} \times \beta\right) \tag{4.4}$$

$$\text{MM\_Cyc} = \left(\gamma + \delta \times C_{in} \times {}^{K}/_{4}\right) \tag{4.5}$$

$$\text{Cyc} = \text{Core\_Iter} \times (\epsilon + \text{Gather\_Cyc} + \text{MM\_Iter} \times \text{MM\_Cyc}). \tag{4.6}$$

where α, β, γ, δ, ϵ are hardware-dependent constants corresponding to the cost in execution cycles for load/store, pointer updates, and arithmetic operations.

The first equation derives from using asynchronous DMA transfers for data gathering. It computes the maximum between the DMA control overhead (first term, dependent on the $2 \times K$ DMA invocations needed to build the two im2col buffers) and the cycles required for the actual transfer (second term, dependent on the size of the actual transmitted data). The MM_Cyc equation computes the cycles of the MatMul loop as a function of the layer parameters, where the division by 4 comes from the use of SIMD operations processing four 8-bit elements per instruction. Here, the constants also account for batch normalization and quantization.

Models for the other two convolution kernels are similar, with different amounts of cycles for the different phases based on data position.

Figure 4.13 shows all kernels' modeled vs. real performance for different parameter sweeps. Although there is an offset between real execution cycles and predicted ones, due to stalls and memory contentions, this gap is almost constant over all the parameters and kernels, hence not changing the ranking between different kernels' for a given set of parameters.

Table 4.5 shows the performance achieved plugging these kernels inside DORY. The table compares the results obtained with the new cycle models with those obtained with other objective functions for the same tiling optimizer, namely the tiles' pure memory occupation and the simplified model based on previously described heuristics. These new models achieve 1.3×/3.6× speed-up for complete layers with different geometries. This is due to an accurate assessment of the execution time of border tiles, for which the computation loop might be under-utilized depending on the amount of data remaining to

Table 4.5: Performance of the TCN kernel library using different optimization criteria for tiling parameters and kernel selection.

layer	MACs/cycle			
($C_{in} \times T \times C_{out}$)	model	heuristic	memory	best kernel
$64 \times 256 \times 64$, $d = 1$, $K = 3$	**15.98**	15.47	12.50	no-im2col
$256 \times 16 \times 256$, $d = 2$, $K = 3$	**14.92**	**14.92**	4.14	im2col
$1024 \times 16 \times 1024$, $d = 2$, $K = 3$	**13.31**	8.94	10.42	indirect

be computed. Further, I want to highlight that the three kernel variants are helpful for different cases. The column *best kernel* shows the kernel selected by the tiling optimizer as the most efficient.

4.3.2 Experimental Results and Discussion

I tested this library on GAP-8 [27], the platform described in Sec. 2.3.2. Since 1D dedicated libraries are not present, I compare this work with two state-of-the-art CNN backends for the same hardware target (PULP-NN [25] and GWT NN-Tool, on the GAP_SDK v3.6 [101]). I also compare it with the Cube-AI toolchain (v5.1.2) [71] executed on the STM32H7 and the STM32L4 MCUs. All experiments refer to int8 quantized layers. Entire networks are trained in a quantization-aware manner, with negligible accuracy loss compared to float versions. I set GAP8, STM32H7, and STM32L4 frequencies at 100 MHz, 480 MHz, and 80 MHz, with a corresponding power consumption of 51 mW, 234 mW, and 10 mW, respectively. As a result, I report GMAC/s, GMAC/s/W, and MACs/cycle as comparison metrics. Note that while the first two are platform-dependent and thus most significant for backends on the same hardware (i.e., the introduced toolkit, PULP-NN, and GWT NN-Tool), the latter is platform-independent, therefore not linked to the frequency or power consumption of the specific platforms.

4.3.2.1 Kernels Comparison

Fig. 4.14 and Fig. 4.15 show a detailed analysis of the three 1D convolution implementations for a $64 \times 256 \times 32$ layer (i.e., $C_{in} = 64$, $T = 256$, $C_{out} = 32$) with $K = 3$, i.e., a small enough layer that entirely fits the L1 memory of GAP8. Fig. 4.14 reports the execution cycles for the data gathering and MatMul loop phases and the additional cycles due to stalls and memory contentions, whereas Fig. 4.15 breaks down the memory occupation. The graphs report the results for both $d = 1$ and $d = 2$.

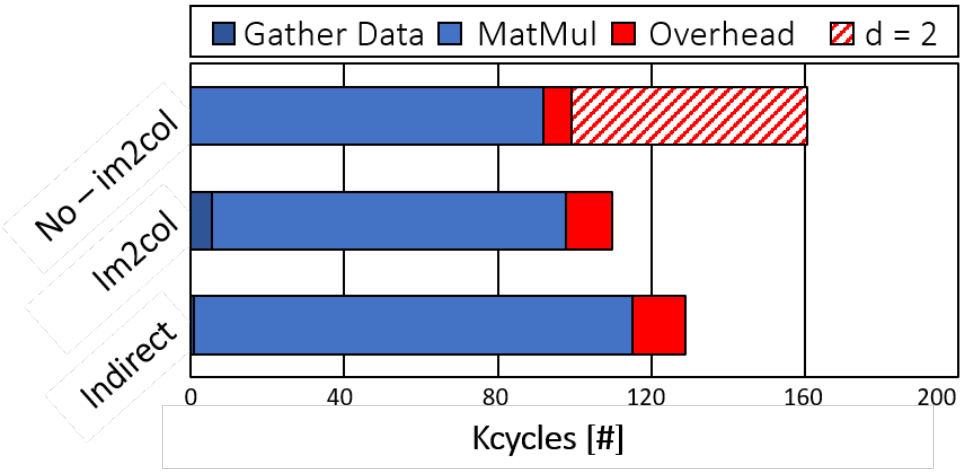

Figure 4.14: Execution cycles of the three 1D convolution kernels on a $64 \times 256 \times 32$ layer. The three kernels achieve 15.1, 13.7, and 12.0 MACs/cycle, respectively. With $d = 2$, the No-im2col performance lowers to 9.7 MACs/cycle.

For $d = 1$, the No-im2col kernel obtains both the minimum number of cycles and the minor memory occupation, as previously said, since it does not have the memory overhead of the im2col nor the performance degradation of the indirect kernel. However, for $d > 1$, the same kernel has significant overheads in operations and memory due to the added zeros in the weight buffer. For $d = 2$, I show 62% more operations and an additional 5 KB of memory, making the No-Im2col kernel the worst of the three. These overheads increase with larger d.

The Im2col kernel uses fewer instructions in the MatMul loop than the Indirect one while spending more time creating its gather buffer. In this example, the trade-off results in an overall lower number of cycles for Im2col. However, note that the gathering overhead is much higher for layers with a larger C_{in} (see Eq. 4.4). Therefore, the ranking among the two depends on the number of channels. Further, the Indirect kernel benefits from a nearly null additional memory, often improving the performance when considering the effect of tiling on large layers as can be seen for end-to-end network execution (Section 4.3.2.3).

4.3.2.2 Comparison with State-of-the-art NN backends

Figure 4.16 shows a complete comparison between the DORY+TCN library and the other backends. The figure reports the performance (in MAC/cycle) for layers with dilation $d \in (1, 2, 4, 16)$. For each value of d, the box plots aggregate the results of multiple layers with different shapes. Specifically, I show $T \in (16, 64)$, $K \in (3, 5, 7)$, and $C_{in} = C_{out} \in (32, 64, 128, 256)$.

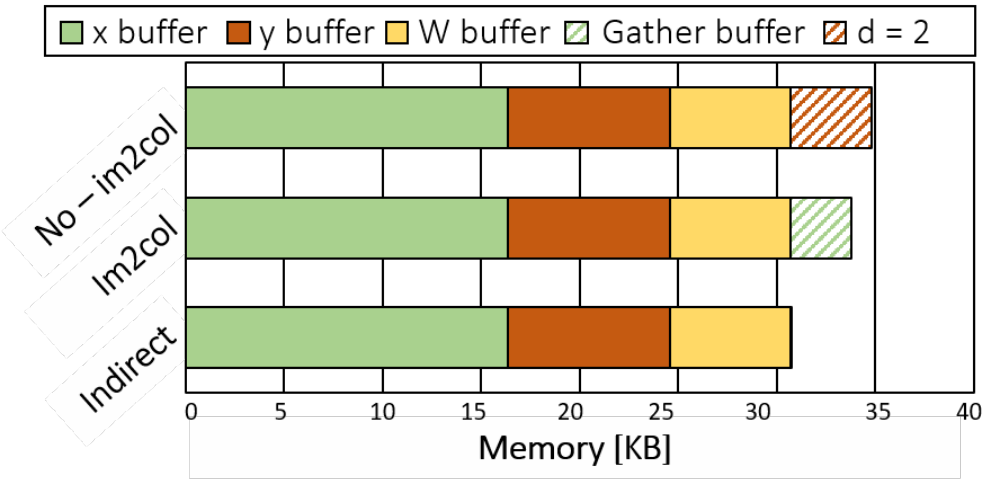

Figure 4.15: Memory occupation of the kernels of Fig. 4.14.

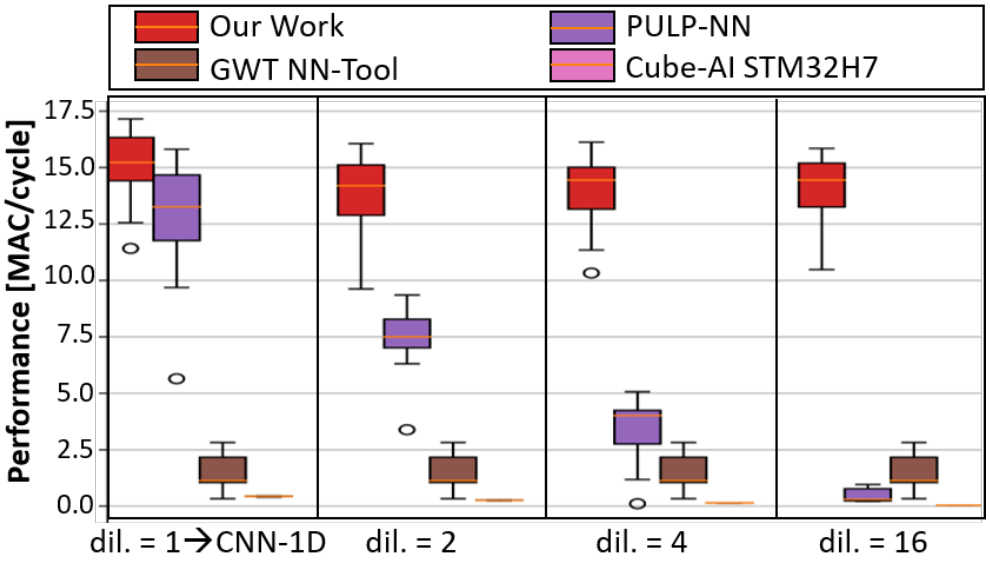

Figure 4.16: Comparison with state-of-the-art CNN backends for edge devices.

This newly introduced solution consistently outperforms the state-of-the-art across different layer shapes and dilation values. In particular, I show dramatically higher performance than GWT NN-Tool, i.e., 9.7× on average. This improvement is due to the CHW format used in its convolutions, which converts to a strongly sub-optimal CT layout for 1D kernels (see Sec. 4.3.1.1 for details). Compared to PULP-NN, I slightly improved the performance for $d = 1$ (1.2×), thanks to eliminating unnecessary im2col buffers and optimizing the internal MatMul loop execution for 1D data. The benefit increases significantly for larger dilation factors (e.g., 28.9× for $d = 16$) since, as mentioned, PULP-NN kernels do not support this fundamental 1D-convolution parameter, which has to be reproduced interleaving weights with 0s. With respect to Cube-AI, a speed-up between 34.7× (for $d = 1$) to 354× (for $d = 16$) is obtained. The higher speed-up is achieved for higher d, given the same reasoning for PULP-NN. Comparing

both of them during single-core execution, the DORY-TCN toolkit still demonstrates 4.7×, 7.0×, 13.0×, and 47.6× higher MACs/cycle. Considering the energy efficiency in GMAC/s/W, the improvement over PULP-NN and GWT NN-Tool is proportional to the speed-up, given that the execution platform is the same. Compared to Cube-AI, instead, considering the best energy configuration for both STM32H7 and GAP8, I obtain 33.1×, 50.0×, 92.3× and 338.4× higher efficiency on average for $d = 1$, 2, 4, 16. Notice that $d = 1$ corresponds to a standard 1D CNN layer; hence, the first set of box plots show that DORY, together with this new library, is outperforming the state-of-the-art not just on dilated TCNs, but also on classical 1D-CNNs.

4.3.2.3 Complete use cases

In this section, I employ DORY together with the proposed kernel library to deploy three state-of-the-art neural networks, shown in Table 4.6, i.e., TEMPONet [34] for gesture recognition, and two ResTCNs from [11], for sound generation and language modeling, respectively. While the number of layers of the three networks is similar (9, 8, and 10), the number of filters per layer, hence the number of parameters and MACs, is increasingly high. Specifically, TEMPONet has a modular structure that shrinks the time dimension while increasing the number of channels up to 128 [34], while the other two TCNs maintain a constant T (16 and 50) with respectively 150 and 450 channels per layer.

I want to highlight two main aspects of these experiments. First, integrating the kernel selection step leads to up to 4.0× speed-up compared to always using a single kernel implementation. While mapping all the layers of TEMPONet to the Im2col kernel leads to a near-optimal implementation, the same strategy applied to the language-modeling TCN yields 4× lower performance than a per-layer selection. Similarly, considering the Indirect kernel solely results in 1.3× lower performance on TEMPONet. Therefore, choosing the appropriate kernel for each layer is key to maximizing performance. In general, for layers with $d = 1$, No-im2col reaches the highest performance, while Im2col and Indirect are optimal for layers with $d > 1$ with a low/high number of channels, respectively.

Table 4.6: End-to-end comparison on three TCNs architectures for different tasks. Abbreviations: OOM: Out of Memory.

	Our Work				PULP-NN	NN-Tool	Cube-AI	
	Indirect	No-Im2col	Im2col	Optimizer				
Platforms [MCU]	GAP8, 1xRISC-V + 8xRISC-V						STM32H7	STM32L4
Power [mW] / Freq. [MHz]	51 mW / 100 MHz						234 mW / 480 MHz	10 mW / 80MHz
TEMPONet - Gesture Recognition - Parameters: 86.5k - MACs: 15.1M								
Time/Inference [ms]	17.18	29.79	13.94	13.60	38.78	103.10	138.29	1408.16
Energy[mJ]	0.88	1.52	0.71	0.69	1.98	5.26	32.36	14.08
MACs/cycle	8.80	5.07	10.84	11.11	3.90	1.47	0.23	0.13
Throughput [GMAC/s]	0.88	0.51	1.08	1.11	0.39	0.15	0.11	0.01
En.Efficiency [GMACs/s/W]	17.25	9.95	21.26	21.79	7.64	2.87	0.47	1.07
ResTCN - Sound Generation - Parameters: 1.13M - MACs: 18.1M								
Time/Inference [ms]	23.91	OOM	24.66	23.45	OOM	568.25	215.79	OOM
Energy[mJ]	1.22	OOM	1.26	1.20	OOM	28.98	50.49	OOM
MACs/cycle	7.57	OOM	7.34	7.72	OOM	0.32	0.17	OOM
Throughput [GMAC/s]	0.76	OOM	0.73	0.77	OOM	0.03	0.08	OOM
En.Efficiency [GMACs/s/W]	14.84	OOM	14.39	15.13	OOM	0.62	0.36	OOM
ResTCN - Language Modeling - Parameters: 2.7M - MACs: 135M								
Time/Inference [ms]	171.00	OOM	665.91	168.51	OOM	4490.00	2315.25	OOM
Energy[mJ]	8.72	OOM	33.96	8.59	OOM	228.99	541.77	OOM
MACs/cycle	7.89	OOM	2.03	8.01	OOM	0.30	0.12	OOM
Throughput [GMAC/s]	0.79	OOM	0.20	0.80	OOM	0.03	0.06	OOM
En.Efficiency [GMACs/s/W]	15.48	OOM	3.98	15.71	OOM	0.59	0.25	OOM

After, I also compare the best performance obtained with state-of-the-art full-stack tools, including their backend and, when available, their tiling mechanism. A minimum speed-up of 2.9× compared to the DORY+PULP-NN stack for the three networks is achieved. However, 2 out of 3 cannot be implemented using this tool given the high memory overhead of the Im2col support buffer, which is always necessary for the PULP-NN backend. On the other hand, directly storing the whole network in the slow GAP-8 L2 memory (512KB) leads to a slowdown of more than 4×. When I compare this new toolkit to Cube-AI and GWT NN-Tool on all the networks, I observe speed-ups of 7.6× to 103×, with at least 20.3× lower energy. In terms of absolute latency numbers, using GAP8 at 100 MHz, I obtained 13.6ms per inference for TEMPONet, and 23.45 / 168.5ms for sound/language processing, respectively.

4.4 A Microcontroller is All You Need: Enabling Transformer Execution on Low-Power IoT Endnodes

I will conclude the fourth Chapter of my thesis with the introduction of a second kernel library for attention based networks, given the spreading of this new topology in many applications also related to edge computing. I will start with re-introducing the fundamental equation of attention, as reported in the Background section:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) \doteq \mathbf{AV} \doteq \underset{\text{over keys}}{\text{SoftMax}}\left(\frac{\mathbf{QK}^{\text{T}}}{\sqrt{P}}\right)\mathbf{V} \tag{4.7}$$

4.4.1 Self-Attention Kernels

Fig. 4.17 shows the Multi-Head Self-Attention operator, the core operator of Transformer networks as described in [40]. Four Linear and two Matmul layers constitute the layer, followed by memory marshaling operations such as transposition, reshape, and concatenation. Even when these kernels are individually optimized using state-of-the-art libraries [25, 33], they result in low data reuse and non-optimal parallelization scaling. Therefore, this section presents a new set of tailored kernels for each internal operation to address these issues. In each sub-section, I describe the optimization of one different layer separately.

4.4.2 Linear Layers

Fig. 4.18 depicts two distinct implementations for the three input Linear layers, i.e., layers that can be reduced to a matrix-matrix multiplication. Implementation ❶, which I will call *Weight-Reuse Linear* (WRL), is used to project the **V** tensor from **X**, while ❷, called *Input-Reuse Linear* (IRL), is used for **Q** and **K**. These kernels differ in i) loop ordering and ii) data layouts.

The WRL kernel produces output data in the HPS layout to remove the data reshaping operator (see Fig. 4.17). I also order the loop as $H \rightarrow P \rightarrow S \rightarrow E$, from outermost to innermost. Indeed, when targeting multi-core platforms such as GAP8, I parallelize this kernel on the H dimension, thus splitting the outermost loop across cores.

On the other hand, the IRL's required layout is HSP, and thus I force the loop nest to $H \rightarrow S \rightarrow P \rightarrow E$, from outermost to innermost. In this kernel, at every iteration of the S loop, a single input time sample ($1 \times E$) is multiplied by a weight-head ($E \times P$). The parallelization is identical to the WRL case, i.e., on the H dimension.

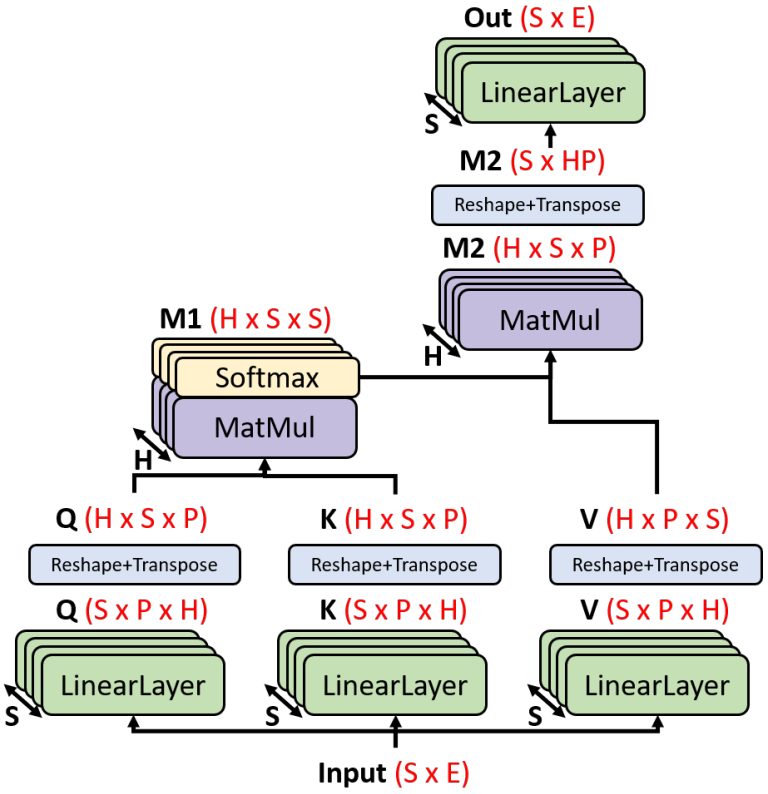

Figure 4.17: Multi-Head Attention module.

The last Linear layer, which projects the output tensor of the matrix multiplication to the final output, uses the $S \rightarrow E \rightarrow H \times P$ loop order, parallelizing on E.

4.4.3 Matrix Multiplications

Also, in the case of matrix multiplications, I optimally order the loop executions and parallelize over the outer dimensions to improve performance. Fig. 4.19 reports two different implementations for the two matrix multiplications in the Self-Attention kernel. The Softmax-Matmul ❸ merges the matrix multiplication with the Softmax operator; it uses the $S \rightarrow H \rightarrow S$ loop order; P is the dimension over which the reduction is performed. After completing each iteration of the H loop, the Softmax is applied to the data produced (e.g., the first one, S_0H_0). The output data of this matrix-multiplication is stored in SHS data layout to allow the second Matmul ❹ to ingest data sequentially and use load/store specialized operations. This implies reading input data for the Softmax-Matmul layer in non-sequential order but still with a regular stride, thus not impairing the performance. For instance, a set of weights, H_0 (dimensions, $P \times S$), is multiplied with the corresponding activation buffer, H_0S_0 (dimensions, $1 \times P$). Then, the set of weights H_1 is multiplied with the input H_1S_0, which is stored S input buffers after the first one (with a stride of $S \times P$). After all head positions relative to sequence S_0 are multiplied with the K matrix, the cycle is repeated for each S_i of Q input.

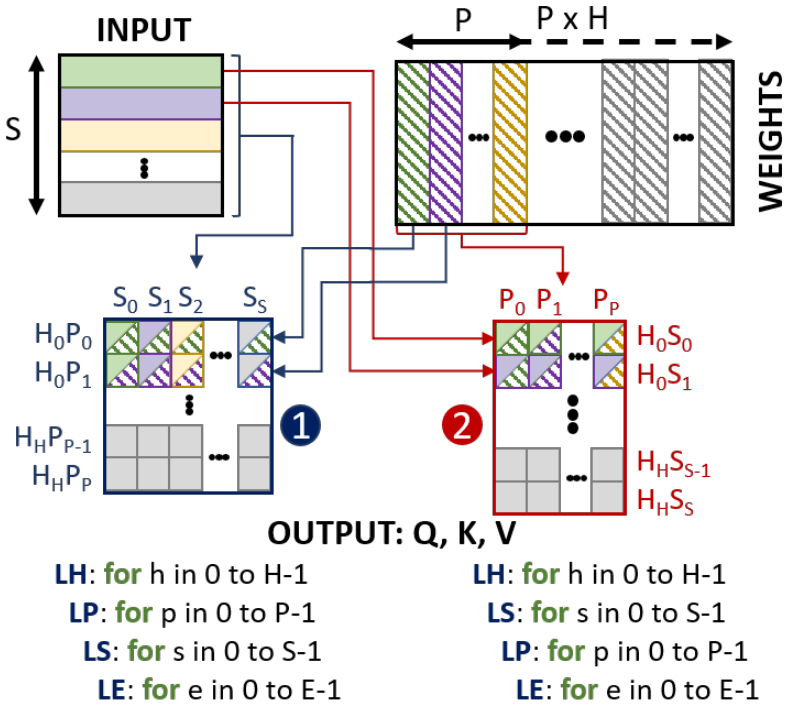

Figure 4.18: Linear layer data flow for *HPS* and *HSP* out data layouts. Output matrices are filled from left to right, top to bottom.

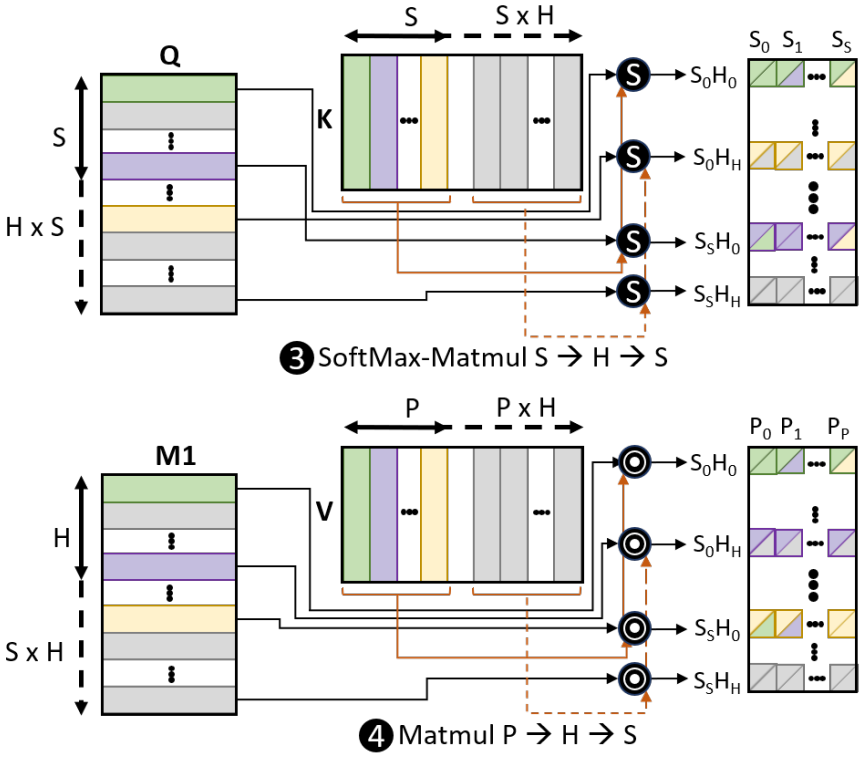

Figure 4.19: Matrix multiplication designed to work with multiple heads with different data layouts.

```
Inputs: I, W; Outputs: Q, K, V
C = H / CORES
H_start = min(C * CORE_ID, H); H_end = min(H_start + C, H)
  LH: for (h = H_start; h < H_end; h++)
    LP: for (p = 0; p < P/2; p++)
      LS: for (s  = 0; s < S/4; s++)
        S0...7 = {0};
        LE: for (e = 0; e < E/4; e++)
          A1 = I(4s); A2 = I(4s + E);
          A3 = I(4s + 2E); A4 = I(4s + 3E);
          B1 = W(hpE + 2pE); B2 = I(hpE + (2p+1)E);
          S0 += sdot4(A1,B1); S1 += sdot4(A1,B2);
          S2 += sdot4(A2,B1); S3 += sdot4(A2,B2);
          S4 += sdot4(A3,B1); S5 += sdot4(A3,B2);
          S6 += sdot4(A4,B1); S7 += sdot4(A4,B2);
        O(h,2p,4s) = quant(S0);
        O(h,2p,4s+1) = quant(S1);
        O(h,2p,4s+2) = quant(S2);
        O(h,2p,4s+3) = quant(S3);
        O(h,2p+1,4s) = quant(S4);
        O(h,2p+1,4s+1) = quant(S5);
        O(h,2p+1,4s+2) = quant(S6);
        O(h,2p+1,4s+3) = quant(S7);
```

Listing 4.3: Example of kernel pseudocode of sub-layer ❶.

Matmul ❹ produces the final tensor, which is then fed to the output projection Linear layer. Given the design of the previous layers, ❶, and ❸, its implementation is straightforward. The loop execution order is $S \rightarrow H \rightarrow P$, with P as innermost dimension. The reduction dimension is the innermost S of the **M1** matrix.

4.4.4 Kernel execution loops

The general rules that I employed for kernel optimizations are 3:

1. Keep the parallelization (when available on the target platform) as much as possible on the H dimension.
2. Exploit output stationarity.
3. Produce the output tensors sequentially (i.e., element $i + 1$ in the innermost dimension is always generated immediately after the i-th element).

The first guideline is particularly convenient when a low number of cores is available, given the typically low number of heads in networks (e.g., eight heads in the original transformer [40]). Furthermore, the heads dimension is present in all the kernels[4]. The

[4]The Linear output layer represents an exception, as I parallelize on E to maintain output stationarity.

second guideline saves memory by avoiding the storage of many intermediate `int32` accumulators for the partial outputs. This is particularly important for memory-starved MCUs, as described for DORY. Besides the memory saving, output stationarity enables optimal exploitation of the dot-product Single Instruction Multiple Data (SIMD) instructions, as demonstrated in [25]. Finally, producing output tensors sequentially prevents undesired additional operations in the innermost loop to compute storage locations.

Listing 4.3 reports an example of the pseudocode of layer ❶ with the `RV32IMCXpulpV2` ISA and GAP8 target. In the innermost loop, I exploit the `sdot4` operator to perform 4 Multiply-and-Accumulate (MAC) operations in a single instruction. Further, inspired by [25], I again perform 8 `sdot4` operations in the same loop iteration as for the TCN kernels, thus eliminating Read-After-Write (RAW) hazards and performing just 6 load operations (4 activations buffers and 2 weights buffers), better exploiting the data reuse in the register file. Incrementing the number of produced output values in a single iteration, e.g., to 16, would cause an increase in the number of utilized registers to 24 (16 for outputs, 4 for inputs, 4 for weights), causing additional load and store operations to spill variables from the register file to the stack to make room for operands. Conversely, reducing the number of registers employed causes a reduction in the MAC/load ratio and impairs the performance.

4.4.5 Quantization

Since commercial off-the-shelf Microcontroller Units (MCUs) feature memory in the order of hundreds of kilobytes up to one megabyte, quantization of both weights and activations is typically used to reduce the memory footprint of trained networks [102]. Besides saving precious memory space on these tightly constrained devices, quantization to 8 bits allows specialized microcontrollers to leverage SIMD instructions, which leads to significant speed-up compared to floating-point computations. To quantize the Self-Attention layers and allow for deployment onto commercial MCUs using the new kernels and the baseline kernels, I used the NEMO toolchain [98]. The NEMO toolchain is used to perform post-training quantization on the floating-point model to an 8 bits integer model after training. I also added dedicated quantized operators (e.g., Softmax) from I-BERT [103] to quantize the Self-Attention layers fully.

4.4.6 TinyRadar Transformer

As a benchmark, I use the TinyRadar network and its corresponding dataset [104] as a use case to demonstrate both the feasibility and the advantages of porting Transformers

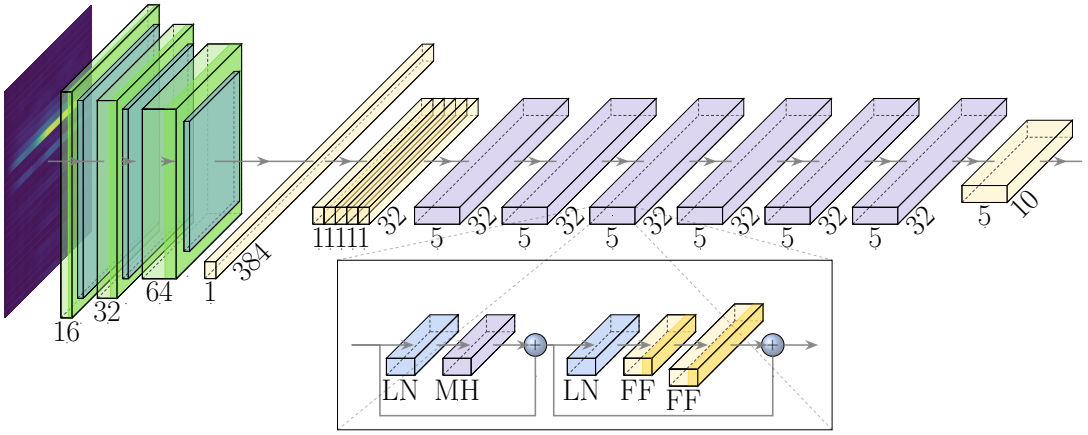

Figure 4.20: Overall architecture of the proposed network. The front end comprises three blocks of pointwise convolution, depthwise convolution, and pooling, followed by a Linear layer. Each of the six encoder blocks (in purple) consists of layer norm layers (LN), a Multi-Head Attention layer (MH), and Linear layers (FF).

on edge using the proposed kernels. TinyRadarNN has a CNN stage, a TCN stage, and a dense layer stage [104]. The TinyRadar dataset consists of a total of over 10000 recordings of 11 hand gestures by 26 people made with a short-range radar. In the original paper, the data is arranged in Spatio-temporal windows that contain samples of the radar reflection amplitude sampled after different amounts of time-of-flight, called range points, on the columns, with consecutive distance samples concatenated along the rows. The working principle of the network architecture is based on splitting spatial and temporal modeling of the gesture recognition problem.

I preserved this structure but replaced the TCN stage, which models the long-term temporal dependencies, with a more advanced 6-layer transformer encoder architecture. The proposed architecture is shown in Figure 4.20. A sequence of 5×32 inputs is fed to the Transformer backend. The 6 layers which constitute the backend are identical to the ones of [41], with $S = 5$, $E = 32$, $P = 32$, and $H = 8$. Finally, the output of the encoder is fed to a dense layer which is used as a classifier, returning a prediction for each time step.

Besides the changes in architecture, I down-sample the inputs by employing 1 sensor and 246 range points instead of 2 sensors and 492 range points to reduce the total number of operations without impairing accuracy.

4.4.7 Experimental Results

I will report the results of this new attention library on a Multi-Head Self-Attention layer with $H = 16$, $P = 64$, $E = 64$, and $S = 32$. As benchmarking platforms, I always

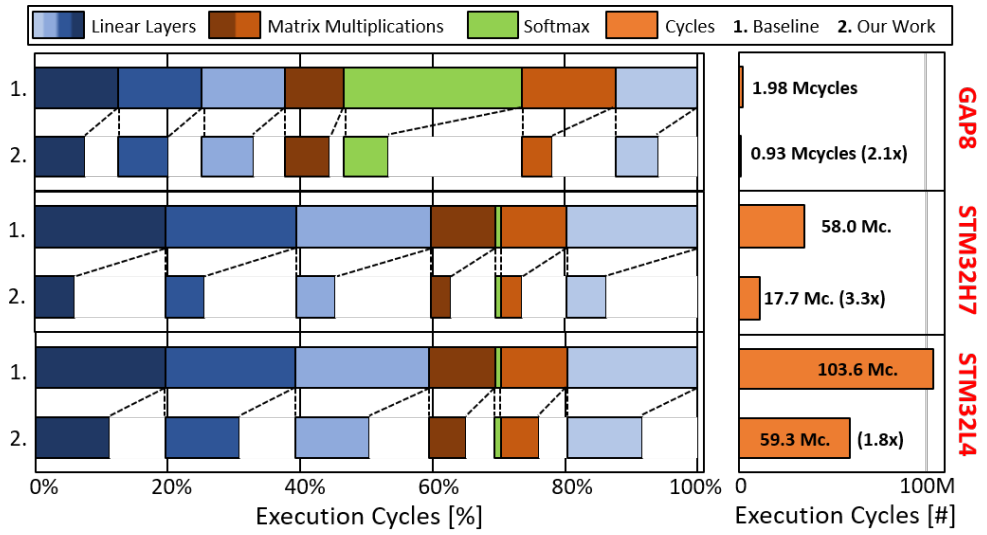

Figure 4.21: Performance description of baselines and proposed kernels on the three different platforms. The x scales are different for each platform given the extremely different upper limits.

use GAP8, STM32H7, and STM32L4. I set the operating frequencies of the GAP8, STM32H7, and STM32L4 at 100 MHz, 480 MHz, and 80 MHz, with a corresponding average power consumption of 51 mW, 234 mW, and 10 mW, respectively. I choose these three operating points as they are the most energy-efficient ones according to [105].

4.4.7.1 Kernel performance

Column 'My Work' of Table 4.7 details the performance of the library on top of the three MCUs. The layer analyzed with $H = 16$, $P = 64$, $E = 64$, and $S = 32$ has 0.48 MMAC and 262k parameters and thus fits the on-chip memory of all three platforms. Overall, the library allows Attention layers to achieve performance comparable to the best performing convolutional layers with both Instruction Set Architectures (ISAs), reaching 11.29 MAC/cycle and 0.61 MAC/cycle on GAP8 and STM32H7, respectively, compared to 12.86 MAC/cycle and 0.71 MAC/cycle for convolutions [105]. The benchmark layer runs on the three platforms in 929 kcycles, 59.4 Mcycles, 17.2 Mcycles, respectively, with a latency of 9.29 ms (GAP8), 35.77 ms (STM32H7), and 741.93 ms (STM32L4).

4.4.7.2 Comparison with the state-of-the-art

I also compare the deployment of a transformer architecture with state-of-the-art CNN libraries, CMSIS-NN [33], and PULP-NN [25]. In particular, I exploit these libraries' optimized Linear layer kernels as a base function for the matrix multiplications

and projection layers, adding extra external loops, the SoftMax operator, and the memory marshaling operators.

Table 4.7: Comparison of my kernel library with PULP-NN and CMSIS-NN onto three commercial MCUs.

MCU	1×RISC-V + 8×RISC-V		1×CortexM4		1×CortexH7	
Power [mW]/ Freq. [MHz]	51 mW / 100 MHz		10 mW / 80 MHz		234 mW / 480 MHz	
Kernels	My Work	PULP-NN+Reshape	My Work	CMSIS-NN+Reshape	My Work	CMSIS-NN+Reshape
Layer: Attention - Heads: 16, Projection: 64, Sequence: 32, Embedding: 64, Operations: 8.4M						
Cycles	929k	1.95M	59.4M	103.59M	17.17M	57.0M
Time/Inference [ms]	9.29	19.52	741.93	1294.92	35.77	118.74
Energy[mJ]	0.47	1.00	7.42	12.95	8.37	27.79
MACs/cycle	11.29	5.37	0.18	0.10	0.61	0.18
Throughput [GMAC/s]	1.13	0.54	0.014	0.008	0.29	0.09
En.Efficiency [GMACs/s/W]	22.13	10.53	1.41	0.81	1.25	0.38

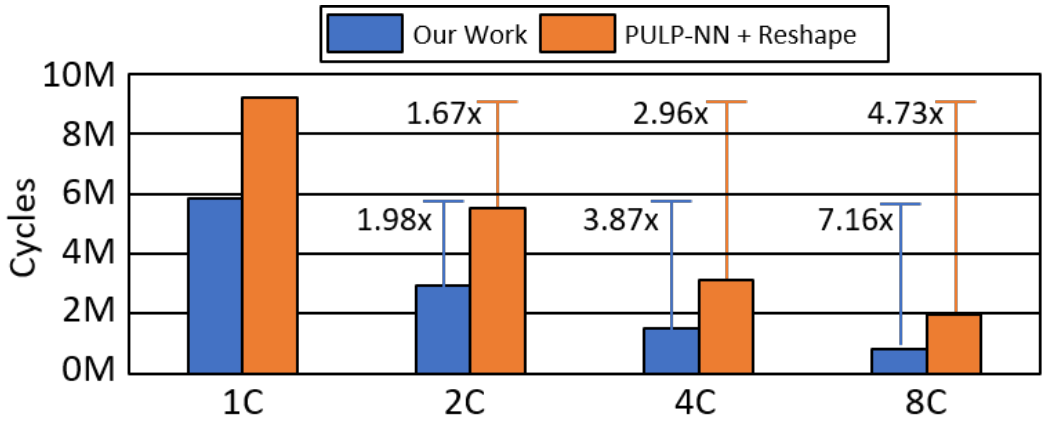

Figure 4.22: Parallelization of the attention layer with 1, 2, 4, and 8 cores on GAP8.

Fig. 4.21 depicts a detailed study on performance improvement of every single part of the Attention layer. On the STM32L4, GAP8, and STM32H7, I obtain a speedup of 1.8×, 2.1×, and 3.3×, respectively. However, the different sublayers demonstrate different speeds-up on the three platforms. Starting from GAP8, the three different components, Linear layers, matrix multiplications, and Softmax, demonstrate a speedup of 1.7×, 2.0×, and 4.0×, respectively. These speedups are due to better data reuse and the removal of reshaping layers. Furthermore, with the parallelization scheme on heads, with each core taking care of a portion of the heads of the Multi-Head Attention, the SoftMax execution can achieve a speedup of up to 8×. On the other hand, parallelizing on S requires synchronization of all the cores after the computation of a single parallelized sequence S of data. Fig. 4.22 details further the speedups on GAP8 of this library compared to the PULP-NN baseline. While the baseline only achieves 4.73× speedup with 8 cores compared to one, I can reach 7.16×, with an improvement of 1.51×.

On the STM32 platforms, the speedup is solely concentrated in Linear and matrix multiplication layers. Remarkably, despite using the same ISA, a dramatically higher speedup of 3.3× compared to 1.8× can be observed between the STM32H7 to the STM32L4. This difference is mostly given by exploiting the dual-issue pipeline and the cache refill on the H7. The kernels significantly boost data locality and reuse, allowing for better cache utilization than the CMSIS-NN baseline. To confirm this fact, disabling the caches reduces the relative speedup of the attention kernels compared to the baseline from 3.3× to 2.4×.

4.4.7.3 TinyRadar Transformer performance

Finally, I briefly discuss the proposed TinyRadar Transformer architecture. I report all accuracy values post-quantization at `int8` precision. The transformer-based network architecture achieves an accuracy of 77.15% on the TinyRadar dataset, outperforming

Table 4.8: Performance of the proposed transformer architecture at fr = 100MHz on GAP8 platform. Abbreviations: At.: Attention. FF: Linear layers in Transformer backend.

	PULP-NN + Reshape			Our Work		
	CNN	Att.	FF	CNN	Att.	FF
MACS [#]	1.87M	1.06M	185k	1.87M	1.06M	185k
Cycles	717.4k	261.6k	94.5k	717.4k	112.5k	94.5k
Lat. [ms]	7.17	2.62	0.95	7.17	1.12	0.95
E. [mJ]	0.37	0.13	0.05	0.37	0.06	0.05

the original architecture by 3.5% and a modified Temporal Convolutional Neural Network (TCN) architecture by ∼5%.

Note that this is achieved with a small network since the TinyRadar Transformer contains 263k parameters, which fits the L2 on-chip memory of GAP8. Tab. 4.8 reports the network's performance running on GAP8 at 100 MHz. The network achieves as low as 9.24 ms latency and 0.47 mJ, 9.6×, and 6.3× lower than the original TinyRadarNN. Specifically, using the new attention library together with DORY for tiling, I improve the performance of the Attention part by 2.32×, with a 14% direct reduction of the overall cycles of the network.

Chapter 5

Biosignal analysis with deep neural networks on the edge

I present in this chapter two main applications of the "technological" tools exposed in the previous two chapters: the first is the heart-rate tracking based on DNN and the second one is the application of transformers to sEMG-based gesture recognition.

5.1 Q-PPG: Energy-Efficient PPG-based Heart Rate Monitoring on Wearable Devices

In this section, I will describe the application of the NAS and deployment algorithms to heart rate tracking. In particular, I will show a Temporal Convolutional Network which is optimized with both the NASes described to minimize the memory and amount of operations weighted for the data type of tensors.

5.1.1 Q-PPG Exploration Flow

The design space exploration achieved through NASes offers various MAE and computational cost trade-offs. The latter is measured in terms of trainable parameters or operations per inference. The inputs of the flow I will present are a training dataset containing PPG and inertial data associated with the corresponding HR label and a so-called *seed* TCN. This "template" network is the one from which all output models are generated through PIT and the channel-wise precision NAS. I successively applied them in 2 phases:

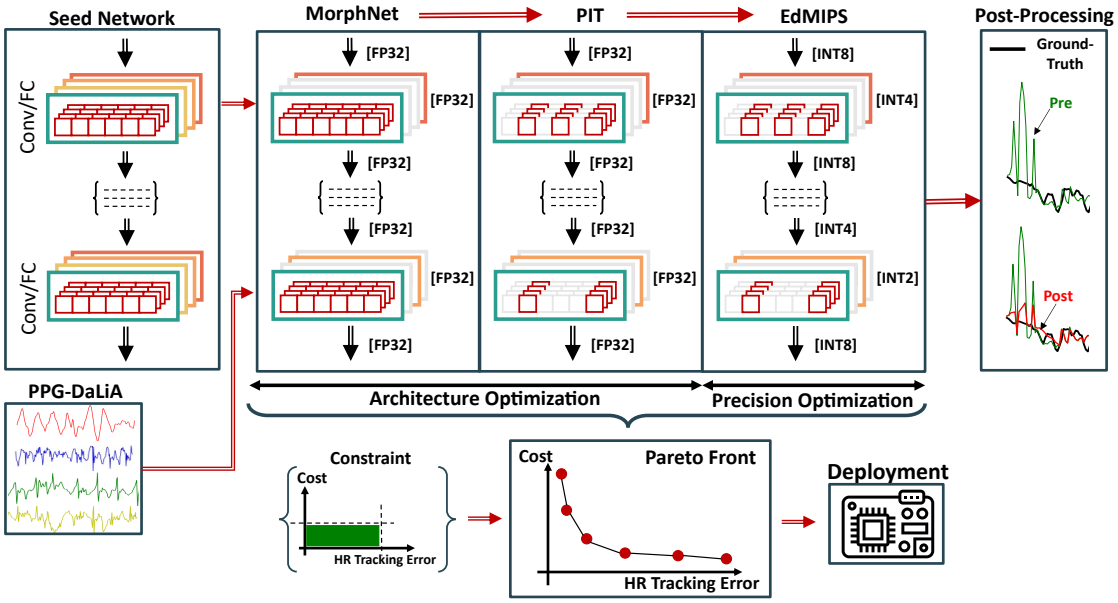

Figure 5.1: Proposed Q-PPG design space exploration flow.

1. *Architecture Optimization*: in this phase, I leverage PIT to trade off computational cost and performance on the template TCN, searching for the optimal number of filters for each layer.

2. *Precision Optimization*: in the second phase, starting from some of the points found by PIT, I enrich the Pareto curve by applying different types of quantization [81] to the weights and activations of the TCNs.

To reach state-of-the-art accuracy, I also apply a low-cost post-processing step to the final TCNs. A high-level diagram of the entire flow is shown in Figure 5.1. Since its final output is a set of quantized TCNs, I name this methodology *Quantized-PPG* (Q-PPG).

Notably, the lowermost part of the picture shows that the Q-PPG exploration has to be performed *only once* for each seed model. Then, the user can pick one point from the Pareto space if different hardware is involved. Specifically, the target platform usually constrains the maximum memory available. Then, the most accurate Q-PPG model that meets those constraints is selected and deployed. Therefore, generating an entire *family* of models rather than a single one makes this methodology efficient and flexible, enabling the deployment of optimized HR tracking solutions not only on a single platform but also on other similar wearable-class systems.

5.1.1.1 Input Data and Seed Network

The Q-PPG exploration phase and the training of the final TCNs use the same input dataset, which is composed of raw sensor data gathered from the PPG sensor and a tri-axial accelerometer. Training samples are partitioned in sliding windows of length T on the four signals. Therefore, the TCNs take a 2-dimensional array of size $(T, 4)$ as input. The target output for training is the ground truth HR estimate, expressed as a scalar real number in BPM. The task to be solved by the NN is indeed a *regression* problem, i.e., the objective of the TCNs is to approximate the HR ground truth value. I use the LogCosh loss function in all training runs to measure the error between the real and predicted HR during training. Note that I verified that the LogCosh outperforms both RMSE and MAE [106] as a loss function, favoring the convergence near the minimum, thanks to its smoother behavior around that point.

The other fundamental input for the design exploration is the *seed* network. All Q-PPG outputs are obtained starting from the seed, varying the number of filters (i.e., the structure) or the tensor precision to trade off computational cost and HR tracking performance. In particular, the Architecture Optimization phase of Q-PPG tries to *reduce/simplify* the seed while maintaining the MAE as low as possible. Therefore, for the flow to cover a rather ample design space, the starting point should be a relatively large and accurate TCN. Specifically, in this work, I used the same seed network used in PIT, i.e., an adapted version of TEMPONet [34]. Compared to the original version, the structure of TEMPONet widens the space explored by reducing the dilation and increasing the number of channels.

Therefore, I first modified the network for compatibility: the number of input channels of the first layer is adjusted to 4 to match the number of features of the input dataset. The last FC layer is modified to change the number of units to 1, as required when performing a scalar regression task. Lastly, the dilation parameters of all convolutional layers in TEMPONet have been set to $d = 1$, while the filter size K has been increased to match the original receptive field. As said, I did these modifications to improve the search space, which encopasses many combinations of K, d, and channels.

5.1.1.2 Architecture Optimization

This section describes how to use the different NAS tools to generate different TCN architectures for HR tracking in the accuracy vs. complexity space. At the top of Figure 5.1, I show the cascade of two different Neural Architecture Search (NAS) tools. The first is called MorphNet [16] for the number of channels exploration, and the second

is PIT. I use PIT to optimize only the d parameter in this exploration, while I do not optimize d.

A summary of the functionality of the two tools used is shown in Figure 5.2. As previously explained, before starting the search, the layers of the seed network are augmented with the different sets of dedicated *masks* (α_i and β_i in the figure), each of which multiplies a subset of the layer's weights. β_i masks are applied only to convolutions to tune dilation, while α_i masks control the number of channels.

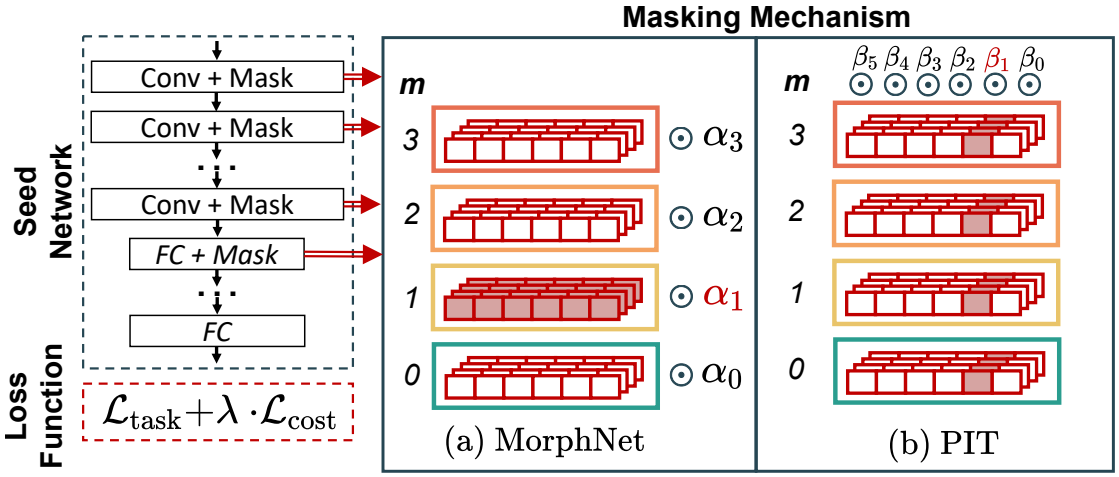

Figure 5.2: High-level scheme of the functionality of the two NAS algorithms used for architecture optimization. Pooling and other layers are not shown for simplicity.

As shown in Figure 5.2 masks are forced to small values during training by adding to the normal loss function $\mathcal{L}_{\text{task}}$ (i.e., LogCosh for HR tracking) the additional *regularization term* $\mathcal{L}_{\text{cost}}$ explained in Sec. 2.2. The different Pareto points obtained in the complexity versus HR tracking error are derived by tuning the constant λ, which regulates the relative importance of the two loss terms.

Instead of solely using PiT, I first used Morphnet to tune the channels. As shown in Figure 5.2, MorphNet [16] indeed masks all weights relative to the same convolution output channel with one α_i. In contrast, I use PIT [107] masks to only tune d. All weights corresponding to the same time-step (and to all output channels) are multiplied by one β_i, with the effect of inserting "holes" in the convolution filters. To clarify the masking process, all weights multiplied with α_1 and β_1 are coloured in red in Figure 5.2a and 5.2b respectively. As shown, α_1 is multiplied with all the weights of filter $\mathbf{W}^1$ (i.e., the filter that comprises weights used to derive output channel C_1). In contrast, β_1 is multiplied with the 1-indexed columns of *all* filters, assuming these are stored in channel-major order. I suggest readers refer to Chapter 3 for the complete discussion of these tools.

Search Protocol I select MorphNet and PIT for the architecture optimization because both the number of channels and the dilation are key parameters that influence the accuracy and complexity of TCNs [11]. In the experiments for this task, I found empirically that running MorphNet first, followed by PIT, yields much better results than the opposite ordering. This phenomenon happens because MorphNet operates in a broader and more fine-grained search space since the possible channel combinations are way more than the regular dilation values in a typical convolutional layer.

Given this observation, I used the following scheduling of algorithms. First, I apply MorphNet to the seed network, with different regularization strengths (from $\lambda = 10^{-6}$ to $\lambda = 10^{-3}$). This results in a first Pareto frontier composed of TCNs with a different number of channels and dilation fixed at 1. Then, I selected some key points from this frontier: specifically, I took the two extremes of the curve (i.e., the TCN achieving the minimum HR tracking error on the validation set and the one with the lowest cost), plus two intermediate solutions to span the space homogeneously. In the second step, taking these 4 networks as seeds for PIT, I repeated the training with different regularization strengths (from $\lambda = 10^{-9}$ to $\lambda = 5 \cdot 10^{-3}$). Consequently, the output of the MorphNet + PIT chain includes 4 (in general, n) sets of TCNs, which are then combined to obtain the final Pareto front. Each NAS execution is preceded by a warm-up phase and followed by a fine-tuning, where only the weights of the seed/optimized TCN are trained to improve the overall accuracy.

5.1.1.3 Precision Optimization

Starting from the architectures generated by the two cascaded NAS tools, I further expand the solutions space by exploring the per-layer arithmetic precision of the TCNs. The quantization technique I use is the same adopted in Sec. 4.2, which was shown to maintain high accuracy even with sub-byte precision while also being hardware friendly. Different from other techniques such as weight clustering [102], this method replaces all floating point multiply-and-accumulate (MAC) operations required for inference with integer MACs. With integer operations, the inference results in a more efficient execution and enables the deployment of the resulting models on hardware without a Floating Point Unit (FPU). The method implements a *linear quantizer*, which transforms each floating point tensor $\mathbf{t}$ (of either weights or activations), with values in the range $[\alpha_\mathbf{t}, \beta_\mathbf{t})$ into a N-bit integer tensor $\widehat{\mathbf{t}}$ as:

$$\widehat{\mathbf{t}} = \text{round}\left(\frac{\mathbf{t} - \alpha_\mathbf{t}}{\varepsilon_\mathbf{t}}\right) \tag{5.1}$$

where $\varepsilon_\mathbf{t} = (\beta_\mathbf{t} - \alpha_\mathbf{t})/(2^N - 1)$ is the smallest value that can be represented in the quantized tensor. The entire inference is then performed using only integer data. Specifically,

the accumulation of partial outputs is performed with `int32` data so that no overflows occur.

Note that every quantization algorithm can be applied to a NN either post-training [2] or using quantization-aware training (QAT) [80, 81].

In Q-PPG, I use QAT, by means of EdMIPS [80], a tool that allows to simultaneously *i)* perform QAT and *ii)* search for the optimal trade-off between each layer's data format and the network's final error. The basic principle of QAT is simulating the effect of quantization (so-called *fake quantization*) during the forward pass of each training iteration while maintaining floating point updates during back-propagation. Figure 5.3 illustrates the functionality of EdMIPS, which relies on a gradient-based optimization method very similar to the one used by the two NASes described in Section 5.1.1.2. Note that I decided to use EdMIPS and not the channel-wise precision search NAS since the latter is more complicated and leads to a much bigger design space. On the other hand, to demonstrate the applicability of the tools to real-life problems, using EdMIPS already shows a big improvement and an extended trade-off between accuracy and performance.

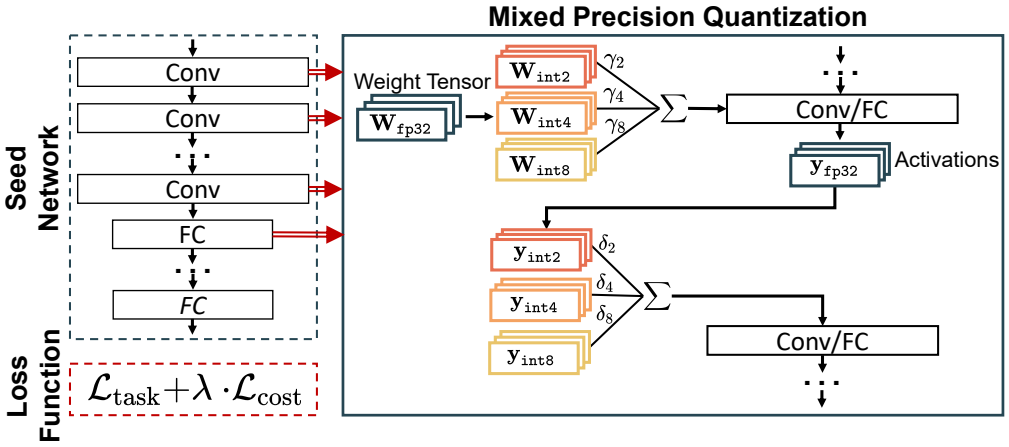

Figure 5.3: EdMIPS flow for arithmetic precision optimization.

Using EdMIPS, all convolutional and FC layers in the network are replaced by meta-layers, identical in terms of the executed operation, but whose weights are obtained as combinations of fake-quantized tensors with different precision. For instance, the 1D-convolution equation is changed to:

$$\mathbf{y}_t^m = \sum_{i=0}^{K-1} \sum_{l=0}^{C_{in}-1} \mathbf{x}_{t\,s-d\,i}^{l} \cdot \widehat{\mathbf{W}}_i^{l,m} \tag{5.2}$$

where:

$$\widehat{\mathbf{W}} = \sum_{p=0}^{P-1} \mathbf{W}\mathbf{q}_p \cdot \gamma_p \tag{5.3}$$

and P is the number of different precision formats considered. $\mathbf{Wq}_p$ is the tensor of fake-quantized weights using the p-th precision, and γ_p is a trainable coefficient associated to it. For instance, if I consider `int2`, `int4`, and `int8` formats, then $\widehat{\mathbf{W}} = \mathbf{Wq}_{\texttt{int2}} \cdot \gamma_{\texttt{int2}} + \mathbf{Wq}_{\texttt{int4}} \cdot \gamma_{\texttt{int4}} + \mathbf{Wq}_{\texttt{int8}} \cdot \gamma_{\texttt{int8}}$. All fake-quantized tensors are obtained from a single, shared set of floating point weights $\mathbf{W}_{\texttt{fp2}}$. A similar transformation is also applied to the outputs of the layer to search for the optimal quantization format for activations. Specifically, the output of the meta-layer is obtained by combining fake-quantized activations as follows:

$$\mathbf{y} = \sum_{p=0}^{P-1} \hat{\mathbf{y}}_p \cdot \delta_p. \tag{5.4}$$

As for all NAS approaches described, γ and δ coefficients are then trained together with the network weights, adding a secondary loss $\mathcal{L}_{\text{cost}}$ that takes into account the cost of each data format. The training algorithm then assigns a more significant coefficient to the fake-quantized tensor, offering the best trade-off between cost and performance.

Search Protocol Within Q-PPG, I apply EdMIPS with the following strategy. First, I perform a *uniform* quantization, i.e., using the same bit-width for all tensors ($P = 1$), to the entire set of TCNs obtained in the architecture optimization phase. I then repeat the QAT with different formats, namely `int2`, `int4`, and `int8`, which are those supported by the backend TCN inference library available for the target hardware [2]. Next, to explore all the possible trafe-odds, I use the tool to search for the best bit-width for each tensor, exploring so-called *mixed-precision* networks [80].

5.1.1.4 Post-processing

The last component of the complete chain of transformation applied is a post-processing step used at runtime to the output of the optimized TCNs. This step is orthogonal and independent of the design space exploration described above. It is motivated by the fact that data-driven models such as TCNs, while very accurate on average, may sometimes incur significant and unpredictable errors, primarily when the processed inputs differ significantly from those seen in the training phase. Specifically, the human heart rate dynamics impose an upper bound on the reasonable variation of the estimate over time in normal conditions. Therefore, when performing continuous HR tracking (e.g., every 2s in the experiments described in Section 5.1.2), a single TCN prediction that differs significantly from all its predecessors is likely due to an error of the model.

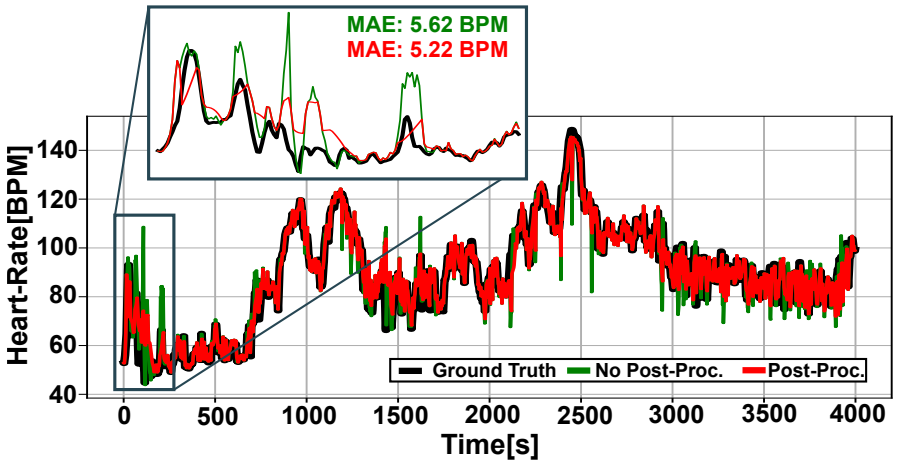

Figure 5.4: HR tracking obtained with the best performing Q-PPG output TCNs before and after post-processing on the subject n.3 of the Dalia dataset.

Based on these considerations, the used post-processing applies a simple filtering, where the latest TCN prediction HR_n are compared with the average of the previous N=10, $E_{n,N} = E[HR_{n-1}, ..., HR_{n-N}]$. If the difference between these two values is larger than a threshold P_{th}, the estimate is *clipped* to $HR_n = E_{n,N} \pm P_{th}$.

5.1.1.5 Fine-Tuning

In one of the experiments, I also considered partially personalized per-subject models instead of population ones. Specifically, after training the models with data of subjects not included in the test-set, I apply a further *fine-tuning* step with a lower learning rate, using the initial 25% of the data relative to the subject under test. The MAE is then computed on the remaining 75% of the data. I applied this technique to match state-of-the-art results and show the improvements on top of them.

5.1.2 Results

In this section, I present experimental results to demonstrate the effectiveness of the proposed methodology for building accurate yet efficient HR tracking solutions based on TCNs. Specifically, Section 5.1.2.1 describes the training protocol. Section 5.1.2.2 shows the results of the *architecture optimization* phase of the flow, which is a set of TCNs with different error and complexity characteristics but still using a floating point data format. These networks are then compared with the state-of-the-art in Section 5.1.2.3, since all existing algorithms tested on PPG-Dalia use float data. Next, Section 5.1.2.4 shows how the error and complexity results change after the *precision optimization* phase. Lastly, Section 5.1.2.5 reports the memory footprint, energy consumption, and latency obtained

deploying some of the final Q-PPG outputs on the platform described in Section 2.3. The proposed flow and all TCN training codes are implemented in Python 3.6. To deploy TCNs on a STM general purpose MCU, I use the open-source Cmix-NN inference library for ARM processors presented in [2], which supports mixed-precision layers with `int2`, `int4` and `int8` formats for weights and activations.

5.1.2.1 The PPG-Dalia Dataset

As a dataset, I used the previously introduced *PPG dataset for motion compensation and heart rate estimation in Daily-Life Activities* (PPG-Dalia) [36]. I validate all models following the cross-validation protocol proposed in [36], denoted as Leave-One-Session-Out (LOSO) cross-validation, where the 15 subjects are organized in four randomly picked data folds. Three folds are used as the training set, while the remaining one is subdivided to form the test set, composed of a single subject and the validation set. Fifteen training iterations are then performed, each with a different test subject, ensuring that its input data are never used to train the model tested on it.

I also compare Q-PPG solutions against both classic and DL methods that have been tested on the same dataset, taking their results directly from the original papers. When analyzing MCU deployments, I consider a real-time constraint of 2s per inference, equal to the time shift between two consecutive samples in the dataset, following previous work [36, 108].

5.1.2.2 Architecture Optimization Results

Figure 5.5 reports the first obtained results thanks to the architecture optimization phase. Specifically, it shows (in green) the Pareto frontiers defined by the different TCN variants discovered by Q-PPG, changing the regularization strength of the two NAS algorithms. Results are reported as MAE versus the number of trainable parameters and MAE versus the number of operations. They include the effect of the runtime post-processing described in Section 5.1.1.4. Note that these models are still not quantized; therefore, all the tensors are in floating-point format.

Besides the outputs of the complete flow, four other results are reported for comparison. The black diamond and triangle correspond respectively to the original TEMPONet, with the dilation of convolutional layers set as in [34], and to the TEMPONet variant with all dilations set to 1, which corresponds to the input seed of Q-PPG. Comparing these two points with the green curve clearly shows that: i) using a hand-tuned TCN initially designed for another task, such as TEMPONet, would be suboptimal, and

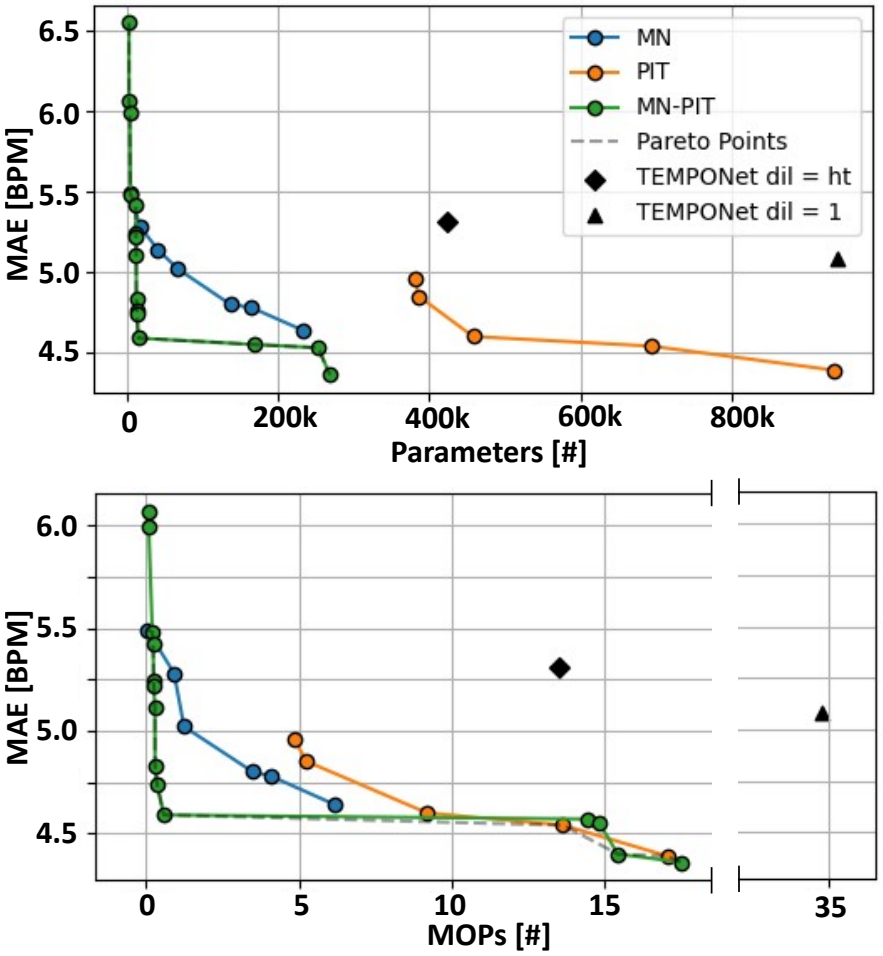

Figure 5.5: Architecture optimization results in the MAE versus n. of parameters and MAE versus Millions of Operations (MOPs) planes. The curve labeled "MN-PIT" corresponds to the sequence of MorhpNet (MN) and PIT used in the proposed Q-PPG flow.

ii) the two NAS algorithms can simultaneously improve the HR tracking performance of the seed, while also dramatically reducing its complexity.

The other curves reported in Figure 5.5 show the results of the isolated application of the two NAS algorithms. The blue points refer to the application of MorphNet (MN) to TEMPONet with hand-tuned dilation. Orange points, instead, correspond to the application of PIT alone, using the TEMPONet variant with $d = 1$ as seed. Comparing these curves with the one with points obtained after applying both NASes shows that it is almost always superior to apply both than to apply only one. In fact, the global Pareto frontier (gray dashed line) is almost always overlapped with the output of the two NASes search. For instance, MorphNet alone can explore a vast space of solutions by tuning the number of channels in each layer but is forced to use sub-optimal hand-tuned dilation values.

Overall, the automatic design space exploration technique proposed in this thesis can span more than two orders of magnitude, both in terms of TCN parameters (3.5k-269k) and OPs (0.1M-17.5M), despite starting from a single seed TCN. The most accurate model obtained achieves an MAE of just 4.36 BPM while requiring around 269k parameters and 17.5M OPs. On the other hand, by only using Morphnet, the best MAE obtained is 4.88 BPM (+0.52 BPM), with a similar number of parameters (230k) and operations (12M). Noteworthy, increasing the number of parameters from 3.5k up to 30k leads to improving the MAE from 6.5 BPM to 4.55 BPM. This huge improvement indicates that the NAS-based flow finds near-optimal models with relatively few parameters, increasing the accuracy for each new parameter added.

5.1.2.3 State-of-the-art comparison

Figure 5.6 compares the searched models (green) with state-of-the-art algorithms, including both classical and deep learning solutions (blue and red points, respectively), in the MAE versus the number of operations space. For Q-PPG, I report the entire Pareto frontier of TCN architectures, i.e., the same points plotted in the lowermost graph of Figure 5.5. The comparison is made with the state-of-the-art considering floating point models. The details of the cross-validated MAE results for each of the 15 PPG-Dalia subjects are reported in Table 5.1. For works proposing multiple models (ours and DeepPPG [36]), the table reports the results of the most accurate one. Q-PPG results are reported both with and without post-processing.

Table 5.1: Comparison with state-of-the-art PPG-based HR monitoring algorithms.

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	Mean
Classical Models																
Schack2017 [109]	33.1	27.8	18.5	28.8	12.6	8.7	20.65	21.8	22.3	12.6	21.1	22.8	27.7	12.1	16.4	20.5
SpaMaPlus [110]	8.86	9.67	6.40	14.10	24.06	11.34	6.31	11.25	16.04	6.17	15.15	12.03	8.50	7.76	8.29	11.06
TAPIR [108]	4.50	4.50	3.20	6.00	5.00	3.40	2.80	6.30	8.00	2.90	5.10	4.70	3.10	5.00	4.10	4.57
CurToSS [111]	5.40	4.30	3.00	8.00	**2.20**	**2.80**	3.30	8.50	12.60	3.60	3.60	6.10	3.00	5.50	3.70	5.04
Deep Learning Models																
DeepPPG [36]	7.73	6.74	4.03	5.90	18.51	12.88	3.91	10.87	8.79	4.03	9.22	9.35	4.29	4.37	4.17	7.65
NAS-PPG [112]	5.46	5.01	3.74	6.48	12.68	10.52	3.31	8.07	7.91	3.29	7.05	6.76	3.84	4.85	3.57	6.02
OurWork, Best MAE	4.29	3.62	2.44	5.73	10.33	5.26	**2.00**	7.09	8.60	3.09	4.99	6.25	**1.92**	3.02	3.55	4.81
+ Post-Processing	3.78	3.36	**2.33**	4.84	9.95	4.38	2.20	5.88	7.59	**2.74**	4.55	5.20	2.14	**2.99**	3.47	4.36
+ Fine-Tuning	**3.25**	**2.55**	2.66	**4.21**	5.41	4.11	2.06	**5.07**	**7.15**	3.04	**3.07**	**3.39**	2.13	3.13	**2.96**	**3.61**

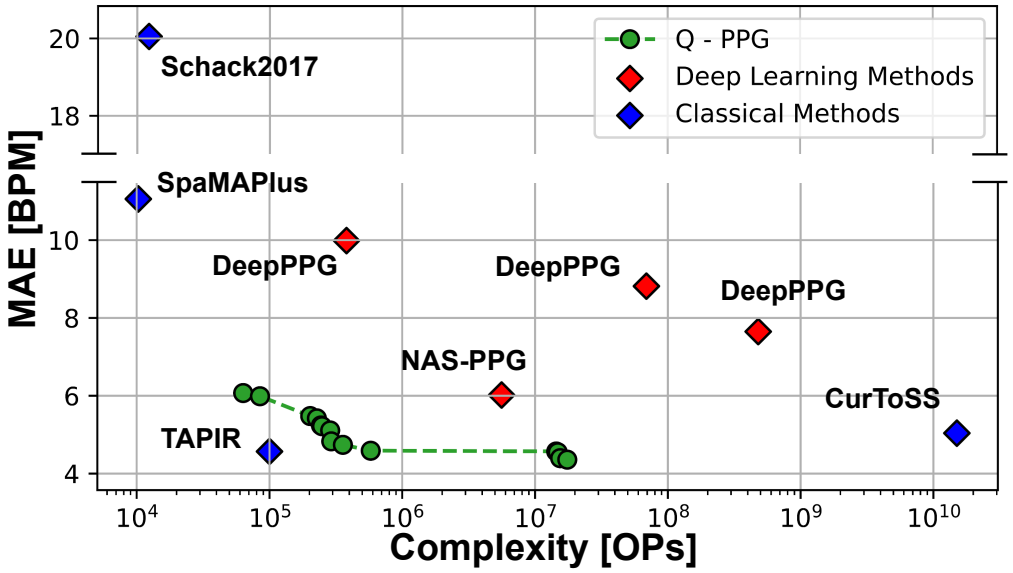

Figure 5.6: Comparison with state-of-the-art algorithms in the MAE versus number of operations space.

As shown, Q-PPG significantly outperforms *all* previous solutions based on deep learning. Compared to DeepPPG, the first approach introducing DL for HR, even with the *simplest* model, I achieved better performance (6.07 vs. 7.65 BPM) and a striking 7572× reduction in complexity. Compared to the recent NAS-PPG, the same Q-PPG model obtains comparable MAE (6.07 vs. 6.02 BPM) with 88.4× fewer operations. Moreover, the best Q-PPG model obtains an MAE that *outperforms all the previous state-of-the-art* methods for this dataset, including TAPIR [108], although at the cost of higher complexity, achieving an average error of just 4.36 (vs. 4.57) BPM. TAPIR is not dominated in the Pareto sense due to its low theoretical complexity. However, it is essential to note that the hand-tuning of the parameters strongly influences this method. Indeed, TAPIR performs poorly with slight parameter modifications and needs different parameter tuning for different datasets as for other classical methods.

Looking at Table 5.1, it is evident that the best TCN performance is strongly impaired by subject 5. This is because this subject's record contains very high HR values, rarely encountered in training data, which are badly predicted by data-driven approaches. Applying the fine-tuning step described, the MAE of the best Q-PPG model further reduces to just 3.61 BPM, given that the MAE of Subject 5 decreases from 9.95 to 5.41 BPM. Overall, fine-tuning improves the performance of other 10 out of 14 subjects.

5.1.2.4 Precision Optimization

Figure 5.7 shows how the MAE versus model size results change when applying different types of quantization to Q-PPG models, enlarging the exploration space. Note that the x-axis of the curve now reports the model size in bytes rather than the number of parameters, and the dark green curve corresponds to the one in the topmost graph of Figure 5.5. The graph reports all the results obtained with uniform and mixed-precision quantization.

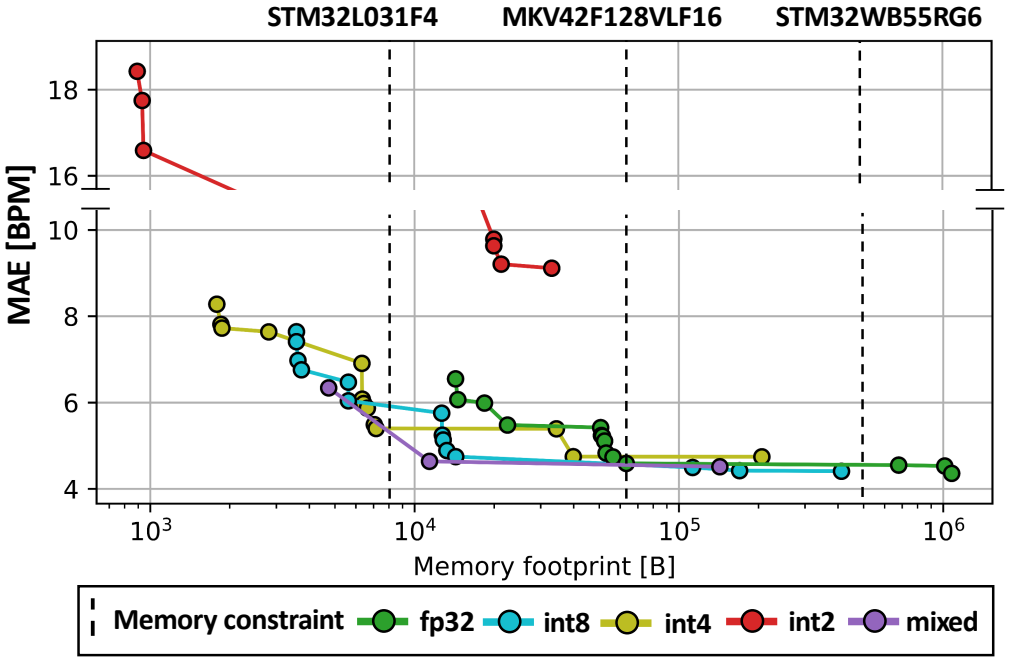

Figure 5.7: MAE versus memory occupation of Q-PPG TCNs quantized with different data formats.

A first important observation is that uniform int8 quantization incurs minimal MAE degradation, as shown by the similar shapes of the dark green and blue curves, despite reducing a factor of 4 in model size. Furthermore, both sub-byte uniform quantization (int4 and int2) and mixed precision have solutions that fall on the global Pareto frontier, demonstrating that all formats are practical in different regions of the design space. Thanks to quantization, the range of model sizes obtained reaches 3 orders of magnitude, from around 1MB (most giant float TCN) to less than 1kB (smallest int2-quantized TCN), with MAE values ranging from 4.36 to ≈20 BPM.

The figure also reports, in the form of vertical dashed lines, the constraints imposed by the Flash memory available in 3 different commercial MCUs. The rightmost line corresponds to the target platform employed (the STM32WB, with 1MB of Flash), whereas the other two correspond, from right to left, to a MKV4 MCU from NXP, based on a Cortex-M4 with 128kB of Flash [113] and to the STM32L031F4, equipped

with a Cortex-M0+ and 16kB of Flash [114]. Typically, an MCU installed on a wearable device has to store the code and data for multiple applications in Flash. Since the specific application set varies from product to product, I assumed that 50% of the total Flash could be devoted to storing the TCN. While the actual constraint may differ in practice, this is just an example to demonstrate a valid principle in general. Once the constraint is defined, picking the best Q-PPG model for given hardware reduces to finding the most accurate Pareto point which respects the constraint. In particular, on the STM32WB I fit the most accurate *quantized* model overall, which requires ≈412 kB of memory, and achieves an MAE of 4.41 BPM.

5.1.2.5 Deployment Results

In this section, I discuss the results obtained deploying three representative TCNs obtained with Q-PPG on the wearable device described in Sec. 2.3.1. Specifically, I deployed the smallest networks with less than 8 BPM and 5 BPM of MAE, (Q-PPG-S and Q-PP-M, respectively), as well as the most accurate of all quantized networks (Q-PPG-L). All deployments have been performed using the Cmix-NN layers library [2], which has been adapted to support 1D convolutions with dilation.

The memory occupation, latency, and energy consumption of the three networks for a single inference are reported in Table 5.2, which also shows the type of quantization used by each of them. As expected, smaller models are associated with a larger MAE. However, Q-PPG-M achieves better latency and energy results with respect to Q-PPG-S. This is because Q-PPG-M also includes `int8` layers, which have higher performance than `int4` in Cmix-NN, since the latter require more complex packing/unpacking operations to match the bit-width of Cortex-M vector ALUs. Interestingly, Q-PPG-M trades-off just 0.23 BPM of MAE for a 36.2× (34.2×) memory (energy) reduction compared to Q-PPG-L.

Table 5.2: Deployment of different Q-PPG networks on the STM32WB55 using Cmix-NN layers [2].

Model	Memory [B]	Latency	Energy	MAE
Q-PPG-S (int4)	1866	71.6 ms	1.79 mJ	7.73 BPM
Q-PPG-M (mixed)	11388	55.7 ms	1.39 mJ	4.64 BPM
Q-PPG-L (int8)	411997	1.90 s	47.65 mJ	4.41 BPM

To compute the power consumption of the entire wearable platform, I also characterized the two sensors (PPG and accelerometer) and the MCU during data gathering, data communication, and inference, as shown in Table 5.3. This is useful to understand how much different networks can impact the total energy consumption of the application

Table 5.3: Energy consumption of the three main components of the system during in phases.

	MAX30101 [mW]	LSM6DSM [mW]	STM32WB [mW]
Inference	5.5 [18.0%]	0.03 [0.1%]	25.0 [81.9%]
Data Comm.	5.5 [28.6%]	0.03 [0.2%]	13.7 [71.2%]
Data Gath.	5.5 [99.3%]	0.03 [0.5%]	0.008 [0.2%]

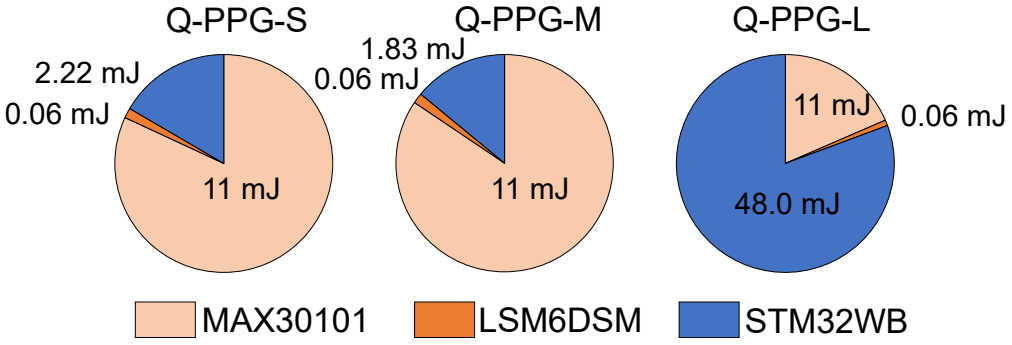

Figure 5.8: Break-down of the energy consumed in the 2s between two successive HR estimations, including data communication, algorithm execution, and waiting time for new data.

running on the edge. The STM32WB stays in *Stop* mode (0.008mW power) between the end of the computation and the moment in which the next window of data is ready to be acquired (gathering phase). After that, it goes in *Idle* mode (13.7mW power) during the data communication phase, enabling only the DMA and the SPI/I2C peripherals. Lastly, it goes in *Active* mode (25.0mW power) only to perform inference. Note that during the data gathering phase, the power consumption is strongly dominated by the MAX30101 power. However, I do not focus on power-saving techniques for the system's sensing parts but only on the minimization of the signal analysis part.

Figure 5.8 reports the energy breakdown of the system in a 2s window (the interval between two HR predictions) for the three networks of Table 5.2 chosen for deployment. SPI/I2C data acquisition and DMA transfers from peripherals to main memory require 15.4ms for both the PPG signal and the acceleration (considering a sampling rate of 32Hz), leading to stable energy consumption of 11 mJ and 0.06 mJ, respectively. Conversely, the execution time for inference ranges from 71.6 ms to 1.9 s, always meeting the real-time constraint of 2s. In particular, by using the Q-PPG-L network as a predictor, I obtained an energy consumption of 48.0 mJ, which is 81.3% of the total consumption of the system. On the other hand, trading off a bit of performance for a lighter network, using the Q-PPG-M network, the energy consumption for inference falls to just 1.83 mJ, which is only 14.1% of the total.

5.2 Bioformers: Embedding Transformers for Ultra-Low Power sEMG-based Gesture Recognition

In this section, I will describe a second application of DNNs to bio signals. In details, I will show the application of attention kernels deployed with DORY on GAP8 to the Bioformer, an architecture designed for gesture recognition using surface EMG data.

This paragraph deeply describes Bioformer, a Vision Transformer (ViT) [41] inspired architecture, thought to significantly reduces the computation complexity for sEMG-based gesture recognition while reaching an accuracy comparable with the state-of-the-art. Simultaneously, I also show a new pre-training protocol to feed more data to the transformer for gesture recognition, inspired by the pre-training of recent ViT architectures.

5.2.1 Bioformer: Network Topology

The network analyzed comprises three modules. First, the input signal is projected onto a space of dimensions $N \times 64$ using a 1D-convolutional layer. I use padding = 0 and stride equal to the filter dimension to aggregate non-overlapping windows of the input signal. Similarly to what is done in ViT for images, the idea is to create a series of N tokens of dimension 64 that encode the input information. This 1D layer creates 1D temporal patches, similar to the image patches of ViT. I tested [1, 5, 10, 20, 30] for the filter dimension. Note that the higher the dimension, the smaller the number N of produced tokens (and therefore, the lower the complexity of the following attention blocks), but the higher the information fused in a single layer. Compared to standard transformers [40], tuning the dimension of this first layer increases the architecture's flexibility, allowing for a trade-off between the total number of operations and the accuracy.

After this small block, the output is processed by the self-attention section. In the rest of the section, I focus on two variants of the Bioformer architecture, both of which exhibit good accuracy on sEMG-based gesture recognition. The parameters of the two networks are all identical except for the number of heads and the number of layers (depth). The first network comprises one attention layer with eight heads, while the second consists of two attention layers with two heads each. These two parameters have been chosen after performing a grid search on depth $\in \{1, 2, 3, 4\}$ and heads $\in \{1, 2, 4, 8\}$. I chose the architectures with the best trade-off of accuracy vs. parameters. The hidden space has dimension 128, while each head has a size P of 32. Similarly to [41], a "class token" is concatenated after the QKV projection step, adding one sample to the

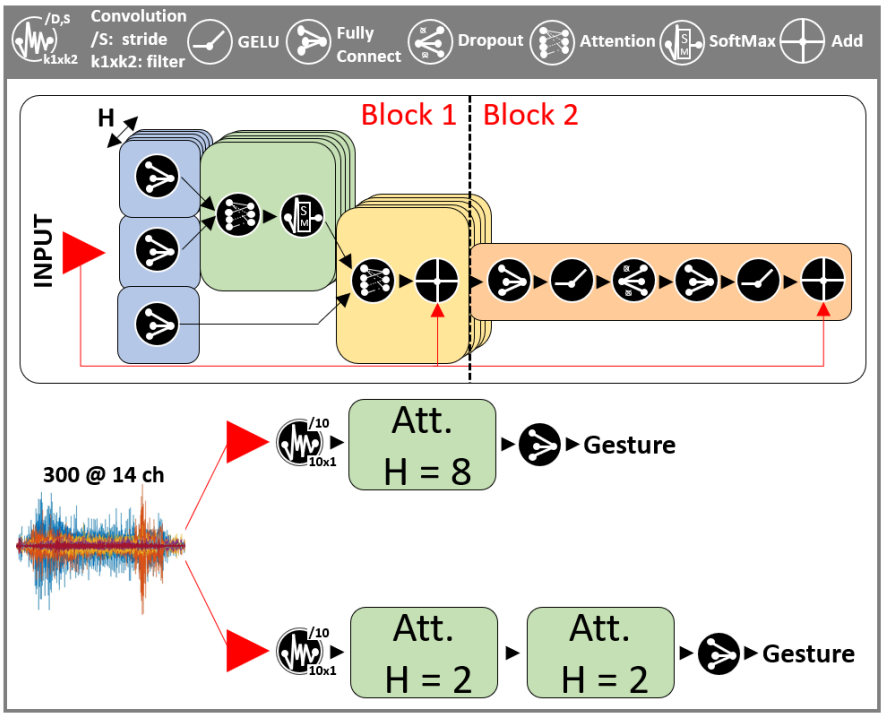

Figure 5.9: In the upper part, the basic MHSA layer used inside the architectures. In the lower part, the two Bioformers architectures that I propose as benchmarks.

sequence length ($(N+1) \times 64$). Noteworthy, these layers are much smaller compared to the state-of-the-art TEMPONet [34], which is constituted by a total of 9 convolutional layers and three fully connected layers. This dissimilarity demonstrates the attention layers' higher symbolic power than classical convolutions in gesture recognition. The final step takes the output "class token" and applies a linear layer with nine neurons per gesture. The SoftMax operator provides the probability for each gesture. Opposite to sequence-to-sequence transformers, where the output takes into account all the output tokens, the class token can be seen as the one that *pays attention* to the relevant elements in the sequence for the classification.

The lower section of Fig. 5.9 summarizes these two network architectures.

5.2.2 Bioformer: Training

Regardless of the employed dataset, the standard training for sEMG gesture recognition is subject-specific, given that the movements and muscle contractions associated with different gestures can differ significantly from one subject to another [34, 39]. On the other hand, it is known that performing a pre-training step on data similar to the ones used for the final training is highly beneficial for DL models, and in particular, for Transformers [12]. For instance, many state-of-the-art image recognition networks

do not perform single-stage training but go through pre-training on the extended Imagenet dataset before fine-tuning the target dataset. The Imagenet pre-training allows the network to start the "real" training from a set of good weights and a non-random accuracy. Similar pre-training + fine-tuning protocols are key elements of most successful transformer models, e.g., in NLP [12, 13].

Based on these assumptions, this work introduces a new two-step training procedure for sEMG-based gesture recognition. Compared to the standard approach, I first perform an inter-subject pre-training, in which data relative to all subjects available in the training dataset is employed. Then, I proceed with subject-specific fine-tuning, common to all state-of-the-art approaches. Despite the task being strictly subject-dependent, one can intuitively imagine that the sEMG signal features useful for gesture classification should be similar for all patients. Indeed, using this protocol, I observe that feeding more data is beneficial for accuracy. With this training, I demonstrate higher accuracy on state-of-the-art networks and the new transformer. To clarify better this proposed protocol, I report below the training procedure that I use for subject 1 of the 10-subject Ninapro DB6 training dataset. First, I train the network for 100 epochs with data from patients 2-10, excluding subject 1, on which I want to test the final model. In this step, the model adjusts the weights to extract general features associated with different hand gestures. Then, I perform 20 epochs of fine-tuning using only the training data of subject 1. During this fine-tuning, the recording sessions of the patients are separated between train and test set, following the classical sequential training protocol used by other state-of-the-art approaches for this task, which mimics a real scenario, using sessions 1-5 for training and 6-10 for testing.

For the parameters during the pre-training step, I use Adam optimizer with a linear warmup of the learning rate from 1e-7 to 5e-4; for the fine-tuning stage, a fixed learning rate of 1e-4 is used, with a reduction of 10× after 10 epochs.

5.2.3 Experimental Setup & Dataset

To validate this new architecture, I employ the public sEMG-based hand gesture recognition dataset called Non-Invasive Adaptive hand Prosthetics Database 6 (NinaPro DB6) [115], which has been explicitly realized to investigate the degradation of sEMG-based hand gesture recognition accuracy over time. The dataset includes 10 non-amputee subjects (3 females, 7 males, average age 27 ± 6 years) who have been asked to undergo 10 gathering sessions. The 10 sessions are distributed over 5 days, one in the morning, one in the afternoon, each including 12 repetitions of the gestures for each patient. The gestures considered include the rest position and seven grasps, covering hand movements

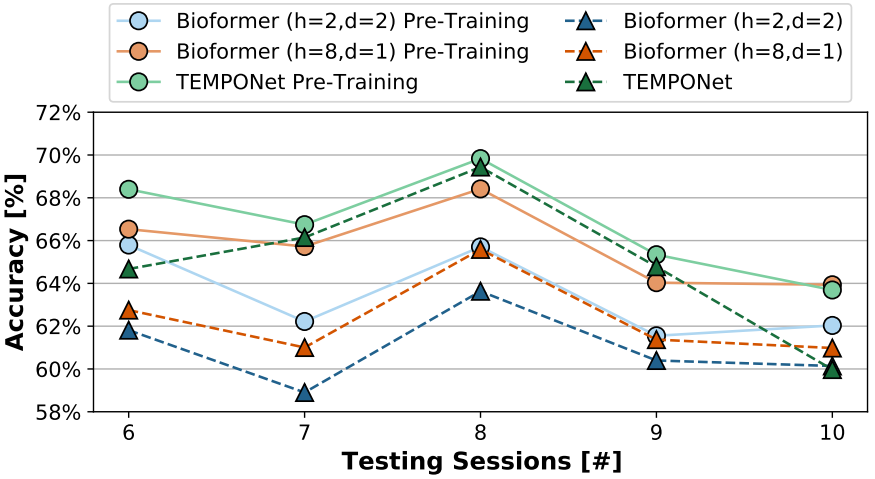

Figure 5.10: Performance variation on the different testing sessions.

typically done during daily activities. Each grasp repetition lasts approximately 6s, followed by 2s of rest. The array of sensors is composed of 14 Delsys Trigno sEMG Wireless electrodes placed on the high half of the forearm, simulating the amputation of the lower half of the forearm. Each sensor gathers the data at a sampling rate of 2 kHz. The dataset is divided into windows of 150 ms (i.e., 300 samples) with a slide between them of 15 ms. Network training is performed on steady gestures, which is done by removing contraction transients: this means that the first and last 1.5 s of each motion is discarded.

For training and validating the model using floating-point (fp32) arithmetic, I employed Python3.7 together with Pytorch1.8.1. I then perform a few epochs of quantization aware training (QAT) to shift from fp32 to integer (int8) arithmetic. I follow the steps described in the previous Sec. 4.4. Finally, I deployed the resulting quantized models on the GAP8 MCU, using the optimized kernels described in Sec. 4.4.

5.2.4 Experimental Results

In this section, I first demonstrate the performance of the Bioformer on the Ninapro DB6. Then, I perform an ablation study to demonstrate i) the pre-training impact on the Bioformer (h = 8, d = 1) and ii) the influence of the filter dimension of the initial convolutional layer on the complexity and on the accuracy of the different architectures. Finally, I discuss the deployment results of the different architectures.

5.2.4.1 Ninapro DB6 benchmark

Fig. 5.10 reports the accuracy of two Bioformers and of the state-of-the-art TEMPONet [34]. Each point corresponds to one of the five testing sessions, and the reported accuracy is the average across patients. Higher session numbers correspond to tests farther in time from the training period. The two Bioformers are composed of a different number of heads and layers. Compared to the reference TCN, the Bioformers achieve slightly lower accuracy both with and without pre-training. Bioformers without pre-training achieve a 2.7%-3.9% lower accuracy on average. However, the accuracy difference w.r.t. TEMPONet decreases for sessions that are farther in time from the training and, therefore, more dissimilar. In particular, the h=8, d=1 Bioformer outperforms TEMPONet on testing session 10 (+ 0.48%). This result suggests that Transformers, thanks to the capability of extracting meaningful features, are more prone to well generalize on more dissimilar data, a key factor for a task where the data can shift over time. Also, note that the application of pre-training is beneficial both for the proposed Bioformers and for TEMPONet. However, the accuracy difference between the two types of models decreases, confirming the superior capability of Transformer-based architectures to benefit from large datasets during training. In the different sessions, I observe an average gain of 3.39%, 2.48%, and 1.80% for Bioformer (h=8, d=1), Bioformer (h=2, d=2), and TEMPONet, respectively.

Overall, the best architecture (i.e., the one with 8 heads) achieves an average 65.73% accuracy, which is 0.73% better than the previous state-of-the-art TEMPONet, and 1.07% lower than the new pre-trained TEMPONet.

5.2.4.2 Ablation Study: pre-training & Patch Dimension

In this paragraph, I detail i) the benefit of applying the new training approach and ii) the impact of the filter dimension of the initial 1D convolutional layer in Bioformers.

Fig. 5.11 details the performance change between standard and two-step training for each subject. Note that the most significant advantages are obtained for subjects with lower accuracy before pre-training. On subjects whose starting accuracy is lower than 60%, the average accuracy improvement is 6.33%, while on the other five subjects, it is just 0.45%, leading to an overall average gain of 3.39%. Solely, Subj.6's accuracy gets worse with the new proposed training. This degradation could be caused by the lower learning rate used in the subject-specific fine-tuning that does not allow the network to converge to the global minimum. This result demonstrates that even tiny transformers can be beneficial, given their optimal deployment on the hardware and the possible high accuracy that could be reached thanks to the extensive dataset used for pre-training.

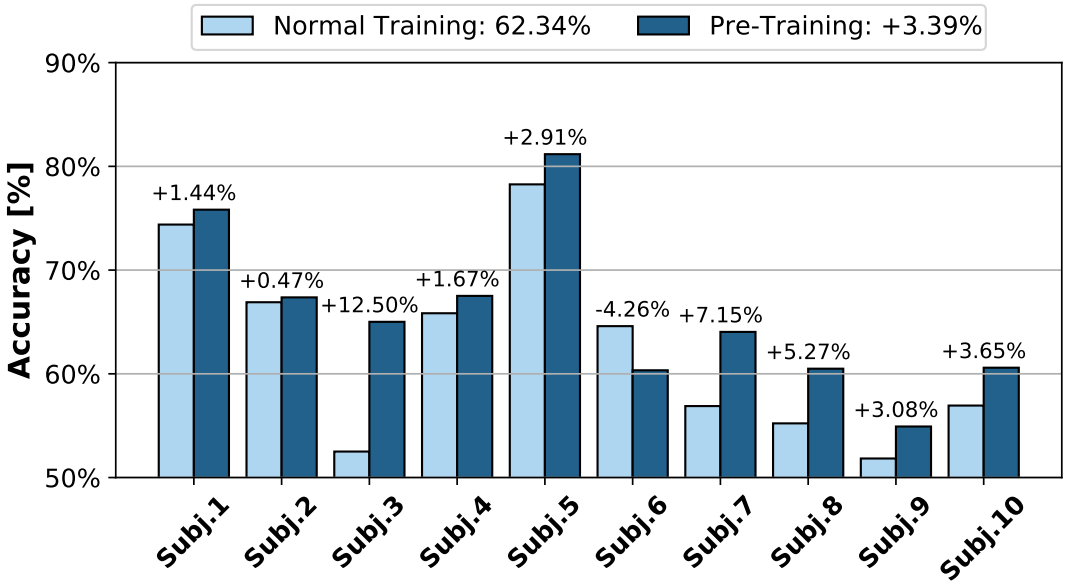

Figure 5.11: Accuracy per subject with intra- and inter-patient training data

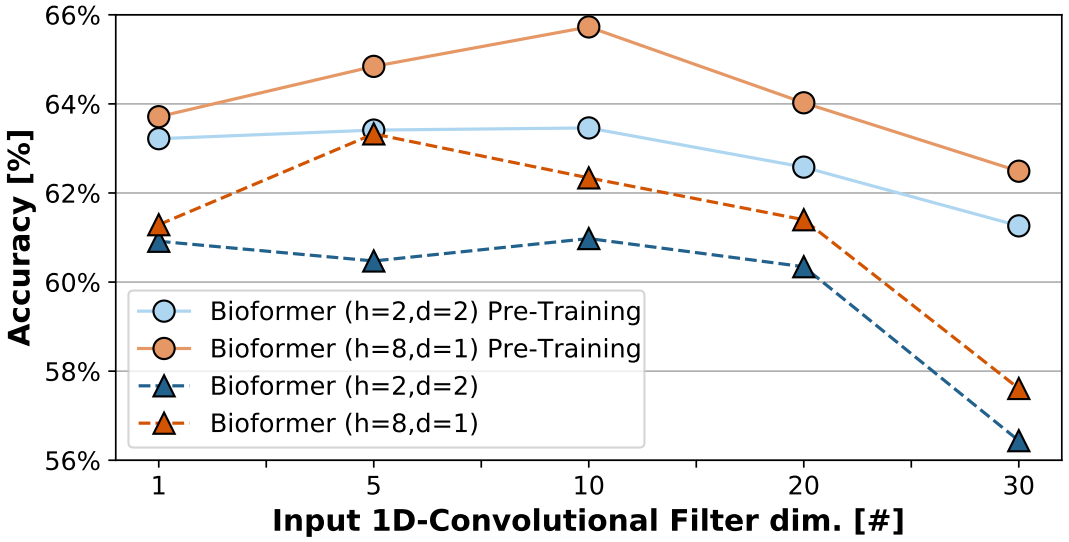

Figure 5.12: Performance using [1,30] filter dimensions for the front-end convolutional layer. Increasing the filter dimension reduces both the number of parameters and the number of operations.

In Fig. 5.12, I show the impact of the filter dimension of the first convolutional layer. Remember that a more comprehensive filter implies a smaller input signal for the attention block. Each solid line represents a Bioformer on which I applied the two-step training (pre-training and fine-tuning), whereas the dashed lines correspond to networks trained with the standard procedure. For most models, a filter dimension equal to 10 results in the best accuracy, despite its lower complexity compared to 1 and 5 (the resulting input sequence length is 30 instead of 60 and 300 for filter sizes 5 and 1[1],

[1]When a filter size of 1 is applied, the 1D-convolutional layer becomes a fully-connected embedding layer.

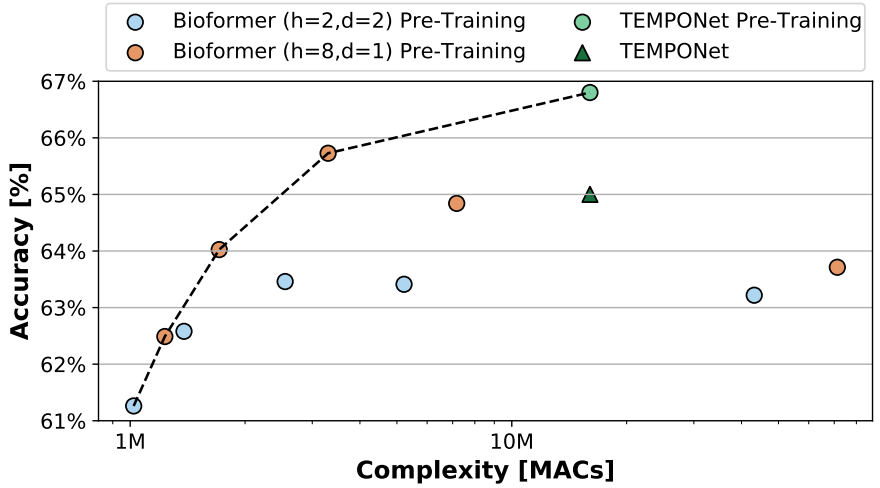

Figure 5.13: Accuracy vs parameters.

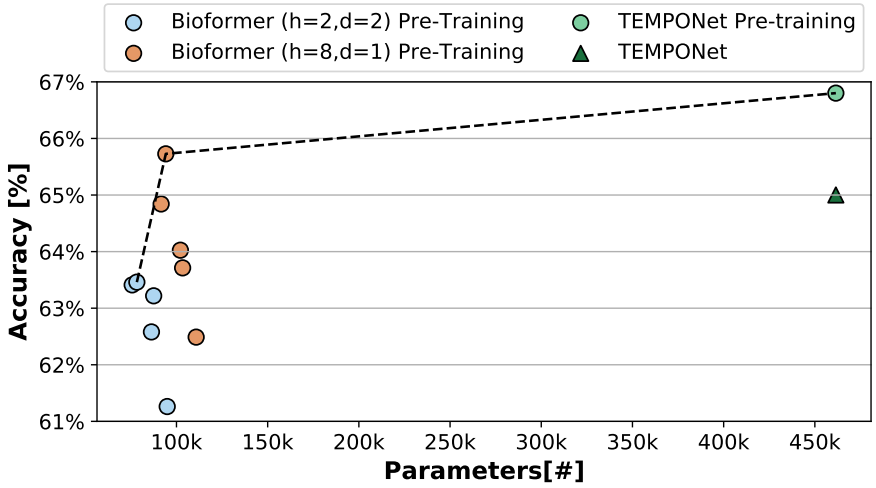

Figure 5.14: Accuracy vs MAC operations.

respectively). Furthermore, despite the resulting lower accuracy, increasing the filter dimension beyond 10 can be beneficial from the deployment point of view, given the reduction in the algorithm's complexity, whose number of operations depends almost linearly on the sequence length. For instance, changing dimension from 10 to 20 on the Bioformer with 8 heads and a depth of 1 only causes a drop of 1.70% of accuracy while reducing the total number of operations by a factor 1.93× and the energy by 2×.

5.2.4.3 Deployment on GAP8

Fig. 5.14 and Fig. 5.13 show the Bioformer architectures and TEMPONet in the N. of Operations versus accuracy and N. of parameters vs. accuracy planes. While the pre-trained TEMPONet reaches the highest accuracy, all other Pareto points are populated by Bioformers. The different points plotted for the same Bioformer refer

Table 5.4: Performance of the quantized Pareto architectures on the GAP8 MCU. Bio1 corresponds to Bioformer (h=8, d=1), Bio2 to Bioformer (h=2, d=2). Abbreviations: Lat.: latency, E.: energy, Q.Acc.: quantized accuracy.

Network	Memory	MMAC	Lat.[ms]	E.[mJ]	Q. Acc.
MCU: GAP8, 100 MHz @ 1V, 51 mW					
Bio1, wind=30	110.8 kB	1.2	1.03	0.052	61.09%
Bio1, wind=20	102.1 kB	1.7	1.37	0.070	63.14%
Bio1, wind=10	94.2 kB	3.3	2.72	0.139	64.69%
Bio2, wind=30	92.2 kB	1.0	1.55	0.079	60.19%
Bio2, wind=10	78.3 kB	2.5	4.82	0.246	62.43%
TEMPONet [34]	461 kB	16.0	21.82	1.11	61.00%

to different filter sizes of the initial 1D Convolutional layer. In the complexity versus accuracy space, I identified two key architectures of Bioformers. The most accurate model (h=8, d=1, filter = 10) outperforms the state-of-the-art TEMPONet and is only 1.07% less accurate than the pre-trained TEMPONet, but shows an impressive 4.9× operations reduction. Instead, the lightest Bioformer (h=2, d=2, filter = 10) on the Pareto frontier reduces the required number of operations of an additional factor 3.3× (16.17× lower than TEMPONet), at the cost of a further 4.47% accuracy drop.

The results of deploying some of these Pareto architectures on GAP8 are shown in Table 5.4. Note that the accuracy of these models, as reported in Table 5.4, is the one obtained after the quantization-aware fine-tuning.

After quantization, the most accurate model yet achieves 64.69% accuracy, consuming an impressively lower 8.0× energy compared to TEMPONet (always considering the same 51 mW of power consumption reported for GAP8 at 100 MHz). This model can also fit a smaller MCU since it only requires 94.2 kB.

The Bioformer with the lowest latency further reduces the energy compared to TEMPONet by 17.3×, with an accuracy reduction of only 3.60% and a comparable memory footprint (110.8 kB). Considering this last model, a 150 ms window classified every 15 ms costs 52 μJ and has a latency of 1.02 ms, while for the remaining time, the GAP8 SoC only collects data. In this step, I can consider idling the 8-core cluster using its embedded hardware synchronization unit [116] and therefore reduce the power consumption to only the 10 mW of the FC, yielding an average power consumption over time of as low as 12.81 mW.

Chapter 6

Conclusions

This thesis showed a complete flow for a generic application, from optimizing the topology to deploying the network on MCUs. In the last chapter, this flow is applied to two bio-inspired tasks, showing how I improved the state-of-the-art performance thanks to the tools presented.

In Chapter 3, I showed a new NAS algorithm, Pruning in Time (PIT). This NAS is optimized for TCNs and can explore a vast, fine-grained search space of architectures with low GPU memory requirements. PIT is the first DMaksingNAS tool that explicitly targets the 1D convolutional networks, targetting both the receptive field and the dilation of all network layers. My thesis demonstrates that PIT can find improved versions of state-of-the-art TCNs, with a memory compression of up to 8.03× (90.8×) and a latency and energy reduction of up to 5.45× (19.6×) without (with a reasonable) accuracy drop on four different benchmarks regarding classification or regression of time-series. After, I introduced a second algorithm, which in turn targets the optimization of the data format of each tensor inside the network. Noteworthy, these two tools can be combined or applied individually. Compared to the existing state-of-the-art algorithm, this tool is particularly innovative in the granularity of precision assigned. The number of bits is not assigned at the granularity of the layer. Instead, every single filter from a layer can assume a different number of bits. To improve both these NASes and generic Dmasking NASes that targets edge deployment, I finally depict a new formulation of the loss that lets them find optimal trade-offs between accuracy and inference complexity under fixed memory constraints.

In Chapter 4, I described DORY, a new tool to deploy the optimized DNN architectures on low-power MCUs. DORY, Deployment Oriented to memoRY, unburdens the programmer from the manual optimizations of neural networks on end nodes. DORY obtains near-optimal tiling in DNNs layers on architectures with three and two memory

levels by combining constraint programming with a set of target-aware heuristics that exploit the target architecture's potential performance, even under stringent memory constraints. I showed its applicability by deploying different neural networks on GAP8, an MCU characterized by a three-level memory and a general-purpose accelerator. I demonstrate 12.6× higher energy efficiency and 7.1× higher performance compared to the industry-standard STM32H743 and up to 26.6% end-to-end inference improvement compared to the proprietary tool from GWT. These results show that part of the Deep Learning Memory Wall, i.e., the limited amount of on-chip memory, can be overcome using optimal multi-level tiling to drive a software-based caching scheme. For instance, in the Results, I showed that GAP8 could execute real-world networks designed for smartphone inference at real-time frame rates with less than 1 MB of on-chip memory. On top of it, in the other two sections of the chapter, I showed two optimized kernels that can be plugged inside DORY for deploying TCNs and Transformers. The first is a library for TCNs to optimize their performance on smart edge nodes. I have shown that by using multiple 1D kernels simultaneously and picking the most efficient per layer of a neural network, I can speed up the execution compared to the state-of-the-art by 3× to 103×. The second library has been made to support for the first time the porting of Transformers from the cloud down to low-power edge devices. I designed a set of optimized yet general kernels, which exploit ARM and RISC-V ISA to improve attention layers' performance. Furthermore, I also show the application of Transformers in a TinyML application, improving the State-of-the-Art (SoA) performance on the TinyRadar dataset by 3.5%, while improving latency and energy by 9.6×, and 6.3× over the performance and energy of the previous SoA network, which is fully dominated.

Finally, I showed two applications that exploit these tools in Chapter 5. In the first work described, I applied neural architecture search to improve the topology and the data format of a network for HR tracking using the PPG signal. I introduced Q-PPG, a new set of quantized deep learning models derived from applying the NAS algorithms exposed in cascade. Q-PPG spans 3 orders of magnitude in memory occupation, with MAEs ranging from a state-of-the-art 4.36 BPM to ≈ 20 BPM, on the PPGDalia dataset. All models are derived from a single seed network through the application of the algorithms in cascade to shrink the model and improve its performance progressively. I also deployed some of the models on the STM32WB55 device, demonstrating that they achieve real-time HR tracking with state-of-the-art accuracy while contributing to 14.1% of the system's total energy consumption when considering sensing and communication. The second application is gesture recognition through sEMG signals. For this task, I demonstrate that tiny Transformers can achieve state-of-the-art performance while strongly reducing the complexity and the memory footprint required for deployment on edge nodes. On Ninapro DB6, the most accurate Bioformer obtains 65.73% accuracy,

better than the previous state-of-the-art accuracy (65.00% of TEMPONet [34]). Deployed on GAP8 with the tools and library introduced in Chapter 4, it consumes just 0.139 mJ with a latency of 2.72 ms.

Bibliography

[1] Tommaso Polonelli, Lukas Schulthess, Philipp Mayer, Michele Magno, and Luca Benini. H-watch: An open, connected platform for ai-enhanced covid19 infection symptoms monitoring and contact tracing. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021. doi: 10.1109/ISCAS51556.2021.9401362.

[2] Alessandro Capotondi, Manuele Rusci, Marco Fariselli, and Luca Benini. CMix-NN: Mixed Low-Precision CNN Library for Memory-Constrained Edge Devices. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2020.

[3] W. He, P. Motlicek, and J. Odobez. Deep neural networks for multiple speaker detection and localization. In *IEEE ICRA*, pages 74–79, 2018. doi: 10.1109/ICRA.2018.8461267.

[4] Marcello Zanghieri, Simone Benatti, Alessio Burrello, Victor Kartsch, Francesco Conti, and Luca Benini. Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor. *IEEE Trans. Biomed. Circuits Syst.*, 2019.

[5] Nhan Duy Truong, Anh Duy Nguyen, Levin Kuhlmann, Mohammad Reza Bonyadi, Jiawei Yang, Samuel Ippolito, and Omid Kavehei. Convolutional neural networks for seizure prediction using intracranial and scalp electroencephalogram. *Neural Networks*, 105:104–111, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2018.04.018.

[6] Alessio Burrello, Francesco Bianco Morghet, Moritz Scherer, Simone Benatti, Luca Benini, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. Bioformers: embedding transformers for ultra-low power semg-based gesture recognition. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1443–1448. IEEE, 2022.

[7] Alessio Burrello, Daniele Jahier Pagliari, Matteo Risso, Simone Benatti, Enrico Macii, Luca Benini, and Massimo Poncino. Q-ppg: Energy-efficient ppg-based

heart rate monitoring on wearable devices. *IEEE Trans. Biomed. Circuits Syst.*, 2021.

[8] Mohsen Azimi, Armin Dadras Eslamlou, and Gokhan Pekcan. Data-driven structural health monitoring and damage detection through deep learning: State-of-the-art review. *Sensors*, 20(10):2778, 2020.

[9] Tania Cerquitelli, Daniele Jahier Pagliari, Andrea Calimera, Lorenzo Bottaccioli, Edoardo Patti, Andrea Acquaviva, and Massimo Poncino. Manufacturing as a data-driven practice: Methodologies, technologies, and tools. *Proc. IEEE*, 109(4): 399–422, 2021. doi: 10.1109/JPROC.2021.3056006.

[10] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv:1506.00019*, 2015.

[11] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[13] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[14] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[15] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proc. IEEE/CVF CVPR*, pages 12965–12974, 2020.

[16] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proc. of the IEEE CVPR*, pages 1586–1595. arXiv, 2018. doi: 10.48550/ARXIV.1711.06798. URL https://arxiv.org/abs/1711.06798.

[17] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proc. IEEE CVPR*, pages 2820–2828, 2019.

[18] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[19] Francesco Conti, Robert Schilling, Pasquale Davide Schiavone, Antonio Pullini, Davide Rossi, Frank Kağan Gürkaynak, Michael Muehlberghuber, Michael Gautschi, Igor Loi, Germain Haugou, et al. An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2481–2494, 2017.

[20] Y. Chen, J. Emer, and V. Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 367–379, 2016.

[21] F. Conti and L. Benini. A Ultra-Low-Energy Convolution Engine for Fast Brain-Inspired Vision in Multicore Clusters. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 683–688, 2015.

[22] Vivienne Sze, Yu-Hsin Chen, Joel Emer, Amr Suleiman, and Zhengdong Zhang. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–8. IEEE, 2017.

[23] Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Davide Rossi, and Luca Benini. XpulpNN: Accelerating Quantized Neural Networkson on RISC-V Processors Through ISA Extensions. In *To appear at Design, Automation and Test in Europe Conference (DATE) 2020*. IEEE, 2020.

[24] G. Desoli, N. Chawla, T. Boesch, S. Singh, E. Guidetti, F. De Ambroggi, T. Majo, P. Zambotti, M. Ayodhyawasi, H. Singh, and N. Aggarwal. A 2.9TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, 2017.

[25] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. PULP-NN: Accelerating Quantized Neural Networks on Parallel Ultra-Low-Power RISC-V Processors. *Philosophical Transactions of the Royal Society A*, 378(2164): 20190155, 2020.

[26] Rasha M Al-Eidan, Hend Al-Khalifa, and Abdul Malik Al-Salman. A review of wrist-worn wearable: Sensors, models, and challenges. *Journal of Sensors*, 2018, 2018.

[27] Eric Flamand, Davide Rossi, Francesco Conti, Igor Loi, Antonio Pullini, Florent Rotenberg, and Luca Benini. GAP-8: A RISC-V SoC for AI at the Edge of the IoT. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–4. IEEE, 2018.

[28] ST Microelectronics. STM32H7, 2022. URL https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html.

[29] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597*, 2021.

[30] Gianmarco Ottavi, Angelo Garofalo, Giuseppe Tagliavini, Francesco Conti, Luca Benini, and Davide Rossi. A mixed-precision risc-v processor for extreme-edge dnn inference. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 512–517, 2020. doi: 10.1109/ISVLSI49217.2020.000-5.

[31] Alessio Burrello, Moritz Scherer, Marcello Zanghieri, Francesco Conti, and Luca 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)*, pages 1–6. IEEE, 2021.

[32] Alessio Burrello, Alberto Dequino, Daniele Jahier Pagliari, Francesco Conti, Marcello Zanghieri, Enrico Macii, Luca Benini, and Massimo Poncino. Tcn mapping optimization for ultra-low power time-series edge inference. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2021.

[33] Liangzhen Lai, Naveen Suda, and Vikas Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[34] M Zanghieri et al. Robust real-time embedded emg recognition framework using temporal convolutional networks on a multicore iot processor. *IEEE transactions on biomedical circuits and systems*, 14(2):244–256, 2019.

[35] Matteo Risso, Alessio Burrello, Daniele Jahier Pagliari, Simone Benatti, Enrico Macii, Luca Benini, and Massimo Poncino. Robust and energy-efficient ppg-based heart-rate monitoring. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[36] Attila Reiss, Ina Indlekofer, Philip Schmidt, and Kristof Van Laerhoven. Deep ppg: large-scale heart rate estimation with convolutional neural networks. *Sensors*, 19 (14):3079, 2019.

[37] Marcello Zanghieri, Simone Benatti, Francesco Conti, Alessio Burrello, and Luca Benini. Temporal variability analysis in semg hand grasp recognition using temporal convolutional networks. In *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 228–232. IEEE, 2020.

[38] Seungwoo Choi, Seokjun Seo, Beomjun Shin, Hyeongmin Byun, Martin Kersner, Beomsu Kim, Dongyoung Kim, and Sungjoo Ha. Temporal convolution for real-time keyword spotting on mobile devices. *arXiv preprint arXiv:1904.03814*, 2019.

[39] P Tsinganos et al. Improved gesture recognition based on semg signals and tcn. In *ICASSP 2019*, pages 1169–1173. IEEE, 2019.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[41] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[42] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *Proc. IEEE/CVF CVPR*, pages 8697–8710, 2018.

[43] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *ArXiv*, abs/1611.02167, 2017.

[44] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *Proc. ICML*, pages 2902–2911. PMLR, 2017.

[45] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2019.

[46] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path mobile automl: Efficient convnet design and nas hyperparameter optimization. *IEEE J. Sel. Topics Signal Process.*, 14(4):609–622, 2020.

[47] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2018.

[48] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proc. ECCV*, pages 116–131, 2018.

[49] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ArXiv*, abs/1905.11946, 2019.

[50] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 1mb model size. *ArXiv*, abs/1602.07360, 2016.

[51] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[52] STMicroelectronics. Stm32wb55, 2022. URL https://www.st.com/en/microcontrollers-microprocessors/stm32wb55rg.html.

[53] Maxim Integrated. Max30101, 2022. URL https://datasheets.maximintegrated.com/en/ds/MAX30101.pdf.

[54] STMicroelectronics. Lsm6dsm, 2022. URL https://www.st.com/resource/en/datasheet/lsm6dsm.pdf.

[55] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini. Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing. *IEEE Journal of Solid-State Circuits*, 54(7):1970–1981, 2019.

[56] Michael Gautschi et al. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[57] Abbas Rahimi, Igor Loi, Mohammad Reza Kakoee, and Luca Benini. A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters. In *2011 Design, Automation & Test in Europe*, pages 1–6. IEEE, 2011.

[58] Davide Rossi, Igor Loi, Germain Haugou, and Luca Benini. Ultra-low-latency lightweight DMA for tightly coupled multi-core clusters. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, 2014.

[59] Cypress. Cypress DRAM, 2019. URL https://www.cypress.com/products/hyperram-memory.

[60] Antonio Pullini, Davide Rossi, Germain Haugou, and Luca 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)*, pages 1–8. IEEE, 2017.

[61] Toshiyo Tamura, Yuka Maeda, Masaki Sekine, and Masaki Yoshida. Wearable photoplethysmographic sensors—past and present. *Electronics*, 3(2):282–302, 2014.

[62] Zhilin Zhang, Zhouyue Pi, and Benyuan Liu. Troika: A general framework for heart rate monitoring using wrist-type photoplethysmographic signals during intensive physical exercise. *IEEE Transactions on biomedical engineering*, 62(2): 522–531, 2014.

[63] Nina Sviridova and Kenshi Sakai. Human photoplethysmogram: new insight into chaotic characteristics. *Chaos, Solitons & Fractals*, 77:53–63, 2015.

[64] Yangsong Zhang, Benyuan Liu, and Zhilin Zhang. Combining ensemble empirical mode decomposition with spectrum subtraction technique for heart rate monitoring using wrist-type photoplethysmography. *Biomedical Signal Processing and Control*, 21:119–125, 2015.

[65] D. Biswas, N. Simões-Capela, C. Van Hoof, and N. Van Helleputte. Heart rate estimation from wrist-worn photoplethysmography: A review. *IEEE Sensors Journal*, 19(16):6560–6570, 2019. doi: 10.1109/JSEN.2019.2914166.

[66] C J De Luca et al. The use of surface electromyography in biomechanics. *Journal of applied biomechanics*, 1997.

[67] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[68] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv:1806.09055*, 2018.

[69] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[70] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[71] ST Microelectornics. X-cube-ai, 2017. URL `https://www.st.com/en/embedded-software/x-cube-ai.html`.

[72] GreenWaves Technologies. Gap8 nntool, 2019. URL `https://greenwaves-technologies.com/manuals/`.

[73] Yanping Chen, Yuan Hao, Thanawin Rakthanmanon, Jesin Zakaria, Bing Hu, and Eamonn Keogh. A general framework for never-ending learning from time series streams. *Data Min. Knowl. Discov.*, 29(6):1622–1664, 2015.

[74] Manfredo Atzori, Arjan Gijsberts, Simone Heynen, Anne-Gabrielle Mittaz Hager, Olivier Deriaz, Patrick Van Der Smagt, Claudio Castellini, Barbara Caputo, and Henning Müller. Building the ninapro database: A resource for the biorobotics community. In *Proc. 4th IEEE RAS & EMBS BioRob*, pages 1258–1265. IEEE, 2012.

[75] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proc. 2016 NAACL*, pages 1480–1489, 2016.

[76] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[77] Colby R Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, et al. Benchmarking tinyml systems: Challenges and direction. *arXiv preprint arXiv:2003.04821*, 2020.

[78] ARM. CMSIS-NN, 2022. URL `https://arm-software.github.io/CMSIS_5/NN/html/index.html`.

[79] Thorir Mar Ingolfsson, Xiaying Wang Michael Hersche, Alessio Burrello, Lukas Cavigelli, and Luca Benini. Ecg-tcn: Wearable cardiac arrhythmia detection with a temporal convolutional network. In *Proc. 3rd IEEE AICAS*, pages 1–4. IEEE, 2021.

[80] Zhaowei Cai and Nuno Vasconcelos. Rethinking differentiable search for mixed-precision neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2020.

[81] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[82] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. Hello edge: Keyword spotting on microcontrollers. *arXiv:1711.07128*, 2017.

[83] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[84] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017. URL https://arxiv.org/abs/1712.05877.

[85] Francesco Daghero, Chen Xie, Daniele Jahier Pagliari, Alessio Burrello, Marco Castellano, Luca Gandolfi, Andrea Calimera, Enrico Macii, and Massimo Poncino. Ultra-compact binary neural networks for human activity recognition on risc-v processors. In *Proc. 18th ACM CF*, 2021. ISBN 9781450384049.

[86] Igor Fedorov, Ramon Matas, Hokchhay Tann, Chuteng Zhou, Matthew Mattina, and Paul Whatmough. Udc: Unified dnas for compressible tinyml models. *arXiv:2201.05842*, 2022.

[87] Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. Data movement is all you need: A case study of transformer networks. *arXiv preprint arXiv:2007.00072*, 2020.

[88] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. Dmazerunner: Executing perfectly nested loops on dataflow accelerators. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–27, 2019.

[89] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE Micro*, 40(3):20–29, 2020.

[90] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 369–383, 2020.

[91] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 304–315. IEEE, 2019.

[92] Lukas Geiger and Plumerai Team. Larq: An Open-Source Library for Training Binarized Neural Networks. *Journal of Open Source Software*, 5(45):1746, January 2020.

[93] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Shlomi Regev, et al. Tensorflow lite micro: Embedded machine learning on tinyml systems. *arXiv preprint arXiv:2010.08678*, 2020.

[94] Manuele Rusci, Alessandro Capotondi, Francesco Conti, and Luca Benini. Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus? In *2018 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–2. IEEE, 2018.

[95] Martín Abadi, Ashish Agarwal, and Paul Barham et al. Tensorflow lite for microcontrollers, 2015. URL `https://www.tensorflow.org/lite/microcontrollers`. Software available from tensorflow.org.

[96] Angelo Garofalo, Manuele Rusci, Francesco Conti, Davide Rossi, and Luca Benini. Pulp-nn: Accelerating quantized neural networks on parallel ultra-low-power risc-v processors. *arXiv preprint arXiv:1908.11263*, 2019.

[97] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64mW DNN-based Visual Navigation Engine for Autonomous Nano-Drones. *IEEE Internet of Things Journal*, 2019.

[98] Francesco Conti. Technical report: Nemo dnn quantization for deployment model, 2020.

[99] Marat Dukhan. The indirect convolution algorithm, 2019.

[100] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus, 2020.

[101] GreenWaves Technologies. Gap8 auto-tiler, 2019. URL `https://greenwaves-technologies.com/manuals/BUILD/AUTOTILER/html/index.html`.

[102] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[103] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. I-bert: Integer-only bert quantization. *arXiv preprint arXiv:2101.01321*, 2021.

[104] 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*, pages 1–1, 2021. doi: 10.1109/JIOT.2021.3067382.

[105] Alessio Burrello, Angelo Garofalo, Nazareno Bruschi, Giuseppe Tagliavini, Davide Rossi, and Francesco Conti. Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus. *IEEE Trans. Comput.*, 2021.

[106] Ralph Neuneier and Hans Georg Zimmermann. How to train neural networks. In *Neural networks: tricks of the trade*, pages 373–423. Springer, 1998.

[107] Matteo Risso, Alessio Burrello, Daniele Jahier Pagliari, Francesco Conti, Lorenzo Lamberti, Enrico Macii, Luca Benini, and Massimo Poncino. Pruning in time (pit): A light-weight network architecture optimizer for temporal convolutional networks. In *Proc. 58th DAC*, pages 1–6, 2021.

[108] Nicholas Huang and Nandakumar Selvaraj. Robust ppg-based ambulatory heart rate tracking algorithm. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 5929–5934. IEEE, 2020.

[109] Tim Schäck, Michael Muma, and Abdelhak M Zoubir. Computationally efficient heart rate estimation during physical exercise using photoplethysmographic signals. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2478–2481. IEEE, 2017.

[110] Seyed Salehizadeh, Duy Dao, Jeffrey Bolkhovsky, Chae Cho, Yitzhak Mendelson, and Ki H Chon. A novel time-varying spectral filtering algorithm for reconstruction of motion artifact corrupted heart rate signals during intense physical activities using a wearable photoplethysmogram sensor. *Sensors*, 16(1):10, 2016.

[111] Menglian Zhou and Nandakumar Selvaraj. Heart rate monitoring using sparse spectral curve tracing. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 5347–5352. IEEE, 2020.

[112] Seok Bin Song, Jung Woo Nam, and Jin Heon Kim. Nas-ppg: Ppg based heart rate estimation using neural architecture search. *IEEE Sensors Journal*, 2021.

[113] NXP. Kv4xp100m168, 2022. URL https://www.nxp.com/docs/en/data-sheet/KV4XP100M168.pdf.

[114] STMicroelectronics. Stm32l031f4, 2022. URL https://www.st.com/en/microcontrollers-microprocessors/stm32l031f4.html.

[115] Francesca Palermo, Matteo Cognolato, Arjan Gijsberts, Henning Muller, Barbara Caputo, and Manfredo Atzori. Repeatability of grasp recognition for robotic hand prosthesis control based on sEMG data. In *2017 International Conference on*

Rehabilitation Robotics (ICORR), pages 1154–1159. IEEE, jul 2017. ISBN 978-1-5386-2296-4. doi: 10.1109/ICORR.2017.8009405. URL https://ieeexplore.ieee.org/document/8009405/.

[116] F. Glaser, G. Haugou, D. Rossi, Q. Huang, and L. Benini. Hardware-accelerated energy-efficient synchronization and communication for ultra-low-power tightly coupled clusters. In *2019 DATE Conference*, pages 552–557, March 2019. doi: 10.23919/DATE.2019.8715266.

[117] Xiangmao Chang, Gangkai Li, Guoliang Xing, Kun Zhu, and Linlin Tu. Deepheart: A deep learning approach for accurate heart rate estimation from ppg signals. *ACM Trans. Sen. Netw.*, 17(2), January 2021. ISSN 1550-4859. doi: 10.1145/3441626. URL https://doi.org/10.1145/3441626.

[118] Heewon Chung, Hoon Ko, Hooseok Lee, and Jinseok Lee. Deep learning for heart rate estimation from reflectance photoplethysmography with acceleration power spectrum and acceleration intensity. *IEEE Access*, 8:63390–63402, 2020.

[119] A Shyam, Vignesh Ravichandran, SP Preejith, Jayaraj Joseph, and Mohanasankar Sivaprakasam. Ppgnet: Deep network for device independent heart rate estimation from photoplethysmogram. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 1899–1902. IEEE, 2019.

[120] KR Arunkumar and M Bhaskar. Robust de-noising technique for accurate heart rate estimation using wrist-type ppg signals. *IEEE Sensors Journal*, 20(14):7980–7987, 2020.

[121] Alistair Doswald, Francesco Carrino, and Fabien. Ringeval. Advanced processing of semg signals for user independent gesture recognition. In *XIII Mediterranean Conference on Medical and Biological Engineering and Computing 2013*, 2014.

[122] Yu Du, Wenguang Jin, Wentao Wei, Yu Hu, and Weidong Geng. Surface EMG-based inter-session gesture recognition enhanced by deep domain adaptation. *Sensors (Switzerland)*, 17(3):6–9, 2017. ISSN 14248220. doi: 10.3390/s17030458.

[123] Koby Crammer, Michael Kearns, and Jennifer Wortman. Learning from multiple sources. *Journal of Machine Learning Research*, 9:1757–1774, 2008. ISSN 1532-4435. doi: 10.1145/1390681.1442790. URL http://portal.acm.org/citation.cfm?id=1442790.

[124] Louis G Tassinary, John T Cacioppo, and Eric J Vanman. The Skeletomotor System : Surface. In . ., 1985.

[125] C J De Luca. The Use of Surface Electromyography. *Journal of applied biomechanics*, 13(July 1993):1–38, 1997.

[126] Rangaraj M Rangayyan. *Biomedical Signal Analysis: A Case-Study Approach.* IEEE/Wiley, New York, NY, 2002.

[127] Marco Tomasini, Simone Benatti, Bojan Milosevic, Elisabetta Farella, and Luca Benini. Power Line Interference Removal for High-Quality Continuous Biosignal Monitoring with Low-Power Wearable Devices. *IEEE Sensors Journal*, 16(10): 3887–3895, 2016. ISSN 1530437X. doi: 10.1109/JSEN.2016.2536363.

[128] Bojan Milosevic, Elisabetta Farella, and Simone Benatti. Exploring Arm Posture and Temporal Variability in Myoelectric Hand Gesture Recognition. *Proceedings of the IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*, 2018-August:1032–1037, 2018. ISSN 21551774. doi: 10.1109/BIOROB.2018.8487838.

[129] Vinicius Horn Cene, Mauricio Tosin, Juliano Machado, and Alexandre Balbinot. Open Database for Accurate Upper-Limb Intent Detection Using Electromyography and Reliable Extreme Learning Machines. *Sensors (Basel, Switzerland)*, 19 (8), 2019. ISSN 14248220. doi: 10.3390/s19081864.

[130] Panagiotis Tsinganos, Bruno Cornelis, Jan Cornelis, Bart Jansen, and Athanassios Skodras. Deep learning in emg-based gesture recognition. In *PhyCS*, 2018.

[131] Wentao Wei, Yongkang Wong, Yu Du, Yu Hu, Mohan Kankanhalli, and Weidong Geng. A multi-stream convolutional neural network for semg-based gesture recognition in muscle-computer interface. *Pattern Recognition Letters*, 119, 12 2017. doi: 10.1016/j.patrec.2017.12.005.

[132] Bernard Hudgins, Philip Parker, and Robert N. Scott. A new strategy for multifunction myoelectric control. *IEEE transactions on bio-medical engineering*, 40: 82–94, 02 1993. doi: 10.1109/10.204774.

[133] K. Englehart and B. Hudgins. A robust, real-time control scheme for multifunction myoelectric control. *IEEE Transactions on Biomedical Engineering*, 50(7):848–854, July 2003. ISSN 0018-9294. doi: 10.1109/TBME.2003.813539.

[134] Claudio Castellini, Emanuele Gruppioni, Angelo Davalli, and Giulio Sandini. Fine detection of grasp force and posture by amputees via surface electromyography. *Journal of Physiology-Paris*, 103:255–262, 2009.

[135] Joseph L. Betthauser, John T. Krall, Rahul R. Kaliki, Matthew S. Fifer, and Nitish V. Thakor. Stable Electromyographic Sequence Prediction during Movement

Transitions using Temporal Convolutional Networks. *International IEEE/EMBS Conference on Neural Engineering*, 2019. ISSN 19483554. doi: 10.1109/NER.2019.8717169.

[136] Manfredo Atzori, Arjan Gijsberts, Simone Heynen, Anne-Gabrielle Mittaz Hager, Olivier Deriaz, Patrick Vand der Smagt, Claudio Castellini, Barbara Caputo, and Henning Müller. Building the NINAPRO Database: A Resource for the Biorobotics Community - HES SO Valais publications - Aigaion 2.0. *Proceedings of the IEEE International Conference on Biomedical Robotics and Biomechatronics*, page 51, 2012. URL http://publications.hevs.ch/index.php/publications/show/1172.

[137] Manfredo Atzori, Arjan Gijsberts, Ilja Kuzborskij, Simone Heynen, Anne-Gabrielle Mittz Hagger, Olivier Deriaz, Claudio Castellini, Henning Müller, and Barbara Caputo. A Benchmark Database for Myoelectric Movement Classification. *IEEE Transactions on Neural Systems & Rehabilitation Engineering*, 23: 73–83, 2013. doi: 10.1109/TNSRE.2014.2328495.

[138] Manfredo Atzori, Arjan Gijsberts, Claudio Castellini, Barbara Caputo, Anne-Gabrielle Mittaz Hager, Simone Elsig, Giorgio Giatsidis, Franco Bassetto, and Henning Müller. Electromyography data for non-invasive naturally-controlled robotic hand prostheses. *Scientific Data*, 1:140053, dec 2014. ISSN 2052-4463. doi: 10.1038/sdata.2014.53. URL http://www.nature.com/articles/sdata201453.

[139] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[140] Ki-Hee Park and Seong-Whan Lee. Movement intention decoding based on deep learning for multiuser myoelectric interfaces. *2016 4th International Winter Conference on Brain-Computer Interface (BCI)*, pages 1–2, 2016.

[141] Manfredo Atzori, Arjan Gijsberts, Ilja Kuzborskij, Simone Elsig, Anne-Gabrielle Mittaz Hager, Olivier Deriaz, Claudio Castellini, Henning Muller, and Barbara Caputo. Characterization of a benchmark database for myoelectric movement classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 23:73–83, 2015.

[142] Manfredo Atzori, Matteo Cognolato, and Henning Müller. Deep learning with convolutional neural networks applied to electromyography data: A resource for the classification of movements for prosthetic hands. *Frontiers in Neurorobotics*, 10, 09 2016. doi: 10.3389/fnbot.2016.00009.

[143] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[144] Angkoon Phinyomark and Erik Scheme. Emg pattern recognition in the era of big data and deep learning. *Big Data and Cognitive Computing*, 2(3):21, 2018.

[145] Colin Lea, Michael D. Flynn, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks for action segmentation and detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1003–1012, 2016.

[146] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[147] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[148] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[149] Yanghao Li, Naiyan Wang, Jianping Shi, Jiaying Liu, and Xiaodi Hou. Revisiting batch normalization for practical domain adaptation. *ArXiv*, abs/1603.04779, 2016.

[150] Yu Du, Wenguang Jin, Wentao Wei, Yu Hu, and Weidong Geng. Surface emg-based inter-session gesture recognition enhanced by deep domain adaptation. In *Sensors*, 2017.

[151] Ulysse Côté Allard, Cheikh Latyr Fall, Alexandre Drouin, Alexandre Campeau-Lecours, Clément Gosselin, Kyrre Glette, François Laviolette, and Benoit Gosselin. Deep learning for electromyographic hand gesture signal classification by leveraging transfer learning. *CoRR*, abs/1801.07756, 2018. URL `http://arxiv.org/abs/1801.07756`.

[152] Sebastian Amsuss, Liliana P. Paredes, Nina Rudigkeit, Bernhard Graimann, Michael J. Herrmann, and Dario Farina. Long term stability of surface EMG pattern classification for prosthetic control. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, pages 3622–3625, 2013. ISSN 1557170X. doi: 10.1109/EMBC.2013.6610327.

[153] Jiayuan He, Dingguo Zhang, Ning Jiang, Xinjun Sheng, Dario Farina, and Xiangyang Zhu. User adaptation in long-term, open-loop myoelectric training: Implications for EMG pattern recognition in prosthesis control. *Journal of Neural Engineering*, 12(4), 2015. ISSN 17412552. doi: 10.1088/1741-2560/12/4/046005.

[154] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, June 2018.

[155] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[156] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[157] Abeg Kumar Jaiswal and Haider Banka. Local pattern transformation based feature extraction techniques for classification of epileptic eeg signals. *Biomedical Signal Processing and Control*, 34:81–92, 2017. ISSN 1746-8094. doi: https://doi.org/10.1016/j.bspc.2017.01.005. URL http://www.sciencedirect.com/science/article/pii/S174680941730006X.

[158] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, and L. Benini. A 64mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 2019. ISSN 2327-4662. doi: 10.1109/JIOT.2019.2917066.

[159] Fabio Montagna, Abbas Rahimi, Simone Benatti, Davide Rossi, and Luca Benini. Pulp-hd: Accelerating brain-inspired high-dimensional computing on a parallel ultra-low power platform. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18, pages 111:1–111:6, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5700-5. doi: 10.1145/3195970.3196096. URL http://doi.acm.org/10.1145/3195970.3196096.

[160] Hirsch Martin, Altenmüller Dirk-Matthias, and Schulze-Bonhage Andreas. Latencies from intracranial seizure onset to ictal tachycardia: A comparison to surface eeg patterns and other clinical signs. *Epilepsia*, 56(10):1639–1647, 2015. doi: 10.1111/epi.13117. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/epi.13117.

[161] Christian Rummel, Eugenio Abela, Ralph G. Andrzejak, Martinus Hauf, Claudio Pollo, Markus Müller, Christian Weisstanner, Roland Wiest, and Kaspar Schindler. Resected brain tissue, seizure onset zone and quantitative eeg measures: Towards prediction of post-surgical seizure control. *PLOS ONE*, 10(10): 1–26, 10 2015. doi: 10.1371/journal.pone.0141023. URL https://doi.org/10.1371/journal.pone.0141023.

[162] M. A. Bin Altaf, C. Zhang, and J. Yoo. A 16-channel patient-specific seizure onset and termination detection soc with impedance-adaptive transcranial electrical stimulator. *IEEE Journal of Solid-State Circuits*, 50(11):2728–2740, Nov 2015. ISSN 0018-9200. doi: 10.1109/JSSC.2015.2482498.

[163] Florian Mormann, Ralph G. Andrzejak, Christian E. Elger, and Klaus Lehnertz. Seizure prediction: the long and winding road. *Brain*, 130(2):314–333, 2007. doi: 10.1093/brain/awl241. URL http://dx.doi.org/10.1093/brain/awl241.

[164] Dieter Schmidt and Matti Sillanpää. Evidence-based review on the natural history of the epilepsies. *Current opinion in neurology*, 25 2:159–63, 2012.

[165] José F. Téllez-Zenteno, Raj Dhar, and Samuel Wiebe. Long-term seizure outcomes following epilepsy surgery: a systematic review and meta-analysis. *Brain*, 128(5): 1188–1198, 2005. doi: 10.1093/brain/awh449. URL http://dx.doi.org/10.1093/brain/awh449.

[166] RummelC, AbelaE, AndrzejakRG, Hauf M, PolloC, MüllerM, and et al. Resected brain tissue, seizure onset zone and quantitative eeg measures: Towards prediction of post-surgical seizure control. *PLOSOne*, 2015. doi: https://doi.org/10.1371/journal.pone.0141023.

[167] Schindler Kaspar, Gast Heidemarie, Stieglitz Lennart, Stibal Alexander, Hauf Martinus, Wiest Roland, Mariani Luigi, and Rummel Christian. Forbidden ordinal patterns of periictal intracranial eeg indicate deterministic dynamics in human epileptic seizures. *Epilepsia*, 52(10):1771–1780, 2011. doi: 10.1111/j.1528-1167.2011.03202.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1528-1167.2011.03202.x.

[168] T. S. Kumar, V. Kanhangad, and R. B. Pachori. Classification of seizure and seizure-free eeg signals using multi-level local patterns. In *2014 19th International Conference on Digital Signal Processing*, pages 646–650, Aug 2014. doi: 10.1109/ICDSP.2014.6900745.

[169] Yılmaz Kaya, Murat Uyar, Ramazan Tekin, and Selçuk Yıldırım. 1d-local binary pattern based feature extraction for classification of epileptic eeg signals. *Applied Mathematics and Computation*, 243:209–219, 2014. ISSN 0096-3003. doi: https://doi.org/10.1016/j.amc.2014.05.128. URL http://www.sciencedirect.com/science/article/pii/S0096300314008285.

[170] C. S. Daw, C. E. A. Finney, and E. R. Tracy. A review of symbolic analysis of experimental data. *Review of Scientific Instruments*, 74(2):915–930, 2003. doi: 10.1063/1.1531823. URL https://doi.org/10.1063/1.1531823.

[171] Schindler Kaspar, Gast Heidemarie, Goodfellow Marc, and Rummel Christian. On seeing the trees and the forest: Single-signal and multisignal analysis of perictal intracranial eeg. *Epilepsia*, 53(9):1658–1668, 2012. doi: 10.1111/j.1528-1167.2012.03588.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1528-1167.2012.03588.x.

[172] William Stacey, Michel Le Van Quyen, Florian Mormann, and Andreas Schulze-Bonhage. What is the present-day eeg evidence for a preictal state? *Epilepsy Research*, 97(3):243–251, 2011. ISSN 0920-1211. doi: https://doi.org/10.1016/j.eplepsyres.2011.07.012. URL http://www.sciencedirect.com/science/article/pii/S0920121111002154. Special Issue on Epilepsy Research UK Workshop 2010 on "Preictal Phenomena".

[173] Md Rezwanul Ahsan, Muhammad I Ibrahimy, Othman O Khalifa, et al. Emg signal classification for human computer interaction: a review. *European Journal of Scientific Research*, 33(3):480–501, 2009.

[174] M. Imani, J. Hwang, T. Rosing, A. Rahimi, and J. M. Rabaey. Low-power sparse hyperdimensional encoder for language recognition. *IEEE Design Test*, 34(6):94–101, Dec 2017. ISSN 2168-2356. doi: 10.1109/MDAT.2017.2740839.

[175] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016. URL http://arxiv.org/abs/1606.04080.

[176] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4080–4090. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6996-prototypical-networks-for-few-shot-learning.pdf.

[177] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schönle, S. Fateh, T. Burger, Q. Huang, and L. Benini. A versatile embedded platform for EMG acquisition and gesture recognition. *IEEE Transactions on Biomedical Circuits and Systems*, 9(5):620–630, Oct 2015. ISSN 1932-4545. doi: 10.1109/TBCAS.2015.2476555.

[178] Kai Keng Ang, Zheng Yang Chin, Chuanchu Wang, Cuntai Guan, and Haihong Zhang. Filter bank common spatial pattern algorithm on bci competition iv datasets 2a and 2b. *Frontiers in Neuroscience*, 6:39, 2012. ISSN 1662-453X. doi: 10.3389/fnins.2012.00039. URL https://www.frontiersin.org/article/10.3389/fnins.2012.00039.

[179] S. Sakhavi, C. Guan, and S. Yan. Parallel convolutional-linear neural network for motor imagery classification. In *2015 23rd European Signal Processing Conference (EUSIPCO)*, pages 2736–2740, Aug 2015. doi: 10.1109/EUSIPCO.2015.7362882.

[180] Deng Wang, Duoqian Miao, and Gunnar Blohm. Multi-class motor imagery EEG decoding for brain-computer interfaces. *Frontiers in Neuroscience*, 6:151, 2012. ISSN 1662-453X. doi: 10.3389/fnins.2012.00151. URL https://www.frontiersin.org/article/10.3389/fnins.2012.00151.

[181] S. Saeedi, R. Chavarriaga, R. Leeb, and José del R. Millán. Adaptive assistance for brain-computer interfaces by online prediction of command reliability. *IEEE Computational Intelligence Magazine*, 11(1):32–39, Feb 2016. ISSN 1556-603X. doi: 10.1109/MCI.2015.2501550.

[182] Dapeng Yang, Li Jiang, Qi Huang, Rongqiang Liu, and Hong Liu. Experimental study of an EMG-controlled 5-dof anthropomorphic prosthetic hand for motion restoration. *Journal of Intelligent & Robotic Systems*, 76(3):427–441, Dec 2014. ISSN 1573-0409. doi: 10.1007/s10846-014-0037-6. URL https://doi.org/10.1007/s10846-014-0037-6.

[183] J. Rosen, M. Brand, M. B. Fuchs, and M. Arcan. A myosignal-based powered exoskeleton system. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 31(3):210–222, May 2001. ISSN 1083-4427. doi: 10.1109/3468.925661.

[184] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias, and J. M. Rabaey. An EMG gesture recognition system with flexible high-density sensors and brain-inspired high-dimensional classifier. In *IEEE International Symposium on Circuits and Systems, ISCAS*, In press 2018.

[185] D. Kleyko, A. Rahimi, D. Rachkovskij, E. Osipov, P. Kanerva, and J. M. Rabaey. Binary hyperdimensional computing: Trade-offs in choice of density and mapping characteristics. In *IEEE Transactions on Neural Networks and Learning Systems, TNNLS*, In press 2018.

[186] T. Wu, P.-C. Huang, A. Rahimi, H. Li, M. Shulaker, J. M. Rabaey, H.-S.P. Wong, and S. Mitra. Brain-inspired computing exploiting carbon nanotube FETs and resistive RAM: Hyperdimensional computing case study. In *IEEE International Solid-State Circuits Conference, ISSCC*, In press 2018.

[187] Jan M. Rabaey. A roadmap to lower supply voltages—a system perspective. In *IEEE International Solid-State Circuits Conference, ISSCC*, 2015.

[188] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL http://proceedings.mlr.press/v54/mcmahan17a.html.

[189] White House Report. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. In *Journal of Privacy and Confidentiality*, 2013.

[190] Ozgur Yilmaz. Symbolic computation using cellular automata-based hyperdimensional computing. *Neural Computation*, 27(12):2661–2692, 2015. doi: 10.1162/NECO_a_00787. URL https://doi.org/10.1162/NECO_a_00787. PMID: 26496041.

[191] D. Kleyko, S. Khan, E. Osipov, and S. P. Yong. Modality classification of medical images with distributed representations based on cellular automata reservoir computing. In *2017 IEEE 14th International Symposium on Biomedical Imaging (ISBI 2017)*, pages 1053–1056, April 2017. doi: 10.1109/ISBI.2017.7950697.

[192] Ashok Litwin-Kumar, Kameron Decker Harris, Richard Axel, Haim Sompolinsky, and L. F. Abbott. Optimal degrees of synaptic connectivity. *Neuron*, 93(5):1153–1164, March 2017. ISSN 0896-6273. doi: 10.1016/j.neuron.2017.01.030. URL http://dx.doi.org/10.1016/j.neuron.2017.01.030.

[193] O Bertrand, F Perrin, and J Pernier. A theoretical justification of the average reference in topographic evoked potential studies. *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, 62(6):462–464, 1985.

ISSN 0168-5597. doi: http://dx.doi.org/10.1016/0168-5597(85)90058-9. URL http://www.sciencedirect.com/science/article/pii/0168559785900589.

[194] Pierre Ferrez and José del R. Millán. *Error-related EEG potentials in brain-computer interfaces*. PhD thesis, STI, Lausanne, 2007.

[195] Dennis J. McFarland, Lynn M. McCane, Stephen V. David, and Jonathan R. Wolpaw. Spatial filter selection for EEG-based communication. *Electroencephalography and Clinical Neurophysiology*, 103(3):386–394, 1997. ISSN 0013-4694. doi: http://dx.doi.org/10.1016/S0013-4694(97)00022-2. URL http://www.sciencedirect.com/science/article/pii/S0013469497000222.

[196] Monitoring error-related potentials. http://bnci-horizon-2020.eu/database/data-sets, 2022.

[197] GAP8 SDK. https://greenwaves-technologies.com/setting-up-sdk/, 2022.

[198] BCI Competition IV-2a (Four class motor imagery). http://bnci-horizon-2020.eu/database/data-sets, 2022.

[199] BNCI Horizon 2020. http://bnci-horizon-2020.eu/, 2022.

[200] R. Chavarriaga and J. d. R. Millán. Learning from EEG error-related potentials in noninvasive brain-computer interfaces. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 18(4):381–388, Aug 2010. ISSN 1534-4320. doi: 10.1109/TNSRE.2010.2053387.

[201] Pierre W. Ferrez and José Del R. Millán. You are wrong!—automatic detection of interaction errors from brain waves. In *In Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005.

[202] P. W. Ferrez and J. del R. Millán. Error-related EEG potentials generated during simulated brain-computer interaction. *IEEE Transactions on Biomedical Engineering*, 55(3):923–929, March 2008. ISSN 0018-9294. doi: 10.1109/TBME.2007.908083.

[203] Xiaofei He, Deng Cai, and Partha Niyogi. Laplacian score for feature selection. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*, NIPS'05, pages 507–514, Cambridge, MA, USA, 2005. MIT Press. URL http://dl.acm.org/citation.cfm?id=2976248.2976312.

[204] Abbas Rahimi, Artiom Tchouprina, Pentti Kanerva, José del R. Millán, and Jan M. Rabaey. Hyperdimensional computing for blind and one-shot classification of EEG error-related potentials. *Mobile Networks and Applications*,

pages 1–12, Oct 2017. ISSN 1572-8153. doi: 10.1007/s11036-017-0942-6. URL https://doi.org/10.1007/s11036-017-0942-6.

[205] Yitong Li, michael Murias, samantha Major, geraldine Dawson, Kafui Dzirasa, Lawrence Carin, and David E Carlson. Targeting EEG/LFP synchrony with neural nets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4623–4633. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7048-targeting-eeglfp-synchrony-with-neural-nets.pdf.

[206] D. Rossi, F. Conti, A. Marongiu, A. Pullini, I. Loi, M. Gautschi, G. Tagliavini, A. Capotondi, P. Flatresse, and L. Benini. PULP: A parallel ultra low power platform for next generation IoT applications. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–39, Aug 2015. doi: 10.1109/HOTCHIPS.2015.7477325.

[207] D. Kleyko and E. Osipov. On bidirectional transitions between localist and distributed representations: The case of common substrings search using vector symbolic architecture. *Procedia Computer Science*, 41:104–113, 2014.

[208] D. Kleyko, E. Osipov, and R. W. Gayler. Recognizing permuted words with Vector Symbolic Architectures: A Cambridge test for machines. *Procedia Computer Science*, 88:169–175, 2016.

[209] R Stanley Williams, Erik P DeBenedictis, IBM Arvind Kumar, Mark Stalzer, Mustafa Badaroglu, Geoff W Burr Qualcomm, An Chen, Shamik Das Globalfoundaries, Andrew B MITRE, Matt Marinella, et al. Ostp nanotechnology-inspired grand challenge: Sensible machines (extended version 2.5). Technical report, Tech. Rep., Oct, 2015.

[210] H. Li, T. F. Wu, A. Rahimi, K. S. Li, M. Rusch, C. H. Lin, J. L. Hsu, M. M. Sabry, S. B. Eryilmaz, J. Sohn, W. C. Chiu, M. C. Chen, T. T. Wu, J. M. Shieh, W. K. Yeh, J. M. Rabaey, S. Mitra, and H. S. P. Wong. Hyperdimensional computing with 3D VRRAM in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition. In *2016 IEEE International Electron Devices Meeting (IEDM)*, pages 16.1.1–16.1.4, Dec 2016. doi: 10.1109/IEDM.2016.7838428.

[211] D. Kleyko and E. Osipov. Brain-like classifier of temporal patterns. In *The Proceeding of the 2nd International Conference on Computer and Information Sciences - ICCOINS*, pages 1–6, 2014.

[212] B. B. Nasution and A. I. Khan. A Hierarchical Graph Neuron Scheme for Real-Time Pattern Recognition. *Neural Networks, IEEE Transactions on*, 19(2):212–229, February 2008.

[213] D. Kleyko, E. Osipov, N. Papakonstantinou, V. Vyatkin, and A. Mousavi. Fault detection in the hyperspace: Towards intelligent automation systems. In *IEEE International Conference on Industrial Informatics, INDIN*, pages 1–6, 2015.

[214] S. I. Gallant and T. W. Okaywe. Representing objects, relations, and sequences. *Neural Computation*, 25(8):2038–2078, 2013.

[215] T.A. Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995.

[216] D. Aerts, M. Czachor, and B. De Moor. Geometric analogue of holographic reduced representation. *Journal of Mathematical Psychology*, 53:389–398, 2009.

[217] D. A. Rachkovskij. Representation and Processing of Structures with Binary Sparse Distributed Codes. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):261–276, 2001.

[218] H. S. P. Wong, H. Y. Lee, S. Yu, Y. S. Chen, Y. Wu, P. S. Chen, B. Lee, F. T. Chen, and M. J. Tsai. Metal Oxide RRAM. *Proceedings of the IEEE*, 100(6): 1951–1970, June 2012. ISSN 0018-9219. doi: 10.1109/JPROC.2012.2190369.

[219] G. Recchia, M. Sahlgren, and P. Kanerva M.N. Jones. Encoding Sequential Information in Semantic Space Models. Comparing Holographic Reduced Representation and Random Permutation. *Computational Intelligence and Neuroscience*, pages 1–18, 2015.

[220] H. Y. Chen, S. Yu, B. Gao, P. Huang, J. Kang, and H. S. P. Wong. HfOx based vertical resistive random access memory for cost-effective 3D cross-point architecture without cell selector. In *Electron Devices Meeting (IEDM), 2012 IEEE International*, pages 20.7.1–20.7.4, Dec 2012. doi: 10.1109/IEDM.2012.6479083.

[221] B. Govoreanu, A. Redolfi, L. Zhang, C. Adelmann, M. Popovici, S. Clima, H. Hody, V. Paraschiv, I. P. Radu, A. Franquet, J. C. Liu, J. Swerts, O. Richard, H. Bender, L. Altimime, and M. Jurczak. Vacancy-modulated conductive oxide resistive RAM (VMCO-RRAM): An area-scalable switching current, self-compliant, highly nonlinear and wide on/off-window resistive switching cell. In *2013 IEEE International Electron Devices Meeting*, pages 10.2.1–10.2.4, Dec 2013. doi: 10.1109/IEDM.2013.6724599.

[222] Seunghyun Lee, Joon Sohn, Hong-Yu Chen, and H-S Philip Wong. Metal Oxide Resistive Memory using Graphene Edge Electrode. *Nature Communications*, September 25, 2015.

[223] T.A. Plate. *Holographic Reduced Representations*. CLSI Publications, 2003.

[224] E. Paxon Frady, Denis Kleyko, Pentti Kanerva, and Friedrich T. Sommer. The capacity of active memorythe information capacity of distributed neural activity. In *Redwood Center Preprint*, 2017.

[225] E. Weiss, B. Cheung, and B. A. Olshausen. Representing spatial structure with complex vectors. In *Proceedings of the International Conference on Learning Representations, ICLR*, 2016.

[226] Pentti Kanerva. Fully Distributed Representation. In , editor, *Proceedings of 1997 Real World Computing Symposium, RWC'97*, pages 358–365. , 1997.

[227] Pentti Kanerva. Computing with 10,000-bit words. In *Proc. 52nd Annual Allerton Conference on Communication, Control, and Computing*, 2014.

[228] Tony Plate. Holographic reduced representations: Convolution algebra for compositional distributed representations. In *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 30–35. Morgan Kaufmann, 1991.

[229] Ross W. Gayler. Multiplicative binding, representation operators & analogy. In *Gentner, D., Holyoak, K. J., Kokinov, B. N. (Eds.), Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences*, pages 1–4, New Bulgarian University, Sofia, Bulgaria, 1998. URL `http://cogprints.org/502/`.

[230] Ross W. Gayler. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. In *Proceedings of the Joint International Conference on Cognitive Science. ICCS/ASCS*, pages 133–138, 2003.

[231] Pentti Kanerva, Jan Kristoferson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society*, page 1036. Erlbaum, 2000. URL `http://www.rni.org/kanerva/cogsci2k-poster.txt`.

[232] Chris Eliasmith. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.

[233] D. Kleyko, E. Osipov, A. Senior, A. I. Khan, and Y. A. Sekercioglu. Holographic graph neuron: A bioinspired architecture for pattern processing. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–13, 2016. ISSN 2162-237X. doi: 10.1109/TNNLS.2016.2535338.

[234] Thomas K Landauer and Susan T. Dutnais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2):211–240, 1997.

[235] Fateme Rasti Najafabadi, Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. Hyperdimensional computing for text classification. *Design, Automation Test in Europe Conference Exhibition (DATE), University Booth*, 2016. URL `https://www.date-conference.com/system/files/file/date16/ubooth/37923.pdf`.

[236] Abbas Rahimi, Pentti Kanerva, and Jan M. Rabaey. A robust and energy efficient classifier using brain-inspired hyperdimensional computing. In *Low Power Electronics and Design (ISLPED), 2016 IEEE/ACM International Symposium on*, August 2016.

[237] O. Räsänen and S. Kakouros. Modeling dependencies in multiple parallel data streams with hyperdimensional computing. *IEEE Signal Processing Letters*, 21 (7):899–903, July 2014. ISSN 1070-9908. doi: 10.1109/LSP.2014.2320573.

[238] O. Räsänen and J. Saarinen. Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–12, 2015. ISSN 2162-237X. doi: 10.1109/TNNLS.2015.2462721.

[239] M. Laiho, J. H. Poikonen, P. Kanerva, and E. Lehtonen. High-dimensional computing with sparse vectors. In *Biomedical Circuits and Systems Conference (BioCAS), 2015 IEEE*, pages 1–4, Oct 2015. doi: 10.1109/BioCAS.2015.7348414.

[240] Dominic Widdows and Microsoft Bing. Reasoning with vectors: a continuous model for fast robust inference. In *Logic Journal of the IGPL*, 2014.

[241] Manish Deo, Jeffrey Schulz, and Lance Brown. White paper: Stratix 10 mx devices solve the memory bandwidth challenge. Altera, WP-01264-1.0, May 2016.

[242] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1915–1929, Aug 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.231.

[243] H. Li and et al. Four-layer 3D vertical RRAM integrated with FinFET as a versatile computing unit for brain-inspired cognitive information processing. In *IEEE Symp. VLSI Technology*, 2016.

[244] Pentti Kanerva. What we mean when we say "what's the dollar of mexico?": Prototypes and mapping in concept space. In *AAAI Fall Symposium: Quantum Informatics for Cognitive, Social, and Semantic Processes*, pages 2–6, 2010.

[245] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: Energy-efficient neuromorphic systems using approximate computing. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design*, ISLPED '14, pages 27–32, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2975-0. doi: 10.1145/2627369.2627613. URL http://doi.acm.org/10.1145/2627369.2627613.

[246] Beinuo Zhang, Zhewei Jiang, Qi Wang, Jae sun Seo, and Mingoo Seok. A neuromorphic neural spike clustering processor for deep-brain sensing and stimulation systems. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on*, pages 91–97, July 2015. doi: 10.1109/ISLPED.2015.7273496.

[247] Aditya Joshi, Johan T. Halseth, and Pentti Kanerva. Language geometry using random indexing. In Jose Acacio de Barros, Bob Coecke, and Emmanuel Pothos, editors, *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers*, pages 265–274, Cham, 2017. Springer International Publishing. ISBN 978-3-319-52289-0. doi: 10.1007/978-3-319-52289-0_21. URL http://dx.doi.org/10.1007/978-3-319-52289-0_21.

[248] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. http://www.statmt.org/europarl/, 2005.

[249] Uwe Quasthoff, Matthias Richter, and Christian Biemann. Biemann c., "corpus portal for search in monolingual corpora. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, 2006.

[250] Thorsten Joachims. Text categorization with suport vector machines: Learning with many relevant features. In *Proceedings of the 10th European Conference on Machine Learning*, ECML '98, pages 137–142, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-64417-2. URL http://dl.acm.org/citation.cfm?id=645326.649721.

[251] T.-T. Liu and J.M. Rabaey. A 0.25 v 460 nw asynchronous neural signal processor with inherent leakage suppression. *Solid-State Circuits, IEEE Journal of*, 48(4): 897–906, April 2013. ISSN 0018-9200. doi: 10.1109/JSSC.2013.2239096.

[252] D. Kuzum, R.G.D. Jeyasingh, Shimeng Yu, and H.-S.P. Wong. Low-energy robust neuromorphic computation using synaptic devices. *Electron Devices, IEEE Transactions on*, 59(12):3489–3494, Dec 2012. ISSN 0018-9383. doi: 10.1109/TED.2012.2217146.

[253] S.D. Levy and R.W. Gayler. Lateral inhibition in a fully distributed connectionist architecture. In *Proceedings of the Ninth International Conference on Cognitive Modeling*, 2009.

[254] Pentti Kanerva. *Sparse Distributed Memory*. The MIT Press, Cambridge, MA, USA, 1988. ISBN 0262111322.

[255] Magnus Sahlgren. An introduction to random indexing. In *In Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, 2005.

[256] TK LANDAUER and ST DUMAIS. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211–240, 1997.

[257] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(1):3–55, January 2001. ISSN 1559-1662. doi: 10.1145/584091.584093. URL http://doi.acm.org/10.1145/584091.584093.

[258] P. K. Artemiadis and K. J. Kyriakopoulos. EMG-based teleoperation of a robot arm using low-dimensional representation. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 489–495, Oct 2007. doi: 10.1109/IROS.2007.4399452.

[259] Dapeng Yang, Li Jiang, Qi Huang, Rongqiang Liu, and Hong Liu. Experimental study of an EMG-controlled 5-dof anthropomorphic prosthetic hand for motion restoration. *J. Intell. Robotics Syst.*, 76:427–441, 2014. ISSN 0921-0296. doi: 10.1007/s10846-014-0037-6. URL http://dx.doi.org/10.1007/s10846-014-0037-6.

[260] Ottobock Sensor 13E200. http://www.ottobock.com/, 2022.

[261] L.J. Hargrove, K. Englehart, and B. Hudgins. A comparison of surface and intramuscular myoelectric signal classification. *Biomedical Engineering, IEEE Transactions on*, 54(5):847–853, May 2007.

[262] Mohammadreza Asghari Oskoei and Huosheng Hu. Myoelectric control systems—a survey. *Biomedical signal processing and control*, 2(4):275–294, 2007.

[263] Haoshi Zhang, Yaonan Zhao, Fuan Yao, Lisheng Xu, Peng Shang, and Guanglin Li. An adaptation strategy of using LDA classifier for EMG pattern recognition. In *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 4267–4270, 2013. doi: 10.1109/EMBC.2013.6610488.

[264] Mohammadreza Asghari Oskoei, Student Member, and Huosheng Hu. Support vector machine-based classification scheme for myoelectric control applied to upper limb, 2022.

[265] M.R. Ahsan, M.I. Ibrahimy, and O.O. Khalifa. Electromygraphy (EMG) signal based hand gesture recognition using artificial neural network (ann). In *Mechatronics (ICOM), 2011 4th International Conference On*, pages 1–6, May 2011.

[266] John V Basmajian and CJ De Luca. Muscles alive. *Muscles alive: their functions revealed by electromyography*, 278:126, 1985.

[267] Hong-Bo Xie, Tianruo Guo, Siwei Bai, and Socrates Dokos. Hybrid soft computing systems for electromyographic signals analysis: a review. *BioMedical Engineering OnLine*, 13(1):1–19, 2014. ISSN 1475-925X. doi: 10.1186/1475-925X-13-8. URL http://dx.doi.org/10.1186/1475-925X-13-8.

[268] Matteo Rossi, Simone Benatti, Elisabetta Farella, and Luca Benini. Hybrid EMG classifier based on hmm and svm for hand gesture recognition in prosthetics. In *Industrial Technology (ICIT), 2015 IEEE International Conference on*, pages 1700–1705. IEEE, 2015.

[269] Fei Sha, Lawrence K Saul, and Daniel D Lee. Multiplicative updates for nonnegative quadratic programming in support vector machines. In *Advances in neural information processing systems*, pages 1041–1048, 2002.

[270] S. Benatti, B. Milosevic, F. Casamassima, P. Schönle, P. Bunjaku, S. Fateh, Q. Huang, and L. Benini. EMG-based hand gesture recognition with flexible analog front end. In *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*, pages 57–60, Oct 2014. doi: 10.1109/BioCAS.2014.6981644.

[271] Zeeshan O Khokhar, Zhen G Xiao, Carlo Menon, et al. Surface EMG pattern recognition for real-time control of a wrist exoskeleton. *Biomedical engineering online*, 9(1):41, 2010.

[272] Dapeng Yang, Jingdong Zhao, Yikun Gu, Li Jiang, and Hong Liu. EMG pattern recognition and grasping force estimation: Improvement to the myocontrol of

multi-dof prosthetic hands. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 516–521. IEEE, 2009.

[273] Jamileh Yousefi and Andrew Hamilton-Wright. Characterizing emg data using machine-learning tools. *Computers in biology and medicine*, 51:1–13, 2014.

[274] Paul Kaufmann, Kevin Englehart, and Marco Platzner. Fluctuating emg signals: Investigating long-term effects of pattern matching algorithms. In *2010 Annual International Conference of the IEEE Engineering in Medicine and Biology*, pages 6357–6360. IEEE, 2010.

[275] Simone Benatti, Elisabetta Farella, Emanuele Gruppioni, and Luca Benini. Analysis of robust implementation of an emg pattern recognition based control. In *Proceedings of the International Joint Conference on Biomedical Engineering Systems and Technologies-Volume 4*, pages 45–54. SCITEPRESS-Science and Technology Publications, Lda, 2014.

[276] Texas Instruments, 2015. URL http://www.ti.com/lit/ds/symlink/ads1298.pdf.

[277] A. D. I. Falih, W. A. Dharma, and S. Sumpeno. Classification of emg signals from forearm muscles as automatic control using naive bayes. In *2017 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 346–351, Aug 2017. doi: 10.1109/ISITIA.2017.8124107.

[278] D. Kleyko, E. Osipov, A. Senior, A. I. Khan, and Y. A. Şekerciogğlu. Holographic graph neuron: A bioinspired architecture for pattern processing. *IEEE Transactions on Neural Networks and Learning Systems*, 28(6):1250–1262, June 2017. ISSN 2162-237X. doi: 10.1109/TNNLS.2016.2535338.

[279] stm32f427vg. http://www.st.com/resource/en/datasheet/stm32f427vg.pdf, 2016.

[280] ARM Cortex M4. https://developer.arm.com/products/processors/cortex-m/cortex-m4, 2013.

[281] OMAP processor. http://www.ti.com, 2013.

[282] A. Pullini, D. Rossi, I. Loi, A. Di Mauro, and L. Benini. Mr. wolf: A 1 gflop/s energy-proportional parallel ultra low power soc for iot edge processing. In *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, pages 274–277, Sept 2018. doi: 10.1109/ESSCIRC.2018.8494247.

[283] S. Benatti, G. Rovere, J. Bösser, F. Montagna, E. Farella, H. Glaser, P. Schönle, T. Burger, S. Fateh, Q. Huang, and L. Benini. 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)*, pages 139–144, June 2017. doi: 10.1109/IWASI.2017.7974234.

[284] A. Burrello, K. Schindler, L. Benini, and A. Rahimi. One-shot learning for iEEG seizure detection using end-to-end binary operations: Local binary patterns with hyperdimensional computing. In *Biomedical Circuits and Systems Conference (BioCAS), 2018 IEEE*, pages 1–4, 2018.

[285] A. Rahimi, S. Datta, D. Kleyko, E. P. Frady, B. Olshausen, P. Kanerva, and J. M. Rabaey. High-dimensional computing as a nanoscalable paradigm. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 64(9):2508–2521, Sept 2017. ISSN 1549-8328. doi: 10.1109/TCSI.2017.2705051.

[286] Thad Starner, Joshua Weaver, and Alex Pentland. Real-time american sign language recognition using desk and wearable computer based video. *IEEE Transactions on pattern analysis and machine intelligence*, 20(12):1371–1375, 1998.

[287] T Scott Saponas, Desney S Tan, Dan Morris, Ravin Balakrishnan, Jim Turner, and James A Landay. Enabling always-available input with muscle-computer interfaces. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, pages 167–176. ACM, 2009.

[288] Fabio Montagna, Simone Benatti, and Davide Rossi. Flexible, scalable and energy efficient bio-signals processing on the pulp platform: A case study on seizure detection. *Journal of Low Power Electronics and Applications*, 7(2):16, 2017.

[289] D. Rossi et al. A self-aware architecture for pvt compensation and power nap in near threshold processors. *IEEE Design Test*, 34(6):46–53, Dec 2017. ISSN 2168-2356.

[290] touch bionics. http://www.touchbionics.com/products, 2018.

[291] Ottobock. https://www.ottobockus.com/prosthetics/upper-limb-prosthetics/solution-overview/myoelectric-prosthetics/, 2018.

[292] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009. ISSN 1866-9956. doi: 10.1007/s12559-009-9009-8. URL http://dx.doi.org/10.1007/s12559-009-9009-8.

[293] Pentti Kanerva. Binary spatter-coding of ordered k-tuples. In , editor, *ICANN'96, Proceedings of the International Conference on Artificial Neural Networks*, volume 1112 of *Lecture Notes in Computer Science*, pages 869–873. Springer, 1996.

[294] Abbas Rahimi, Simone Benatti, Pentti Kanerva, Luca Benini, and Jan M. Rabaey. Hyperdimensional biosignal processing: A case study for EMG-based hand gesture recognition. In *IEEE International Conference on Rebooting Computing*, October 2016.

[295] A. Rahimi, P. Kanerva, L. Benini, and J. M. Rabaey. Efficient biosignal processing using hyperdimensional computing: Network templates for combined learning and classification of exg signals. *Proceedings of the IEEE*, pages 1–21, 2018. ISSN 0018-9219. doi: 10.1109/JPROC.2018.2871163.

[296] Abbas Rahimi, Pentti Kanerva, José del R Millán, and Jan M. Rabaey. Hyperdimensional computing for noninvasive brain–computer interfaces: Blind and one-shot classification of EEG error-related potentials. *10th ACM/EAI International Conference on Bio-inspired Information and Communications Technologies (BICT)*, 3 2017. doi: 10.4108/eai.22-3-2017.152397.

[297] D. Rossi et al. Energy-efficient near-threshold parallel computing: The pulpv2 cluster. *IEEE Micro*, 37(5):20–31, September 2017. ISSN 0272-1732. doi: 10.1109/MM.2017.3711645.

[298] P. Davide Schiavone et al. Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications. In *PATMOS*, pages 1–8, Sept 2017. doi: 10.1109/PATMOS.2017.8106976.

[299] Sidharth Pancholi and Amit M Joshi. Portable emg data acquisition module for upper limb prosthesis application. *IEEE Sensors Journal*, 18(8):3436–3443, 2018.

[300] M. R. Ahsan, M. I. Ibrahimy, and O. O. Khalifa. Electromygraphy (emg) signal based hand gesture recognition using artificial neural network (ann). In *2011 4th International Conference on Mechatronics (ICOM)*, pages 1–6, May 2011. doi: 10.1109/ICOM.2011.5937135.

[301] Janne M Hahne, F Biessmann, Ning Jiang, H Rehbaum, Dario Farina, FC Meinecke, K-R Müller, and LC Parra. Linear and nonlinear regression techniques for simultaneous and proportional myoelectric control. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 22(2):269–279, 2014.

[302] Ning Jiang, Kevin B Englehart, and Philip A Parker. Extracting simultaneous and proportional neural control information for multiple-dof prostheses from the

surface electromyographic signal. *IEEE transactions on Biomedical Engineering*, 56(4):1070–1080, 2009.

[303] Kevin R Wheeler, Mindy H Chang, and Kevin H Knuth. Gesture-based control and emg decomposition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 36(4):503–514, 2006.

[304] D. Farina and A. Holobar. Characterization of human motor units from surface emg decomposition. *Proceedings of the IEEE*, 104(2):353–373, 2016.

[305] Yu Hu, Yongkang Wong, Wentao Wei, Yu Du, Mohan Kankanhalli, and Weidong Geng. A novel attention-based hybrid cnn-rnn architecture for semg-based gesture recognition. *PloS one*, 13(10), 2018.

[306] Umberto Barone and Roberto Merletti. Design of a portable, intrinsically safe multichannel acquisition system for high-resolution, real-time processing hd-semg. *IEEE Transactions on Biomedical Engineering*, 60(8):2242–2252, 2013.

[307] Jian Wu, Lu Sun, and Roozbeh Jafari. A wearable system for recognizing american sign language in real-time using imu and surface emg sensors. *IEEE J. Biomedical and Health Informatics*, 20(5):1281–1290, 2016.

[308] Xilin Liu, Jacob Sacks, Milin Zhang, Andrew G Richardson, Timothy H Lucas, and Jan Van der Spiegel. The virtual trackpad: An electromyography-based, wireless, real-time, low-power, embedded hand-gesture-recognition system using an event-driven artificial neural network. *IEEE Trans. Circuits Syst. II Express Briefs*, 64: 1257–1261, 2017.

[309] Paolo Gentile, Marco Pessione, Antonio Suppa, Alessandro Zampogna, and Fernanda Irrera. Embedded wearable integrating real-time processing of electromyography signals. In *Multidisciplinary Digital Publishing Institute Proceedings*, page 600, 2017.

[310] Simone Benatti, Filippo Casamassima, Bojan Milosevic, Elisabetta Farella, Philipp Schönle, Schekeb Fateh, Thomas Burger, Qiuting Huang, and Luca Benini. A versatile embedded platform for emg acquisition and gesture recognition. *IEEE transactions on biomedical circuits and systems*, 9(5):620–630, 2015.

[311] Mohammadreza Asghari Oskoei, Huosheng Hu, et al. Support vector machine-based classification scheme for myoelectric control applied to upper limb. *IEEE Trans. Biomed. Engineering*, 55(8):1956–1965, 2008.

[312] Haoshi Zhang, Yaonan Zhao, Fuan Yao, Lisheng Xu, Peng Shang, and Guanglin Li. An adaptation strategy of using lda classifier for emg pattern recognition. In

Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE, pages 4267–4270. IEEE, 2013.

[313] Asim Waris, Imran Khan Niazi, Mohsin Jamil, Kevin Englehart, Winnie Jensen, and Ernest Nlandu Kamavuako. Multiday evaluation of techniques for emg based classification of hand motions. *IEEE journal of biomedical and health informatics*, 2018.

[314] Asim Waris, Irene Mendez, Kevin Englehart, Winnie Jensen, and Ernest Nlandu Kamavuako. On the robustness of real-time myoelectric control investigations: A multiday fitts' law approach. *Journal of Neural Engineering*, 2018.

[315] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. doi: 10.1145/130385.130401. URL http://doi.acm.org/10.1145/130385.130401.

[316] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[317] Halima Bensmail and Gilles Celeux. Regularized gaussian discriminant analysis through eigenvalue decomposition. *Journal of the American statistical Association*, 91(436):1743–1748, 1996.

[318] R. Meattini, S. Benatti, U. Scarcia, D. De Gregorio, L. Benini, and C. Melchiorri. An semg-based human–robot interface for robotic hands using machine learning and synergies. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 2018.

[319] Simone Benatti, Bojan Milosevic, Elisabetta Farella, Emanuele Gruppioni, and Luca Benini. A prosthetic hand body area controller based on efficient pattern recognition control strategies. *Sensors*, 17(4):869, 2017.

[320] Z Zainuddin, N Mahat, and Y Abu Hassan. Improving the convergence of the backpropagation algorithm using local adaptive techniques. In *International Conference on Computational Intelligence*, pages 173–176. Citeseer, 2004.

[321] A. J. Ishak, S. A. Ahmad, A. C. Soh, N. A. Naraina, R. M. R. Jusoh, and W. Chikamune. Design of a wireless surface emg acquisition system. In *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1–6, Nov 2017.

[322] Marie-Françoise Lucas, Adrien Gaufriau, Sylvain Pascual, Christian Doncarli, and Dario Farina. Multi-channel surface emg classification using support vector machines and signal-based wavelet optimization. *Biomedical Signal Processing and Control*, 3(2):169–174, 2008.

[323] Sebastian Bitzer and Patrick Van Der Smagt. Learning emg control of a robotic hand: towards active prostheses. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2819–2823. IEEE, 2006.

[324] Ahmet Alkan and Mücahid Günay. Identification of emg signals using discriminant analysis and svm classifier. *Expert Systems with Applications*, 39(1):44–47, 2012.

[325] Claudio Castellini, Emanuele Gruppioni, Angelo Davalli, and Giulio Sandini. Fine detection of grasp force and posture by amputees via surface electromyography. *Journal of Physiology-Paris*, 103(3-5):255–262, 2009.

[326] Claudio Castellini and Patrick van der Smagt. Surface emg in advanced hand prosthetics. *Biological cybernetics*, 100(1):35–47, Jan 2009.

[327] P. Zhang and J. Peng. Svm vs regularized least squares classification. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 1, pages 176–179 Vol.1, Aug 2004. doi: 10.1109/ICPR.2004.1334050.

[328] T. Scott Saponas et al. Making muscle-computer interfaces more practical. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 851–854, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753451. URL http://doi.acm.org/10.1145/1753326.1753451.

[329] Jun Liu, Fan Zhang, and He Helen Huang. An open and configurable embedded system for emg pattern recognition implementation for artificial arms. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 4095–4098. IEEE, 2014.

[330] Xiaorong Zhang, He Huang, and Qing Yang. Real-time implementation of a self-recovery emg pattern recognition interface for artificial arms. In *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5926–5929. IEEE, 2013.

[331] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[332] B. Milosevic, S. Benatti, and E. Farella. Design challenges for wearable emg applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1432–1437, March 2017. doi: 10.23919/DATE.2017.7927217.

[333] Bojan Milosevic, Simone Benatti, and Elisabetta Farella. Design challenges for wearable emg applications. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1432–1437. IEEE, 2017.

[334] Alessio Burrello, Daniele Jahier Pagliari, Andrea Bartolini, Luca Benini, Enrico Macii, and Massimo Poncino. Predicting hard disk failures in data centers using temporal convolutional neural networks. In *Euro-Par 2020: Parallel Processing Workshops*, volume 12480, page 277. Nature Publishing Group, 2020.

[335] T. Scott Saponas et al. Making muscle-computer interfaces more practical. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 851–854, 2010. ISBN 978-1-60558-929-9.

[336] Bernard Hudgins, Philip Parker, and Robert N Scott. A new strategy for multifunction myoelectric control. *IEEE transactions on biomedical engineering*, 40(1): 82–94, 1993.

[337] Hanrui Wang, Zhekai Zhang, and Song Han. Spatten: Efficient sparse attention architecture with cascade token and head pruning. *preprint arXiv:2012.09852*, 2020.

[338] `https://greenwaves-technologies.com/gap8_gap9/`, 2022.

[339] J Cacioppo et al. *The skeletomotor system.* Cambridge University Press, 1990.

[340] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[341] Alessio Burrello, Moritz Scherer, Marcello Zanghieri, Francesco Conti, and Luca 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)*, pages 1–6, 2021. doi: 10.1109/COINS51742.2021.9524173.

[342] A Krasoulis et al. Effect of user practice on prosthetic finger control with an intuitive myoelectric decoder. *Frontiers in neuroscience*, 13:891, 2019.

[343] `http://ninaweb.hevs.ch/DB8`, 2022.

[344] A Krasoulis et al. Myoelectric digit action decoding with multi-output, multi-class classification: an offline analysis. *Scientific reports*, 10(1):1–10, 2020.

[345] P Koch et al. Regression of hand movements from semg data with recurrent neural networks. In *2020 Int. Conf. EMBC*, 2020.

[346] Tianzhe Bao, Yihui Zhao, Syed Ali Raza Zaidi, Shengquan Xie, Pengfei Yang, and Zhiqiang Zhang. A deep kalman filter network for hand kinematics estimation using semg. *Pattern Recognition Letters*, 143:88–94, 2021.

[347] C Lea et al. Temporal convolutional networks for action segmentation and detection. In *IEEE Conf. CVPR*, 2017.

[348] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[349] `https://github.com/pulp-platform/nemo`, 2022.

[350] 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*, 13(3):516–528, 2019. doi: 10.1109/TBCAS.2019.2914476.

[351] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2021.

[352] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant, 2019.

[353] Felice T Sun, Martha J Morrell, and Robert E Wharen. Responsive cortical stimulation for the treatment of epilepsy. *Neurotherapeutics*, 5(1):68–74, 2008.

[354] Somayya Madakam, Vihar Lake, Vihar Lake, Vihar Lake, et al. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(05):164, 2015.

[355] J Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[356] S Rebuffi et al. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[357] T Hayes et al. Memory efficient experience replay for streaming learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9769–9776. IEEE, 2019.

[358] S Disabato et al. Incremental on-device tiny machine learning. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pages 7–13, 2020.

[359] D Kiyasseh et al. Clops: Continual learning of physiological signals. *arXiv preprint arXiv:2004.09578*, 2020.

[360] L Pellegrini et al. Latent replay for real-time continual learning (2019). *arXiv preprint arXiv:1912.01100*, 2019.

[361] L Ravaglia et al. Memory-latency-accuracy trade-offs for continual learning on a risc-v extreme-edge node. In *2020 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6. IEEE, 2020.

[362] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Reply to huszár: The elastic weight consolidation penalty is empirically valid. *Proceedings of the National Academy of Sciences*, 115(11):E2498–E2498, 2018.

[363] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet Things J.*, 3(5):637–646, Oct 2016. ISSN 2372-2541. doi: 10.1109/JIOT.2016.2579198.

[364] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. *ACM SIGARCH Computer Architecture News*, 45(2):548–560, 2017.

[365] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

[366] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices, 2020.

[367] ST Microelectronics. STM32 Cube IDE, 2022. URL https://www.st.com/en/development-tools/stm32cubeide.html.

[368] Shraman Ray Chaudhuri, Elad Eban, Hanhan Li, Max Moroz, and Yair Movshovitz-Attias. Fine-Grained Stochastic Architecture Search. *arXiv:2006.09581*, 2020.

[369] Alessio Burrello, Alberto Dequino, Daniele Jahier Pagliari, Francesco Conti, Marcello Zanghieri, Enrico Macii, Luca Benini, and Massimo Poncino. Tcn mapping optimization for ultra-low power time-series edge inference. In *Proc. IEEE/ACM ISLPED*, pages 1–6, 2021. doi: 10.1109/ISLPED52811.2021.9502494.

[370] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. *arXiv:1602.07360*, 2016.

[371] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.

[372] Ahmed Elthakeb, Prannoy Pilligundla, FatemehSadat Mireshghallah, Amir Yazdanbakhsh, Sicuan Gao, and Hadi Esmaeilzadeh. Releq: an automatic reinforcement learning approach for deep quantization of neural networks. In *NeurIPS ML for Systems workshop, 2018*, 2019.

[373] Daniele Palossi, Nicky Zimmerman, Alessio Burrello, Francesco Conti, Hanna Muller, Luca Maria Gambardella, Luca Benini, Alessandro Giusti, and Jerome Guzzi. Fully onboard ai-powered human-drone pose estimation on ultralow-power autonomous flying nano-uavs. *IEEE Internet of Things Journal*, 9(3):1913–1929, 2022. doi: 10.1109/JIOT.2021.3091643.

[374] Alessio Burrello, Daniele Jahier Pagliari, Pierangelo Maria Rapa, Matilde Semilia, Matteo Risso, Tommaso Polonelli, Massimo Poncino, Luca Benini, and Simone Benatti. Embedding temporal convolutional networks for energy-efficient ppg-based heart rate monitoring. *ACM Trans. Comput. Healthcare*, 3(2), 2022. ISSN 2691-1957. doi: 10.1145/3487910.

[375] Brian Ramprasad, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. Sustainable computing on the edge: A system dynamics perspective. In *ACM Proc. of HotMobile '21*, page 64–70, 2021. ISBN 9781450383233.

[376] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF ICCVW*, pages 3009–3018. IEEE, 2019.

[377] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620, 2019.

[378] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114. PMLR, 2019.

[379] Matteo Risso, Alessio Burrello, Francesco Conti, Lorenzo Lamberti, Yukai Chen, Luca Benini, Enrico Macii, Massimo Poncino, and Daniele Jahier Pagliari. Lightweight neural architecture search for temporal convolutional networks at the edge. *IEEE Transactions on Computers*, pages 1–1, 2022. doi: 10.1109/TC.2022.3177955.

[380] Francesco Daghero, Daniele Jahier Pagliari, and Massimo Poncino. Energy-efficient deep learning inference on edge devices. In Shiho Kim and Ganesh Chandra Deka, editors, *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, volume 122 of *Advances in Computers*, pages 247–301. Elsevier, 2021. doi: https://doi.org/10.1016/bs.adcom.2020.07.002.

[381] Bert Moons, Bert De Brabandere, Luc Van Gool, and Marian Verhelst. Energy-efficient convnets through approximate computing. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8, 2016. doi: 10.1109/WACV.2016.7477614.

[382] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Adv. Neural Inf. Process. Syst.*, 28, 2015.

[383] Alex Krizhevsky, Geoffrey Hinton, et al. Cifar-10, 2009. URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[384] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Proc. ECCV*, pages 740–755, 2014.

[385] Francesco Daghero, Daniele Jahier Pagliari, and Massimo Poncino. Energy-efficient deep learning inference on edge devices. In Shiho Kim and Ganesh Chandra Deka, editors, *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, volume 122 of *Advances in Computers*, chapter 8, pages 247–301. Elsevier, 2021. doi: https://doi.org/10.1016/bs.adcom.2020.07.002.

[386] Elmustafa sayed ali ahmed and Zeinab Kamal Aldein Mohammed. Internet of things applications, challenges and related future technologies. *world scientific news*, 01 2017.

[387] Leonardo Cecconi, Sander Smets, Luca Benini, and Marian Verhelst. Optimal tiling strategy for memory bandwidth reduction for cnns. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 89–100. Springer, 2017.

[388] Giuseppe Tagliavini, Germain Haugou, Andrea Marongiu, and Luca Benini. Adrenaline: An openvx environment to optimize embedded vision applications on many-core accelerators. In *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*, pages 289–296. IEEE, 2015.

[389] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. DMazeRunner: Executing Perfectly Nested Loops on Dataflow Accelerators. *ACM Trans. Embed. Comput. Syst.*, 18, October 2019. ISSN 1539-9087.

[390] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, 2017.

[391] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[392] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[393] Hongxing Gao, Wei Tao, Dongchao Wen, Tse-Wei Chen, Kinya Osa, and Masami Kato. Ifq-net: Integrated fixed-point quantization networks for embedded vision.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 607–615, 2018.

[394] Feng Zhu, Ruihao Gong, Fengwei Yu, Xianglong Liu, Yanfei Wang, Zhelong Li, Xiuqi Yang, and Junjie Yan. Towards unified int8 training for convolutional neural network. *arXiv preprint arXiv:1912.12607*, 2019.

[395] Bram-Ernst Verhoef, Nathan Laubeuf, Stefan Cosemans, Peter Debacker, Ioannis Papistas, Arindam Mallik, and Diederik Verkest. Fq-conv: Fully quantized convolution for efficient and accurate inference. *arXiv preprint arXiv:1912.09356*, 2019.

[396] Xiaying Wang, Michele Magno, Lukas Cavigelli, and Luca Benini. FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things. *IEEE Internet of Things Journal*, 2020.

[397] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64-mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, 6(5):8357–8371, 2019.

[398] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[399] Olakunle Elijah, Tharek Abdul Rahman, Igbafe Orikumhi, Chee Yen Leow, and MHD Nour Hindia. An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges. *IEEE Internet of Things Journal*, 5(5): 3758–3773, 2018.

[400] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using internet-of-things (iot) sensing with cloud-based processing: Opportunities and challenges. In *2015 IEEE International Conference on Services Computing*, pages 285–292. IEEE, 2015.

[401] SONY. Spresense, 2019. URL `https://developer.sony.com/develop/spresense/`.

[402] Naser Hossein Motlagh, Miloud Bagaa, and Tarik Taleb. UAV-based IoT platform: A crowd surveillance use case. *IEEE Communications Magazine*, 55(2):128–134, 2017.

[403] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News*, 45(2):1–12, June 2017. ISSN 0163-5964. doi: 10.1145/3140659.3080246. URL https://doi.org/10.1145/3140659.3080246.

[404] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 1–12, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450348928. doi: 10.1145/3079856.3080246. URL https://doi.org/10.1145/3079856.3080246.

[405] Francesco Conti, Manuele Rusci, and Luca Benini. The Memory Challenge in Ultra-Low Power Deep Learning. In *NANO-CHIPS 2030*, pages 323–349. Springer, 2020.

[406] James Manyika. *The Internet of Things: Mapping the value beyond the hype*. McKinsey Global Institute, 2015.

[407] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P. Sheth. Machine learning for internet of things data analysis: a survey. *Digital Communications and Networks*, 4(3):161–175, 2018. ISSN 2352-8648. doi: https://doi.org/10.1016/j.dcan.2017.10.002.

[408] Alessio Burrello, Francesco Conti, Angelo Garofalo, Davide Rossi, and Luca Benini. Work-in-Progress: DORY: Lightweight Memory Hierarchy Management for Deep NN Inference on IoT Endnodes. In *2019 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–2. IEEE, 2019.

[409] Neil Tan et al. μtensor, 2019. URL `https://github.com/uTensor/uTensor`.

[410] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549*, 2019.

[411] Google AI. Or tools, 2015. URL `https://developers.google.com/optimization/`.

[412] Christian Pinto and Luca Benini. A highly efficient, thread-safe software cache implementation for tightly-coupled multicore clusters. In *2013 IEEE 24th International Conference on Application-Specific Systems, Architectures and Processors*, pages 281–288. IEEE, 2013.

[413] Maurice Peemen, Arnaud AA Setio, Bart Mesman, and Henk Corporaal. Memory-centric accelerator design for convolutional neural networks. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 13–19. IEEE, 2013.

[414] Arthur Stoutchinin, Francesco Conti, and Luca Benini. Optimally scheduling cnn convolutions for efficient memory access. *arXiv preprint arXiv:1902.01492*, 2019.

[415] Francesco Conti, Davide Rossi, Antonio Pullini, Igor Loi, and Luca Benini. PULP: A Ultra-Low Power Parallel Accelerator for Energy-Efficient and Flexible Embedded Vision. *Journal of Signal Processing Systems*, 84(3):339–354, 2016.

[416] Selma Saidi, Pranav Tendulkar, Thierry Lepley, and Oded Maler. Optimizing two-dimensional DMA transfers for scratchpad Based MPSoCs platforms. *Microprocessors and Microsystems*, 37(8), 2013.

[417] Anjali S Yeole and Dhananjay R Kalbande. Use of internet of things (iot) in healthcare: A survey. In *Proceedings of the ACM Symposium on Women in Research 2016*, pages 71–76, 2016.

[418] Hao-Yu Jan, Mei-Fen Chen, Tieh-Cheng Fu, Wen-Chen Lin, Cheng-Lun Tsai, and Kang-Ping Lin. Evaluation of coherence between ecg and ppg derived parameters on heart rate variability and respiration in healthy volunteers with/without controlled breathing. *Journal of Medical and Biological Engineering*, 39(5):783–795, 2019.

[419] Neurosky, 2020. URL 'http://neurosky.com/wp-content/uploads/2016/06/TOF-side-by-side-competitor-comparison.pdf'.

[420] Aaron Lefohn, Mike Houston, Johan Andersson, Ulf Assarsson, Cass Everitt, Kayvon Fatahalian, Tim Foley, Justin Hensley, Paul Lalonde, and David Luebke. Beyond programmable shading (parts i and ii). In *ACM SIGGRAPH Courses*, pages 1–312, 2009. doi: 10.1145/1667239.1667246. URL http://doi.acm.org/10.1145/1667239.1667246.

[421] Anna Shcherbina, C Mikael Mattsson, Daryl Waggott, Heidi Salisbury, Jeffrey W Christle, Trevor Hastie, Matthew T Wheeler, and Euan A Ashley. Accuracy in wrist-worn, sensor-based measurements of heart rate and energy expenditure in a diverse cohort. *Journal of personalized medicine*, 7(2):3, 2017.

[422] Maarten Falter, Werner Budts, Kaatje Goetschalckx, Véronique Cornelissen, and Roselien Buys. Accuracy of apple watch measurements for heart rate and energy expenditure in patients with cardiovascular disease: Cross-sectional study. *JMIR mHealth and uHealth*, 7(3):e11889, 2019.

[423] NXP. Nxp lpc4300, 2022. URL https://www.nxp.com/products/processors-and-microcontrollers/arm-microcontrollers/general-purpose-mcus/lpc4300-cortex-m4-m0.

[424] STMicroelectronics. Mlcore, 2022. URL https://www.st.com/resource/en/application_note/dm00563460-lsm6dsox-machine-learning-core-stmicroelectronics.pdf.

[425] BiosignalsPLUX. Respiban professional 2019, 2019. URL https://biosignalsplux.com/products/wearables/respiban-pro.html.

[426] Empatica. E4 wristband 2014, 2022. URL https://www.empatica.com/en-eu/research/e4/.

[427] Zhilin Zhang. Iee signal processing cup 2015 dataset, 2022. URL https://sites.google.com/site/researchbyzhang/ieeespcup2015.

[428] Apple. Apple watch series, 2022. URL https://www.apple.com/lae/watch/.

[429] Fitbit. Fitbit charge 4, 2022. URL https://www.fitbit.com/global/us/products/trackers/charge4.

[430] ARM. Arm helium, 2017. URL https://www.arm.com/why-arm/technologies/helium.

[431] ST Microelectornics. Stm32h7, 2022. URL https://www.st.com/en/microcontrollers-microprocessors/stm32h7-series.html.

[432] ST Microelectornics. Stm32l476, 2022. URL https://www.st.com/resource/en/datasheet/stm32l476je.pdf.

[433] Steven N. Baldassano, Benjamin H. Brinkmann, Hoameng Ung, Tyler Blevins, Erin C. Conrad, Kent Leyde, Mark J. Cook, Ankit N. Khambhati, Joost B. Wagenaar, Gregory A. Worrell, and Brian Litt. Crowdsourcing seizure detection: algorithm development and validation on human implanted device recordings. *Brain*, 140(6):1680–1691, 2017.

[434] M. A. Bin Altaf and J. Yoo. A 1.83μj/classification, 8-channel, patient-specific epileptic seizure classification soc using a non-linear support vector machine. *IEEE Transactions on Biomedical Circuits and Systems*, 10(1):49–60, Feb 2016. ISSN 1932-4545. doi: 10.1109/TBCAS.2014.2386891.

[435] D. Sopic, A. Aminifar, and D. Atienza. e-Glass: A wearable system for real-time detection of epileptic seizures. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2018. doi: 10.1109/ISCAS.2018.8351728.

[436] Kaspar Schindler, Christian Rummel, Ralph G. Andrzejak, Marc Goodfellow, Frédéric Zubler, Eugenio Abela, Roland Wiest, Claudio Pollo, Andreas Steimer, and Heidemarie Gast. Ictal time-irreversible intracranial eeg signals as markers of the epileptogenic zone. *Clinical Neurophysiology*, 127(9):3051–3058, 2016. ISSN 1388-2457. doi: https://doi.org/10.1016/j.clinph.2016.07.001.

[437] P. P. Muhammed Shanir, Kashif Ahmad Khan, Yusuf Uzzaman Khan, Omar Farooq, and Hojjat Adeli. Automatic seizure detection based on morphological features using one-dimensional local binary pattern on long-term EEG. *Clinical EEG and Neuroscience*, 49(5):351–362, 2018. doi: 10.1177/1550059417744890.

[438] W. Zhou, Y. Liu, Q. Yuan, and X. Li. Epileptic seizure detection using lacunarity and bayesian linear discriminant analysis in intracranial EEG. *IEEE TBME*, 60 (12):3375–3381, Dec 2013. ISSN 0018-9294. doi: 10.1109/TBME.2013.2254486.

[439] Z. Zhang and K. K. Parhi. Low-complexity seizure prediction from iEEG/sEEG using spectral power and ratios of spectral power. *IEEE Transactions on Biomedical Circuits and Systems*, 10(3):693–706, June 2016. ISSN 1932-4545. doi: 10.1109/TBCAS.2015.2477264.

[440] Daniel M. Goldenholz, Amanda Kuhn, Alison Austermuehle, Martin Bachler, Christopher Mayer, Siegfried Wassertheurer, Sara K. Inati, and William H. Theodore. Long-term monitoring of cardiorespiratory patterns in drug-resistant epilepsy. *Epilepsia*, 58(1):77–84, 2017. doi: 10.1111/epi.13606.

[441] Judith van Andel, Constantin Ungureanu, Johan Arends, Francis Tan, Johannes Van Dijk, George Petkov, Stiliyan Kalitzin, Thea Gutter, Al de Weerd, Ben Vledder, Roland Thijs, Ghislaine van Thiel, Kit Roes, and Frans Leijten. Multimodal, automated detection of nocturnal motor seizures at home: Is a reliable seizure detector feasible? *Epilepsia Open*, 2(4):424–431, Dec 2017. ISSN 2470-9239. doi: 10.1002/epi4.12076.

[442] Xiang Zhang, Lina Yao, Xianzhi Wang, Jessica Monaghan, David Mcalpine, and Yu Zhang. A survey on deep learning based brain computer interface: Recent advances and new frontiers. *arXiv preprint arXiv:1905.04149*, 2019.

[443] Alessio Burrello, Simone Benatti, Kaspar Anton Schindler, Luca Benini, and Abbas Rahimi. An ensemble of hyperdimensional classifiers: Hardware-friendly short-latency seizure detection with automatic ieeg electrode selection. *IEEE journal of biomedical and health informatics*, 2020.

[444] Oh-Young Kwon and Sung-Pa Park. Depression and anxiety in people with epilepsy. *Journal of Clinical Neurology*, 10:175–188, 2014. ISSN 1738-6586.

[445] Ramy Hussein, Hamid Palangi, Rabab Ward, and Z Jane Wang. Robust detection of epileptic seizures using deep neural networks. In *Proc. IEEE ICASSP*, 2018.

[446] A. Page, C. Sagedy, E. Smith, N. Attaran, T. Oates, and T. Mohsenin. A flexible multichannel EEG feature extractor and classifier for seizure detection. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(2):109–113, Feb 2015. ISSN 1549-7747. doi: 10.1109/TCSII.2014.2385211.

[447] M. Shoaran, M. Farivar, and A. Emami. Hardware-friendly seizure detection with a boosted ensemble of shallow decision trees. In *Proc. IEEE EMBC*, pages 1826–1829, 2016. doi: 10.1109/EMBC.2016.7591074.

[448] Benjamin H. Brinkmann, Joost Wagenaar, Drew Abbot, Phillip Adkins, Simone C. Bosshard, Min Chen, Quang M. Tieng, Jialune He, F. J. Muñoz-Almaraz, Paloma Botella-Rocamora, Juan Pardo, Francisco Zamora-Martinez, Michael Hills, Wei Wu, Iryna Korshunova, Will Cukierski, Charles Vite, Edward E. Patterson, Brian Litt, and Gregory A. Worrell. Crowdsourcing reproducible seizure forecasting in human and canine epilepsy. *Brain*, 139(6):1713–1722, 2016. doi: 10.1093/brain/aww045.

[449] G. K. Bergey. Neurostimulation in the treatment of epilepsy. *Exp. Neurol.*, 244: 87–95, Jun 2013.

[450] P. Afra, C. C. Jouny, and G. K. Bergey. Duration of complex partial seizures: an intracranial EEG study. *Epilepsia*, 49(4):677–684, Apr 2008.

[451] Joel Mendez, Sarah Hood, Andy Gunnel, and Tommaso Lenzi. Powered knee and ankle prosthesis with indirect volitional swing control enables level-ground walking and crossing over obstacles. *Science Robotics*, 5(44), 2020.

[452] Alessio Burrello, Kaspar Anton Schindler, Luca Benini, and Abbas Rahimi. Hyperdimensional computing with local binary patterns: One-shot learning for seizure onset detection and identification of ictogenic brain regions from short-time ieeg recordings. *IEEE transactions on bio-medical engineering*, 67(2):601–613, 2020.

[453] K. Schindler, H. Gast, L. Stieglitz, A. Stibal, M. Hauf, R. Wiest, L. Mariani, and C. Rummel. Forbidden ordinal patterns of periictal intracranial EEG indicate deterministic dynamics in human epileptic seizures. *Epilepsia*, 52(10):1771–1780, Oct 2011.

[454] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *European Conference on Computer Vision*, pages 47–54. Springer, 2016.

[455] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, volume 89, pages 89–94. Presses universitaires de Louvain, 2015.

[456] Alessio Burrello, Davide Brunelli, Marzia Malavisi, and Luca Benini. Enhancing structural health monitoring with vehicle identification and tracking. In *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6. IEEE, 2020.

[457] Lei Ren, Yuxin Liu, Xiaokang Wang, Jinhu Lü, and M Jamal Deen. Cloud-edge based lightweight temporal convolutional networks for remaining useful life prediction in iiot. *IEEE Internet of Things Journal*, 2020.

[458] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[459] Thorir Mar Ingolfsson, Michael Hersche, Xiaying Wang, Nobuaki Kobayashi, Lukas Cavigelli, and Luca Benini. Eeg-tcnet: An accurate temporal convolutional network for embedded motor-imagery brain-machine interfaces, 2020.

[460] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[461] Yuan Shangguan, Jian Li, Qiao Liang, Raziel Alvarez, and Ian McGraw. Optimizing speech recognition for the edge, 2019.

[462] D. J. Pagliari, Roberta Chiaro, Yukai Chen, E. Macii, and M. Poncino. Optimal input-dependent edge-cloud partitioning for rnn inference. *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 442–445, 2019.

[463] Victor Kartsch, Giuseppe Tagliavini, Marco Guermandi, Simone Benatti, Davide Rossi, and Luca Benini. Biowolf: A sub-10-mw 8-channel advanced brain–computer interface platform with a nine-core processor and ble connectivity. *IEEE transactions on biomedical circuits and systems*, 13(5):893–906, 2019.

[464] K. Schindler, C. Rummel, R. G. Andrzejak, M. Goodfellow, F. Zubler, E. Abela, R. Wiest, C. Pollo, A. Steimer, and H. Gast. Ictal time-irreversible intracranial EEG signals as markers of the epileptogenic zone. *Clin Neurophysiol*, 127(9): 3051–3058, 09 2016.

[465] C. Geier, S. Bialonski, C. E. Elger, and K. Lehnertz. How important is the seizure onset zone for seizure dynamics? *Seizure*, 25:160–166, Feb 2015.

[466] C. Harden, T. Tomson, D. Gloss, J. Buchhalter, J. H. Cross, E. Donner, J. A. French, A. Gil-Nagel, D. C. Hesdorffer, W. H. Smithson, M. C. Spitz, T. S. Walczak, J. W. Sander, and P. Ryvlin. Practice guideline summary: Sudden unexpected death in epilepsy incidence rates and risk factors: Report of the Guideline Development, Dissemination, and Implementation Subcommittee of the American Academy of Neurology and the American Epilepsy Society. *Neurology*, 88(17): 1674–1680, Apr 2017.

[467] J. S. Naftulin, O. J. Ahmed, G. Piantoni, J. B. Eichenlaub, L. E. Martinet, M. A. Kramer, and S. S. Cash. Ictal and preictal power changes outside of the seizure focus correlate with seizure generalization. *Epilepsia*, 59(7):1398–1409, Jul 2018.

[468] N. M. Wetjen, W. R. Marsh, F. B. Meyer, G. D. Cascino, E. So, J. W. Britton, S. M. Stead, and G. A. Worrell. Intracranial electroencephalography seizure onset patterns and surgical outcomes in nonlesional extratemporal epilepsy. *J. Neurosurg.*, 110(6):1147–1152, Jun 2009.

[469] C. C. Jouny, P. J. Franaszczuk, and G. K. Bergey. Characterization of epileptic seizure dynamics using Gabor atom density. *Clin Neurophysiol*, 114(3):426–437, Mar 2003.

[470] S. Wiebe, W. T. Blume, J. P. Girvin, and M. Eliasziw. A randomized, controlled trial of surgery for temporal-lobe epilepsy. *N. Engl. J. Med.*, 345(5):311–318, Aug 2001.

[471] Y. Nagahama, A. J. Schmitt, D. Nakagawa, A. S. Vesole, J. Kamm, C. K. Kovach, D. Hasan, M. Granner, B. J. Dlouhy, M. A. Howard, and H. Kawasaki. Intracranial EEG for seizure focus localization: evolving techniques, outcomes, complications, and utility of combining surface and depth electrodes. *J. Neurosurg.*, pages 1–13, May 2018.

[472] Benjamin W Nelson, Carissa A Low, Nicholas Jacobson, Patricia Areán, John Torous, and Nicholas B Allen. Guidelines for wrist-worn consumer wearable assessment of heart rate in biobehavioral research. *NPJ Digital Medicine*, 3(1):1–9, 2020.

[473] M. Schmuck, L. Benini, and A. Rahimi. Hardware Optimizations of Dense Binary Hyperdimensional Computing: Rematerialization of Hypervectors, Binarized Bundling, and Combinational Associative Memory. *ArXiv e-prints*, July 2018.

[474] H. Shiao, V. Cherkassky, J. Lee, B. Veber, E. E. Patterson, B. H. Brinkmann, and G. A. Worrell. Svm-based system for prediction of epileptic seizures from iEEG signal. *IEEE Transactions on Biomedical Engineering*, 64(5):1011–1022, May 2017. ISSN 0018-9294. doi: 10.1109/TBME.2016.2586475.

[475] Felix Rosenow and Hans Lüders. Presurgical evaluation of epilepsy. *Brain*, 124 (9):1683–1700, 2001. doi: 10.1093/brain/124.9.1683. URL http://dx.doi.org/10.1093/brain/124.9.1683.

[476] Matthew A. Kelly, Dorothea Blostein, and D. J. K. Mewhort. Encoding structure in holographic reduced representations. *Canadian Journal of Experimental Psychology*, 67(2):79–93, 2013. doi: 10.1037/a0030301.

[477] Stephen Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7(2):123–169, 1986. ISSN 0196-8858.

[478] Turkey N. Alotaiby, Saleh A. Alshebeili, Tariq Alshawi, Ishtiaq Ahmad, and Fathi E. Abd El-Samie. EEG seizure detection and prediction algorithms: a survey. *EURASIP Journal on Advances in Signal Processing*, 2014(1):183, Dec 2014. ISSN 1687-6180. doi: 10.1186/1687-6180-2014-183.

[479] Yinxia Liu, Weidong Zhou, Qi Yuan, and Shuangshuang Chen. Automatic seizure detection using wavelet transform and svm in long-term intracranial EEG. *IEEE transactions on neural systems and rehabilitation engineering*, 20(6):749–755, 2012.

[480] Thomas De Cooman, Carolina Varon, Anouk Van de Vel, Katrien Jansen, Berten Ceulemans, Lieven Lagae, and Sabine Van Huffel. Adaptive nocturnal seizure detection using heart rate and low-complexity novelty detection. *Seizure*, 59:48–53, 2018. ISSN 1059-1311.

[481] Arends Johan B. A. M. Movement-based seizure detection. *Epilepsia*, 59(S1): 30–35, 2018. doi: 10.1111/epi.14053.

[482] Sriram Ramgopal, Sigride Thome-Souza, Michele Jackson, Navah Ester Kadish, Iván Sánchez Fernández, Jacquelyn Klehm, William Bosl, Claus Reinsberger, Steven Schachter, and Tobias Loddenkemper. Seizure detection, seizure prediction, and closed-loop warning systems in epilepsy. *Epilepsy & Behavior*, 37:291–307, 2014. ISSN 1525-5050.

[483] Han-Tai Shiao, Vladimir Cherkassky, Jieun Lee, Brandon Veber, Edward E Patterson, Benjamin H Brinkmann, and Gregory A Worrell. Svm-based system for prediction of epileptic seizures from iEEG signal. *IEEE Trans. Biomed. Engineering*, 64(5):1011–1022, 2017.

[484] Amir Hossein Ansari, Vladimir Matic, Maarten De Vos, Gunnar Naulaers, PJ Cherian, and Sabine Van Huffel. Improvement of an automated neonatal seizure detector using a post-processing technique. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 5859–5862. IEEE, 2015.

[485] Piyush Swami, Tapan K Gandhi, Bijaya K Panigrahi, Manjari Tripathi, and Sneh Anand. A novel robust diagnostic model to detect seizures in electroencephalography. *Expert Systems with Applications*, 56:116–130, 2016.

[486] Paula Branco, Luís Torgo, and Rita P Ribeiro. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, 49(2):31, 2016.

[487] DE Hernández, L Trujillo, E Z-Flores, OM Villanueva, and O Romo-Fewell. Detecting epilepsy in EEG signals using time, frequency and time-frequency domain

features. In *Computer Science and Engineering—Theory and Applications*, pages 167–182. Springer, 2018.

[488] Blerim Emruli, Fredrik Sandin, and Jerker Delsing. Vector space architecture for emergent interoperability of systems by learning from demonstration. *Biologically Inspired Cognitive Architectures*, 11:53–64, 2015. ISSN 2212-683X.

[489] P. Kanerva. Fully distributed representation. *In Real world computing symposium*, 97:358–365, 1997.

[490] D. Kleyko, A. Rahimi, D. A. Rachkovskij, E. Osipov, and J. M. Rabaey. Classification and recall with binary hyperdimensional computing: Tradeoffs in choice of density and mapping characteristics. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–19, 2018. ISSN 2162-237X. doi: 10.1109/TNNLS.2018.2814400.

[491] Ralph G. Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E. Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Phys. Rev. E*, 64:061907, Nov 2001. doi: 10.1103/PhysRevE.64.061907.

[492] T. Sunil Kumar, Vivek Kanhangad, and Ram Bilas Pachori. Classification of seizure and seizure-free EEG signals using local binary patterns. *Biomedical Signal Processing and Control*, 15:33–40, 2015. ISSN 1746-8094.

[493] V. Cherkassky, B. Veber, J. Lee, H. T. Shiao, N. Patterson, G. A. Worrell, and B. H. Brinkmann. Reliable seizure prediction from EEG data. In *Proc. IJCNN*, 2015. doi: 10.1109/IJCNN.2015.7280327.

[494] O. Räsänen. Generating Hyperdimensional Distributed Representations from Continuous Valued Multivariate Sensory Input. In *Proceedings of the 37th Annual Meeting of the Cognitive Science Society*, pages 1943–1948, 2015.

[495] Andriy Temko. Accurate heart rate monitoring during physical exercises using ppg. *IEEE Transactions on Biomedical Engineering*, 64(9):2016–2024, 2017.

[496] Pritam Sarkar and Ali Etemad. Cardiogan: Attentive generative adversarial network with dual discriminators for synthesis of ecg from ppg. *arXiv preprint arXiv:2010.00104*, 2020.

[497] Zhilin Zhang. Photoplethysmography-based heart rate monitoring in physical activities via joint sparse spectrum reconstruction. *IEEE transactions on biomedical engineering*, 62(8):1902–1910, 2015.

[498] Mahdi Boloursaz Mashhadi, Ehsan Asadi, Mohsen Eskandari, Shahrzad Kiani, and Farokh Marvasti. Heart rate tracking using wrist-type photoplethysmographic (ppg) signals during physical exercise with simultaneous accelerometry. *IEEE Signal Processing Letters*, 23(2):227–231, 2015.

[499] Dwaipayan Biswas, Luke Everson, Muqing Liu, Madhuri Panwar, Bram-Ernst Verhoef, Shrishail Patki, Chris H Kim, Amit Acharyya, Chris Van Hoof, Mario Konijnenburg, et al. Cornet: Deep learning framework for ppg-based heart rate estimation and biometric identification in ambulant environment. *IEEE transactions on biomedical circuits and systems*, 13(2):282–291, 2019.

[500] Heewon Chung, Hooseok Lee, and Jinseok Lee. Finite state machine framework for instantaneous heart rate validation using wearable photoplethysmography during intensive exercise. *IEEE journal of biomedical and health informatics*, 23(4):1595–1606, 2018.

[501] Dwaipayan Biswas, Neide Simões-Capela, Chris Van Hoof, and Nick Van Helleputte. Heart rate estimation from wrist-worn photoplethysmography: A review. *IEEE Sensors Journal*, 19(16):6560–6570, 2019.

[502] Hang Sik Shin, Chungkeun Lee, and Myoungho Lee. Adaptive threshold method for the peak detection of photoplethysmographic waveform. *Computers in biology and medicine*, 39(12):1145–1152, 2009.

[503] X. Chang, G. Li, L. Tu, G. Xing, and T. Hao. Deepheart: Accurate heart rate estimation from ppg signals based on deep learning. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 371–379, 2019.

[[van Gent] t al.(2019)[van Gent], Farah, [van Nes], and [van Arem]]RollingMean2019 Paul [van Gent], Haneen Farah, Nicole [van Nes], and Bart [van Arem]. Heartpy: A novel heart rate algorithm for the analysis of noisy signals. *Transportation Research Part F: Traffic Psychology and Behaviour*, 66:368–378, 2019. ISSN 1369-8478. doi: https://doi.org/10.1016/j.trf.2019.09.015. URL `http://www.sciencedirect.com/science/article/pii/S1369847818306740`.

[505] Puyudi Yang, Cho-Jui Hsieh, and Jane-Ling Wang. History pca: A new algorithm for streaming pca, 2018.

[506] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[507] Eunhyeok Park, Dongyoung Kim, Soobeom Kim, Yong-Deok Kim, Gunhee Kim, Sungroh Yoon, and Sungjoo Yoo. Big/little deep neural network for ultra low power inference. In *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 124–132. IEEE, 2015.

[508] Leandro Giacomini Rocha, Dwaipayan Biswas, Bram-Ernst Verhoef, Sergio Bampi, Chris Van Hoof, Mario Konijnenburg, Marian Verhelst, and Nick Van Helleputte. Binary cornet: accelerator for hr estimation from wrist-ppg. *IEEE Transactions on Biomedical Circuits and Systems*, 14(4):715–726, 2020.

[509] Z Ge, PWC Prasad, N Costadopoulos, Abeer Alsadoon, AK Singh, and A Elchouemi. Evaluating the accuracy of wearable heart rate monitors. In *2016 2nd International Conference on Advances in Computing, Communication, & Automation (ICACCA)(Fall)*, pages 1–6. IEEE, 2016.

[510] Daniele Jahier Pagliari, Enrico Macii, and Massimo Poncino. Dynamic Bit-width Reconfiguration for Energy-Efficient Deep Learning Hardware. In *Proceedings of the International Symposium on Low Power Electronics and Design*, ISLPED '18, New York, NY, USA, 2018. ACM. URL http://doi.acm.org/10.1145/3218603.3218611.

[511] L. Mocerino and A. Calimera. Fast and accurate inference on microcontrollers with boosted cooperative convolutional neural networks (bc-net). *IEEE Transactions on Circuits and Systems I: Regular Papers*, pages 1–12, 2020. doi: 10.1109/TCSI.2020.3039116.

[512] Francesco Daghero, Alessio Burrello, Daniele Jahier Pagliari, Luca Benini, Enrico Macii, and Massimo Poncino. Energy-Efficient Adaptive Machine Learning on IoT End-Nodes With Class-Dependent Confidence. In *Proceedings of the International Conference on Electronics Circuits and Systems*, ICECS 2020, pages 1–4. IEEE, 2020.

[513] Mahmoud Tavakoli, Carlo Benussi, Pedro Alhais Lopes, Luis Bica Osorio, and Anibal T de Almeida. Robust hand gesture recognition with a double channel surface emg wearable armband and svm classifier. *Biomedical Signal Processing and Control*, 46:121–130, 2018.

[514] Bardienus P Duisterhof, Srivatsan Krishnan, Jonathan J Cruz, Colby R Banbury, William Fu, Aleksandra Faust, Guido CHE de Croon, and Vijay Janapa Reddi. Learning to seek: Autonomous source seeking with deep reinforcement learning onboard a nano drone microcontroller. *arXiv preprint arXiv:1909.11236*, 2019.