A Construction Kit for Efficient Low Power Neural Network Accelerator Designs

PETAR JOKIC, CSEM, Switzerland and ETH Zurich, Switzerland
ERFAN AZARKHISH, ANDREA BONETTI, MARC PONS, and STEPHANE EMERY, CSEM, Switzerland
LUCA BENINI, ETH Zurich, Switzerland and University of Bologna, Italy

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

In contrast to previous surveys, this work provides a *quantitative* overview of neural network accelerator optimization approaches that have been used in recent works and reports their *individual* effects on edge processing performance. The list of optimizations and their quantitative effects are presented as a construction kit, allowing to assess the design choices for each building block individually. Reported optimizations range from up to 10,000× memory savings to 33× energy reductions, providing chip designers with an overview of design choices for implementing efficient low power neural network accelerators.

CCS Concepts: • **Computing methodologies → Neural networks**; • **Hardware → Application specific integrated circuits**;

Additional Key Words and Phrases: Edge processing, hardware accelerator, design optimization

ACM Reference format:
Petar Jokic, Erfan Azarkhish, Andrea Bonetti, Marc Pons, Stephane Emery, and Luca Benini. 2022. A Construction Kit for Efficient Low Power Neural Network Accelerator Designs. *ACM Trans. Embedd. Comput. Syst.* 21, 5, Article 56 (October 2022), 36 pages.
https://doi.org/10.1145/3520127

This work was supported in part by the Electronic Components and Systems for European Leadership Joint Undertaking ANDANTE under Grant 876925 and in part by the Swiss National Science Foundation (SNSF) BRIDGE under Grant 40B2-0_181010.

Authors' addresses: P. Jokic, CSEM: Technoparkstrasse 1, Zurich, ZH (state), 8005, Switzerland; email: petar.jokic@csem.ch; E. Azarkhish, A. Bonetti, M. Pons, and S. Emery, CSEM, Technoparkstrasse 1, Zurich, ZH, 8005, Switzerland; emails: {erfan.azarkhish, andrea.bonetti, marc.pons, stephane.emery}@csem.ch; L. Benini, ETH Zurich: Gloriastrasse 35, Zurich, ZH, 8092, Switzerland, Switzerland and University of Bologna: Viale del Risorgimento 2, Bologna, BO, 40136, Italy; email: lbenini@iis.ee.ethz.ch.

1539-9087/2022/10-ART56 $15.00
https://doi.org/10.1145/3520127

1 INTRODUCTION

Machine learning (ML) algorithms, especially **neural networks (NN)**, have been widely used to provide smart systems with complex data analysis capabilities like visual object detection [1] and audio key-word spotting [2]. While NNs enable algorithm developers to implement difficult-to-model tasks, given that sufficient training data is available, their computational complexity is challenging the design of processing hardware. Miniaturized and battery-powered ML applications thus require high computational throughput while being limited to low power consumption, rendering computing efficiency a key design objective for low power ML accelerators. The rapid progress of ML algorithm research further challenges designers to quickly adopt newly introduced network features, requiring fast development times. While **fully-connected (FC)** and **convolutional neural networks (CNN)** like AlexNet [3] have dominated the field during the past decade, **residual networks (ResNet)** [4], **recurrent NNs (RNN)** and derivations like dense CNNs [5] have gained importance, claiming ever-improving algorithm performance. What remains constant are the working horses of NNs, namely parallelized **multiply-and-accumulate (MAC)** operations.

Improving computing efficiency has been a key driver for the invention of laptops (~2000), smartphones (~2010) and high-performance server operation [6]. The current decade (~2020) strives for ubiquitous smart devices like wearables and **internet-of-things (IoT)** nodes [7], capable of processing sensory data. Performing data analysis on the sensor nodes at the edge of a connected network, so-called edge processing [8] or edge intelligence [9], can significantly reduce latency and power consumption but requires efficient ML accelerators.

Thus, edge processing is increasingly used in applications where long battery lifetimes are mandatory, e.g. in smart glasses with object detection [10], face detection [11], or hand-gesture and speech recognition [12], as well as smart cameras with automatic acquisition using scene classification [13], smart doorbells with face recognition [14], or tools that help blind people read texts and recognize people [15]. Many more could benefit from edge ML in the future [16, 17]. An overview of ML applications that are feasible on current hardware platforms is summarized in [18], illustrating the challenge of the limited edge processing power budget (<1W).

Existing ML accelerator chips cover the application domain from **ultra-low power (ULP)** and low-complexity processing, implementing 18uW key-word spotting with 105kB on-chip memory [19], to high-throughput server-grade acceleration, provided by chips like Google TPUv3 [20] or Graphcore IPU [21], consuming more than 100W. Allowing edge processing devices to run more complex ML applications, which can currently only be computed in cloud servers, requires more efficient edge ML accelerators. To identify relevant accelerator designs among the vast number and diversity of publications proposing efficient implementations, quantitative surveys are necessary. However, existing surveys often only provide qualitative comparisons or benchmark implementations on a system-level, obscuring the individual effects of each employed optimization technique. Comparing these optimizations is essential for motivating design choices during the development of new accelerators, currently requiring time-consuming literature research.

This work summarizes and, for the first time, quantitatively compares design optimizations of existing NN accelerators for tiny (<10mW) and edge (<1W) processing applications. It is presented as a construction kit, listing optimization options for each building block along with their reported quantitative effects, enabling **application-specific integrated circuits (ASIC)** designers to evaluate and assess optimizations for new implementations. Figure 1 illustrates the covered building blocks on a generic end-to-end edge processing system and localizes optimizations within the edge ML design flow.

The paper is organized as follows: Section 2 presents related surveys that complement this work. Section 3 introduces basic notations used throughout the paper, followed by an overview

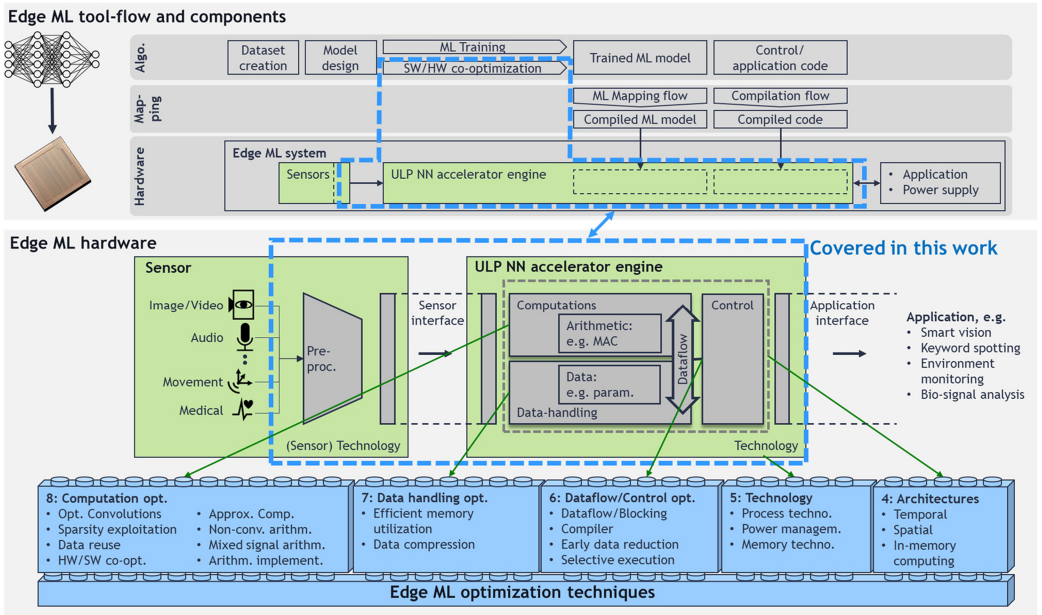

Fig. 1. Overview of the edge ML tool flow with a summary of the discussed hardware optimizations for NN accelerators (with chapter numbers of the paper).

of architectures in Section 4, technological optimizations in Section 5, dataflow optimizations in Section 6, data handling optimizations in Section 7, computation optimizations in Section 8, and finally, the quantitative comparison of all relevant optimizations in Section 9.

2 RELATED WORK

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

To keep track of the vast number of academic and industrial ML hardware accelerators proposed every year, a periodically updated online list [22] is provided, listing the main performance metrics for each chip to compare them in terms of power, throughput, and computational efficiency. A similar survey in paper form was presented in 2019 [23] and updated in 2020 [24]. It lists the computational performance and precision of academic research works and commercially available devices. A survey from 2017 claims to cover the past 35 years of works in neuromorphic computing, listing more than 2,600 references that accelerated the research field since the 1980s [25]. It covers models, learning approaches and hardware ranging from analog to digital implementations, covering programmable FPGAs and custom ASICs.

Sze et al. [26] provide a thorough survey on efficient processing of **deep NNs (DNN)**, covering historic aspects, common layer types, training frameworks and popular datasets, extending their previous work [27]. The sections on hardware platforms and energy efficient dataflows are motivated in their preceding work [28], proposing edge processing for extracting meaningful information to reduce the extreme amount of data produced by the ever-increasing number of sensors in connected devices. A similar summary is presented in [29] and a more FPGA-focused one in [30].

Another well-structured and exhaustive survey on DNN acceleration is presented in [31], covering existing hardware acceleration approaches including software optimizations, and a chapter on security of DNN approaches and their benchmarking. Survey [32] presents a broad overview of

Table 1. Selected ML Accelerator Chips from Previous Years

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

*including skipped operations.

the ML field, focusing on big data, training techniques, and applications. A similarly broad view, additionally covering the transition from modelling biological NNs to implementing artificial NNs in hardware is presented in [33], focusing on novel memories and their use-cases in the field.

In the survey of [34], various ML accelerators and processing blocks are presented and compared to their research group's own works. They list neuromorphic processors, including spiking NN engines, ranging from fully digital to fully analog computations, and discuss possible future directions. Survey [35] also discusses the architectures of selected DNN accelerators and explains their working principles and supported networks.

Surveys [9] and [36] present an extended view on edge intelligence, covering pure edge processing and combinations with cloud processing. They discuss related optimization strategies, namely compression and early model exit.

While the listed related works give an excellent overview of existing ML accelerators and optimization techniques, none of them attempted to quantitatively compare used optimization approaches, as covered in this work, allowing designers to evaluate and assess optimizations for new implementations.

This work focuses on (deep) NN inference, noting that the field of ML is much broader, containing other approaches like support vector machines, decision trees and many others.

3 CONSTRUCTION KIT FOREWORD

ML accelerators often combine multiple optimization techniques, as shown in Table 1, summarizing a selection of relevant accelerator chips from the past six years. This complicates the assessment of individual optimizations as their effects are obscured by system-level benchmarking. Thus, we only add individual optimizations to the comparison and only if sufficient quantitative information is reported. We focus on five performance indicators, namely (1) energy/power, (2) area (cost), (3) memory size, (4) computational throughput, and (5) impact on algorithmic accuracy.

In the following sections, we briefly discuss the importance of a system-level view for meaningful optimization evaluations and introduce some basic performance indicators and notations that are used throughout the paper.

3.1 System-Level View

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

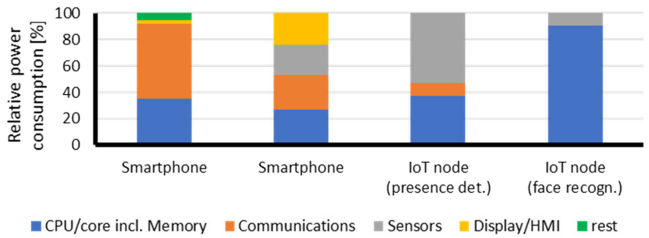

Fig. 2. Power break-down of mobile/edge systems (smartphones [44], presence detection IoT node [45], and face recognition IoT node [46]).

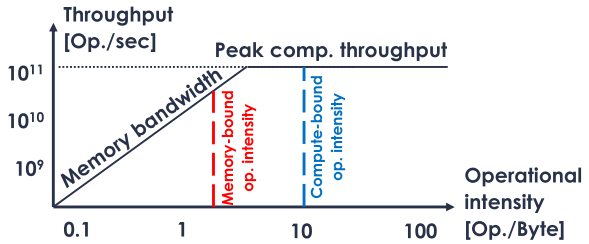

Fig. 3. Roofline plot showing two operating points, constrained by the available memory bandwidth and the computational throughput.

power-dominating sub-systems, allowing to optimize them based on the Pareto principle. Figure 2 shows the power breakdown of four mobile system implementations: two smartphone analyses were taken from a smartphone battery usage review [44], showing dominating communication power, while two IoT nodes were evaluated in a visual presence detection system [45] and an always-on face recognition system [46], reporting dominating sensor and processing power. While core optimizations would only enable marginal system improvements in the first three systems, the fourth example shows a typical edge ML IoT node with processing-dominated power distribution, enabling significant system improvements through core optimizations.

3.2 Meaningful Performance Indicators and Metrics

To identify useful optimization strategies, relevant performance indicators and benchmarks must be chosen. Parameter quantization, for example, can easily reduce memory size but also heavily impacts accuracy [47, 48]. Similarly, the throughput is subject to large variations across different workloads as pointed out in [23], reporting 20× lower throughput than stated in the datasheet. Thus, application-relevant benchmarks are indispensable. For a fair cross-device comparison, standard ML benchmarks have been created for smartphones [49] and for general purpose devices (MLPerf) [50] and are now adopted to edge devices (TinyMLPerf) [51]. This is similar to microcontroller benchmarks (e.g. CoreMark [52]).

Roofline models [53] are used to visualize the architectural boundaries of a system's operating points, namely the memory bandwidth and the peak computational throughput, as illustrated in Figure 3. Depending on the operating point, the system operates in a memory- or a computation-bound region.

NN accelerator performance metrics are often limited to the peak throughput, in **tera operations per second (TOPS)**, and the computational power efficiency (TOPS/W). Figure 4 illustrates that these metrics can be misleading in edge applications, operating in the low power region where extrapolating the efficiency becomes inaccurate (idle power).

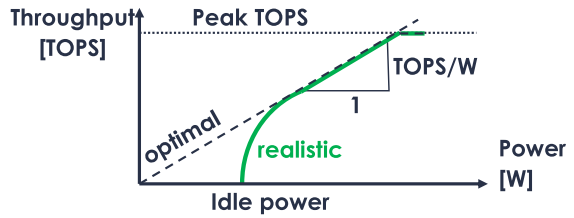

Fig. 4. NN processing throughput versus power consumption.

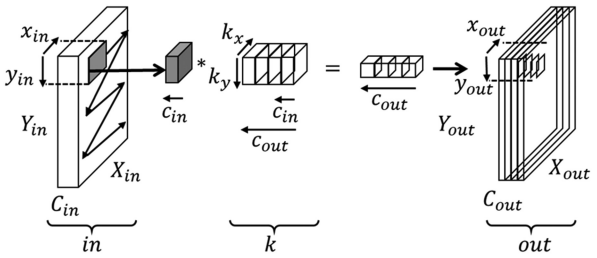

Fig. 5. Generic CNN layer with dimension notations.

3.3 Neural Network Notations

Figure 5 illustrates a generic NN layer along with the notations of layer dimensions that we use throughout the paper. The input activations *in* have C_{in} input channels and extend $X_{in} \cdot Y_{in}$ in spatial dimensions. All C_{out} kernels contain $C_{in} \cdot k_x \cdot k_y$ parameters and generate output activations *out* of dimension $X_{out} \cdot Y_{out}$ with C_{out} output channels. While a generic convolution layer is shown here, it also covers dense (FC) layers, which have restricted dimensions $k_x = k_y = X_{in} = Y_{in}$ and $X_{out} = Y_{out} = 1$. Other layer types can be described using the same notation, e.g. ResNets have additional bypass inputs in_{by} of the same dimension as *in* and are added point-wise. Depth-wise separable CNNs split kernels in the C_{in} dimension, yielding $C_{out} = C_{in}$, avoiding cross-channel links.

4 HARDWARE ARCHITECTURES

Two main architectural concepts are dominating today's NN accelerators [27]: temporal and spatial architectures. This section discusses them and adds emerging in-memory computing as a third concept, offering a distinct data flow. While temporal architectures sequentially access the memory, process the data, and finally return the results, spatial architectures aim at reducing these data movements by sharing activations and parameters across neighboring processing elements. In-memory computing offers an orthogonal concept that tries to entirely remove data movements by computing data directly inside the memory.

4.1 Temporal Architecture

Temporal architectures comprise, among others, the wide-spread **central processing units (CPU)** and **graphic processing units (GPU)**, featuring a central control unit that schedules tasks and distributes computations across **arithmetic logic units (ALU)**. Data is moved from the memory to the ALU and back, offering a high parallelization potential using **single instruction, multi-data (SIMD)** instructions for vector processing. This is based on the traditional Von Neuman architecture [54], characterized through a generic **single instruction, single data (SISD)** processing scheme that stores both instructions and data in the memory. It typically features a

control unit that reads instructions and data from the memory, an ALU to perform the operations, and a data bus to communicate across blocks and through a peripheral interface.

Multi-issue processors [55], e.g. super-scalar processors, extend this concept by enabling multiple parallel operations through a single instruction, reducing the control overhead, to increase the operational intensity (see Figure 3).

4.2 Spatial Architecture

In contrast to temporal architectures, spatial architectures allow their arithmetic units, often called **processing elements (PE)**, to move data between neighboring PEs, allowing to reduce memory accesses by employing local buffers.

Systolic arrays are pipelined 2D spatial architectures that enable data reuse across neighboring PEs. This can be exploited for implementing efficient **general matrix multiplication (GEMM)** through contraction ("systole" in old Greek) of computations, reusing results from adjacent nodes, and thus minimizing memory accesses. This is exploited in many prominent accelerators like Google TPU [56] and Eyeriss [57]. Replacing single-PE systolic arrays with tensor-PEs, each one computing an entire matrix multiplication per cycle, further allows reducing area and power in a 16nm process by 2.1× and 1.4x, respectively [58]. Increased intra-PE data reuse and fewer pipeline buffer registers enable this improvement but require efficient load distribution to avoid low PE utilization. SCALE-Sim [59] provides a simulation tool to evaluate such design parameters, comparing different dataflows and arrays.

4.3 In-memory Computing

The motivation behind in-memory computing, or **compute-in-memory (CIM)**, is the data-intensive nature of neural network inference, requiring high memory bandwidths which result in memory-dominated performance [60], the so-called memory wall [61]. To mitigate this, computations can be directly performed in the memory, where high data access rates are available at a much lower power cost. While the computational efficiency can be largely increased with this approach, it increases the overhead in the memory and limits the flexibility. Combined with analog memory types, the efficiency can be further improved by computing directly in the analog domain. Analog CIM computes matrix multiplications by breaking them down into vector dot products [62], multiplying analog input voltages (activations) with NVM cell conductances (weights) and accumulating the resulting column current (MAC result). Special design considerations to achieve high accuracy inference along with high computational efficiency are discussed in [60].

SRAM-based CIM is presented in [63], using a 55nm 8T 3.8kb SRAM macro with support for 1–4b input activation and 1–5b weights. The 130nm circuit in [64] demonstrates simple down-sampled MNIST computations, reporting 13× system energy reduction for 1b weight and 5b activation precision compared to a traditional (out-of-memory) computation. CIM integrated in the analog SRAM periphery is evaluated in [65] on a simulated 65nm process, reporting 4.9× system energy efficiency and 2.4× throughput improvement compared to digital processing. A 384kb SRAM-based 8b precision CIM in 28nm [66] is demonstrated to have 28% area overhead compared to a pure SRAM array while achieving up to 22.75TOPS/W throughput.

Non-volatile memory (NVM)-based CIM implementations are summarized in [62]. Their previous survey [67] provides a comprehensive list of the utilized emerging NVMs, illustrating that NVM can increase bit density and reduce leakage compared to SRAM and possibly store multiple bits per cell. In [68] an MRAM-based 54×108 CIM crossbar is presented in a 180nm process (MRAM on top of CMOS circuit). Over-lifetime variations of up to 4.2% and device-to-device variations of 4.5% have been reported, requiring special considerations (e.g. [69]).

Among many other RRAM works, a 1Mb multibit CIM on a 55nm process [70] and a 128×64b CIM array [71] on a 90nm process are presented. NVM-based CIM is an active field of research with various directions, for example 3D CIM architectures [72], reporting up to 28.6× higher energy efficiency compared to 2D chips. A CIM benchmarking tool [73] further reports advantages of NVM- over SRAM-based CIM implementations in a 32nm process, while 7nm SRAM CIM still outperforms any NVM-based work in throughput and both area and energy efficiency.

Note that also DRAM has been used for CIM [74], however, targeting high performance server applications which goes beyond the scope of this work.

5 TECHNOLOGY

Integrated circuits (IC) for NN accelerators are strongly influenced by the underlying semiconductor technology. This chapter introduces common process technologies and related power optimization techniques that are employed in various optimizations throughout the paper, followed by an overview of memories, which are often linked to process technologies and tend to dominate the performance of ML hardware accelerators (see Section 3.1).

5.1 Semiconductor Process Technology

Annual improvements in semiconductor manufacturing technology have been a major driving force in the chip industry as indicated by the (now saturating) Moore's law [75, 76], empirically predicting the doubling of component density on chips every 1–2 years. However, the smaller process nodes increased the static power consumption, resulting in the end of Dennard scaling [77]. This related heuristic scaling trend factor describes the annual shrinking of the minimum feature size in silicon chips and related power savings, culminating in the so-called power wall [78], limiting process improvements because the increased power density has approached physical limits of silicon-based circuits over the past few years.

This process scaling enables significant power reductions [79], as it yields lower supply voltages and smaller switching capacitances. However, improved process technologies are often linked to a significant cost increase and might not support all features of older technologies (e.g. special memory types, photo diodes, etc.). Multi-die solutions can mitigate this problem, exploiting the properties of multiple processes across the dies, each one optimized for a specific target like reduced cost, specialized memory support, or high logic density [17, 80]. Combining multi-process solutions with 3D die stacking additionally provides short communication paths and increased densities, as shown in the 8-die-stacked NN accelerator with 96MB of memory [81].

The following sub-sections give a brief introduction of the main semiconductor process technologies used today as they will be referred to throughout the paper. We limit the scope to **complementary metal-oxide-semiconductor (CMOS)** technology, which dominates digital designs and refer the interested reader to the annual IEEE white paper on future directions of semiconductor technologies (e.g. the 2020 update [82]) for a look into possible future directions.

5.1.1 Bulk. Bulk technology is based on standard silicon wafers and mainly evolves through spatial scaling. However, these annual scaling improvements are slowing down due to increasing difficulties with electrostatic and short-channel effects [83].

Deeply depleted channel (DDC) technology improves bulk technology by introducing multiple vertical doping regions in the channel, forming a threshold setting region and a bias-controllable screening region [84]. This enables reduced supply voltages, low leakage, low process variation as well as improved body biasing characteristics, allowing to dynamically adjust the threshold voltage of the transistor.

5.1.2 FinFET. The introduction of 3D gates in so-called FinFETs, improved on the performance of bulk CMOS by enabling lower supply voltages, and thus reduced power consumption, as shown for 22nm tri-gate FinFET [83] and later 14nm FinFET [85]. However, the improved performance comes at a higher production cost due to the complex 3-dimensional structures, requiring more fabrication masks than the bulk technology [86–88].

5.1.3 FD-SOI. **Fully depleted silicon-on-insulator (FD-SOI)** technology employs very thin insulating layers in the substrate of the transistors, reducing leakage. In [87] a 22nm FD-SOI technology is presented, achieving on par power and performance efficiency compared to 16/14nm FinFET technology while being lower cost due to the use of a planar processes, requiring fewer masks. Thus, FD-SOI is more suitable for low-end mobile and IoT applications where cost is an important factor. A detailed comparison between FD-SOI and FinFET is provided in [86], reporting superior performance of FinFETs in terms of power, delay, and density, for which FD-SOI can compensate through **body-biasing (BB)**. BB allows to dynamically adjust the threshold voltage, boosting speed with a forward body bias or reducing leakage using reverse body biasing (higher threshold).

5.1.4 Specialized Processes. Recent advances in materials and manufacturing technologies have enabled the integration of novel memory technologies (both volatile and non-volatile) close to processing logic. They offer advantageous characteristics that go beyond transistor density scaling. However, most of them require special process technologies, making them more difficult to integrate within the widely available processes. More details on novel memories can be found in Section 5.3.

5.2 Power Management

The power consumption of ICs [78] can be decomposed into dynamic and static power. Dynamic power is described in Equation (1), taking the circuit switching operations into account. Variable U is the supply voltage, C the switching capacitance, α the switching activity and f the frequency of the circuit. Static power, often also called leakage power, is described in Equation (2), reflecting the current consumption I_{leak} when no switching takes place.

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

$$P_{static} \sim U \cdot I_{leak} \tag{2}$$

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

5.2.1 Sub- and Near-Threshold Operation. Sub- and near-threshold operation exploits the supply voltage knob, operating transistors below or close to their threshold voltage, respectively, to reduce power consumption [89]. This enables to reduce dynamic power quadratically and static power linearly, as shown in Equations (1) and (2).

However, these savings come at the cost of slower transistor operation, limiting the application frequency. In embedded IoT applications, energy is usually more important than power consumption, as it directly determines the lifetime of the battery. Thus, the **minimum energy point (MEP)** is identified for each application by adjusting the supply voltage such that the total energy for a

specific workload is minimal. Sub- and near-threshold operation extends the supported supply voltage range, reaching the MEP for a large set of workload scenarios.

Lowering the voltage implies higher sensitivity to process variations, which must be carefully evaluated during the design phase to ensure robustness under all operating conditions. Special layout considerations and compensation techniques (see ABB and DVFS) can reduce the effects.

The 180nm sub-threshold standard cell library developed in [89] demonstrates an extended 0.4–1.0V supply voltage, reducing power by 5× compared to a standard low power library at 1.0V. Their follow-up work presents 1kb sub-threshold SRAM [90] and a 32b microcontroller [91] design, achieving 0.84–3.2nW (3.8×) power scalability for 0.27–0.6V voltage scaling and 7× power scalability for 0.37–1.8V voltage scaling, respectively.

5.2.2 ABB. Adaptive body biasing (ABB) [92] adjusts the bias voltage of the transistor body to control its threshold voltage, influencing its speed and power consumption as discussed in Section 5.2.1. FD-SOI and DDC can fully exploit BB, while the effect in bulk technology largely depends on the design parameters. Adapting the BB to the operating point enables to reduce the adverse effect of sub-threshold operation on timing across process and temperature variations (e.g. worst case corner distance from 21× to 0.2× [93]). This allows to implement a wide range of timing-clean operating points from fast to slow and low power. Note that increasing the speed through a strong forward body bias also increases the leakage.

Publications on ABB report 30× frequency and 20× leakage scaling on a RISC microcontroller core and SRAM [92].

5.2.3 DVFS. **Dynamic voltage, frequency, (and accuracy) scaling (DVF(A)S)** allows to trade-off speed against power consumption through supply voltage scaling. The basic principle was already evaluated in the 1990s, allowing CPUs to lower the frequency and voltage for low intensity tasks, reporting power reductions of 1.05–4× [94], and 9.2× [95].

DVAS [96] extends the principle to dynamic accuracy scaling through adjustable arithmetic bit-widths, allowing for lower supply voltages due to shortened critical paths. Demonstrated on a simulated 40nm 16bit array multiplier, DVAS achieves 11.7× energy reduction for scaling down to 8b precision, noting that the critical path length is reduced by 40%. Pipelined architectures require bypassable pipeline registers to evenly distribute the path length reductions, adding area and energy overhead. On a 16b multiplier, 11.1× energy reduction is reported for 8b operation at 8% energy overhead. Envision [39] also combines accuracy scaling with DVFS on a 28nm process, running at constant throughput while scaling precision from 1·16b operations to 4·4b operations at a quarter of the 16b frequency. Combined with body biasing to reduce leakage at low frequencies, power is reduced by 25×.

5.3 Memory

Data handling is dominating today's accelerator power consumption and area, requiring careful selection of the memory type and the access strategies. Memory access energy is subject to large variations across memory types and sizes, as summarized for a standard 45nm SRAM [79, 97] in Table 2. DRAM is reported to have 200× higher access energy than a small 8kB SRAM (for 64b word width) [79].

Figure 6 shows an overview of the existing memory types, distinguishing storage (e.g. non-volatile hard disk) and memory (e.g. RAM) due to their significant difference in access times, density and power consumption [98]. Systems with long idle/sleep phases might not be dominated by access power but (idle) leakage power. In that case, rare accesses to (power-intensive) non-volatile memories could be cheaper than retaining data over a long period of time, constantly consuming leakage power.

Table 2. Energy Per Memory Access in 45nm SRAM

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

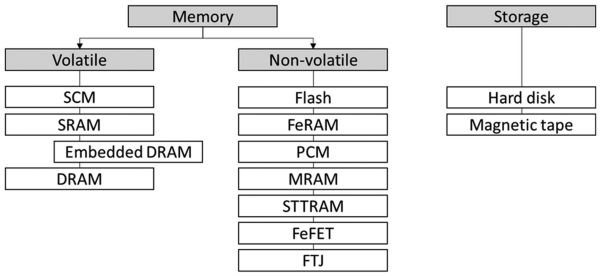

Fig. 6. Overview of memory technologies.

The range of available memory technologies is rapidly increasing, being an active area of research. A recent overview of novel memories used in neuromorphic computing [99] notes that the memory technologies mostly differ in their writing speed, while the reading process is dominated by the sensing circuit interfacing the memory cells. Survey [98] provides an overview of recent non-volatile memories, focusing on PCM, STTRAM, RRAM, and FeFET.

In the following sections we summarize novel memory types, as they are important components of CIM implementations, as well as the dominantly used SRAM and standard cell memories. Quantitative optimization results from the literature are mainly available for the more mature FeFET, SRAM, SCM and DRAM types.

5.3.1 PCM. **Phase change memory (PCM)** [98] is based on the heat-induced reversible phase transition of chalcogenides. It can switch between the low-resistance crystalline phase and the high-resistance amorphous phase, implementing a bipolar switch. The relatively large switching current requires powerful access circuits, dominating the size. Speed is limited by the transition from amorphous to crystalline phase while the reverse process dominates the power consumption. Endurance and speed are estimated around 1G cycles and <100ns, respectively [98]. TSMC presents a 1Mb PCM memory array in 40nm [100], reporting 300uA write current at 100ns write speed, achieving >200k cycles endurance.

5.3.2 STTRAM. **Spin-transfer-torque RAM (STTRAM)** [98] is based on a **magnetic tunnel junction (MTJ)** with two ferromagnetic layers separated by an ultra-thin tunnel oxide layer. It

uses the spin transfer torque of spin-polarized electrons to change the resistance of the memory element, enabling non-volatile states. The access circuit dominates its size due to the relatively high writing currents but is still smaller than SRAM.

STTRAM is demonstrated on 7–8Mb arrays (industrialized 1Gb cluster [101]) in 22–28nm [102, 103], reporting densities up to 10.6Mb/mm^2 and write endurance of >1M–10G cycles.

5.3.3 RRAM. **Resistive RAM (RRAM)** [98] stores information by modulating the resistance of a metal oxide and is therefore often called memristor. A similar approach is **conductive-bridge RAM (CBRAM)**, that forms a conductive metallic bridge in the on-state and interrupts it for the off-state. Endurance of 1M–1G cycles have been reported. However, there are tradeoffs between speed, power, and endurance.

Various RRAM implementations in 14–40nm have been shown [70, 104–107], reporting 0.9–244.8Mb/mm^2 density at supply voltages down to 0.7V for 1Mb–32Gb sizes.

5.3.4 FeFET. **Ferroelectric FET (FeFET)** [98] employs a ferroelectric gate dielectricum that allows to change the resistance of the FET in a non-volatile fashion. This principle is similar to the Flash technology, that uses a floating gate instead. While the power consumption is low due to the low leakage through the gate oxide, the switching speed is high (~20ns). However, its endurance is relatively low, at 10k–100k cycles. Its similarity with standard CMOS transistors makes FeFET compatible with many standard processes.

In [108], a 10Mb FeFET memory array is implemented in a 22nm process, reporting 200k cycles endurance at 2.5–4.5V supply voltage. Their previous work [109] presents a 64×64b array in the same 22nm process and compare it to a 6T SRAM array of the same size, reporting 74× lower static power and >5.3× lower area for FeFET cells (without peripherals). Writing is 10× more energy-intensive and 10× slower while reading costs 1.6× more energy but is 1.5× faster.

A similar, but less mature, type of ferroelectric memory is **ferroelectric tunnel junction (FTJ)**. The tunneling resistance of its ferroelectric layer between two metal electrodes can be adjusted through ferroelectric polarization reversal [98].

5.3.5 SRAM. **Static random-access memory (SRAM)** is the most often used on-chip memory as it can be easily implemented along with digital circuits. It features higher memory density than standard-cell memory as foundries use "layout pushed rules" to optimize SRAM bit-cell area beyond standard layout rules.

Standard SRAM bit-cells use **6 transistors (6T)**, but many larger cell structures have been proposed for power reductions. For low leakage operation a 7-transistor SRAM is presented in [17], reducing area by 18% and 50% compared to standard low leakage 8-and 10-transistor designs, respectively, while achieving similar performance and leakage power.

Power optimizations using non-uniform memory hierarchy in SRAM is presented in a 40nm NN accelerator [110], allowing up to 60% power savings when accessing the smallest instead of the largest level (32× smaller memory). The 67.5kB memory is split into four levels (1.5, 6, 12, and 48kB). SRAM access energy is shown to increase nearly linearly with the memory size above ~100kB [97] as shown in Table 2.

Low leakage SRAM sleep-mode retention is presented on a 55nm process [111], reporting 26× lower retention power for a 16kB memory. Leakage is reduced by optimizing the design rules (increasing area by 2.7×) and the process corner. An additional sleep controller with a charge-pump for the retention voltage allows to power-gate the rest of the chip. Leakage is also reduced in [80], using 180nm low leakage 10T SRAM along with a 65nm 8T SRAM for dense scratchpad memory. The low-leakage memory consumes 4242× less standby power while being 11.3× larger in area.

To optimize the data access for 2D structures like CNNs, a transpose SRAM has been proposed [112], allowing to selectively read a row vector or a column vector of data in parallel, reducing power consumption by 47%.

5.3.6 SCM. **Standard-cell memory (SCM)** is implemented directly in digital logic using flip-flops or latches. This allows exploiting voltage scaling capabilities and avoids dependencies on vendor-specific memory generators.

A dedicated placement strategy for SCM (instead of standard logic **place-and-route (P&R)**) is presented in [113], reporting area and power savings on a 28nm process. Their experiments on 256b–32kb SCM macros show area reductions of >35% compared to standard P&R with access energy reduction of up to 65% for reading and 50% for writing. SCM macro sizes of up to 1kb are shown to be smaller than SRAM, but are already 2–3× larger for 4kb macros (larger area of D-latch cell compared to 6T SRAM bit-cell starts dominating).

A BNN accelerator [114] with a hybrid memory consisting of 456kB SRAM and 8kB SCM demonstrates power savings of SCM compared to SRAM. It reduces the supply voltage to 0.4V when SCM is used, while SRAM requires 0.6V, leading 3× energy savings in 22nm post-layout measurements.

5.3.7 DRAM. **Dynamic RAM (DRAM)** is a volatile high-density memory that stores information as a capacitor charge, which is periodically refreshed. DRAM is usually not compatible with standard logic processes, requiring separate DRAM dies. However, the **hybrid memory cube (HMC)** architecture [115] proposes high density DRAM access to standard logic processes through 3D stacking of DRAM dies on top of a logic die using through-silicon vias. HMC is implemented in TETRIS [116], providing DRAM access to a 45nm processing die. 3D DRAM is shown to consume 3.5–4.2× more energy compared to on-chip 256kB SRAM, but 1.5× less than a planar baseline DRAM at 4.1× higher throughput.

Embedded DRAM (eDRAM) [117] is a CMOS-compatible derivative of DRAM, targeting high density volatile memory. A 4-transistor 8kb eDRAM array is presented in 28nm [117], reporting 17% lower area and 23% lower static power consumption compared to 6T SRAM at equal voltage. The same author also evaluates a 2T design [118], showing slightly lower retention power than low power SRAM cells but 2.5–3.8× lower size for a simulated 28nm 4kb array.

5.3.8 Flash. Flash memory [119] dominates NVM technology on the market, being embedded in most commercial chips where data retention during off-state is required. It can be implemented using standard logic process flows by adding a few additional masking layers, e.g. three extra masks on a 65nm process [120].

6 DATAFLOW AND CONTROL OPTIMIZATIONS

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

6.1 Dataflow and Blocking

NN accelerators have optimized memory allocation and access patterns for efficient computation, splitting a task into smaller blocks that fit on the available resources. The utilized blocking (scheduling) strategy impacts the data access order and thus possible data reuse within the computation blocks. Higher reuse can reduce the number of memory accesses, decreasing the total power consumption [97].

6.1.1 Layer-wise Processing. Traditionally, NNs are processed layer-by-layer, also called layer-wise or layer-first approach. This enables the reuse of layer parameters (e.g. convolution weights), as they are repeatedly used across the layer. However, this also implies that at least one complete layer is always buffered in memory, as explained in more detail in Section 7.1.

6.1.2 Depth-first Processing. The increasing size of feature maps (e.g. higher resolution images) requires large activation buffers to be allocated for layer-wise processing. However, networks can be processed in a "depth-first" streaming fashion instead [121], allowing each layer to buffer only a minimum set of input activations that are needed for computing the next set of output activations. Up to 200× memory bandwidth reduction or alternatively up to 10,000× memory space reduction is reported for this approach. In a follow-up work [122] depth-first processing is implemented on a FPGA and benchmarked on five models reporting throughput increase of 0.91–1.27× and memory bandwidth reduction of 3.9–81×.

A similar work on "fused layers" [123] observes that each output of a convolution layer only depends on a small region of input values. Tracking these dependencies back through multiple layers, results in a pyramid-shaped region. The paper proposes to compute the entire pyramid until the final output while only storing the computed intermediate features instead of buffering each complete. The additional cost for buffering intermediate results locally is traded-off against external memory accesses, achieving up to 95% reduction in off-chip memory traffic for running VGGNet-E.

6.1.3 Loop Ordering and Optimization. Neural network can generally be described using nested loops, with the outer most one looping through the layers of the network. The ordering of these loops influences the possible parallelisms and the required memory size. To process larger networks on limited on-chip memory resources, loop tiling is used: the workload of each layer can be split into overlapping tiles, which are processed sequentially. Parallelization and data reuse can be increased by unrolling parts of the sequential loops and thus parallelizing their computations. Unrolling the entire kernel computation is shown in [124], achieving unprecedented power efficiency at the cost of larger area and limited layer size support.

Various tools have been proposed to optimize loop ordering [97, 125–127], improving energy efficiency and memory size. Commonly implemented microcontrollers provide specialized libraries to make NN processing more efficient. For example, ARM provides the CMSIS-NN library for its Cortex-M microcontrollers [128], supporting CNN, FC, and pooling layers, as well as 8b or 16b fixed point precision. Evaluated on a network running the CIFAR-10 task, a 4.6× improvement in throughput and 4.9× in energy efficiency is reported, compared to a DSP-functions-limited baseline code. A biomedical signal analysis application [129] reports energy savings of 41.6% for enabling 8 core processing instead of single core, noting that the overhead of multi-core execution is fully compensated by efficiency improvements above a certain throughput. Block and memory power gating shows 16.8% energy savings but must be traded off against restart time and related energy and storage implication.

6.2 Early Data Reduction

The high cost of data access during NN inference motivated to reduce the data flow from the sensor, condensing information early and thus minimizing costly data transfers.

One approach is to process the first layer(s) of an NN in the sensor itself. In [130] diffraction gratings above the pixels are used to optically detect Gabor filter-like patterns, as they are often found in the first layer of NNs. Evaluated on the MNIST and CIFAR-10 tasks, sensor communication bandwidth reductions of 10× are reported, with moderate accuracy impacts of −0.1% and

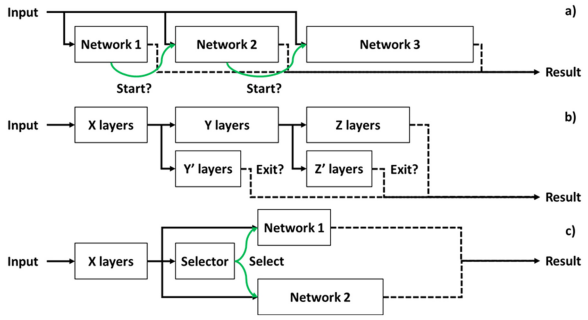

Fig. 7. Selective execution and early exiting approaches: (a) scalable-effort execution, (b) early exiting, and (c) selective execution.

−4.6%, respectively. However, the first layer of the employed LeNet-5 only accounts for 3.8% of all operations, rendering the computation reduction negligible.

Another study implements the first CNN layer in the optical domain using a controllable (grayscale) mask [131]. All filters (output channels) are displayed in the same plane, allowing the image sensor behind to capture all convolution results in parallel, forwarding them to the last layers implemented in the digital domain. Evaluated on MNIST and EMNIST tasks, operation reductions of 250× and 460× are reported while accuracy drops by 0.4% and 1.7%, respectively.

The analog nature of most sensor signals requires **analog-to-digital conversion (ADC)** which can be exploited by implementing matrix multiplication directly in the ADC [132]. An algorithmic reformulation is shown to implement a simple classification task using boosted linear classifiers, embedded in a single matrix transformation. The matrix multiplication is implemented in the feedback path of the SAR ADC, reporting 13× and 29× energy savings compared to SVM-based implementation with similar accuracy for ECG arrhythmia detection and 160×120 pixel gender classification tasks.

RedEye [133] implements an image sensor with analog on-die CNN processing capabilities on a simulated 180nm process. It uses SAR ADCs and tunable capacitors to implement weighted summation to mimic MAC operations, reporting 73% system energy reductions for running the first 1–5 layers of an 8bit GoogleNet on the ImageNet task.

Early data reduction is also implemented in distributed computing [134, 135], splitting the DNN computation across the edge and the cloud to reduce costly data communications, latency and preserve privacy by keeping raw data at the edge.

Furthermore, application-specific early data reduction mechanisms exist: visual attention [10] is shown to reduce the object recognition workload in smart glasses by limiting the analyzed region to the detected eye-gaze direction.

6.3 Selective Execution and Early Abortion

This section presents techniques to dynamically adapt the network complexity to the input, enabling "simple" inputs to be analyzed with a fraction of the network capacity without decreasing the accuracy for complex ones. Average latency and power consumption can thus be reduced if simple inputs dominate the execution. Figure 7 illustrates the techniques, covering (a) hierarchically scalable effort [39, 136], (b) early exiting [137, 138], and (c) selective execution [139].

6.3.1 Hierarchically Scalable Effort. Identifying early exits during training enables 2–6× latency reduction on MNIST and CIFAR-10 tasks [136]. A scalable-effort approach [140] proposes to use a chain of networks with increasing complexity, allowing simple inputs to complete processing

with smaller networks than more complex ones. Evaluated on various classification tasks, they achieve 1.2–9.8× average reduction of operations per benchmark. The Envision NN accelerator [39] demonstrates this on a face recognition task, hierarchically increasing the network complexity, starting with a 12MOP network for presence detection (6.4mW, active 98% of time), followed by a network recognizing the owner, a set of 10 identities, 100 identities, and finally 5,760 different identities (77mW, 0.01% of time).

6.3.2 Early Exiting. Adding special output classifiers after every few layers allows for terminating an NN execution early if classifiers report a high confidence [137]. The trained network is analyzed after each layer to estimate a gain metric, quantifying the ratio between reduced number of operations and increased overhead due to the added classifier. Benchmarked on two 6-and 8-layer networks with one and two early exits, respectively, 1.73× and 1.91× reduction in number of operations are reported at iso accuracy. The same technique is used in the 12-class keyword spotting accelerator [138], reporting 69% of the inputs exiting early, reducing the average power consumption by 22% compared to always executing the complete network.

6.3.3 Selective Execution. Selective execution [139] enables different execution paths that can be selected depending on the input provided: an embedded selector network decides which branch to execute, providing less complex network branches for simpler inputs to reduce the average number of computations.

7 DATA HANDLING OPTIMIZATIONS

The data-intensive nature of NNs challenges the memories and related access energy efficiencies. In many systems more than 50% of the total power consumption is related to memories and data handling [79]. This chapter discusses optimizations for efficient memory utilization and compression, reducing the amount of data to be accessed. Related sections cover the efficient reuse of data across computation elements (Section 8.3) and the reduction of data through computational optimizations (Section 8).

7.1 Efficient Memory Mapping

Efficient memory utilization reduces the required memory space, saving power, chip area and thus IC cost.

The traditionally used buffering scheme for layer-wise NN processing is often called ping-pong buffering [126], following a double buffing approach to allow simultaneous reading of input activations and writing to output activations. It maps the activations of subsequent layers to two disjunctive memory regions, which must therefore allocate at least the maximum sum of any two subsequent layers. By allowing the activation memory regions to overlap during layer-wise processing, memory savings of up to 50% compared to standard ping-pong double buffering can be achieved [141]. The extent of savings depends on the layer dimensions and increases for large layers with small kernel sizes.

7.2 Data Compression

Compression reduces the memory footprint of data content and can be adopted in NN accelerator designs.

Run-length compression (RLC) of zero values is used in Eyeriss [40] to reduce the memory footprint and bandwidth. It encodes the number of zero entries in 5b, followed by the next non-zero value, reporting 1.2–1.9× reduced memory accesses for AlexNet. Eyeriss v2 [43] uses a “compressed sparse column format” for both weights and activations, allowing to skip sparse

operations directly in the compressed form, reducing memory bandwidth and energy. Compared to RLC, it simplifies the addressing of sliding window striding.

Loss-less Huffman coding [142], is shown to reduce weight memory by 20–30%. It employs variable-length codewords, providing smaller bit-widths for more common values, reducing the overall memory. An edge ML Huffman-coding DMA is shown to reduce data bandwidth by up to 5.8× [39].

Weight sharing is used in [142], replacing weights with table indices, referencing a limited number of physically stored values. The upper bound of memory savings is defined by the weight bit-width $N_{width,w}$ and the number of table entries N_{values} as shown in Equation (3). The 45nm accelerator EIE [38] reports 8× energy savings through weight sharing (4b indices referring to 16 16b weight values).

$$N_{wshare,max} = N_{width,w}/\log_2(N_{values}) \tag{3}$$

8 COMPUTATION OPTIMIZATIONS

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

8.1 Optimized Convolution Implementations

The dominance of convolution operations in many network architectures [143, 144] motivated optimized convolution implementations, aiming for similar algorithmic behavior while reducing computational complexity and resources.

8.1.1 Separable Convolutions. Separable convolutions are based on a separable filter approximation, splitting higher dimensional kernels (e.g. a $k_x \cdot k_y$ 2D convolution) into multiple lower-dimensional ones (e.g. 2 1D convolutions of $k_x \cdot 1$ and $1 \cdot k_y$), significantly reducing the number of operations.

A 2D approach is used in [112], replacing a 5×5 convolution layer with a horizontal and a vertical 5×1 and 1×5 layer, reducing the number of operations by 4×. Evaluated on an LFW face recognition NN, the total number of operations is reduced by 1.7x, while accuracy was decreased by 1%. To optimize parallel data access for the vertical direction, a transpose SRAM (T-SRAM) is proposed, enabling both row and column vector readout, reducing power by 47%.

Depth-wise separable convolutions (DSC) separate the kernel only in the depth dimension, convolving inputs in the spatial directions, followed by a pointwise (across depth) convolution that combines the filtered inputs to an output. In contrast to standard convolutions that combine the filtering and output generation, DSCs enable memory savings and reduces MAC operations. MobileNets [145] use such DSCs, reporting computation and parameter size reductions according to Equation (4). Evaluated for the ImageNet task, parameters can be reduced by 7× while the number of MAC operations is reduced by 8.5× at an accuracy drop of just 1%.

$$reduction = \frac{1}{c_{out}} + \frac{1}{k_x \cdot k_y} \tag{4}$$

8.1.2 Frequency Domain Computation of CNN (FDC). Transforming a convolution operation into the Fourier domain results in a point-wise multiplication as shown in Equation (5) [146]. This property can be exploited to reduce the number of operations for computing CNNs. While the forward and backward transformations, using **fast Fourier transformation (FFT)**, increase the

computational effort and memory needs, the operation count can be significantly reduced for large input and kernel sizes, achieving 1.75–5.3× faster computation at iso-accuracy for 3×3 and 11×11 kernels in experiments on various layer sizes [146]. The reported speedup s, in terms of operation counts, is shown in Equation (6) for batch size B. However, this method requires large memories for the FFT and the subsequent matrix multiplication. A recent study [147] builds up on the FFT-based approach, additionally exploiting tiling, result-reuse, and symmetry of real-valued FFTs, roughly cutting the number of operations and Fourier outputs in half. It demonstrates 0.96–1.74× increase in throughput compared to the pure FFT-base approach on 9×9–3×3 kernel sizes.

$$f * g = \mathcal{F}^{-1}\left\{\mathcal{F}\left\{f\right\} \cdot \mathcal{F}\left\{g\right\}\right\} \tag{5}$$

$$s = \frac{\left(B \cdot c_{in} \cdot c_{out} \cdot x_{out}^2\right)}{\left(2 \cdot x_{out}^2 \cdot \log\left(x_{out}\right) \cdot \left(c_{out} \cdot B + c_{in} \cdot B + c_{in} \cdot c_{out}\right) + 4B \cdot c_{in} \cdot c_{out} \cdot x_{out}^2\right)} \tag{6}$$

8.1.3 Winograd Algorithm. For highly parallelized convolution computations, the sliding window operation can be flattened to convert it into a large point-wise matrix multiplication. The overlapping windows create significant redundancy with neighboring data, allowing to combine certain kernel-weights offline, which reduces the number of multiplications at the cost of more additions [144]. This so-called Winograd convolution achieves 1.4–2.26 speedup for computing 3×3 convolutions.

8.1.4 Strassen Algorithm. Large matrix multiplications, as used in NNs with large kernels, can be efficiently computed using the recursive Strassen algorithm, reducing the number of operations [148]. Evaluated on AlexNet, operations are reduced by 47%: while the dominating 3×3 and 5×5 convolutions are reduced in terms of operations, the 11×11 convolution in the first layer suffers from an increase of 18% compared to standard multiplication.

8.2 Sparsity Exploitation and Pruning

Le Cun et al. [149] observed more than three decades ago that a significant number (75%) of NN parameters can be removed without affecting its algorithmic accuracy. A recent exhaustive survey [150] provides explanations to this phenomenon and estimates 10-100× model size reduction for various networks. It focuses on sparsification methods that set parts of a network to zero (pruning), while keeping its complexity constant. They reduce a network's size to its minimum required complexity by creating a (too) high dimensional representation to improve the training, knowing that the network can be reduced again through pruning. Previous works, report weight sparsity ratios of up to 99.996% for a LeNet-5 network with 99.3% accuracy on the MNIST task [151]. We refer the interested reader to the literature for more details on the main reduction techniques: model down-sizing through neural architecture search [152], operator factorization [153], quantization (Section 8.4.3), compression (Section 7.2), parameter sharing [154], and sparsification [150]. A survey on hardware acceleration of compressed models can be found in [155].

To create more sparse models, a three-step approach is proposed [156], first learning the importance of each connection, then dropping low-weight ones and finally fine-tuning the training. Evaluated on AlexNet and VGG16 running the ImageNet task, they report 9× and 13× parameter reduction at iso-accuracy. Energy-aware pruning [157] optimizes the pruning strategy to achieve a minimum energy cost. Observing that layers which are pruned in an early stage, tend to have larger sparsity, they start pruning energy-intensive layers first, estimating their cost based on the number of computations and memory accesses. Evaluated on AlexNet running the ImageNet task, reports 3.7× energy reduction and 11× weight reduction at <1% accuracy drop.

Envision [39] exploits sparsity by skipping sparse memory accesses and MAC operations using a sparsity map (1b entries per value). It reports 1.6× system energy saving for 30–60% activation sparsity. Similarly, NullHop [158] exploits sparsity by skipping sparse computations using a sparsity map combined with non-zero value list compression, achieving up to 3.68× throughput increase in 28nm synthesis results.

Zero-value skipping logic using a zero-free neuron array format is evaluated on a 65nm accelerator [159], reporting 1.37× average throughput increase for various networks at the cost of 25% memory increase (for zero sparsity).

A special form of temporal sparsity is proposed in CBinfer [160] and discussed in Section 8.3.

8.3 Data Reuse

Memory data that is used multiple times within a short period of time can be buffered in a local cache memory to allow cheaper and faster access. Special focus is set on data that has been fetched from energy expensive external memory. Three main structures of such data reuse have been studied extensively [27] and are summarized below, namely **row-stationary (RS), weight stationary (WS)** and **output stationary (OS)** approaches. While a WS setup minimizes movements of weights, as it is used in CIM architectures, OS keeps the partial sums for computing each output feature local, and RS approaches combine weight and activation data reuse.

Eyeriss [40, 57] implements an RS approach to maximize on-chip data-reuse and reduce costly external memory accesses. The 65nm chip can buffer one activations row (up to 224 16b values) and one weight row (up to 12 16b values), increasing energy efficiency by 1.4–2.5×.

A weight stationary approach is presented in [161], reducing the weight memory accesses by moving activations along cached weights during convolution computations.

OS processing is used in ShiDianNao [37], computing one output per processing element in its 8×8 array (64 outputs of the same output channel). This avoids moving partial sums to the memory and allows sharing inputs with neighboring PEs, reducing the memory bandwidth by up to 10×. Compared to their prior work DianNao [162], featuring no local data reuse, it allows to reduce power consumption by more than 1.66x, while increasing throughput by 1.87×. CORAL [163] employs coarse-grained reconfigurable MAC arrays that allow programming different OS data reuse options. Another output stationary approach is used in Hyperdrive [164], keeping the feature maps stored entirely on-chip and streaming in weights. This is motivated by the use of binary weights, consuming 16× less memory bandwidth than the 16b float activations.

Eyeriss v2 [43] implements a flexible **network-on-chip (NoC)** that can be reconfigured into four main reuse modes, enabling high activation and weight reuse in convolution layers while the dense layer mode can maximize activations broadcast because weights cannot be reused. Evaluated on MobileNet, they report a throughput increase of 5.6×.

Temporal data reuse is proposed in CBinfer [160], demonstrating that CNN-based CV applications with a static field of view can reuse large portions of each CNN layer and only compute those features that changed over time. They first detect changes in the input, generating a temporal sparsity map to update the connected output features for which the inputs exceed a calibrated threshold and buffer the new feature map. Evaluated on a 5-layer CNN for 10-class scene segmentation, this achieves an average speed-up of 8.6×.

8.4 Hardware/Software Co-optimization

A recent publication proposes less artificial intelligence [165], suggesting that today's networks are too high dimensional and thus prone to overfitting limited datasets, providing some intuition why high sparsity and quantization are viable optimization strategies. Today's NN algorithms exploit

this observation, being developed with the challenges of resource-constrained edge ML hardware in mind. Thus, optimized algorithms like MobileNets [145] or SqueezeNets [166], reducing complexity through separable convolutions and kernel size minimization, have been introduced. These co-optimization strategies are covered in this section.

8.4.1 Complexity Scaling. The growing number of network architectures created a large design space, shifting the design strategies from hand-crafted architectures to (semi) automatic **network architecture search (NAS)**, identifying optimal trade-offs between design constraints like accuracy and computational complexity. Frameworks like adaDeep [167] provide multi-dimensional model selection strategies to find the optimal model scaling strategies for a specific model and use-case. Dynamic complexity scaling is proposed by once-for-all networks [168], which are trained once but can then be deployed in various down-scaled complexities (in depth, width, kernel size, and resolution) without re-training. This allows deploying them on platforms ranging from high-performance cloud servers to low-power edge devices, with low accuracy degradations for the reduced-size versions.

8.4.2 Energy-quality-scaling. **Energy-quality (EQ)** scalable systems [169] introduce a quality metric that describes a network's complexity. This allows to identify the knobs for power-scaling through (acceptable) quality degradations. The presented framework for EQ-scalable systems and EQ architectures [169] helps identifying applications where sensing and/or processing quality degradation is acceptable, for example in noise-resilient applications like computer vision. The list of quality knobs for dynamically adjusting EQ scaling encompasses arithmetic precision, bit error rates, sampling rates, algorithmic complexity and more. Follow-up works exploit this concept for implementing ULP voice activity detection [170] or always-on computer vision system [171]. Voice activity detection is shown to support EQ scaling [170], achieving 3.5× lower energy for 2% accuracy degradation using decision trees on a 28nm chip. Joint voltage and EQ scaling applied to a traditional computer vision task [171] is shown to achieve 3× lower energy through EQ scaling and 3.4× lower energy through VDD scaling on a 40nm process.

8.4.3 Quantization and Reduced Precision. NNs can tolerate significant parameter and activation quantization with negligible effects on accuracy [47]. To reduce the effect of reduced precision, quantization-aware training techniques are used (e.g. straight-through estimators [172]) to keep the model derivable for back-propagation. Quantization works especially well in networks which are limited in training data, while the accuracy degradation increases for smaller (complexity-limited) networks, which cannot compensate for the loss of information [173]. Lowering precision reduces memory size, simplifies computational arithmetic, and lowers the power consumption, which also motivated Google to add 8bit support in their TPUs [56]. The viability of fully **binary NNs (BNNs)** [174], limiting activation and weight precision to 1bit, finally demonstrates the full range of quantization possibilities. BNNs have considerable accuracy degradations but allow to process multiplications using simple XNOR logic, reducing area and power needs.

Survey [48] provides an in-depth analysis of quantization schemes with a special section on sub-8bit quantization and a short overview of quantization-optimized hardware. Weight sharing is another form of quantization, limiting the number of supported values, as discussed in Section 7.2.

Energy and area savings of reduced precision arithmetic have motivated quantization optimizations. The energy for additions and multiplications are evaluated on a 45nm process [79], reporting 14× and 20× MAC energy reduction for 8b int compared to 32b int and 32b float, respectively, as summarized in Table 3. A 45nm overview [175] reports power and area increase with bit-width for adders (linear increase) and multipliers (quadratic increase), as shown in Table 4. Comparing a MAC-combination, total area and power are reduced by 13× and 10.8x, respectively, for 32b to

Table 3. 45nm Energy per Operation for Different Precisions

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

Table 4. 45nm Adders/Multipliers for Different Precisions

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

Table 5. 65nm Multipliers for Different Precisions

Precision		Int16	Float32
Multiplier	Area [um^2]	1309	7998
	Power [uW]	577	4230

8b, and by 3.6× and 3.6x, respectively, for 16b to 8b. Similarly, this was shown on multipliers for a 65nm process [162] as illustrated in Table 5.

The 65nm accelerator in [162], achieves 6.1× reduction in area and 7.33× in power consumption for implementing 16b fixed point multipliers instead of baseline 32b floating point arithmetic while maintaining comparable accuracy. Envision [39] exploits 1–16b dynamic precision scaling combined with DVFS, reporting reductions in energy per MAC operation (relative to 16b) of >5× for 8b and >50× for 4b precision using sub-word parallel computations in a 28nm process.

UNPU [176] employs a non-linear quantization support that replaces a 16b multiplier by a 2×4-bit lookup table, indexing 16 16b activations and 16 16b weights, with the result of each combination stored in the table. They report 79% power reduction and 93% lower latency while the area is reduced by 1.3× compared to instantiating a 16b multiplier.

A review on scalable precision MAC architectures [177] compares recent 2-8b scalable implementations, discussing spatial and temporal MAC architectures and benchmarking them on a 28nm process using a data-gated 8b-input MAC as baseline. Throughput is roughly increased quadratically for cutting precision by a factor of 2, reaching up to 14.5× for 2b precision. Area is increased 1.1-4.4× for parallel precision-scalable designs while bit-serial implementations can reduce area by up to 40%. The overhead for scalability-support increases the energy per operation in full precision modes by up to 52% for single level, up to 94% for dual level scalability, and up to 14× for multi-cycle bit-serial MACs. The energy per operation only reduces linearly with precision in the baseline but decreases supra-linearly for the scalable MACs, achieving up to 4× lower energy at 2b precision (overhead compensated at 6-4b precision for parallel and at 2b for bit-serial MACs).

An XNOR-based 22nm BNN accelerator [114] exploits the reduced precision by utilizing SCM instead of SRAM, enabling lower power consumption. Similarly, the 65nm binary-weight (12b activation) CNN accelerator YodaNN [41], reports improved performance using voltage-scalable SCM. Combining binary weights (instead of 12b) with SCM (replacing SRAM) allows them to reduce the power by 11.6×.

Cross-layer bit-width optimization [178], shows more than 20% parameter size reduction compared to homogeneous bit-width fixed-point quantization at iso-accuracy on the CIFAR-10 task. Furthermore, knowledge distillation can be used for low-precision quantization [179], improving the accuracy of a highly quantized model using "distilled" knowledge from a larger (higher precision) teacher network during training.

8.5 Approximate Computing

Approximate computing trades power consumption, speed, and area off against arithmetic accuracy [180]. Approximation approaches are either based on **voltage over-scaling (VOS)** below the technology's threshold voltage or on functional modifications ranging from algorithm- to circuit-level [181]. It differs from energy-quality-scaling (Section 8.4.2, due to its circuit-based scaling approach.

A 2020 survey [182] on approximate computing for DNNs reports power, delay and area numbers from synthesized approximate adders, multipliers and dividers using a 28nm process at 1V. It reports up to 69% energy savings (power-delay product) for an image sharpening task while for JPEC compression only 20% savings are achieved.

An earlier survey [181] reports an approximate integer data format and related arithmetic operation implementations in 45nm [175], limiting values and computation precisions to a dynamically selected range of most significant non-zero bits. This achieves 55-65% power reduction compared to accurate computations at <0.5% accuracy drop in KNN and SVM tasks. Approximate computing using 2- and 3-bit adder designs [183] reports further power and accuracy improvements.

VOS introduces bit errors due to missed timing constraints and other unwanted effects but reduces power consumption as shown on a 28nm CNN accelerator [184]. Reducing the SRAM voltage from nominal 1.0V to 0.51V enables 3.12× memory and 2.13× system power reduction for running a 9-layer fully binary CNN at <1% accuracy drop. They report stronger effects on accuracy from weight errors than activation errors, enabling further activation memory voltage scaling.

8.6 Non-Conventional Arithmetic

To further optimize NN computations, non-conventional computer arithmetic has been surveyed [185], comparing currently used CMOS technology and alternative emerging technologies for implementing computer arithmetic. Also alternative number systems are evaluated, for example a logarithmic system on a 65nm process [186], showing 3× higher energy per addition compared to floating point, but 1.5× lower for multiplications, 17× lower for divisions and 38× lower for square root computations.

8.6.1 Spiking Arithmetic. Biological neurons in the human brain function with spike-based signaling and computing, inspiring researchers to rethink the traditional level-based arithmetic in ICs [187]. Neuromorphic spiking arithmetic is employed in IBM's 28nm TrueNorth [188], implementing a total of one million digital spiking neurons, and Intel Loihi [189], implementing 131k neurons in a 14nm process. Due to the significantly different computing paradigm, which cannot be directly compared with other optimization approaches, we refer the reader to the specific literature for more details [187, 190, 191].

8.6.2 Hyperdimensional Computing. Hyperdimensional computing is another brain-inspired computing approach that encodes information in very high-dimensional binary vectors with thousands of entries, called hypervectors. The similarity of information contained in hypervectors is encoded in a distance metric, making them robust against bit errors and thus suitable for VOS. Tasks like image recognition are performed by comparing the distance of an input vector (e.g. features of an image) with a known reference vector. We refer to the specific literature [185, 192] for more details as this goes beyond the++ scope of this work.

8.7 Mixed Precision Arithmetic

The analog nature of most sensor signals and power-advantages of computing in the analog domain motivated mixed signal arithmetic for computing DNN [193, 194].

Survey [34] compares a set of analog DNN accelerator architectures, reporting 40-80% lower area as well as 70% and 40% reduced power compared to digital implementations for 130nm and 65nm designs, respectively. This shows that analog circuits do not scale equally well with reduced process nodes as their digital counterparts. Other properties, like the intrinsic computation parallelism from Kirchhoff's law can compensate for this reduce advantage in smaller node sizes.

Analog implementations usually require peripheral circuits that can diminish analog computation advantages at increasing design efforts, as illustrated in [195, 196], implementing the same accelerator in a 28nm process but using analog or digital MAC-accumulation circuits, respectively. Evaluated on a fully binary CIFAR-10 network at iso-accuracy, the energy per inference dropped from 14.4uJ (digital) down to 3.79uJ (analog), which is a system-level energy improvement of 3.8x, while the energy for the underlying MAC computation dropped by nearly 12.9× (>3× more). The 28nm analog-domain computations (8b dot product) are presented in [197], reporting nearly 75% energy spent on ADC and control logic. Bong et al. [198] implement a hierarchical analog-digital hybrid binary decision tree engine on a 65nm image sensor, running 60% of the algorithm in the analog domain, reducing the inference energy by 39% compared to digital computation. A 130nm 32×32 analog MAC array multiplying a DAC-converted vector with a 32×32 matrix is presented in [199]. Compared to a multi-core processor baseline, power and area are reduced by 71% and 43%, increasing throughput by 10.3×. The CIM approaches discussed in Section 4.3 exploit the advantages of mixed signal processing, keeping the data in memory to avoid losses data movement and digitalization losses. An RRAM-based analog crossbar [200], compares performance to equivalent implementations with digital RRAM-usage and SRAM-based memory. It reports 270× energy and 540× latency improvements over digital RRAM, and 430× energy and 34× latency improvements over SRAM implementations, showing latency issues for digital RRAM.

8.8 Arithmetic Implementations

NN training is usually executed with 32bit floating point precision, that can be significantly lowered during inference. Each layer has a specific sensitivity to quantization and can thus be implemented with adapted (minimal) bit-widths [201]. Selecting the arithmetic precision and the data types allows to optimize implementations, increasing throughput and energy efficiency as shown in Section 8.4.3.

Motivated by the finding that the required precision varies across DNN layers [201], a 65nm bit-serial DNN engine is implemented based on the 16bit DaDianNao architecture [202]. Evaluated on nine common DNNs with per-layer-minimized precision, it reports 1.2×–4.76× (2.0× on average) increased energy efficiency at iso-accuracy compared to the baseline accelerator.

Variable-precision bit-serial MAC can also be implemented using look-up tables [42], reporting energy savings of 23%, 27%, 41% and 54% with respect to standard fixed-point MAC for 16-, 8-, 4-, and 1-bit weight precision, respectively (16b activation). A similar, Booth-Wallace multiplier-based multi-precision implementation was shown in [39], supporting 16b multiplications, that can be split into 4×4b multiplications.

9 QUANTITATIVE COMPARISON OF OPTIMIZATIONS

This chapter summarizes the quantitative effects of the discussed edge ML accelerator optimizations. Table 6 lists each optimization approach with a brief description of the implementation setup that was used to demonstrate the effect in the referenced publication. Five performance indicators quantify each technique: the memory usage impacts the (often dominating) energy for

Table 6. Overview of Optimization Strategies and Reported Advantages

Field (ch.)	Optimization approach	Reported impact					Reference work		Remarks
		Mem. reduction	Throughp. increase	Area reduction	Power/E. reduction	Algo. accuracy	Work (Year)	Implementation performance	
Architecture (4)	Replace sys. array PEs with tensor PEs	–	–	2.1×	1.4× system	–	[58] (2020)	16nm accelerator running various nets (e.g. ResNet-50, MobileNetV1)	
	Replace MAC with CIM-MAC	–	improved	unknown	13× system	declined	[64] (2017)	128×128 CIM array in 130nm running MNIST task (1b w., 5b act.)	Tiny nets only
	Replace MAC with SRAM CIM-MAC	–	2.4×	unknown	4.9× system	declined	[65] (2018)	LeNet5 network running MNIST task on simulated 65nm accelerator	
	Add 8b CIM to pure SRAM	–	improved	0.78× (28% lar.)	unknown	–	[66] (2021)	384kB SRAM with 8b CIM in 28nm process	
Power management (5.2)	Sub-threshold operation	–	0.23× (4.4× low.)	0.53× (1.87× lar.)	5× power	–	[89] (2013)	180nm sub-threshold standard cell compared to standard 180nm library	
	Sub-threshold operation	–	declined	–	7× power	–	[91] (2016)	180nm MCU: 13kHz @ 0.48V vs. 25MHz @ 1.8V	
	Sub-threshold operation	–	declined	–	3.8× power	–	[90] (2015)	180nm 1kb SRAM @ March test: 530Hz @ 0.27V vs. 200kHz @ 0.6V	
	ABB	–	−30× low.	–	≤ 20× leakage	–	[92] (2019)	55nm RISC MCU and SRAM	
	DVFS	–	–	–	1.05–4×	–	[94] (1994)	CPU @ varying intensity using DVFS	
	DVFS	–	–	–	9.2×	–	[95] (1996)	CPU @ 1.5V and 1/10 frequency DVFS vs. CPU @ 3V and 90% idle	
	DVFAS	–	–	–	25×	declined	[39] (2017)	28nm acc.: $4{\cdot}4b$ with DVFAS vs. $1 \cdot 16b$ @ constant throughput	0.65–1.1V, (BB<1.2V)
	DVAS	–	improved	0.85–0.9× (lar.)	11.1/0.9× (8b/16b)	declined	[96] (2015)	40nm 16b Baugh-Wooley multiplier with DVAS vs. gating unused bits	Overhead: add. logic
Memory (5.3)	Reduce mem size for lower access energy	improved	–	improved	Lin. with size	–	[97] (2016)	45nm SRAM implementations: 1-1024kB	Non-lin. if small size
	Replace 6T SRAM with FeFET	–	R/W: 1.5x/ 0.1×	>5.3×	R/W: 0.6/0.1×	–	[109] (2019)	22nm 64×64b array: FeFET vs. 6T SRAM (FeFET read/write: @4V/1V)	Stat. pow. 74× lower
	Reduce SRAM area: low leak. 7T vs. 8/10T	–	unknown	18% 50%	unknown	–	[17] (2019)	8kB SRAM in 180nm using 7T vs. 8T and 10T low leakage SRAM	
	Non-uniform SRAM instead of uniform	–	–	0.98 (2% larger)	60% (1.5 vs. 48kB)	–	[110] (2017)	4-level SRAM memory (1.5-48kB) in 40nm at 0.65V and 3.9MHz	
	ULP SRAM sleep mode	–	declined	0.37× (2.7× larger)	26×	–	[111] (2020)	16kB SRAM with charge pump for low power retention in 55nm	Slow proc. corner use
	Reduce SRAM leakage 10T vs. 8T	–	unknown	0.09× (11.3× lar.)	4242× retention	–	[80] (2013)	Low leakage 180nm 10T SRAM vs. 65nm 8T SRAM	
	Transpose SRAM vs. row-only SRAM	–	–	declined	1.9×	.	[112] (2018)	65nm accelerator with T-SRAM vs. SRAM for 5×1 & 1×5 sep. conv.	Overhead: add. logic

(Continued)

Table 6. Continued

Field (ch.)	Optimization approach	Reported impact					Reference work		Remarks
		Mem. reduction	Throughp. increase	Area reduction	Power/E. reduction	Algo. accuracy	Work (Year)	Implementation performance	
Memory (5.3)	Replace SRAM with SCM	–	unknown	2–3×(<1kb) <0.3× else	>2×	–	[113] (2016)	8kb SCM vs. SRAM in 28nm using dedicated SCM placement	
	Optimized placement strategy for SCM	–	unknown	>35%	R:≤65%, W:≤50%	–	[113] (2016)	256b–32kb SCM macros in 28nm: special placement vs. standard	R/W:Read/ Write
	Replace SRAM with SCM	–	–	unknown	2-3×	–	[114] (2018)	BNN accelerator in 22nm using SRAM@0.6V or SCM@0.4V	
	3D HMC DRAM instead of 2D DRAM	–	4.1×	improved	1.5×	–	[116] (2017)	HMC DRAM dies on 45nm logic die running various networks	
	Replace SRAM with eDRAM (4T)	–	–	17%	23%	–	[117] (2018)	28nm 8kb eDRAM array vs. SRAM (both@0.7V and room temperature)	
	Replace ULP SRAM with eDRAM (2T)	–	unknown	2.5–3.8×	unknown	–	[118] (2019)	28nm 4kb eDRAM memory array at 0.4Vvs. SRAM	Simulation only
Dataflow & Control (6)	Depth-first instead of layer-wise processing	Size/BW:10^4/200×		improved	–	–	[121] (2019)	Mem. reduct.: e.g. SRGAN (UHD): 19'633×[1], MobileNetV2: >20×[1]	Limits #layers
	Depth-first instead of layer-wise processing	chip:0.8× Ext.: 20×	(0.94×)	–	improved	–	[123] (2016)	VGGNet-E on FPGA (AlexNet: only 28% ext. reduc.)	Less ext. memory
	Optical convolution implementing 1st layer	2.5–10× E/MNIST	250×–460× E/MNIST [2]	–	improved	1.7/0.4% E/MNIST	[131] (2020)	CNN layer in front of 4 dense layers on MNIST and EMNIST task	Only on first layer
	In-sensor processing of first CNN layers	–	–	–	73/45% sys/CNN	declined	[133] (2016)	Sim. 180nm SAR ADCs and tunable cap. running 8b GoogleNet	first 1–5 layers
	Add hierarchically scalable effort	declined	improved	–	~10×	–	[39] (2017)	28nm DNN accelerator running 12MOP–30.8GOP networks	Overhead: more nets
	Add hierarchically scalable effort	declined	1.2–9.8× (avg.)	–	improved	–	[140] (2015)	Hier. scaled SVM, NN, and dec. tree on MNIST and other tasks	Overhead for diff. inp
	Introduce early exit for conditional ex.	declined	1.73–1.91×	–	improved	improved	[137] (2016)	6- and 8-layer NN for MNIST task with 1 and 2 early exits	Overhead: add. lay.
	Introduce early exit for conditional ex.	declined	–	–	1.22×	declined	[138] (2020)	12-class Google speech command task with early exit on 22nm acc.	Overhead: add. lay.
Data handling (7)	Replace ping-pong buffering	Up to 2×	–	improved	–	–	[141] (2020)	Act. (total) memory reduction e.g. DMCNN-VD: 48.8% (48.2%)	
	Compression using Huffman coding	1.2-1-3× (weights)	–	improved	–	–	[142] (2016)	Encode weights using Huffman coding in AlexNet and VGG16	
	Compression using Huffman coding	Up to 5.8×	–	improved	–	–	[39] (2017)	28nm DNN accelerator running face recognition CNNs	

(Continued)

Table 6. Continued

Field (ch.)	Optimization approach	Reported impact					Reference work		Remarks
		Mem. reduction	Throughp. increase	Area reduction	Power/E. reduction	Algo. accuracy	Work (Year)	Implementation performance	
Data handling (7)	Compression using RLC coding	1.2–1.9×	–	improved	–	–	[40] (2016)	65nm process running AlexNet	
	Weight sharing	<4× for weights	–	–	8×	declined	[38] (2016)	45nm accelerator with weight sharing (16 entries of 16b values)	
Computation (8.1)	Replace CNN with depth-wise sep. conv.	>7×	8.5×	improved	–	−1.1%	[145] (2017)	Depth-wise sep. MobileNet vs. pure CNN on ImageNet task	
	Sep. conv. instead of 5×5: 1×5 & 5×1	improved	4× (1.7× tot. net.)	–	–	−1%	[112] (2018)	5-layer CNN for LFW face recog. with sep. conv. vs. normal conv.	
	FFT-based convolution	declined	1.75-5.3×	declined	improved	–	[146] (2014)	Computing 3×3 – 11×11 CNN kernels via FFT vs. conventional	Small ker. profit less
	Opt. FFT-based conv. using fine-grain FFT	improved	0.93–1.74×	–	improved	–	[147] (2020)	Fine-grained FFT-based conv. vs. pure FFT conv. (9×9–3×3 kernels)	
	Use Winograd fast convolution	–	1.48–2.26×	–	–	declined	[144] (2016)	Winograd 3×3 convolution on VGG E network (batch size 64-1)	Errors for large kern.
	Use Strassen algo. for CNN matrix mult.	–	24-47%	–	–	–	[148] (2014)	Strassen algo. on AlexNet conv. layers 2–5 (5×5, 3×3) on CPU	More eff. If large mat.
Computation (8.2)	Sparsity exploitation + Pruning approach	Weights: 9×–13×	improved	–	–	–	[156] (2015)	Network pruning on AlexNet and VGG16 for ImageNet	
	Sparsity expl.: skip mem. acc. & MAC op.	–	improved	–	1.6×	–	[39] (2017)	28nm DNN accelerator with 1b sparsitiy map (16b activations)	30°C60% sparsity
	Sparsity expl. with energy-aware pruning	11× weights	unknown	improved	3.7×	0–(−1)%	[157] (2017)	AlexNet on ImageNet task	
	Skipping sparse operations	improved	3.68×	–	–	declined	[158] (2019)	28nm CNN acc. with sparsity map and NZVL compression	Simulation only
	Add zero-value skipping logic	improved	1.37×	unknown	improved	–	[159] (2016)	65nm acc. with zero-skipping logic and zero-free data encoding	Overhead: add. logic
Computation (8.3)	Add data reuse: row-stationary processing	declined	improved	declined	1–4–2.5×	–	[40] (2016)	65nm 168 PE row-stationary accelerator running AlexNet	SRAM and ext. DRAM
	Add data reuse: output stationary	declined	1.87×	0.3 (5.5× larger)	1.66×	–	[37] (2015)	65nm 64PE OS acc, with vs. without reuse on various CNNs	Network-dependent
	Add data reuse: NoC for flex. reuse modes	declined	5.6×	declined	1.8×	–	[43] (2019)	65nm NN accelerator with 192 PEs running MobileNet	
	Data reuse: change-based temp. sparsity	declined	8.6×	declined	–	–	[160] (2017)	On 5-layer CNN for 10-class scene segmentation	Entire net in memory

(Continued)

Table 6. Continued

Field (ch.)	Optimization approach	Reported impact					Reference work		Remarks
		Mem. reduction	Throughp. increase	Area reduction	Power/E. reduction	Algo. accuracy	Work (Year)	Implementation performance	
Computation (8.4)	Reduce precision int32/float32 to int8	improved	unknown	Unknown	14x/20× (int/float)	declined	[79] (2014)	45nm arithmetic compared	
	Reduce precision int32/int16 to int8	improved	unknown	13x/3.6× (32/16b)	10.8/3.6× (32/16b)	declined	[175] (2017)	45nm arithmetic compared	
	Reduced precision(16b → 8/4b)	Up to 4×	Up to 4×	–	>5x/50× (8b/4b)	declined	[39] (2017)	28nm acc. with sub-word parallel 1-16b MAC precision	
	Replace 16b mult. with 4bx4b LUT	–	14.3×	1.3×	4.7×	declined	[176] (2018)	65nm acc running ImageNet task (quantized vs. 32b baseline)	LUT: 256 16b result
	Replace MAC with scal. prec. (8b→8-2b)	–	Up to 14.5× (2b)	>0.2/<1.4× (para./ser.)	up to 4× (2b)	declined	[177] (2019)	28nm scalable prec. MAC designs (parallel and serial) vs. 8b MAC	<14× Pow. overh@8b
	Reduce w. prec- 12b→1b, SRAM→SCM	Weights: 12×	unknown	1.2×	11.6×	declined	[41] (2018)	Binary weight accelerator in 65nm with SCM	SCM repl. SRAM
	Energy-quality scaling	–	–	–	3.5×	−2%	[170] (2020)	Voice activity detection using decision trees on 28nm chip	
	Energy-quality scal.: acc., feat./mem. size	–	–	–	3×	declined	[171] (2020)	40nm image feature detection with controllable det. thresh. (accuracy)	
Comput. (8.5)	Approx. computing	–	–	improved	20–69%	declined	[182] (2020)	28nm approx. mult./adders run. image sharpening, JPEG compr.	
	Approx. computing	–	–	improved	55/65% knn/SVM	declined	[175] (2017)	45nm approx. comp. engine vs. accurate 32bit implementation	
	Approx. computing: SRAM VOS	–	0.095× (10.5× sl.)	–	3.12×	0–(−1)%	[184] (2018)	28nm fully binary CNN acc. running 9-layer CNN	
Computation (8.7)	Replace dig. with mixed-sig. arithmetic	–	1.58×	0.31 (3.2× larger)	3.8/12.9xsys/MAC		[195] (2018)	28nm fully binary CNN acc. with analog MAC vs. digital	Ana. peri. limit gains
	Replace float arith. with logarithmic	–	unknown	unknown	0.33/1.5× add,mul.		[186] (2016)	65nm logarithmic arithmetic system vs. conventional float	
	Replace dig. with mixed-signal arith.	–	–	–	1.39×	declined	[198] (2017)	65nm mixed analog/digital comp. vs. pure digital face detection	60% algo. in analog
	Replace dig. with mixed-signal arith.	–	10.3×	1.7×	3.4× power	–	[199] (2011)	130nm 32×32 analog MAC array for matrix-vector multiplication	In/output in digital
	Replace dig. MAC with ana. RRAM- CIM	–	34×	11×	430×	declined	[200] (2018)	Simulated accelerator design in 16nm, running MNIST task	Dig. acc.: SRAM

(Continued)

Table 6. Continued

Field (ch.)	Optimization approach	Reported impact					Reference work		Remarks
		Mem. reduction	Throughp. increase	Area reduction	Power/E. reduction	Algo. accuracy	Work (Year)	Implementation performance	
Comput. (8.8)	Replace dig. with mixed-sign. arithmetic	improved	1.3–5.3	0.95×	1.2×–4.8× (2.0× av.)	–	[201] (2017)	65nm var. perc. bit-serial acc. vs. 16b baseline (on 9 common DNNs)	
	Replace MAC with var. prec. LUT MAC	–	–	–	16-1b w.: 1.2-1.54×	–	[42] (2019)	65nm NN acc. with var. prec. bit-serial LUT MAC vs. fixed 16b MAC	Activation: 16b

data handling, the throughput determines the processing latency, the chip area directly impacts manufacturing cost, and the power/energy reductions translate into longer battery lifetimes. However, optimizations might influence the algorithmic accuracy, which is therefore listed in the fifth impact column. Note that optimization effects might significantly differ across varying implementation setups (e.g. other network models utilized) and must thus be carefully evaluated for other implementation options. Where none or only qualitative information was available, we noted "unknown", or "improved"/"declined" performance, respectively.

Architectural optimizations (Section 4) report up to 13× power savings with increased throughput using CIM. However, most CIM implementations only support small networks. Power management (Section 5.2) allows for significant leakage and dynamic power reductions, but negatively impacts the throughput due to the reduced operating frequency. Optimizing the memory offers reduced access energy and mainly lower static power, while affecting the required area. Optimized placement and low-leakage SRAM types allow trading area off against power. Various dataflow and data handling options (Sections 6–7) offer improved throughput or reduced memory requirements (up to 10,000× lower size). Computation improvements (Section 8) higher throughput and efficiency using quantization but must be traded off against accuracy deteriorations.

As an example, the designer of a new edge ML accelerator, requiring to run low-complexity ML tasks at a very low power consumption, could identify DVFAS [39] in Table 6 as useful optimization, reporting 25× power reduction at constant throughput. The reported accuracy impact must be considered but might be acceptable for simple tasks. To further improve the power consumption, SCM can be chosen instead of standard SRAM, adding another 2–3× reduction in (memory) power consumption [114]. If the selected network tolerates sparsity, the support of weight and activation sparsity from Table 6 might be another option to drastically decrease the power further. However, replacing the digital processing with a mixed-signal implementation promises lower power gains and probably comes at the cost of increased area, as reported in the literature [195].

10 CONCLUSION

We presented a survey of design optimization strategies for low power neural network accelerators. The compiled list of optimization approaches provides quantitative performance impact measures for each approach, allowing accelerator designer to estimate their impact in future designs. Covered optimization approaches encompass a wide range of components in ML accelerators. They range from architectural and technological options to dataflow, data handling and computation optimizations. Each approach was evaluated based on five performance metrics, namely energy/power reduction, area (cost) reduction, memory size savings, computational throughput increase, and impact on algorithmic accuracy. The reported optimizations provide up to 10,000× memory savings and up to 33× energy reductions.

REFERENCES

[1] Z. Zhao, P. Zheng, S. Xu, and X. Wu. 2019. Object detection with deep learning: A review. *IEEE Trans. Neural Networks and Learning Systems* 30 (2019), 3212–3232.

[2] Y. Zhang, N. Suda, L. Lai, and V. Chandra. 2017. Hello Edge: Keyword spotting on microcontrollers. *CoRR,* vol. abs/1711.07128, 2017.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *International Conference on Neural Information Processing Systems - Volume 1*, USA, 2012.

[4] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *IEEE Computer Vision and Pattern Recognition*.

[5] G. Huang, Z. Liu, and K. Q. Weinberger. 2016. Densely connected convolutional networks. *CoRR,* vol. abs/1608.06993, 2016.

[6] J. Koomey, S. Berard, M. Sanchez, and H. Wong. 2011. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing* 33 (2011), 46–54.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems* 29 (2013), 1645–1660.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things J.* 3 (2016), 637–646.

[9] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE* 107 (2019), 1738–1762.

[10] I. Hong et al. 2016. A 2.71 nJ/Pixel Gaze-Activated object recognition system for low-power mobile smart glasses. *IEEE Journal of Solid-State Circuits* 51 (2016), 45–55.

[11] R. LiKamWa, Z. Wang, A. Carroll, F. X. Lin, and L. Zhong. 2014. Draining our glass: An energy and heat characterization of Google glass. In *5th Asia-Pacific Workshop on Systems*, New York, NY, USA, 2014.

[12] S. Park, S. Choi, J. Lee, M. Kim, J. Park, and H. Yoo. 2016. 14.1 A 126.1mW real-time natural UI/UX processor with embedded deep-learning core for low-power smart glasses. In *Int. Solid-State Circuits Conference*, 2016.

[13] Google, Google Clips specifications, 2017. [Online]. Available: https://support.google.com/googleclips/answer/7545447?hl=en. [Accessed 21 05 2019].

[14] Xiaomi, Xiaomo AI door bell overview, 2019. [Online]. Available: https://www.xiaomitoday.com/2019/10/29/xiaomi-xiaomo-mijia-ai-face-identifcation-1080p-door-bell/. [Accessed 09 02 2021].

[15] Orcam, Orcam MyEye 2 specifications, 2020. [Online]. Available: https://www.orcam.com/en/myeye2/specification [Accessed 09 02 2021].

[16] S. Alyamkin et al. 2019. Low-power computer vision: Status, challenges, and opportunities. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9 (2019), 411–421.

[17] S. Oh et al. 2019. IoT2 — The Internet of Tiny Things: Realizing mm-scale sensors through 3D die stacking. In *Design, Automation Test in Europe Conference Exhibition*, 2019.

[18] M. E. Ilas and C. Ilas. 2020. Towards real-time and real-life image classification and detection using CNN: A review of practical applications requirements, algorithms, hardware and current trends. In *Int. Symp. for Design and Technology in Electronic Packaging*, 2020.

[19] J. S. P. Giraldo, S. Lauwereins, K. Badami, H. V. Hamme, and M. Verhelst. 2019. 18μW SoC for near-microphone keyword spotting and speaker verification. In *Symposium on VLSI Circuits*, 2019.

[20] T. Norrie et al. 2021. The design process for Google's training chips: TPUv2 and TPUv3. *IEEE Micro* 41 (2021), 56–63, 2021.

[21] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza. 2019. Dissecting the graphcore IPU architecture via microbenchmarking. *CoRR,* vol. abs/1912.03413, 2019.

[22] K. Guo et al. 2020. Neural network accelerator comparison. 2020. [Online]. Available: http://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/. [Accessed 22 03 2021].

[23] A. Reuther et al. 2019. Survey and benchmarking of machine learning accelerators. In *IEEE High Perf. Extreme Computing Conf.*, 2019.

[24] A. Reuther et al. 2020. Survey of machine learning accelerators. In *IEEE High Perf. Extreme Computing Conf.*, 2020.

[25] C. D. Schuman et al. 2017. A survey of neuromorphic computing and neural networks in hardware. *CoRR,* vol. abs/1705.06963, 2017.

[26] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer. 2020. Efficient processing of deep neural networks. *Synthesis Lectures on Comp. Architecture*, 2020.

[27] V. Sze, Y. Chen, T. Yang, and J. S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE* 105 (2017), 2295–2329.

[28] V. Sze, Y. Chen, J. Emer, A. Suleiman, and Z. Zhang. 2017. Hardware for machine learning: Challenges and opportunities. In *IEEE Custom Integrated Circuits Conference*, 2017.

[29] S. Bodiwala and N. Nanavati. 2020. Efficient hardware implementations of deep neural networks: A survey. In *International Conference on Inventive Systems and Control*, 2020.
[30] S. Mittal. 2018. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications,* 2018.
[31] M. Capra et al. 2020. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access* 8 (2020), 225134–225180.
[32] V. Gadepally et al. 2019. AI enabling technologies: A survey. *CoRR,* vol. abs/1905.03592, 2019.
[33] J. Tang et al. 2019. Bridging biological and artificial neural networks with emerging neuromorphic devices: Fundamentals, progress, and challenges. *Advanced Materials* 31 (2019), 1902761.
[34] K. J. Lee, J. Lee, S. Choi, and H. J. Yoo. 2020. The development of silicon for AI: Different design approaches. *IEEE Trans. Circuits and Systems I: Regular Papers* 67 (2020), 4719–4732.
[35] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6 (2020), 264–274.
[36] D. Xu et al. 2003. Edge intelligence: Architectures, challenges, and applications. *CoRR,* vol. abs/2003.12172, 2020.
[37] Z. Du et al. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *Int. Symposium on Computer Architecture*, 2015.
[38] S. Han et al. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Int. Symposium on Computer Architecture*, 2016.
[39] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst. 2017. 14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI. In *IEEE Int. Solid-State Circuits Conference*, 2017.
[40] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze. 2017. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. of Solid-State Circuits* 52, 1 (2017), 127–138.
[41] R. Andri, L. Cavigelli, D. Rossi, and L. Benini. 2018. YodaNN: An architecture for ultralow power binary-weight CNN acceleration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems* 37 (2018), 48–60.
[42] J. Lee et al. 2019. UNPU: An energy-efficient deep neural network accelerator with fully variable weight bit precision. *IEEE J. Solid-State Circ.* 54 (2019), 173–185.
[43] Y. Chen, T. Yang, J. Emer, and V. Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9 (2019), 292–308.
[44] P. K. D. Pramanik et al. 2019. Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage. *IEEE Access* 7 (2019), 182113–182172.
[45] I. Miro-Panades et al. 2020. SamurAI: A 1.7MOPS-36GOPS adaptive versatile IoT node with 15,000× peak-to-idle power reduction, 207ns wake-up time and 1.3TOPS/W ML efficiency. In *IEEE Symposium on VLSI Circuits*, 2020.
[46] J.-H. Kim, C. Kim, K. Kim, and H.-J. Yoo. 2019. An ultra-low-power analog-digital hybrid CNN face recognition processor integrated with a CIS for always-on mobile devices. In *IEEE Int. Symposium on Circuits and Systems*, 2019.
[47] P. Gysel, M. Motamedi, and S. Ghiasi. 2016. Hardware-oriented approximation of convolutional neural networks. *CoRR,* vol. abs/1604.03168, 2016.
[48] A. Gholami et al. 2021. A survey of quantization methods for efficient neural network inference. *CoRR,* 2021.
[49] A. Ignatov et al. 2019. AI benchmark: All about deep learning on smartphones in 2019. *CoRR,* vol. abs/1910.06663, 2019.
[50] V. J. Reddi et al. 2019. MLPerf inference benchmark. *CoRR,* vol. abs/1911.02549, 2019.
[51] C. R. Banbury et al. 2020. Benchmarking TinyML systems: Challenges and direction. *CoRR,* vol. abs/2003.04821, 3 2020.
[52] EEMBC, Exploring CoreMark – A Benchmark Maximizing Simplicity and Efficacy, 2009. [Online] Available: https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf. [Accessed 03 02 2021].
[53] S. Williams, A. Waterman, and D. Patterson. 2019. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (2019), 65–76.
[54] J. Neumann. 1993. First draft of a report on the EDVAC. *IEEE Annals of the History of Computing* 15 (1993), 27–75.
[55] J. Silc, B. Robic, and T. Ungerer. 1999. Multiple-issue processors. In *Processor Architecture: From Dataflow to Superscalar and Beyond*, Berlin: Springer Berlin, 1999, 123–219.
[56] N. P. Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Int. Symposium on Computer Architecture*, 2017.
[57] Y. Chen, J. Emer, and V. Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *In International Symposium on Computer Architecture*, 2016.
[58] Z. Liu, P. N. Whatmough, and M. Mattina. 2020. Systolic tensor array: An efficient structured-sparse GEMM accelerator for mobile CNN inference. *IEEE Comp. Architecture Letters* 19 (2020), 34–37.

[59] A. Samajdar, Y. Zhu, P. N. Whatmough, M. Mattina, and T. Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator. *CoRR*, vol. abs/1811.02883, 2018.
[60] T.-J. Yang and V. Sze. 2019. Design considerations for efficient deep neural networks on processing-in-memory accelerators. In *IEEE International Electron Devices Meeting*, 2019.
[61] S. A. McKee. 2004. *Reflections on the Memory Wall.* In *Proceedings of the 1st Conference on Computing Frontiers*, New York, NY, USA, 2004.
[62] S. Yu, X. Sun, X. Peng, and S. Huang. 2020. Compute-in-Memory with emerging nonvolatile-memories: Challenges and prospects. In *IEEE Custom Integrated Circuits Conference*, 2020.
[63] X. Si et al. 2019. 24.5 A Twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning. In *IEEE International Solid- State Circuits Conference*, 2019.
[64] J. Zhang, Z. Wang, and N. Verma. 2017. In-memory computation of a machine-learning classifier in a standard 6T SRAM array. *IEEE Journal of Solid-State Circuits* 52 (2017), 915–924.
[65] M. Kang, S. Lim, S. Gonugondla, and N. R. Shanbhag. 2018. An in-memory VLSI architecture for convolutional neural networks. *IEEE J. Emerging and Sel. Topics in Circ. and Sys.* 8 (2018), 494–505.
[66] J.-W. Su et al. 2021. 16.3 A 28nm 384kb 6T-SRAM computation-in-memory macro with 8b precision for AI edge chips. In *2021 IEEE International Solid- State Circuits Conference*, 2021.
[67] S. Yu. 2018. Neuro-inspired computing with emerging nonvolatile memorys. *Proceedings of the IEEE* 106 (2018), 260–285.
[68] F. Cai et al. 2019. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations. *Nature Electronics* 2 (2019) 1, 7 2019.
[69] S. Dash, Y. Luo, A. Lu, S. Yu, and S. Mukhopadhyay. 2021. Robust processing-in-memory with multi-bit ReRAM using Hessian-driven mixed-precision computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1–1, 2021.
[70] C. Xue et al. 2019. 24.1 A 1Mb multibit ReRAM computing-in-memory macro with 14.6ns parallel MAC computing time for CNN based AI edge processors. In *IEEE Int. Solid- State Circuits Conference*, 2019.
[71] S. Yin et al. 2019. Monolithically integrated RRAM- and CMOS-based in-memory computing optimizations for efficient deep learning. *IEEE Micro* 39 (2019), 54–63.
[72] G. Murali, X. Sun, S. Yu, and S. K. Lim. 2021. Heterogeneous mixed-signal monolithic 3-D in-memory computing using resistive RAM. *IEEE Trans. Very Large Scale Integration Systems* 29 (2021), 386–396.
[73] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu. 2019. DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies. In *IEEE International Electron Devices Meeting*, 2019.
[74] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo. 2020. McDRAM v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge. *IEEE Access* 8 (2020), 135223–135243.
[75] G. E. Moore. Cramming more components onto integrated circuits. *Reprinted from Electronics* 38, 8, (1965), 114. *IEEE Solid-State Circuits Society Newsletter*, vol. 11, pp. 33–35, 2006.
[76] R. S. Williams. 2017. What's next? [The end of Moore's law]. *Computing in Science Engineering* 19 (2017), 7–13.
[77] M. Bohr. 2007. A 30 year retrospective on Dennard's MOSFET scaling paper. *IEEE Solid-State Circ. Society Newsl.* 12 (2007), 11–13.
[78] P. Bose. 2011. Power Wall. In *Encyclopedia of Parallel Computing, D. Padua, Hrsg.*, Boston, MA: Springer US, 2011, 1593–1608.
[79] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *IEEE International Solid-State Circuits Conference*, 2014.
[80] Y. Lee et al. 2013. A modular 1 mm³die-stacked sensing platform with low power I²C inter-die communication and multi-modal energy harvesting. *IEEE J. Solid-State Circuits* 48, 1 (2013), 229–243.
[81] K. Ueyoshi et al. 2018. QUEST: A 7.49TOPS multi-purpose log-quantized DNN inference engine stacked on 96MB 3D SRAM using inductive-coupling technology in 40nm CMOS. In *IEEE Int. Solid-State Circuits Conference*, 2018.
[82] IRDS. 2021. *International Roadmap for Devices and Systems: 2020 Update,* 2020. [Online] Available: https://irds.ieee.org/editions/2020. [Accessed 11 05 2021].
[83] C. Auth et al. 2012. A 22nm high performance and low-power CMOS technology featuring fully-depleted tri-gate transistors, self-aligned contacts and high density MIM capacitors. In *Symp. on VLSI Technology*, 2012.
[84] H. N. Patel et al. 2016. A 55nm ultra low leakage deeply depleted Channel technology optimized for energy minimization in subthreshold SRAM and logic. In *European Solid-State Circuits Conference*, 2016.
[85] S. Natarajan et al. 2014. A 14nm logic technology featuring 2nd-generation FinFET, air-gapped interconnects, self-aligned double patterning and a 0.0588 μm2 SRAM cell size. In *IEEE Int. Electron Dev. Meeting*, 2014.

[86] O. Weber. 2017. FDSOI vs FinFET: Differentiating device features for ultra low power IoT applications. In *IEEE Int. Conf. on IC Design and Technology*, 2017.
[87] R. Carter et al. 2016. 22nm FDSOI technology for emerging mobile, Internet-of-Things, and RF applications. In *IEEE Int. Electron Dev. Meet.*, 2016.
[88] J.-H. Lee, J.-W. Lee, H.-A.-R. Jung, and B.-K. Choi. 2009. Comparison of SOI FinFETs and Bulk FinFETs. *ECS Trans.* 19 (2009), 101–112.
[89] M. Pons et al. 2013. Ultra low-power standard cell design using planar bulk CMOS in subthreshold operation. In *International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2013.
[90] M. Pons et al. 2015. A 1kb single-side read 6T sub-threshold SRAM in 180 nm with 530 Hz frequency 3.1 nA total current and 2.4 nA leakage at 0.27 V. In *IEEE SOI-3D-Subthr. Microel. Tech. Unified Conf.*, 2015.
[91] M. Pons et al. 2016. Sub-threshold latch-based icyflex2 32-bit processor with wide supply range operation. In *Europ. Solid-State Circuits Conf.*, 2016.
[92] M. Pons et al. 2019. A 0.5 V 2.5 μW/MHz microcontroller with analog-assisted adaptive body bias PVT compensation with 3.13nW/kB SRAM retention in 55nm deeply-depleted Channel CMOS. In *IEEE Custom Integrated Circuits Conference*, 2019.
[93] T. C. Müller et al. 2017. PVT compensation in Mie Fujitsu 55 nm DDC: A standard-cell library based comparison. In *IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference*, 2017.
[94] M. Weiser, B. Welch, A. Demers, and S. Shenker. 1994. Scheduling for reduced CPU ewnergy. In *USENIX Conference on Operating Systems Design and Implementation*, USA, 1994.
[95] T. D. Burd and R. W. Brodersen. 1996. Processor design for portable systems. In *Technologies for Wireless Computing*, USA, Kluwer Academic Publishers, 1996, 203–221.
[96] B. Moons and M. Verhelst. 2015. DVAS: Dynamic voltage accuracy scaling for increased energy-efficiency in approximate computing. *In Int. Symposium on Low Power Electronics and Design*, 2015.
[97] X. Yang et al. 2016. A systematic approach to blocking convolutional neural networks. *CoRR*, vol. abs/1606.04209, 2016.
[98] A. Chen. 2016. A review of emerging non-volatile memory (NVM) technologies and applications. *Solid-State Electronics* 125 (2016), 25–38.
[99] J. Park. 2020. Neuromorphic computing using emerging synaptic devices: A retrospective summary and an outlook. *Electronics* 9, 2020.
[100] J. Y. Wu et al. 2018. A 40nm low-power logic compatible phase change memory technology. In *IEEE Int. Electron Devices Meeting*, 2018.
[101] K. Lee et al. 2019. 1Gbit high density embedded STT-MRAM in 28nm FDSOI technology. In *IEEE Int. Electron Devices Meeting*, 2019.
[102] L. Wei et al. 2019. 13.3 A 7Mb STT-MRAM in 22FFL FinFET technology with 4ns read sensing time at 0.9V using write-verify-write scheme and offset-cancellation sensing technique. In *IEEE International Solid- State Circuits Conference*, 2019.
[103] Y. J. Song et al. 2018. Demonstration of highly manufacturable STT-MRAM embedded in 28nm logic. In *IEEE Int. Electron Devices Meeting*, 2018.
[104] T. Liu et al. 2013. A 130.7mm2 2-layer 32Gb ReRAM memory device in 24nm technology. In *IEEE Int. Solid-State Circuits Conference*, 2013.
[105] C. Chou et al. 2018. An N40 256K×44 embedded RRAM macro with SL-precharge SA and low-voltage current limiter to improve read and write performance. In *IEEE Int. Solid - State Circuits Conference*, 2018.
[106] P. Jain et al. 2019. 13.2 A 3.6Mb 10.1Mb/mm2 embedded non-volatile ReRAM macro in 22nm FinFET technology with adaptive forming/set/reset schemes yielding down to 0.5V with sensing time of 5ns at 0.7V. In *IEEE Int. Solid-State Circuits Conference*, 2019.
[107] J. Yang et al. 2021. 24.2 A 14nm-FinFET 1Mb embedded 1T1R RRAM with a 0.022μ m2 cell size using self-adaptive delayed termination and multi-cell reference. In *IEEE Int. Solid- State Circuits Conf.*, 2021.
[108] S. Beyer et al. 2020. FeFET: A versatile CMOS compatible device with game-changing potential. In *IEEE Int. Memory Workshop*, 2020.
[109] D. Reis et al. 2019. Design and analysis of an ultra-dense, low-leakage, and fast FeFET-based random access memory array. *IEEE J. Exploratory Solid-State Comp. Dev. and Circ.* 5 (2019), 103–112.
[110] S. Bang et al. 2017. 14.7 A 288μW programmable deep-learning processor with 270KB on-chip weight storage using non-uniform memory hierarchy for mobile intelligence. In *IEEE Int. Solid-State Circuits Conf.*, 2017.
[111] J. Lee et al. 2020. AμProcessor layer for mm-scale die-stacked sensing platforms featuring ultra-low power sleep mode at 125°C. In *2020 IEEE Asian Solid-State Circuits Conference*, 2020.
[112] K. Bong, S. Choi, C. Kim, D. Han, and H. J. Yoo. 2018. A low-power convolutional neural network face recognition processor and a CIS integrated with always-on face detector. *IEEE Journal of Solid-State Circuits* 53, 1 2018, 115–123.

[113] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg. 2016. Power, area, and performance optimization of standard cell memory arrays through controlled placement. *ACM Trans. Des. Autom. Electron. Syst.* 21, 5 (2016).
[114] F. Conti, P. Davide Schiavone, and L. Benini. 2018. XNOR Neural Engine: A hardware accelerator IP for 21.6 fJ/op binary neural network inference. *ArXiv e-prints,* 7 2018.
[115] J. Jeddeloh and B. Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *Symposium on VLSI Technology*, 2012.
[116] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis. 2017. TETRIS: Scalable and efficient neural network acceleration with 3D memory. *SIGARCH Comput. Archit. News* 45, 4 (2017), 751–764.
[117] R. Giterman, A. Fish, A. Burg, and A. Teman. 2018. A 4-transistor nMOS-only logic-compatible gain-cell embedded DRAM with over 1.6-ms retention time at 700 mV in 28-nm FD-SOI. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 4 (2018), 1245–1256.
[118] R. Giterman, A. Teman, and A. Fish. 2018. A 14.3pW sub-threshold 2T gain-cell eDRAM for ultra-low power IoT applications in 28nm FD-SOI. In *IEEE SOI-3D-Subthreshold Microel. Tech. Unified Conf.*, 2018.
[119] P. Pavan, R. Bez, P. Olivo, and E. Zanoni. 1997. Flash memory cells-an overview. *Proceedings of the IEEE* 85 (1997), 1248–1271.
[120] Cypress, SONOS flash technology, 2019. [Online]. Available: https://www.cypress.com/file/123341/download. [Accessed 21 03 2021].
[121] K. Goetschalckx and M. Verhelst. 2019. Breaking high-resolution CNN bandwidth barriers with enhanced depth-first execution. *IEEE J. on Emerging and Sel. Topics in Circ. and Sys.* 9 (2019), 323–331.
[122] S. Colleman and M. Verhelst. 2021. High-utilization, high-flexibility depth-first CNN coprocessor for image pixel processing on FPGA. *IEEE Trans. Very Large Scale Integration Systems* 29 (2021), 461–471.
[123] M. Alwani, H. Chen, M. Ferdman, and P. Milder. 2016. Fused-layer CNN accelerators. In *IEEE/ACM Int. Symp. on Microarchitecture*, 2016.
[124] M. Scherer, G. Rutishauser, L. Cavigelli, and L. Benini. 2021. CUTIE: Beyond PetaOp/s/W Ternary DNN inference acceleration with better-than-binary energy efficiency. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 1–1, 2021.
[125] A. Stoutchinin, F. Conti, and L. Benini. 2019. Optimally scheduling CNN convolutions for efficient memory access. *CoRR,* vol. abs/1902.01492, 2019.
[126] K. Siu, D. M. Stuart, M. Mahmoud, and A. Moshovos. 2018. Memory requirements for convolutional neural network hardware accelerators. In *IEEE Int. Symp. on Workload Characterization*, 2018.
[127] S. Zheng et al. 2020. Efficient scheduling of irregular network structures on CNN accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39 (2020), 3408–3419.
[128] L. Lai, N. Suda, and V. Chandra. 2018. CMSIS-NN: Efficient neural network kernels for Arm Cortex-M CPUs. *CoRR,* vol. abs/1801.06601, 2018.
[129] E. De Giovanni et al. 2020. Modular design and optimization of biomedical applications for ultralow power heterogeneous platforms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39 (2020), 3821–3832.
[130] H. G. Chen et al. 2016. ASP Vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels. *CoRR,* vol. abs/1605.03621, 2016.
[131] P. Pad et al. 2020. Efficient neural vision systems based on convolutional image acquisition. In *Conf. on Comp. Vision and Pattern Recog.*, 2020.
[132] Z. Wang, J. Zhang, and N. Verma. 2015. Realizing low-energy classification systems by implementing matrix multiplication directly within an ADC. *IEEE Trans. Biomed. Circuits Syst.* 9 (2015), 825–837.
[133] R. LiKamWa, Y. Hou, Y. Gao, M. Polansky, and L. Zhong. 2016. RedEye: Analog ConvNet image sensor architecture for continuous mobile vision. In *ACM/IEEE Int. Symp. on Computer Architecture*, 2016.
[134] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2017. Distributed deep neural networks over the cloud, the edge and end devices. In *IEEE Int. Conference on Distributed Computing Systems*, 2017.
[135] Z. Zhao, K. M. Barijough, and A. Gerstlauer. 2018. DeepThings: Distributed adaptive deep learning inference on resource-constrained IoT edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37 (2018), 2348–2359.
[136] S. Teerapittayanon, B. McDanel, and H. T. Kung. 2017. BranchyNet: Fast inference via early exiting from deep neural networks. *CoRR,* vol. abs/1709.01686, 2017.
[137] P. Panda, A. Sengupta and K. Roy. 2016. Conditional deep learning for energy-efficient and enhanced pattern recognition. In *2016 Design, Automation Test in Europe Conference Exhibition*, 2016.
[138] P. P. Bernardo, C. Gerum, A. Frischknecht, K. Lübeck, and O. Bringmann. 2020. UltraTrail: A configurable ultralow-power TC-ResNet AI accelerator for efficient keyword spotting. *IEEE Trans. on Computer-Aided Design of Integrated Circ. and Sys.* 39 (2020), 4240–4251.

[139] L. Liu and J. Deng. 2017. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *CoRR,* vol. abs/1701.00299, 2017.
[140] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib. 2015. Scalable-effort classifiers for energy-efficient machine learning. In *Design Automation Conference*, New York, NY, USA, 2015.
[141] P. Jokic, S. Emery, and L. Benini. 2020. Improving memory utilization in convolutional neural network accelerators. *IEEE Embedded Systems Letters,* 1–1, 2020.
[142] S. Han, H. Mao, and W. J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *Int. Conf. on Learning Representations*, 2016.
[143] Y. Liang, L. Lu, Q. Xiao, and S. Yan. 2020. Evaluating fast algorithms for convolutional neural networks on FPGAs. *IEEE Trans. on Computer-Aided Design of Integrated Circ. and Sys.* 39 (2020), 857–870.
[144] A. Lavin and S. Gray. 2016. Fast algorithms for convolutional neural networks. In *CVPR,* 2016.
[145] A. G. Howard et al. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *CoRR,* vol. abs/1704.04861, 2017.
[146] M. Mathieu, M. Henaff, and Y. LeCun. 2014. Fast training of convolutional networks through FFTs. In *Int. Conference on Learning Representations, Banff, AB, Canada*, 2014.
[147] Y. Zhang and X. Li. 2020. Fast convolutional neural networks with fine-grained FFTs. In *International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2020.
[148] J. Cong and B. Xiao. 2014. Minimizing computation in convolutional neural networks. In *Artificial Neural Networks and Machine Learning – ICANN 2014*, Cham, 2014.
[149] Y. LeCun, J. S. Denker, and S. A. Solla. 1989. Optimal brain damage. In *NIPS*, 1989.
[150] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *CoRR,* vol. abs/2102.00554, 2021.
[151] D. Molchanov, A. Ashukha, and D. Vetrov. 2017. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, 2017.
[152] T. Elsken, J. H. Metzen, and F. Hutter. 2019. Neural architecture search: A survey. *J. Mach. Learn. Res.* 20, 1 (2019), 1997–2017.
[153] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013.
[154] B. A. Plummer, N. Dryden, J. Frost, T. Hoefler, and K. Saenko. 2020. Shapeshifter networks: Decoupling layers from parameters for scalable and effective deep learning. *CoRR,* vol. abs/2006.10598, 2020.
[155] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE* 108 (2020), 485–532.
[156] S. Han, J. Pool, J. Tran, and W. Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 2015.
[157] T. Yang, Y. Chen, and V. Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
[158] A. Aimar et al. 2019. NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps. *IEEE Trans. on Neural Netw. and Learning Sys.* 30 (2019), 644–656.
[159] J. Albericio et al. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *Int. Symp. on Computer Architecture*, 2016.
[160] L. Cavigelli, P. Degen, and L. Benini. 2017. CBinfer: Change-based inference for convolutional neural networks on video data. *CoRR,* vol. abs/1704.04313, 2017.
[161] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello. 2014. A 240 G-ops/s mobile coprocessor for deep neural networks. In *IEEE Conf. on Computer Vision and Pattern Recognition Workshops*, 2014.
[162] T. Chen et al. 2014. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *SIGARCH Comput. Archit. News.* 42, 2 (2014), 269–284.
[163] Z. Yuan, Y. Liu, J. Yue, J. Li, and H. Yang. 2017. CORAL: Coarse-grained reconfigurable architecture for convolutional neural networks. In *IEEE ISLPED*, 2017.
[164] R. Andri, L. Cavigelli, D. Rossi, and L. Benini. 2019. Hyperdrive: A multi-chip systolically scalable binary-weight CNN inference engine. *IEEE J. on Emerg. and Sel. Topics in Circ. and Sys.* 9 (2019), 309–322.
[165] F. H. Sinz, X. Pitkow, J. Reimer, M. Bethge, and A. S. Tolias. 2019. Engineering a \less artificial intelligence. *Neuron* 103 (2019), 967–979.
[166] F. N. Iandola et al. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR,* vol. abs/1602.07360, 2016.
[167] S. Liu et al. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Int. Conf. on Mobile Systems, Applications, and Services*, New York, NY, USA, 2018.

[168] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han. 2020. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.
[169] M. Alioto, V. De, and A. Marongiu. 2018. Energy-quality scalable integrated circuits and systems: continuing energy scaling in the twilight of Moore's law. *IEEE J. on Emerging and Selected Topics in Circuits and Systems* 8 (2018), 653–678.
[170] J. H. Teo, S. Cheng, and M. Alioto. 2020. Low-energy voice activity detection via energy-quality scaling from data conversion to machine learning. *IEEE Trans. on Circ. and Sys.* 67 (2020), 1378–1388.
[171] A. Alvarez, G. Ponnusamy, and M. Alioto. 2020. Energy-quality scalable memory-frugal feature extraction for always-on deep sub-mW distributed vision. *IEEE Access* 8 (2020), 18951–18961.
[172] P. Yin et al. 2019. Understanding straight-through estimator in training activation quantized neural nets. *CoRR,* vol. abs/1903.05662, 2019.
[173] W. Sung, S. Shin, and K. Hwang. 2015. Resiliency of deep neural networks under quantization. *CoRR,* vol. abs/1511.06488, 2015.
[174] M. Courbariaux and Y. Bengio. 2016. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or −1. *CoRR,* vol. abs/1602.02830, 2016.
[175] M. Gao, Q. Wang, A. S. K. Nagendra, and G. Qu. 2017. A novel data format for approximate arithmetic computing. In *Asia and South Pacific Design Automation Conference*, 2017.
[176] D. Shin, J. Lee, J. Lee, J. Lee, and H. Yoo. 2018. DNPU: An energy-efficient deep-learning processor with heterogeneous multi-core architecture. *IEEE Micro* 38 (2018), 85–93.
[177] V. Camus, L. Mei, C. Enz, and M. Verhelst. 2019. Review and benchmarking of precision-scalable multiply-accumulate unit architectures for embedded neural-network processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9 (2019), 697–711.
[178] D. D. Lin, S. S. Talathi and V. S. Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *Int. Conf. on Machine Learning - Volume 48*, New York, NY, USA, 2016.
[179] A. K. Mishra and D. Marr. 2017. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *CoRR,* vol. abs/1711.05852, 2017.
[180] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. 2015. Approximate computing and the quest for computing efficiency. In *Design Automation Conference*, 2015.
[181] M. Gao, Q. Wang, M. T. Arafin, Y. Lyu, and G. Qu. 2017. Approximate computing for low power and security in the Internet of Things. *Computer* 50 (2017), 27–34.
[182] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han. 2020. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE* 108 (2020), 2108–2135.
[183] S. Tajasob, M. Rezaalipour, M. Dehyadegari, and M. N. Bojnordi. 2018. Designing efficient imprecise adders using multi-bit approximate building blocks. In *Int. Symp. on Low Power Electronics and Design*, New York, NY, USA, 2018.
[184] L. Yang, D. Bankman, B. Moons, M. Verhelst, and B. Murmann. 2018. Bit error tolerance of a CIFAR-10 binarized convolutional neural network processor. In *IEEE Int. Symp. on Circuits and Systems*, 2018.
[185] L. Sousa. 2021. Nonconventional computer arithmetic circuits, systems and applications. *IEEE Circ. and Sys. Magazine,* 21 (2021), 6–40.
[186] Y. Popoff et al. 2016. High-efficiency logarithmic number unit design based on an improved cotransformation scheme. In *Design, Automation Test in Europe Conference Exhibition*, 2016.
[187] K. Roy, A. Jaiswal, and P. Panda. 2019. Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575 (2019), 607—617.
[188] F. Akopyan et al. 2015. TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip. *Trans. on Comp.-Aided Design of Integr. Circ. and Sys.* 34 (2015), 1537–1557.
[189] M. Davies et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro* 38 (2018), 82–99.
[190] H. Paugam-Moisy and S. M. Bohte. 2012. Computing with spiking neuron networks. In *Handbook of Natural Computing*, G. Rozenberg, T. Back and J. Kok, Hrsg., (Eds.). Springer-Verlag, 335–376.
[191] H. Jang, O. Simeone, B. Gardner, and A. Gruning. 2019. An introduction to probabilistic spiking neural networks: Probabilistic models, learning rules, and applications. *Sig. Proc. Mag.* 36 (2019), 64–77.
[192] Kanerva and Pentti. 2009. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation* 1, 6 (2009).
[193] B. Murmann. 2021. Mixed-signal computing for deep neural network inference. *IEEE Trans. on Very Large Scale Integr. Sys.* 29 (2021), 3–13.
[194] W. Haensch, T. Gokmen, and R. Puri. 2019. The next generation of deep learning hardware: Analog computing. *Proceedings of the IEEE* 107 (2019), 108–122.
[195] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann. 2018. An always-on 3.8 uJ/86% CIFAR-10 mixed-signal binary CNN processor with all memory on chip in 28nm CMOS. In *IEEE International Solid - State Circuits Conference*, 2018.

[196] B. Moons, D. Bankman, L. Yang, B. Murmann, and M. Verhelst. 2018. BinarEye: An always-on energy-accuracy-scalable binary CNN processor with all memory on chip in 28nm CMOS. *CoRR*, vol. abs/1804.05554, 2018.
[197] D. Bankman and B. Murmann. 2016. An 8-bit, 16 input, 3.2 pJ/op switched-capacitor dot product circuit in 28-nm FDSOI CMOS. In *IEEE Asian Solid-State Circuits Conference*, 2016.
[198] K. Bong et al. 2017. 14.6 A 0.62mW ultra-low-power convolutional-neural-network face-recognition processor and a CIS integrated with always-on Haar-like face detector. In *IEEE Int. Solid-State Circuits Conf.*, 2017.
[199] J. Oh, J. Park, G. Kim, S. Lee, and H. Yoo. 2011. A 57mW embedded mixed-mode neuro-fuzzy accelerator for intelligent multi-core processor. In *IEEE International Solid-State Circuits Conference*, 2011.
[200] M. J. Marinella et al. 2018. Multiscale co-design analysis of energy, latency, area, and accuracy of a ReRAM analog neural training accelerator. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8 (2018), 86–101.
[201] S. Anwar, K. Hwang, and W. Sung. 2015. Fixed point optimization of deep convolutional neural networks for object recognition. In *IEEE Int. Conference on Acoustics, Speech and Signal Processing*, 2015.
[202] P. Judd, J. Albericio, and A. Moshovos. 2017. Stripes: Bit-serial deep neural network computing. *IEEE Comp. Arch. Letters* 16 (2017), 80–83.

Received July 2021; revised January 2022; accepted February 2022