Efficient Processing of Deep Neural Networks

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel Emer
Massachusetts Institute of Technology

Reference:
V. Sze, Y.-H.Chen, T.-J. Yang, J. S. Emer, "Efficient Processing of Deep Neural Networks,"
Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers, 2020

For book updates, sign up for mailing list at http://mailman.mit.edu/mailman/listinfo/eems-news

June 15, 2020

Abstract

This book provides a structured treatment of the key principles and techniques for enabling efficient processing of deep neural networks (DNNs). DNNs are currently widely used for many artificial intelligence (AI) applications, including computer vision, speech recognition, and robotics. While DNNs deliver state-of-the-art accuracy on many AI tasks, it comes at the cost of high computational complexity. Therefore, techniques that enable efficient processing of deep neural networks to improve key metrics—such as *energy-efficiency*, *throughput*, and *latency*—without sacrificing accuracy or increasing hardware costs are critical to enabling the wide deployment of DNNs in AI systems.

The book includes background on DNN processing; a description and taxonomy of hardware architectural approaches for designing DNN accelerators; key metrics for evaluating and comparing different designs; features of DNN processing that are amenable to hardware/algorithm co-design to improve energy efficiency and throughput; and opportunities for applying new technologies. Readers will find a structured introduction to the field as well as formalization and organization of key concepts from contemporary work that provide insights that may spark new ideas.

Contents

Preface

Deep neural networks (DNNs) have become extraordinarily popular; however, they come at the cost of high computational complexity. As a result, there has been tremendous interest in enabling efficient processing of DNNs. The challenge of DNN acceleration is threefold:

- to achieve high performance and efficiency,
- to provide sufficient flexibility to cater to a wide and rapidly changing range of workloads, and
- to integrate well into existing software frameworks.

In order to understand the current state of art in addressing this challenge, this book aims to provide an overview of DNNs, the various tools for understanding their behavior, and the techniques being explored to efficiently accelerate their computation. It aims to explain foundational concepts and highlight key design considerations when building hardware for processing DNNs rather than trying to cover all possible design configurations, as this is not feasible given the fast pace of the field (see Figure 1). It is targeted at researchers and practitioners who are familiar with computer architecture who are interested in how to efficiently process DNNs or how to design DNN models that can be efficiently processed. We hope that this book will provide a structured introduction to readers who are new to the field, while also formalizing and organizing key concepts to provide insights that may spark new ideas for those who are already in the field.

Organization

This book is organized into three modules that each consist of several chapters. The first module aims to provide an overall background to the field of DNN and insight on characteristics of the DNN workload.

- Chapter 1 provides background on the context of why DNNs are important, their history, and their applications.
- Chapter 2 gives an overview of the basic components of DNNs and popular DNN models currently in use. It also describes the various resources used for DNN research and development. This includes discussion of the various software frameworks, and the public datasets that are used for training and evaluation.

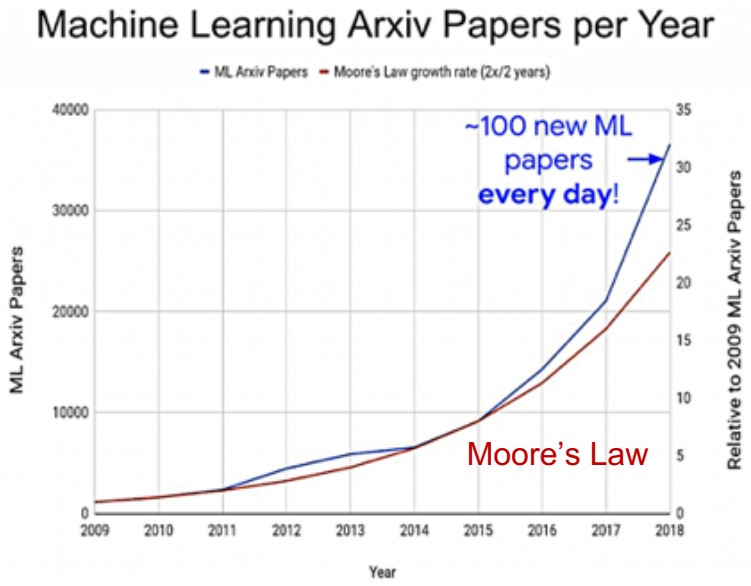


Figure 1: It's been observed that the number of ML publications are growing exponentially at a faster rate than Moore's law! (Figure from [1].)

The second module focuses on the design of hardware for processing DNNs. It discusses various architecture design decisions depending on the degree of customization (from general purpose platforms to full custom hardware) and design considerations when mapping the DNN workloads onto these architectures. Both temporal and spatial architectures are considered.

- Chapter 3 describes the key metrics that should be considered when designing or comparing various DNN accelerators.
- Chapter 4 describes how DNN kernels can be processed, with a focus on temporal architectures such as CPUs and GPUs. To achieve greater efficiency, such architectures generally have a cache hierarchy and coarser-grained computational capabilities, e.g., vector instructions, making the resulting computation more efficient. Frequently for such architectures, DNN processing can be transformed into a matrix multiplication, which has many optimization opportunities. This chapter also discusses various software and hardware optimizations used to accelerate DNN computations on these platforms without impacting application accuracy.
- Chapter 5 describes the design of specialized hardware for DNN processing, with a focus on spatial architectures. It highlights the processing order and resulting data movement in the hardware used to process a DNN, and the relationship to a loop nest representation of a DNN. The order of the loops in the loop nest is referred to as the *dataflow*, and it determines how often each piece of data needs to be moved. The limits of the loops in the loop nest describe how to break the DNN workload into smaller pieces, referred to as *tiling/blocking* to account for the limited storage capacity at different levels of the memory hierarchy.
- Chapter 6 presents the process of *mapping* a DNN workload on to a DNN accelerator. It describes the steps required to find an optimized mapping including enumerating all legal mappings, and searching those mappings by employing models that project throughput and energy efficiency.

The third module discusses how additional improvements in efficiency can be achieved either by moving up the stack through the co-design of the algorithms and hardware, or down the stack by using mixed signal circuits, and new memory or device technology. In the cases where the algorithm is modified, the impact on accuracy must be carefully evaluated.

- Chapter 7 describes how reducing the precision of data and computation can result in increased throughput and energy efficiency. It discusses how to reduce precision using quantization and the associated design considerations, including hardware cost and impact on accuracy.
- Chapter 8 describes how exploiting sparsity in DNNs can be used to reduce the footprint of the data, which provides an opportunity to reduce storage requirements, data movement, and arithmetic operations. It describes various sources of sparsity and techniques to increase sparsity. It then discusses how sparse DNN accelerators can translate sparsity into improvements in energy-efficiency and throughput. It also presents a new abstract data representation that can be used to express and obtain insight about the dataflows for a variety of sparse DNN accelerators.
- Chapter 9 describes how to optimize the structure of the DNN models (i.e., the 'network architecture' of the DNN) to improve both throughput and energy efficiency while trying to minimize impact on accuracy. It discusses both manual design approaches as well as automatic design approaches (i.e., neural architecture search).
- Chapter 10, on advanced technologies, discusses how mixed-signal circuits and new memory technologies can be used to bring the compute closer to the data (e.g., processing in memory) to address the expensive data movement that dominates throughput and energy consumption of DNNs. It also briefly discusses the promise of reducing energy consumption and increasing throughput by performing the computation and communication in the optical domain.

What's New?

This book is an extension of a tutorial paper written by the same authors entitled "Efficient Processing of Deep Neural Networks: A Tutorial and Survey" that appeared in the *Proceedings of the IEEE* in 2017 and slides from short courses given at ISCA and MICRO in 2016, 2017, and 2019 (slides available at http://eyeriss.mit.edu/tutorial.html). This book includes recent works since the publication of the tutorial paper along with a more in-depth treatment of topics such as dataflow, mapping, and processing in memory. We also provide updates on the fast-moving field of co-design of DNN models and hardware in the areas of reduced precision, sparsity, and efficient DNN model design. As part of this effort, we present a new way of thinking about sparse representations and give a detailed treatment of how to handle and exploit sparsity. Finally, we touch upon recurrent neural networks, auto encoders, and transformers, which we did not discuss in the tutorial paper.

Scope of book

The main goal of this book is to teach the reader how to tackle the computational challenge of efficiently processing DNNs rather than how to design DNNs for increased accuracy. As a result, this book does not cover training (only touching on it lightly), nor does it cover the theory of deep learning or how to design

DNN models (though it discusses how to make them efficient) or use them for different applications. For these aspects, please refer to other references such as Goodfellow's book [2] and Stanford cs231n course notes [3].

Acknowledgements

The authors would like to thank Margaret Martonosi for her persistent encouragement to write this book. We would also like to thank Liane Bernstein, Davis Blalock, Natalie Enright Jerger, Jose Javier Gonzalez Ortiz, Fred Kjolstad, Yi-Lun Liao, Andreas Moshovos, Boris Murmann, James Noraky, Angshuman Parashar, Michael Pellauer, Clément Pit-Claudel, Sophia Shao, Mahmhut Ersin Sinangil, Po-An Tsai, Marian Verhelst, Tom Wenisch, Diana Wofk, Nellie Wu, and students in our "Hardware Architectures for Deep Learning" class at MIT, who have provided invaluable feedback and discussions on the topics described in this book. We would also like to express our deepest appreciation to Robin Emer for her suggestions, support, and tremendous patience during the writing of this book.

As mentioned earlier in the Preface, this book is an extension of an earlier tutorial paper, which was based on tutorials we gave at ISCA and MICRO. We would like to thank David Brooks for encouraging us to do the first tutorial at MICRO in 2016, which sparked the effort that led to this book.

This work was funded in part by DARPA YFA, the DARPA contract HR0011-18-3-0007, the MIT Center for Integrated Circuits and Systems (CICS), the MIT-IBM Watson AI Lab, the MIT Quest for Intelligence, the NSF E2CDA 1639921, and gifts/faculty awards from Nvidia, Facebook, Google, Intel, and Qualcomm.

Part II

Design of Hardware for Processing DNNs

Chapter 3

Key Metrics and Design Objectives

Over the past few years, there has been a significant amount of research on efficient processing of DNNs. Accordingly, it is important to discuss the key metrics that one should consider when comparing and evaluating the strengths and weaknesses of different designs and proposed techniques and that should be incorporated into design considerations. While efficiency is often only associated with the number of operations per second per Watt (e.g., floating-point operations per second per Watt as FLOPS/W or tera-operations per second per Watt as TOPS/W), it is actually composed of many more metrics including accuracy, throughput, latency, energy consumption, power consumption, cost, flexibility, and scalability. Reporting a comprehensive set of these metrics is important in order to provide a complete picture of the trade-offs made by a proposed design or technique.

In this chapter, we will

- discuss the importance of each of these metrics;
- breakdown the factors that affect each metric. When feasible, present equations that describe the relationship between the factors and the metrics;
- describe how these metrics can be incorporated into design considerations for both the DNN hardware and the DNN model (i.e., workload); and
- specify what should be reported for a given metric to enable proper evaluation.

Finally, we will provide a case study on how one might bring all these metrics together for a holistic evaluation of a given approach. But first, we will discuss each of the metrics.

3.1 Accuracy

Accuracy is used to indicate the quality of the result for a given task. The fact that DNNs can achieve state-of-the-art accuracy on a wide range of tasks is one of the key reasons driving the popularity and wide use of DNNs today. The units used to measure accuracy depend on the task. For instance, for image classification,

accuracy is reported as the percentage of correctly classified images, while for object detection, accuracy is reported as the mean average precision (mAP), which is related to the trade off between the true positive rate and false positive rate.

Factors that affect accuracy include the difficulty of the task and dataset.[1] For instance, classification on ImageNet is much more difficult than on MNIST, and object detection or semantic segmentation is more difficult than classification. As a result, a DNN model that performs well on MNIST may not necessarily perform well on ImageNet.

Achieving high accuracy on difficult tasks or datasets typically requires more complex DNN models (e.g., a larger number of MAC operations and more distinct weights, increased diversity in layer shapes, etc.), which can impact how efficiently the hardware can process the DNN model.

Accuracy should therefore be interpreted in the context of the difficulty of the task and dataset.[2] Evaluating hardware using well-studied, widely used DNN models, tasks, and datasets can allow one to better interpret the significance of the accuracy metric. Recently, motivated by the impact of the SPEC benchmarks for general purpose computing [113], several industry and academic organizations have put together a broad suite of DNN models, called *MLPerf*, to serve as a common set of well-studied DNN models to evaluate the performance and enable fair comparison of various software frameworks, hardware accelerators, and cloud platforms for both training and inference of DNNs [114].[3] The suite includes various types of DNNs (e.g., CNN, RNN, etc.) for a variety of tasks including image classification, object identification, translation, speech-to-text, recommendation, sentiment analysis, and reinforcement learning.

3.2 Throughput and Latency

Throughput is used to indicate the amount of data that can be processed or the number of executions of a task that can be completed in a given time period. High throughput is often critical to an application. For instance, processing video at 30 frames per second is necessary for delivering real-time performance. For data analytics, high throughput means that more data can be analyzed in a given amount of time. As the amount of visual data is growing exponentially, high-throughput big data analytics becomes increasingly important, particularly if an action needs to be taken based on the analysis (e.g., security or terrorist prevention; medical diagnosis or drug discovery). Throughput is often generically reported as the number of operations per second. In the case of inference, throughput is reported as inferences per second or in the form of runtime in terms of seconds per inference.

Latency measures the time between when the input data arrives to a system and when the result is generated. Low latency is necessary for real-time interactive applications, such as augmented reality, autonomous navigation, and robotics. Latency is typically reported in seconds.

Throughput and latency are often assumed to be directly derivable from one another. However, they are actually quite distinct. A prime example of this is the well-known approach of batching input data (e.g., batching

[1] Ideally, robustness and fairness should be considered in conjunction with accuracy, as there is also an interplay between these factors; however, these are areas of on-going research and beyond the scope of this book.

[2] As an analogy, getting 9 out of 10 answers correct on a high school exam is different than 9 out of 10 answers correct on a college-level exam. One must look beyond the score and consider the difficulty of the exam.

[3] Earlier DNN benchmarking efforts including DeepBench [115] and Fathom [116] have now been subsumed by MLPerf.

multiple images or frames together for processing) to increase throughput since it amortizes overhead, such as loading the weights; however, batching also increases latency (e.g., at 30 frames per second and a batch of 100 frames, some frames will experience at least 3.3 second delay), which is not acceptable for real-time applications, such as high-speed navigation where it would reduce the time available for course correction. Thus, achieving low latency and high throughput simultaneously can sometimes be at odds depending on the approach and both should be reported.[4]

There are several factors that affect throughput and latency. In terms of throughput, the number of inferences per second is affected by

$$\frac{\text{inferences}}{\text{second}} = \frac{\text{operations}}{\text{second}} \times \frac{1}{\frac{\text{operations}}{\text{inference}}}, \tag{3.1}$$

where the number of *operations per second* is dictated by both the DNN hardware and DNN model, while the number of *operations per inference* is dictated by the DNN model.

When considering a system comprised of multiple processing elements (PEs), where a PE corresponds to a simple or primitive core that performs a single MAC operation, the number of operations per second can be further decomposed as follows:

$$\frac{\text{operations}}{\text{second}} = \underbrace{\left(\frac{1}{\frac{\text{cycles}}{\text{operation}}} \times \frac{\text{cycles}}{\text{second}}\right)}_{\text{for a single PE}} \times \text{number of PEs} \times \text{utilization of PEs}. \tag{3.2}$$

The first term reflects the peak throughput of a single PE, the second term reflects the amount of parallelism, while the last term reflects degradation due to the inability of the architecture to effectively utilize the PEs.

Since the main operation for processing DNNs is a MAC, we will use number of operations and number of MAC operations interchangeably.

One can increase the peak throughput of a single PE by increasing the number of *cycles per second*, which corresponds to a higher clock frequency, by reducing the critical path at the circuit or micro-architectural level, or the number of *cycles per operations*, which can be affected by the design of the MAC (e.g., a non-pipelined multi-cycle MAC would have more cycles per operation).

While the above approaches increase the throughput of a single PE, the overall throughput can be increased by increasing the *number of PEs*, and thus the maximum number of MAC operations that can be performed in parallel. The number of PEs is dictated by the area density of the PE and the area cost of the system. If the area cost of the system is fixed, then increasing the number of PEs requires either increasing the area

[4]The phenomenon described here can also be understood using Little's Law [117] from queuing theory, where the relationship between average throughput and average latency are related by the average number of tasks in flight, as defined by

$$\overline{\text{throughput}} = \frac{\overline{\text{tasks-in-flight}}}{\overline{\text{latency}}}.$$

A DNN-centric version of Little's Law would have throughput measured in inferences per second, latency measured in seconds, and inferences-in-flight, as the tasks-in-flight equivalent, measured in the number of images in a batch being processed simultaneously. This helps to explain why increasing the number of inferences in flight to increase throughput may be counterproductive because some techniques that increase the number of inferences in flight (e.g., batching) also increase latency.

density of the PE (i.e., reduce the area per PE) or trading off on-chip storage area for more PEs. Reducing on-chip storage, however, can affect the utilization of the PEs, which we will discuss next.

Increasing the density of PEs can also be achieved by reducing the logic associated with delivering operands to a MAC. This can be achieved by controlling multiple MACs with a single piece of logic. This is analogous to the situation in instruction-based systems such as CPUs and GPUs that reduce instruction bookkeeping overhead by using large aggregate instructions (e.g., single-instruction, multiple-data (SIMD) / Vector Instructions; single-instruction, multiple-threads (SIMT) / Tensor Instructions), where a single instruction can be used to initiate multiple operations.

The number of PEs and the peak throughput of a single PE only indicate the theoretical maximum throughput (i.e., peak performance) when all PEs are performing computation (100% utilization). In reality, the achievable throughput depends on the actual utilization of those PEs, which is affected by several factors as follows:

$$\text{utilization of PEs} = \frac{\text{number of active PEs}}{\text{number of PEs}} \times \text{utilization of active PEs}. \tag{3.3}$$

The first term reflects the ability to distribute the workload to PEs, while the second term reflects how efficiently those active PEs are processing the workload.

The *number of active PEs* is the number of PEs that receive work; therefore, it is desirable to distribute the workload to as many PEs as possible. The ability to distribute the workload is determined by the flexibility of the architecture, for instance the on-chip network, to support the layer shapes in the DNN model.

Within the constraints of the on-chip network, the *number of active PEs* is also determined by the specific allocation of work to PEs by the mapping process. The mapping process involves the placement and scheduling in space and time of every MAC operation (including the delivery of the appropriate operands) onto the PEs. Mapping can be thought of as a compiler for the DNN hardware. The design of on-chip networks and mappings are discussed in Chapters 5 and 6.

The *utilization of the active PEs* is largely dictated by the timely delivery of work to the PEs such that the active PEs do not become idle while waiting for the data to arrive. This can be affected by the bandwidth and latency of the (on-chip and off-chip) memory and network. The bandwidth requirements can be affected by the amount of data reuse available in the DNN model and the amount of data reuse that can be exploited by the memory hierarchy and dataflow. The dataflow determines the order of operations and where data is stored and reused. The amount of data reuse can also be increased using a larger batch size, which is one of the reasons why increasing batch size can increase throughput. The challenge of data delivery and memory bandwidth are discussed in Chapters 5 and 6. The *utilization of the active PEs* can also be affected by the imbalance of work allocated across PEs, which can occur when exploiting sparsity (i.e., avoiding unnecessary work associated with multiplications by zero); PEs with less work become idle and thus have lower utilization.

There is also an interplay between the number of PEs and the utilization of PEs. For instance, one way to reduce the likelihood that a PE needs to wait for data is to store some data locally near or within the PE. However, this requires increasing the chip area allocated to on-chip storage, which, given a fixed chip area, would reduce the number of PEs. Therefore, a key design consideration is how much area to allocate to compute (which increases the number of PEs) versus on-chip storage (which increases the utilization of

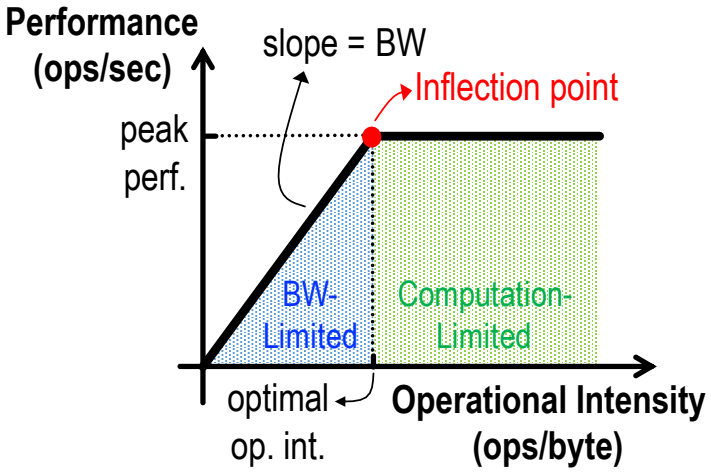


Figure 3.1: The roofline model. The peak *operations per second* is indicated by the bold line; when the operation intensity, which dictates by amount of compute per byte of data, is low, the *operations per second* is limited by the data delivery. The design goal is to operate as close as possible to the peak *operations per second* for the operation intensity of a given workload.

PEs).

The impact of these factors can be captured using Eyexam, which is a systematic way of understanding the performance limits for DNN processors as a function of specific characteristics of the DNN model and accelerator design. Eyexam includes and extends the well-known roofline model [118]. The roofline model, as illustrated in Figure 3.1, relates average bandwidth demand and peak computational ability to performance. Eyexam is described in Chapter 6.

While the number of *operations per inference* in Equation (3.1) depends on the DNN model, the *operations per second* depends on both the DNN model and the hardware. For example, designing DNN models with efficient layer shapes (also referred to efficient network architectures), as described in Chapter 9, can reduce the number of MAC operations in the DNN model and consequently the number of *operations per inference*. However, such DNN models can result in a wide range of layer shapes, some of which may have poor utilization of PEs and therefore reduce the overall *operations per second*, as shown in Equation (3.2).

A deeper consideration of the *operations per second*, is that all operations are not created equal and therefore *cycles per operation* may not be a constant. For example, if we consider the fact that anything multiplied by zero is zero, some MAC operations are ineffectual (i.e., they do not change the accumulated value). The number of ineffectual operations is a function of both the DNN model and the input data. These ineffectual MAC operations can require fewer cycles or no cycles at all. Conversely, we only need to process effectual (or non-zero) MAC operations, where both inputs are non-zero; this is referred to as exploiting sparsity, which is discussed in Chapter 8.

Processing only effectual MAC operations can increase the *(total) operations per second* by increasing the *(total) operations per cycle*.[5] Ideally, the hardware would skip all ineffectual operations; however, in practice, designing hardware to skip all ineffectual operations can be challenging and result in increased

[5]By *total* operations we mean both effectual and ineffectual operations.

hardware complexity and overhead, as discussed in Chapter 8. For instance, it might be easier to design hardware that only recognizes zeros in one of the operands (e.g., weights) rather than both. Therefore, the ineffectual operations can be further divided into those that are exploited by the hardware (i.e., skipped) and those that are unexploited by the hardware (i.e., not skipped). The number of operations actually performed by the hardware is therefore *effectual operations plus unexploited ineffectual operations*.

Equation (3.4) shows how *operations per cycle* can be decomposed into

1. the number of *effectual operations plus unexploited ineffectual operations per cycle*, which remains somewhat constant for a given hardware accelerator design;
2. the ratio of *effectual operations* over *effectual operations plus unexploited ineffectual operations*, which refers to the ability of the hardware to exploit ineffectual operations (ideally unexploited ineffectual operations should be zero, and this ratio should be one); and
3. the number of *effectual operations out of (total) operations*, which is related to the amount of sparsity and depends on the DNN model.

As the amount of sparsity increases (i.e., the number of *effectual operations out of (total) operations* decreases), the *operations per cycle* increases, which subsequently increases *operations per second*, as shown in Equation (3.2):

$$\begin{aligned}\frac{\text{operations}}{\text{cycle}} = &\frac{\text{effectual operations + unexploited ineffectual operations}}{\text{cycle}} \\ &\times \frac{\text{effectual operations}}{\text{effectual operations + unexploited ineffectual operations}} \times \frac{1}{\frac{\text{effectual operations}}{\text{operations}}}.\end{aligned} \tag{3.4}$$

However, exploiting sparsity requires additional hardware to identify when inputs are zero to avoid performing unnecessary MAC operations. The additional hardware can increase the critical path, which decreases cycles per second, and also reduce area density of the PE, which reduces the number of PEs for a given area. Both of these factors can reduce the *operations per second*, as shown in Equation (3.2). Therefore, the complexity of the additional hardware can result in a trade off between reducing the number of *unexploited ineffectual operations* and increasing critical path or reducing the number of PEs.

Finally, designing hardware and DNN models that support reduced precision (i.e., fewer bits per operand and per operations), which is discussed in Chapter 7, can also increase the number of *operations per second*. Fewer bits per operand means that the memory bandwidth required to support a given operation is reduced, which can increase the utilization of PEs since they are less likely to be starved for data. In addition, the area of each PE can be reduced, which can increase the number of PEs for a given area. Both of these factors can increase the *operations per second*, as shown in Equation (3.2). Note, however, that if *multiple* levels of precision need to be supported, additional hardware is required, which can, once again, increase the critical path and also reduce area density of the PE, both of which can reduce the *operations per second*, as shown in Equation (3.2).

In this section, we discussed multiple factors that affect the number of inferences per second. Table 3.1 classifies whether the factors are dictated by the hardware, by the DNN model or both.

Table 3.1: Classification of factors that affect inferences per second.

Factor	Hardware	DNN Model	Input Data
operations per inference		✓	
operations per cycle	✓		
cycles per second	✓		
number of PEs	✓		
number of active PEs	✓	✓	
utilization of active PEs	✓	✓	
effectual operations out of (total) operations		✓	✓
effectual operations plus unexploited ineffectual operations per cycle	✓		

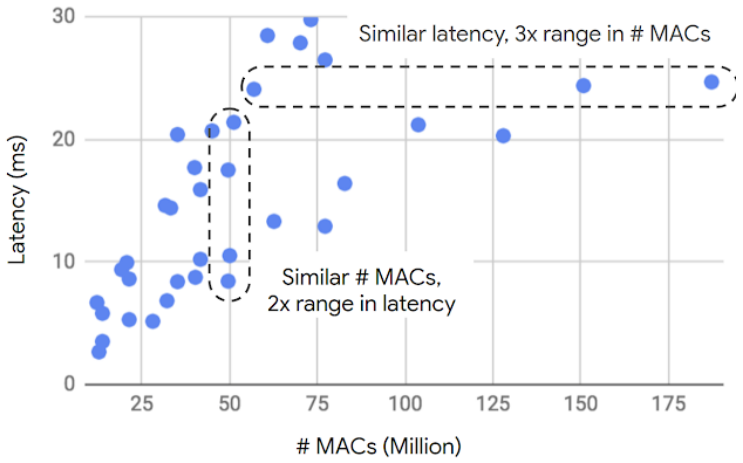


Figure 3.2: The number of MAC operations in various DNN models versus latency measured on Pixel phone. Clearly, the number of MAC operations is not a good predictor of latency. (Figure from [119].)

In summary, the number of MAC operations in the DNN model alone is not sufficient for evaluating the throughput and latency. While the DNN model can affect the number of MAC operations per inference based on the network architecture (i.e., layer shapes) and the sparsity of the weights and activations, the overall impact that the DNN model has on throughput and latency depends on the ability of the hardware to add support to recognize these approaches without significantly reducing utilization of PEs, number of PEs, or cycles per second. This is why the number of MAC operations is not necessarily a good proxy for throughput and latency (e.g., Figure 3.2), and it is often more effective to design efficient DNN models with hardware in the loop. Techniques for designing DNN models with hardware in the loop are discussed in Chapter 9.

Similarly, the number of PEs in the hardware and their peak throughput are not sufficient for evaluating the throughput and latency. It is critical to report actual runtime of the DNN models on hardware to account for other effects such as utilization of PEs, as highlighted in Equation (3.2). Ideally, this evaluation should be performed on clearly specified DNN models, for instance those that are part of the MLPerf benchmarking suite. In addition, batch size should be reported in conjunction with the throughput in order to evaluate latency.

3.3 Energy Efficiency and Power Consumption

Energy efficiency is used to indicate the amount of data that can be processed or the number of executions of a task that can be completed for a given unit of energy. High energy efficiency is important when processing DNNs at the edge in embedded devices with limited battery capacity (e.g., smartphones, smart sensors, robots, and wearables). Edge processing may be preferred over the cloud for certain applications due to latency, privacy, or communication bandwidth limitations. Energy efficiency is often generically reported as the number of operations per joule. In the case of inference, energy efficiency is reported as inferences per joule or the inverse as energy consumption in terms of joules per inference.

Power consumption is used to indicate the amount of energy consumed per unit time. Increased power consumption results in increased heat dissipation; accordingly, the maximum power consumption is dictated by a design criterion typically called the thermal design power (TDP), which is the power that the cooling system is designed to dissipate. Power consumption is important when processing DNNs in the cloud as data centers have stringent power ceilings due to cooling costs; similarly, handheld and wearable devices also have tight power constraints since the user is often quite sensitive to heat and the form factor of the device limits the cooling mechanisms (e.g., no fans). Power consumption is typically reported in watts or joules per second.

Power consumption in conjunction with energy efficiency limits the throughput as follows:

$$\frac{\text{inferences}}{\text{second}} \leq \text{Max}\left(\frac{\text{joules}}{\text{second}}\right) \times \frac{\text{inferences}}{\text{joule}}. \tag{3.5}$$

Therefore, if we can improve energy efficiency by increasing the number of *inferences per joule*, we can increase the number of *inferences per second* and thus throughput of the system.

There are several factors that affect the energy efficiency. The number of inferences per joule can be decomposed into

$$\frac{\text{inferences}}{\text{joule}} = \frac{\text{operations}}{\text{joule}} \times \frac{1}{\frac{\text{operations}}{\text{inference}}}, \tag{3.6}$$

where the number of operations per joule is dictated by both the hardware and DNN model, while the number of operations per inference is dictated by the DNN model.

There are various design considerations for the hardware that will affect the energy per operation (i.e., joules per operation). The energy per operation can be broken down into the energy required to move the input and output data, and the energy required to perform the MAC computation

$$\text{Energy}_{total} = \text{Energy}_{data} + \text{Energy}_{MAC}. \tag{3.7}$$

For each component the joules per operation[6] is computed as

[6]Here, an operation can be a MAC operation or a data movement.

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Multiply	0.2
32b Multiply	3.1
16b FP Multiply	1.1
32b FP Multiply	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

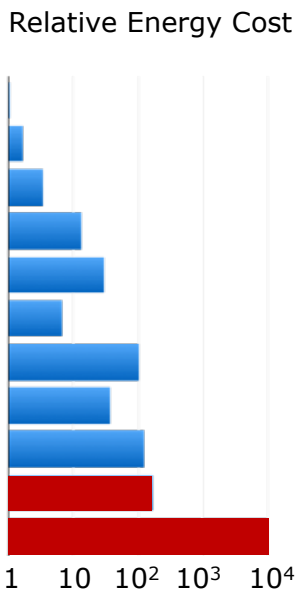


Figure 3.3: The energy consumption for various arithmetic operations and memory accesses in a 45 nm process. The relative energy cost (computed relative to the 8b add) is shown on a log scale. The energy consumption of data movement (red) is significantly higher than arithmetic operations (blue). (Figure adapted from [120].)

$$\frac{\text{joules}}{\text{operation}} = \alpha \times C \times {V_{DD}}^2, \tag{3.8}$$

where C is the total switching capacitance, V_{DD} is the supply voltage, and α is the switching activity, which indicates how often the capacitance is charged.

The energy consumption is dominated by the data movement as the capacitance of data movement tends to be much higher that the capacitance for arithmetic operations such as a MAC (Figure 3.3). Furthermore, the switching capacitance increases the further the data needs to travel to reach the PE, which consists of the distance to get out of the memory where the data is stored and the distance to cross the network between the memory and the PE. Accordingly, larger memories and longer interconnects (e.g., off-chip) tend to consume more energy than smaller and closer memories due to the capacitance of the long wires employed. In order to reduce the energy consumption of data movement, we can exploit data reuse where the data is moved once from distant large memory (e.g., off-chip DRAM) and reused for multiple operations from a local smaller memory (e.g., on-chip buffer or scratchpad within the PE). Optimizing data movement is a major consideration in the design of DNN accelerators; the design of the dataflow, which defines the processing order, to increase data reuse within the memory hierarchy is discussed in Chapter 5. In addition, advanced device and memory technologies can be used to reduce the switching capacitance between compute and memory, as described in Chapter 10.

This raises the issue of the appropriate scope over which energy efficiency and power consumption should be reported. Including the entire system (out to the fans and power supplies) is beyond the scope of this book. Conversely, ignoring off-chip memory accesses, which can vary greatly between chip designs, can easily result in a misleading perception of the efficiency of the system. Therefore, it is critical to not only report the

energy efficiency and power consumption of the chip, but also the energy efficiency and power consumption of the off-chip memory (e.g., DRAM) or the amount of off-chip accesses (e.g., DRAM accesses) if no specific memory technology is specified; for the latter, it can be reported in terms of the total amount of data that is read and written off-chip per inference.

Reducing the joules per MAC operation itself can be achieved by reducing the switching activity and/or capacitance at a circuit level or micro-architecture level. This can also be achieved by reducing precision (e.g., reducing the bit width of the MAC operation), as shown in Figure 3.3 and discussed in Chapter 7. Note that the impact of reducing precision on accuracy must also be considered.

For instruction-based systems such as CPUs and GPUs, this can also be achieved by reducing instruction bookkeeping overhead. For example, using large aggregate instructions (e.g., single-instruction, multiple-data (SIMD) / Vector Instructions; single-instruction, multiple-threads (SIMT) / Tensor Instructions), a single instruction can be used to initiate multiple operations.

Similar to the throughput metric discussed in Section 3.2, the number of *operations per inference* depends on the DNN model, however the *operations per joules* may be a function of the ability of the hardware to exploit sparsity to avoid performing ineffectual MAC operations. Equation (3.9) shows how *operations per joule* can be decomposed into:

1. the number of *effectual operations plus unexploited ineffectual operations per joule*, which remains somewhat constant for a given hardware architecture design;
2. the ratio of *effectual operations* over *effectual operations plus unexploited ineffectual operations*, which refers to the ability of the hardware to exploit ineffectual operations (ideally unexploited ineffectual operations should be zero, and this ratio should be one); and
3. the number of *effectual operations out of (total) operations*, which is related to the amount of sparsity and depends on the DNN model.

$$\begin{aligned}\frac{\text{operations}}{\text{joule}} = &\frac{\text{effectual operations + unexploited ineffectual operations}}{\text{joule}} \\ &\times \frac{\text{effectual operations}}{\text{effectual operations + unexploited ineffectual operations}} \\ &\times \frac{1}{\frac{\text{effectual operations}}{\text{operations}}}.\end{aligned} \tag{3.9}$$

For hardware that can exploit sparsity, increasing the amount of sparsity (i.e., decreasing the number of *effectual operations out of (total) operations*) can increase the number of *operations per joule*, which subsequently increases *inferences per joule*, as shown in Equation (3.6). While exploiting sparsity has the potential of increasing the number of *(total) operations per joule*, the additional hardware will decrease the *effectual operations plus unexploited ineffectual operations per joule*. In order to achieve a net benefit, the decrease in *effectual operations plus unexploited ineffectual operations per joule* must be more than offset by the decrease of *effectual operations out of (total) operations*.

In summary, we want to emphasize that the number of MAC operations and weights in the DNN model are not sufficient for evaluating energy efficiency. From an energy perspective, all MAC operations or weights are not created equal. This is because the number of MAC operations and weights do not reflect where the data is accessed and how much the data is reused, both of which have a significant impact on the *operations per joule*. Therefore, the number of MAC operations and weights is not necessarily a good proxy for energy consumption and it is often more effective to design efficient DNN models with hardware in the loop. Techniques for designing DNN models with hardware in the loop are discussed in Chapter 9.

In order to evaluate the energy efficiency and power consumption of the entire system, it is critical to not only report the energy efficiency and power consumption of the chip, but also the energy efficiency and power consumption of the off-chip memory (e.g., DRAM) or the amount of off-chip accesses (e.g., DRAM accesses) if no specific memory technology is specified; for the latter, it can be reported in terms of the total amount of data that is read and written off-chip per inference. As with throughput and latency, the evaluation should be performed on clearly specified, ideally widely used, DNN models.

3.4 Hardware Cost

In order to evaluate the desirability of a given architecture or technique, it is also important to consider the *hardware cost* of the design. Hardware cost is used to indicate the monetary cost to build a system.[7] This is important from both an industry and a research perspective to dictate whether a system is financially viable. From an industry perspective, the cost constraints are related to volume and market; for instance, embedded processors have a much more stringent cost limitations than processors in the cloud.

One of the key factors that affect cost is the chip area (e.g., square millimeters, mm^2) in conjunction with the process technology (e.g., 45 nm CMOS), which constrains the amount of on-chip storage and amount of compute (e.g., the number of PEs for custom DNN accelerators, the number of cores for CPUs and GPUs, the number of digital signal processing (DSP) engines for FPGAs, etc.). To report information related to area, without specifying a specific process technology, the amount of on-chip memory (e.g, storage capacity of the global buffer) and compute (e.g., number of PEs) can be used as a proxy for area.

Another important factor is the amount of off-chip bandwidth, which dictates the cost and complexity of the packaging and printed circuit board (PCB) design (e.g., High Bandwidth Memory (HBM) [121] to connect to off-chip DRAM, NVLink to connect to other GPUs, etc.), as well as whether additional chip area is required for a transceiver to handle signal integrity at high speeds. The off-chip bandwidth, which is typically reported in gigabits per second (Gbps), sometimes including the number of I/O ports, can be used as a proxy for packaging and PCB cost.

There is also an interplay between the costs attributable to the chip area and off-chip bandwidth. For instance, increasing on-chip storage, which increases chip area, can reduce off-chip bandwidth. Accordingly, both metrics should be reported in order to provide perspective on the total cost of the system.

[7]There is also cost associated with operating a system, such as the electricity bill and the cooling cost, which are primarily dictated by the energy efficiency and power consumption, respectively. There is also cost associated with designing the system. The operating cost is covered by the section on energy efficiency and power consumption and we limited our coverage of design cost to the fact that custom DNN accelerators have a higher design cost than off-the-shelf CPUs and GPUs. We consider anything beyond this, e.g., the economics of the semiconductor business, including how to price platforms, is outside the scope of this book.

Of course reducing cost alone is not the only objective. The design objective is invariably to maximize the throughput or energy efficiency for a given cost, specifically, to maximize *inferences per second per cost* (e.g., $) and/or *inferences per joule per cost*. This is closely related to the previously discussed property of utilization; to be cost efficient, the design should aim to utilize every PE to increase inferences per second, since each PE increases the area and thus the cost of the chip; similarly, the design should aim to effectively utilize all the on-chip storage to reduce off-chip bandwidth, or increase operations per off-chip memory access as expressed by the roofline model (see Figure 3.1), as each byte of on-chip memory also increases cost.

3.5 Flexibility

The merit of a DNN accelerator is also a function of its *flexibility*. Flexibility refers to the range of DNN models that can be supported on the DNN processor and the ability of the software environment (e.g., the mapper) to maximally exploit the capabilities of the hardware for any desired DNN model. Given the fast-moving pace of DNN research and deployment, it is increasingly important that DNN processors support a wide range of DNN models and tasks.

We can define *support* in two tiers: the first tier requires that the hardware only needs to be able to *functionally* support different DNN models (i.e., the DNN model can run on the hardware). The second tier requires that the hardware should also *maintain efficiency* (i.e., high throughput and energy efficiency) across different DNN models.

To maintain efficiency, the hardware should not rely on certain properties of the DNN models to achieve efficiency, as the properties cannot be guaranteed. For instance, a DNN accelerator that can efficiently support the case where the entire DNN model (i.e., all the weights) fits on-chip may perform extremely poorly when the DNN model grows larger, which is likely given that the size of DNN models continue to increase over time, as discussed in Section 2.4.1; a more flexible processor would be able to efficiently handle a wide range of DNN models, even those that exceed on-chip memory.

The degree of flexibility provided by a DNN accelerator is a complex trade-off with accelerator cost. Specifically, additional hardware usually needs to be added in order to flexibly support a wider range of workloads and/or improve their throughput and energy efficiency. We all know that specialization improves efficiency; thus, the design objective is to reduce the overhead (e.g., area cost and energy consumption) of supporting flexibility while maintaining efficiency across the wide range of DNN models. Thus, evaluating flexibility would entail ensuring that the extra hardware is a net benefit across multiple workloads.

Flexibility has become increasingly important when we factor in the many techniques that are being applied to the DNN models with the promise to make them more efficient, since they increase the diversity of workloads that need to be supported. These techniques include DNNs with different network architectures (i.e., different layer shapes, which impacts the amount of required storage and compute and the available data reuse that can be exploited), as described in Chapter 9, different levels of precision (i.e., different number of bits for across layers and data types), as described in Chapter 7, and different degrees of sparsity (i.e., number of zeros in the data), as described in Chapter 8. There are also different types of DNN layers and computation beyond MAC operations (e.g., activation functions) that need to be supported.

Actually getting a performance or efficiency benefit from these techniques invariably requires additional

hardware, because a simpler DNN accelerator design may not benefit from these techniques. Again, it is important that the overhead of the additional hardware does not exceed the benefits of these techniques. This encourages a hardware and DNN model *co-design* approach.

To date, exploiting the flexibility of DNN hardware has relied on mapping processes that act like static per-layer compilers. As the field moves to DNN models that change dynamically, mapping processes will need to dynamically adapt at runtime to changes in the DNN model or input data, while still maximally exploiting the flexibility of the hardware to improve efficiency.

In summary, to assess the flexibility of DNN processors, its efficiency (e.g., inferences per second, inferences per joule) should be evaluated on a wide range of DNN models. The MLPerf benchmarking workloads are a good start; however, additional workloads may be needed to represent efficient techniques such as efficient network architectures, reduced precision and sparsity. The workloads should match the desired application. Ideally, since there can be many possible combinations, it would also be beneficial to define the range and limits of DNN models that can be *efficiently* supported on a given platform (e.g., maximum number of weights per filter or DNN model, minimum amount of sparsity, required structure of the sparsity, levels of precision such as 8-bit, 4-bit, 2-bit, or 1-bit, types of layers and activation functions, etc.).

3.6 Scalability

Scalability has become increasingly important due to the wide use cases for DNNs and emerging technologies used for scaling up not just the size of the chip, but also building systems with multiple chips (often referred to as chiplets) [122] or even wafer-scale chips [123]. Scalability refers to how well a design can be scaled up to achieve higher throughput and energy efficiency when increasing the amount of resources (e.g., the number of PEs and on-chip storage). This evaluation is done under the assumption that the system does not have to be significantly redesigned (e.g., the design only needs to be replicated) since major design changes can be expensive in terms of time and cost. Ideally, a scalable design can be used for low-cost embedded devices and high-performance devices in the cloud simply by scaling up the resources.

Ideally, the throughput would scale linearly and proportionally with the number of PEs. Similarly, the energy efficiency would also improve with more on-chip storage, however, this would be likely be nonlinear (e.g., increasing the on-chip storage such that the entire DNN model fits on chip would result in an abrupt improvement in energy efficiency). In practice, this is often challenging due to factors such as the reduced utilization of PEs and the increased cost of data movement due to long distance interconnects.

Scalability can be connected with cost efficiency by considering how *inferences per second per cost* (e.g., $) and *inferences per joule per cost* changes with scale. For instance, if throughput increases linearly with number of PEs, then the *inferences per second per cost* would be constant. It is also possible for the *inferences per second per cost* to improve super-linearly with increasing number of PEs, due to increased sharing of data across PEs.

In summary, to understand the scalability of a DNN accelerator design, it is important to report its performance and efficiency metrics as the number of PEs and storage capacity increases. This may include how well the design might handle technologies used for scaling up, such as inter-chip interconnect.

3.7 Interplay Between Different Metrics

It is important that all metrics are accounted for in order to fairly evaluate all the design trade-offs. For instance, without the accuracy given for a specific dataset and task, one could run a simple DNN and easily claim low power, high throughput, and low cost—however, the processor might not be usable for a meaningful task; alternatively, without reporting the off-chip bandwidth, one could build a processor with only multipliers and easily claim low cost, high throughput, high accuracy, and low *chip* power—however, when evaluating *system* power, the off-chip memory access would be substantial. Finally, the test setup should also be reported, including whether the results are measured or obtained from simulation[8] and how many images were tested.

In summary, the evaluation process for whether a DNN system is a viable solution for a given application might go as follows:

1. the accuracy determines if it can perform the given task;
2. the latency and throughput determine if it can run fast enough and in real time;
3. the energy and power consumption will primarily dictate the form factor of the device where the processing can operate;
4. the cost, which is primarily dictated by the chip area and external memory bandwidth requirements, determines how much one would pay for this solution;
5. flexibility determines the range of tasks it can support; and
6. the scalability determines whether the same design effort can be amortized for deployment in multiple domains, (e.g., in the cloud and at the edge), and if the system can efficiently be scaled with DNN model size.

[8] If obtained from simulation, it should be clarified whether it is from synthesis or post place-and-route and what library corner (e.g., process corner, supply voltage, temperature) was used.

Chapter 10

Advanced Technologies

As highlighted throughout the previous chapters, data movement dominates energy consumption. The energy is consumed both in the access to the memory as well as the transfer of the data. The associated physical factors also limit the bandwidth available to deliver data between memory and compute, and thus limits the throughput of the overall system. This is commonly referred to by computer architects as the "memory wall."[1]

To address the challenges associated with data movement, there have been various efforts to bring compute and memory closer together. Chapters 5 and 6 primarily focus on how to design spatial architectures that distribute the on-chip memory closer to the computation (e.g., scratch pad memory in the PE). This chapter will describe various other architectures that use *advanced memory*, *process*, and *fabrication technologies* to bring the compute and memory together.

First, we will describe efforts to bring the off-chip high-density memory (e.g., DRAM) closer to the computation. These approaches are often referred to as *processing near memory* or *near-data processing*, and include memory technologies such as embedded DRAM and 3-D stacked DRAM.

Next, we will describe efforts to integrate the computation *into* the memory itself. These approaches are often referred to as *processing in memory* or *in-memory computing*, and include memory technologies such as Static Random Access Memories (SRAM), Dynamic Random Access Memories (DRAM), and emerging non-volatile memory (NVM). Since these approaches rely on mixed-signal circuit design to enable processing in the analog domain, we will also discuss the design challenges related to handling the increased sensitivity to circuit and device non-idealities (e.g., nonlinearity, process and temperature variations), as well as the impact on area density, which is critical for memory.

Significant data movement also occurs between the sensor that collects the data and the DNN processor. The same principles that are used to bring compute near the memory, where the weights are stored, can be used to bring the compute *near* the sensor, where the input data is collected. Therefore, we will also discuss how to integrate some of the compute *into* the sensor.

Finally, since photons travel much faster than electrons and the cost of moving a photon can be *independent* of distance, processing in the optical domain using light may provide significant improvements in energy

[1] Specifically, the memory wall refers to data moving between the off-chip memory (e.g., DRAM) and the processor.

Table 10.1: Example of recent works that explore processing near memory. For I/O, TSV refers to through-silicon vias, while TCI refers to ThruChip Interface which uses inductive coupling. For bandwidth, *ch* refers to number of parallel communication channels, which can be the number of tiles (for eDRAM) or the number of vaults (for stacked memory). The size of stacked DRAM is based on Hybrid Memory Cube (HMC) Gen2 specifications.

	Technology	**Size**	**I/O**	**Bandwidth**	**Evaluation**
DaDianNao [151]	eDRAM	32MB	on-chip	18 ch×310 GB/s = 5580 GB/s	Simulated
Neurocube [315]	Stacked DRAM	2GB	TSV	16 ch×10 GB/s = 160 GB/s	Simulated
Tetris [316]	Stacked DRAM	2GB	TSV	16 ch×8 GB/s = 128 GB/s	Simulated
Quest [317]	Stacked SRAM	96MB	TCI	24 ch×1.2 GB/s = 28.8 GB/s	Measured
N3XT [318]	monolithic 3-D	4GB	ILV	16 ch×48 GB/s = 768 GB/S	Simulated

efficiency and throughput over the electrical domain. Accordingly, we will conclude this chapter by discussing the recent work that performs DNN processing in the optical domain, referred to as *Optical Neural Networks*.

10.1 Processing Near Memory

High-density memories typically require a different process technology than processors and as a result are often fabricated as separate chips; as a result, accessing high-density memories requires going off-chip. The bandwidth and energy cost of accessing high-density off-chip memories are often limited by the number of I/O pads per chip and the off-chip interconnect channel characteristics (i.e., its resistance, inductance, and capacitance). Processing near memory aims to overcome these limitations by bringing the compute near the high-density memory to reduce access energy and increase memory bandwidth. The reduction in access energy is achieved by reducing the length of the interconnect between the memory and compute, while the increase in bandwidth is primarily enabled by increasing the number of bits that can be accessed per cycle by allowing for a wider interconnect and, to a lesser extent, by increasing the clock frequency, which is made possible by the reduced interconnect length.

Various recent advanced memory technologies aim to enable processing near memory with differing integration costs. Table 10.1 summarizes some of these efforts, where high-density memories on the order of tens of megabytes to gigabytes are connected to the compute engine at bandwidths of tens to hundreds of gigabytes per second. Note that currently most academic evaluations of DNN systems using advanced memory technologies have been based on simulations rather than fabrication and measurements.

In this section, we will describe the cost and benefits of each technology and provide examples of how they have been used to process DNNs. The architectural design challenges of using processing-near-memory include how to allocate data to memory since the access patterns for high-density memories are often limited (e.g., data needs to be divided into different banks and vaults in the DRAM or stacked DRAM, respectively), how to design the network-on-chip between the memory and PEs, how to allocate the chip area between on-chip memory and compute now that off-chip communication less expensive, and how to design the memory hierarchy and dataflow now that the data movement costs are different.

10.1.1 Embedded High-Density Memories

Accessing data from off-chip memory can result in high energy cost as well as limited memory bandwidth (due to limited data bus width due to number of I/O pads, and signaling frequency due to the channel characteristics of the off-chip routing). Therefore, there has been a significant amount of effort toward embedding high-density memory on-chip. This includes technology such as *embedded DRAM (eDRAM)* [319] as well as *embedded non-volatile (eNVM)* [320], which includes embedded Flash (eFlash) [321], magnetic random-access memory (MRAM) [322], resistive random-access memory (RRAM) [323, 324], and phase change memory (PCRAM) [325].

In DNN processing, these high-density memories can be used to store tens of megabytes of weights and activations on chip to reduce off-chip access. For instance, DaDianNao [151] uses 16×2MB eDRAM tiles to store the weights and 2×2MB eDRAM tiles to store the input and output activations; furthermore, all these tiles (each with 4096-bit rows) can be accessed in parallel, which gives extremely high memory bandwidth.[2] The downside of eDRAM is that it has a lower density than off-chip DRAM and can increase the fabrication cost of the chip. In addition, it has been reported that eDRAM scaling is slower than SRAM scaling [326], and thus the density advantage of eDRAM over SRAM will reduce over time. In contrast, eNVMs have gained popularity in recent years due to its increased density as well as its non-volatility properties and reduction in standby power (e.g., leakage, refresh, etc.) compared to eDRAM [326].

10.1.2 Stacked Memory (3-D Memory)

Rather than integrating DRAM into the chip itself, the DRAM can also be stacked on top of the chip using through-silicon vias (TSVs). This technology is often referred to as *3-D memory*,[3] and has been commercialized in the form of Hybrid Memory Cube (HMC) [327] and High Bandwidth Memory (HBM) [121]. 3-D memory delivers an order of magnitude higher bandwidth and reduces access energy by up to 5× relative to existing 2-D DRAMs, as TSVs have lower capacitance than typical off-chip interconnects.

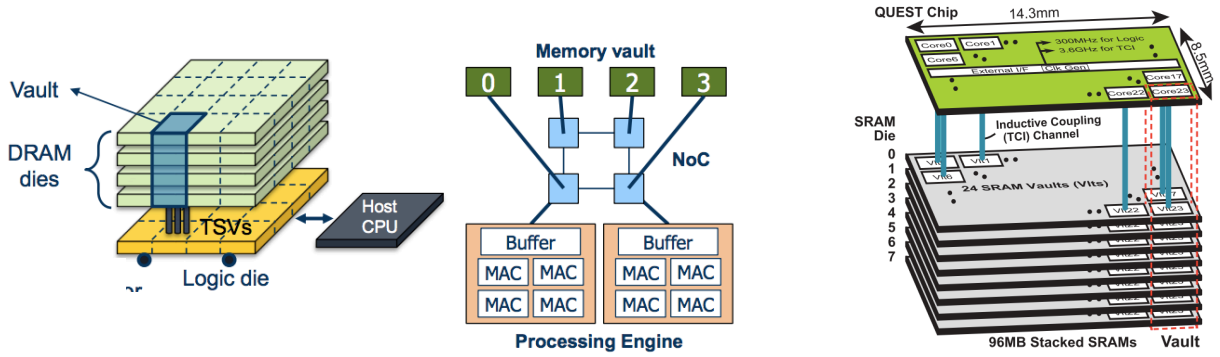
Recent works have explored the use of HMC for efficient DNN processing in a variety of ways. For instance, Neurocube [315], shown in Figure 10.1(a), uses HMC to bring the memory and computation closer together. Each DRAM vault (vertically stacked DRAM banks) is connected to a PE containing a buffer and several MACs. A 2-D mesh network-on-chip (NoC) is used to connect the different PEs, allowing the PEs to access data from different vaults. One major design decision involves determining how to distribute the weights and activations across the different vaults to reduce the traffic on the NoC.

Another example that uses HMC is Tetris [316], which explores the use of HMC with the Eyeriss spatial architecture and row-stationary dataflow. It proposes allocating more area to computation than on-chip memory (i.e., larger PE array and smaller global buffer) in order to exploit the low-energy and high-throughput properties of the HMC. It also adapts the dataflow to account for the HMC and smaller on-chip memory.

SRAM can also be stacked on top of the chip to provide 10× lower latency compared to DRAM [317]. For instance, Quest [317], shown in Figure 10.1(b), uses eight 3-D stacked SRAM dies to store both the weights

[2] DaDianNao [151] assumes that the DNN model can fit into the 32MB of eDRAM allocated to the weights. In practice, this implies that the design either limits the size of DNN model, or requires access to off-chip memory if the size of the DNN model exceeds the capacity of the eDRAM.

[3] Also referred to as "in-package" memory since both the memory and compute can be integrated into the same package.



(a) Neurocube (Figure from [315])

(b) Quest (Figure from [328])

Figure 10.1: Stacked memory systems. (a) DRAM using through-silicon vias (TSV) and (b) SRAM using inductive coupling.

and the activations of the intermediate feature maps when processing layer by layer. The SRAM dies are connected to the chip using inductive-coupling die-to-die wireless communication technology, known as a ThruChip Interface (TCI) [329], which has lower integration cost than TSV.

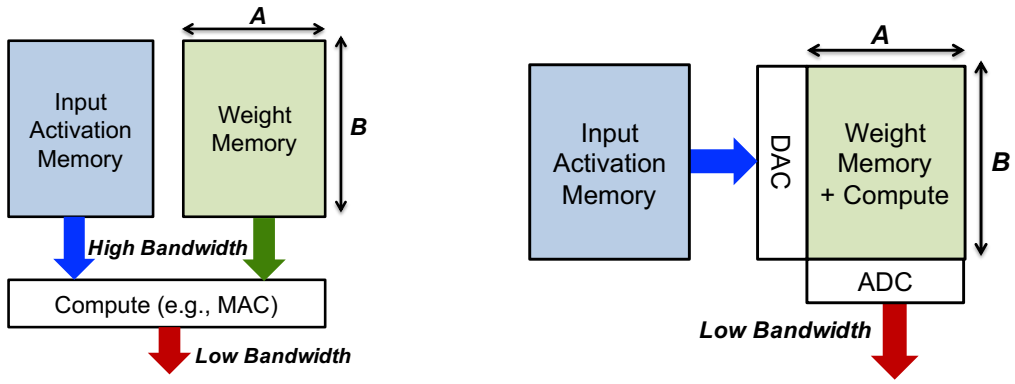
The above 3-D memory designs involve using TSV or TCI to connect memory and logic dies that have been separately fabricated. Recent breakthroughs in nanotechnology have made it feasible to directly fabricate thin layers of logic and memory devices on top of each other, referred to as monolithic 3-D integration. Interlayer vias (ILVs), which have several orders of magnitude denser vertical connectivity than TSV, can then be used to connect the memory and compute. Current monolithic 3-D integration systems, such as N3XT, use on-chip non-volatile memory (e.g., resistive RAM (RRAM), spin-transfer torque RAM (STT-RAM) / magnetic RAM (MRAM), phase change RAM (PCRAM)), and carbon nanotube logic (CNFET). Based on simulations, the energy-delay product of ILVs can be up to two orders of magnitude lower than 2-D systems on deep neural network workloads, compared to 8× for TSV [318].[4]

In order to fully understand the impact of near memory processing it is important to analyze the impact that the added storage layer has on the mappings that are now available. Specifically, the new memories are faster, but also smaller, so optimal mappings will be different.

10.2 Processing in Memory

While the previous section discussed methods to bring the compute near the memory, this section discusses *processing in memory*, which brings the compute *into* the memory. We will first highlight the differences between processing in memory and conventional architectures, then describe how processing in memory can be performed using different memory technologies including NVM, SRAM, and DRAM. Finally, we will highlight various design challenges associated with processing-in-memory accelerators that are commonly found across technologies.

[4]The savings are highest for DNN models and configurations with low amounts of data reuse (e.g., FC layers with small batch size) resulting in more data movement across ILV.



(a) Conventional (b) Processing in memory

Figure 10.2: Comparison of conventional processing and processing in memory.

DNN processing can be performed using matrix-vector multiplication (see Figures 4.2 and 4.3), as discussed in Chapter 4. For conventional architectures, both the input activation vector and the weight matrix are read out from their respective memories and processed by a MAC array, as shown in Figure 10.2(a); the number of weights that can be read at once is limited by the memory interface (e.g., the read out logic and the number of memory ports). This limited memory bandwidth for the weights (e.g., a row of A weights per cycle in Figure 10.2(b)) can also limit the number of MAC operations that can be performed in parallel (i.e., operations per cycle) and thus the overall throughput (i.e., operations per second).

Processing-in-memory architectures propose moving the compute into the memory that stores the weight matrix, as shown in Figure 10.2(b). This can help reduce the data movement of the weights by avoiding the cost of reading the weight matrix; rather than reading the weights, only the computed results such as the partial sums or the final output activations are read out of the memory. Furthermore, processing in memory architectures can also increase the memory bandwidth, as the number of weights that can be accessed in parallel is no longer limited by the memory interface; in theory, the entire weight matrix (e.g., $A \times B$ in Figure 10.2(b)) could be read and processed in parallel.

Figure 10.3 shows a weight-stationary dataflow architecture that is typically used for processing in memory. The word lines (WLs) are used to deliver the input activations to the storage elements, and the bit lines (BLs) are used to read the computed output activations or partial sums. The MAC array is implemented using the storage elements (that store the weights), where a multiplication is performed at each storage element, and the accumulation across multiple storage elements on the same column is performed using the bit line. In theory, a MAC array of B rows of A elements can access all $A \times B$ weights at the same time, and perform up to A dot products in parallel, where each sums B elements (i.e., $A \times B$ MAC operations per cycle).

Similar to other weight-stationary architectures, the input activations can be reused across the different columns (up to A times for the example given in Figure 10.3), which reduces number of input activation reads. In addition, since a storage element tends to be smaller in area than the logic for a digital MAC (10 to 100× smaller in area and 3 to 10× smaller in edge length [330]), the routing capacitance to deliver the input activations can also be reduced, which further reduces the energy cost of delivering the input activations. Depending on the format of the inputs and outputs to the array, digital-to-analog converters (DACs) and analog-to-digital converters (ADCs) may also be required to convert the word line and bit line values, respectively; the cost of the DAC scales with the precision of the input activations driven on the word

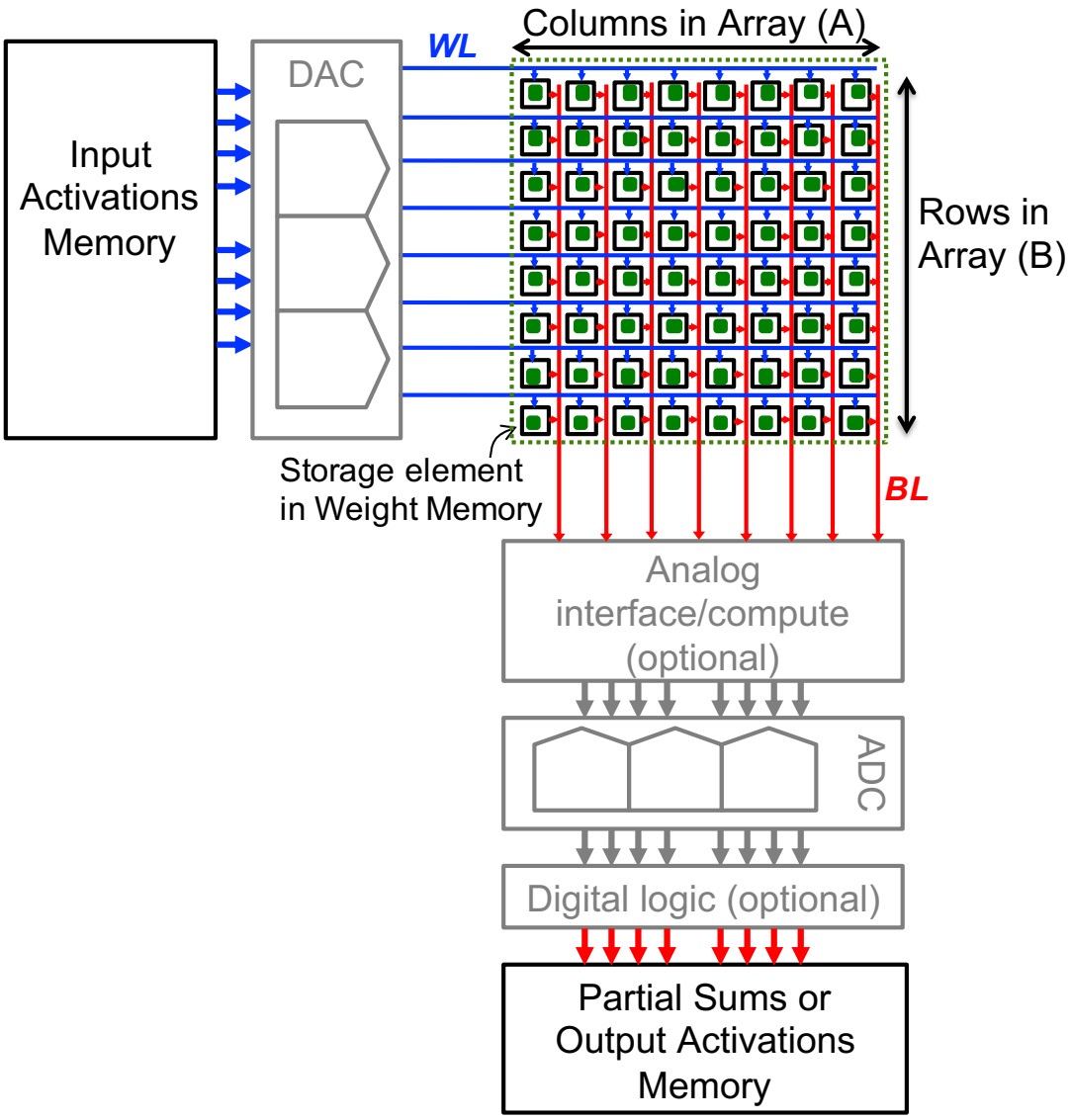


Figure 10.3: Typical dataflow for processing-in-memory accelerators.

line, while the cost of the ADC scales with the precision of the partial sums, which depends on the precision of the weights and input activations, and the number of values accumulated on the bit line (up to B).[5]

An alternative way to view processing in memory is to use the loop nest representation introduced in Chapter 5. Design 20 illustrates a processing-in-memory design for an FC layer with M output channels and where the input activations are flattened along the input channel, height and width dimensions (CHW). The computation take place in one cycle computing all the results in a single cycle in line 7. For this design, some of the mapping constraints are that $A \geq M$ and $B \geq C \times H \times W$.[6] Note, that when $A \neq M$ or $B \neq C \times H \times W$ under-utilization will occur, as described in Section 10.2.4.

Design 20 FC layer for Processing in Memory

```
1    i = Array(CHW)          # Input activations
2    f = Array(M, CHW)       # Filter weights
3    o = Array(M)            # Output partial sums
4
5    parallel-for m in [0, M):
6        parallel-for chw in [0, CHW):
7                o[m] += i[chw] * f[m, chw]
```

A processing in memory design can also handle convolutions, as illustrated in the loop nest in Design 21. Here, we show a toy design of just a 1-D convolution with multiple input channels (C) and multiple output channels (M). The entire computation takes Q steps as the only temporal step is the **for** loop (line 8). Interpreting the activity in the body of the loop (line 10), we see that in each cycle all filter weights are used ($M \times S \times C$) each as part a distinct MAC operation, the same input activation is used multiple times ($C \times S$) and multiple output partial sums are accumulated into (M). This design reflects the Toeplitz expansion of the input activations (see Section 4.1), so the same input activations will be delivered multiple times, since the same value for the input activation index w will be generated for different qs. For the processing in memory convolution design, some of the mapping constraints are that $A \geq M$ and $B \geq C \times S$. Note, that when $A \neq M$ or $B \neq C \times S$ under-utilization will occur, as described in Section 10.2.4.

In the next few sections (Sections 10.2.1, 10.2.2, and 10.2.3), we will briefly describe how processing in memory can be performed using different memory technologies. Section 10.2.4 will then give an overview of some of the key design challenges and decisions that should be considered when designing processing-in-memory accelerators for DNNs. For instance, many of these designs are limited to reduced precision (i.e., low bit-width) due to the non-idealities of the devices and circuits used for memories.

[5]The number of bits that an ADC can correctly resolve also depends on its thermal noise (typically some multiple of kT/C, where k is the Boltzmann constant, T is the temperature, and C is the capacitance of the sampling capacitor). For instance, an N-bit ADC has 2^{N-1} decision boundaries (see Section 7.2.1). However, if the thermal noise is large, the location of the 2^{N-1} decision boundaries will move around, dynamically and randomly, and this will affect the resulting accuracy of the DNN being processed. Therefore, designing a low noise ADC is an important consideration. Note that the thermal noise of the ADC scales with the power consumption and the area of the ADC. Accordingly, it is important that the ADC's thermal noise be considered when evaluating the accuracy as demonstrated in [331, 332, 333], as the design of the ADC involves a trade-off between power, area, and accuracy.

[6]For this example, we disallow the cases where $A < M$ or $B < C \times H \times W$, since that would require multiple passes and updates of the weights, which reduces the potential benefits of processing in memory.

Design 21 1-D Weight-Stationary Convolution Dataflow for Processing in Memory

```
i = Array(C, W)          # Input activations
f = Array(M, C, S)       # Filter weights
o = Array(M, Q)          # Output partial sums

parallel-for m in [0, M):
    parallel-for s in [0, S):
        parallel-for c in [0, C]:
            for q in [0, Q):
                w = q + s
                o[m, q] += i[c, w] * f[m, c, s]
```

10.2.1 Non-Volatile Memories (NVM)

Many recent works have explored enabling processing-in-memory using *non-volatile memories (NVM)* due to their high density and thus potential for replacing off-chip memory and reducing off-chip data movement. Advanced non-volatile high-density memories use programmable resistive elements, commonly referred to as *memristors* [334], as storage elements. These NVM devices enable increased density since memory and computation can be densely packed with a similar density to DRAM [335].[7]

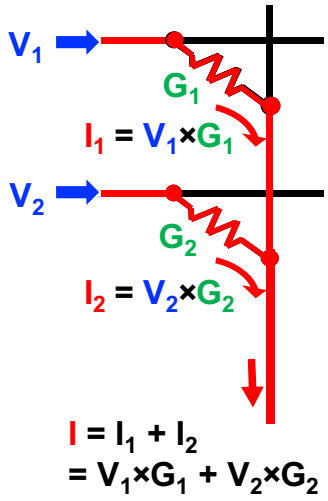
Non-volatile memories exploit Ohm's law by using the conductance (i.e., the inverse of the resistance) of a device to represent a filter weight and the voltage across the device to represent the input activation value. So the resulting current can be interpreted as the product (i.e., a partial sum). This is referred to as a *current-based* approach. For instance, Figure 10.4(a) shows how a multiplication can be performed using the conductance of the NVM device as the weight, and the voltage on the word line as the input activation, and the current output to the bit line as the product of the two. The accumulation is done by summing the currents on the bit line based on Kirchhoff's current law. Alternatively, for Flash-based NVM, the multiplication is performed using the current-voltage (IV) characteristics of the floating-gate transistor, where the threshold voltage of the floating-gate transistor is set based on the weight, as shown in Figure 10.4(c). Similar to the previously described approaches, a voltage proportional to the input activation can be applied across the device, and the accumulation is performed by summing output current of the devices on the bit line.

NVM-based processing-in-memory accelerators have several unique challenges, as described in [339, 340]. First, the cost of programming the memristors (i.e., writing to non-volatile memory) can be much higher than SRAM or DRAM; thus, typical solutions in this space require that the non-volatile memory to be sufficiently large to hold *all* weights of the DNN model, rather than changing the weights in the memory for each layer or filter during processing.[8] As discussed in Chapter 3, this may reduce flexibility as it can limit the size of the DNN model that the accelerator can support.

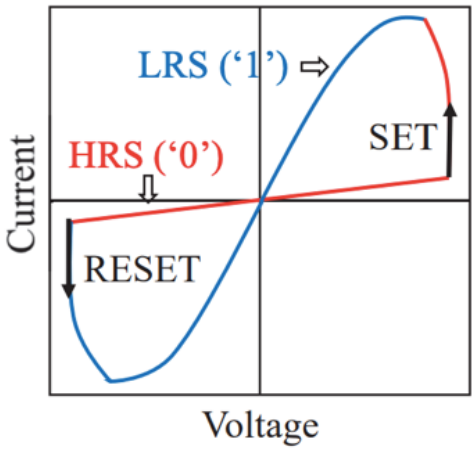
Second, the NVM devices can also suffer from device-to-device and cycle-to-cycle variations with nonlinear conductance across the conductance range [339, 340, 341]. This affects the number of bits that can be stored per device (typically 1 to 4) and the type of signaling used for the input and output activations. For instance,

[7] To improve density, the resistive devices can be inserted between the cross-point of two wires and in certain cases can avoid the need for an access transistor [336]. Under this scenario, the device is commonly referred to as a cross-point element.

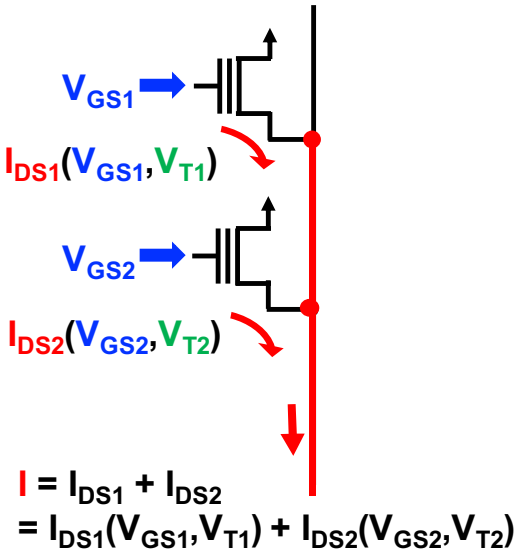
[8] This design choice to hold all weights of the DNN is similar to the approach taken in some of the FPGA designs such as Brainwave [208] and FINN [225], where the weights are pinned on the on-chip memory of the FPGA during synthesis.



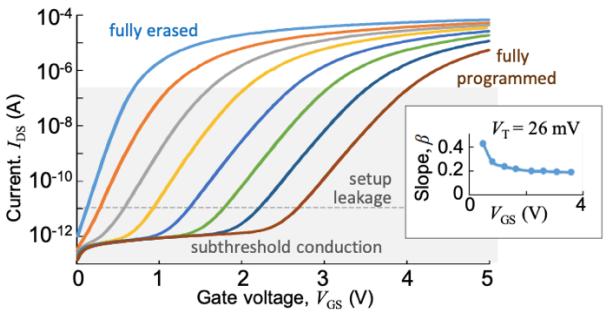
(a) MAC operation using *resistive* NVM devices.



(b) Example I-V curve of a resistive NVM device. (Figure from [337])



(c) MAC operation using *floating gate* NVM devices.



(d) I-V curve of floating gate device.

Figure 10.4: Performing a multiplication and accumulation using the storage element. Input activation is encoded as a voltage amplitude (V_i). (a) For memristors, G_i is the conductance (i.e., 1/resistance) of a resistive device set according to the weight, and bit line current I is the accumulated partial sum value [328]. (b) The current-voltage (I-V) characteristics of the resistive device. The slope of the curve is inversely proportional to the resistance (recall $R = V/I$). Typically, the device can take on just two states: LRS is the low resistive state (also referred to as R_{ON}) and HRS is the high resistive state (also referred to as R_{OFF}).(c) and (d) For floating-gate transistors, the multiplication is performed using its current-voltage (I-V) characteristics, where the weight sets the threshold voltage (as illustrated by the different color lines representing different threshold voltages), and bit line current I is the accumulated partial sum value [338].

rather than encoding the input activation in terms of voltage amplitude, the input can also be encoded in time using pulse width modulation with a *fixed* voltage (i.e., a unary coding), and the resulting current can be accumulated over time on a capacitor to generate the output voltage [342].

Finally, the NVM devices cannot have negative resistance, which presents a challenge for supporting negative weights. One approach is to represent signed weights using differential signaling that requires two storage elements per weight; accordingly, the weights are often stored using two separate arrays [343]. Another approach is to avoid using signed weights. For instance, in the case of binary weights, rather than representing the weights as $[-1, 1]$ and performing binary multiplications, the weights can be represented as $[0, 1]$ and perform XNOR logic operations, as discussed in Chapter 7, or NAND logic operations, as discussed in [344].

There are several popular candidates for NVM devices including phase change RAM (PCRAM), resistive RAM (RRAM or ReRAM), conductive bridge RAM (CBRAM), and spin transfer torque magnetic RAM (STT-MRAM) [345]. These devices have different trade-offs in terms of endurance (i.e., how many times it can be written), retention time (i.e., how often it needs to be refreshed and thus how frequently it needs to be written), write current (i.e., how much power is required to perform a write), area density (i.e., cell size), variations, and speed. An in-depth discussion of how these device properties affect the performance of DNN processing can be found in [340]; Gokmen et al. [342] flips the problem and describes how these devices should be designed such that they can be better suited for DNN processing.[9]

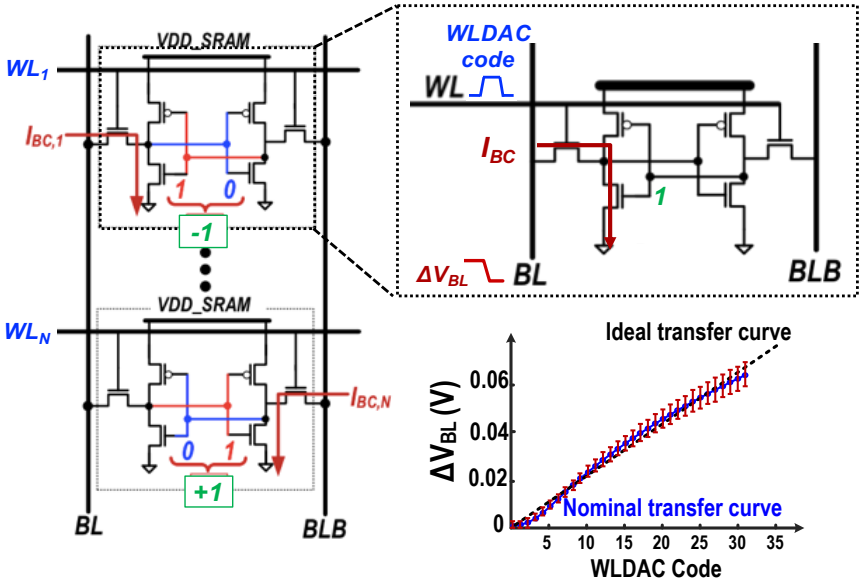
Recent works on NVM-based processing-in-memory accelerators have reported results from both simulation [328, 337, 346, 347] as well as fabricated test chips [348, 343]. While works based on simulation demonstrate functionality on large DNN models such as variants of VGGNet [72] for image classification on ImageNet, works based on fabricated test chips still demonstrate functionality on simple DNN models for digit classification on MNIST [348, 343]. Simulations often project capabilities beyond the current state-of-the-art. For instance, while works based on simulation often assume that all 128 or 256 rows can be activated at the same time, works based on fabricated test chips only activate up to 32 rows at once to account for process variations and limitations in the read out circuits (e.g., ADC); these limitations will be discussed more in Section 10.2.4. It should also be noted that fabricated test chips typically only use one bit per memristor [348, 343, 349].

10.2.2 Static Random Access Memories (SRAM)

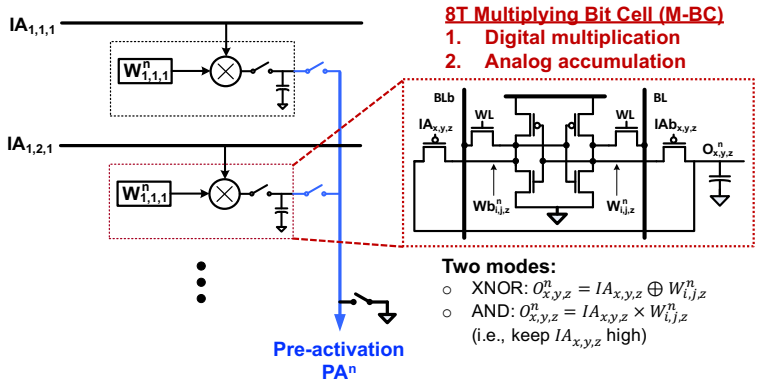
Many recent works have explored the use of the SRAM bit cell to perform computation. They can be loosely classified into current-based and charge-based designs.

Current-based designs use the current-voltage (IV) characteristics of the bit cell to perform a multiplication, which is similar to the NVM current-based approach described in Section 10.2.1. For instance, Figure 10.5(a) shows how the input activation can be encoded as a voltage amplitude on the word line that controls the current through the pull-down network of a bit cell (I_{BC}) resulting in a voltage drop (V_{BL}) proportional to the word line voltage [350]. The current from multiple bit cells (across different rows on the same column) add together on the bit line to perform the accumulation [350]. The resulting voltage drop on the bit line is then proportional to the dot product of the weights and activations of the column.

[9] [340, 342] also describe how these devices might be used for training DNNs if the weights can be updated in parallel and in place within the memristor array.



(a) Multiplication using a 6T SRAM bit-cell and accumulation by current summing on bit lines. (Figure from [350])



(b) Multiplication using a 8T SRAM bit-cell and a local capacitor and accumulation using charge sharing across local capacitors. (Figure from [351])

Figure 10.5: Performing a multiplication and accumulation using the storage element. (a) Multiplication can be performed using a SRAM bit-cell by encoding the input activation as a voltage amplitude on the word line that controls the current through the pull-down network of the bit cell (I_{BC}) resulting in a voltage drop (V_{BL}) proportional to the word line voltage. If a zero (weight value of -1) is stored in the bit cell, the voltage drop occurs on BL, while if a one (weight value of $+1$) is stored the voltage drop occurs on BLB. The current from multiple bit-cells within a column add together. (b) Binary multiplication (XNOR) is performed by connection transistors and local capacitor. Accumulation is performed by charge sharing across local capacitors in bit-cells from the same column.

The above current-based approach is susceptible to the variability and nonlinearity of the word line voltage-to-current relationship of the pull-down network in the bit cell; this create challenges in representing the weights precisely. Charge-based approaches avoid this by using *charge sharing* for the multiplication, where the computation based on the capacitance ratio between capacitors, which tends to be more linear and less sensitive to variations.

Figure 10.5(b) shows how a binary multiplication (i.e., XNOR) via charge sharing can be performed by conditionally charging up a local capacitor within a bit cell, based on the XNOR between the weight value stored in the bit cell and the input activation value that determines the word line voltage [351]. Accumulation can then be performed using charge sharing across the local capacitors of the bit cells on a bit line [351]. Other variants of this approach include performing the multiplication directly with the bit line [352], and charge sharing across different bit lines to perform the accumulation [352, 353, 354].

One particular challenge that exists for SRAM-based processing-in-memory accelerators is maintaining bit cell stability. Specifically, the voltage swing on the bit line typically needs to be kept low in order to avoid a read disturb (i.e., accidentally flipping the value stored in the bit cell when reading). This limits the voltage swing on the bit line, which affects the number of bits that can be accumulated on the bit line for the partial sum; conventional SRAMs only resolve one bit on the bit line. One way to address this is by adding extra transistors to isolate the storage node in the bit cell from the large swing of the bit line [352]; however, this would increase the bit cell area and consequently reduce the overall area density.

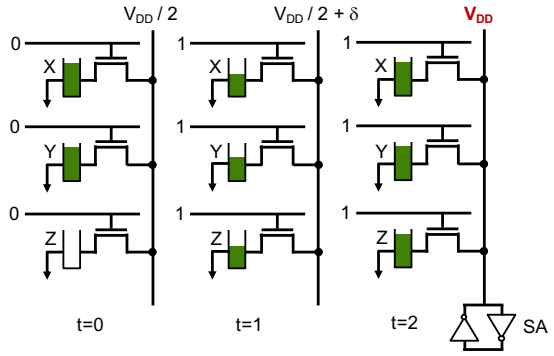
Recent works on SRAM-based processing-in-memory accelerators have reported results from fabricated test chips [350, 351, 352, 353, 354]. In these works, they demonstrate functionality on simple DNN models for digit classification on MNIST, often using layer-by-layer processing, where the weights are updated in the SRAM for each layer. Note that in these works, the layer shapes of the DNN model are often custom designed to fit the array size of the SRAM to increase utilization; this may pose a challenge in terms of flexibility, as discussed in Chapter 3.[10]

10.2.3 Dynamic Random Access Memories (DRAM)

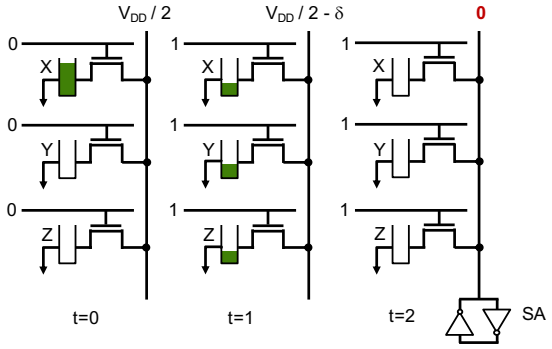
Recent works have explored how processing in memory may be feasible using DRAM by performing bit-wise logic operations when reading multiple bit cells. For instance, Figure 10.6 shows how AND and OR operations can be performed by accessing three rows in parallel [355]. When three rows are accessed at the same time, the output bit line voltage will depend on the average of the charge stored in the capacitors of the bit cells in three rows (note that the charge stored in capacitor of a bit cell depends on if the bit cell is storing a one or zero). Therefore, if the majority of the values of the bit cells are one (at least two out of three), then the output is a one; otherwise, the output is a zero. More precisely, if X, Y, and Z represent the logical values of the three cells, then the final state of the bit line is $XY + YZ + ZX$. If $Z = 1$, then this is effectively an OR operation between X and Y; if $Z = 0$, then this is effectively an AND operation between X and Y. The bit-wise logic operations can be built up into MAC operations across multiple cycles [356], similar to bit-serial processing described in Chapter 7.

It is important to note that the architecture of processing in memory with DRAM differs from the processing in memory with NVM and SRAM (described in Sections 10.2.1 and 10.2.2, respectively) in that: (1) for

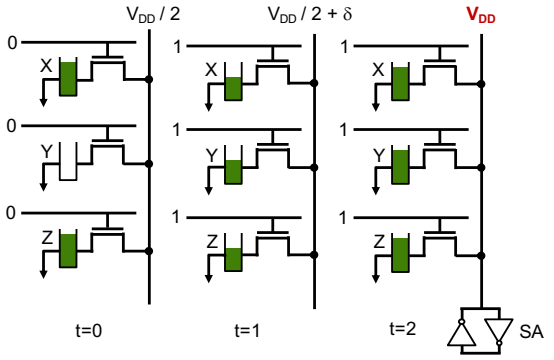
[10] It should be noted that since SRAM is less dense than typical off-chip memory (e.g., DRAM), they are not designed to replace off-chip memory or specifically addressing the "memory wall," which pertains to off-chip memory bandwidth; instead, most SRAM-based processing-in-memory accelerators focus on reducing the on-chip data movement.



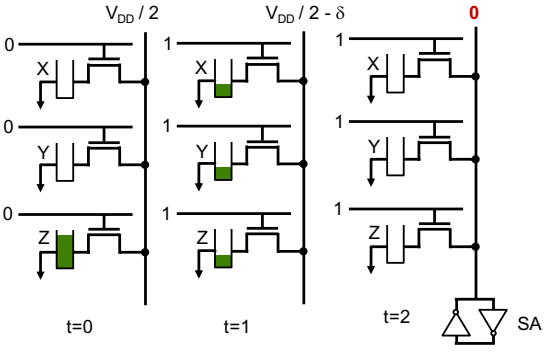
(a) Z=0 and AND (X=1, Y=1) = 1.



(b) Z=0 and AND (X=1, Y=0) = 0.



(c) Z=1 and OR (X=1, Y=0) = 1.



(d) Z=1 and OR (X=0, Y=0) = 0.

Figure 10.6: Compute in DRAM based on charge sharing. Z controls whether an AND or OR is performed on input X and Y. At time $t = 0$, the local capacitor of the bit cells for X, Y and Z are charged to V_{DD} for one and 0 for zero, and the bit line is pre-charged to $V_{DD}/2$. At time $t = 1$, the accessed transistors to the bit cells are enabled, and the capacitors are shorted together with the bit line. Charge sharing distributes the charge between the capacitors to ensure that the voltage across each capacitor is the same; therefore the resulting voltage on the bit line is proportional to the average charge across the three capacitors. If the majority of the capacitors stored at one (i.e., V_{DD}), then the voltage on the bit line would be above $V_{DD}/2$ (i.e., $+\delta$); otherwise, the voltage on the bit line drops below $V_{DD}/2$ (i.e., $-\delta$). At time $t = 2$, the sense amplifiers (SA) on the bit line amplify the voltage to full swing (i.e., $V_{DD}/2+\delta$ becomes V_{DD} or $V_{DD}/2-\delta$ becomes 0), such that the output of the logic function $XY + YZ + ZX$ can be resolved on the bit line. Note that this form of computing is destructive, so we need to copy data beforehand.

DRAM, a bit-wise operation requires three storage elements from different rows, whereas for NVM and SRAM, a MAC operation can be performed with a single storage element; and (2) for DRAM, only one bit-wise operation is performed per bit line and the accumulation occurs over time, whereas for NVM and SRAM, the accumulation of multiple MAC operations is performed on the bit line.[11] As a result, for DRAM the parallel processing can only be enabled across bit lines (A in Figure 10.3), since only one operation can be performed per bit line, whereas for NVM and SRAM, the parallel processing can be enabled across both the bit lines and the word lines (A and B in Figure 10.3), since multiple operations can be performed per bit line. In addition, for DRAM, multiple cycles are required to build up a MAC operation from a bit-wise logic operation, which reduces throughput. Thus, a challenge for DRAM-based processing-in-memory accelerators is to ensure that there is sufficient parallelism across bit lines (A) to achieve the desired improvements in throughput.

Other challenges for DRAM-based processing-in-memory accelerators include variations in the capacitance in the different bit cells, changing charge in capacitor of bit cell over time due to leakage, and detecting small changes in the bit line voltage. In addition, additional hardware may be required in the memory controller to access multiple rows at once and/or to convert the bit-wise logic operations to MAC operation, all of which can contribute to energy and area overhead.

While many of the recent works on DRAM-based processing-in-memory accelerators have been based on simulation [355, 356], it should be noted that performing AND and OR operations have been demonstrated on off-the shelf, unmodified, commercial DRAM [358]. This was achieved by violating the nominal timing specification and activating multiple rows in rapid succession, which leaves multiple rows open simultaneously and enables charge sharing on the bit line.

10.2.4 Design Challenges

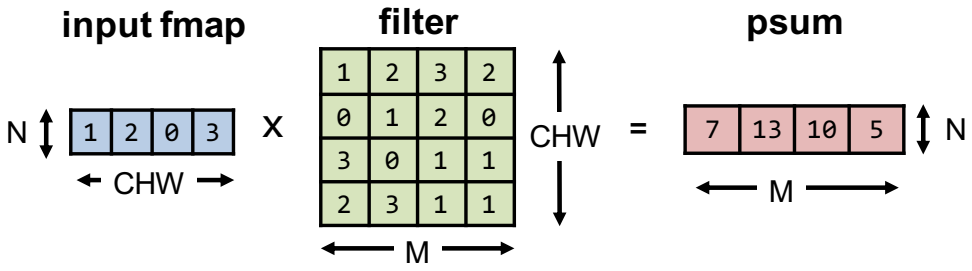
Processing-in-memory accelerators offer many potential benefits including reduced data movement of weights, higher memory bandwidth by reading multiple weights in parallel, higher throughput by performing multiple computations in parallel, and lower input activation delivery cost due to increased density of compute. However, there are several key design challenges and decisions that need to be considered in practice. Analog processing is typically required to bring the computation into the array of storage elements or into its peripheral circuits; therefore the major challenges for processing in memory are its sensitivity to circuit and device non-idealities (i.e., nonlinearity and process, voltage and temperature variations).[12] Solutions to these challenges often require trade offs between energy efficiency, throughput, area density, and accuracy,[13] which reduce the achievable gains over conventional architectures. Architecture-level energy and area estimation tools such as Accelergy can be used to help evaluate some of these trade offs [359].

In this section, when applicable we will use a toy example of a matrix vector multiplication based on a FC layer shown in Figure 10.7. A loop-nest representation of the design is shown in Design 22, where

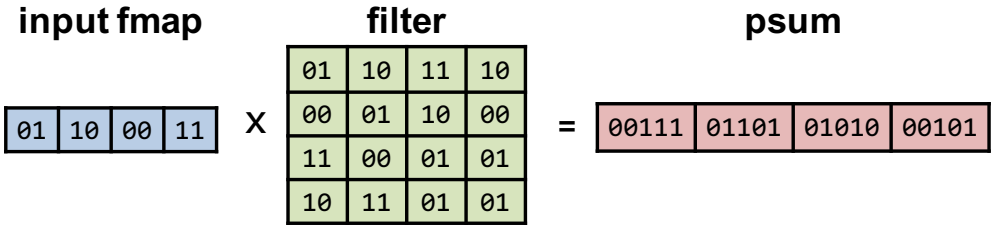
[11] This bit-wise (bit-serial) approach has also been explored for SRAM [357].

[12] Note that per chip training (i.e., different DNN weights per chip instance) may help address nonlinearity and chip to chip variability, but is expensive in practice. In addition, while adapting the weights can help address *static* variability, *dynamic* variability, such as a change in temperature, remains a challenge.

[13] It should be noted that the loss in accuracy might not only be due to the reduced precision of the computations in the DNN model (discussed in Chapter 7), which can be replicated on a conventional processor, but also due to circuit/device non-idealities and limitations, including ADC precision and thermal noise. Unfortunately, these factors have rarely been decoupled during reporting in literature, which can make it difficult to understand the design trade offs.



(a) Decimal



(b) Binary

Figure 10.7: Toy example of matrix vector multiplication for this section. This example uses an FC layer with $N = 1$, $CHW = 4$, and $M = 4$.

Design 22 Toy matrix multiply loop nest

```
i = Array(CHW)          # Input activations
f = Array(CHW, M)       # Filter weights
o = Array(M)            # Output partial sums

parallel-for m in [0, M):
    parallel-for chw in [0, CHW):
        o[m] += i[chw] * f[chw, m]
```

$CHW = M = 4$. In theory, the entire computation should only require one cycle as all the 16 weights can be accessed in parallel and all the 16 MAC operations can be performed in parallel.

Number of Storage Elements per Weight

Ideally, it would be desirable to be able to use one storage element (i.e., one device or bit cell) per weight to maximize density. In practice, multiple storage elements are required per weight due to the limited precision of each device or bit cell (typically on the order of 1 to 4 bits). As a result, multiple low-precision storage elements are used to represent a higher precision weight. Figure 10.8 shows how this applies to our toy example.

For non-volatile memories (e.g., RRAM), multiple storage elements can also be used per weight to reduce the effect of devices variation (e.g., average 3×3 devices per weight [341]) or to represent a signed weight

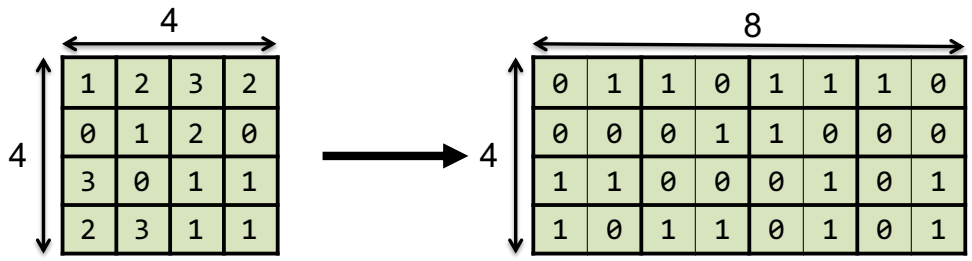


Figure 10.8: Example of multiple storage elements per weight. In our toy example we use 2 bits per weight so the storage cost goes from 4×4 to 4×8.

(i.e., since resistance is naturally non-negative, differential coding using two arrays is often used [341]). Finally, in the case of SRAMs, often additional transistors are required in the bit cell to perform an operation, which increases the area per bit cell. All of the above factors reduce the density and/or accuracy of the system.

Array Size

Ideally, it would be desirable to have a large array size ($A \times B$) in order to allow for high weight read bandwidth and high throughput. In addition, a larger array size improves the area density by further amortizing the cost of the peripheral circuits, which can be significant (e.g., the peripheral circuits, i.e., ADC and DAC, can account for over 50% of the energy consumption of NVM-based designs [328, 348]). In practice, the size of array limited by several factors.

1. The resistance and capacitance of word line and bit line wires, which impacts robustness, speed, and energy consumption.

   For instance, the bit line capacitance impacts robustness for charge domain approaches where charge sharing is used for accumulation, as a large bit line capacitance makes it difficult to sense the charge stored on the local capacitor in the bit cell; the charge stored on the local capacitor can be an input value for DRAM-based designs or a product of weight and input activation for SRAM-based designs. An example of using charge sharing to sense the voltage across a local capacitor is shown in Figure 10.9. Specifically, the change in bit line voltage (ΔV_{BL}) is

   $$\Delta V_{BL} = (V_{DD} - V_{local}) \frac{C_{local}}{C_{local} + C_{BL}} \tag{10.1}$$

   where C_{local} and C_{BL} are the capacitance of the local capacitor and bit line, respectively, and V_{local} is the voltage across the local capacitor (due to the charge stored on the local capacitor), and V_{DD} is the supply voltage. If the local capacitor is only storing binary values, then V_{local} can either be V_{DD} or 0. ΔV_{BL} must be sufficiently large such that we can measure any change in V_{local}; the more bits we want to measure on the bit line (i.e., bits of the partial sum or output activation), the larger the required ΔV_{BL}. However, the size of C_{local} is limited by the area density of the storage element; for instance, in [351], C_{local} is limited to 1.2fF. As a result, the minimum value of ΔV_{BL} limits the size of C_{BL}, which limits the length of the bit line.

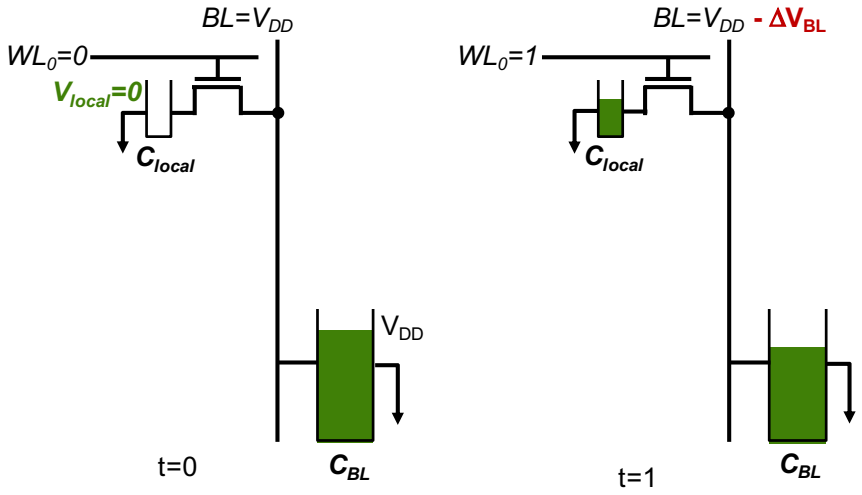


Figure 10.9: Change in bit line voltage ΔV_{BL} is proportional to $\frac{C_{local}}{C_{local}+C_{BL}}$. The bit line is precharged to V_{DD} at $t = 0$, and we read the value on the local capacitor at $t = 1$.

Similarly, the bit line resistance impacts robustness for current domain approaches where current summing is used for accumulation, as a large bit line resistance makes it difficult to sense the change in the resistance in the NVM device, as shown in Figure 10.10. Specifically, the change in bit line voltage due to change on the resistance is

$$\Delta V_{BL} = V_{HIGH} - V_{LOW} = V_{in} R_{BL} \frac{R_{OFF} - R_{ON}}{(R_{ON} + R_{BL})(R_{OFF} + R_{BL})} \tag{10.2}$$

where R_{ON} and R_{OFF} are the minimum and maximum resistance of the NVM device (proportional to the weight), respectively, R_{BL} is the resistance of the bit line, and V_{in} is the input voltage (proportional to the input activation). The $R_{OFF} - R_{ON}$ is limited by the NVM device [341]. As a result, the minimum value of ΔV_{BL} limits the size of R_{BL}, which again limits the length of the bit line.

2. The utilization of the array will drop if the workload cannot fill entire column or entire row, as shown in Figure 10.11(a). If the DNN model has few weights per filter and does not require large dot products, e.g., $C \times H \times W \leq B$, where C, H and W, are the dimensions of the filter (FC layer), and B is the number of rows in the array, then there will be $B - C \times H \times W$ idle rows in the array. If the DNN model has few output channels and does not have many dot products, e.g., $M \leq A$, where M is the number of output channels and A is the number of columns in the array, then there will be $A - M$ idle columns in the array.[14] This becomes more of an issue when processing efficient DNN models, as described in Chapter 9, where the trend is to reduce the number of weights per filter. In digital designs, flexible mapping can be used to increase utilization across different filter shapes, as discussed in Chapter 6; however, this is much more challenging to implement in the analog domain. One option is to redesign the DNN model specifically for processing in memory with larger filters and fewer

[14]Note that if $C \times H \times W > B$ or $M > A$, temporal tiling will need to be applied, as discussed in Chapter 4, and multiple passes (including updating weights in the array) will be required to complete the MAC operations. Furthermore, recall that if the completed sum (final psum) can be computed within a single pass (i.e., $C \times H \times W \leq B$), then precision of the ADC can be reduced to the precision of the output activation. However, when multiple passes are needed, the ADC needs greater precision because the results of each pass need to be added together to form the completed sum; otherwise, there may be an accuracy loss.

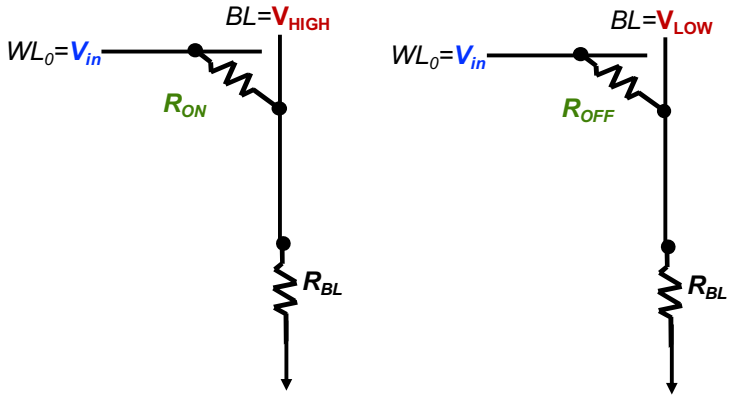


Figure 10.10: Change in bit line voltage $\Delta V_{BL} = V_{HIGH} - V_{LOW}$ is proportional to $R_{BL}\frac{R_{OFF}-R_{ON}}{(R_{ON}+R_{BL})(R_{OFF}+R_{BL})}$. R_{ON} (also referred to as LRS) and R_{OFF} (also referred to as HRS) are the minimum and maximum resistance of the NVM device, respectively.

layers [314], which increases utilization of the array and reduces input activation data movement; however, the accuracy implications of such DNN models requires further study. Figure 10.11(b) shows how this applies to our toy example.

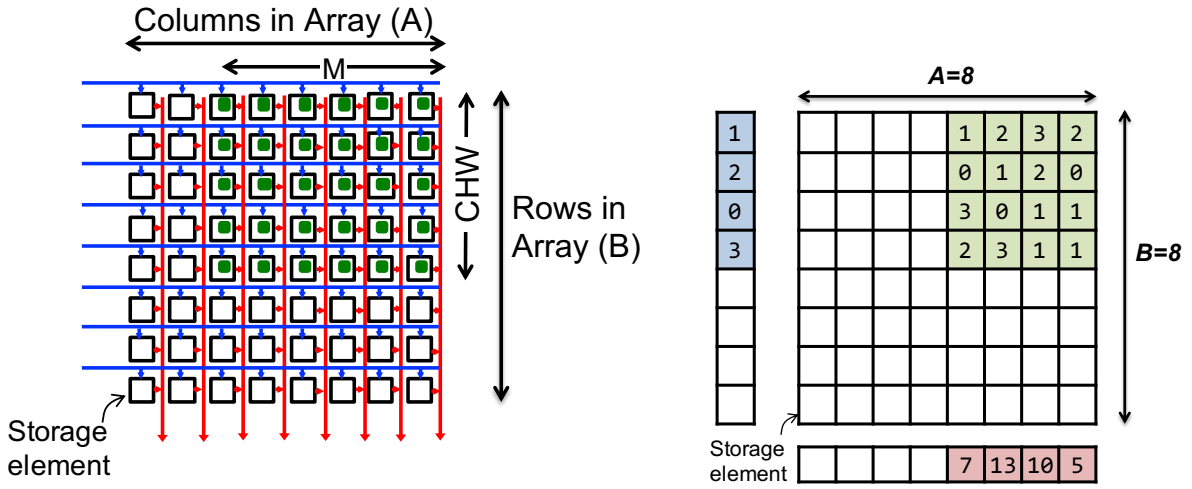
As a result, typical fabricated array sizes range from 16b×64 [352] to 512b×512 [351] for SRAM and from 128×128 to 256×256 [341] for NVM. This limitation in array size affects throughput, area density and energy efficiency. Multiple arrays can be used to scale up the design in order to fit the entire DNN Model and increase throughput [328, 346]. However, the impact on amortizing the peripheral cost is minimal. Furthermore, an additional NoC must be introduced between the arrays. Accordingly, the limitations on energy efficiency and area density remain.

Number of Rows Activated in Parallel

Ideally, it would be desirable to use all rows (B) at once to maximize parallelism for high bandwidth and high throughput. In practice, the number of rows that can be used at once is limited to by several factors.

1. The number of bits in the ADC, since more rows means more bits are required to resolve the accumulation (i.e., the partial sums will have more bits). Some works propose using fewer bits for ADC than the maximum required [360, 361], however, this can reduce the accuracy.[15]
2. The cumulative effect of the device variations can decrease the accuracy.

[15]The number of bits required by the ADC depends on the number of values being accumulated on the bit line (i.e., number of rows activated in parallel), whether the values are sparse [360] (i.e., zero values will not contribute to the accumulated sum), and whether the accumulated sum is a partial sum or a fully accumulated sum (i.e., it only needs to go through a nonlinear function to become an output activation). Using less than the maximum required ADC bits for the fully accumulated sum has less impact on accuracy than on the partial sum, since the fully accumulated sum is typically quantized to the bit-width of the input activation for the next layer, as discussed in Chapter 7. However, the ability to fully accumulate the sum on a bit line depends on the whether the number of rows in the array is large enough to hold all the weights for a given filter (i.e., $B \geq C \times H \times W$).



(a) Workload size versus array size

(b) Toy Example

Figure 10.11: Array utilization. (a) Impact of array size on utilization. (b) Example of utilization if size of weight memory was 8×8. Even though in theory we should be able to perform 64 MAC operations in parallel, only 16 of the storage elements are used (utilization of 25%); as a result, only 16 MAC operations are performed in parallel, specifically, 4 dot products of 4 elements.

3. The maximum voltage drop or accumulated current that can be tolerated by the bit line.[16] This can be particularly challenging for advanced process technologies (e.g., 7 nm and below) due to the increase in bit line resistance and increased susceptibility to electromigration issues, which limits the maximum current on the bit line.

As a result, the typical number of rows activated in parallel is 64 [341] or below [343]. A digital accumulator can be used after each ADC to accumulate across all B rows in B/64 cycles [341]; however, this reduces throughput and increases energy due to multiple ADC conversions. To reduce the additional ADC conversion, recent work has explored performing the accumulation in the analog domain [347]. Figure 10.12 shows how this applies to our toy example. Design 23 shows the corresponding loop nest, and illustrates the multiple cycles it takes to perform all the MACs.

Number of Columns Activated in Parallel

Ideally, it would be desirable to use all columns (A) at once to maximize parallelism for high bandwidth and high throughput. In practice, the number of columns that can be used are limited by whether the area of ADC can pitch-match the width of the column, which is required for a compact area design; this can be challenging when using high-density storage elements such as NVM devices. A common solution is to time multiplex the ADC across a set of eight columns, which means that only A/8 columns are used in parallel [341]; however, this reduces throughput. Figure 10.13 shows how this applies to our toy example, and Design 24 shows the corresponding loop nest.

[16] For instance, for a 6T SRAM bit cell, a large voltage drop on the bit line can cause the bit cell to flip (i.e, an unwanted write operation on the bit cell); using 8T bit cell can prevent this at the cost of increased area.

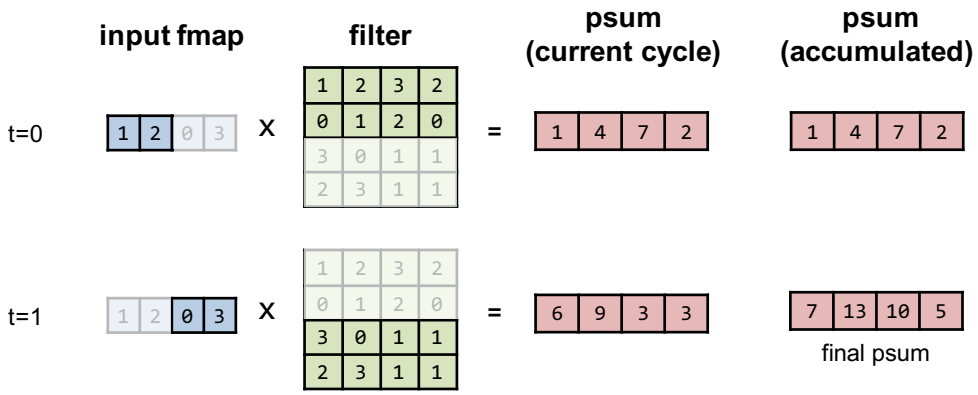


Figure 10.12: Example of limited number of rows activated in parallel. If the ADC is only 3-bits, only two rows can be used at a time. It would take two cycles (time steps) to complete the computation. There are two columns for psum in the figure: (1) psum (current cycle) corresponds to psum resulting from the dot product computed at the current cycle; (2) psum (accumulated) corresponds to the accumulated value of the psums across cycles. At $t = 1$, the psum of [6, 9, 3, 3] is computed and added (e.g., with a digital adder) to the psum at $t = 0$ of [1, 4, 7, 2] to achieve the final psum [7, 13, 10, 5], as shown in the figure.

Design 23 Toy matrix multiply loop nest with limited number of parallel active rows

```
i = Array(CHW)          # Input activations
f = Array(CHW, M)       # Filter weights
o = Array(M)            # Output partial sums

parallel-for m in [0, M):
    parallel-for chw1 in [0, CHW/2):
        for chw0 in [0, 2):
            chw = chw1*2 + chw0
            o[m] += i[chw] * f[chw, m]
```

Design 24 Toy matrix multiply loop nest with limited number of parallel active columns

```
i = Array(CHW)          # Input activations
f = Array(CHW, M)       # Filter weights
o = Array(CHW)          # Output partial sums

parallel-for m1 in [0, M/2):
    parallel-for chw in [0, CHW):
        for m0 in [0, 2):
            m = m1*2 + m0
            o[m] += i[chw] * f[chw, m]
```

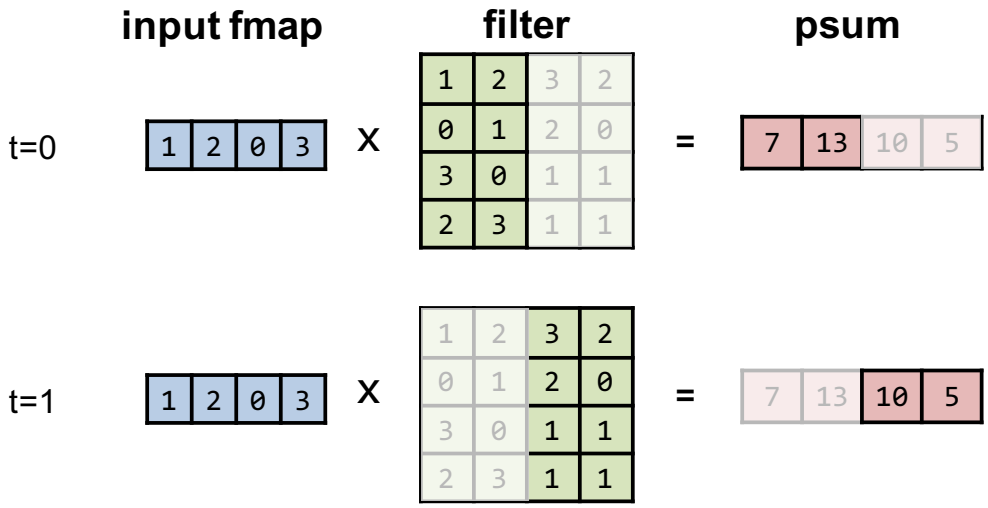


Figure 10.13: Example of limited number of columns activated in parallel. If the width of an ADC is equal to two columns, then the columns need to be time multiplexed. It would take two cycles to complete the computation. If we combined this with the previously described parallel row limitations, it would take four cycles to complete the computation.

Time to Deliver Input

Ideally, it would be desirable for all bits in the input activations to be encoded onto the word line in the minimum amount of time to maximize throughput; a typical approach is to use voltage amplitude modulation [350]. In practice, this can be challenging due to

1. the nonlinearity of devices makes encoding input value using voltage amplitude modulation difficult, and
2. the complexity of the DAC that drives the word line scales with the number of bits

As a result, the input activations are often encoded in time (e.g., pulse-width modulation [353, 354] or number of pulses [361][17]), with a fixed voltage (DAC is only 1-bit) where the partial sum is determined by accumulating charge over time; however, this reduces throughput.[18] Figure 10.14 shows how this applies to our toy example. One approach to reduce the complexity of the DAC or current accumulation time is to reduced the precision of the input activations, as discussed in Chapter 7; however, this will also reduce accuracy.

[17] Using pulses increases robustness to nonlinearity at the cost of increased switching activity.

[18] Alternatively, a single pulse can be used for the input activations if the weights are replicated across multiple rows (e.g., 2^{N-1} rows for an N-bit activation) [332]. This is a trade-off between time and area.

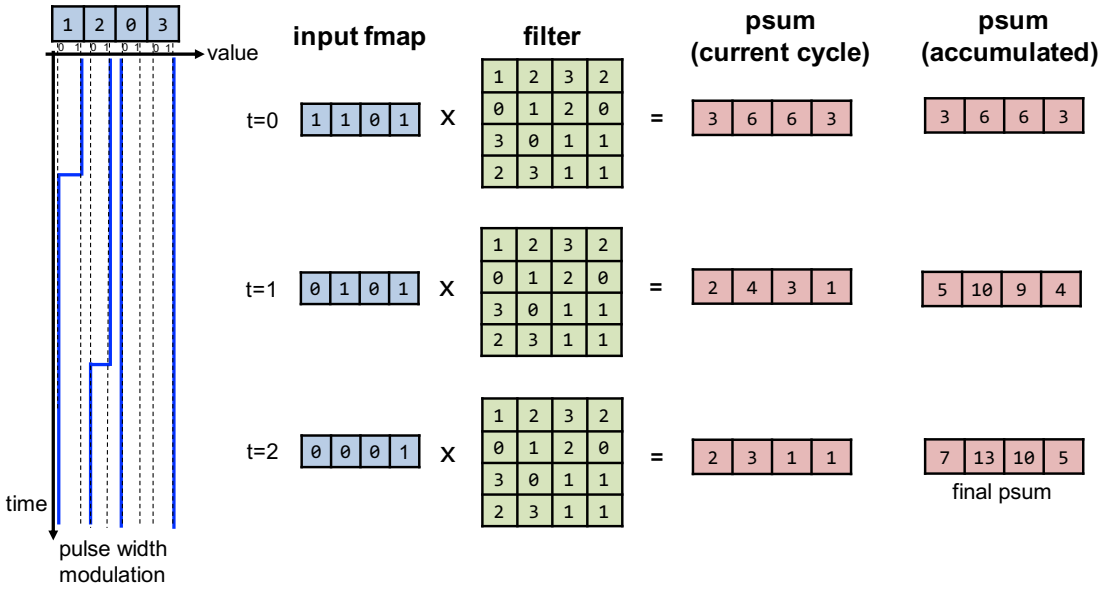


Figure 10.14: Example of performing pulse-width modulation of the input activations with a 1-bit DAC. It would take three cycles to complete the computation if all weights can be used at once. Specifically, the input activations would be signaled across time as $[1, 1, 0, 1] + [0, 1, 0, 1] + [0, 0, 0, 1] = [1, 2, 0, 3]$, where the width of the pulse in time corresponds to the value of the input. There are two columns for psum in the figure: (1) psum (current cycle) corresponds to psum resulting from the dot product computed at the current cycle; (2) psum (accumulated) corresponds to the accumulated value of the psums across cycles. Note that if we combined the limitation illustrated in this figure with the previously described parallel row and columns limitations, it would take 12 cycles to complete the computation.

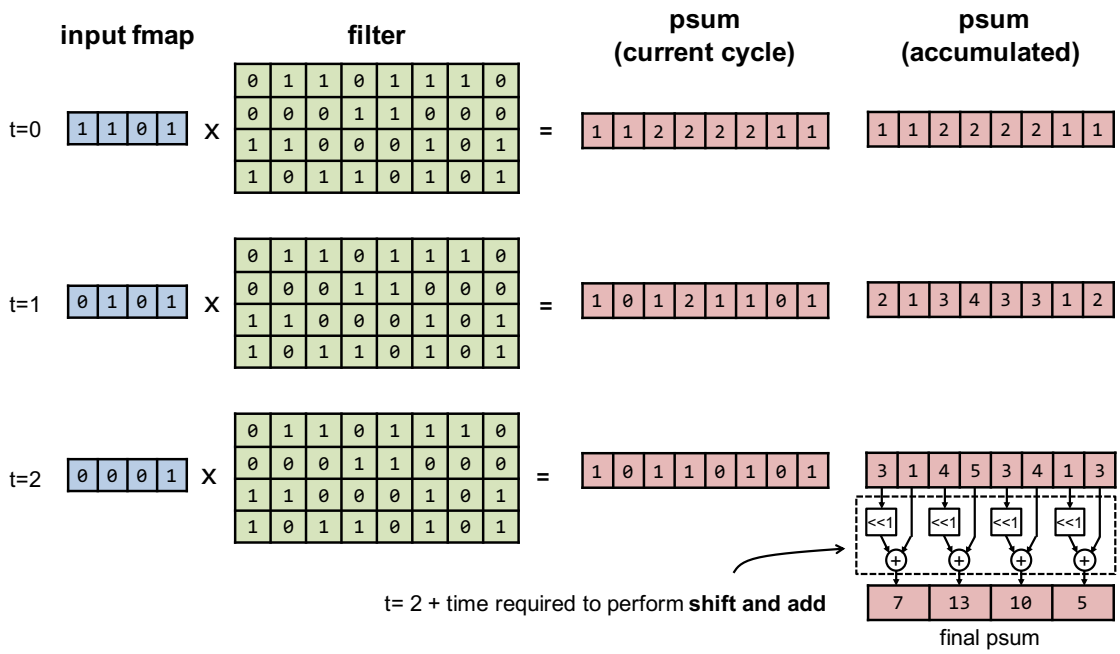


Figure 10.15: Example of time to compute MAC operation if each storage element can only perform one-bit operations. It takes three cycles to deliver the input (similar to Figure 10.14). There are two columns for psum in the figure: (1) psum (current cycle) corresponds to psum resulting from the dot product computed at the current cycle; (2) psum (accumulated) corresponds to the accumulated value of the psums across cycles. In addition, extra cycles are required at the end to combine accumulated bits from each bit line to form the final output sum. The number of cycles required to perform the shift and add would depend on the number of bit lines divide by the number of sets of shift-and-add logic.

Time to Compute a MAC

Ideally, it would be desirable for a MAC to be computed in a single cycle. In practice, the storage element (bit cell or device) typically can only perform one-bit operations (e.g., XNOR and AND), and thus multiple cycles are required to build up to a multi-bit operation (e.g., full adder and multiplication) [330]. Figure 10.15 shows how this applies to our toy example. This also requires additional logic after the ADC to combine the one-bit operations into a multi-bit operation. However, this will reduce both the throughput, energy and density.

10.3 Processing in Sensor

In certain applications, such as image processing, the data movement from the sensor itself can account for a significant portion of the overall energy consumption. Accordingly, there has been work on bringing the processing near or into the sensor, which is similar to the work on bringing the processing near or into memory discussed in the previous sections. In both cases, the goal is to reduce the amount of data read out of the memory/sensor and thus the number of ADC conversions, which can be expensive. Both cases also

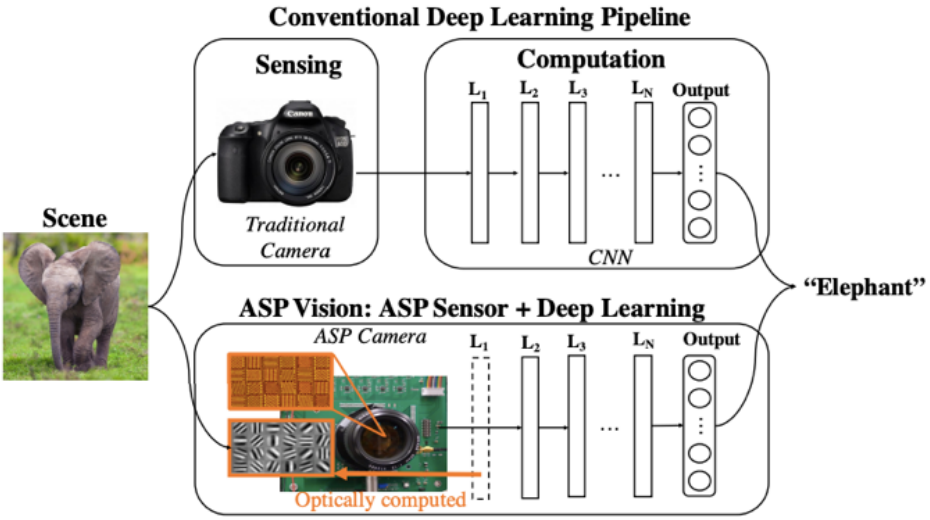


Figure 10.16: Use ASP in front end to perform processing of first layer (Figure from [366])

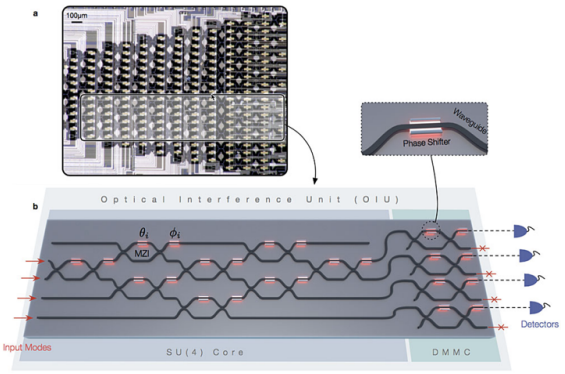
require moving the computation into the analog domain and consequently suffer from increased sensitivity to circuit non-idealities. While processing near memory and processing in memory focus on reducing data movement of the weights of the DNN model, processing near sensor and processing in sensor focus on reducing the data movement of the inputs to the DNN model.

Processing near sensor has been demonstrated for image processing applications, where computation can be performed in the analog domain before the ADC in the peripheral of the image sensor. For instance, Zhang et al. [362] and Lee et al. [363] use switched capacitors to perform 4-bit multiplications and 3-bit by 6-bit MAC operations, respectively. RedEye [364] proposes performing the entire convolution layer (including convolution, max pooling and quantization) in the analog domain before the ADC. It should be noted that the results in [364] are based on simulations, while [362, 363] report measurements from fabricated test chips.

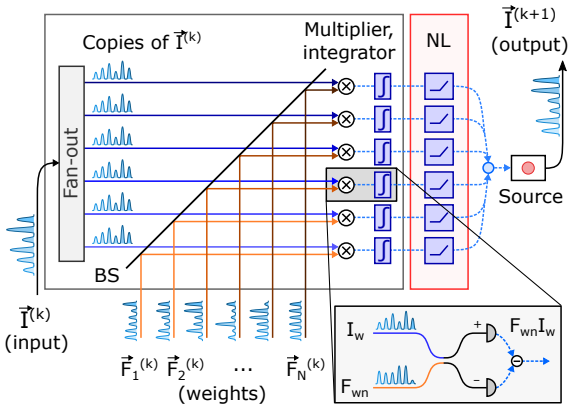
It is also feasible to embed the computation not just before the ADC, but directly into the sensor itself (i.e., processing in sensor). For instance, in [365] an Angle Sensitive Pixels sensor is used to compute the gradient of the input, which along with compression, reduces the data movement from the sensor by 10×. In addition, since the first layer of the DNN often outputs a gradient-like feature map, it may be possible to skip the computations in the first layer, which further reduces energy consumption, as discussed in [366, 367].

10.4 Processing in the Optical Domain

Processing in the optical domain is an area of research that is currently being explored as an alternative to all-electronic accelerators [368]. It is motivated, in part, by the fact that photons travel much faster than electrons, and the cost of moving a photon can be *independent* of distance. Furthermore, multiplication can be performed passively (for example with optical interference [369, 370], with reconfigurable filters [371],



(a) Optical neural network using optical interference units. (Figure from [369])



(b) Optical neural network using coherent detection. BS: beamsplitter; NL: nonlinearity. (Figure from [370])

Figure 10.17: Optical neural networks

or static phase masks [372]) and detection can occur at over 100 GHz. Thus, processing in the optical domain may provide significant improvements in energy efficiency and throughput over the electrical domain.

Much of the recent work in the optical computing has focused on performing matrix multiplication, which can be used for DNN processing; these works are often referred to as photonic accelerators or *optical neural networks*. For instance, Shen et al. [369] present a programmable nanophotonic processor where the input activations are encoded in the amplitudes of optical pulses (light) that travel through an array of on-chip interferometers (composed of beamsplitters) that represent the weight matrix, where the weights determine the amount of light that is passed to the output. This is effectively a weight-stationary dataflow. The accumulation is performed based on the accumulated light from various waveguides at the photodetector.

Alternatively, Hamerly et al. [370], shown in Figure 10.17(b), demonstrate matrix multiplication based on coherent detection, where both the weights and activations are encoded on-the-fly into light pulses, and are interfered in free-space on a beamsplitter to perform multiplication. Since, in this scheme, there is no need for on-chip interferometers (which have a large footprint), this approach may be more scalable, at the cost of added complexity in alignment. This is effectively an output-stationary dataflow, where the output is accumulated on the photodetector as an analog electronic signal.

There is negligible power loss in the computation when processing in the optical domain. Most of the power dissipation occurs when converting between electrical and optical domains, specifically, in the converter to generate the light and the detector to collect the photons. Therefore, similar to the processing in memory work, the larger the array (or in this case the matrix), the more these conversion costs can be amortized.

Note, however, that while computing in the optical domain may be energy efficient, the non-idealities in the optical devices (e.g., crosstalk between detectors, errors in phase encoding, photodetection noise) can lead to a reduction in accuracy. To address this accuracy loss, Bernstein et al. [373] propose a hybrid electronic-optics approach where the data transfer is done in the optical domain to exploit the distance-independent cost of photons, while the computation itself (i.e., MAC operation) is performed digitally in the electrical domain to avoid the non-idealities of the optical devices.

Recent works on optical neural networks have reported results based on simulations [370] or simulations

based on data that has been extrapolated from experimental results [369]. These works demonstrate functionality on simple DNN models for digit classification and vowel recognition.

Chapter 11

Conclusion

The use of deep neural networks (DNNs) has recently seen explosive growth. They are currently widely used for many artificial intelligence (AI) applications including computer vision, speech recognition, and robotics and are often delivering better than human accuracy. However, while DNNs can deliver this outstanding accuracy, it comes at the cost of high computational complexity. With the stagnation of improvements in general-purpose computation [10], there is a movement toward more domain-specific hardware, and in particular for DNN processing. Consequently, techniques that enable efficient processing of DNNs to improve *energy-efficiency* and *throughput* without sacrificing *accuracy* with cost-effective hardware are critical to expanding the deployment of DNNs in both existing and new domains.

Creating a system for efficient DNN processing should begin with understanding the current and future applications and the specific computations required for both now and the potential evolution of those computations. Therefore, this book surveyed a number of the current applications, focusing on computer vision applications, the associated algorithms, and the data being used to drive the algorithms. These applications, algorithms, and input data are experiencing rapid change. So extrapolating these trends to determine the degree of flexibility desired to handle next generation computations becomes an important ingredient of any design project.

During the design-space exploration process, it is critical to understand and balance the important system metrics. For DNN computation these include the accuracy, energy, throughput and hardware cost. Evaluating these metrics is, of course, key, so this book surveyed the important components of a DNN workload. In specific, a DNN workload has two major components. First, the workload consists of the "network architecture" of the DNN model including the "shape" of each layer and the interconnections between layers. These can vary both within and between applications. Second, the workload consists of the specific data input to the DNN. This data will vary with the input set used for training or the data input during operation for inference.

This book also surveyed a number of avenues that prior work have taken to optimize DNN processing. Since data movement dominates energy consumption, a primary focus of some recent research has been to reduce data movement while maintaining accuracy, throughput, and cost. This means selecting architectures with favorable memory hierarchies like a spatial array, and developing dataflows that increase data reuse at the low-cost levels of the memory hierarchy. We have included a taxonomy of dataflows and an analysis of their characteristics. Understanding the throughput and energy efficiency of a DNN accelerator depends upon

how each DNN workload maps to the hardware. Therefore, we discussed the process of optimally mapping workloads to the accelerator and the associated throughput and energy models.

The DNN domain also affords an excellent opportunity for hardware/algorithm co-design. Many works have aimed to save storage space and energy by changing the representation of data values in the DNN. We distill and present the key concepts from these approaches. Still other work saves energy and sometimes increases throughput by increasing and then exploiting sparsity of weights and/or activations. We presented a new abstract data representation that enables a systematic presentation of designs focused on exploiting sparsity. Co-design needs to be aware of the impact on accuracy. Therefore, to avoid losing accuracy it is often useful to modify the network or fine-tune the network's weights to accommodate these changes. Thus, this book both reviewed a variety of these techniques and discussed the frameworks that are available for describing, running and training networks.

Finally, DNNs afford the opportunity to use mixed-signal circuit design and advanced technologies to improve efficiency. These include using memristors for analog computation and 3-D stacked memory. Advanced technologies can also facilitate moving computation closer to the source by embedding computation near or within the sensor and the memories. Of course, all of these techniques should also be considered in combination, while being careful to understand their interactions and looking for opportunities for joint hardware/algorithm co-optimization.

In conclusion, although much work has been done, DNNs remain an important area of research with many promising applications and opportunities for innovation at various levels of hardware design. We hope this book provides a structured way of navigating the complex space of DNN accelerators designs that will inspire and lead to new advances in the field.

Bibliography

[1] J. Dean, "Machine learning for systems and systems for machine learning," in *Presentation at 2017 Conference on Neural Information Processing Systems*, 2017.

[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[3] F.-F. Li, A. Karpathy, and J. Johnson, "Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition," http://cs231n.stanford.edu/.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015.

[5] L. Deng, J. Li, J.-T. Huang, K. Yao, D. Yu, F. Seide, M. Seltzer, G. Zweig, X. He, J. Williams *et al.*, "Recent advances in deep learning for speech research at Microsoft," in *ICASSP*, 2013.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *NeurIPS*, 2012.

[7] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *ICCV*, 2015.

[8] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[10] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 27–29.

[11] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, July 2001.

[12] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[13] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, 2016.

[14] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

[15] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *ICCV*, 2017, pp. 843–852.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[18] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," in *ICLR*, 2014.

[19] Y. LeCun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard, "Handwritten digit recognition: applications of neural network chips and automatic learning," *IEEE Commun. Mag.*, vol. 27, no. 11, pp. 41–46, Nov 1989.

[20] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 IRE WESCON Convention Record*, 1960.

[21] B. Widrow, "Thinking about thinking: the discovery of the LMS algorithm," *IEEE Signal Process. Mag.*, 2005.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[24] "Complete Visual Networking Index (VNI) Forecast," Cisco, June 2016.

[25] J. Woodhouse, "Big, big, big data: higher and higher resolution video surveillance," technology.ihs.com, January 2016.

[26] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *CVPR*, 2014.

[27] J. Long, E. Shelhamer, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," in *CVPR*, 2015.

[28] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *NeurIPS*, 2014.

[29] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[30] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[31] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *CoRR abs/1609.03499*, 2016.

[32] H. Y. Xiong, B. Alipanahi, L. J. Lee, H. Bretschneider, D. Merico, R. K. Yuen, Y. Hua, S. Gueroussov, H. S. Najafabadi, T. R. Hughes *et al.*, "The human splicing code reveals new insights into the genetic determinants of disease," *Science*, vol. 347, no. 6218, p. 1254806, 2015.

[33] J. Zhou and O. G. Troyanskaya, "Predicting effects of noncoding variants with deep learning-based sequence model," *Nature methods*, vol. 12, no. 10, pp. 931–934, 2015.

[34] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of dna-and rna-binding proteins by deep learning," *Nature biotechnology*, vol. 33, no. 8, pp. 831–838, 2015.

[35] H. Zeng, M. D. Edwards, G. Liu, and D. K. Gifford, "Convolutional neural network architectures for predicting dna–protein binding," *Bioinformatics*, vol. 32, no. 12, pp. i121–i127, 2016.

[36] M. Jermyn, J. Desroches, J. Mercier, M.-A. Tremblay, K. St-Arnaud, M.-C. Guiot, K. Petrecca, and F. Leblond, "Neural networks improve brain cancer detection with raman spectroscopy in the presence of operating room light artifacts," *Journal of Biomedical Optics*, vol. 21, no. 9, pp. 094 002–094 002, 2016.

[37] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, "Deep learning for identifying metastatic breast cancer," *arXiv preprint arXiv:1606.05718*, 2016.

[38] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," in *NeurIPS Deep Learning Workshop*, 2013.

[40] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[41] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[42] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, "From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots," in *ICRA*, 2017.

[43] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik, "Cognitive mapping and planning for visual navigation," in *CVPR*, 2017.

[44] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *ICRA*, 2016.

[45] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," in *NeurIPS Workshop on Learning, Inference and Control of Multi-Agent Systems*, 2016.

[46] N. Hemsoth, "The Next Wave of Deep Learning Applications," Next Platform, September 2016.

[47] A. Suleiman, Y.-H. Chen, J. Emer, and V. Sze, "Towards closing the energy gap between HOG and CNN features for embedded vision," in *ISCAS*, 2017.

[48] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[49] F. V. Veen, "The Neural Network Zoo," The Asimov Institute Blog, 2016. [Online]. Available: https://www.asimovinstitute.org/neural-network-zoo/

[50] F. Kjolstad, S. Kamil, S. Chou, D. Lugato, and S. Amarasinghe, "The tensor algebra compiler," *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–29, 2017.

[51] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *ICML*, 2010.

[52] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML*, 2013.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.

[54] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *ICLR*, 2016.

[55] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[56] X. Zhang, J. Trmal, D. Povey, and S. Khudanpur, "Improving deep neural network acoustic models using generalized maxout networks," in *ICASSP*, 2014.

[57] Y. Zhang, M. Pezeshki, P. Brakel, S. Zhang, , C. Laurent, Y. Bengio, and A. Courville, "Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks," in *Interspeech*, 2016.

[58] Shelhamer, Evan and Donahue, Jeff and Lon, Jon, "Deep Learning for Vision Using CNNs and Caffe: A Hands-On Tutorial," 2016.

[59] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM International Conference on Multimedia*, 2014.

[60] A. Dosovitskiy, J. T. Springenberg, M. Tatarchenko, and T. Brox, "Learning to generate chairs, tables and cars with convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 4, pp. 692–705, 2016.

[61] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *ICCV*, 2011, pp. 2018–2025.

[62] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, 2016. [Online]. Available: http://distill.pub/2016/deconv-checkerboard

[63] D. Wofk, F. Ma, T. Yang, S. Karaman, and V. Sze, "FastDepth: Fast Monocular Depth Estimation on Embedded Systems," in *ICRA*, May 2019, pp. 6101–6108.

[64] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *ECCV*, 2014.

[65] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.

[66] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?(No, It Is Not About Internal Covariate Shift)," in *NeurIPS*, 2018.

[67] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[68] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, C. Ledig, and Z. Wang, "Is the deconvolution layer the same as a convolutional layer?" *arXiv preprint arXiv:1609.07009*, 2016.

[69] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *ICASSP*, 2013.

[70] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[71] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks," in *ICLR*, 2014.

[72] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *ICLR*, 2015.

[73] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper With Convolutions," in *CVPR*, 2015.

[74] M. Lin, Q. Chen, and S. Yan, "Network in Network," in *ICLR*, 2014.

[75] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *CVPR*, 2016.

[76] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," in *AAAI*, 2017.

[77] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[78] G. Urban, K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson, "Do Deep Convolutional Nets Really Need to be Deep and Convolutional?" *ICLR*, 2017.

[79] "Caffe LeNet MNIST," http://caffe.berkeleyvision.org/gathered/examples/mnist.html.

[80] "Caffe Model Zoo," http://caffe.berkeleyvision.org/model_zoo.html.

[81] "Matconvnet Pretrained Models," http://www.vlfeat.org/matconvnet/pretrained/.

[82] "TensorFlow-Slim image classification library," https://github.com/tensorflow/models/tree/master/slim.

[83] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[84] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *BMVC*, 2017.

[85] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE, 2017, pp. 5987–5995.

[86] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *ICML*, 2019.

[87] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[88] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[89] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[90] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[91] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *ICCV*, 2015.

[92] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *CVPR*, 2017.

[93] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *ICCV*, 2017.

[94] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[95] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "UNPU: A 50.6 TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*. IEEE, 2018, pp. 218–220.

[96] J. Giraldo and M. Verhelst, "Laika: A 5uw programmable lstm accelerator for always-on keyword spotting in 65nm cmos," in *ESSCIRC 2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2018, pp. 166–169.

[97] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[98] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[99] "Deep Learning Frameworks," https://developer.nvidia.com/deep-learning-frameworks.

[100] Y.-H. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, 2017.

[101] "Open Neural Network Exchange (ONNX)," https://onnx.ai/.

[102] C. J. B. Yann LeCun, Corinna Cortes, "THE MNIST DATABASE of handwritten digits," http://yann.lecun.com/exdb/mnist/.

[103] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of neural networks using dropconnect," in *ICML*, 2013.

[104] A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," https://www.cs.toronto.edu/~kriz/cifar.html.

[105] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, 2008.

[106] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, 2010.

[107] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[108] "Pascal VOC data sets," http://host.robots.ox.ac.uk/pascal/VOC/.

[109] "Microsoft Common Objects in Context (COCO) dataset," http://mscoco.org/.

[110] "Google Open Images," https://github.com/openimages/dataset.

[111] "YouTube-8M," https://research.google.com/youtube8m/.

[112] "AudioSet," https://research.google.com/audioset/index.html.

[113] "Standard Performance Evaluation Corporation(SPEC)," https://www.spec.org/.

[114] "MLPref," https://mlperf.org/.

[115] "DeepBench," https://github.com/baidu-research/DeepBench.

[116] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *IISWC*, 2016, pp. 1–10.

[117] J. D. Little, "A proof for the queuing formula: L= λ w," *Operations research*, vol. 9, no. 3, pp. 383–387, 1961.

[118] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, Apr 2009.

[119] B. Chen and J. M. Gilbert, "Introducing the CVPR 2018 On-Device Visual Intelligence Challenge," Google AI Blog, 2018. [Online]. Available: https://ai.googleblog.com/2018/04/introducing-cvpr-2018-on-device-visual.html

[120] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *IEEE ISSCC*, 2014.

[121] J. Standard, "High bandwidth memory (HBM) DRAM," *JESD235*, 2013.

[122] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, and et al., "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 14–27. [Online]. Available: https://doi-org.libproxy.mit.edu/10.1145/3352460.3358302

[123] S. Lie, "Wafer Scale Deep Learning," in *Hot Chips 31 Symposium (HCS), 2019 IEEE*, 2019.

[124] S. Condon, "Facebook unveils Big Basin, new server geared for deep learning," ZDNet, March 2017.

[125] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 60–71.

[126] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *SIGPLAN Not.*, vol. 26, no. 4, pp. 63–74, Apr. 1991. [Online]. Available: http://doi.acm.org/10.1145/106973.106981

[127] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'78)*. Tucson, Arizona: ACM, 1978, pp. 84–96.

[128] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," in *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '13. New York, NY, USA: ACM, 2013, pp. 519–530. [Online]. Available: http://doi.acm.org/10.1145/2491956.2462176

[129] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze *et al.*, "{TVM}: An automated end-to-end optimizing compiler for deep learning," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 578–594.

[130] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," in *ICANN*, 2014.

[131] D. H. Bailey, K. Lee, and H. D. Simon, "Using Strassen's algorithm to accelerate the solution of linear systems," *The Journal of Supercomputing*, vol. 4, no. 4, pp. 357–371, 1991.

[132] S. Winograd, *Arithmetic complexity of computations*. Siam, 1980, vol. 33.

[133] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *CVPR*, 2016.

[134] Nvidia, "NVDLA Open Source Project," 2017. [Online]. Available: http://nvdla.org/

[135] C. Dubout and F. Fleuret, "Exact acceleration of linear object detectors," in *ECCV*, 2012.

[136] J. S. Lim, "Two-dimensional signal and image processing," *Englewood Cliffs, NJ, Prentice Hall, 1990, 710 p.*, 1990.

[137] "Intel Math Kernel Library," https://software.intel.com/en-us/mkl.

[138] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN: Efficient Primitives for Deep Learning," *arXiv preprint arXiv:1410.0759*, 2014.

[139] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, 1965.

[140] C. E. Leicerson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top," 2020, unpublished manuscript.

[141] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *ISCA*, 2016.

[142] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," *IEEE Journal of Solid-State Circuits*, vol. 22, no. 1, pp. 28–34, 1987.

[143] C.-E. Lee, Y. S. Shao, A. Parashar, J. Emer, S. W. Keckler, and Z. Zhang, "Stitch-X: An Accelerator Architecture for Exploiting Unstructured Sparsity in Deep Neural Networks," in *SysML Conference*, 2018.

[144] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.

[145] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *CVPR Workshop*, 2014.

[146] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P. Graf, "A Massively Parallel Coprocessor for Convolutional Neural Networks," in *ASAP*, 2009.

[147] S. Park, K. Bong, D. Shin, J. Lee, S. Choi, and H.-J. Yoo, "A 1.93TOPS/W scalable deep learning/inference processor with tetra-parallel MIMD architecture for big-data applications," in *ISSCC*, 2015.

[148] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A Dynamically Configurable Coprocessor for Convolutional Neural Networks," in *ISCA*, 2010.

[149] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, "Towards an embedded biologically-inspired machine vision processor," in *FPT*, 2010.

[150] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A Convolutional Network Accelerator," in *GLVLSI*, 2015.

[151] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *MICRO*, 2014.

[152] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning," in *ASPLOS*, 2014.

[153] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *FPGA*, 2015.

[154] B. Moons and M. Verhelst, "A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets," in *Symp. on VLSI*, 2016.

[155] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting Vision Processing Closer to the Sensor," in *ISCA*, 2015.

[156] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep Learning with Limited Numerical Precision," in *ICML*, 2015.

[157] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for Convolutional Neural Networks," in *ICCD*, 2013.

[158] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," in *ISCA*, 2017.

[159] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *ISSCC*, 2016.

[160] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[161] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "Flexflow: A flexible dataflow accelerator architecture for convolutional neural networks," in *HPCA*, 2017, pp. 553–564.

[162] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, 2017.

[163] H. Kwon, A. Samajdar, and T. Krishna, "Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects," in *ASPLOS*, 2018, pp. 461–475.

[164] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *MICRO*, 2016.

[165] Azizimazreah, Arash and Chen, Lizhong, "Shortcut Mining: Exploiting Cross-layer Shortcut Reuse in DCNN Accelerators," in *HPCA*, 2019.

[166] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, and M. Kuroda, T.and Motomura, "BRein Memory: A 13-Layer 4.2 K Neuron/0.8 M Synapse Binary/Ternary Reconfigurable In-Memory Deep Neural Network Accelerator in 65nm CMOS," in *Symp. on VLSI*, 2017.

[167] J. Fowers and K. Ovtcharov and M. Papamichael and T. Massengill and M. Liu and D. Lo and S. Alkalay and M. Haselman and L. Adams and M. Ghandi and S. Heil and P. Patel and A. Sapek and G. Weisz and L. Woods and S. Lanka and S. K. Reinhardt and A. M. Caulfield and E. S. Chung and D. Burger, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *ISCA*, 2018.

[168] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 807–820. [Online]. Available: https://doi-org.libproxy.mit.edu/10.1145/3297858.3304014

[169] M. Pellauer, Y. S. Shao, J. Clemons, N. Crago, K. Hegde, R. Venkatesan, S. W. Keckler, C. W. Fletcher, and J. Emer, "Buffets: An efficient and composable storage idiom for explicit decoupled data orchestration," in *Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, 2019.

[170] J. Nickolls and W. J. Dally, "The GPU Computing Era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, March 2010.

[171] E. G. Cota, P. Mantovani, G. D. Guglielmo, and L. P. Carloni, "An analysis of accelerator coupling in heterogeneous architectures," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[172] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, K. Gururaj, and G. Reinman, "Accelerator-rich architectures: Opportunities and progresses," in *Proceedings of the Design Automation Conference (DAC)*, 2014.

[173] J. E. Smith, "Decoupled Access/Execute Computer Architectures," in *ISCA*, April 1982, pp. 112–119.

[174] *FIFO Generator v13.1: LogiCORE IP Product Guide, Vivado Design Suite*, PG057, April 5, Xilinx, 2017.

[175] *FIFO: Intel FPGA IP User Guide*, Updated for Intel Quartus Prime Design Suite: 18.0, Intel, 2018.

[176] A. Yazdanbakhsh, H. Falahati, P. J. Wolfe, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "Ganax: A unified mimd-simd acceleration for generative adversarial networks," *ISCA*, 2018.

[177] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale DNN processor for real-time AI," in *The International Symposium on Computer Architecture (ISCA)*, 2018.

[178] T. J. Ham, J. L. Aragón, and M. Martonosi, "Desc: Decoupled supply-compute communication management for heterogeneous architectures," in *MICRO*, December 2015, pp. 191–203.

[179] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI," in *ISSCC*, 2017, pp. 246–247.

[180] S. Yin, P. Ouyang, S. Tang, F. Tu, X. Li, L. Liu, and S. Wei, "A 1.06-to-5.09 tops/w reconfigurable hybrid-neural-network processor for deep learning applications," in *VLSI Circuits, 2017 Symposium on*. IEEE, 2017, pp. C26–C27.

[181] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo, "UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *ISSCC*, 2018, pp. 218–220.

[182] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[183] Y.-H. Chen, J. Emer, and V. Sze, "Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators," *IEEE Micro's Top Picks from the Computer Architecture Conferences*, vol. 37, no. 3, May-June 2017.

[184] R. M. Karp, R. E. Miller, and S. Winograd, "The Organization of Computations for Uniform Recurrence Equations," *Journal of the ACM*, vol. 14, no. 3, pp. 563–590, 1967.

[185] L. Lamport, "The Parallel Execution of DO loops," *Communications of the ACM*, vol. 17, no. 2, pp. 83–93, feb 1974.

[186] Wu, Yannan N. and Emer, Joel S. and Sze, Vivienne, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.

[187] B. Dally, "Power, Programmability, and Granularity: The Challenges of ExaScale Computing," in *IEEE IPDPS*, 2011.

[188] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards Energy-proportional Datacenter Memory with Mobile DRAM," in *ISCA*, 2012.

[189]

[190] B. Pradelle, B. Meister, M. Baskaran, J. Springer, and R. Lethin, "Polyhedral Optimization of TensorFlow Computation Graphs," in *Workshop on Extreme-scale Programming Tools (ESPT)*, November 2017.

[191] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *ISPASS*, 2019.

[192] J. Ragan-Kelley, C. Barnes, A. Adams, S. Paris, F. Durand, and S. Amarasinghe, "Halide: A language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines," *SIGPLAN Not.*, vol. 48, no. 6, p. 519–530, Jun. 2013. [Online]. Available: https://doi.org/10.1145/2499370.2462176

[193] R. Baghdadi, J. Ray, M. B. Romdhane, E. D. Sozzo, A. Akkas, Y. Zhang, P. Suriana, S. Kamil, and S. P. Amarasinghe, "Tiramisu: A polyhedral compiler for expressing fast and portable code," in *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2019, Washington, DC, USA, February 16-20, 2019*, M. T. Kandemir, A. Jimborean, and T. Moseley, Eds. IEEE, 2019, pp. 193–205. [Online]. Available: https://doi.org/10.1109/CGO.2019.8661197

[194] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolić, *Digital integrated circuits: a design perspective*. Pearson Education Upper Saddle River, NJ, 2003, vol. 7.

[195] B. Ramkumar and H. M. Kittur, "Low-power and area-efficient carry select adder," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 2, pp. 371–375, Feb 2012.

[196] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *ICLR*, 2016.

[197] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional Neural Networks using Logarithmic Data Representation," *arXiv preprint arXiv:1603.01025*, 2016.

[198] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-Efficient Neural Networks Using Logrithmic Computations," in *ICASSP*, 2017.

[199] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented Approximation of Convolutional Neural Networks," in *ICLR*, 2016.

[200] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing Neural Networks with the Hashing Trick," in *ICML*, 2015.

[201] W. Dally, "High-Performance Hardware for Machine Learning," Tutorial at Neurips 2015, 2015. [Online]. Available: https://media.nips.cc/Conferences/2015/tutorialslides/Dally-NIPS-Tutorial-2015.pdf

[202] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Prentice-Hall, 1963.

[203] D. Williamson, "Dynamically scaled fixed point arithmetic," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1991.

[204] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. Pai, and N. Rao, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *NeurIPS*, 2017.

[205] T. P. Morgan, "Nvidia Pushes Deep Learning Inference With New Pascal GPUs," Next Platform, September 2016.

[206] S. Higginbotham, "Google Takes Unconventional Route with Homegrown Machine Learning Chips," Next Platform, May 2016.

[207] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[208] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman *et al.*, "Serving DNNs in Real Time at Datacenter Scale with Project Brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[209] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[210] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[211] V. Camus, L. Mei, C. Enz, and M. Verhelst, "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

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

[213] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," in *ISCA*, 2018.

[214] L. Mei, M. Dandekar, D. Rodopoulos, J. Constantin, P. Debacker, R. Lauwereins, and M. Verhelst, "Sub-word parallel precision-scalable MAC engines for efficient embedded DNN inference," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2019, pp. 6–10.

[215] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *MICRO*, 2016.

[216] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos, "Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[217] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *MICRO*, 2017.

[218] S. Ryu, H. Kim, W. Yi, and J.-J. Kim, "BitBlade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[219] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[220] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," in *ECCV*, 2016.

[221] F. Li and B. Liu, "Ternary weight networks," in *NeurIPS Workshop on Efficient Methods for Deep Neural Networks*, 2016.

[222] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *CVPR*, 2017.

[223] S. Yin, P. Ouyang, J. Yang, T. Lu, X. Li, L. Liu, and S. Wei, "An ultra-high energy-efficient reconfigurable processor for deep neural networks with binary/ternary weights in 28nm CMOS," in *Symp. on VLSI*, 2018.

[224] R. Andri, L. Cavigelli, D. Rossi, and L. Benini, "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights," in *ISVLSI*, 2016.

[225] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *ISFPGA*, 2017.

[226] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "WRPN: Wide Reduced-Precision Networks," in *ICLR*, 2018.

[227] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: ineffectual-neuron-free deep neural network computing," in *ISCA*, 2016.

[228] Y. Lin, C. Sakr, Y. Kim, and N. Shanbhag, "PredictiveNet: an energy-efficient convolutional neural network via zero prediction," in *ISCAS*, 2017.

[229] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, "Prediction based Execution on Deep Neural Networks," in *ISCA*, 2018.

[230] V. Akhlaghi, A. Yazdanbakhsh, K. Samadi, R. K. Gupta, and H. Esmaeilzadeh, "SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks," in *ISCA*, 2018.

[231] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard*. Springer Science & Business Media, 1992.

[232] I. T. U. (ITU), "Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services," ITU-T, Tech. Rep., 2003.

[233] ——, "Recommendation ITU-T H.265: High Efficiency Video Coding," ITU-T, Tech. Rep., 2013.

[234] M. Mahmoud, K. Siu, and A. Moshovos, "Diffy: a Deja vu-Free Differential Deep Neural Network Accelerator ," in *MICRO*, 2018.

[235] M. Riera, J. Maria Arnau, and A. Gonzalez, "Computation Reuse in DNNs by Exploiting Input Similarity," in *ISCA*, 2018.

[236] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "Eva^2: Exploiting Temporal Redundancy in Live Computer Vision," in *ISCA*, 2018.

[237] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-SoC Co-Design for Low-Power Mobile Continuous Vision," in *ISCA*, 2018.

[238] Z. Zhang and V. Sze, "FAST: A framework to accelerate super-resolution processing on compressed videos," in *CVPR Workshop on New Trends in Image Restoration and Enhancement*, 2017.

[239] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[240] T.-J. Yang, M. D. Collins, Y. Zhu, J.-J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L.-C. Chen, "Deeperlab: Single-shot image parser," *arXiv preprint arXiv:1902.05093*, 2019.

[241] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *International Conference on 3D Vision (3DV)*, 2016, pp. 239–248.

[242] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[243] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *ECCV*, 2016.

[244] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *CVPR*, 2016.

[245] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *ECCV*, 2016.

[246] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "UCNN: Exploiting Computational Reuse in Deep Neural Networks via Weight Repetition," in *ISCA*, 2018.

[247] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" in *MLSys*, 2020.

[248] M. C. Mozer and P. Smolensky, "Using relevance to reduce network size automatically," *Connection Science*, vol. 1, no. 1, pp. 3–16, 1989.

[249] S. A. Janowsky, "Pruning versus clipping in neural networks," *Physical Review A*, vol. 39, no. 12, p. 6600, 1989.

[250] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE transactions on neural networks*, vol. 1, no. 2, pp. 239–242, 1990.

[251] R. Reed, "Pruning algorithms-a survey," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[252] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," in *CVPR*, 2017.

[253] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv preprint arXiv:1902.09574*, 2019.

[254] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICRL*, 2019.

[255] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage," in *NeurIPS*, 1990.

[256] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *NeurIPS*, 2015.

[257] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications," in *ECCV*, 2018.

[258] T.-J. Yang, Y.-H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *Asilomar Conference on Signals, Systems, and Computers*, 2017.

[259] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "*NeuralPower*: Predict and deploy energy-efficient convolutional neural networks," in *Proceedings of the Ninth Asian Conference on Machine Learning*, 2017.

[260] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *NeurIPS*, 2016.

[261] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *ICLR*, 2017.

[262] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.

[263] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *CVPR*, 2017.

[264] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN Pruning to the Underlying Hardware Parallelism," in *ISCA*, 2017.

[265] X. Wang, J. Yu, C. Augustine, R. Iyer, and R. Das, "Bit prudent in-cache acceleration of deep convolutional neural networks," in *HPCA*, 2019.

[266] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *ASPLOS*, 2019.

[267] A. Renda, J. Frankle, and M. Carbin, "Comparing Rewinding and Fine-tuning in Neural Network Pruning," in *ICLR*, 2020.

[268] V. Tresp, R. Neuneier, and H.-G. Zimmermann, "Early brain damage," in *NeurIPS*, 1997.

[269] X. Jin, X. Yuan, J. Feng, and S. Yan, "Training Skinny Deep Neural Networks with Iterative Hard Thresholding Methods," *arXiv preprint arXiv:1607.05423*, 2016.

[270] Y. Guo, A. Yao, and Y. Chen, "Dynamic Network Surgery for Efficient DNNs," in *NeurIPS*, 2016.

[271] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *ICRL*, 2019.

[272] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *ICLR*, 2016.

[273] S. Chou, F. Kjolstad, and S. Amarasinghe, "Format Abstraction for Sparse Tensor Algebra Compilers," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 123:1–123:30, nov 2018.

[274] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Workshop on Irregular Applications: Architectures and Algorithms*. ACM, 2015, pp. 1–7.

[275] K. Hegde, H. Asghari-Moghaddam, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Fletcher, "ExTensor: An Accelerator for Sparse Tensor Algebra," in *MICRO*, 2019.

[276] N. Sato and W. F. Tinney, "Techniques for Exploiting the Sparsity of the Network Admittance Matrix," *IEEE Transactions on Power Apparatus and Systems*, vol. 82, no. 69, pp. 944–950, 1963.

[277] A. Buluç and J. R. Gilbert, "On the representation and multiplication of hypersparse matrices," in *IEEE International Symposium on Parallel and Distributed Processing, (IPDPS).*, apr 2008, pp. 1–11.

[278] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, jul 1948.

[279] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *MICRO*, 2019.

[280] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *ISCA*, 2016.

[281] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *ISCA*, 2016.

[282] A. Delmas Lascorz, P. Judd, D. M. Stuart, Z. Poulos, M. Mahmoud, S. Sharify, M. Nikolic, K. Siu, and A. Moshovos, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *ASPLOS*, 2019.

[283] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size," *ICLR*, 2017.

[284] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *CVPR*, June 2018.

[285] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[286] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, "Deep layer aggregation," in *CVPR*, June 2018.

[287] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation," in *NeurIPS*, 2014.

[288] V. Lebedev, Y. Ganin, M. Rakhuba1, I. Oseledets, and V. Lempitsky, "Speeding-Up Convolutional Neural Networks Using Fine-tuned CP-Decomposition," *ICLR*, 2015.

[289] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications," in *ICLR*, 2016.

[290] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *ICLR*, 2017.

[291] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *ICML*, 2017.

[292] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," *ICLR*, 2019.

[293] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *ICML*, 2018.

[294] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," *CoRR*, vol. abs/1807.11626, 2018. [Online]. Available: http://arxiv.org/abs/1807.11626

[295] L.-C. Chen, M. D. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, "Searching for efficient multi-scale architectures for dense image prediction," in *NeurIPS*, 2018.

[296] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *CVPR*, June 2018.

[297] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *CVPR*, June 2018.

[298] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *AAAI*, 2018.

[299] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical Representations for Efficient Architecture Search," in *ICLR*, 2018.

[300] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," in *ICML 2018 AutoML Workshop*, Jul. 2018.

[301] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.

[302] C. Liu, L.-C. Chen, F. Schroff, H. Adam, W. Hua, A. Yuille, and F.-F. Li, "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation," in *CVPR*, 2019.

[303] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, and F. Hutter, "Towards automatically-tuned neural networks," in *Proceedings of the Workshop on Automatic Machine Learning*, ser. Proceedings of Machine Learning Research, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., vol. 64. New York, New York, USA: PMLR, 24 Jun 2016, pp. 58–65.

[304] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *CVPR*, June 2018.

[305] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, "Chamnet: Towards efficient network design through platform-aware model adaptation," in *CVPR*, June 2019.

[306] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," *CoRR*, vol. abs/1812.03443, 2018. [Online]. Available: http://arxiv.org/abs/1812.03443

[307] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *ICLR*, 2019.

[308] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *ICCV*, 2019.

[309] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model Compression," in *SIGKDD*, 2006.

[310] L. Ba and R. Caurana, "Do Deep Nets Really Need to be Deep?" *NeurIPS*, 2014.

[311] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in *NeurIPS Deep Learning Workshop*, 2014.

[312] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for Thin Deep Nets," *ICLR*, 2015.

[313] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," in *ICLR*, 2018.

[314] T.-J. Yang and V. Sze, "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," in *IEDM*, 2019.

[315] D. Kim, J. Kung, S. Chai, S. Yalamanchili, and S. Mukhopadhyay, "Neurocube: A programmable digital neuromorphic architecture with high-density 3D memory," in *ISCA*, 2016.

[316] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *ASPLOS*, 2017.

[317] K. Ueyoshi, K. Ando, K. Hirose, S. Takamaeda-Yamazaki, M. Hamada, T. Kuroda, and M. Motomura, "QUEST: Multi-Purpose Log-Quantized DNN Inference Engine Stacked on 96-MB 3-D SRAM Using Inductive Coupling Technology in 40-nm CMOS," *IEEE Journal of Solid-State Circuits*, pp. 1–11, 2018.

[318] M. M. S. Aly, T. F. Wu, A. Bartolo, Y. H. Malviya, W. Hwang, G. Hills, I. Markov, M. Wootters, M. M. Shulaker, H.-S. P. Wong *et al.*, "The N3XT approach to energy-efficient abundant-data computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 19–48, 2019.

[319] D. Keitel-Schulz and N. Wehn, "Embedded DRAM development: Technology, physical design, and application issues," *IEEE Des. Test. Comput.*, vol. 18, no. 3, pp. 7–15, 2001.

[320] A. Chen, "A review of emerging non-volatile memory (nvm) technologies and applications," *Solid-State Electronics*, vol. 125, pp. 25–38, 2016.

[321] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, *Flash memories*. Springer Science & Business Media, 2013.

[322] O. Golonzka, J.-G. Alzate, U. Arslan, M. Bohr, P. Bai, J. Brockman, B. Buford, C. Connor, N. Das, B. Doyle *et al.*, "MRAM as embedded non-volatile memory solution for 22FFL FinFET technology," in *IEDM*, 2018.

[323] S.-S. Sheu, M.-F. Chang, K.-F. Lin, C.-W. Wu, Y.-S. Chen, P.-F. Chiu, C.-C. Kuo, Y.-S. Yang, P.-C. Chiang, W.-P. Lin *et al.*, "A 4Mb embedded SLC resistive-RAM macro with 7.2 ns read-write random-access time and 160ns MLC-access capability," in *ISSCC*, 2011.

[324] O. Golonzka, U. Arslan, P. Bai, M. Bohr, O. Baykan, Y. Chang, A. Chaudhari, A. Chen, N. Das, C. English *et al.*, "Non-Volatile RRAM Embedded into 22FFL FinFET Technology," in *2019 Symposium on VLSI Technology*, 2019.

[325] G. De Sandre, L. Bettini, A. Pirola, L. Marmonier, M. Pasotti, M. Borghi, P. Mattavelli, P. Zuliani, L. Scotti, G. Mastracchio *et al.*, "A 90nm 4Mb embedded phase-change memory with 1.2 V 12ns read access time and 1MB/s write throughput," in *ISSCC*, 2010.

[326] J. T. Pawlowski, "Vision of Processor-Memory Systems," Keynote at MICRO-48, 2015. [Online]. Available: https://www.microarch.org/micro48/files/slides/Keynote-III.pdf

[327] J. Jeddeloh and B. Keeth, "Hybrid memory cube new DRAM architecture increases density and performance," in *Symp. on VLSI*, 2012.

[328] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *ISCA*, 2016.

[329] D. Ditzel, T. Kuroda, and S. Lee, "Low-cost 3d chip stacking with thruchip wireless connections," in *Hot Chips-A Symposium on High Performance Chips (Aug. 2014)*, 2014.

[330] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, and P. Deaville, "In-memory computing: Advances and prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.

[331] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "An Always-On 3.8 μJ/86% CIFAR-10 Mixed-Signal Binary CNN Processor With All Memory on Chip in 28-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 1, pp. 158–172, 2018.

[332] D. Bankman, J. Messner, A. Gural, and B. Murmann, "RRAM-Based In-Memory Computing for Embedded Deep Neural Networks," in *Asilomar Conference on Signals, Systems, and Computers*, 2019.

[333] S. Ma, D. Brooks, and G.-Y. Wei, "A binary-activation, multi-level weight RNN and training algorithm for processing-in-memory inference with eNVM," *arXiv preprint arXiv:1912.00106*, 2019.

[334] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[335] L. Wilson, "International technology roadmap for semiconductors (ITRS)," *Semiconductor Industry Association*, 2013.

[336] J. Liang and H.-S. P. Wong, "Cross-point memory array without cell selectors—Device characteristics and data storage pattern dependencies," *IEEE Transactions on Electron Devices*, vol. 57, no. 10, pp. 2531–2538, 2010.

[337] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *ISCA*, 2016.

[338] X. Guo, F. M. Bayat, M. Bavandpour, M. Klachko, M. Mahmoodi, M. Prezioso, K. Likharev, and D. Strukov, "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded nor flash memory technology," in *IEDM*, 2017.

[339] S. B. Eryilmaz, S. Joshi, E. Neftci, W. Wan, G. Cauwenberghs, and H.-S. P. Wong, "Neuromorphic architectures with electronic synapses," in *ISQED*, 2016.

[340] W. Haensch, T. Gokmen, and R. Puri, "The Next Generation of Deep Learning Hardware: Analog Computing," *Proceedings of the IEEE*, pp. 1–15, 2018.

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

[342] T. Gokmen and Y. Vlasov, "Acceleration of deep neural network training with resistive cross-point devices: design considerations," *Frontiers in neuroscience*, vol. 10, p. 333, 2016.

[343] W. Chen, K. Li, W. Lin, K. Hsu, P. Li, C. Yang, C. Xue, E. Yang, Y. Chen, Y. Chang, T. Hsu, Y. King, C. Lin, R. Liu, C. Hsieh, K. Tang, and M. Chang, "A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *ISSCC*, 2018.

[344] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "NAND-Net: Minimizing Computational Complexity of In-Memory Processing for Binary Neural Networks," in *HPCA*, 2019.

[345] Lu, Darsen, "Tutorial on Emerging Memory Devices," 2016.

[346] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *HPCA*, 2017.

[347] T. Chou, W. Tang, J. Botimer, and Z. Zhang, "CASCADE: Connecting RRAMs to Extend Analog Dataflow In An End-To-End In-Memory Processing Paradigm," in *MICRO*, 2019.

[348] F. Su, W.-H. Chen, L. Xia, C.-P. Lo, T. Tang, Z. Wang, K.-H. Hsu, M. Cheng, J.-Y. Li, Y. Xie *et al.*, "A 462GOPs/J RRAM-Based Nonvolatile Intelligent Processor for Energy Harvesting IoE System Featuring Nonvolatile Logics and Processing-In-Memory," in *VLSI Technology, 2017 Symposium on*, 2017, pp. T260–T261.

[349] M. Prezioso, F. Merrikh-Bayat, B. Hoskins, G. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[350] J. Zhang, Z. Wang, and N. Verma, "A machine-learning classifier implemented in a standard 6T SRAM array," in *Symp. on VLSI*, 2016.

[351] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A Mixed-Signal Binarized Convolutional-Neural-Network Accelerator Integrating dense Weight Storage and Multiplication for Reduced Data Movement," in *Symp. on VLSI*, 2018.

[352] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *ISSCC*, 2018, pp. 488–490.

[353] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, "A Multi-Functional In-Memory Inference Processor Using a Standard 6T SRAM Array," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, 2018.

[354] M. Kang, S. Lim, S. Gonugondla, and N. R. Shanbhag, "An In-Memory VLSI Architecture for Convolutional Neural Networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2018.

[355] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *MICRO*, 2017.

[356] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "DRISA: A DRAM-based Reconfigurable In-Situ Accelerator," in *MICRO*, 2017.

[357] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *ISCA*, 2018.

[358] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "ComputeDRAM: In-memory compute using off-the-shelf DRAMs," in *MICRO*, 2019.

[359] Wu, Yannan N. and Sze, Vivienne and Emer, Joel S. , "An Architecture-Level Energy and Area Estimator for Processing-In-Memory Accelerator Designs," in *ISPASS*, 2020.

[360] H. Jia, Y. Tang, H. Valavi, J. Zhang, and N. Verma, "A microprocessor implemented in 65nm cmos with configurable and bit-scalable accelerator for programmable in-memory computing," *arXiv preprint arXiv:1811.04047*, 2018.

[361] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, "A 351TOPS/W and 372.4GOPS Compute-in-Memory SRAM Macro in 7nm FinFET CMOS for Machine-Learning Applications," in *ISSCC*, 2020.

[362] J. Zhang, Z. Wang, and N. Verma, "A matrix-multiplying ADC implementing a machine-learning classifier directly with data conversion," in *ISSCC*, 2015.

[363] E. H. Lee and S. S. Wong, "A 2.5 GHz 7.7 TOPS/W switched-capacitor matrix multiplier with co-designed local memory in 40nm," in *ISSCC*, 2016.

[364] R. LiKamWa, Y. Hou, J. Gao, M. Polansky, and L. Zhong, "RedEye: analog ConvNet image sensor architecture for continuous mobile vision," in *ISCA*, 2016.

[365] A. Wang, S. Sivaramakrishnan, and A. Molnar, "A 180nm CMOS image sensor with on-chip opto-electronic image compression," in *CICC*, 2012.

[366] H. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. Molnar, "ASP Vision: Optically Computing the First Layer of Convolutional Neural Networks using Angle Sensitive Pixels," in *CVPR*, 2016.

[367] A. Suleiman and V. Sze, "Energy-efficient HOG-based object detection at 1080HD 60 fps with multi-scale support," in *SiPS*, 2014.

[368] Q. Cheng, J. Kwon, M. Glick, M. Bahadori, L. P. Carloni, and K. Bergman, "Silicon photonics codesign for deep learning," *Proceedings of the IEEE*, pp. 1–22, 2020.

[369] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature Photonics*, vol. 11, no. 7, p. 441, 2017.

[370] R. Hamerly, L. Bernstein, A. Sludds, M. Soljačić, and D. Englund, "Large-scale optical neural networks based on photoelectric multiplication," *Physical Review X*, vol. 9, no. 2, pp. 021–032, 2019.

[371] A. N. Tait, M. A. Nahmias, B. J. Shastri, and P. R. Prucnal, "Broadcast and weight: an integrated network for scalable photonic spike processing," *Journal of Lightwave Technology*, vol. 32, no. 21, pp. 4029–4041, 2014.

[372] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, and A. Ozcan, "All-optical machine learning using diffractive deep neural networks," *Science*, vol. 361, no. 6406, pp. 1004–1008, 2018.

[373] L. Bernstein, A. Sludds, R. Hamerly, V. Sze, J. Emer, and D. Englund, "Digital Optical Neural Networks for Large-Scale Machine Learning," in *Conference on Lasers and Electro-Optics(CLEO)*, 2020.

Author Biographies

Vivienne Sze received a B.A.Sc. (Hons) degree in Electrical Engineering from the University of Toronto, Toronto, ON, Canada, in 2004, and S.M. and Ph.D. degrees in Electrical Engineering from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2006 and 2010, respectively. In 2011, she received the Jin-Au Kong Outstanding Doctoral Thesis Prize in Electrical Engineering at MIT.

She is an Associate Professor at MIT in the Electrical Engineering and Computer Science Department. Her research interests include energy-aware signal processing algorithms, and low-power circuit and system design for portable multimedia applications including computer vision, deep learning, autonomous navigation, image processing, and video compression. Prior to joining MIT, she was a Member of Technical Staff in the Systems and Applications R&D Center at Texas Instruments (TI), Dallas, TX, where she designed low-power algorithms and architectures for video coding. She also represented TI in the JCT-VC committee of ITU-T and ISO/IEC standards body during the development of High Efficiency Video Coding (HEVC), which received a Primetime Engineering Emmy Award. Within the committee, she was the primary coordinator of the core experiment on coefficient scanning and coding, and she chaired/vice-chaired several ad hoc groups on entropy coding. She is a co-editor of *High Efficiency Video Coding (HEVC): Algorithms and Architectures* (Springer, 2014).

Prof. Sze is a recipient of the inaugural ACM-W Rising Star Award, the 2019 Edgerton Faculty Achievement Award at MIT, the 2018 Facebook Faculty Award, the 2018 & 2017 Qualcomm Faculty Award, the 2018 & 2016 Google Faculty Research Award, the 2016 AFOSR Young Investigator Research Program (YIP) Award, the 2016 3M Non-Tenured Faculty Award, the 2014 DARPA Young Faculty Award, the 2007 DAC/ISSCC Student Design Contest Award and a co-recipient of the 2018 VLSI Best Student Paper Award, the 2017 CICC Outstanding Invited Paper Award, the 2016 IEEE Micro Top Picks Award, and the 2008 A-SSCC Outstanding Design Award. She currently serves on the technical program committee for the International Solid-State Circuits Conference (ISSCC) and the SSCS Advisory Committee (AdCom). She has served on the technical program committees for VLSI Circuits Symposium, Micro and the Conference on Machine Learning and Systems (MLSys), as a guest editor for the IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), and as a Distinguished Lecturer for the IEEE Solid-State Circuits Society (SSCS). Prof. Sze was Program Co-chair of the 2020 Conference on Machine Learning and Systems (MLSys) and teaches the MIT Professional Education course on Designing Efficient Deep Learning Systems.

For more information about Prof. Sze's research, please visit the Energy-Efficient Multimedia Systems group at MIT: `http://www.rle.mit.edu/eems/`.

Yu-Hsin Chen received a B. S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2009, and M. S. and Ph.D. degrees in Electrical Engineering and Computer Science (EECS) from Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2013 and 2018, respectively. He received the 2018 Jin-Au Kong Outstanding Doctoral Thesis Prize in Electrical Engineering at MIT and the 2019 ACM SIGARCH/IEEE-CS TCCA Outstanding Dissertation Award. He is currently a Research Scientist at Facebook focusing on hardware/software co-design to enable on-device AI for AR/VR systems. Previously, he was a Research Scientist in Nvidia's Architecture Research Group.

He was the recipient of the 2015 Nvidia Graduate Fellowship, 2015 ADI Outstanding Student Designer Award, and 2017 IEEE SSCS Predoctoral Achievement Award. His work on the dataflows for CNN accelerators was selected as one of the Top Picks in Computer Architecture in 2016. He also co-taught a tutorial on "Hardware Architectures for Deep Neural Networks" at MICRO-49, ISCA2017, MICRO-50, and ISCA2019.

Tien-Ju Yang received a B. S. degree in Electrical Engineering from National Taiwan University (NTU), Taipei, Taiwan, in 2010, and an M. S. degree in Electronics Engineering from NTU in 2012. Between 2012 and 2015, he worked in the Intelligent Vision Processing Group, MediaTek Inc., Hsinchu, Taiwan, as an engineer. He is currently a Ph.D. candidate in Electrical Engineering and Computer Science at Massachusetts Institute of Technology, Cambridge, MA, working on energy-efficient deep neural network design. His research interest spans the area of deep learning, computer vision, machine learning, image/video processing, and VLSI system design. He won first place in the 2011 National Taiwan University Innovation Contest. He also co-taught a tutorial on "Efficient Image Processing with Deep Neural Networks" at IEEE International Conference on Image Processing 2019.

Joel S. Emer received B.S. (Hons.) and M.S. degrees in Electrical Engineering from Purdue University, West Lafayette, IN, USA, in 1974 and 1975, respectively, and a Ph.D. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 1979.

He is currently a Senior Distinguished Research Scientist with Nvidia's Architecture Research Group, Westford, MA, USA, where he is responsible for exploration of future architectures and modeling and analysis methodologies. He is also a Professor of the Practice at the Massachusetts Institute of Technology, Cambridge, MA, USA. Previously, he was with Intel, where he was an Intel Fellow and the Director of Microarchitecture Research. At Intel, he led the VSSAD Group, which he had previously been a member of at Compaq and Digital Equipment Corporation. Over his career, he has held various research and advanced development positions investigating processor micro-architecture and developing performance modeling and evaluation techniques. He has made architectural contributions to a number of VAX, Alpha, and X86 processors and is recognized as one of the developers of the widely employed quantitative approach to processor performance evaluation. He has been recognized for his contributions in the advancement of simultaneous multithreading technology, processor reliability analysis, cache organization, pipelined processor organization and spatial architectures for deep learning.

Dr. Emer is a Fellow of the ACM and IEEE and a member of the NAE. He has been a recipient of numerous public recognitions. In 2009, he received the Eckert-Mauchly Award for lifetime contributions in computer architecture. He received the Purdue University Outstanding Electrical and Computer Engineer Alumni Award and the University of Illinois Electrical and Computer Engineering Distinguished Alumni Award in 2010 and 2011, respectively. His 1996 paper on simultaneous multithreading received the ACM/SIGARCH-

IEEE-CS/TCCA: Most Influential Paper Award in 2011. He was named to the ISCA and MICRO Halls of Fame in 2005 and 2015, respectively. He has had six papers selected for the IEEE Micro's Top Picks in Computer Architecture in 2003, 2004, 2007, 2013, 2015, and 2016. He was the Program Chair of the International Symposium on Computer Architecture (ISCA) in 2000 and the International Symposium on Microarchitecture (MICRO) in 2017.