How to Evaluate Deep Neural Network Processors

TOPS/W (Alone) Considered Harmful

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, Joel S. Emer

***Abstract*—A significant amount of specialized hardware has been developed for processing deep neural networks (DNNs) in both academia and industry. This article aims to highlight the key concepts required to evaluate and compare these DNN processors. We will discuss existing challenges such as the flexibility and scalability needed to support a wide range of neural networks, as well as design considerations for both the DNN processors and the DNN models themselves. We will also describe specific metrics that can be used to evaluate and compare existing solutions.**

This article is based on the tutorial entitled "How to Understand and Evaluate Deep Learning Processors" that was given at the 2020 International Solid-State Circuits Conference (ISSCC), and excerpts from the book on "Efficient Processing of Deep Neural Networks".

I. Introduction

Over the past few years, there has been a significant amount of research on enabling the efficient processing of deep neural networks (DNNs). The challenge of efficient DNN processing depends on balancing multiple objectives:

- high performance (including accuracy) and efficiency (including cost),
- enough flexibility to cater to a wide and rapidly changing range of workloads, and
- good integration with existing software frameworks.

DNN computations are composed of several processing layers (Figure 1), where for many layers the main computation is a weighted sum; in other words, the main computation for DNN processing is often a multiply-accumulate (MAC) operation. The arrangement of the MAC operations within a layer is defined by the layer shape; for instance, Table I and Figure 2 highlight the shape parameters for layers used in convolutional neural networks (CNNs), which are a popular type of DNN. Because the shape parameters can vary across layers, DNNs come in a wide variety of shapes and sizes depending on the application.[1] This variety is one of the motivations for flexibility, and causes the objectives listed above to be highly interrelated.

Figure 3 illustrates the hardware architecture of a typical DNN processor, which is comprised of an array of processing elements (PEs), where each PE contains MAC units to perform the computation and optionally some local storage, and an inter-PE communication network. The entire PE array is also connected via an on-chip network to a large buffer (GLB), which in turn is connected off-chip to DRAM memory. DNN

[1]The DNN research community often refers to the shape and size of a DNN as its 'network architecture'. However, to avoid confusion with the use of the word 'architecture' by the hardware community, we will talk about 'DNN models' and their shape and size in this article.

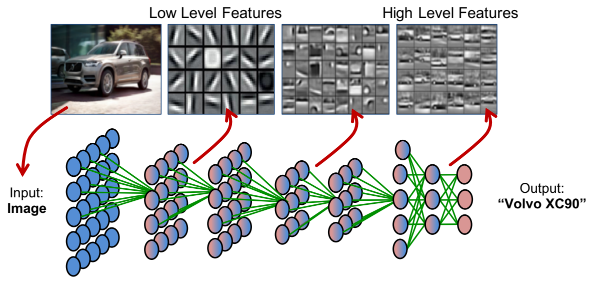


Fig. 1. Example of image classification using a deep neural network. (Figure adapted from [1].) The deep neural network is composed of multiple layers, and the number of layer is referred to as the depth of the network. Note that the extracted features go from low level to high level as we go deeper into the network.

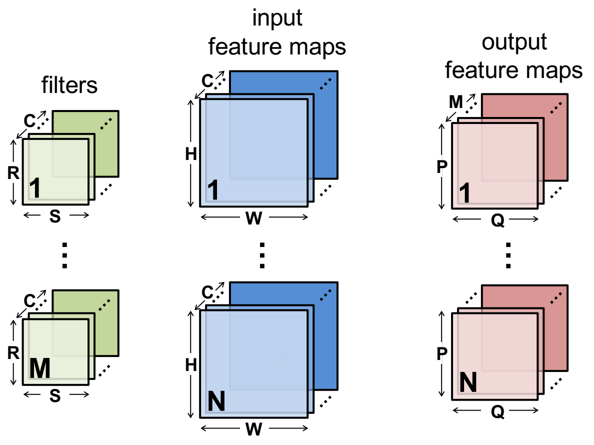


Fig. 2. The shape parameters of layers used in DNNs.

Shape Parameter	Description
N	batch size of 3-D feature map
M	# of 3-D filters / # of channels of output feature map (output channels)
C	# of channels of filter / input feature map (input channels)
H/W	spatial height/width of input feature map
R/S	spatial height/width of filter
P/Q	spatial height/width of output feature map

TABLE I
THE SHAPE PARAMETERS FOR LAYERS USED IN DNNS.

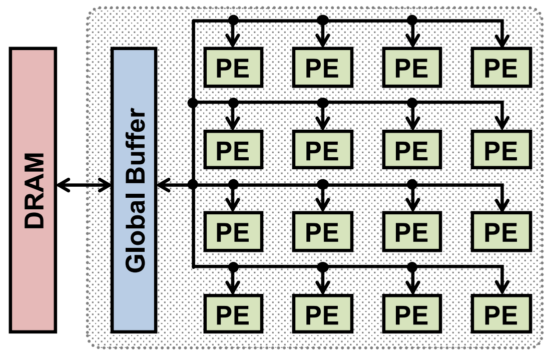


Fig. 3. Typical hardware architecture of a DNN processor.

processor designs tend to vary in terms of the number of PEs, the number of levels in the memory hierarchy, the amount of storage at each level, and how the PEs and memory are connected through the on-chip network.

Given the combination of such hardware and the associated DNN models, it is important to discuss the key metrics that one should consider when comparing and evaluating the strengths and weaknesses of different designs. They can also be used to evaluate proposed techniques, and 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.[2] 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 article, we will

- discuss the importance of each of these metrics;
- breakdown the factors that affect each metric. When feasible, we will 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; and
- specify what should be reported for a given metric to enable proper evaluation.

Finally, we will highlight tools that can be used to evaluate some of these metrics early in the design process (to enable rapid design exploration), and 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.

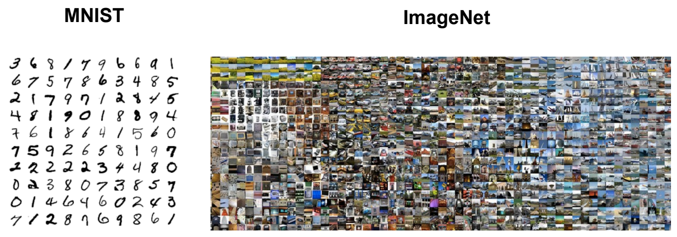


Fig. 4. MNIST (10 classes, 60k training, 10k testing) [4] versus ImageNet (1000 classes, 1.3M training, 100k testing) [3] dataset.

II. 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 true positives, false positives, and false negatives.

Factors that affect accuracy include the difficulty of the task and dataset.[3] For instance, classification on the ImageNet dataset [3]) is much more difficult than on the MNIST dataset [4] (Figure 4), and object detection is usually 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.[4] 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 [5], 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 architectures, and cloud platforms for both training and inference of DNNs [6].[5] The suite includes various types of DNNs (e.g., convolutional neural networks (CNNs), recurrent neural networks (RNNs), etc.) for a variety of tasks including image classification, object identification, translation, speech-

[2]Note that TOPS/W efficiency is typically reported at (and often along with) the peak performance in tera-operations per second (TOPS), which gives the maximum efficiency since it assumes maximum utilization and thus maximum amortization of overhead; however, this does not tell the complete story because processors typically do not operate at their peak TOPS, and their efficiency degrades at lower utilization. It is a well-known challenge to achieve energy-proportional computing, where the efficiency stays constant across performance [2].

[3]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 article.

[4]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.

[5]Earlier DNN benchmarking efforts including DeepBench [7] and Fathom [8] have now been subsumed by MLPerf.

to-text, recommendation, sentiment analysis, and reinforcement learning.

III. 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 often 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.

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 per inference.

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 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 metrics should be reported.[6]

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

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

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.

[6]The phenomenon described here can also be understood using Little's Law [9] 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.

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:

$$\frac{\text{operations}}{\text{second}} = \underbrace{\left(\frac{1}{\frac{\text{cycles}}{\text{operation}}} \times \frac{\text{cycles}}{\text{second}}\right)}_{\text{for a single PE}} \times \text{number of PEs} \times \text{utilization of PEs} \tag{2}$$

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 achieved 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 bit-serial 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 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 reducing 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.

Reducing the area per PE 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), or 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:

$$\text{utilization of PEs} = \frac{\text{number of active PEs}}{\text{number of PEs}} \times \text{utilization of active PEs} \tag{3}$$

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[7]; 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 different layer shapes in a DNN model as explored in [10, 11].

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. The mapper can be thought of as a compiler for the DNN processor [12]. The mapping process on layer-by-layer basis is explored in detail in [13–15]. Additional challenges regarding the flexibility of mapping are discussed in Section VI.

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 [15, 16]. 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 the area allocation between compute (which increases the number of PEs) versus on-chip storage (which increases the utilization of 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 processor design. Eyexam includes and extends the well-known roofline model [17]. The roofline model, as illustrated in Figure 5, relates average bandwidth demand and peak computational ability to performance.

The goal of Eyexam is to provide a fine-grain performance profile for a DNN processor. It is a sequential analysis process that involves seven major steps as shown in Figure 6. The process starts with the assumption that the architecture has infinite processing parallelism, storage capacity and data bandwidth. Therefore, it has infinite performance (as measured in MACs/cycle).

[7]The ratio of active PEs to the total number of PEs can be referred to as the *active PE percentage*.

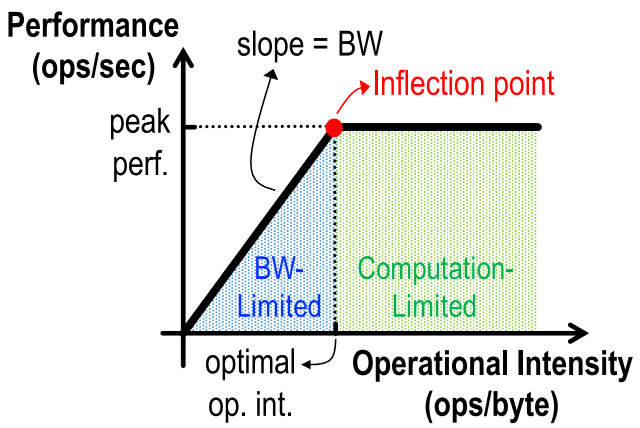


Fig. 5. The roofline model. The peak *operations per second* is indicated by the bold line; when the operation intensity, which dictates by amount of compute per byte of data, is low, the *operations per second* is limited by the data delivery. The design goal is to operate as close as possible to the peak *operations per second* for the operation intensity of a given workload.

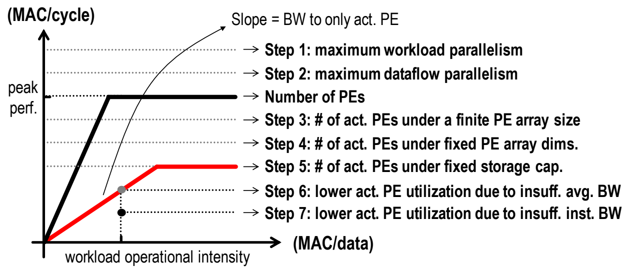


Fig. 6. Impact of Eyexam steps on the roofline model.

For each of the following steps, certain constraints are added to reflect changes in the assumptions on the DNN processor or workload. The associated performance loss can therefore be attributed to that change, and the final performance at one step becomes the upper-bound for the next step.

Step 1 (Layer Shape and Size): In this first step, we look at the impact of the workload constraint, so that there is all spatial (i.e., parallel) processing, and no temporal (i.e., serial) processing. Therefore, the performance upper bound is determined by the finite size of the workload (i.e., the number of MAC operations in the layer).

Step 2 (Dataflow): In this step, we specify the dataflow, which determines the order of operations and where data is stored and reused, and examine the impact of this architectural constraint. Imposing a dataflow forces a serialization of processing and reduces the performance upper bound, which is the maximum parallelism of the dataflow.

Step 3 (Number of PEs): In this step, we restrict the system to a finite number of PEs, and look at the impact of this architectural constraint. A finite number of PEs can degrade performance whenever there is more parallel work to do than that number of PEs. In addition, some of the PEs will be idle (i.e., reduce number of active PEs) if the amount of work is not an integer multiple of the number of PEs (i.e., the work cannot be equally divided amongst the PEs).

Step 4 (Physical dimensions of the PE array): In this step, we consider the physical dimensions of the PE array and data delivery network (e.g., arranging 12 PEs as 3×4, 2×6 or 4×3, etc.). The spatial partitioning and associated on-chip network is often constrained per data type (e.g., input activation or filter weight), which can cause additional performance loss because the required data cannot be delivered to the PEs.

Step 5 (Storage Capacity): In this step, we consider the impact of making the buffer storage finite capacity. Lack of storage can limit parallelism when there is insufficient storage to hold intermediate results, and thus degrade performance.

Step 6 (Data Bandwidth): In this step, we consider the impact of a finite bandwidth for delivering data across the different levels of the memory hierarchy. The amount of data that needs to be transferred between each level of the memory hierarchy for each step of computation and the available data bandwidth determines if the PEs can be kept busy.

Step 7 (Varying Data Access Patterns): In this step, we consider the impact of bandwidth varying across time due to the dynamically changing data access patterns (Step 6 only addresses average bandwidth). This includes ramp up time to initially load values, and ramp down to drain values after completion. There exist many common solutions to address this issue, including using double buffering, but these can increase area or reduce amount of reuse.

Table II summarizes the constraints applied at each step of the Eyexam process.

Up until this point, we have discussion how hardware design decisions impact performance (i.e., throughput and latency). We will now discuss how the choice of DNN model can also have an impact. Specifically, while the number of *operations per inference* in Equation 1 depends on the DNN model, the *operations per second* depends on both the DNN model and the hardware. Thus, designing DNN models with efficient layer shapes (also referred to efficient network architectures) (e.g., MobileNet [18]) 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 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. A variety of hardware architectures have been proposed to exploit sparsity [19–21].

Processing only effectual MAC operations can increase the *(total) operations per second* by increasing the *(total) operations per cycle*.[8]. 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 hardware complexity and overhead. 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 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 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.

As the amount of sparsity increases (i.e., the number of *effectual operations out of (total) operations* decreases), the *operations per cycle* increases, as shown in Equation 4; this subsequently increases *operations per second*, as shown in Equation 2:

$$\begin{aligned}\frac{\text{operations}}{\text{cycle}} &= \frac{\text{effectual operations + unexploited ineffectual operations}}{\text{cycle}}\\ &\times \frac{\text{effectual operations}}{\text{effectual operations + unexploited ineffectual operations}}\\ &\times \frac{1}{\frac{\text{effectual operations}}{\text{operations}}}\end{aligned} \tag{4}$$

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 increase the area 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 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) 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 active 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*

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

Step	Constraint	Type	New Performance Bound	Reason for Performance Loss
1	Layer Size and Shape	Workload	Max workload parallelism	Finite workload size
2	Dataflow loop nest	Architectural	Max dataflow parallelism	Restricted dataflows defined by loop nest
3	Number of PEs	Architectural	Max PE parallelism	Additional restriction to mappings due to shape fragmentation
4	Physical dimensions of PEs array	Architectural	Number of active PEs	Additional restriction to mappings due to shape fragmentation for *each* dimension
5	Fixed Storage Capacity	Architectural	Number of active PEs	Additional restriction to mappings due to storage of intermediate data (depends on dataflow)
6	Fixed Data Bandwidth	Microarchitectural	Max data bandwidth to active PEs	Insufficient average bandwidth to active PEs
7	Varying Data Access Patterns	Microarchitectural	Actual measured performance	Insufficient instant bandwidth to active PEs

TABLE II
SUMMARY OF STEPS IN EYEXAM.

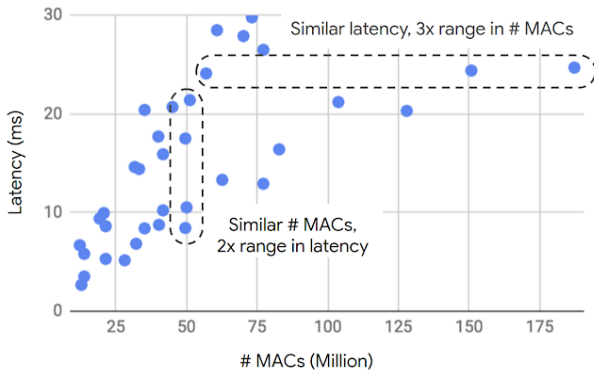


Fig. 7. The number of MAC operations in various DNN models versus latency measured on Pixel phone. Clearly, the number of MAC operations is not a good predictor of latency. (Figure from [27])

second, as shown in Equation 2. Note, however, that if *multiple* levels of precision need to be supported, additional hardware is required [22], which can, once again, increase the critical path and also increase the area of the PE, both of which can reduce the *operations per second*, as shown in Equation 2.

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

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 (see Figure 7), and it is often more effective to design efficient DNN models with hardware in the loop. Various works have proposed techniques for designing DNN models with hardware in the loop [23–26].

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 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.

IV. 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 and energy consumption is reported as 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:

$$\frac{\text{inferences}}{\text{second}} \leq \text{Max}\left(\frac{\text{joules}}{\text{second}}\right) \times \frac{\text{inferences}}{\text{joule}} \quad (5)$$

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.

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	✓		

TABLE III
CLASSIFICATION OF FACTORS THAT AFFECT INFERENCES PER SECOND.

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

$$\frac{\text{inferences}}{\text{joule}} = \frac{\text{operations}}{\text{joule}} \times \frac{1}{\frac{\text{operations}}{\text{inference}}}, \quad (6)$$

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

$$\text{Energy}_{total} = \text{Energy}_{data} + \text{Energy}_{MAC} \quad (7)$$

For each component the joules per operation[9] is computed as

$$\frac{\text{joules}}{\text{operation}} = \alpha \times C \times {V_{DD}}^2, \quad (8)$$

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 8). Furthermore, the switching capacitance increases with the distance 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 a 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 processors as explored in [16, 28]. In addition, advanced device and memory technologies can be used to reduce the switching capacitance between compute and memory, for instance by enabling in-memory computing [29, 30].

[9]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

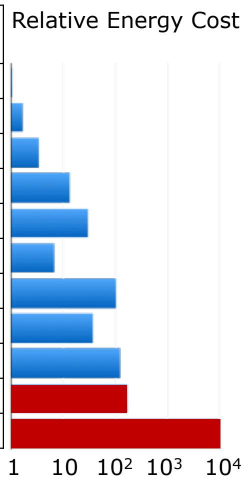


Fig. 8. The energy consumption for various arithmetic operations and memory accesses in a 45nm process. The relative energy cost (computed relative to the 8b add) is shown on a log scale. The energy consumption of data movement (red) is significantly higher than arithmetic operations (blue). (Figure adapted from [31])

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 article. 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 8. 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), or tensor instructions), a single instruction can be used to initiate multiple operations.

Similar to the throughput metric discussed in Section III, 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 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.

$$\frac{\text{operations}}{\text{joule}} = \frac{\text{effectual operations + unexploited ineffectual operations}}{\text{joule}} \times \frac{\text{effectual operations}}{\text{effectual operations + unexploited ineffectual operations}} \times \frac{1}{\frac{\text{effectual operations}}{\text{operations}}} \tag{9}$$

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 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.

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. Various tools can be used to help with energy estimation, as shown in Figure 9.

V. 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[10]. This is important from both an industry and a research perspective as it dictates 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., 45nm CMOS), which constrains the amount of on-chip storage and amount of compute (e.g., the number of PEs for DNN processors, 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, one can report the amount of on-chip memory (e.g, storage capacity of the global buffer) and compute (e.g., number of PEs) 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) [32] to connect to off-chip DRAM, NVLink [33] 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) and 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.

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

[10]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. In addition, there is 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 DNN processors have a higher design cost (after amortization) than off-the-shelf CPU and GPU. We consider anything beyond this, e.g. the economics of the semiconductor business, including how to price platforms, is outside the scope of this article.

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 5), as each byte of on-chip memory also increases cost.

VI. Flexibility

The merit of a DNN processor 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 of DNN models are diverse and evolving rapidly. For instance, a DNN processor 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; 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 processor presents a complex trade-off with processor 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. 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), different levels of precision (i.e., different number of bits across layers and data types), and different degrees of sparsity (i.e., number of zeros in the data). There are also different types of DNN layers and computations 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. 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.).

VII. Scalability

Scalability has become increasingly important due to the wide use cases for DNNs. This is demonstrated by 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) [34] or even wafer-scale chips [35]. Scalability refers to how well a design can be scaled up to achieve higher performance (i.e., latency and 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 performance would scale linearly and proportionally with the number of PEs. When the problem size (e.g., the batch size) is held constant, this is referred to as strong scaling, and is the more challenging type of scaling. On the other hand, scaling performance while allowing the problem size to increase (e.g., by increasing batch size) is called weak scaling and is also an important objective in some situations.

Similarly, the energy efficiency would also improve with more on-chip storage, however, this would be likely be non-linear (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* and *inferences per joule per cost* changes with scale. For instance, if throughput increases linearly with number of PEs (with proportional scaling of all storage), then the *inferences per second per cost* could 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. On the other hand, inferences per joule per cost might remain constant or even improve as a consequence of more sharing of data by multiple PEs.

In summary, to understand the scalability of a DNN processor 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.

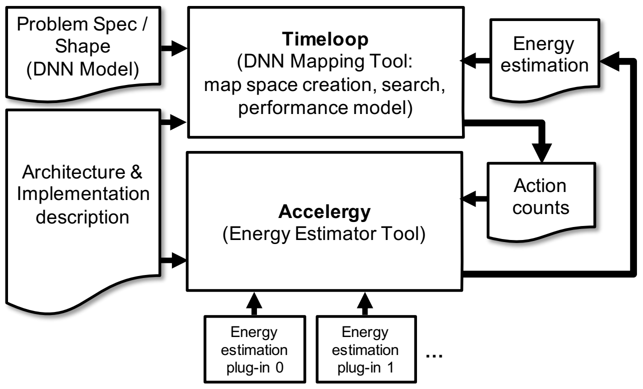
VIII. Interplay between different metrics

It is important that all metrics are accounted for in order to fairly evaluate the design trade-offs. For instance, without the accuracy given for a specific dataset and task, one could run a simple DNN model 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 MACs 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[11] and how many images were tested.

Clearly there are many important metrics to consider when designing DNN processors. At the same time, the design space for DNN processors is very large. As a result, it would helpful to have the ability to rapidly explore the design space early in the design process, and be able to accurately estimate the various metrics for proposed designs. An accurate estimation requires proper consideration of how properties of the hardware such as mapping, and properties of the workload such as DNN model shape, precision and sparsity, impact metrics such as throughput and energy efficiency. One example tool set that allows one to perform rapid exploration and evaluation is the combination of Timeloop [14] and Accelergy [36] as depicted in Figure 9. Both tools accept a template-based specification of a proposed architecture and, given a DNN model description, Timeloop searches for an optimal mapping using an analytical performance model and generates activity counts that allow Accelergy to generate architecture-level energy estimates (all before a detailed RTL description of the design is available). Accelergy also accepts component energy costs, which is especially useful for understanding the impact of new technologies [37].

IX. Summary

In this article, we have presented the various key metrics for evaluating DNN processors, discussed the importance for each metric, their interrelationships and, where appropriate, included equations that can be used to tease apart the factors that contribute to those metrics. We have also shown how those metrics are related to both the hardware design and the



Problem Spec / Shape (DNN Model)
Timeloop (DNN Mapping Tool: map space creation, search, performance model)
Energy estimation
Architecture & Implementation description
Action counts
Accelergy (Energy Estimator Tool)
Energy estimation plug-in 0
Energy estimation plug-in 1
...

Fig. 9. Timeloop [14] with integration of Accelergy [36] as energy estimation model. Timeloop sends projected action counts for a mapping to Accelergy and receives an energy estimation to guide its search. Accelergy plug-ins allow for customization of component energy estimation. These tools are available at http://accelergy.mit.edu/tutorial.html.

DNN models, and highlight why hardware/model co-design is important. Finally, given those metrics 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) the 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.

Portions of this article are based on our book entitled "Efficient Processing of Deep Neural Network" [38]. This excerpt has described various metrics that are important for evaluating DNN processor. The remainder of the book expands on how to design DNN processors and DNN models that optimize for these metrics.

References

[1] 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.
[2] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.
[3] 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.
[4] C. J. B. Yann LeCun, Corinna Cortes, "THE MNIST DATABASE of handwritten digits," http://yann.lecun.com/exdb/mnist/.
[5] "Standard Performance Evaluation Corporation(SPEC)," https://www.spec.org/.

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

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

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

[8] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *International Symposium on Workload Characterization (IISWC)*, 2016.

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

[10] 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.

[11] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)* , 2018.

[12] 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.

[13] 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.

[14] 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 *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019.

[15] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *International Symposium on Microarchitecture (MICRO)*, 2019.

[16] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[17] 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.

[18] 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.

[19] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: ineffectual-neuron-free deep neural network computing," in *International Symposium on Computer Architecture (ISCA)*, 2016.

[20] 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 *International Symposium on Computer Architecture (ISCA)*, 2016.

[21] 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 *International Symposium on Computer Architecture (ISCA)*, 2017.

[22] 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.

[23] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks using Energy-Aware Pruning," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[24] 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 *European Conference on Computer Vision (ECCV)*, 2018.

[25] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[26] T.-J. Yang and V. Sze, "Design Considerations for Efficient Deep Neural Networks on Processing-in-Memory Accelerators," in *International Electron Devices Meeting (IEDM)*, 2019.

[27] 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

[28] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

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

[30] 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.

[31] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *International Solid-State Circuits Conference (ISSCC)* , 2014.

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

[33] "NVIDIA NVlink," https://www.nvidia.com/en-us/design-visualization/nvlink-bridges/.

[34] 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 *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

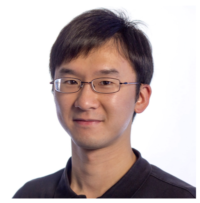
[35] S. Lie, "Wafer Scale Deep Learning," in *Hot Chips 31 Symposium (HCS), 2019 IEEE*, 2019.

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

[37] Y. N. Wu, V. Sze, and J. S. Emer, "An Architecture-Level Energy and Area Estimator for Processing-In-Memory Accelerator Designs," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2020.

[38] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*. Morgan & Claypool Publishers, 2020.

Vivienne Sze (S'04–M'10–SM'16) is an Associate Professor at MIT in the Electrical Engineering and Computer Science Department. Her research focuses on computing systems that enable energy-efficient machine learning, computer vision, and video compression/processing for a wide range of applications, including autonomous navigation, digital health, and the internet of things. She is a recipient or co-recipient of various awards including the AFOSR and DARPA Young Faculty Award, the Edgerton Faculty Award, faculty awards from Google, Facebook, and Qualcomm, the Symposium on VLSI Circuits Best Student Paper Award, the IEEE Custom Integrated Circuits Conference Outstanding Invited Paper Award, and the IEEE Micro Top Picks Award.



Yu-Hsin Chen (S'11) received the B. S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2009, and the M. S. and Ph.D. degrees in Electrical Engineering and Computer Science (EECS) from MIT, Cambridge, MA, in 2013 and 2018, respectively. 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 2018 Jin-Au Kong Outstanding Doctoral Thesis Prize in Electrical Engineering at MIT, 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.

Tien-Ju Yang (S'11) received a B. S. degree in electrical engineering in 2010 and an M. S. degree in electronics engineering in 2012 from National Taiwan University, Taiwan. He is currently a Ph.D. candidate in electrical engineering and computer science at Massachusetts Institute of Technology, USA, working on 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 ICIP2019.

Joel S. Emer (M'73—SM'03—F'04) is a Senior Distinguished Research Scientist with Nvidia's Architecture Research Group and a Professor of the Practice at the Massachusetts Institute of Technology. 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. He has made architectural contributions to a number of VAX, Alpha, and X86 processors and has contributed to the quantitative approach to processor performance evaluation, simultaneous multithreading technology, processor reliability analysis, cache organization, and spatial architectures for deep learning. He is a Fellow of the ACM and member of the NAE. He received the Eckert-Mauchly Award and ECE alumni awards from Purdue University and University of Illinois. He has had six papers selected for the IEEE Micro's Top Picks in Computer Architecture.