INVITED PAPER

Self-Aware Neural Network Systems: A Survey and New Perspective

This article reviews advances in self-aware neural networks and presents a case study of a neural network system that deploys self-awareness to adapt to varying demands in performance and energy.

By Zidong Du, *Member IEEE*, Qi Guo, *Member IEEE*, Yongwei Zhao, Tian Zhi,
Yunji Chen, *Senior Member IEEE*, and Zhiwei Xu, *Senior Member IEEE*

ABSTRACT | Neural network (NN) processors are specially designed to handle deep learning tasks by utilizing multilayer artificial NNs. They have been demonstrated to be useful in broad application fields such as image recognition, speech processing, machine translation, and scientific computing. Meanwhile, innovative self-aware techniques, whereby a system can dynamically react based on continuously sensed information from the execution environment, have attracted attention from both academia and industry. Actually, various self-aware techniques have been applied to NN systems to significantly improve the computational speed and energy efficiency. This article surveys state-of-the-art self-aware NN systems (SaNNSs), which can be achieved at different layers, that is, the architectural layer, the physical layer, and the circuit layer. At the architectural layer, SaNNS can be characterized from a data-centric perspective where different data properties (i.e., data value, data precision, dataflow, and data distribution) are exploited. At the physical layer, various parameters of physical implementation are considered. At the circuit layer, different logics and devices can be used for high efficiency. In fact, the self-awareness of existing SaNNS is still in a preliminary form. We propose a comprehensive SaNNS from a new perspective, that is, the model layer, to exploit more opportunities for high efficiency. The proposed system is called as MinMaxNN, which features model switching and elastic sparsity based on monitored information from the execution environment. The model switching mechanism implies that models (i.e., min and max model) dynamically switch given different inputs for both efficiency and accuracy. The elastic sparsity mechanism indicates that the sparsity of NNs can be dynamically adjusted in each layer for efficiency. The experimental results show that compared with traditional SaNNS, MinMaxNN can achieve 5.64× and 19.66% performance improvement and energy reduction, respectively, without notable loss of accuracy and negative effects on developers' productivity.

KEYWORDS | Self-aware neural network (NN) processors.

I. INTRODUCTION

Recent research in deep neural networks (DNNs) has achieved state-of-the-art performance on a broad range

Manuscript received May 30, 2019; revised November 10, 2019 and February 1, 2020; accepted February 26, 2020. Date of publication March 24, 2020; date of current version June 18, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1003101, Grant 2018AAA0103300, Grant 2017YFA0700900, Grant 2017YFA0700902, and Grant 2017YFA0700901; in part by NSF of China under Grant 61732007, Grant 61432016, Grant 61532016, Grant 61672491, Grant 61602441, Grant 61602446, Grant 61732002, Grant 61702478, and Grant 61732020; in part by the Beijing Natural Science Foundation under Grant JQ18013; in part by the National Science and Technology Major Project under Grant 2018ZX01031102; in part by the Transformation and Transfer of Scientific and Technological Achievements of Chinese Academy of Sciences under Grant KFJ-HGZX-013; in part by the Key Research Projects in Frontier Science of Chinese Academy of Sciences under Grant QYZDB-SSW-JSC001; in part by the Strategic Priority Research Program of Chinese Academy of Science under Grant XDB32050200 and Grant XDC01020000; in part by the Standardization Research Project of Chinese Academy of Sciences under Grant BZ201800001; and in part by the Beijing Academy of Artificial Intelligence (BAAI) through the Beijing Nova Program of Science and Technology under Grant Z191100001119093. *(Zidong Du and Qi Guo contributed equally to this work.) (Corresponding author: Yunji Chen.)*

Zidong Du, **Yongwei Zhao**, and **Tian Zhi** are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China, and also with Cambricon Technologies, Beijing 100037, China.

Qi Guo and **Zhiwei Xu** are with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China.

Yunji Chen is with the State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China, also with the School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100864, China, also with the Institute of Brain-Intelligence Technology, Zhangjiang Laboratory (BIT, ZJLAB), Shanghai Research Center for Brain Science and Brain-Inspired Intelligence (Shanghai Brain/AI), Shanghai 201210, China, and also with the CAS Center for Excellence in Brain Science and Intelligence Technology (CEBSIT), Shanghai 200031, China (e-mail: cyj@ict.ac.cn).

Digital Object Identifier 10.1109/JPROC.2020.2977722

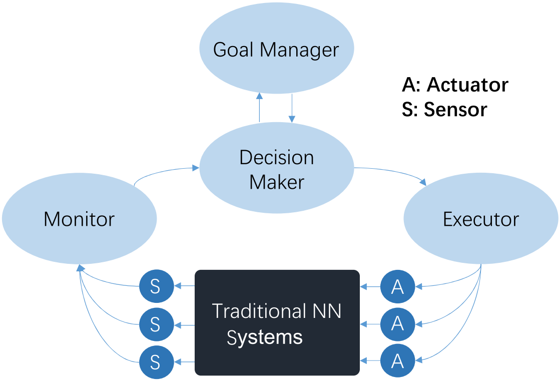

Fig. 1. ***Conceptual diagram of SaNNSs. There are four crucial elements in an SaNNS, that is, the monitor, the decision maker, the GM, and the executor. The monitor collects information sensed from the execution via various sensors. The decision-maker determines the actions according to the collected information and prioritized/correlated constraints from the GM. The executor generates corresponding control commands, which are then enforced by the actuators.***

of applications, including image recognition [1], speech processing [2], machine translation [3], automatic database indexing [4], and scientific computing [5]. Such effective neural networks (NNs) pose great pressure on both the computation and memory due to their deep and large topology. To alleviate such pressure, customized NN processors gain increasing interests [6]–[8]. For example, DianNao, an accelerator proposed in 2014 [9], achieves comparable performance (0.8×) but higher energy efficiency (2.18×) to Nvidia K20M GPU. DaDianNao [7], a multicore accelerator extended from DianNao in 2015, improves the performance and energy efficiency to 20× and 330.56×, respectively, compared to K20M GPU. Many other accelerators have been proposed to further improve the performance and energy efficiency [10]–[24].

Self-awareness, whereby a system can dynamically react based on continuously sensed information from the execution environment, becomes critical for both the efficiency and reliability of the NN system. Recently, several self-awareness techniques have been employed in NN accelerators for further improving the performance and energy efficiency [10], [12], [13], [20], [25]–[28], leading to self-aware NN systems (SaNNSs). Moreover, the reliability issue in these efficient NN systems could be even more critical. For those NNs which trade the accuracy for higher efficiency in performance or energy, their outputs are more vulnerable as incorrect results could happen to any of the input samples unpredictably. Without self-awareness for controlling these unpredictable behaviors, NN systems are not reliable any more, especially in life-critical scenarios.

Fig. 1 shows the ideal conceptual diagram of the SaNNS, which introduces several crucial elements, that is, the monitor, the decision maker, the goal manager (GM), and the executor, to traditional NN processors. Also, execution information is collected by the sensors and control commands are executed by the actuators. Following this paradigm, many modern NN accelerators can be (at least partially) treated as SaNNS. For example, various sparse NN processors, for example, Cambricon-X [20] and EIE [26], employ additional logics for monitoring nonexisting neurons and synapses of processed NNs at run time, determining whether the corresponding computation should be ignored, and enforcing computation bypass in the pipeline. In addition to architectural supports, self-awareness can also be achieved at other layers such as physical layer [16], [29]–[33] and circuit layer [26], [34]–[38].

In this article, we survey state-of-the-art SaNNS and show that self-awareness can be achieved at different layers, that is, the architectural layer, the physical layer, and the circuit layer. At the architectural layer, SaNNS can be characterized from a data-centric perspective as different data (including neurons and weights) properties can be exploited for high performance and energy efficiency, given specific constraints. Such data properties include data value, data precision, dataflow, and data distribution. Taking the data value as an example, the value of neurons and weights can be monitored at runtime and zero values can be bypassed during computation. At the physical layer, self-awareness is achieved by adjusting different physical parameters, such as voltage, frequency, and retention time, under specific target constraints. At the circuit layer, the self-awareness is built upon newly designed logics or devices, which can be utilized for specific computation or memory access. At this layer, it is possible to investigate more aggressive and fine-grained policies for self-awareness.

By reviewing the employed techniques, we observe that the self-awareness of existing SaNNS is still in a preliminary stage. Moreover, existing SaNNS implements self-awareness typically at relatively low levels (i.e., the architectural, physical, and circuit layer) without exploiting high-level application/algorithm semantics. Thus, we propose to exploit the self-awareness at the model layer, which can take the application characteristics into consideration for high efficiency. The proposed system is called MinMaxNN that dynamically switches different NN models, for example, the min model for fast execution, and the max model for high accuracy, based on dynamically monitored application behaviors. In addition to such model switching mechanism, MinMaxNN also features the elastic sparsity mechanism, whereby the sparsity of each layer can be dynamically adjusted for higher efficiency. Experimental results show that compared with traditional SaNNS, MinMaxNN can achieve 5.64× and 19.66% performance improvement and energy reduction, respectively, without notable loss of accuracy and negative effects on developers' productivity.

The rest of this article proceeds as follows. Section II provides the background of NN algorithms and NN processors. Section III surveys existing SaNNSs. Section IV discusses several design issues of an SaNNS. Section V proposes MinMaxNN by adjusting both models and sparsity based on sensed application behaviors. Finally, Section VI concludes this article.

II. BACKGROUND

In this section, we give the prime of SaNNSs, including the concept of self-awareness, NN algorithms, NN accelerators, and NN system software, and NN applications.

A. Self-Awareness

The concept of self-awareness in computing systems originates from Kephart's work on autonomic computing in 2003 [40], and the complete definition of self-awareness is offered by Kounev [41] and Kounev *et al.* [42], which includes self-reflection, self-prediction, and self-adaption. The self-reflection indicates that the system should be aware of the execution environment including software and hardware infrastructure, the operational goals such as quality-of-service (QoS) requirements, and the dynamic changes in environment and goals. The self-prediction indicates that the system should be able to predict the effect of a dynamic environment (e.g., resource allocation) and goals (e.g., QoS requirement). The self-adaption indicates that the system should adapt to the changes to guarantee that the goals (e.g., QoS, performance, and power) can be satisfied. In addition to Kounev's definition, Jantsch *et al.* [43] also discuss that self-awareness is more than self-reflection + self-prediction + self-adaption, as learning from the history and dynamic goal management is also an essential factor for achieving self-awareness.

As an illustrative example, Fig. 2 shows the system architecture of the cyber–physical SoC (CPSoC) that enhances traditional multiprocessor SoCs (MPSoCs) with additional on-chip and cross-layer sensing and actuation mechanism for self-awareness. In this system, the indispensable monitoring–deciding–execution loop is implemented across the hardware, operating system, middleware, and application layers, which allows the interaction with the environment in real-time to dynamically adapt system behavior while satisfying various design goals (e.g., QoS, power, and energy efficiency) [39]. However, recall the complete definition of self-awareness, the CPSoC is still far from an ideal self-aware system based on several observations. The first is that the goal management is not well studied, especially the interaction between the goal management and decision making for coordinating multiple design goals. The second is that only limited history information and learning models are employed for self-prediction. The third is that the decision-making mechanism is relatively simple due to various resource limits (e.g., performance, area, and power constraints). Therefore, there is still a lot of room for enhancing the self-awareness capabilities of traditional computing systems.

B. Algorithms

NNs consist of many layers where each layer is composed of massive neurons. A neuron in the layer is connected to other neurons in adjacent layers of the current layer through weighted edges, which are called synapses.

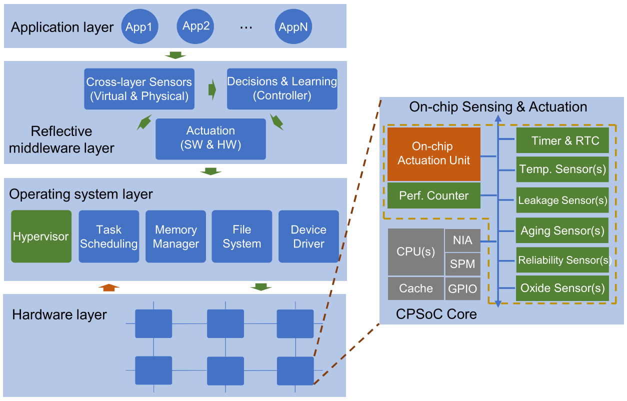

Fig. 2. ***System architecture of the self-aware CPSoC that enhances traditional MPSoCs with additional on-chip and cross-layer sensing and actuation mechanism for self-awareness [39]. The self-awareness in CPSoC is implemented from different layers, including hardware, operating system, middleware, and application layer. At the hardware layer, there exist multiple sensors and actuators in both CPSoC cores and NoC routers as the CPSoC computational fabric typically consists of tiled homogeneous or heterogeneous CPSoC cores by using NoC interconnect. At the application, operating system and middleware layer, there exits multiple virtual sensors and actuators, including system utilization and task scheduling, which can combine with actual sensors and actuators at other layers for multi or cross-layer interactions and interventions.***

The neuron is fundamental to NNs, and there are four commonly used neuron models, including the basic neuron model, pooling model, recurrent neuron model, and LSTM neuron model. The basic McCulloch and Pitts model was proposed in the pioneering work of McCulloch and Pitts in 1943 [44]. It integrates information of inputs by multiplying each input with the weights of connected synapse correspondingly, then passing the accumulated multiplication results to the following neurons through an activation function. Many types of layers use this neuron model, such as convolutional (CONV) layer and fully connected (FC) layer. The pooling neuron downsamples input to gather features filtering out noises, which is usually used in CONV NN for image-related tasks. The pooling neuron usually performs a selection operation to allow the maximum input pass through or averaging all its inputs, known as maximum pooling or average pooling, respectively. A recurrent neuron model is much like the basic McCulloh–Pitts model except that previous neuron output is used as a current input again. The LSTM model is more complex than a recurrent neuron, which contains four parts, including a memory cell, an input gate, an output gate, and a forget gate. Some other variants of LSTM omit one or more gates deliberately or introduce new gates, such as gated recurrent units (GRUs) [45].

Together with different neuron models, various connections among layers form different types of layers, such as

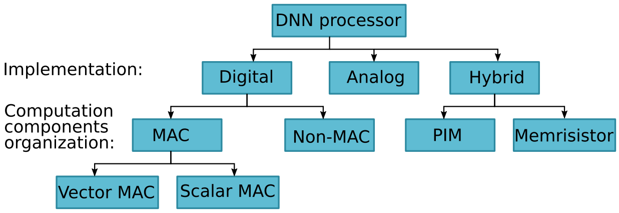

Fig. 3. ***Classification of DNN processors.***

CONV layer, FC layer, batch normalization (BN) layer, and LSTM layer. Moreover, from the perspective of connections, neurons can be connected to one neuron (e.g., BN), multiple neurons (e.g., CONV), or all neurons in the previous layer(s) (e.g., FC) or in its own layer (e.g., LSTM).

C. Accelerators

Together with the resurgence of NNs, DNN processors evolve rapidly since the year of 2014, and Fig. 3 shows the classification of DNN processors from the hardware perspective. Roughly, DNN processors can be classified into three categories: digital circuits, analog circuits, and hybrid circuits. The digital circuits can be further classified into MAC-based and non-MAC-based organization, in terms of the organization of computation components. MAC-based architectures are the most common organization in DNN processors because most of the operations in DNNs can be aggregated into vector inner production and element-wise operations [6], [46]. Non-MAC-based organization mainly includes newly introduced devices, such as memristor [47], [48]. The pure analog DNN processors are rarely seen as analog signals that need to be converted into digital signals for current host systems [49]. This leads to the digital-analogy hybrid circuits, which mainly include process-in-memory (PIM)-related architectures [50]–[52] and memristor-related architectures [5], [47], [48].

D. System Software

The system software for NN processors involves operating system, runtime system, library API, domain-specific language (DSL)/compiler, and programming frameworks, which are in charge of bridging the gap between the underlying hardware and user-level applications, as shown in Fig. 4. The programming frameworks can be classified into two categories: layer-based such as Caffe [53] and graph-based such as TensorFlow [54], MXNet [55], and PyTorch [56]. The library API encapsulates commonly used primitives such as CONV, POOL, and FC for providing high-performance portable APIs [57]–[59], while the DSL/compiler is mainly used for adapting evolving deep learning algorithms [60]. The runtime system provides functionalities for user-level device management, task scheduling, and memory management [54], [61]. The operating system mainly targets for the memory management and device drivers of NN hardware, and the main functions could be implemented with lightweight microservices on NN processors. In fact, cross-layer self-awareness can be achieved from different layers, including programming frameworks, runtime systems, and operating systems. For example, the widely used TensorFlow allows the same program to be deployed on different devices such as CPUs, GPUs, and TPUs, and it is achieved by employing a placement algorithm in the runtime system for automatically distributing different operations to different devices subject to various constraints [54].

E. Applications

DNNs have been demonstrated to be successful in many fields, especially in image and video processing. For example, the Deep Residual Network [62] containing 152 layers (i.e., ResNet-152) achieves 3.57% error when performing classification tasks on ImageNet [1] data set, which is better than human-level accuracy (i.e., 5.X%). In speech recognition, Microsoft proposed an LSTM-based model that outperforms human transcribers on various data sets [63]. In addition to traditional video/image/audio processing, deep learning has shown its potential in many other fields such as machine translation and recommendation. Google's neural machine translation (GNMT) system approaches the performance achieved by average human translators on some test cases [64], and deep learning models can predict the effects on cellular processes caused by genetic variation [65]. Unlike video/image/audio processing that are relatively error-tolerant and pay more attention on the ensembled statistical results, for example, top-1/top-5 accuracy of 50 000 images, many applications require highly confident results for processing each single sample, especially in real-world critical fields such as automotive and recommendation. Such robustness advances higher requirements on the theory, algorithm, and application practice of NNs. Therefore, the distinct feature of NN systems, that is, the accuracy and robustness should be considered as one of the top-most design targets, pose challenges on designing SaNNSs, including monitoring, decision making, goal management, and execution.

III. SELF-AWARENESS IN EXISTING NN SYSTEMS

Various self-awareness techniques have been employed in NN systems with/without intention to provide the ability of

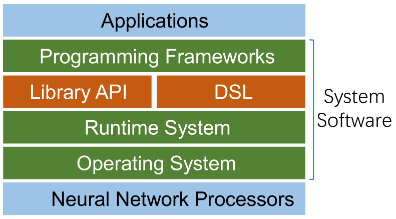

Fig. 4. ***System software stack of NN processors, where the self-awareness can be implemented in different components, including programming frameworks, runtime system, and operating system.***

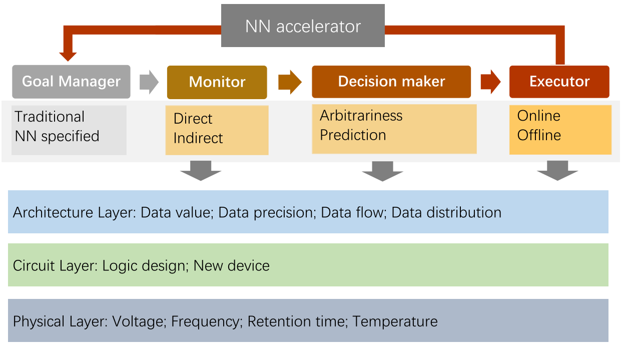
NN accelerator
Goal Manager
Monitor
Decision maker
Executor
Traditional
NN specified
Direct
Indirect
Arbitrariness
Prediction
Online
Offline
Architecture Layer: Data value; Data precision; Data flow; Data distribution
Circuit Layer: Logic design; New device
Physical Layer: Voltage; Frequency; Retention time; Temperature

Fig. 5. ***Entire paradigmatic architecture of SaNNSs.***

self-reflection, self-prediction, and self-adaption. However, we found that self-awareness is not mature in existing NN systems as one or more components (which are defined as the core components in self-aware computing system, as shown in Fig. 1) are in rudimentary forms or even absent. Thus, instead of strictly dividing the entire system into stated components with clear boundaries, we investigate the self-awareness in NN systems from a vertical point of view. More specifically, the self-awareness can be achieved at different layers, that is, the architectural layer, the circuit layer, and the physical layer.

A. Overall

In Fig. 5, we show the entire paradigmatic architecture of SaNNSs, which consists of four major parts, that is, GM, monitor, decision-maker, and executor.

1) Goal Manager: As there are multiple goals for designing a computing system, the key responsibility of the GM is to formulate and then dynamically prioritize various quantitative goals for design and optimization. In addition to traditional design goals such as performance and energy efficiency, there are several distinct design goals of NN systems.

As stated, the design goals include traditional goals that self-awareness systems pursue, and NN-specific goals. Traditional goals include performance, energy efficiency, QoS, reliability, and security. In particular, performance and energy efficiency are the key reasons for using customized NN processors. Customized NN processors achieve much higher performance and energy efficiency than traditional CPUs/GPUs, and thus act as alternatives to GPUs/GPUs in both academia and industry. For example, Google announced TPU and keeps upgrading to provide cloud computing services with 100 Petaflops peak performance in each pod [66].

Particularly, NN systems are bounded by specific goals, that is, accuracy, to pursue other traditional design goals. Accuracy is a statistic metric, which reports the correctness on a number of test samples, usually the test data set. A recent work focus on accuracy improvement without considering the hardware design [67]. In particular, NNs involve two different phases, inference and training. During the inference phase, for each input sample, NN systems usually output a vector (or a scalar if only one output neuron) readout, which is further regarded as classification results or possibility representation. For inference, accuracy is closely related to the performance of the NN system. During the training phase, NN systems are provided with reference results for the training samples to adjust the parameters in networks gradually—errors between network vector readouts and reference results are propagated back through network layers.

In the GM, all the above design goals should be formulated, correlated, and prioritized based on the sensed information. For example, Gobieski *et al.* [68] proposed an intermittence-aware NN system for energy-harvesting devices in IoT applications. In this system, the performance, energy, and accuracy are formulated and correlated with an analytic model so as to accelerate intermittent DNN inference given severe resource constraints. Moreover, the accuracy and performance goal could be dynamically adjusted according to the available energy in the hardware budget.

2) Monitor: The monitor collects information sensed from the execution environment via various sensors. These sensors can be either physical for directly measuring or virtual for indirectly predicting/estimating metrics of the environment. The information sensed by the monitor can be roughly classified into three groups, that is, internal states (such as temperature and power), internal targets (such as QoS and energy consumption), and external change (such as fluctuated input requests and power supplying), and all the information can be directly measured or indirectly predicted in the monitor.

Modern devices already provide hardware support to directly measure the internal states. One example is the memory controller, which is already capable of measuring the bus utilization directly [69] from its hardware modules directly. Another example of indirectly predicting the interesting metrics of the system is predicted based on the architecture (uniform serial processing element, USPE), where an ineffectual neuron is predicted for skipping the computations related to the ineffectual neuron [14].

3) Decision Maker: The decision-maker determines the actions based on the collected information from the monitor and the prioritized constraints from the GM. The decisions are formulated in rules, and the rules can be triggered once a condition, an action, or a predicted reward value is reached. More specifically, take the sparse NN processor Cambricon-X as an example, once the synapses are monitored and detected as zero values, the corresponding neurons will not be selected for computation. Actually, the rules are implemented in hardware called indexing module (IM). Another example is a unique weight CNN (UCNN) Accelerator [70] that detects repetitive weights on-the-fly for reducing multipliers in inner production computation and weights memory access. These rules

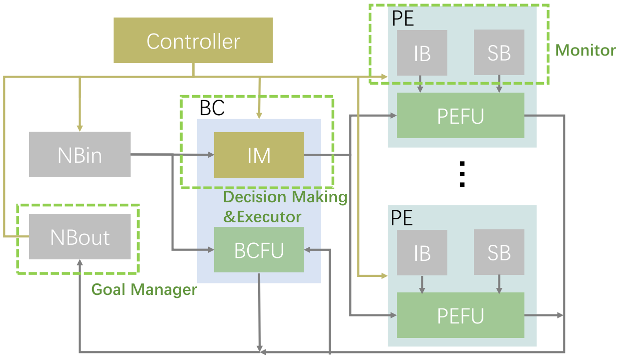

Fig. 6. ***Microarchitecture of Cambricon-X. Its IM module plays the role of decision making and executor; for static neuron sparsity, it filters out neurons based on the indexes from each PE, and for dynamic neuron sparsity, it decides whether to treat a neuron as a nonexisting one. Its SB and IB modules in each PE together play the role of monitoring; it fetches compacted stored sparse synapses and corresponding indexes.***

are also implemented in hardware for efficiency purposes. In the multiple classifier system (MCS) [67], the output classification result of an input sample will be the result of the classifier with the highest confidence.

4) Executor: The executor is mainly in charge of generating the software or hardware control commands for enforcing decisions. There are two different types of enforcement mechanisms, that is, online and offline. Regarding the online execution, still, we take the Cambricon-X as an illustrative example. In Cambricon-X, the executor is the buffer controller (BC) for determining which neurons should be transferred for computation, and for orchestrating the computation in processing elements (PEs). With the help of BC, less computation-intensive operations will be performed. Regarding the offline execution, we can take the network quantization as an example, where the models can be quantified in advance and quantized operations will be performed during execution.

Fig. 6 shows the detailed architecture of Cambricon-X and the related components included in a self-awareness system. In Cambricon-X, the NBout is closely related to the GM, where the accuracy is directly computed on the data from NBout. The IM plays the role of the Decision Maker and the Executor, which determines whether a neuron should be filtered out. In each PE, the index buffer (IB) and synapse buffer (SB) together play the role of a monitor, where they fetch compacted stored sparse synapses and corresponding indexes. Apparently, it is very hard to describe the clear boundaries of different components in existing SaNNS, as this area is very preliminary. In Sections III-B–III-D, we introduce the related work from a vertical perspective.

B. Architectural Layer

We investigate the self-awareness at the architectural layer from a data-centric rather than operating-centric perspective, where we take the data rather the operation as our primary considering factor during the investigation. The intuition is that for NN applications, key design targets (such as performance, energy, and accuracy) are mainly determined by different data properties, including data value, data precision, data flow, and data distribution. A distinct example is sparse NN, for zero-value neurons and weights, performance and energy efficiency can be significantly improved at the cost of negligible accuracy loss. The detailed taxonomy is shown in Fig. 7.

1) Data Value: The concrete value of data in NN systems—input and intermediate data—can be exploited for achieving various design goals, including high performance and energy efficiency.

a) Input data: Applications with sequential inputs, such as computer vision applications and speech-related applications, usually contain temporary redundancy among the continuous inputs. The temporary redundancy implies that two continuously inputs, for example, two video frames or two audio frames, do not change much [see Fig. 8(a)]. Thus, it is feasible for NN systems to leverage such temporary redundancy by skipping processing the same parts in the latter input. As NN algorithms are able to tolerate a certain amount of errors in both data and computation, temporary redundancy can be further exploited by skipping not only the same but also the similar parts in latter input. To obtain the final results of the latter input, different approaches are proposed, including direct input similarity, differential computing, and activation motion compensation (see Fig. 8).

In [12], the proposed DNN accelerator leverages the input similarity directly by skipping the computations related to inputs that have negligible change [see Fig. 8(b)]. Together with the quantization technique to increase the similarity among frames, results show 60% of the inputs are not changed, leading to an energy savings of 63% on average.

Diffy [25] uses the differences between the current frame and the keyframe for computing only the different parts (delta) [see Fig. 8(c)]. Diffy lowers the on-chip storage requests by a factor of 32% and thus boosts the performance.

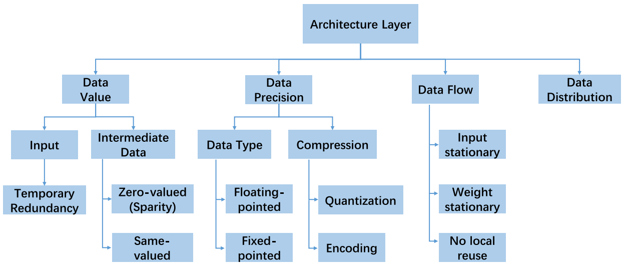

Fig. 7. ***Proposed taxonomy at the architectural layer from a data-centric perspective.***

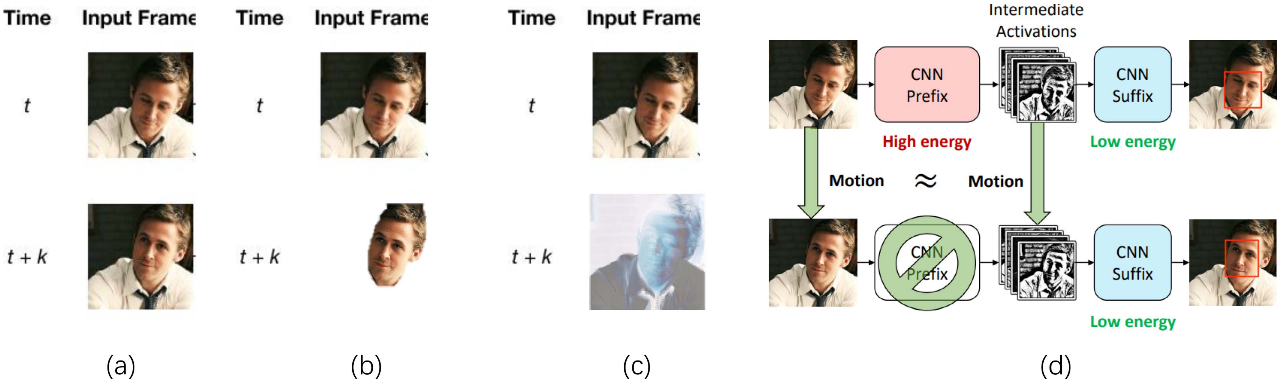

Fig. 8. ***Temporal redundancy. (a) Key frame and similar frame. (b) Direct input similarity. (c) Differential among two frames. (d) Activation motion compensation.***

In [13], the proposed accelerator processes the inputs using a motion estimation-based technique, that is, activation motion compensation. It detects changes among the visual inputs and incrementally updates a previously computed feature maps [see Fig. 8(d)]. The accelerator reduces the average energy per frame by 54%, 62%, and 87% for three CNNs with less than 1% loss in vision accuracy.

b) Intermediate data: The intermediate data, including synaptic weights and neuron outputs, have been studied intensively on two special cases—zero-valued data (sparsity) and same-valued data.

Commonly, synaptic weights are investigated into the special case of zero-valued synapses, that is, sparsity revisited in the past few years. Song *et al.* [71] proposed to prune the "unimportant" synapses—synapses with small weights—from the network topology following a three-step procedure. The entire three-step procedure is shown in Fig. 9, where the sparse network is trained iteratively with the same inputs. The first step is to train the network to learn the importance of synapses—the synapses with larger weights are considered important synapses. Based on the learned synapses, "unimportant" synapses will be pruned from the network in the second step. The final step retrains the network to learn the final weights with sparse connections. Such a procedure could be applied multiple times, in order to obtain as sparse as possible networks that have acceptable accuracy. Note that a neuron is useless if it has no input connections or output connections; thus, it is removed implicitly from the network during the above three-step pruning of synapses. Also, neurons can be pruned directly from the network topology. Although the pruning method mentioned above changes the topology of NNs, sparsity due to neuron activations (i.e., ReLU) varies from different inputs without changing the network topology. Overall, sparsity in neuron networks can be classified into two major categories: static and dynamic (see Fig. 10). The static sparsity indicates the synapse sparsity and neuron sparsity, and the dynamic sparsity contains only neuron sparsity as it is caused by zero-valued neurons which vary with different inputs (thus dynamically changed sparsity).

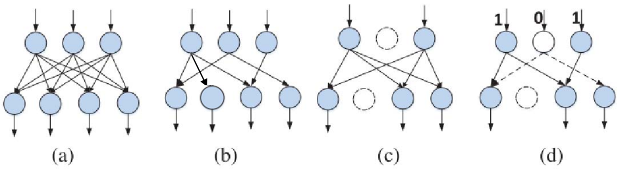

Fig. 10. ***Sparsity in an NN [21]. (a) Original dense network. (b) Static sparsity: synapses. (c) Static sparsity: neurons. (d) Dynamic sparsity.***

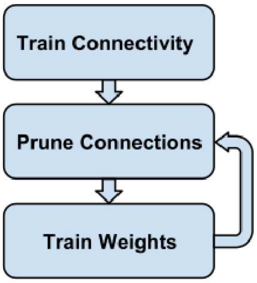

Fig. 9. ***Train a sparse NN [71].***

Many recent works have devoted to support the sparsity in network processing for reduced computations and memory accesses. EIE [26] performs sparse matrix–vector multiplication, especially for FC layers using a compressed sparse column (CSC) format for static sparsity. It proves that sparse data could save more than half (65.16%) of the energy by avoiding weight references in typical deep learning applications. SCNN [27] adopts a 2-D organization of computing elements with sparse dataflow and indexes stored and computed for data indexing. SCNN is also capable of accommodating both static and dynamic sparsity. Cnvlutin [28] aims at exploiting dynamic neuron sparsity, but it cannot exploit synapse sparsity. Cnvlutin also exploits the possibility of skipping near-zero neurons by setting the neurons to zeros directly. Cambricon-X [20] adopts a totally different sparse data representation, that is, binary masking format, together with a specially designed IM to filter neurons with compactly stored synapses. As the compressed synaptic weights are reduced to a similar size, binary masking format, using 1-bit instead of two data in CSC or CSR, is suitable for representing the sparse synapses and leads to higher efficient accelerators. Compressing DMA engine [72] compresses the data transferred between host CPUs and accelerating devices to a compact form to leverage the output neuron sparsity (activations), especially in training.

Additionally, researchers also explore the possibility of directly setting neurons to zeros to improve the sparsity in networks. In [14], the proposed DNN model predicts the ineffective neurons to completely avoid the related computations by skipping over those neurons. SnaPEA [17] reorders the weights to predict the neuron outputs earlier before finishing the computing, and thus speculatively cut the computation and set the neuron outputs with the predicted value.

Although the above works focus on using zero-valued data in networks for computation, there also exists the potential of reusing the computed data (same-valued data). UCNN [70] exploits the repeated weights to reduce multipliers in inner production computation and weights memory accesses, and the reuse of CNN subcomputations (e.g., dot products, sum-of-product-of-sums, and forward partial results) to reduce computations. Shortcut mining [73] proposes a new reuse opportunity, the largely unexplored shortcuts and feature maps in residual networks, to reduce the off-chip traffic.

c) Self-awareness: For inputs, as NN algorithms are able to tolerate a certain amount of errors in both data and computation, temporary redundancy can be further exploited with skipping not only the same but also the similar parts in latter input. For example, work [12] skips the computations related to inputs that have negligible change with quantization technique to increase the similarity among frames. Additionally, for intermediate data, sparsity, especially the static synapses sparsity, is achieved by removing the "unimportant" synapses in the network. But the "importance" is predefined and thus can be adjusted for trading performance and efficiency. The "same-valued" technique can also be extended to "similar-valued" the technique to reuse the computed results of similar data instead of the same data, such as weight repetition architectures in UCNN [70] can exploit more opportunities with similar weights (e.g., quantization). In summary, the data value is able to be monitored and tuned for achieving SaNNSs.

2) Data Precision: Data precision can be exploited in two ways for different design targets, especially high performance and energy efficiency: changing the data type from floating-point to fixed-point or from more bit-width to less bit-width, and compressing the data using techniques such as quantization. The former is closely related to single data, and the latter works on a group of data.

a) Data type: There exist two common types of data representation NN systems, that is, fixed-point data and floating-point data. The floating-point data is critical for training NNs due to its ability to represent very small numbers. However, as float32 is costly in terms of area and power in hardware implementation and float16 has a limited range, various floating formats have been proposed, including the brain float16 (bfloat16) which uses 8 bits for exponents and 7 bits for fractions [74]. Compared with the traditional float16, bfloat16 provides a larger range for data representation, and thus it is able to avoid the overflow in large networks.

For inference-only NN systems, fixed-point data with fewer bits have proven to sufficient. The 16-bit fixed-point data have been widely employed in many NN accelerators, such as DianNao Family [75]. As 8-bit is proven to be capable of processing inference for most of the networks, not only the academic prototypes but many commercial products implement 8-bit fixed-point computing components. For example, TPU v1 [76], specialized systolic array accelerator, uses 8-bit fixed-point in each of the PE, delivering a 92TOPs peak performance. NVIDIA integrated tensor cores equipped with 8-bit fixed-point units in its latest products, such as Tesla V100 and achieving 125TOPs peak performance [77]. Cambricon released two acceleration cards for servers, integrating 8-bit fixed-point (INT8) for a maximum 128TOPs peak performance [78].

b) Compression: Compression typically works on a group of data, which are expressed using a smaller size of data through compressing techniques. The most common compression technique is the quantization, which maps the original data to a small number of limited quantization levels. Such quantization levels reflect the data precision, where data usually use indexes with fewer bits to index the quantization levels. The quantization levels represent the original N data with a K-entry dictionary and $\log(K)$-bit for each of the data (thus $N \times M$ bits versus $K \times M + \log(K) \times N$ bits). Many works leverage a global quantization, that is, performing quantization over an entire weight matrix [24], [79]. Cambricon-S [21] uses local quantization, that is, performing quantization over the local area of weights, to deliver a higher compression ratio. OLAccel [11] performs outlier-aware quantization, where the low precision part (e.g., $\sim$97%) uses a lower bit (e.g., 4 bits) for quantization and the high precision part (e.g., $\sim$3%) performs 16-bit quantization. The encoding techniques, such as entropy encoding (e.g., Huffman encoding) [26], can be further applied to reduce the bits for indexing each entry. It encodes the entries with variable-length indexes where more common entries are represented with fewer bits.

c) Self-awareness: Dynamically changing the data precision is very effective for achieving self-awareness. Generally, in this system, the monitor can detect the

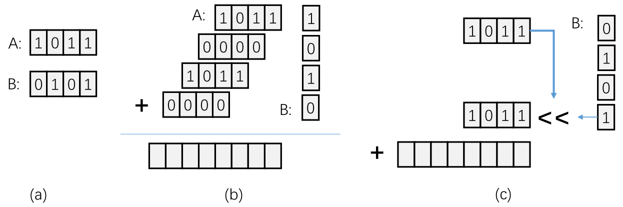

Fig. 11. ***Bit-serial computing. (a) Two data (4-bit A and B) for multiplication. (b) Parallel computing unit—array multiplier, where A is expanded into a 4 × 4 array to compute in parallel. (c) Serial computing unit, where A is accumulated in four cycles with the help of B to get the final result.***

change of input data such as different bit-width, and the decision-maker can determine whether unnecessary computation can be bypassed. One of the most important techniques to dynamically change data precision is using serial computing units, which can finish computing in the time with regard to data precision. An illustrative example is shown in Fig. 11, where a 4-bit data multiplication can be performed in four cycles, and in each cycle A is accumulated based on one bit in B. There are many works built upon bit-serial computing. Stripes [24] explores the reduced bit precision using bit-serial computing to dynamically adapt to different bit-width in different layers. Bit-pragmatic [79] further explores the dynamic sparsity with bit-serial computing. Bit-prudent [50] and Neural Cache [51] study the bit-serial computation with in-SRAM architectures. Bit-fusion [80] introduces dynamic bit-level fusion/decomposition in NNs and designs a new accelerator that consists of an array of bit-level PEs that dynamically fuse to match the bit-width network layers.

3) Dataflow: As there exist many memory-intensive operations in NN algorithms, the accompanying costly data movement should be carefully addressed by designing appropriate dataflow, that is, the mechanism of data handling, in NN architectures. The dataflow is strictly determined by the organization of computational components, and the most common computational components are MACs, including vector MACs and scalar MACs. A vector MAC performs an inner product on two input vectors and is able to accumulate the results locally. The vector MACs can be further organized with different topologies. For example, DianNao [6] employs 16 int-16 vector MACs, where the neurons are broadcast to these MACs to perform inner products with independent weights. A scalar MAC performs multiplication on two input scalars and is able to accumulate the results locally as well. The scalar MACs can be organized into 1-D, 2-D, or even 3-D topologies (such as a systolic array) to cope with different computational flows. For example, ShiDianNao [8], TPU [76], and Eyeriss [81] employ 2-D mesh topology of $N \times N$ scalar MACs, allowing data sharing among a region of MACs. The key difference between the vector and scalar MAC-based architecture is that they are in favor of different atomic operations. More specifically, the vector MAC-based architecture usually addresses all the vector operations such as vector inner product and vector element-wise operations (e.g., vector addition). The scalar MAC-based architecture copes with spatial data flows dedicated for CONV operations to reuse the input neurons or synaptic weights among neighboring MACs. Both the vector and scalar MAC-based architectures are capable of mapping with different dataflows. From the perspective of data stationary, the dataflow can be classified into output stationary, weight stationary, and no local reuse (NLR) dataflow, as presented in [81].

a) Self-awareness: In addition to traditional dataflows as introduced, dynamic dataflow mechanisms can be exploited for achieving self-awareness. Dynamic dataflows can be deployed for different networks, even for different layers. This is achieved by monitoring the parameters/topologies of processed networks/layers, and then dynamically selecting the best dataflow for mapping in decision making. Flexflow [82] demonstrates a design that provides feature-map, neuron, and synapse-level parallelism within a layer, by different mapping strategies across PE rows and columns. MAERI [19] proposes to use a reconfigurable network-on-chip (NoC) to support different dataflows, especially crossing layers. In more detail, MAERI leverages a reconfigurable augmented reduction tree (ART) topology to reconstruct virtual neurons with different capabilities, supporting different dataflows. Morph [83] provides a design space exploration mechanism and corresponding flexible architecture for accelerating 3-D CONV NNs (3-D CNNs).

4) Data Distribution: In contrast to the stated work mainly focusing on data representation, there also exist several works utilizing data distribution, that is, the clustering characteristics of neurons and weights, for better efficiency. The data distribution such as weight value can be controlled and changed at training time, and the carefully pruned models can be exploited by designing proper hardware components. For example, both Han *et al.* [26] and Hegde *et al.* [70] leverage weight-sharing techniques to train the weights with limited values. Cambricon-S [21] proposes a coarse-grained pruning technique to obtain a more regular connection among the input–output neurons. Thus, Cambricon-S is able to leverage such regular sparsity with a shared IM. Scalpel [22] also studies a new pruning method, SIMD-aware weight pruning and node pruning, to shape the sparse network more hardware-friendly. Thus, such sparse networks could also benefit traditional CPUs and GPUs for high efficiency. Alternating direction method of multipliers (ADMM)-NN [84] leverages the ADMM to prune weights in NNs, resulting in a much sparse network, 1910× and 231× reduction on LeNet-5 and AlexNet. PERMDNN [85] exploits permuted diagonal matrices to generate and execute hardware-friendly structured sparse DNN models. It eliminates the drawbacks of indexing overhead, nonheuristic compression effects, and time-consuming retraining.

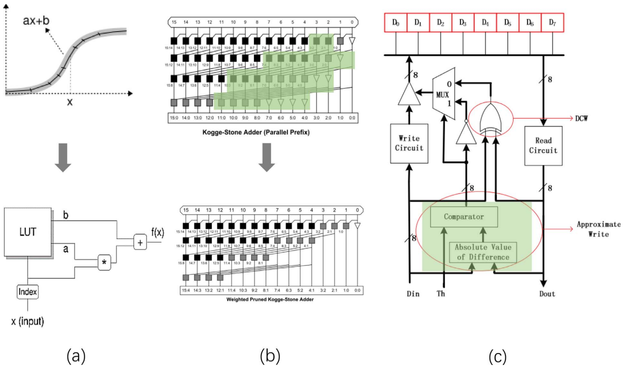

Fig. 12. ***Circuit layer approximation. (a) Piecewise linear approximation of sigmoidal function [86]: a multiplier, an adder, and a small LUT. (b) Logic minimized adder [87] (green area: gates are pruned). (c) Approximate write in PCM [88] (green area: comparing the current data and stored data to decide whether writing to PCM).***

C. Circuit Layer

We investigate the self-awareness at the circuit layer with respect to circuit design, and it is typically achieved by leveraging the logical level changes or new devices for new, usually more efficient, hardware circuit design.

For logical level changes, the basic principle is to use less logic gates to approximate the functionality of original hardware modules, the computation of operators, or the storage of data. For example, complex activation functions, such as the sigmoidal function ($f(x) = 1/(1+e^{-x})$) and hyperbolic tangent function ($f(x) = (e^{x} - e^{-x})/(e^{x}+e^{-x})$), in NNs are costly if implemented in hardware. As approximate substitutions, piecewise linear approximation [see Fig. 12(a)], using a multiplier and an adder with a small lookup table (LUT) storing coefficients ($x \to f(x) : a_i * x + b_i$), is used widely [6]–[8], [37], [86]. Du *et al.* [37] leveraged approximate operators [87] to design an inexact NN accelerator with ∼50% energy savings. The much simpler approximate operators are achieved by using inexact logical minimization (ILM) to reduce logic gates, such as removing the lower bits logic gates in a kogge-stone adder to have an inexact kogge-stone adder as shown in Fig. 12(b). Fang *et al.* [88] proposed approximate write—writing only largely changed data to PCM [see Fig. 12(c)], to reduce the write energy costs and also increase the lifetime of PCM.

Additionally, new devices or modified devices are introduced in a circuit to replace the original devices, such as memory. For example, 8-T or 10-T cell-based SRAM are introduced to provide more robust and reliable performance by isolating the read and write paths [34], [35]. Probabilistic CMOS [36] technique reports a new switch (probabilistic behavior when scaling the voltage to an extremely low level) as new devices for potential circuit design. Moreover, since NNs can naturally tolerate a certain amount of errors, many of these inexact or approximate techniques are applied to leverage such nature. Also, the retraining technique is widely applied to recover the accuracy of NN systems for tolerating more errors. For example, Du *et al.* [37] recovered the accuracy of the inexact accelerator to almost original value on different data sets by retraining. Han *et al.* [26] retrained the NNs with iteratively pruned to sparse networks and achieved negligible accuracy loss. Deng *et al.* [38] retrained hardware NNs to mitigate timing errors caused by parameter variation.

Though new logic designs or devices are able to provide the foundation for dynamically reacting based on sensed information from the execution environment, it is admitted that the self-awareness at the circuit layer is in a very preliminary form. For example, the nonlinear operations such as sigmoid are directly monitored and thus substituted with linear approximation at runtime in [6]. The self-awareness can be explored in-depth by utilizing the most appropriate logics or devices at fine granularity with minimal switching overheads. A potential example is to use different approximate logics for computing different parts of a typical NN operation such as CONV, where important weights can be computed with more accurate logics.

D. Physical Layer

We investigate the self-awareness at the physical layer in terms of physical parameters, including voltage, frequency, temperature, and retention time. Different from the circuit layer that focuses on a logic level such as how a multiplier is implemented, the physical layer focuses on the physical level techniques such as how the digital logic (OR/AND gates) is achieved with analog elements (transistors). These techniques have been intensively studied in both academia and industry to directly lower the energy costs with techniques such as power gating, clock gating, and dynamic voltage and frequency scaling (DVFS). Apparently, such systems typically monitor the execution behaviors (e.g., faults, error rate, power, and performance), and thus the adjust physical parameters such as voltage and frequency in the decision-making under specific target constraints.

Truffle [29] uses a dual-voltage design, where a high voltage is used for precise operations and a low voltage is for approximate operations. This is achieved by using an ISA extension to decide when to use which voltage for computing. Relax [30] is an architectural framework for software recovery of hardware faults from techniques such as voltage scaling and over-clocking. Razor [31] tunes the supplying voltage by monitoring the error rate during circuit operation, thereby eliminating the need for voltage margins and exploiting the data dependence of circuit delay. AxNN [32] proposes a systematic methodology to generate approximate NN designs that can be executed by a quality-configurable neuromorphic processing engine. RANA [16] proposes a retention-aware neural acceleration framework for CNN accelerators to save total system energy consumption with refresh-optimized eDRAM.

Table 1 Summation of Self-Aware Techniques

Layer	Class	Description	Examples
Architecture Layer	Data value	Leverage *similar-valued* data	Inference ([12], [13], [25], [70], [73])
		Skip the *zero-valued* data	Inference ([20], [26]–[28], [71], [72], [89]), Training ([72])
		Set the data to zero	Inference ([14], [17])
	Data precision	Replace operators with shorter bit-width ones	Inference ([75]–[78], [90]), Training ([77], [90])
		Compress data in storage and transmission	Inference ([11], [21], [24], [26], [79], [91])
		Use serial computing units	Inference ([50], [51], [79], [80])
	Data flow	Dynamic dataflow	Inference ([19], [82], [83], [92])
	Data distribution	Pre-change the data distribution	Inference ([21], [22], [26], [70], [84], [85], [93])
Circuit Layer	Logic level	Replace the accurate logic with approximate logic	Inference ([6]–[8], [37], [86], [87])
	Device level	Use new devices or modified devices	Inference([34]–[36], [94]–[96])
Physical Layer	Physical parameters	Tune the physical parameters directly or indirectly	Inference([16], [29]–[33])

Wang *et al.* [33] proposed an iteratively retraining-based method with timing analysis to determine the highest acceptable frequency in NN accelerators.

Similar to techniques at the circuit layer, errors or faults introduced in the circuit layer need to be tolerated by reexecuting on accurate devices or retraining NNs on specified devices. AxNN [32] retrains the networks to find the suitable approximating configuration, replacing the found neurons with cost-effective, approximate versions. RANA [16] retrains the networks to adjust the weights based on the predefined failure rate.

E. Summary

In Table 1, we summarize the mentioned techniques with their representative work in inference and training. At the architectural layer, the three basic techniques related to data value are leveraging the similar-valued data or the results of similar-valued data, skipping the zero-valued data, and setting data to zero directly. Three basic techniques related to data precision are using short bit-width operators, compressing data in storage and transmission, and using serial computing units. The major technique related to data flow is dynamically changing the data flow. The major technique related to data distribution is prechanging the data distribution. As existing self-awareness techniques mainly focus on the inference, to our best efforts, we only found a few works on SaNNS for training, where skipping the zero-valued data [72] and using short bit-width operators [77], [90] are used.

In Table 2, several representative NN systems are categorized based on the definition of an ideal SaNNS consisting of four crucial elements, that is, the GM, the monitor, the decision maker, and the executor, as described in Fig. 1. In fact, the self-awareness of existing SaNNS is in a very preliminary form for four reasons. The first is that some key components may not exist in the current NN systems. For example, DianNao only has the executor for enforcing 16-bit fixed-point computation. The second is that even some of the NN system have all key components, and such components are relatively simple and intuitive. For example, in Cambricon-X, the GM is very simple as the accuracy is directly computed on the data from the NBout buffer. The third is that the boundaries of stated key components may be unclear in the current NN systems. For example, in Cambricon-X, the IM of Cambricon-X works as both the monitor and decision-maker, since it can be aware of the data sparsity and then filter the zero values at runtime. The fourth is that the disparity among different computing components in NN systems is not taken into account. Such disparity could lead to the interaction wall, that is, the bottleneck in interaction caused by data exchange and control signaling, which hinders the efficiency of the NN systems.

Intuitively, the extent of self-awareness of various NN systems can be qualitatively measured according to the key components they possessed. Generally, the system with more and dedicated key components may have better self-awareness. For example, Cambricon-X (which has all four key components) is much more self-aware than DianNao (which only has one key component, i.e., the executor), which complies with the intuition.

Moreover, the extent of self-awareness of various NN systems should be quantitatively measured by the

Table 2 Summary of Representative SaNNSs With Respect to Four Key Components

Name	Layer	Goal Manager	Monitor	Decision Maker	Executor
Cambricon-X [20]	architectural (data value)	Y	Y	Y	Y
EIE [26]	architectural (data value)	Y	Y	Y	Y
DianNao [6]	architectural (data precision)	-	-	-	Y
Stripes [97]	architectural (data precision)	Y	Y	Y	Y
FlexFlow [82]	architectural (dataflow)	Y	Y	Y	Y
ADMM-NN [84]	architectural (data distribution)	Y	Y	Y	-
Approximate operator [37]	circuit	-	-	-	Y
Hardware NN Retrain [38]	circuit	-	Y	-	Y
RANA [16]	physical	Y	Y	Y	Y
AxNN [32]	physical	Y	Y	Y	Y

execution results (e.g., performance, energy, and QoS), since the main purpose of self-awareness is to make the system adaptable to the dynamically changing execution environment. For example, if the goal is to achieve high performance and energy efficiency, for different NN systems, given various execution scenarios, the one with the best performance and energy efficiency should be regarded as the one with the best self-awareness.

IV. KEY ISSUES

In this section, we discuss several key issues for achieving self-awareness, mainly focusing on the crucial design targets in NN systems such as performance and accuracy.

A. System Modeling

The first key issue in designing SaNNSs is system modeling, which characterizes the relationship among key components of an SaNNS including the GM, the monitor, and the decision maker. Generally, the model takes the dynamically changed metrics such as power and temperature from the monitor as inputs, and it can provide estimated values of various design goals such as performance, energy, and accuracy for the decision making. The model also takes inputs from the GM, which prioritizes different optimization goals, for determining the values of correlated design goals. As the system modeling plays an important role, it is worth to be well studied in SaNNS, so as to accurately predict the values of optimization goals under stringent time constraints.

B. Performance Measurement

As one of the most important metrics, performance needs to be measured frequently at runtime from the monitor for decision making and feedback collection. There are different challenges for measuring the performance of inference and training in an SaNNS.

For inference, performance measurement can be classified into two categories: the same task with/without batch, and QoS of multiple tasks. The former indicates the performance when processing the same task but with different input samples. As multiple samples can be processed either one by one or in a chunk (i.e., batch), performance measured should also consider the effect of batch processing. For example, batch processing could reuse the loaded on-chip weights among the same layers for different samples, thus reducing the memory accesses for energy efficiency and performance improvement. The latter indicates the performance when processing different tasks with independent NN models and inputs. Similar to data-center systems, QoS is also introduced to measure the performance of NN systems.

For training, performance measurement is related to the time spent on training a network to its convergence. Researchers, especially the algorithm researchers, take great effort to both optimize the training method and leverage massive specialized hardware. For example, in 2017, Facebook took one hour on 256 GPUs to train the ResNet on ImageNet data set [98]. In 2018, Google improved the spent time to 2.2 min to train ResNet on ImageNet while using TPU v3 pod [99]. In 2019, this number was improved to only 74.4 s when training the RestNet on ImageNet with 2048 GPUs [100].

In addition to providing different measurement methodologis for different performance definitions, the measurement overhead should also be carefully considered in an SaNNS.

C. Accuracy Measurement and Recovery

1) Accuracy Measurement: Similar to the performance, measuring the accuracy, which is a unique feature of an NN system, differs a lot in inference and training phases.

For the inference phase, there exists a large gap between the final accuracy and computation results (such as the classification results) that is threefold. First, the NN vector readout is usually obtained with softmax or maximum function, where the relative relationship among the output neurons is more important than the actual value. For example, for image classification, once the winner neuron holds the largest value, the NN outputs the same classification label to the input image, that is, the classification result remains. Second, for each input sample in inference, we can never tell whether the NN result is correct without the input inference label. In real-life scenarios, there are usually no reference labels but the NN system is expected to give reliable labels for current inputs. Third, even if we are able to learn the correctness of a single input sample, we still cannot tell whether the accuracy is affected, since the accuracy is a statistical metric over multiple samples, such as the whole test data set. Therefore, in an SaNNS for textitinference, such a gap needs to be bridged to provide reliable results.

For the training phase, it is more complicated for measuring the accuracy. Although accuracy is measured based on inference correctness on a set of input samples, only the outputs of the loss function can be measured during training. Since the outputs of loss function do not have certain references, it is unclear whether the convergence speedup of NN is affected and whether the NN is well-trained when converged.

2) Error Resilience: Error resilience represents the ability of SaNNS to tolerate errors from input data and computation. In this article, we define the error as a broader scope which contains both the inexactness and approximation in the literature. Despite the error resilience naturally from the NN applications, SaNNS should also take care of other error resilience techniques, such as reexecution and error correction. Additionally, each error could possibly affect later executions, that is, error propagation, which should be carefully investigated.

3) Accuracy Recovery: As stated, though NN applications are typically tolerant to a certain extent of accuracy loss, error resilience techniques can be utilized for recovering accuracy sacrificed for performance/energy, especially for the cases when the application is very sensitive to the errors. A commonly used technique is retraining, where the NNs are retrained iteratively on given data sets. However, for real-life scenarios where the recovering effects are hard to obtain, recovering accuracy is hard to achieve with retraining techniques. Thus, it is worth exploring the alternatives for recovering accuracy in a cost-effective manner.

V. SELF-AWARE NN SYSTEM

In contrast to existing SaNNS where self-awareness is achieved from the architectural, circuit, or physical layers, where the self-awareness is in a very preliminary form, we propose to achieve self-awareness at a higher level, that is, model layer. From the perspective of NN models, we design a system called MinMaxNN that can achieve high efficiency by dynamically switching different NN models. MinMaxNN is able to monitor the execution accuracy to selectively deploy the NN models based on the decision made from the predicted results. MinMaxNN leverages two NN models, the min model which is small, fast, and efficient but with relatively low accuracy, and the max model which is large, slow but with high accuracy. MinMaxNN deploys the min model for unimportant inputs and max model for only crucial inputs. Furthermore, the MinMaxNN is able to adjust its sparsity dynamically in each layer for higher efficiency. To the best of our knowledge, MinMaxNN is the first SaNNS from the perspective of NN models.

The intuition of designing such a model-switching and sparsity-elastic MinMaxNN is twofold. First, the importance of different inputs varies significantly and using only an accurate model could lead to an inefficient solution. For many real-life applications such as cameras, running the max model for video inputs without moving objects is unnecessary and inefficient. Instead, running the min model for these inputs is a more efficient alternative. In this case, MinMaxNN could perform detection using the min model and recognition using the max model for both effectiveness and efficiency. Second, despite the efficiency of sparse networks, its adaption to different scenarios is also challenging for life-critical applications. In order to prevent significant loss of accuracy, current sparse NNs usually require long-time retraining and iterative fine-tuning process. The retrained networks with fixed sparsity are sensitive to retraining data sets and hard to be applied to other data sets. To adapt to different scenarios while still leveraging the efficiency of sparsity, we propose the sparsity-elastic mechanism in MinMaxNN.

As introduced in Section IV, there are several key issues to address for designing MinMaxNN. Regarding the system modeling (Key Issue A), we propose two models in the proposed self-learning profiler and the GM, respectively. In the self-learning profiler, a model is deployed for characterizing the relationship between current sparsity configuration and the network outputs [output quality measurement, such as mean square error (MSE)]. In the GM, a model is used for characterizing the relationship between current output quality and final accuracy. Regarding the performance measurement (Key Issue B), since the proposed MinMaxNN mainly focuses on inference-only scenarios currently, measurement simply takes the time spent on each input sample into consideration. Regarding the accuracy-related issues (Key Issue C), we propose a self-adaptive executor to adjust the network sparsity for accuracy recovery and it also periodically re-executes some input samples with the dense model (i.e., max model) to update the GM.

In the following parts of this section, we will elaborate and evaluate the architecture of the proposed model-switching and sparsity-elastic MinMaxNN system.

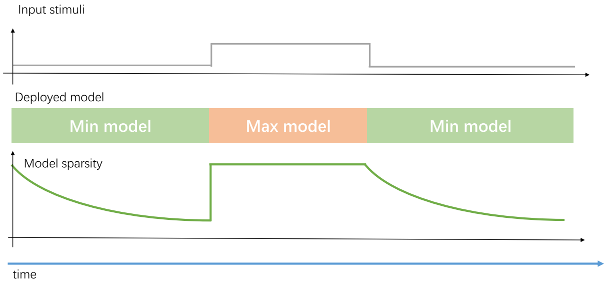

Fig. 13. ***Proposed working flow of MinMaxNN.***

A. Preliminaries

1) Min-Max NNs: We propose the min–max model switching mechanism, where the max model is large and accurate for highly reliable results, and the min model is small with acceptable accuracy for high performance/energy efficiency. In this article, we use the AlexNet as the min model and the ResNet-50 as max model.[1] Note that the min model and max can vary according to design requirements, and we select AlexNet and ResNet-50 in this article as an illustrative example.

2) Overall Working Flow: Fig. 13 illustrates the working flow of MinMaxNN. Roughly, when processing a certain inference task (e.g., Classification and object tracking) on a continuous input video stream, MinMaxNN is able to perform such task using either min model or max model. As the two pretrained models have different accuracy results and efficiency results, MinMaxNN is able to dynamically change the deployed model based on the input stimuli measurement. The input stimuli measurement is defined based on the application scenarios, such as the number of recognized objects. The MinMaxNN commonly uses the min model to process each input frame and regularly

[1] AlexNet has 0.73 GFlops in total with its five CONV layers and three FC layers. ResNet-50 has 5.2× more operations (3.8 GFlops) than AlexNet with its 49 CONV layers and 1 FC layer.

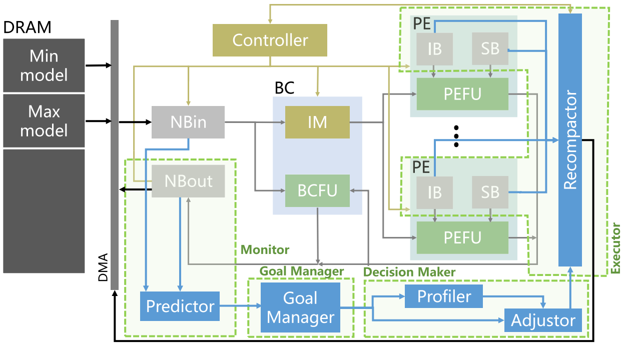

Fig. 14. ***Architecture of MinMaxNN.***

monitors the input stimuli. Once the input stimuli measurement varies drastically to larger/smaller than a predefined threshold, MinMaxNN will change to the max model to process these inputs until the input stimuli return to lower state. Note that MinMaxNN will gradually increase the model sparsity with monitoring the outputs quality to leverage the sparsity in the network for delivering high efficiency.

B. Architecture

Fig. 14 shows the overall architecture of MinMaxNN, which consists of two on-chip buffers for neurons (NBin and NBout), a BC module, several PEs, a controller, and several self-awareness functional units. The NBin and NBout load inputs for BC and store outputs to DRAM from BC. The BC features an IM for indexing needed neurons for each PE and a BC functional unit (BCFU) to perform operations that PEs are incapable of. MinMaxNN adopts a distributed manner to organize computations allocated to different PEs. Each PE contains local IBs for indexes and an SB together with a PE functional unit (PEFU) to perform the major computation of NNs. The control unit, that is, controller, manages all other modules in MinMaxNN.

1) Computing Modules: MinMaxNN contains three computing modules, that is, PEFU distributed in each PE and BCFU in BC [see Fig. 14 (green blocks)]. Each PEFU performs the basic inner production operation and vector elementary operations with its N multipliers and N-in adder tree. The BCFU in BC performs the remaining operations in processing NNs, including activation functions, local accumulation, and transcendental functions. For performing the NN computations, MinMaxNN will load inputs to NBin and synapses together with indexes to SBs and IBs. The loaded inputs will be sent to BC for filtering out the necessary neurons, based on indexes sent from each PE (static sparsity) and neuron values (dynamic sparsity). The selected results will be distributed to PEs for the major computation of NN further. Also, the results are collected from each PE to directly store to NBout, or processed then stored to NBout.

2) Storage Modules: MinMaxNN contains three types of on-chip storage based on stored data, that is, neural buffers, SBs, and control buffer. Neuron buffers (i.e., NBin and NBout) store the neurons, including inputs and output results. Since the neurons are shared among different PEs, we place the selection processing outside PEs and thus have the IM module in BC. The IM module filters only needed neurons based on the indexes from IB in each PE and transfers the only needed neurons to each PE. In order to maximally utilize the N multiplier in each PEFU, we allow the MinMaxNN to filter $N \times N$ neurons at a time, so as to provide at least N useful data for each PE. Also, the neurons which are zero-valued (e.g., due to ReLU activation) will also be filtered out in IM. SBs are distributed in each PE locally. Since the synapses are seldom shared among different PEs that are working on different output neurons, each SB only needs to store needed synapses locally. Indexes for neurons are also locally stored in each PE. In MinMaxNN, we adopt a binary mask indexing method, where 1-bit stands for whether the synapse exists in the network with stored “1” or “0.”

3) Controller Modules: The controller takes the responsibility to navigate the whole process, including data movements, memory reads and writes, and execution coordination. It loads instructions from its inner instruction buffer, decodes them and sends signals to all related control registers. We use our library-based compiler to generate the VLIW-style instructions.

4) Self-Awareness Modules: Different from the Cambricon-X accelerator (see Fig. 6), MinMaxNN consists of four new modules, that is, Predictor, GM, Profiler, and Recompactor. The Predictor predicts whether the next inputs will be critical based on the stored history records and the changes among inputs and outputs. The GM manages the goals that MinMaxNN targets at—detecting whether the deployed model needs to be switched, checking whether sparsity is acceptable for current inputs. The Profiler maintains the relationship between sparsity and network errors, and the relationship between NN models and accuracy. The Recompactor changes the sparsity of NNs and compresses the network to a compact form (or vice versa).

The above new modules together perform the functionalities mentioned in Fig. 5, that is, monitoring, goal management, decision making, and execution.

a) Monitoring: The Predictor performs the functionality of monitoring on two different types of data, that is, inputs and outputs. The former is related to model switching, through monitoring the input stimuli whether changed drastically. The latter is related to model sparsity, through monitoring the quality of outputs, such as correctness, MSE, intersection over union (IoU), or signal-to-noise ratio (SNR), and expected average overlap (EAO). The Predictor also records the above information to its stored history, so as to predict whether the MinMaxNN needs to be adjusted.

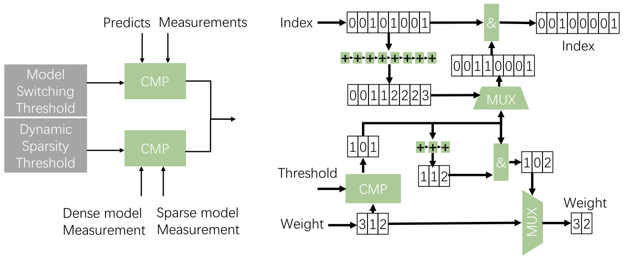

Fig. 15. ***Left: GM. Right: Recompactor.***

b) Goal management: The GM performs the functionality of goal management. Fig. 15 (left) shows the details of the GM module, which contains two different parts for managing the model switching and dynamic sparsity, respectively. For model switching, the GM receives the predicted results and current measurements from the Predictor. Compared to its recorded threshold, the GM sends the comparison results to both the Profiler and the Adjustor. For dynamic sparsity, during the sensing period, NBout sends its stored network outputs of both the dense model and the sparse model to the GM for further evaluating whether current sparsity meets the accuracy limitation. The GM records the threshold of measurement which is predefined and compares the new measurement results of the sparse model from the dense model. Also, the results are sent to both the Profiler and the Adjustor.

c) Decision making: For model switching, the Profiler decides whether to switch the NN from the min model to the max model if input stimuli measurement is larger than a threshold and from the max model to the min model if input stimuli return to a steady state. For dynamic sparsity, after reading the comparison results (correctness, MSE, SNR, etc.) from the GM, the Profiler performs the functionality of decision making. Particularly, the Profiler maintains the relationship between the sparsity and measurements by profiling the sparse model and dense model when executing the same inputs. When the current measurement is larger/smaller than a threshold with a certain of amount, the Profiler finds the suitable sparsity with its stored profiling results. In order to avoid drastic shifting in output quality, the Profiler adopts a conservative strategy to adjust the network sparsity. For example, if current MSE is 10% larger than the MSE threshold, the Profiler will reduce the sparsity of all layers—such as reducing by a factor of 2%.

d) Execution: The role of execution is achieved by the Adjustor and the Recompactor. For model switching, the Adjustor notifies the controller with the entry address of new program in DRAM, so that MinMaxNN is able to execute a new model for next inputs. The Recompactor is not utilized for model switching. For dynamic sparsity, the Adjustor notifies the Recompactor with target sparsity after the sparsity is determined by the Profiler. The Recompactor compacts the synapses into the new sparsity form in SB in each PE, where only the useful synapses are kept. Fig. 15 (right) shows the detailed architecture of the Recompactor. Roughly, Recompactor first uses the threshold to filter the unimportant synaptic weights, where the corresponding weights and indexes will be set to zeros. The zeros in weights will be removed so as to store the weights into a compacted form. Also, the Recompactor updates the indexes in IB. In order to keep high efficiency, the Recompactor will not immediately compact the network model for the next inputs. Instead, the Recompactor works when processing the next input, and thus avoids reloading the synapses twice. The recompacted network model will be used for the following new inputs.

C. Model Switching

MinMaxNN leverages two NN models, that is, the min model and the max model for unimportant inputs and crucial inputs, respectively. We train the two models, that is, AlexNet and ResNet-50 in this article, separately to obtain their normal accuracy. In the inference processing, MinMaxNN is able to switch between the two different models based on certain criteria. In some cases where MinMaxNN is able to perform online learning, MinMaxNN can also update its two models periodically.

MinMaxNN switches its running model based on the input stimuli measurement of the latest frames of a video. In this article, we use the output confidence score as the input stimuli measurement, which indicates the degree of difficulty of recognition in the current frame in object recognition tasks. More specifically, the small model is switched to the large one when the average confidence score on the latest frames is below a certain threshold, and the large model is switched to the small one when the score is larger than another threshold value. During the processing of video streams, the difficulty of recognition is determined by the confidence score due to the similarity of nearby video frames. Thus, MinMaxNN can automatically switch models according to the recognition difficulty of frames, so that it can run a large model for difficult frames to ensure recognition accuracy and run the small model for easy frames to save energy.

D. Elastic Sparsity

MinMaxNN is able to perform elastic sparsity which adjusts the network sparsity for delivering both high performance and efficiency under the accuracy constraint. The procedure is illustrated in Fig. 16. MinMaxNN selects frames from the video stream periodically and uses those frames as keyframes to adjust sparsity. For a keyframe at time T, MinMaxNN will perform both the dense model and the sparse model (both are the min model) for this frame to obtain the difference of output quality between these two models. The measured results will be sent to the GM and the Profiler to decide whether current output quality (such as correctness, MSE, IoU, SNR, or EAO, etc.) is acceptable. If acceptable, for frame $T + 1$ and

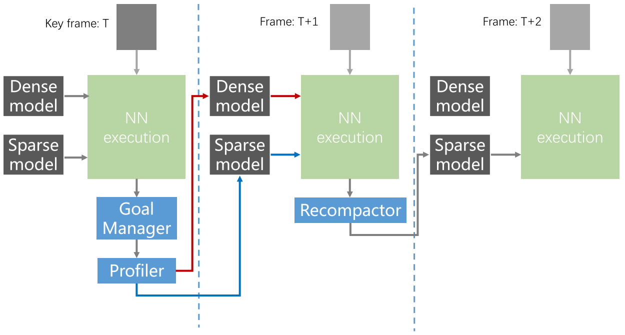

Fig. 16. *MinMaxNN will perform recompact if the measurement of key frame is unsatisfied (red path in processing frame $T+1$), or keep the current sparse model for processing the following frames (blue path in processing frame $T+1$).*

following frames till the next keyframe, MinMaxNN will keep processing with the same sparse model (blue path in processing frame $T+1$). If not, the sparsity will be changed while processing frame $T+1$, where MinMaxNN will perform the dense model for the current frame as well as recompacting the synapses to newly specified sparsity (red path in processing frame $T+1$). After recompacted, MinMaxNN will perform frame $T+2$ and the following frames use the new sparse model.

E. System Evaluation

1) NN Systems: We build our self-aware MinMaxNN based on the state-of-the-art NN system, that is, Cambricon-X [20], which is self-unaware and runs the same model all the time. To guarantee a fair comparison for hardware characteristics, we implement the MinMaxNN with the same parameters as the Cambricon-X for the basic computation components and on-chip storages. Specifically, the MinMaxNN system contains 16 PEs, where each PE contains 16 multipliers and a 16-in adder tree. In the BCFU, the MinMaxNN system contains 16 ALUs for processing results from the 16 PEs in parallel. We use 8-kB NBin, 8-kB NBout, and 16 × 2-kB SB in these two systems. We use Verilog for writing RTL description of the MinMaxNN system and Synopsys tool-chain with TSMC 65nm library for synthesizing, placing, and routing, to obtain the hardware characteristics. To obtain the exact performance, we build a cycle-accurate simulator with hardware parameters. The energy results are evaluated based on the simulation trace from simulator and hardware characteristics from the layout. Energy consumption of DRAM accesses is estimated with CACTI 6.0 [101]. In a nutshell, with self-awareness, MinMaxNN is able to achieve an average speedup of 5.64× with only 19.66% energy with the min–max model. The extra costs of self-awareness are also negligible, only 0.036-mm^2 extra area and 2.72-mW extra power on self-awareness modules, 0.56% and 0.29% of the total area and power costs of Cambricon-X.

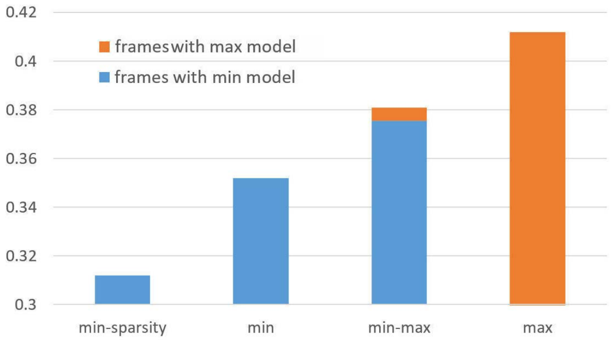

Fig. 17. *Average accuracy of MinMaxNN accelerator on VOT2018.*

2) Benchmarks: We select SiamRPN++ [102] model and VOT2018 [103] data set as our benchmark. The SiamRPN++ model is a single-target tracking model built on different backbone models. We choose an AlexNet backbone as the min model and a ResNet-50 backbone as the max model. We evaluate the models on each video separately, and report averaged speedups compared to the baseline system. We also report relative energy savings. We use EAO [104] scores as the accuracy metrics. The definition of EAO requires to be evaluated on all videos in the data set, thus we do not report the accuracy of individual videos.

3) Accuracy: Fig. 17 reports the EAO scores of our four models, that is, min-sparsity model—min model with elastic sparsity enabled, min model—min model only with elastic sparsity disabled, the min–max model—model switching between min and max model with sparsity enabled, and max model—max model only without elastic sparsity disabled. Unsurprisingly, the min-sparsity model lowers the high accuracy achieved by the max model to the worst accuracy, 0.10 reduction in EAO score. But it can be observed that MinMaxNN running the min–max model achieves comparable accuracy as max model. The min–max model improves the EAO score to 0.381 from 0.312 of min-sparsity model, 0.352 of min model, filling 69% and 48% of accuracy gaps, respectively, with using the max model for only 8% video frames.

4) Performance: Fig. 18 reports the speedups of the above four models running on MinMaxNN. MinMaxNN running the min–max switching model with elastic sparsity

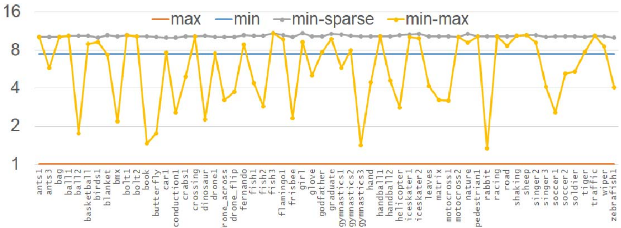

Fig. 18. *Average speedup of MinMaxNN accelerator on all benchmark videos in VOT2018.*

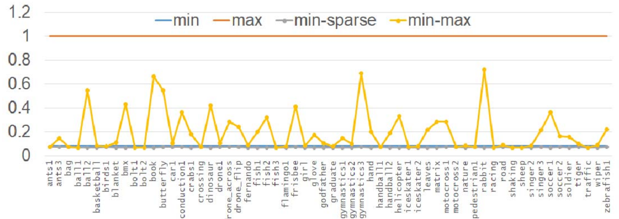

Fig. 19. ***Average energy consumption of MinMaxNN accelerator on all benchmark videos in VOT2018.***

enabled, achieves 5.64× speedup over the max model. It can also be observed that MinMaxNN with min and min-sparsity models could achieve 7.42×, 10.34× speedups when compared to the max model, respectively. However, without self-aware model switching, MinMaxNN is unable to recover from the bad results, leading to low accuracy results.

5) Energy: The relative energy consumption of MinMaxNN and baseline system is shown in Fig. 19. MinMaxNN running with the min–max model consumes 19.66% energy of the total energy of the max model, while MinMaxNN running with min model consumes 8.49% energy of the max model.

In a nutshell, MinMaxNN, which is self-aware of the models, could achieve 5.64× speedups and 19.66% energy costs when comparing to the full dense model, with acceptable accuracy.

6) Runtime Tradeoff: In the runtime, MinMaxNN switches between min model and max model with elastic sparsity, leading to different speedups and energy savings. In Fig. 20, we show the runtime results of two videos, iceskater2 and girl, as driving examples to illustrate such trade-offs between sparsity and benefits, including speedups and energy savings. On the iceskater2 video, MinMaxNN is able to achieve an average sparsity of 23.11%, an average speedup of 8.02×, and 11.35× energy savings with the min–max model when compared against the max model. Particularly, the video has a clear background and a nearly steady object (the female ice skater), thus the sparsity ratio gradually increases from 15% at the beginning to 32% at the 580th frame. At the 600th frame, the ice skater under tracking rotates and becomes blurred. This frame triggered the model to switch to the max model for the five consecutive frames, due to the increasing of output confidence score from the significant variation of input. Meanwhile, the GM found that the output of the pruned min model has a significant difference compared to the output of the original min model, and decides to reset to the beginning pruning rate (15%).

On the girl video, MinMaxNN is able to achieve an average sparsity of 26.68% with an average speedup of 7.63× and an average energy saving of 10.42 with the min–max model when compared with the max model. The video tracks the little girl among pedestrians and vehicles. The output confidence score can become very low when the girl is overlapped. Precisely, MinMaxNN switches to the max model for those low score cases, which happens four times during the whole video. Fig. 20 shows when a pedestrian covers the girl, the max model is switched on; while during other frames, the GM keeps increasing the sparsity.

F. Discussion

1) Sparsity: In MinMaxNN, sparsity is carefully leveraged in two ways. First, sparse networks can be directly supported by MinMaxNN. MinMaxNN runs a sparse network while using the IM model to filter input neurons for sending only needed neurons to all the PEs, that is, processing the static sparsity. The neurons that have a value of zero are filtered out, that is, processing the dynamic sparsity. Second, most importantly, MinMaxNN is able to gradually turn a dense model into a sparse model by recompacting the network model using the Recompactor (see Section V-D). Thus, in MinMaxNN, both the min and max models can be retrained so as to achieve better performance and energy efficiency.

2) Multimodel Switching: In this article, our implemented MinMaxNN mainly focuses on model switching between only two candidates, that is, Min and Max models, without considering the multimodel cases. The simplest way to realize multimodel switching is to deploy multiple usable models in the DRAM, and the NN system determines which model to run based on the information monitored and goals predefined in the GM. However, this may lead to the extra burden of training multiple models, which is costly in terms of time and energy. Another possible solution could be supporting adaptive networks such

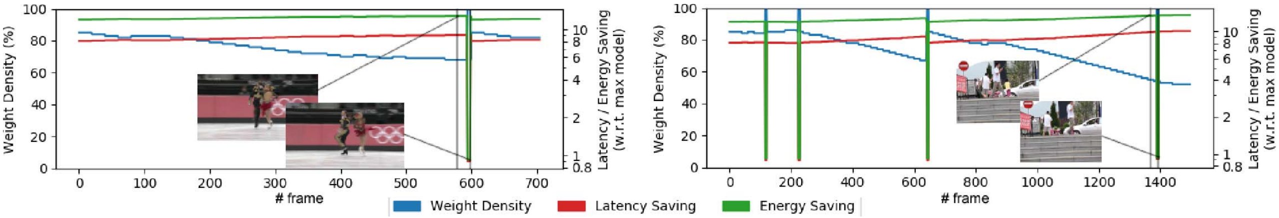

Fig. 20. ***Runtime results: weight density (i.e., the number of nonzero values to the total number of weights), speedups, and energy savings (left: iceskater2. right: girl).***

as networks proposed in [105] where the residual networks can skip several layers in its inference process. These adaptive networks can also be supported by MinMaxNN directly since it is also a model-level technique that can directly be reflected in the instruction sequences.

3) Comparison With Other SaNNS: In this article, rather than the architectural, circuit, and physical layer, MinMaxNN benefits from the self-awareness in the model layer for three advantages.

1) *High-efficiency:* MinMaxNN is able to directly improve the hardware efficiency of accelerators, that is, 5.64× speedup, with negligible costs of self-awareness modules, that is, less than 1% cost in area and power. At the architectural layer, the effective technique of sparsity could only achieve 3× speedup over the dense networks [20]. At the circuit layer, the probabilistic CMOS techniques could only lead to 2× performance improvement [36]. Other solutions like directly working on max model with static/dynamic precision scaling [91], [106] could only bring ∼2× energy benefit or ∼2× performance improvement (estimated by the achieved sparsity).
2) *Easy implementation:* MinMaxNN still leverages the traditional digital circuit implementation flow, which does not require complex, customized design flow as many techniques at the circuit and physical layer. Design with the new features, such as dual-voltage, over-clocking, and voltage scaling, requires self-managed design flow, as they cannot be directly supported by current CMOS design tools [29]–[31].
3) *Scalability:* Most importantly, most of the self-awareness techniques at the architectural, circuit, and physical layers can be applied in MinMaxNN for further efficiency improvement, for example, coarse-grained sparsity [21].

VI. CONCLUSION

We utilize an SaNNS framework to review existing works and offer the following three conclusions.

1) Existing NN processors exploit self-awareness at three levels: the architectural level, the physical level, and the circuit level. At the architectural layer, SaNNS can be characterized from a data-centric perspective where different data properties (i.e., data value, data precision, dataflow, and data distribution) are exploited. At the physical layer, various parameters of physical implementation are considered for trading high efficiency. At the circuit layer, different logics and devices can be used for high efficiency with more aggressive and fine-grained policies for self-awareness.
2) Self-awareness utilization in existing SaNNS is still at an early stage. Existing SaNNS exploit self-awareness typically at relatively low levels (i.e., the architecture, physical, circuit layers) without exploiting high-level application behaviors and algorithm semantics.
3) We propose to implement the self-awareness at the model layer, taking the application characteristics into consideration for high efficiency, which are absent in existing works. The proposed MinMaxNN system is able to leverage the model layer self-awareness with 5.64× and 19.66% performance improvement and energy reduction, respectively, without notable loss of accuracy and negative effects on developers' productivity. ■

REFERENCES

[1] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," Sep. 2014, p. 34, *arXiv:1409.0575*. [Online]. Available: http://arxiv.org/abs/1409.0575

[2] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech Language Process.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2015.

[3] M. Denkowski and A. Lavie, "Meteor universal: Language specific translation evaluation for any target language," in *Proc. 9th Workshop Stat. Mach. Transl.*, 2014, pp. 376–380. [Online]. Available: http://www.aclweb.org/anthology/W/W14/W14-3348

[4] A. Sharma, F. M. Schuhknecht, and J. Dittrich, "The case for automatic database administration using deep reinforcement learning," 2018, *arXiv:1801.05643*. [Online]. Available: http://arxiv.org/abs/1801.05643

[5] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *Proc. 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 367–382.

[6] T. Chen *et al.*, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, Salt Lake City, UT, USA, 2014, pp. 269–284. [Online]. Available: http://dl.acm.org/citation.cfm?id=2541967

[7] Y. Chen *et al.*, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2014, pp. 609–622.

[8] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. 42nd Annu. Int. Symp. Comput. Archit.*, 2015, pp. 92–104.

[9] T. Chen *et al.*, "A small-footprint accelerator for large-scale neural networks," *ACM Trans. Comput. Syst.*, vol. 33, no. 2, pp. 1–27, May 2015.

[10] K. Hegde, R. Agrawal, Y. Yao, and C. W. Fletcher, "Morph: Flexible acceleration for 3D CNN-based video understanding," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2018, pp. 933–946. [Online]. Available: https://arxiv.org/pdf/1810.06807.pdf

[11] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 688–698.

[12] M. Riera, J.-M. Arnau, and A. Gonzalez, "Computation reuse in DNNs by exploiting input similarity," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 57–68.

[13] M. Buckler, P. Bedoukian, S. Jayasuriya, and A. Sampson, "EVA^2: Exploiting temporal redundancy in live computer vision," in *Proc. 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 533–546.

[14] M. Song, J. Zhao, Y. Hu, J. Zhang, and T. Li, "Prediction based execution on deep neural networks," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 752–763.

[15] J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 1–14.

[16] F. Tu, W. Wu, S. Yin, L. Liu, and S. Wei, "RANA: Towards efficient neural acceleration with refresh-optimized embedded DRAM," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2018, pp. 340–352.

[17] 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 *Proc. 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 662–673.

[18] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. 45th Annu. Int. Symp. Comput. Archit.*, 2017, pp. 764–775. [Online]. Available: http://arxiv.org/abs/1712.01507

[19] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *Proc. 23rd Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2018, pp. 461–475.

[20] S. Zhang *et al.*, "Cambricon-X: An accelerator for

sparse neural networks," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2016, pp. 1–12.

[21] X. Zhou *et al.*, "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2018, pp. 15–28.

[22] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 548–560.

[23] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 535–547. [Online]. Available: http://arxiv.org/abs/1607.00064

[24] P. Judd, J. Albericio, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, vol. 6056, Oct. 2016, pp. 1–12. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7529197

[25] M. Mahmoud, K. Siu, and A. Moshovos, "Diffy: A Déjà vu-free differential deep neural network accelerator," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2018, pp. 134–147.

[26] S. Han *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, vol. 16, Jun. 2016, pp. 243–254. [Online]. Available: http://arxiv.org/abs/1602.01528%5Cnhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7551397, doi: 10.1109/ISCA.2016.30.

[27] A. Parashar *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2017, pp. 27–40.

[28] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-neuron-free deep neural network computing," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 1–13.

[29] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Vancouver, BC, Canada, 2012, pp. 449–460.

[30] M. D. Kruijf, S. Nomura, and K. Sankaralingam, "Relax: An architectural framework for software recovery of hardware faults," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Saint-Malo, France, 2010, pp. 497–508.

[31] D. Ernst *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. 36th Annu. Int. Symp. Microarchit. (MICRO)*. Washington, DC, USA: IEEE Computer Society, Dec. 2003, pp. 7–18. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1253179

[32] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2014, pp. 27–32. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627613

[33] Y. Wang, J. Deng, Y. Fang, H. Li, and X. Li, "Resilience-aware frequency tuning for neural-network-based approximate computing chips," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2736–2748, Oct. 2017.

[34] 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 *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 488–490.

[35] S. Choi, J. Lee, K. Lee, and H.-J. Yoo, "A 9.02 mW CNN-stereo-based real-time 3D hand-gesture recognition processor for smart mobile devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 224–226.

[36] L. N. B. Chakrapani, J. George, B. Marr, B. E. S. Akgul, and K. V. Palem, "Probabilistic design: A survey of probabilistic CMOS technology and future directions for terascale IC design," in *VLSI-SoC: Research Trends in VLSI and Systems on Chip* (IFIP International Federation for Information Processing), vol. 249, G. De Micheli, S. Mir, and R. Reis, Eds. Boston, MA, USA: Springer, 2008.

[37] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1223–1235, Aug. 2015.

[38] J. Deng *et al.*, "Retraining-based timing error mitigation for hardware neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2015, pp. 593–596.

[39] S. Sarma, N. Dutt, P. Gupta, N. Venkatasubramanian, and A. Nicolau, "CyberPhysical-system-on-chip (CPSoC): A self-aware MPSoC paradigm with cross-layer virtual sensing and actuation," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2015, pp. 625–628.

[40] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[41] S. Kounev, "Engineering of self-aware IT systems and services: State-of-the-art and research challenges," in *Computer Performance Engineering. EPEW* (Lecture Notes in Computer Science), vol. 6977, N. Thomas, Eds. Berlin, Germany: Springer, 2011.

[42] S. Kounev, X. Zhu, J. O. Kephart, and M. Kwiatkowska, "Model-driven algorithms and architectures for self-aware computing systems (Dagstuhl seminar 15041)," Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, Tech. Rep., 2015, vol. 5, no. 1, doi: 10.4230/DagRep.5.1.164.

[43] A. Jantsch, N. Dutt, and A. M. Rahmani, "Self-awareness in systems on chip—A survey," *IEEE Des. Test*, vol. 34, no. 6, pp. 8–26, Dec. 2017.

[44] W. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biol.*, vol. 52, nos. 1–2, pp. 99–115, 1990.

[45] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*. [Online]. Available: http://arxiv.org/abs/1406.1078

[46] S. Liu *et al.*, "Cambricon: An instruction set architecture for neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, Jun. 2016, pp. 393–405, doi: 10.1109/ISCA.2016.42.

[47] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, Feb. 2017, pp. 541–552, doi: 10.1109/HPCA.2017.55.

[48] P. Chi *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, Jun. 2016, pp. 27–39, doi: 10.1109/ISCA.2016.13.

[49] R. LiKamWa, Y. Hou, Y. Gao, M. Polansky, and L. Zhong, "RedEye: Analog ConvNet image sensor architecture for continuous mobile vision," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, Jun. 2016, pp. 255–266, doi: 10.1109/ISCA.2016.31.

[50] X. Wang, J. Yu, C. Augustine, R. Iyer, and R. Das, "Bit prudent in-cache acceleration of deep convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 81–93.

[51] C. Eckert *et al.*, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *Proc. 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 383–396. [Online]. Available: http://arxiv.org/abs/1805.03718

[52] H. Mao, M. Song, T. Li, Y. Dai, and J. Shu, "LerGAN: A zero-free, low data movement and PIM-based GAN architecture," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2018, pp. 669–681.

[53] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," 2014, *arXiv:1408.5093*. [Online]. Available: http://arxiv.org/abs/1408.5093

[54] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," 2016, *arXiv:1605.08695*. [Online]. Available: http://arxiv.org/abs/1605.08695

[55] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*. [Online]. Available: http://arxiv.org/abs/1512.01274

[56] A. Paszke *et al.*, "Automatic differentiation in PyTorch," in *Proc. NIPS Autodiff Workshop*, 2017, pp. 1–4.

[57] F. Chollet *et al.*, "Keras," Tech. Rep., 2015.

[58] L. Yeager, J. Bernauer, A. Gray, and M. Houston, "DIGITS: The deep learning GPU training system," in *Proc. ICML AutoML Workshop*, 2015, pp. 1–4.

[59] S. Dieleman *et al.*, "Lasagne: First release," Tech. Rep., Aug. 2015, doi: 10.5281/zenodo.27878.

[60] L. Truong *et al.*, "Latte: A language, compiler, and runtime for elegant and efficient deep neural networks," in *Proc. ACM SIGPLAN Conf. Program. Lang. Design Implement.*, 2016, pp. 209–223. [Online]. Available: http://dl.acm.org/citation.cfm?id=2908080.2908105

[61] S. Xu *et al.*, "Cavs: An efficient runtime system for dynamic neural networks," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*, 2018, pp. 937–950.

[62] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778, doi: 10.1109/CVPR.2016.90.

[63] W. Xiong *et al.*, "Achieving human parity in conversational speech recognition," 2016, *arXiv:1610.05256*. [Online]. Available: http://arxiv.org/abs/1610.05256

[64] A. Eriguchi, K. Hashimoto, and Y. Tsuruok, "Tree-to-sequence attentional neural machine translation," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2016, pp. 823–833.

[65] S. Webb, "Deep learning for biology," *Nature*, vol. 554, no. 7693, pp. 555–557, Feb. 2018.

[66] R. Ananthanarayanan, P. Brandt, M. Joshi, and M. Sathiamoorthy, "Opportunities and challenges of machine learning accelerators in production," in *Proc. USENIX Conf. Oper. Mach. Learn. (OpML)*. Santa Clara, CA, USA: USENIX Association, 2019, pp. 1–3. [Online]. Available: https://www.usenix.org/conference/opml19/presentation/ananthanarayanan

[67] H. A. Kholerdi, N. Taherinejad, and A. Jantsch, "Enhancement of classification of small data sets using self-awareness—An Iris flower case-study," *Proc. IEEE Int. Symp. Circuits Syst.*, May 2018, pp. 1–5.

[68] G. Gobieski, N. Beckmann, and B. Lucia, "Intelligence beyond the edge: Inference on intermittent embedded systems," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, pp. 199–213. [Online]. Available: http://arxiv.org/abs/1810.07751

[69] *Intel Performance Counter Monitor—A Better Way to Measure CPU Utilization*. Accessed: May 4, 2019. [Online]. Available: https://software.intel.com/en-us/articles/intel-performance-counter-monitor/

[70] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. W. Fletcher, "UCNN: Exploiting computational reuse in deep neural networks via weight repetition," in *Proc. 45th Annu. Int. Symp.

Comput. Archit., 2018, pp. 674–687. [Online]. Available: http://arxiv.org/abs/1804.06508

[71] H. Song, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.

[72] M. Rhu, M. O. Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA engine: Leveraging activation sparsity for training deep neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 78–91.

[73] A. Azizimazreah and L. Chen, "Shortcut mining: Exploiting cross-layer shortcut reuse in DCNN accelerators," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, no. 1, Feb. 2019, pp. 94–105.

[74] (2019). *Using Bfloat16 With Tensorflow Models*. [Online]. Available: https://cloud.google.com/tpu/docs/bfloat16

[75] Y. Chen, T. Chen, X. Zhiwei, and O. Temam, "DianNao family: Energy-efficient hardware accelerators for machine learning," *Commun. ACM*, vol. 57, no. 5, p. 109, 2014. https://10.1145/2594446%5Cnhttps://ejwl.idm.oclc.org/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=95797996&site=ehost-live

[76] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Toronto, ON, Canada, Jun. 2017, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/3079856.3080246

[77] NVIDIA Corporation. (2018). *NVIDIA Tesla V100 GPU Architecture*. [Online]. Available: https://images.nvidia.com/content/volta-architecture/pdf/voltaarchitecture-whitepaper.pdf

[78] Cambricon. *MLU100—Cambricon*. Accessed: May 24, 2019. [Online]. Available: http://www.cambricon.com/

[79] J. Albericio *et al.*, "Bit-pragmatic deep neural network computing," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchit.*, Oct. 2017, pp. 1–16.

[80] A. D. Lascorz *et al.*, "Bit-tactical: A software/hardware approach to exploiting value and bit sparsity in neural networks," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2019, pp. 749–763.

[81] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Seoul, South Korea, Jun. 2016, pp. 367–379, doi: 10.1109/ISCA.2016.40.

[82] W. Lu, G. Yan, J. Li, S. Gong, Y. Han, and X. Li, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Austin, TX, USA, Feb. 2017, pp. 553–564, doi: 10.1109/HPCA.2017.29.

[83] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized coarse-grained dataflow for scalable NN accelerators," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS)*, 2019, pp. 807–820, doi: 10.1145/3297858.3304014.

[84] A. Ren *et al.*, "ADMM-NN: An algorithm-hardware co-design framework of DNNs using alternating direction method of multipliers," in *Proc. 24th Int. Conf. Archit. Support Program. Lang. Oper. Syst.*, 2018, pp. 925–938. [Online]. Available: http://arxiv.org/abs/1812.11677

[85] C. Deng, S. Liao, Y. Xie, K. K. Parhi, X. Qian, and B. Yuan, "PermDNN: Efficient compressed DNN architecture with permuted diagonal matrices," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2018, pp. 189–202.

[86] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *Proc. 39th Annu. Int. Symp. Comput. Archit. (ISCA)*, 2012, pp. 356–367. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6237031

[87] A. Lingamneni *et al.*, "Energy parsimonious circuit design through probabilistic pruning," in *Proc. Design, Automat. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2011, pp. 1–6.

[88] Y. Fang, H. Li, and X. Li, "SoftPCM: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write," in *Proc. IEEE 21st Asian Test Symp.*, Nov. 2012, pp. 131–136.

[89] J.-F. Zhang, C.-E. Lee, C. Liu, Y. S. Shao, S. W. Keckler, and Z. Zhang, "SNAP: A 1.67–21.55 TOPS/W sparse neural acceleration processor for unstructured sparse deep neural network inference in 16 nm CMOS," in *Proc. Symp. VLSI Circuits*, 2019, pp. C306–C307.

[90] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "LNPU: A 25.3 TFLOPS/W sparse deep-neural-network learning processor with fine-grained mixed precision of FP8-FP16," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 142–144.

[91] M. Wess, S. M. P. Dinakarrao, and A. Jantsch, "Weighted quantization-regularization in dnns for weight memory minimization toward HW implementation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2929–2939, Nov. 2018.

[92] J. Lee, D. Shin, J. Lee, J. Lee, S. Kang, and H.-J. Yoo, "A full HD 60 fps CNN super resolution processor with selective caching based layer fusion for mobile devices," in *Proc. Symp. VLSI Circuits*, 2019, pp. C302–C303.

[93] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, "TIE: Energy-efficient tensor train-based inference engine for deep neural network," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 264–278.

[94] H. Tsai *et al.*, "Inference of long-short term memory networks at software-equivalent accuracy using 2.5 m analog phase change memory devices," in *Proc. Symp. VLSI Technol.*, 2019, pp. T82–T83.

[95] J. Yue *et al.*, "A 65 nm 0.39-to-140.3TOPS/W 1-to-12b unified neural network processor using block-circulant-enabled transpose-domain acceleration with 8.1× higher TOPS/mm^2 and 6T HBST-TRAM-based 2D data-reuse architecture," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 138–140.

[96] X. Si *et al.*, "A twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 396–398.

[97] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, and A. Moshovos, "Stripes: Bit-serial deep neural network computing," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Oct. 2016, pp. 1–12.

[98] P. Goyal *et al.*, "Accurate, large minibatch SGD: Training ImageNet in 1 hour," 2017, *arXiv:1706.02677*. [Online]. Available: http://arxiv.org/abs/1706.02677

[99] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," 2018, *arXiv:1811.06992*. [Online]. Available: http://arxiv.org/abs/1811.06992

[100] M. Yamazaki *et al.*, "Yet another accelerated SGD: ResNet-50 training on ImageNet in 74.7 seconds," 2019, *arXiv:1903.12650*. [Online]. Available: http://arxiv.org/abs/1903.12650

[101] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchit. (MICRO)*, Dec. 2007, pp. 3–14.

[102] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, "SiamRPN++: Evolution of siamese visual tracking with very deep networks," 2018, *arXiv:1812.11703*. [Online]. Available: http://arxiv.org/abs/1812.11703

[103] M. Kristan *et al.*, "A novel performance evaluation methodology for single-target trackers," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 11, pp. 2137–2155, Nov. 2016.

[104] M. Kristan *et al.*, "The visual object tracking vot2015 challenge results," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, Dec. 2015, pp. 1–23.

[105] M. Figurnov *et al.*, "Spatially adaptive computation time for residual networks," in *Proc. CVPR*, no. 1, Jul. 2017, pp. 1039–1048.

[106] Q. Zhang, T. Wang, Y. Tian, F. Yuan, and Q. Xu, "ApproxANN: An approximate computing framework for artificial neural network," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, no. 2, Apr. 2015, pp. 701–706.

ABOUT THE AUTHORS

Zidong Du (Member, IEEE) received the bachelor's degree in engineering from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2011, and the Ph.D. degree from the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing, in 2016.

He is currently an Associate Professor with the Intelligent Processor Research Center, ICT, CAS. His current research interests include artificial intelligence, machine learning, neural network, computer architecture, accelerator, inexact/approximate computing, and hardware designs.

Dr. Du has been awarded several important honors, including the Best Paper Award of Architectural Support for Programming Languages and Operating Systems (ASPLOS), special prize of Presidential Scholarship for postgraduate students of CAS.

Qi Guo (Member, IEEE) received the B.E. degree in computer science from Tongji University, Shanghai, China, in 2007, and the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2012.

From 2012 to 2014, he was a Staff Researcher at IBM Research, Beijing. From 2014 to 2015, he was a Postdoctoral Researcher with Carnegie Mellon University, Pittsburgh, PA, USA. He is currently a Professor with the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include computer architecture, programming models, and machine learning.

Yongwei Zhao received the bachelor's degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, China, in 2015. He is currently working toward the Ph.D. degree at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, and the University of Chinese Academy of Sciences, Beijing.

Tian Zhi received the bachelor's degree in biomedical engineering from Zhejiang University, Hangzhou, China, in 2009, and the Ph.D. degree from the Institute of Electronics, Chinese Academy of Sciences, Beijing, in 2014.

She is currently an Assistant Professor with the Intelligent Processor Research Center, Institute of Computing Technology, CAS.

Yunji Chen (Senior Member, IEEE) is currently a Full Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He also leads his laboratory to develop neural network processors. He has authored or coauthored one book and over 100 papers on various conferences, including the International Symposium on Computer Architecture (ISCA), High-Performance Computer Architecture (HPCA), the International Symposium on Microarchitecture (MICRO), Architectural Support for Programming Languages and Operating Systems (ASPLOS), the International Conference on Software Engineering (ICSE), the IEEE International Solid-State Circuits Conference (ISSCC), Hot Chips, the International Joint Conference on Artificial Intelligence (IJCAI), the association for computing machinery (ACM)/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), and the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA) and journals, including the IEEE Journal of Solid-State Circuits, the IEEE Transactions on Computers, the IEEE Transactions on Parallel and Distributed Systems, the IEEE Transactions on Image Processing, the IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, ACM TOCS, ACM TIST, the ACM Transactions on Architecture and Code Optimization (ACM TACO), the ACM Transactions on Design Automation of Electronic Systems (ACM TODAES), *ACM Computing Surveys*, and *IEEE Micro*.

Dr. Chen was a TPC member of MICRO 2014, ASPLOS 2015, IPDPS 2015, ISCA 2016, IPDPS 2016, MICRO 2016, DAC 2017, HPCA 2018, and MICRO 2018. He was a recipient of the ASPLOS 2014 and MICRO 2014 Best Paper Awards for advances in neural network processors. He was the General Co-Chair of ASPLOS 2017.

Zhiwei Xu (Senior Member, IEEE) received the B.S. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 1982, the M.S. degree from Purdue University, West Lafayette, IN, USA, in 1984, and the Ph.D. degree from the University of Southern California, Los Angeles, CA, USA, in 1987.

He is currently a Professor and the Chief Technology Officer (CTO) with the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS), Beijing. His prior industrial experience includes the Chief Engineer of Dawning Corporation (now Sugon as listed in Shanghai Stock Exchange), a leading high-performance computer vendor in Beijing, China. He currently leads the Cloud-Sea Computing Systems, a strategic priority research project of the CAS that aims at developing billion-thread computers with elastic processors.