Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey

GIORGOS ARMENIAKOS, National Technical University of Athens, Greece
GEORGIOS ZERVAKIS, Karlsruhe Institute of Technology, Germany
DIMITRIOS SOUDRIS, National Technical University of Athens, Greece
JÖRG HENKEL, Karlsruhe Institute of Technology, Germany

Deep Neural Networks (DNNs) are very popular because of their high performance in various cognitive tasks in Machine Learning (ML). Recent advancements in DNNs have brought levels beyond human accuracy in many tasks, but at the cost of high computational complexity. To enable efficient execution of DNN inference, more and more research works, therefore, are exploiting the inherent error resilience of DNNs and employing Approximate Computing (AC) principles to address the elevated energy demands of DNN accelerators. This article provides a comprehensive survey and analysis of hardware approximation techniques for DNN accelerators. First, we analyze the state of the art, and by identifying approximation families, we cluster the respective works with respect to the approximation type. Next, we analyze the complexity of the performed evaluations (with respect to the dataset and DNN size) to assess the efficiency, potential, and limitations of approximate DNN accelerators. Moreover, a broad discussion is provided regarding error metrics that are more suitable for designing approximate units for DNN accelerators as well as accuracy recovery approaches that are tailored to DNN inference. Finally, we present how Approximate Computing for DNN accelerators can go beyond energy efficiency and address reliability and security issues as well.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Hardware** → **Logic circuits**; • **Computer systems organization** → **Architectures**; • **Computing methodologies** → **Neural networks;**

Additional Key Words and Phrases: Approximate computing, arithmetic circuits, deep neural networks, error metrics, hardware approximation

ACM Reference format:
Giorgos Armeniakos, Georgios Zervakis, Dimitrios Soudris, and Jörg Henkel. 2022. Hardware Approximate Techniques for Deep Neural Network Accelerators: A Survey. *ACM Comput. Surv.* 55, 4, Article 83 (November 2022), 36 pages.
https://doi.org/10.1145/3527156

This work is partially supported by the German Research Foundation (DFG) through the project "ACCROSS: Approximate Computing aCROss the System Stack" HE 2343/16-1.
Authors' addresses: G. Armeniakos and D. Soudris, National Technical University of Athens, Athens, Greece; emails: {armeniakos, dsoudris}@microlab.ntua.gr; G. Zervakis and J. Henkel, Karlsruhe Institute of Technology, Karlsruhe, Germany; emails: {georgios.zervakis, henkel}@kit.edu.

0360-0300/2022/11-ART83 $15.00
https://doi.org/10.1145/3527156

1 INTRODUCTION

Advancements in **Deep Learning (DL)** with **Deep Neural Networks (DNNs)** have delivered beyond human levels of accuracy on many AI tasks [125]. An increasing number of embedded devices rely on DL and DNNs to deliver sophisticated services such as machine translation [15], object detection [73], healthcare [9, 77], and so forth. However, these accuracy improvements came at the cost of a vast increase in computational demands, leading to the emergence of customized hardware DNN accelerators [58, 125]. It is noteworthy that recent **Convolution Neural Networks (CNNs)** require tens of billions of **multiply-accumulate (MAC)** operations [125]. To satisfy such demands, DNN accelerators integrate thousands of MAC units; e.g., Google TPU [58] comprises 64K MACs, while Samsung's **neural processing unit (NPU)** contains 6K MAC units [91]. This immense number of MAC units combined with high parallelization results in high energy demands. This problem is intensified especially when considering the growth of Edge AI that requires even more complex **neural networks (NNs)** to operate on a wide spectrum of energy- and resource-restricted devices.

Over the past decade, **Approximate Computing (AC)** [42] has been established as a new design paradigm for energy-efficient circuits. AC goes beyond typical/emerging design approaches [96] and exploits the inherent ability of a large number of applications to produce results of acceptable quality, despite some errors (approximations) in their computations. Leveraging this property, AC approximates the hardware execution of the error-resilient computations in a manner that favors performance and energy [125]. Driven by the high potential for energy efficiency and exploiting the error tolerance of NNs [124, 136], research on approximate NN implementations has been rapidly growing over the last years. Figure 1 is a representative example of this trend. Figure 1 depicts the number of publications in three major design automation conferences that apply approximations in CNN inference.

Considering the high demand for edge AI [45], the billions of mobile devices running DNN inference, and the rapid growth of AI chips,[1] our focus in this survey is to study, analyze, and elucidate the impact of hardware approximation techniques on the efficiency and accuracy of DNN inference accelerators. Prior research on DNN accelerators reports that between 30% and 80% of the system energy is consumed by DRAM [63], with data movement dominating the energy consumption [130]. Still, the processing units (e.g., MACs) of DNN accelerators feature considerable power consumption [4, 125]. Hence, considering high utilization and continuous operation, high energy is also consumed by the processing units and could be prohibitive, for example, in battery-power-embedded devices [125]. In addition, the very high power consumed by the processing units in a confined area may lead to unsustainable power densities with far-reaching impact on the temperature, performance, and reliability of DNN accelerators [4]. Although several works examine approximate memories for DNNs [28, 63, 64, 107], such works are out of the scope of our survey, which focuses on computational approximation. Note, nevertheless, that compute-based (our survey) and memory-based approximations are mainly complementary. Finally, although approximate computing mainly targets energy efficiency in DNN accelerators (Sections 3 to 5), several works apply approximations to tackle reliability and security issues (Section 6).

The ever-increasing demand for efficient DNN inference and the prominent outcomes of AC applications have attracted significant research interest. As shown in Table 1, several surveys address similar topics to our work. A survey of approximate arithmetic units (e.g., adders and multipliers) is presented in [56]. Nevertheless, in [56], only a simple DNN use case example is used as a proof of concept. On the other hand, [98] presents a comprehensive study of approximate

[1]Google [58], Samsung [91], Intel [129], IBM [1], Huawei [72], Cerebras [13], Groq [36], Graphcore [35], Arm [7], NVIDIA [86], and so forth.

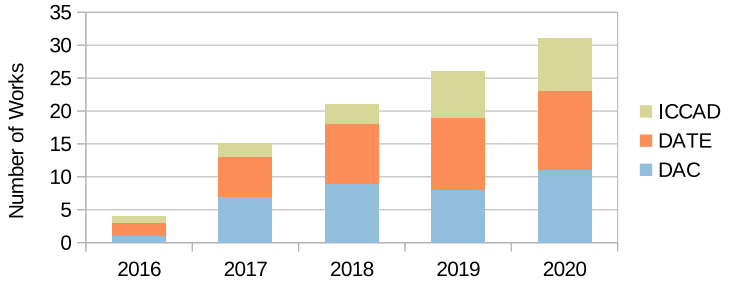

Fig. 1. Number of publications that apply any type of approximation on DNN inference. The past 5 years and three major design automation conferences are considered.

Table 1. Recent Relevant Surveys

Ref.	Year	Description/Focus
[56]	2020	Approximate arithmetic circuits
[98]	2018	Approximate circuits with limited discussion on DNN accelerators with emphasis on software DNN approximation
[125]	2020	Software-based approximation for DNN accelerators
[14]	2018	
[17]	2020	DNN accelerator architectures
[116]	2017	Software and hardware optimization for DNN accelerators
[12, 27]	2020	
[32]	2021	Quantization techniques
[71]	2021	Software-based pruning and quantization techniques
[99]	2021	NN architectures

circuits, discussing also DNN-specific approximation techniques. However, in [98], software-based approximation techniques (such as quantization and pruning) are mainly reviewed, while regarding hardware-based approximation, only a limited discussion based on approximate multipliers is included. In [125] and [14] the impact of DNN approximation techniques is reviewed with the main focus on software-based approaches. In [17], a survey of DNN accelerator architectures is provided, while [12] reviews hardware and software optimization methods for DNN accelerators. Similarly to [12], [116] and [27] present very comprehensive surveys on software optimizations/approximations and hardware architectures for DNNs. However, hardware DNN approximations are not the target of [12, 17, 27, 116]. Finally, [32, 71] present a thorough analysis of software-based approximation methods such as quantization and pruning, while [99] provides a comprehensive review of recent NN architectures. Approximate DNN accelerators are out of the scope of [32, 71, 99]. *On the other hand, our work surveys the state of the art of approximate DNN accelerators. Specifically, our work focuses and provides in-depth discussion of DNN-specific approximate techniques that are implemented at the hardware level (e.g., logic approximation) and/or modify the architecture of the accelerator.*

2 BRIEF BACKGROUND ON DEEP NEURAL NETWORKS

Deep neural networks consist of artificial neurons. The computation model of a neuron is illustrated in Figure 2 and given by Equation (1). Each neuron performs a weighted sum of all its inputs and then a bias term is added for a possible offset [12]. The result is passed through the activation function, from which the output of the neuron is obtained. Neurons are represented as

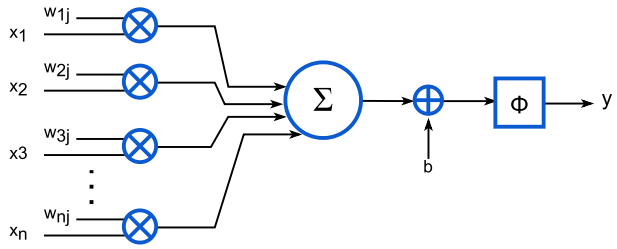

Fig. 2. Schematic of a neuron.

nodes in a graph and are organized in layers. In DL, a layer is a function that receives inputs from the previous layers and passes outputs to the next layers [33]. It is usually uniform, and it only consists of one type of activation function, pooling, convolution, and so forth.

$$y_j = \Phi\left(\sum_{k=0}^{n-1} x_k w_{kj} + b\right), \tag{1}$$

where y_j is the output of the neuron, w_{kj} are the neuron's weights, n is the number of weights, x_k are the neuron's inputs, b is the bias of the neuron, and Φ is the activation function.

The most popular and widely used neural networks today are Multi-Layer Perceptrons, Convolutional Neural Networks, Recurrent Neural Networks, and Transformers [58, 125]. Specifically:

(1) ***Multi-Layer Perceptrons* (MLPs)**: Each node in a layer is composed of a nonlinear function of a weighted sum of all the previous outputs (fully connected) [116].
(2) *Convolutional Neural Networks*: They are mainly composed of convolutional, pooling, and fully connected layers and exploit the concept of shared weights and are designed to learn spatial hierarchies of features [116].
(3) ***Recurrent Neural Networks* (RNNs)**: Each layer is composed of nonlinear functions of the weighted sums of the outputs and the previous state. **Long Short-Term Memory (LSTM)** is the most common RNN. The weights are reused across time steps. A key feature of LSTMs is to decide what to forget and what to forward to the next layer [50].
(4) *Transformers*: They handle sequential input data as RNNs, but they differ since they use a different mechanism called "self-attention" that weights the significance of each input part and enables parallel data processing [113].

The goal of our work is to survey the state of the art of hardware approximation techniques for DNN accelerators, without any constraints on the DNN type, though, as will be shown in Section 5.2, the majority of the examined works mainly use only CNNs in their analysis/evaluation.

2.1 Layers

2.1.1 Fully Connected (FC) Layers. In a **fully connected (FC)** layer, the input and output neurons are connected to each other by flattening the matrix into a vector. Every output neuron performs a weighted sum of every input neuron. Typically, as convolution layers, FC layers are followed by a non-linear activation and/or bias addition. FC layers are usually used as the classifier in the final stage of a DNN. Contrary to convolutional layers, which are compute intensive, FC layers are memory intensive due to the many neuron synapses.

2.1.2 Convolutional Layers. This layer carries the main portion of a network's computational load. It performs a dot product between two matrices, where the one matrix is an input feature map and the other is a set of weights known as kernel. Figure 3(a) illustrates the convolution op-

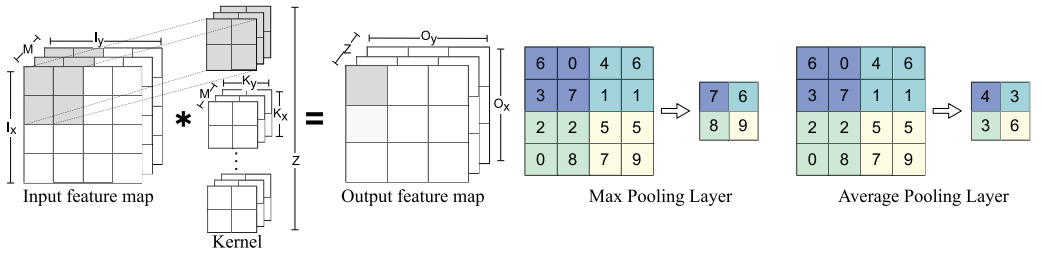

Fig. 3. (a) Convolution and (b) Max pooling and average pooling examples.

eration between an input of size $[I_x \times I_y \times M]$ and Z filters of size $[K_x \times K_y \times M]$. The depth of the output (output feature map) is Z. Once the output feature map is computed, typically the operation of pooling is performed. The size of the kernel depends on the size of the receptive field and consequently of the weight matrix. The distance between adjacent receptive fields is determined by the stride parameter. All neurons of a layer share the same weight matrix, trying to detect the same feature in different locations of the layer. To detect multiple features, a convolutional layer has many channels, i.e., many feature maps. Due to their high computational intensity,[2] convolution layers consist of the main approximation target, as Section 3 reveals.

2.1.3 Pooling Layers. Pooling layers are placed after the convolutional layers. Their primary use is to reduce the number of activations of a layer and thus reduce the memory demands and computations needed in the later layers. This layer substantially down-samples outputs by returning a single value of each group depending on the pooling strategy, e.g., max-/average-pooling (Figure 3(b)). In max-pooling, the maximum value of the nearby neurons is the output, while in average-pooling, the output is their average value. As Figure 3(b) shows, the inputs of the next layers are significantly reduced. The pooling layer type can be exploited to apply customized approximation (see Section 3).

2.1.4 Activation Functions. Activation functions are non-linear transformations that are applied after the weighted sum of the inputs of a neuron. The activation function increases the fitting ability of NNs and helps solve complex problems that cannot be solved with linear algebra. The most commonly used activation function is the **Rectified Linear Unit (ReLU),** which forces negative values to be zero and keeps positive values unchanged:

$$y = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise.} \end{cases} \tag{2}$$

Some other activation functions are Sigmoid and TanH, which normalize the output in the range of $(0, 1)$ and $(-1, 1)$, respectively, while the Softmax function normalizes numbers in the range of $(0, 1)$ with the restriction that their sum should definitely be equal to 1. Many works leverage the activation function to apply optimized approximation (see Section 3). For example, when ReLu is used, the overall accuracy is mainly defined by the accuracy of the positive values.[3]

2.1.5 Normalization Layers. These layers exploit the fact that neural networks have usually a normal distribution and help keep input values in the same range. The latter speed up the training process and use higher learning rates so that layers do not have to adapt to a different distribution

[2]GEMM operations consume more than 70% of the inference time of modern DNNs [125].
[3]Without loss of generality accurate sign calculation is assumed.

at each training step, thus making learning easier. A widely used normalization method is Batch Normalization [54], which transforms x according to the following expression:

$$y(x) = \gamma \odot \frac{x - \mu_x}{\sigma_x} + \beta, \tag{3}$$

where μ_x and σ_x are the mean and standard deviation of the input tensor x and γ, β are the scale and shift parameters, respectively. Those are learned with the rest of the model parameters during training.

2.2 Training and Inference

2.2.1 Training. During training, the network tries to learn the weight values. A labeled dataset is used for the training process. A variant of the stochastic gradient descent algorithm, which is iterative, is mainly used in training. The main processes of training are the forward and backward propagation and the weight gradient and update. In the forward pass, the neurons in each layer are evaluated by traversing all layers in succession from first to last. In backpropagation, the outputs of the network are compared with the golden outputs and the resulting error is propagated back through the network layers. The weight update is then performed by accumulating the product of the forward pass activations and the backpropagation errors corresponding to a given weight. Training is usually executed on distributed systems with many workers and can become a very time-consuming procedure. For example, Facebook required 1 hour for the 90-epoch ImageNet training with ResNet-50 using *32 CPUs and 256 NVIDIA P100 GPUs* [34].

A common problem in the training process is overfitting. Overfitting has not yet been proven mathematically but only experimentally and refers to a network trained so much that it produces overly complex and unrealistic class boundaries when data meticulously fits into the model and is memorized. This leads to poor performance when a new input was never seen before. Some techniques that help to avoid overfitting by making the model simpler are dropout [115], early stopping [110], weight decay [70], and learning with noise [83]. As discussed in Section 3, the approximation noise induced by the approximate circuits might help in mitigating overfitting.

The works that we studied in our survey have widely employed approximation-aware (re)training (details in Section 4.1) as an error compensation mechanism to mitigate the accuracy loss due to the introduced hardware approximation. However, given the increased time complexity of training (as mentioned above), retraining can be very time consuming, and in the case that approximate hardware emulation is required, the time required can become unsustainable [81]. Moreover, it is highly possible that approximation-aware (re)training can be even infeasible, due to proprietary models and/or datasets [81]. Therefore, alternative methods are also explored such as fine-tuning and other statistical approaches (details in Section 4.2). Finally, it should be mentioned that quantization-aware training has gained a lot of popularity since it enables remarkable model compression and very low-bitwidth integer-only arithmetic inference [21].

2.2.2 Inference. During inference, the already trained NN is used to derive predictions against new unseen data. Inference involves only the forward pass. Training identifies the model parameters, while inference uses the model to make predictions. In contrast to the training procedure, inference is typically executed on a single device [125] (cloud or even on a mobile/edge/IoT device) where latency requirements [58] as well as energy constraints can become very tight. The larger a DNN, the more power and energy is consumed to run inference, and the higher the latency will be. Hence, although the trained model could be directly deployed to run inference, this is rarely the case and several optimizations are examined to meet real-world requirements. To that end, hardware approximation techniques, which constitute the focus of our survey, have been widely studied to enable efficient DNN inference.

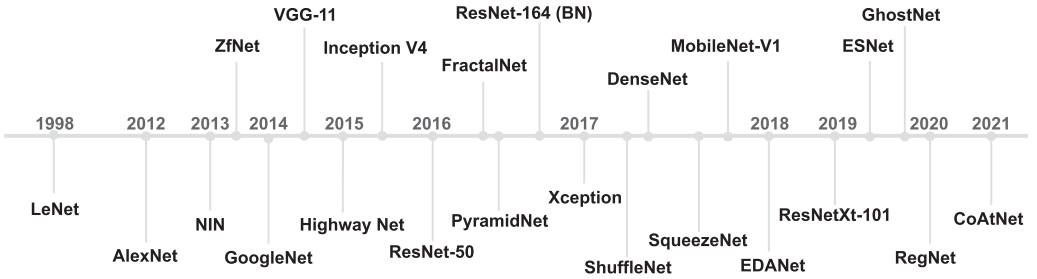

Fig. 4. Timeline of notable DNNs.

Table 2. Common Datasets Used in DNN Evaluation

Dataset	Images	Classes	Size	Input Size	Year
MNIST [69]	60K	10	50 MB	28 × 28	1998
SVHN [85]	600K	10	2.5 GB	32 × 32	2011
CIFAR [66]	60K	10/100	170 MB	32 × 32	2009
ImageNet [26]	1.5M	1,000	150 GB	256 × 256	2009

2.3 Models and Datasets

Over the decades, significant research effort has been carried out to improve the performance of DNNs, and particularly CNNs, through novel architectures. Figure 4 presents some notable CNN models published over time. CNNs have been applied to vision tasks since the 1980s when [68] proposed the first multilayer CNN named ConvNet. LeNet (1998) [69], an improved version of ConvNet, achieved significant milestones in recognition tasks. However, the neverending requirement for higher accuracy led to many new, deeper, and vastly more complex models.

An important aspect in DNNs is the complexity of the task that they have to address. Datasets are fundamental to test a DNN's accuracy. Table 2 presents the characteristics of the most commonly used datasets in the works we reviewed in Section 3. Many datasets might exist for the same task, but different datasets are hardly comparable and their difficulty can significantly vary. Different datasets reflect different models, and more complex datasets require more complex networks. The latter translates to more weights and consequently a larger number of operations (MACs).

3 HARDWARE APPROXIMATIONS FOR DNNS

In this section, the state of the art of hardware approximate computing techniques mainly for deep CNN inference is discussed. Note that although some of these techniques rely on (re)training to mitigate the accuracy loss due to approximation, training is used only as a mechanism to improve the accuracy of the approximate inference and it is not the target of the approximation itself. In addition, after identifying common patterns in examined techniques, we organize them in groups with respect to the type of applied approximation. As illustrated in Figure 5, hardware DNN approximation can be clustered into three wide categories: *Computation Reduction*, *Approximate (Arithmetic) Units*, and *Precision Scaling*. It is noteworthy that although these approximation categories are orthogonal, the state of the art applies, mainly, approximations from one category or combines Precision Scaling with approximations from another category.

3.1 Precision Scaling

Low-precision computation is the key to enable high compute densities in DNN hardware accelerators across cloud and edge platforms. One of the first and most widely used approximation

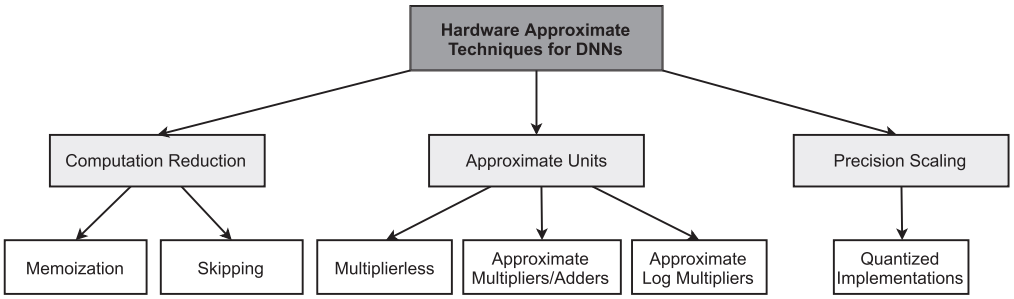

Fig. 5. Clustering of hardware DNN approximation techniques.

techniques to enable effective precision scaling is quantization. Quantized hardware implementations feature reduced bitwidth dataflow and arithmetic units (as illustrated in Figure 6), thus attaining very high energy, latency, and bandwidth gains compared to **32-bit floating-point (FP32)** implementations. Traditionally FP32 was used in DNN inference. Rather than executing all the required mathematical operations with ordinary 32-bit/16-bit floating point (as in CPUs and GPUs), quantization allows us to exploit smaller integer operations instead. Moreover, quantized implementations reduce the size of the model linearly, leading to high storage gains and low memory transfers. In integer-arithmetic-only inference, weights and activations are quantized to low-bitwidth (e.g., 8-bit) integers and biases are quantized to 32-bit or lower [55]. Other quantization approaches mainly target model compression and quantize only the weights, e.g., [137]. Advancements in quantization methods have demonstrated that **integer 8-bit (INT8)** DNN inference can achieve almost identical accuracy with FP32 [58]. Finally, a significant advantage of quantization is that although it directly impacts the hardware requirements, the accuracy loss is fully controlled and defined at the software level. In other words, the hardware gains will depend only on the supported precision(s) of the accelerator, while the accuracy will depend on the employed quantization method, though the latter assumes that the accumulators of the DNN accelerator have enough precision to avoid any overflow and accurately accumulate the partial sums [18, 40, 58, 116]. If this is not the case, then approximate results may be obtained since the intermediate partial sum might be clipped by a maximum value defined by the precision of the accumulator. However, the works that we studied in this survey do not consider such an approximation and the size of the accumulator is selected large enough to avoid any overflow, e.g., based on the largest filter size. Concluding, studying quantization methods and quantized hardware implementations is out of the scope of this work, and comprehensive discussions can be found in many works [14, 20, 21, 27, 32, 71, 98, 116, 125]. A brief discussion is included in this section for completeness reasons and since many of the approximate techniques discussed hereafter are compatible and/or orthogonal with quantized implementations. Nevertheless, quantized implementations will not be further analyzed.

On the software side, there are multiple quantization methods and several ways to map the data on the compressed precision levels. For example:

- The simplest method is mapping through static **fixed-point (fxp)** quantization. An N-bit fixed-point number is represented by $(-1)^s \times m \times 2^{-f}$, where s is the sign bit, m is the (N-1)-bit mantissa, and f is a scale factor. The energy and area of an fxp multiplier scales approximately quadratically with the number of bits. For example, an 8-bit fxp multiply consumes 15.5× less energy with 12.4× less area than a 32-bit fixed point multiply, and 18.5× less energy with 27.5× less area than a 32-bit fp multiply [51].
- In dynamic fixed-point format, numbers in a similar dynamic range are grouped together and share a common fraction length. This fraction length is chosen based upon the dynamic

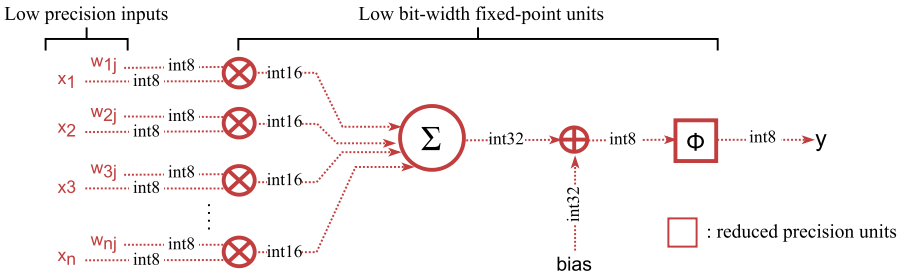

Fig. 6. Schematic of a neuron when applying Precision Scaling approximation. The precision of all the neuron's components is affected (reduced). This example is adapted from [55] and illustrates an integer-arithmetic-only 8-bit inference.

range of each of the three layers, i.e., inputs, weights, and outputs. The proper use of this method combined with the examined network's weight analysis can achieve even higher-accuracy results than the simple fxp method [95].

- Another approach is to use a simple mapping function such as a power-of-two function, where the distance between different quantization levels varies and implementations can be done with simple logic such as a shift operation [74].
- Regarding reduced-precision floating-point numbers, several formats have been explored since 2015 [39]. Some of them include IEEE FP16 1-5-10, BFloat 1-8-7 [128], and DLFloat16 1-6-9 [2] representations. **Hybrid-FP8 (HFP8)** supports two formats, FP8 1-4-3 and FP8 1-5-2 [125], while Minifloat supports any exponent and mantissa combination [40].

Hereafter, we present some state-of-the-art quantization techniques that enable the exploitation of precision-scaled hardware by mitigating the accuracy loss due to the low numerical precision.

Post-training Quantization: Uniform symmetric [65] and asymmetric min/max [55] are **post-training quantization (PTQ)** methods and achieve very high accuracy at 8 bits. Similarly, a post-training quantization method, ACIQ, is proposed in [8]. ACIQ uses an optimal clipping for quantization that limits the range of activation values in order to reduce the rounding errors while also containing most of the un-quantized information. In both activations and weights, a bit allocation is applied for each channel to minimize the **mean square error (MSE)**. A bias correction scheme is also introduced to fix the deviation that occurred due to quantization. These three methods can be combined to restore most of the accuracy loss without the need for retraining. [127] presents a PTQ procedure that supports linear quantization for activations and linear, power-of-two, and two-hot quantization for the weights. In [127] the error generated in each layer is used to adjust the quantization step size for both features and weights in an iterative way. Ristretto [40] is an approximation framework that includes dynamic fxp, minifloat, and power-of-two number formats and performs automatic network quantization by evaluating different bitwidths and number representations to find the right balance between compression rate and network accuracy.

Quantization-aware Training: **Quantization-aware training (QAT)** is an approach for training quantized networks. In QAT, the forward pass simulates the quantized inference, while backpropagation is performed as usual and weights and biases are in floating point [55]. The latter is crucial since accumulating the gradients in quantized precision can result in zero or high error gradients [32]. In [59] a quantization method was proposed in which the boundaries of quantization values are parameterized and trained. Afterward, values that are smaller than the lower bound are pruned. The quantizer attempts to optimize the trainable parameters with respect to the task loss of the entire network and can be applied in both activations and weights with extremely low bitwidth (2/3/4-bit), achieving state-of-the-art classification accuracies. An additional example

is [135], where training parameters of the batch normalization layers at high precision are included. In [135] the quantizer introduces a perturbation to the model parameters and they are jointly trained together, so that the model can converge to a point with a better loss [32]. The dynamic range and quantization levels can be parameterized in different ways and trained using iterative optimizations. A weight quantization scheme, **statistics-aware weight binning (SAWB)**, is also proposed in [20]. SAWB identifies the optimal scaling factor that minimizes the quantization error based on statistical characteristics of the weights' distribution (i.e., shape of distribution and representative values throughout the training) without the need for an exhaustive search. [20] demonstrates that very high accuracy for extremely low bitwidths (less than or equal to 2 bits for weights and activations) can be achieved. **PArameterized Clipping acTivation (PACT)** [21] uses an activation clipping parameter that is learned and optimized via backpropagation during training to find the right quantization scale. PACT demonstrates that although it focuses on activation quantization, also different weight quantization can normally be enabled, delivering accuracies similar to FP32 representation with only 4-bit quantized CNNs. Consequently, to quantize the weights, it uses DoReFa [138]. DoReFa is an aggressive and heuristic linear quantization that uses extremely low-precision weights and activations to all layers, excluding only the first and the last ones, while gradients are also quantized during the backward pass of the training procedure.

Binary/Ternary Quantization: More aggressive precision scaling can be employed to generate *binary* and *ternary* networks. Such networks achieve the lowest computational bitwidth and can lead to significant acceleration over higher precisions (e.g., binary arithmetic on NVIDIA V100 GPUs is 8× higher than INT8 [32]). However, they require customized hardware accelerators to be executed efficiently and training. Moreover, for such low precision (binary valued weights), due to the typically small derivatives, it is not effective to update the weights with gradient decent methods [71]. BinaryConnect [23] proposed for the first time to use binary weights in {−1,1} and [5] used full-precision activations and binary weights. In **Binarized Neural Networks (BNNs)** [53] and XNOR-Nets [97] both weights and activations are quantized in binary format. Such extremely low-bitwidth formats can replace the costly MAC units by simple XNOR gates followed by popcount (i.e., count the number of 1s). In [3] a CNN accelerator named XNORBIN is proposed with over 25× higher energy efficiency on competitive models such as AlexNet, while the XNOR Neural Engine [22] is a configurable hardware accelerator integrated into a microcontroller system, which can fully compute convolutional and dense layers of popular CNNs. Finally, ternary weight networks use a similar approach but the weights are in {−1,0,1}. A ternary quantization is also presented in [139]. In this approach the authors started from a model trained in full precision and then they converted the weights in 2 bits including a fine-tuning process to restore accuracy loss.

On the hardware side, low-bitwidth implementations are almost mainstream today. For example, Eyeriss [18] and DaDianNao [16] used 16-bit, while Eyeriss V2 [19], Google TPU v1 [58], and Samsung NPU [91] employ 8-bit MAC units. Moreover, many low-bitwidth transprecision architectures are proposed. Loom [111] uses bit-serial multiplicators, and both weights and activations have fully variable bitwidth, from 1 bit to 16 bits, while the matrix-matrix multiplication core BISMO [120] supports precision levels from 8 bits down to 1 bit. BitFusion [112] and BitBlade [102] also implement variable precision operations from 1 up to 16 bits for DNNs with optimized summations using a spatial approach. In [114] fundamental bit decomposition architectures (vertical and horizontal decomposition) are further approximated by constraining the maximum value of the partial sums. IBM RAPID [30] uses DLFloat16, 2-bit (INT2) and 1-bit fxp, while [1] supports DLFloat16 and HFP8 formats as well as INT4 and INT2 formats for highly scaled inference. Intel Spring Hill [129] supports FP16 as well as INT8, INT4, INT2, and even 1-bit precision operations natively. Finally, NVIDIA Tensor Cores offer a full range of precisions, i.e., TF32, Bfloat16, FP16, INT8, and INT4 [86].

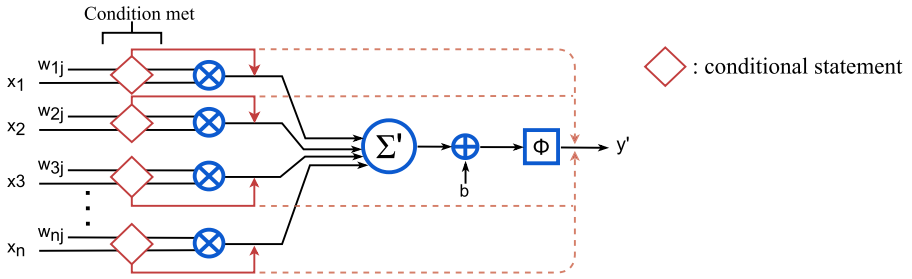

Fig. 7. Schematic of a neuron when applying computation reduction approximation. A conditional branch is employed to skip or not some of the neuron's computations.

3.2 Computation Reduction

During DNN inference, millions of multiplications are performed in the convolution operations [125], leading to high latency and energy consumption, even when considering quantized implementations. The Computation Reduction approximation category aims at systematically avoiding, at the hardware level, the execution of some computations, e.g., multiplications and convolution operations. As a result, it significantly decreases the executed workload. Computation Reduction is further subdivided into the Memoization and Skipping approximation families. Computation Reduction (illustrated in Figure 7) uses a conditional statement to avoid a computation and estimate its output (Memoization) or discard it entirely (Skipping). Among the most popular, effective, and extensively used DNN approximation techniques that reduce the number of the required computations is the software-based DNN pruning. Pruning removes connections, filters, and/or channels based on varying importance criteria and can be divided into structured (coarse-grained) and fine-grained pruning. Pruning actually generates a compressed variant of the initial network and is executed offline before inference. On the other hand, in the hardware approximation techniques that we study in this section, the approximation originates from the hardware itself since a conditional statement is integrated in the accelerator and decides at runtime if a computation will be skipped/estimated or not. Hence, although several architectures exist, e.g., with zero-skipping support, to optimally support the software-based pruning approximation [43, 90], such architectures are not inherently approximate since they will skip computations that do not need to be executed (e.g., multiplication by zero), while the examined Computation Reduction approximation techniques will skip computations that many times should be executed in order to obtain full accuracy.

3.2.1 Skipping. Skipping approximations aim at reducing the executed workload. Such approaches perform a simple computation and evaluate (predict) if a more complex one can be eliminated. Hence, this approximation family enables *dynamic approximation* at runtime. The efficiency of the Skipping approximation relies on how often a computation can be skipped, the complexity of the conditional prediction, and the complexity of the skipped operation. Piyasena et al. [94] leverage the widely used ReLu activation function to eliminate redundant computations. [94] estimates the sign of the convolution output using a low-cost prediction scheme. In this scheme, a power-of-two weight quantization is applied so that multiplications can be replaced with simple logic shifters. If the estimated sign of the approximate output is negative, the convolution operation is skipped through the clock-gated circuitry, or else the original convolution is performed. [119] proposes a similar strategy, but the sign estimation is done either after representing weights in ternary format or after using a sign function, which simplifies the computations while maintaining the prediction accuracy. Kim and Seo [61] exploit the max-pooling layers and adopt a precision-

cascading scheme to predict and calculate only the maximum value of a convolution operation. This technique, combined with a zero-skipping scheme, can efficiently avoid redundant computations without affecting neuron synapses that contribute a lot in classification accuracy. In [48], the weights of each layer of a given CNN are clustered offline in groups. K-means is used for clustering and weights within a cluster feature the highest similarity to each other while weights of different clusters exhibit the least similarity. During inference (i.e., at runtime), only some weight groups are used while the weights of the rest groups are assumed to be zero. Finally, the difference of the two output neurons with the highest values is calculated. If the difference is above a given threshold, the obtained prediction is the output of [48], or else the inference is repeated using gradually more weight groups. Finally, Huan et al. [52] introduced **Near-Zero Approximation (NZA)**. NZA exploits the fact that when the multiplication operands are very small (close to zero), the product will be almost zero. [52] counts the leading zeros of the multiplication operands and if their number is above a threshold, the product is assumed to be zero and the multiplication is skipped.

3.2.2 Memoization. The second subcategory of the Computation Reduction is Memoization. Memoization avoids a computation (e.g., multiplication or convolution) by replacing its output with the output of a previously performed similar computation. Hence, the efficiency of this approach depends on the input similarity (i.e., how often a replacement takes place) as well as the complexity of the eliminated computation. Jiao et al. [57] apply Memoization through a configurable **Bloom Filter (BF)** unit that stores the product of frequently computed patterns and avoids performing the respective multiplications. A memoization set of 3,000 images was used in [57] to identify such patterns. Mocerino et al. [76] proposed a CAM-enhanced **floating-point unit (FPU)** to implement Memoization. Pre-computed multiplication results are reused whenever a similar input pattern occurs, thus avoiding unnecessary computations of frequent operations. To increase the frequency of patterns, a clustering approach based on the Jenks Natural Breaks algorithm is applied to weights and activations. The processing unit of [76] is pipelined and consists of two CAMs (one for the weights and one for the activations) and an SRAM. If the input pattern is precomputed, the product is loaded from the memory and the multiplier is avoided by clock-gated signals. On the other hand, [101] showed that more than 60% of the inputs of the network layer exhibit negligible changes with respect to the previous execution. Based on that fact, they proposed a method to reuse some results from the previous execution, avoiding all the computations associated with those results.

3.3 Approximate Units

DNN hardware accelerators comprise thousands of MAC units [58]. This wide category improves the energy consumption and/or latency of DNN accelerators by employing approximate circuits that replace accurate MAC units (Figure 8). Approximate Units can be further divided into three approximation families: *Approximate Multipliers/Adders*, *Multiplierless*, and *Approximate Log-Multipliers*. Briefly, Approximate Multipliers/Adders modify the circuit implementation of the multiplier/adder (e.g., logic approximation), Multiplierless replaces the multiplication with a simpler operation (e.g., addition), and the Approximate Log-Multipliers family replaces the exact binary multiplier with a logarithmic multiplier that is further approximated.

3.3.1 Approximate Multipliers/Adders. Considering the vast number of MAC operations required in the inference phase, several works focus on approximating the circuit of the MAC unit itself. Exploiting a constant energy gain per MAC operation performed, very high energy gains are obtained at the inference level. Targeting approximate MAC circuits, the state of the art mainly approximates the multiplier, since it is more complex and power consuming than the adder [10, 78, 108, 134].

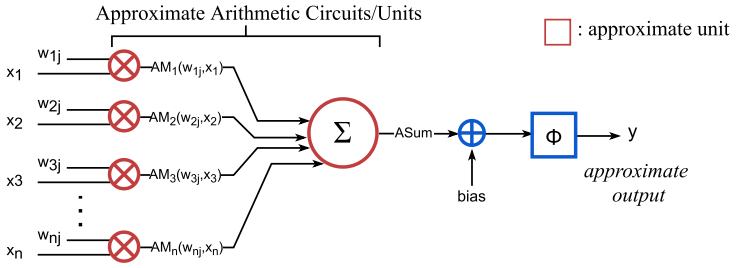

Fig. 8. Schematic of a neuron when applying Approximate Units approximation. The multiplication and/or addition units of the neuron are replaced by approximate ones.

Mrazek et al. [79] employ a **Cartesian genetic programming (CGP)**-based optimization—since it is intrinsically multi-objective and produces efficient approximate arithmetic circuits [78]—to generate approximate multipliers for inference accelerators [79]. The multipliers generated by [79] satisfy a given worst-case error constraint and ensure that multiplication by 0 is always accurate. An iterative optimization procedure is used to identify the error constraint for the generation of approximate multipliers in CGP optimization so that an inference accuracy loss threshold is satisfied. During the iterative procedure, after replacing the accurate multipliers with the approximate ones, the network is retrained to obtain the best-quality results [79]. Similarly, Vasicek et al. [121] use CGP-based optimization to generate approximate multipliers. In order to avoid time consuming CNN evaluation during the optimization phase, [121] used the **Weighted Mean Error Distance (WMED)** metric to quantify the accuracy of the approximate multipliers. To calculate WMED, the significance of each error is determined by the probability mass function of the network's weight distribution. Ansari et al. [6] evaluated 600 approximate multipliers (500 CGP-based ones and 100 variants of deliberately designed multipliers) in CNN inference showing that they can deliver significant gains in terms of power and area for a minimal accuracy loss. Moreover, [6] discussed that the induced approximation noise helps to mitigate the overfitting problem, and thus can even improve the obtained accuracy. After analyzing 600 approximate multipliers, a significant conclusion of [6] showed that when designing approximate multipliers for CNN inference, the most important error metrics are the error variance and the root mean square error. Similar to [79], [121] and [6] apply retraining to mitigate the accuracy loss due to the approximate multiplications. Nevertheless, approximation-aware retraining can be very time consuming, as discussed in Section 2.2.

Mrazek et al. [80] extended the EvoApprox8b library [78] and generated $8 \times N$-bit approximate multipliers. CGP-optimization and the quality metric of [79] are used for the generation of the approximate multipliers. [80] evaluated the generated approximate multipliers in CNN inference. Through a comprehensive analysis, [80] demonstrated that for less complex CNNs (ResNets [46] on CIFAR10), approximate multipliers may deliver considerable power savings for minimal accuracy loss (even without retraining). A similar approach that aims to eliminate multiplications by quantizing one term in power-of-two format is presented in [74]. In this method, during the forward pass weights are converted in ternary format, while in backpropagation weights and activations are quantized up to 4 bits to improve the accuracy. Nevertheless, this is not the case for more complex CNNs (ResNet-164 on CIFAR100), where even for 10% energy reduction the accuracy loss is considerable. Leveraging that weights are known after training, CAxCNN [100] uses the **Canonic Sign Digit (CSD)** representation to encode the weights. CSD uses ternary form $\{-1, 0, 1\}$, and to represent a binary number, CSD features the least number of non-zeros $\{-1, 1\}$. In addition, adjacent bits cannot be both non-zero. Exploiting these two features of CSD, [100] applied truncation

and generated approximate CSD multipliers with a very small footprint as well as low latency. Although [100] performs an optimization search to identify the optimal truncation parameter, CAxCNN does not require retraining. Exploiting that different layers feature varying resilience to approximation, ALWANN [81] applied a non-uniform approximation. ALWANN generates a heterogeneous DaDianNao architecture [16] by using heterogeneous **processing elements (PEs)**. The employed PEs are built upon different approximate multipliers from EvoApproxLib [78]. ALWANN [81] implements a layer-wise approximation in which each layer is mapped to a specific PE type. A genetic optimization procedure is used to identify the approximate multiplier per PE as well as the layer mapping to PEs. ALWANN avoids retraining and recovers some of the accuracy loss by employing a simple, approximation-aware weight-tuning procedure. Similarly, Zervakis et al. [131] also applied layer-wise approximation. [131] used wire-by-switch replacement to generate an approximate multiplier with three accuracy (relative error) modes. Hence, using this reconfigurable multiplier, [131] generated a homogeneous approximate architecture. Through an exhaustive exploration, [131] determined the accuracy mode per convolution layer and generated the respective accuracy-energy consumption Pareto front. Tasoulas et al. [118] introduced the weight-oriented approximation. [118] generated a **low-variance approximate multiplier (LVRM)** with three approximation modes (i.e., three error variance values). A greedy procedure is used in [118] to map weight ranges to the approximation modes of LVRM. The significance of each convolution layer is also used in the mapping procedure; i.e., weights of less sensitive layers are entirely mapped to the highest approximation. In addition, [118] proposed a bias correction method in order to avoid retraining and mitigate the accuracy loss due to the approximate multiplications. Hammad et al. [41] performed approximate multiplication using the **Dynamic and Static Segmented Multipliers (DSM, SSM)**, which perform the multiplication with m-bit input segments (where m is smaller than the input bitwidth). In SSM the most significant segment that contains a 1 is used (static), while in DSM the segment is dynamically selected based on a **leading one detector (LOD)**. To attain high accuracy, [41] generated a reconfigurable accelerator that comprises low-precision (low m) and high-precision (high m) approximate multipliers. A low-cost classifier is trained to predict the required precision (low or high) for each input image. At runtime, a controller decides the precision level and then inference is executed using the respective approximate multipliers. Guo et al. [38] proposed an approximate multiplier that can support one 16-by-8-bit multiplication or two 16-by-4-bit multiplications and uses an approximate adder to add/merge the outputs of the sub-multiplications. The proposed approximate adder extends the block-based adder GeAr [108]. [38] observed that in a quantized CNN, the inputs of the multipliers roughly follow a Gaussian distribution instead of a uniform distribution. Exploiting the correlation of the bits for Gaussian distributed inputs, [38] generated approximate adders with an unequally sized block structure to trade off between accuracy and circuit delays. [38] considers an Eyeriss-like architecture [18] that uses the proposed approximate reconfigurable multipliers and employs different quantization precision for different layers (i.e., 8-bit or 4-bit). Exploiting the proposed reconfigurable approximate multiplication, layers with 4-bit weights are executed at higher throughput.

Hanif et al. [44] consider a systolic MAC array architecture [58] and introduce a curable approximation technique. "Curable" approximation refers to approximation approaches that feature an internal error compensation mechanism that enables them to self-correct the induced error. This is mainly achieved by estimating the error at runtime and compensating it at a later stage. CANN [44] splits the adder of the MAC unit into two parts (low and high) and cancels the carry propagation from the low to the high part. Hence, the carry chain (and thus the delay) of the MAC unit is decreased. To cure the introduced error, the output carry of the low part is accumulated in the next cycle by the neighboring MAC unit. The errors generated by the eliminated carries of the border MAC units are not cured. Zervakis et al. [133] considered also a systolic MAC

array architecture [58] and replaced the accurate multipliers with the approximate perforated ones [134]. The perforated multipliers omit the generation of some partial products and thus the induced error is known a priori [134]. [133] introduced a control variate approximation technique to heal the approximate multiplication error at runtime. [133] leverages that the weights are known after training and that the error of the perforated multipliers can be rigorously expressed in order to formulate a control variate that efficiently estimates the runtime convolution error based on the values of the input activations. An additional column of MAC units is required to accumulate the control variate and compensate the error.

Concluding, the integration of Approximate Multipliers/Adders in neural network accelerators has attracted significant research interest over the last years. The approximation techniques that belong in this family can be further organized as follows:

- [6, 79, 81, 100, 121] generate NN-specific approximate accelerators, i.e., apply NN-specific approximations. On the other hand, [38, 41, 44, 80, 118, 131, 133] generate generic approximate accelerators.
- [6, 79, 121] apply retraining to mitigate the accuracy loss, while [38, 41, 80, 81, 100, 118, 131] do not require/apply retraining, and [44, 133] employ a runtime curable approximation technique.
- [6, 44, 79, 80, 100, 121, 133] apply static approximation, while [38, 41, 81, 118, 131] employ dynamically reconfigurable approximate architectures.

3.3.2 Multiplierless. The Multiplierless subcategory aims at maximizing the gains by eliminating the expensive multiplication circuits. To achieve this, multipliers are replaced by circuits that implement a simpler operation. Parmar and Sridharan [92] exploited the fact that scaling the input feature map does not affect the features extracted by max-pooling and reduced the span of the scaled weights to $[-1, 1]$. This condition allowed to introduce in the convolution equation trigonometric functions, which can be implemented by the low-cost CORDIC algorithm. In [29], authors proposed **reconfigurable constant coefficient multipliers (RCCMs)** that use only adders and shifters. The supported coefficients are extracted offline based on a distribution matching technique that allows specific RCCMs to be selected depending on the model's weights. Sarwar et al. [105] employed multiplierless neurons by replacing multipliers with simplified shifts and add operations controlled by a unit. The so-called **Alphabet Set Multipliers (ASMs)** compose a precomputer bank to compute lower-order multiples of the input based on some small-bit sequences termed alphabets ($\{1, 2, 3, 5, \ldots\}$), an adder, and one or more select, shift, and control logic units. The size of the alphabet defines the accuracy as well as the energy benefits of ASM. An efficient retraining is finally performed in order to tune the weights and mitigate the accuracy degradation due to ASMs.

3.3.3 Approximate Log-Multipliers. The Log-Multipliers subcategory converts multiplications into additions by taking an approximate logarithm. Mitchell [75] proposed an approximate multiplier that employs the log multiplication property. [75] proposed to compute approximate binary log and antilog by a linear approximation of the log-antilog curves between each power-of-two interval. Saadat et al. [103] extended [75] to generate a minimally biased approximate multiplier. [103] observed that in Mitchell's algorithm the error value is always negative. Through a mathematical analysis, [103] demonstrated that with the addition of a constant correction term, overall the error is reduced and the average error is pushed close to zero. In addition, [103] applied truncation to reduce the size of the main components required (i.e., adder and barrel shifters). Kim et al. [62] optimized Mitchell's logarithmic multiplier for approximate CNN inference. [62] improved Mitchell's implementation (LOD, shift, and decoder blocks); introduced a zero-checking block, which is mandatory to improve the performance of CNNs; and further approximated the

design by applying truncation and one's complement for negation. In [93], another approximate logarithmic multiplier with two stages of approximations was proposed. During the first stage, the two operands are split into two parts and the proposed multiplier selects either the upper part (if it contains at least one non-zero bit) or the lower part (if not) for the following computations. The second stage of approximation concerns the binary-to-logarithm conversion, where, in order to reduce more the complexity of circuitry, only a number of (leftmost) bits in the mantissa part are kept. Vogel et al. [126] introduced a quantization scheme to fractional powers-of-two (e.g., $2^{1/4}$) and showed that the latter provides higher resolution at higher values and the granularity of weight distribution becomes more fine-grained. Based on the proposed quantization, [126] replaced the binary MAC units with logarithmic PEs that use an adder, a lookup table (for the required exponents), and a barrel shifter before accumulating the result.

4 ERROR COMPENSATION TECHNIQUES

Although DNNs feature an inherent error resilience, naive or aggressive approximation may result in unacceptable accuracy loss. In addition, complex networks can become very sensitive to even slight approximation [80, 118, 132]. In this section, we discuss techniques employed by the state of the art to achieve high accuracy albeit with the applied approximations. Such techniques enable satisfying tight accuracy constraints and/or increasing the applied approximation to further boost the attained gains.

4.1 Accuracy Recovery with Retraining

Mrazek et al. [79] showed that when approximate multiplication is used instead of an accurate one, the classification accuracy of the examined network decreased to almost 10%. However, [79] applied approximation-aware retraining and after only five epochs, the accuracy was recovered to more than 90% for the MNIST dataset. The backpropagation algorithm was employed in [79] using the approximate multipliers in the forward pass. A similar approach is followed in [6]. Despite the high accuracy achieved, retraining can become very time consuming since (1) retraining large NNs can be very slow and (2) in the feed-forward it requires emulation of the approximate hardware. To accelerate the accuracy evaluation when using approximate multipliers, [123] proposed TFApprox, a GPU-based hardware emulation framework that extends TensorFlow and supports approximate multiplication through lookup tables.

A hindering factor to efficient retraining can be the non-uniformity of the approximate multipliers. Although a proper learning rate, i.e., a multiplication factor in the weight update equation [88], can efficiently adjust the network with approximations in place, many approximate multipliers may require careful regulation. For example, in the ASM multipliers [105] using one alphabet, the allowed weight levels are 0×, 1×, 2×, 4×, and 8×. The distance between 2× and 4× is 2×, while the distance between 4× and 8× is 4×. In this case, a low learning rate would not be enough for weights to be updated properly and overcome the distance barrier between allowed levels. This would cause weights to condense in a specific level, resulting in a high network accuracy loss. The same effect would have a high learning rate too. Hence, [105] used initially the highest learning rate that was used to train the CNN without approximation. If the accuracy improves and satisfies given constraints, retraining is carried on with the same learning rate for a few more iterations and until no significant improvement in the accuracy is observed. If the accuracy does not improve, the learning rate is reduced by a factor and the approximate CNN is further retrained. This process of regulating the learning rate is continued until the accuracy improvement saturates.

Beyond the conventional backpropagation algorithm, some works have proposed extended formulations for approximate DNN retraining. AxTrain [47] is a hardware-oriented framework for DNN training that supports approximate inference. In [47] two DNN training techniques were

introduced, referred to as *passive* and *active* methods. During retrain a stochastic error model is backpropagated to the network parameters in order to minimize the noise sensitivity and the network's accuracy. Substantially, the passive method concerns the training procedure, trying to recover accuracy loss, while the active method helps the network learn the noise distribution with minor modifications in each epoch and become more robust to hardware approximations. A novel regularization [11] method, called *alpha regularization*, to bias the training algorithm for approximate CNN was presented in ProxSim [24]. Similarly with passive training of AxTrain, ProxSim simulates approximate hardware elements during the computations of CNN forward pass and then a regularization term is added to minimize the propagated approximation error for each CNN layer. Although this method was about 2% slower than AxTrain, it appeared to be more efficient in 90% of the cases, as it delivered better improvements in accuracy. Considering implementations with large approximation errors, [67] presented a novel methodology for efficient error recovery through **Knowledge Distillation (KD)** [49] for approximate DNNs. This methodology consists of two stages in which FP information is distilled first into a quantized model and then into an approximated model. This recovery error scheme achieved a small accuracy loss (<3%), having energy savings but no improvement in retraining time. On the contrary, [25] achieved a reduction in retraining time of up to 11×, compared to a behavioral simulation of approximate multipliers in DNNs using ProxSim. In [25] an obtained error model was added to each NN layer before the activation function, targeting to an improvement in the DNN generalization. This generalization can be considered as a regularization method, which leads to better and faster results than training with the behavioral simulation.

4.2 Statistical Error Compensation

Several works investigate alternatives to retraining in order to improve the accuracy achieved. As aforementioned, retraining is time consuming and might not always be feasible (e.g., proprietary datasets). ALWANN [81] proposed a fast weight-tuning algorithm that adapts the weights according to the employed approximate multiplier and does not require any preprocessing or inference evaluation. [81] replaced the weights in each layer based on the error characteristics of the employed approximate multiplier. Each weight w was replaced by w' as follows:

$$\underset{\forall w'}{\operatorname{argmin}} \sum_{\forall \alpha} |M_{ax}(a, w') - a \cdot w|, \tag{4}$$

where M_{ax} corresponds to the approximate multiplication. Using Equation (4), [81] selected the value w' that minimizes the sum of absolute differences (error) between the output of the approximate and accurate multiplication over all inputs ($\forall \alpha$). In other words, given an approximate multiplier, [81] updated the weights so that the **Mean Error Distance (MED)** of the performed approximate multiplications is minimized.

The main computation of a convolution operation is given by

$$Y_o = \sum_{i=1}^{N} W_{o,i} X_i + b_o, \tag{5}$$

where $W_{o,i}$ are the filter's weights, X_i are the input activations, and N is the number of weights.

The error (ϵ) of an approximate multiplier can be viewed as a random variable defined by its mean value $\mathrm{E}[\epsilon]$ and its variance $\mathrm{Var}(\epsilon)$ [118]. Therefore, if the approximate multiplication error is systematic, it can be compensated by a constant correction term [118]. Given the convolution operation (Equation (5)) and following this reasoning, [118] proposed a bias-update method to encompass this correction term and compensate, thus, the error induced by the approximate multiplications. Tasoulas et al. [118] proposed to replace the bias term b_o in Equation (5) by b'_o. The

latter is given by

$$b'_o = b_o + \sum_{i=1}^{N} \mathrm{E}[\epsilon_{W_{o,i}}], \tag{6}$$

where $\mathrm{E}[\epsilon_{W_{o,i}}]$ is the mean error of the approximate multiplication $W_{o,i} \times X_i, \forall X_i$. Hence, the mean convolution error is given by [118]

$$\begin{aligned} \mathrm{E}[\epsilon_{Y_o}] &= \mathrm{E}[Y_o - Y'_o] \\ &= b_o - b'_o - \sum_{i=1}^{N} \mathrm{E}[\epsilon_{W_{o,i}}] = 0. \end{aligned} \tag{7}$$

As a result, by just updating the bias term using Equation (6), the mean convolution error is effectively nullified. However, fully exact inference cannot be achieved since the convolution error features non-zero variance. To demonstrate the impact of the bias update, [118] showed that for the same accuracy loss constraints, the bias update enables using higher approximation and more energy-efficient multipliers. For example, for 0.5% accuracy loss constraint, using the bias update [118] achieved 1.4× higher energy reduction compared to the case that does not consider the bias update.

Finally, Zervakis et al. [133] introduced a control variate approximation to improve the accuracy of the convolution operation. Instead of Equation (5), [133] proposed to compute

$$Y_o = b_o + \sum_{i=1}^{N} M_{ax}(W_{o,i}, X_i) + V_o. \tag{8}$$

Again M_{ax} corresponds to the approximate multiplication and V_o is the proposed control variate. The selection of V_o depends on the approximate multiplier. The control variate V_o estimates the convolution error at runtime and through an extra addition, V_o is added to the approximate convolution result to mitigate the error. It is mandatory that V_o can be easily computed in order not to annihilate the gains of the employed approximation (M_{ax}). In [133], V_o is calculated as a function of the input activations and the average value of the weights. Specifically, in [133], V_o is given by

$$V_o = \overline{W} \sum_{i=1}^{N} (X_i \bmod 2^m), \quad \overline{W} = \frac{1}{N} \sum_{i=1}^{N} W_{o,i}, \tag{9}$$

where m is a configuration parameter of the perforated approximate multiplier [134] used in [133]. Higher m refers to higher approximation and higher energy gains. [133] demonstrated that the proposed control variate approximation technique nullifies the mean convolution error and minimizes its variance. Over VGG-13/16, ResNet-44/56, ShuffleNet, and GoogleNet trained on CIFAR10, [133] improved the inference accuracy, on average, from 0.86% (when $m = 1$) up to 21% (when $m = 3$) compared to the same approximation without the control variate (i.e., using [134] in Equation (8) without V_o). For the same CNNs on CIFAR100, the respective improvement is from 3.6% to 21%.

4.3 Error Metric Optimization

Many hardware approximation algorithms and frameworks are usually guided by the **mean relative error distance (MRED)** metric [122, 131]. Nevertheless, MRED might not be an optimal metric for approximate DNN inference accelerators. To increase the achieved accuracy, several works optimize the generated approximate multipliers targeting specific error metrics. The generated approximate multipliers are most suitable for DNNs and can achieve higher accuracy when

combined with the previously analyzed compensation techniques or even when applied in isolation. In [79] the authors design approximate multipliers that satisfy

$$\begin{aligned} &|M_{ax}(w,a) - w \cdot a| \leq c \; \forall a, \; \forall w \text{ and} \\ &M_{ax}(0,a) = M_{ax}(w,0) = 0 \; \forall a, \; \forall w. \end{aligned} \tag{10}$$

Again M_{ax} corresponds to the approximate multiplication and c is an error threshold. In other words, [79] ensures that the worst-case error of the approximate multiplier is below a given threshold and that multiplication by 0 is always accurate. Then, [79] performs an exploration to find the maximum value of c that a given DNN can tolerate. Mrazek et al. [79] concluded that although the impact of approximate multipliers on the accuracy is DNN specific, aiming for high accuracy it is mandatory to have the accurate multiplication by 0. The same error metric is used in [80]. Vasicek et al. [121] used the **weighted mean error distance (WMED)**, which considers the input data distribution:

$$\frac{1}{|\{a|\forall a\}||\{w|\forall w\}|} \sum_{\forall a} \sum_{\forall w} \mathrm{D}(w)|M_{ax}(w,a) - w \cdot a| \leq c, \tag{11}$$

where D is the probability mass function and c is an error threshold. Using Equation (11) and exploiting that the weights are known after training (and thus D), [121] assigns higher significance to the weights that appear more often (i.e., higher $\mathrm{D}(w)$). Hence, [121] tries to ensure that the more often multiplications are performed more accurately, the higher the inference accuracy overall. Ansari et al. [6] evaluated several error metrics in order to identify critical features that render an approximate multiplier suitable for DNN inference. Specifically, the **error rate (ER)**, the **error distance (ED)**, the **absolute ED (AED),** and the **relative ED (RED)** metrics were examined. Using these error metrics, nine relevant error features of the approximate multipliers were evaluated. These features are reported in Table 3. Through extensive experimentation, [6] concluded that the most important features that make an approximate multiplier superior to others are Var(ED) and RMS(ED). Tasoulas et al. [118] reached the same conclusion. Through a rigorous mathematical analysis, [118] demonstrated that the mean convolution error can be canceled using a constant correction term as Equations (6) and (7) show. Thus, [118] deduced that the ED variance (Var(ED)) is a more important error feature when designing approximate multipliers for DNN inference. [118] showed that the mean and variance values of ϵ_{Y_o}, i.e., of the convolution error (ED), are given by

$$\begin{aligned} &\mathrm{E}[\epsilon_{Y_o}] = 0 \text{ and} \\ &\mathrm{Var}(\epsilon_{Y_o}) = \sum_{i=1}^{N} \mathrm{Var}(\epsilon) = N\mathrm{Var}(\epsilon), \end{aligned} \tag{12}$$

where $Var(\epsilon)$ is the error (ED) variance of the approximate multiplier and N is the number of the filter's weights. Note that the variance after the constant compensation (i.e., Equation (6)) equals the mean squared error (i.e., square of RMS(ED)). Hence, the mathematical analysis of [118] is validated by the experimental findings of [6] and vice versa.

5 ENERGY-ACCURACY EVALUATION

In this section we evaluate the efficiency, in terms of potential energy reduction and accuracy loss, of hardware approximation when targeting CNN accelerators. As we will observe in the remainder of this analysis, both the accuracy loss and the energy savings depend on the complexity of the evaluated use cases/benchmarks (i.e., neural network, dataset, and/or precision). Hence, we first provide a comprehensive discussion regarding the complexity of the considered use cases and then we evaluate the accuracy-energy results as obtained from the respective publications of Section 3.

Table 3. Error Features Evaluated in [6]

Feature	Description
ER	Error rate
Var(ED)	Variance of ED
E[ED]	Mean value of ED
RMS(ED)	Root mean square of ED values
Var(RED)	Variance of RED
E[RED]	Mean value of RED
RMS(RED)	Root mean square of RED values
Var(AED)	Variance of AED
E[AED]	Mean value of AED

5.1 Assessing the Complexity of the Evaluation Scenarios

To assess the efficiency of approximate DNN accelerators, it is mandatory to analyze the datasets that are used to evaluate the accuracy loss due to the introduced approximation. In other words, to evaluate how efficient an approximation is, we need to determine the complexity of the benchmark that was used to measure the attained accuracy after the approximation. For example, although the MNIST dataset is widely used in early ML research, it is a fairly simple dataset and it is fairly easy to achieve high accuracy even with high approximation [79]. As a result, MNIST cannot be considered as a representative example of the complex services that modern DNN-based systems deliver today. Hence, impressing results observed for the MNIST dataset are hardly expected to be achieved in more complex datasets.

Among all the works analyzed in our survey (techniques discussed in Sections 3.2 and 3.3), various datasets have been used to evaluate the accuracy of the proposed DNN approximation techniques. In Figure 9, we present the respective dataset allocation. As shown, the most popular datasets are MNIST, SVHN, CIFAR10, and ImageNet. Overall, there is a quite balanced research effort distributed among the simple MNIST dataset and the more complex CIFAR10 and ImageNet. It is noteworthy that the Computation Reduction approximation category mainly targets MNIST, while the Approximate Units category focuses on CIFAR10 and ImageNet.

Despite the fact that several works targeted the fairly simple MNIST dataset, Figure 10 demonstrates that DNN hardware approximation follows the current research trend. The evolution of DNNs has led researchers to target more complex use cases. Over time, as shown in Figure 10, more and more DNN hardware approximation works focus on the ImageNet dataset, while fewer and fewer works target MNIST. For example, the percentage of approximate DNN techniques that used ImageNet increased from 24% in 2016/2017 to 42% in 2020/2021. In the same period the percentage of works that use MNIST decreased from 38% to 14%.

Moreover, in addition to the considered dataset, it is also essential to examine the complexity of the DNNs that are used to evaluate the accuracy of the DNN approximation techniques. In Figure 11, we present the most widely used DNNs evaluated in our survey. As shown, 30% of the accuracy evaluations are performed on the fairly simple LeNet network. Nonetheless, significant research also targets complex networks such as the VGG (19%) and ResNet (27%) networks. Figure 12 presents a more descriptive view of Figure 11. In Figure 12, we analyze the size (in terms of numbers of required MAC operations) of the DNNs used in the accuracy evaluation of the state-of-the-art approximation techniques. Note that the numbers of MAC operations depends on both the number of the DNN parameters and the input size (i.e., dataset used). As shown in Figure 12, many approximate works (i.e., the 44%) examine DNNs that require only a few tens of millions (<100M) MAC operations. Nevertheless, significant research (>24%) is also performed on larger DNNs that require billions (>2G) of MAC operations.

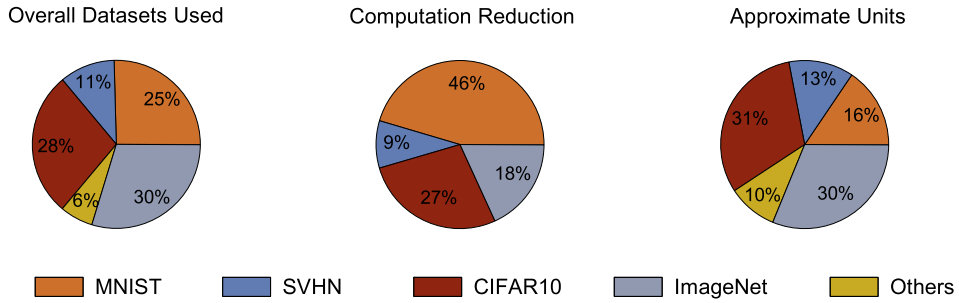

Fig. 9. The allocation of the datasets that are used in the evaluation of hardware DNN approximation. The pies present the percentage of the accuracy/energy evaluations that examine the respective dataset. The middle and right pies refer to the Computation Reduction and Approximate Units categories, while the left pie presents the aggregated results among all the approximate works. This figure evaluates the complexity of the performed evaluations with respect to the dataset difficulty. To generate this figure, all the works described in Section 3.2 and 3.3 have been considered.

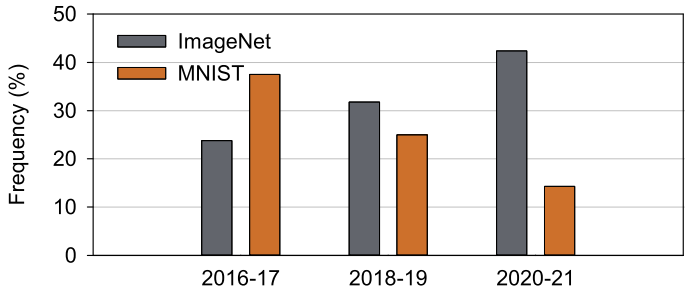

Fig. 10. Evolution of the number of approximate works (in percentage) that target MNIST and ImageNet. All the works of Sections 3.2 and 3.3 are considered to generate this figure.

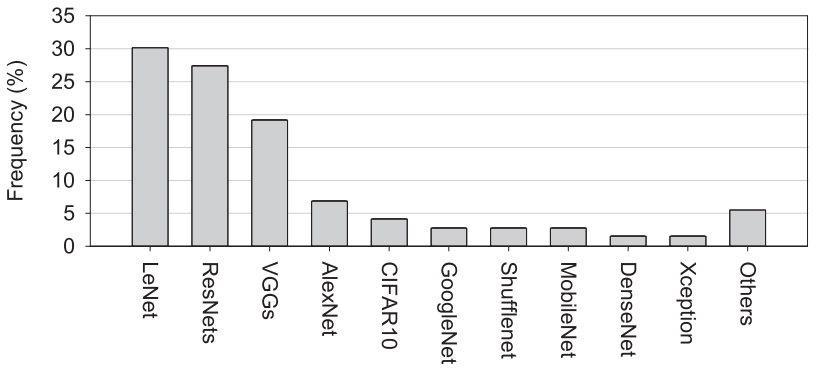

Fig. 11. The DNNs that are examined in the accuracy/energy evaluations of the approximate works. Each bar represents the number of works (in percentage) that used the respective DNN.

It is noteworthy that although the main objective of hardware approximation is energy efficiency (that is crucial especially for embedded devices), only a small portion of DNN approximation techniques targeted mobile-oriented DNNs such as MobileNet and Squeezenet (included in others). Although this might be explained by the fact that such networks are already

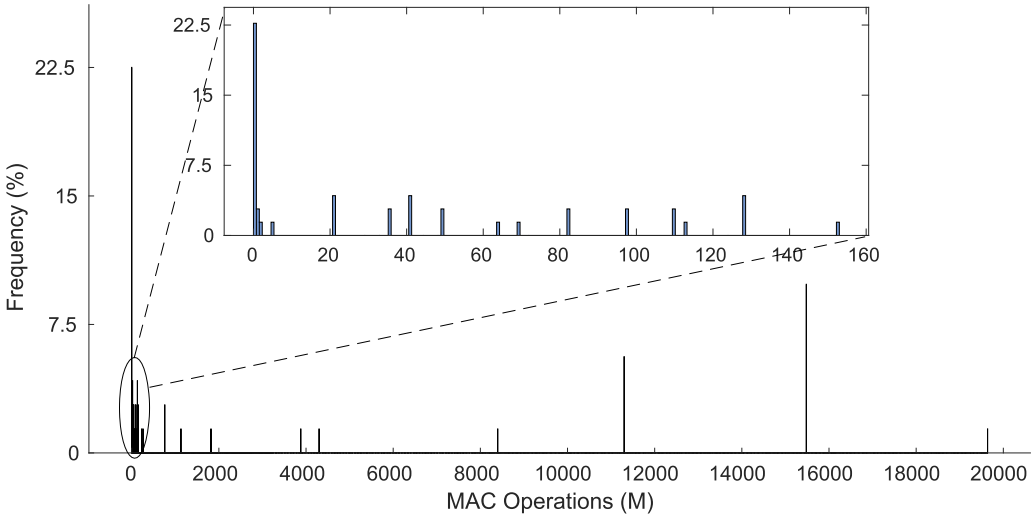

Fig. 12. The number of Mac Operations required by the DNNs that are used in the accuracy/energy evaluations of the approximate works. Each bar represents the number of works (in percentage) that considered a DNN of the respective size. This figure evaluates the complexity of the performed evaluations w.r.t. the DNN size.

very compressed, thus becoming very sensitive to further approximation, a wider and more comprehensive evaluation (i.e., more approximation techniques must be evaluated on such challenging architectures [55]) is required to draw such conclusions.

Finally, the complexity of the evaluation is highly correlated to the precision that is used to represent the weights and activations. Low-precision representations require low-bitwidth arithmetic and thus smaller circuits (e.g., multipliers and adders). As a result, they constitute more challenging use cases to apply approximate computing since they exhibit limited space for additional approximation. In other words, compared to 8-bit implementations, it is easier to achieve high energy savings combined with small accuracy loss when 32 bits are used for weights and activations. However, note that 8-bit precision is mainly used today in the state-of-the-art exact DNN accelerators [58].

Figure 13 presents the weight precision that is used in the evaluation of the approximate DNN techniques. Similar results are obtained for the precision of the activations. As shown in Figure 13, a considerable amount of the approximation techniques (>30%) use high precision (i.e., ≥16b). Nevertheless, the majority of the works (>40%) examine the conventional 8b precision. It is noteworthy that many works (>18%) examined also more challenging cases in which very low precision is used for the weights (≤6b).

5.2 Performance Analysis

In this subsection we evaluate the energy reduction and accuracy loss achieved by the CNN hardware approximation techniques that are analyzed in Sections 3.2 and 3.3. Tables 4 to 6 present the corresponding results. For each technique, the columns Neural Network and Dataset present the neural network model and dataset that was used in the respective evaluation. The columns #Conv Layers and #MAC Ops report the number of convolutional layers and the number of MAC operations required for the corresponding neural network. The required MAC operations of [41, 80, 81, 118] are calculated using the data reported in the corresponding paper, while for

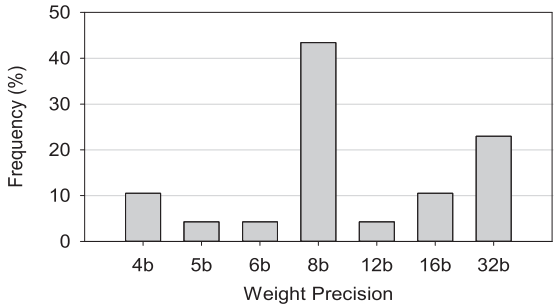

Fig. 13. The weight precision that is used in the accuracy/energy evaluations of the approximate works. Each bar represents the number of works (in percentage) that used the respective precision. This figure evaluates the complexity of the performed evaluations with respect to the inference precision.

the rest of the works we used pytorchcv [106]. Some works don't provide adequate information to calculate the MAC operations of their employed networks. The column Precision Baseline refers to the precision that the exact (baseline) design uses for the weights and activations. An x-bit baseline uses x bits to represent the weights and activations as well as an x-bit exact multiplier to perform the multiplications. The column Precision Approximate refers to the precision that the approximate implementation uses for the weights and activations. The column Acc Loss/Energy Sav. presents the energy reduction and accuracy loss of the approximate implementation with respect to the corresponding baseline.

Table 4 presents the evaluation of the Computation Reduction category and its Memoization and Skipping subcategories. As shown in Table 4, the Memoization approximation family achieves very high energy reduction (up to 77%) for a minimal accuracy loss (0.53%) on average. Nevertheless, only 32-bit precision is considered in this family and the examined networks are shallow (up to eight convolutional layers). On the other hand, the Skipping family examines deeper networks, e.g., VGG-16 and MobileNet with 13 and 35 convolution layers, respectively. In addition, the DNNs examined from the Skipping family feature many MAC operations (4.8G on average). With respect to the employed precision, the Skipping approximation techniques mainly use high precision (i.e., 16-bit and 32-bit precision). However, [61] and [94] use lower precision, i.e., 12-bit and 8-bit, respectively. Again, when considering simpler evaluation cases (e.g., LeNet, MNIST, and/or 32-bit precision), Skipping approximation delivers minimal accuracy loss and very high energy gains. On the contrary, this is not the case for more complex evaluations. When considering 8-bit precision, the energy gains drop to around 10%. Similarly, when ImageNet is considered (even with 32-bit precision), the energy savings drop significantly and the accuracy loss increases. For example, [48] achieved 24% energy reduction and 1.43% accuracy loss on VGG-16, while the respective values are 15% and 2.7% when considering MobileNet. Notably, with 12-bit precision, [61] features 1.7× better energy efficiency (TOPS/W) and only 0.8% accuracy loss for VGG-16 on the ImageNet dataset.

Tables 5 and 6 present the respective analysis for the Approximate Units category. The results for the Multiplierless and Log-Multipliers families are reported in Table 5. As shown in Table 5, the Multipliers approximation targets low-precision inference. This family exhibits a broad evaluation that covers a wide spectrum from simple use cases to very complex ones (such as ResNet-164 on CIFAR100 and ResNet-50 on ImageNet). Remarkably, with 8-bit precision, [105] achieved 53% energy reduction (compared to the 12-bit baseline) and only 0.37% accuracy loss for the very challenging ResNet-164 on CIFAR100. Note that although [105] used only 4-bit for simpler benchmarks (e.g., MNIST, SVHN), it required 8-bit precision for more complex evaluations (e.g., CIFAR10 and

Table 4. Evaluation of Computation Reduction Approximation Category

Ref	Neural Network	Dataset	Precision Approximate	Precision Baseline	#Conv Layers	#MAC Ops	Acc. Loss (%) / Energy Sav. (%)
			Memoization				
[76]	LeNet	MNIST	32b	32b	2	0.34M	0.50/69
	LeNet	GTRSB	32b	32b	2	0.34M	0.10/22
	Custom	GSC	32b	32b	-	-	2.00/57
[57]	LeNet	MNIST	32b	32b	2	0.34M	0.50/45
[101]	Kaldi	Librispeech	32b	32b	0	4.64M	0.47/49
	C3D	UCF101	32b	32b	8	0.11G	1.38/77
	Autopilot	Videos	32b	32b	5	2.22M	0.06/76
			Skipping				
[61]	VGG-16	ImageNet	12b	12b	13	15.48G	0.80/-[1]
[48]	VGG-16	ImageNet	32b	32b	13	15.48G	1.43/24
	MobileNetV2	ImageNet	32b	32b	35	0.33G	2.70/15
	LeNet300-100	MNIST	32b	32b	0	0.27M	0.12/89
	LeNet	MNIST	32b	32b	2	0.34M	1.29/51
	CIFAR10	CIFAR10	32b	32b	-	-	2.00/68
[94]	VGG-16	ImageNet	8b	8b	13	15.48G	0.21/10
	AlexNet	ImageNet	8b	8b	5	1.13G	0.27/10
	CIFAR10-Quick	CIFAR10	8b	8b	3	0.33M	0.45/12
	LeNet	MNIST	8b	8b	2	0.34M	−0.02/10
[119]	Custom	MNIST	32b	32b	2	-	−0.20/14
	Custom	CIFAR10	32b	32b	2	-	1.10/14
[52]	Custom	MNIST	16b	16b	-	0.08M	<1.00/74

[1][61] reports 1.7× higher energy efficiency (TOPS/W).

CIFAR100). As Table 5 also illustrates, the Approximate Log-Multipliers family focuses on more complex datesets such as CIFAR10 and ImageNet and examines varying precision values (4-bit to 32-bit). As in the previous techniques, when 32-bit is used, very high energy reduction is achieved (more than 70%) with a negligible accuracy loss (less than 0.5%). On the other hand, when the employed precision decreases, the energy savings drop significantly (down to 22% for 4 bits) and the accuracy loss increases (to 4.32% for 4 bits). Still, the obtained energy savings are considerable.

Table 6 reports the respective results for the Approximate Multipliers/Adders family (subcategory of Approximate Units). This is the largest approximation family, comprising the most works. The latter can be explained by the vast research activities that focused on approximate multiplication and addition circuits since the introduction of approximate computing [56]. As shown in Table 6, the approximation techniques of this family mainly target low-precision implementations (i.e., 8-bit mostly) and examine a variety of datasets and DNNs (from simple to more complex ones). As in the previous approximation categories, when considering less complex benchmarks, very high energy savings are achieved, combined with minimal accuracy loss. For example, for LeNet-5 on MNIST, [79] delivered 77% energy reduction and only 0.09% accuracy loss. Similarly, using 16-bit precision, [41] attained 81% energy savings and 0.47% accuracy loss for VGG-19 on ImageNet. In more complex evaluations, the obtained energy gains are still significant albeit being decreased. Remarkably, using 8-bit the dynamic weight-oriented approximation of [118] achieved 19% energy reduction and 2% accuracy loss for MobileNet on CIFAR100, while the curable control variate approximation of [133] delivered 35% energy savings and only 0.03% accuracy loss for VGG-16 on CIFAR100. Compared to the 8-bit baseline, [80] used an approximate 8×4 multiplier

Table 5. Evaluation of Approximate Units Approximation Category (Multiplierless and Log-Multipliers)

Ref	Neural Network	Dataset	Precision Approx. (W/A)	Precision Baseline	#Conv Layers	#MAC Ops	Acc. Loss (%) / Energy Sav. (%)
Multiplierless							
[105]*	MLP	MNIST	4b	12b	0	0.08M	0.35/61
	MLP	TiCH	4b	12b	0	0.42M	1.71/79
	MLP	SVHN	4b	12b	0	1.05M	1.68/79
	NIN	CIFAR10	8b	12b	2	0.97M	0.00/53
	ResNet-164(BN)	CIFAR100	8b	12b	163	0.26G	0.37/53
[29]*	AlexNet	ImageNet	4b/8b	8b	5	1.13G	−0.30/25
	ResNet-18	ImageNet	4b/8b	8b	17	1.82G	0.90/-[1]
	ResNet-50	ImageNet	4b/8b	8b	49	3.88G	0.40/-[1]
[92]	VGG-16	ImageNet	8b	8b	13	15.48G	0.10/55[2]
Approximate Log-Multipliers							
[93]*	ResNet-20	CIFAR10	8b	32b	19	41.29M	3.14/53
	ResNet-20	CIFAR10	16b	32b	19	41.29M	0.50/76
[62]	Cuda-convnet	CIFAR10	32b	32b	3	0.33M	0.00/74
	AlexNet	ImageNet	32b	32b	5	1.13G	0.30/74
[126]	VGG-16	ImageNet	5b	8b	13	15.48G	2.72/22[3]
	AllCNN	ImageNet	4b	8b	9	-	4.32/22[3]
[103]	AlexNet	ImageNet	32b	32b	5	1.13G	−1.00/73[3]

[1][29] reports only area reduction (up to 55% LUTs reduction).
[2][92] is compared only to recently proposed architectures and reports about 55% power savings.
[3]The same operating frequency is assumed for the approximate and baseline designs.
*Retraining/fine-tuning is used (see Section 4.1).

(8 bits for the activations and 4 bits for the weights) and achieved 70% energy reduction and 1.21% accuracy loss for ResNet-26 on CIFAR10. Nevertheless, compared to the accurate 8×4 multiplier, these values translate to 15% energy reduction and 0.08% accuracy loss.

Finally, the above analysis (Tables 4 to 6) is summarized in Figure 14. To generate Figure 14, we identified the most widely used datasets as well as the most common network sizes (≤100M or >100M MAC operations[4]) and we set two precision levels, i.e., low precision (≤8 bits) and high precision (>8 bits). Then, we created a decision tree that helps the reader to identify the optimal approximation technique/family with respect to the complexity of the evaluation (i.e., dataset, size of the network, and precision) and the desired accuracy loss constraint. For the accuracy loss we considered two thresholds, i.e., small accuracy loss (less than 1%) and moderate accuracy loss (less than 5%). The leafs represent the two approximation techniques (highest and runner-up) that achieve the highest energy reduction in each case. In addition, the respective energy reduction is also reported below the corresponding technique. The techniques are color-coded with respect to the approximation family that they belong to, and underlined techniques require DNN retraining to achieve the respective accuracy loss threshold. Note that some tree branches are empty since the respective cases haven't been considered in the evaluation of the examined approximate DNN works. For example, considering the MNIST/SVHN datasets, only small DNNs are evaluated since they can achieve almost perfect accuracy. As shown in Figure 14, when high precision is used (mainly 32-bit), the energy savings are maintained and remain very high (more than 74%) in all cases. However, when low precision is examined (mainly 8-bit), the energy savings mainly drop as the complexity of the evaluation increases.

[4]As an example, ResNet-44 on CIFAR10 requires 97.8M MAC operations.

Table 6. Evaluation of Approximate Units Approximation Category (Approximate Multipliers/Adders)

Ref	Neural Network	Dataset	Precision Approx. (W/A)	Precision Baseline	#Conv Layers	#MAC Ops	Acc. Loss (%) / Energy Sav. (%)
Approximate Multipliers/Adders							
[79]*	LeNet	MNIST	8b	8b	2	0.34M	0.09/77
	LeNet	SVHN	8b	8b	2	0.34M	−0.07/57
	LeNet	MNIST	12b	12b	2	0.34M	−0.02/66
	LeNet	SVHN	12b	12b	2	0.34M	−0.01/66
[6]*	MLP	MNIST	8b	8b	0	0.24M	−0.01/71
	LeNet	SVHN	8b	8b	2	0.34M	−0.07/71
[80]*	ResNet-8	CIFAR10	6b/8b	8b	7	21.10M	0.32/37
	ResNet-14	CIFAR10	6b/8b	8b	13	35.30M	0.18/37
	ResNet-20	CIFAR10	5b/8b	8b	18	49.50M	0.34/56
	ResNet-26	CIFAR10	4b/8b	8b	25	63.60M	1.21/70
[121]	MLP	MNIST	8b	8b	-	0.24M	−0.20/60
	LeNet	SVHN	8b	8b	2	0.34M	−0.12/70
[44]	LeNet	CIFAR10	8b	8b	2	0.34M	−1.08/46
[81]†	ResNet-8	CIFAR10	8b	8b	7	21.10M	0.10/16
	ResNet-14	CIFAR10	8b	8b	13	35.30M	−0.32/20
	ResNet-50	CIFAR10	8b	8b	49	0.11G	−0.15/17
[41]	VGG-19	ImageNet	16b	16b	16	19.64G	0.47/81
	Xception	ImageNet	16b	16b	36	8.40G	0.90/81
	DenseNet201	ImageNet	16b	16b	200	3.35G	1.14/85
[100]	LeNet	MNIST	8b	8b	2	0.34M	0.03/-[1]
	CIFAR10	CIFAR10	8b	8b	-	-	0.62/-[1]
	AlexNet	ImageNet	8b	8b	5	1.13G	0.02/-[1]
	VGG-16	ImageNet	8b	8b	13	15.48G	4.80/-[1]
[38]	VGG-16	ImageNet	8b/16b	16b	13	15.48G	3.00/37
[133]†	GoogleNet	CIFAR10	8b	8b	22	0.76G	−0.16/23
	GoogleNet	CIFAR100	8b	8b	22	0.76G	0.05/23
	ResNet-44	CIFAR10	8b	8b	43	97.80M	0.03/23
	ResNet-44	CIFAR100	8b	8b	43	97.80M	0.77/23
	shufflenet	CIFAR10	8b	8b	3	80.85M	−0.48/35
	shufflenet	CIFAR100	8b	8b	3	80.85M	0.20/23
	VGG-13	CIFAR10	8b	8b	10	0.23G	−0.30/35
	VGG-13	CIFAR100	8b	8b	10	0.23G	0.89/23
	VGG-16	CIFAR10	8b	8b	13	0.15G	0.38/35
	VGG-16	CIFAR100	8b	8b	13	0.15G	0.03/35
	ResNet-56	CIFAR10	8b	8b	55	0.13G	0.49/23
	ResNet-56	CIFAR100	8b	8b	55	0.13G	−0.34/23
[118]†	ResNet-20	LISA	8b	8b	21	40.80M	0.50/20
	ResNet-32	GTSRB	8b	8b	33	69.10M	0.50/15
	ResNet-44	LISA	8b	8b	45	97.40M	0.50/20
	ResNet-56	LISA	8b	8b	57	0.13G	0.50/22
	MobileNet-V2	CIFAR100	8b	8b	35	82.10M	2.00/19
	VGG-11	CIFAR10	8b	8b	8	153M	1.00/19
	VGG-13	CIFAR100	8b	8b	10	0.23G	0.50/19
[131]	ResNet-8	CIFAR10	8b	8b	7	21.10M	0.50/19

[1][100] reports only area benefits in terms of BEL usage from 45% up to 97%.
*Retraining/fine-tuning is used (see Section 4.1).
†Statistical error compensation technique is used (see Section 4.2).

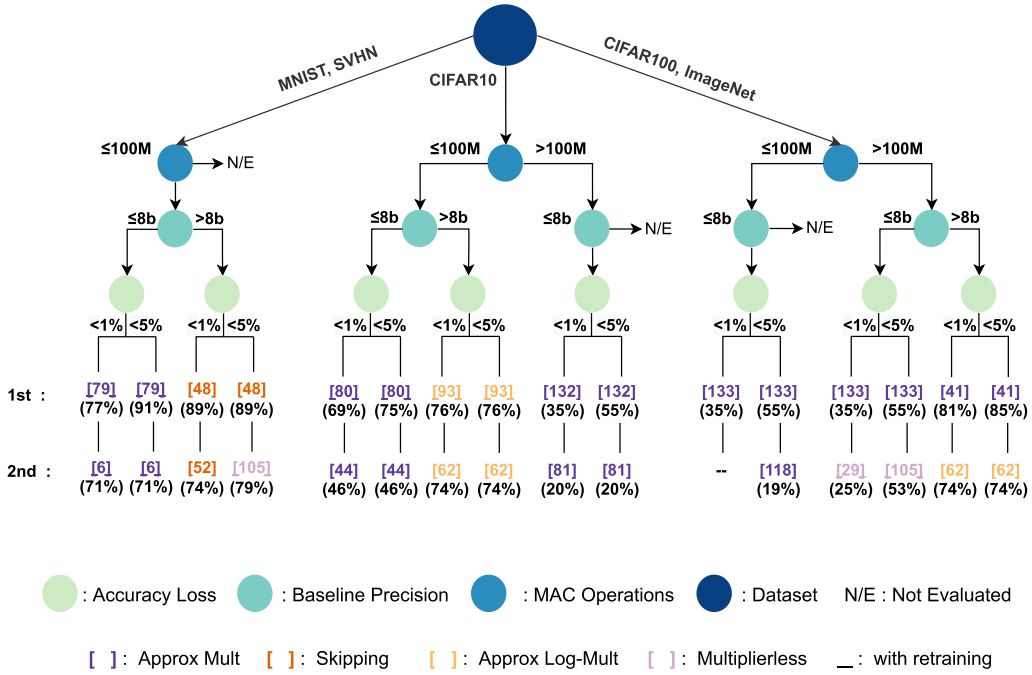

Fig. 14. Classification of the optimal hardware approximation techniques with respect to the complexity of the performed evaluation, i.e., dataset, DNN size, precision, and an accuracy loss threshold (1% and 5%). The leafs present the corresponding optimal approximate technique (i.e., highest energy reduction) as well as the respective runner-up technique. Below each technique the attained energy reduction is reported. The approximate color-coded with respect to the approximation family (see Figure 5) that they belong. Underlined techniques require retraining.

Overall, the Approximate Multipliers/Adders family dominates Figure 14, but this might also be subject to the fact that the Approximate Multipliers/Adders is the largest examined approximation family. Nonetheless, we can observe a considerable diversity in the tree's leafs with respect to the approximation family. Specifically for the Approximate Units category (Section 3.3), as shown in Figure 14, techniques from all three families (i.e., Multiplierless, Approximate Log-Multipliers, Approximate Multipliers/Adders) appear among the optimal solutions. When high precision is required, the Approximate Log-Multipliers constitute mainly the best solutions. On the other hand, when considering the most challenging evaluation (i.e., complex dataset and low bitwidth), the Approximate Multipliers/Adders and Multiplierless families prevail. It is noteworthy that Multiplierless is represented by two different works ([105] and [29]) that both require retraining, however. On the other hand, the Approximate Multipliers/Adders is represented by one work (i.e., [133]) that employs a statistical error compensation method (see Section 4.2). It is noteworthy that although the impact of the error compensation techniques is not always comprehensively analyzed in the respective works, from Figure 14 we can deduce that such techniques are mandatory to achieve high energy savings combined with low accuracy loss.

Moreover, we observe, in Figure 14, that the approximation techniques that require or do not require retraining are quite balanced. Out of all the techniques reported in the leafs of Figure 14 [6, 29, 79, 80, 93, 105] require retraining, while [41, 44, 48, 52, 62, 81, 118, 133] do not. Hence, although significant research focused on approximation-aware retraining, in the challenging evaluations (e.g., ImageNet in Figure 14), the optimal techniques do not apply retraining. The

latter could be explained by the fact that retraining for ImageNet is very time consuming. This further highlights the need for more efficient approximation-aware retraining or for techniques that apply curable approximations without retraining. Finally, hardware approximation for DNNs can deliver significant energy savings even when considering complex scenarios and low-accuracy loss constraints. However, by highlighting the big difference in energy savings between the "easy" and "complex" evaluation scenarios, it is derived that significant research is still required in more challenging benchmarks.

6 NOT JUST ENERGY EFFICIENCY

In the previous sections, we analyzed the impact of approximate computing on the energy efficiency and accuracy of DNN accelerators, demonstrating that significant energy gains are achieved for a minimal accuracy loss. In this section, our analysis goes beyond the energy efficiency of DNNs. Recent research has shown that approximate computing principles can also enable designers to overcome degradation effects (reliability-aware approximation) and security weaknesses (defensive approximation), as well.

6.1 Reliability-Aware Approximation

In contrast to traditional thermal management approaches [87], Amrouch et al. [4] employed approximate computing as a solution. Through precise chip modeling and multi-physics simulations using Ansys, [4] demonstrated that DNN accelerators are subject to excessive power density and elevated on-chip temperatures. [4] proposed the first hybrid thermal management for DNN accelerators that employs runtime approximation as a cooling mechanism. Dynamic precision scaling through clock gating and low-bitwidth quantization is applied in [4]. As a result, at the cost of some accuracy loss, reduced switching activity and thus lower power and power density are achieved. Hence, as [4] demonstrated, for the same cooling cost, precision scaling can reduce the power and thus the temperature. Similarly, the power gain of precision scaling can be traded to increase both the frequency and the cooling cost and achieve higher performance for the same temperature and total power consumption. It is noteworthy that for the 85℃ temperature constraint, precision scaling boosted the efficiency (throughput/Joule) of the DNN accelerator by 1.5× [4].

The very high utilization of the DNN accelerator's MAC units exposes the underlying transistors to continuous stress with very little time for relaxation and recovery [104]. As a result, transistors age faster. The presence of excessive temperatures [4] exacerbates further the problem as the majority of mechanisms behind transistor aging exponentially depend on the operating temperature [104]. Salamin et al. [104] proposed a circuit-aging-aware approximation framework that applies a graceful-approximation technique to suppress, over time, the aging effects in DNN accelerators. Through aging-aware cell libraries, [104] analyzed the delay of MAC units at varying aging levels. Exploiting that lower-bitwidth inputs activate shorter paths and thus the circuit can operate faster, [104] performed static time analysis to identify the maximum precision for each MAC input so that no aging-induced timing errors occur. The obtained precision was used to quantize the weights and activations of the NN at runtime. A library of quantization methods [8, 55, 65] was used to achieve the highest accuracy. [104] eliminated the aging-induced timing guardbands, boosting the throughput by 23%, and delivered a progressive accuracy degradation over time. At 10 years' aging, the accuracy loss was only 2.96% on average [104].

6.2 Defensive Approximations

Recent research showed that DNNs have intrinsic security weaknesses and are susceptible to adversarial attacks [82, 109]. Approximate computing has emerged as a means for making DNN models more robust against such attacks, while maintaining its effective tradeoffs, i.e., high energy

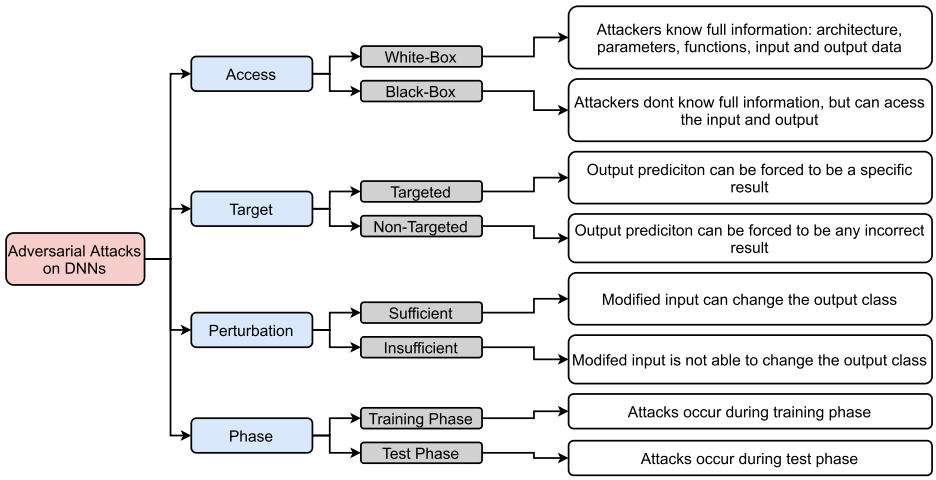

Fig. 15. Classification of DNN adversarial attacks.

savings for a small accuracy loss. The scope of an adversarial attack, in the case of vision tasks, is to introduce a noise in the input image to create a spiteful sample, which is misclassified by the DNN. Two categories of adversarial attacks that have preoccupied defensive approximations, regarding the knowledge of attackers, are White-Box attacks (when attackers know the training data, model parameters, and model architecture) and Black-Box attacks (when they don't know model information) [84]. The rest of them can be categorized according to different properties, e.g., the target of attack, the kind of perturbation, and the phase in which an attack occurs. These attributes are summarized and explained in Figure 15.

Most of the proposed methods that used approximate computing as defense aim at increasing the generalization of DNNs, as observations show they perform better against various types of attacks. It has been also noticed that low-precision models exhibit, in general, higher adversarial accuracy compared to the full-precision models with identical network structures [31, 89]. This could be explained due to quantization effect that enhances the amount of non-linearity, which prevents small changes in the input from drastically altering the output and forcing a misclassification.

Guesmi et al. [37], in order to handle such threats, proposed an approximate CNN implementation, where the exact multiplier is replaced with an approximate FP multiplier that injects data-dependent noise in convolution operations. This approximate mantissa multiplier induced an error, which was propagated through the whole model. It was observed that this error could further the difference between the first class and the "runner-up" and help the classifier to generalize and enhance its confidence. Experimental evaluation over LeNet-5 trained on MNIST showed that, for a negligible accuracy loss, the proposed defensive approximate scheme made the model 87.5% more robust against Black-Box attacks than the conventional CNN, with 50% power and 67% area reduction.

A quantization-based defense, which exploits low-precision implementations, was proposed in [60]. First, [60] selects a number of quantization levels based on the application's resilience to errors and perturbations. Next, an additional layer is added at the input of the network that has a one-to-one relation with the input pixels. This relation with the rest of the quantization scheme supports different configurations based on whether training is needed or not. The idea of this proposed defense is based on the observation that when the input of a CNN is quantized, the confidence of a clean image's prediction remains almost the same. On the contrary, the confidence

of an incorrect classification of a perturbed image is decreased. Evaluating the proposed scheme under different white-box and black-box settings showed an increase in the classification accuracy of perturbed images by up to 50% and 96% for the CIFAR10 and MNIST datasets, respectively.

7 CONCLUSIONS, CHALLENGES, AND PERSPECTIVE

In this article, hardware approximation techniques for DNNs are reviewed, characterized, classified, and evaluated. Moreover, we provide a comprehensive analysis of error metrics and error mitigation approaches specific for DNN approximations in order to provide an in-depth analysis of the studied field. Note that, in addition to the traditional exploitation of Approximate Computing for energy reduction, we present how approximate computing can be employed in DNNs to address reliability and security concerns. Our analysis clustered the hardware DNN approximation techniques into three categories: Precision Scaling, Computation Reduction, and Approximate Units.

Precision Scaling is the most widely used method and already adopted by most commercial DNN accelerators. Advancements in quantization-aware (re-)training methods have led to minimal accuracy loss even with 4-bit or 2-bit inference. Nevertheless, note that quantization-aware training can be very time consuming and post-training quantization approaches are efficient for 8-bit inference—which is considered mainstream today—and with some limitations might enable 4-bit inference.

The Computation Reduction approximation is demonstrated to deliver very high energy reduction for minimal accuracy loss. However, this approximate category mainly examines 32-bit inference, and the energy gains dropped significantly when considering more challenging evaluations such as 8-bit inference and/or ImageNet. As a result, to obtain conclusive results regarding the Computation Reduction, a more in-depth comprehensive analysis is required with respect either to more challenging evaluation scenarios or to NNs that indeed require high-precision inference.

After Precision Scaling, the Approximate Units category has attracted the highest research interest. Typically, Approximate Units are combined with low precision (mainly 8-bit inference). Again, despite the high energy gains reported in many cases, a more comprehensive and challenging evaluation is required. Although the results seem promising, evaluations on the state-of-the-art ImageNet datasets are still limited. Still, it is noteworthy that for small DNNs the Approximate Multipliers/Adders family delivers immense energy reduction with negligible accuracy loss. The latter appears ideal for IoT devices that need to run sophisticated DNN-based services. Moreover, although some works aimed at evaluating the tradeoff between low precision (8-bit and below) and approximate units in DNN inference and showed that a combination of the two outperforms the isolated application of very low precision, this correlation is not comprehensively analyzed yet. Finally, note that an inherent limitation is this category that many techniques require retraining to recover the accuracy loss. In contrast to quantization-aware training that can run efficiently on CPUs and GPUs, retraining with approximate units requires hardware emulation that can even become infeasible in complex DNNs. To avoid retraining, curable approximation and/or statistical error compensation methods appear to be very promising solutions but are still understudied.

Hardware approximation for DNNs has shown remarkable advancements over the past years, moving from simple DNNs to very complex ones. Although Approximate Computing has demonstrated great potential through some impressive results, significant innovation is required to enable hardware approximation to be actively adopted in the design of complex DNN accelerators. Finally, a crucial but not well-addressed topic is the relation between approximate computing and the standardization of ML-based systems in safety-critical applications. Safety standards for systems that are ML based are yet to be formalized [117], and as a result, the impact of approximate computing on the system's certification remains unclear. The examined hardware approximation techniques are deterministic and are not expected to impact the certification process. On the other hand,

despite the high inference accuracy achieved by these approximations, the ML system will not work as it was trained to do (due to the induced approximation during inference). Thus, this might hinder the certification of approximate ML-based systems in safety-critical scenarios. Nevertheless, such issues could be solved through approximation-aware retraining. As discussed in Section 6, approximate computing enhances the reliability and robustness of DNN accelerators and thus might ease the certification of the system. Overall, a deep investigation and analysis is required.

REFERENCES

[1] Ankur Agrawal et al. 2021. 9.1 A 7nm 4-Core AI chip with 25.6TFLOPS hybrid FP8 training, 102.4TOPS INT4 inference and workload-aware throttling. In *International Solid- State Circuits Conference (ISSCC'21)*, Vol. 64. 144–146. https://doi.org/10.1109/ISSCC42613.2021.9365791

[2] A. Agrawal et al. 2019. DLFloat: A 16-b floating point format designed for deep learning training and inference. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH'19)*. 92–95.

[3] A. Al Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini. 2018. XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks. In *IEEE Symposium in Low-Power and High-Speed Chips*. 1–3.

[4] H. Amrouch, G. Zervakis, S. Salamin, H. Kattan, I. Anagnostopoulos, and J. Henkel. 2020. NPU thermal management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3842–3855.

[5] Renzo Andri, Lukas Cavigelli, Davide Rossi, and L. Benini. 2016. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. In *Computer Society Annual Symposium on VLSI (ISVLSI'16)*. 236–241.

[6] Mohammad Saeed Ansari, Vojtech Mrazek, Bruce F. Cockburn, Lukas Sekanina, Zdenek Vasicek, and Jie Han. 2020. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 28, 2 (2020), 317–328.

[7] Arm. 2020. *Arm Ethos-N Processors*. https://developer.arm.com/ip-products/processors/machine-learning/arm-ethos-n.

[8] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).

[9] C. Barata and J. S. Marques. 2019. Deep learning for skin cancer diagnosis with hierarchical architectures. In *IEEE 16th International Symposium on Biomedical Imaging*. 841–845.

[10] Kartikeya Bhardwaj, Pravin S. Mane, and Jörg Henkel. 2014. Power- and area-efficient approximate Wallace tree multiplier for error-resilient systems. In *15th International Symposium on Quality Electronic Design*. 263–269. https://doi.org/10.1109/ISQED.2014.6783335

[11] Chris M. Bishop. 1995. Training with noise is equivalent to Tikhonov regularization. *Neural Computation* 7 (1995), 108–116.

[12] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. 2020. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. *IEEE Access* 8 (2020), 225134–225180.

[13] Cerebras. 2021. *Cerebras Wafer Scale Engine*. https://cerebras.net/.

[14] C. Chen, J. Choi, K. Gopalakrishnan, V. Srinivasan, and S. Venkataramani. 2018. Exploiting approximate computing for deep learning acceleration. In *Design, Automation Test in Europe Conference Exhibition (DATE'18)*. 821–826.

[15] Mia Xu Chen et al. 2018. The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 76–86.

[16] Yunji Chen et al. 2014. DaDianNao: A machine-learning supercomputer. In *Annu. Int. Symp. Microarch.* 609–622. https://doi.org/10.1109/MICRO.2014.58

[17] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. 2020. A survey of accelerator architectures for deep neural networks. *Engineering* 6, 3 (2020), 264–274.

[18] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Annual International Symposium on Computer Architecture (ISCA'16)*. 367–379. https://doi.org/10.1109/ISCA.2016.40

[19] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A flexible accelerator for emerging deep neural networks on Mobile devices. *Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (2019), 292–308. https://doi.org/10.1109/JETCAS.2019.2910232

[20] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi (Viji) Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. 2019. Accurate and efficient 2-bit quantized neural networks. In *Proc. of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 348–359.

[21] J. Choi, Z. Wang, S. Venkataramani, P. I-Jen Chuang, V. Srinivasan, and K. Gopalakrishnan. 2018. PACT: Parameterized clipping activation for quantized neural networks. *ArXiv* (2018). http://arxiv.org/abs/1503.02531.
[22] F. Conti, P. Davide Schiavone, and L. Benini. 2018. XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference. *IEEE Trans. on Computer-Aided Design of Integ. Circ. and Sys.* 37, 11 (2018), 2940–2951.
[23] M. Courbariaux, Y. Bengio, and J. David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Proc. of the 28th Int. Conf. on Neural Information Processing Systems*. 3123–3131.
[24] Cecilia De la Parra, Andre Guntoro, and A. Kumar. 2020. ProxSim: GPU-based simulation framework for cross-layer approximate DNN optimization. In *Design, Automation Test in Europe Conference Exhibition (DATE'20)*. 1193–1198.
[25] Cecilia De la Parra, Andre Guntoro, and A. Kumar. 2021. Efficient accuracy recovery in approximate neural networks by systematic error modelling. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC'21)*. 365–371.
[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.
[27] Lei Deng, Guoqi Li, Song Han, L. P. Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108 (2020), 1–48.
[28] Zhaoxia Deng, C. Xu, Qiong Cai, P. Faraboschi, and H. Packard. 2015. Reduced-precision memory value approximation for deep learning. *ArXiv*. https://arxiv.org/abs/1511.05236.
[29] Julian Faraone et al. 2020. AddNet: Deep neural networks using FPGA-optimized multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28 (2020), 115–128.
[30] Bruce Fleischer et al. 2018. A scalable multi- TeraOPS deep learning processor core for AI training and inference. In *2018 IEEE Symposium on VLSI Circuits*. 35–36. https://doi.org/10.1109/VLSIC.2018.8502276
[31] Angus Galloway, Graham W. Taylor, and Medhat Moussa. 2018. Attacking binarized neural networks. *ArXiv* (2018). https://arxiv.org/abs/1711.00449.
[32] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A survey of quantization methods for efficient neural network inference. *ArXiv* (2021). https://arxiv.org/abs/2103.13630.
[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.
[34] Priya Goyal et al. 2018. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677
[35] Graphcore. 2020. *Intelligence Processing Unit*. https://www.graphcore.ai/products/ipu.
[36] Groq. 2021. *Tensor Streaming Processor*. https://groq.com/technology/.
[37] Amira Guesmi et al. 2020. Defensive approximation: Enhancing CNNs security through approximate computing. *ArXiv* (2020). https://arxiv.org/abs/2006.07700.
[38] Chuliang Guo, Li Zhang, Xian Zhou, Weikang Qian, and Cheng Zhuo. 2020. A reconfigurable approximate multiplier for quantized CNN applications. In *Asia and South Pacific Design Automation Conference (ASP-DAC'20)*. 235–240.
[39] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, Volume 37. 1737–1746.
[40] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi. 2018. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Trans. on Neural Networks and Learning Sys.* 29, 11 (2018) 5784–5789.
[41] Issam Hammad, Ling Li, Kamal El-Sankary, and W. Martin Snelgrove. 2021. CNN inference using a preprocessing precision controller and approximate multipliers with various precisions. *IEEE Access* 9 (2021), 7220–7232.
[42] Jie Han and Michael Orshansky. 2013. Approximate computing: An emerging paradigm for energy-efficient design. In *18th IEEE European Test Symposium (ETS'13)*. 1–6. https://doi.org/10.1109/ETS.2013.6569370
[43] Song Han et al. 2016. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*. 243–254.
[44] Muhammad Abdullah Hanif, Faiq Khalid, and Muhammad Shafique. 2019. CANN: Curable approximations for high-performance deep neural network accelerators In. *Design Automation Conference (DAC'19)*.
[45] Cong Hao, Jordan Dotzel, Jinjun Xiong, Luca Benini, Zhiru Zhang, and Deming Chen. 2021. Enabling design methodologies and future trends for edge AI: Specialization and co-design. *IEEE Design & Test* 38, 4 (2021), 7–26.
[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*. 770–778.
[47] Xin He, Liu Ke, Wenyan Lu, Guihai Yan, and Xuan Zhang. 2018. AxTrain: Hardware-oriented neural network training for approximate inference. In *Proc. of the Int. Symp. on Low Power Electronics and Design*. Article 20, 6 pages.
[48] Maedeh Hemmat, Joshua San Miguel, and Azadeh Davoodi. 2020. AirNN: A featherweight framework for dynamic input-dependent approximation of CNNs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 10 (2021), 2090–2103.

[49] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.
[50] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[51] M. Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC'14)*. 10–14.
[52] Yuxiang Huan, Yifan Qin, Yantian You, Lirong Zheng, and Zhuo Zou. 2016. A multiplication reduction technique with near-zero approximation for embedded learning in IoT devices. In *International System on Chip Conference*, 102–107.
[53] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *International Conference on Neural Information Processing Systems*. 4114–4122.
[54] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, Vol. 37. 448–456.
[55] Benoit Jacob et al. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2704–2713.
[56] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han. 2020. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proc. IEEE* 108, 12 (2020), 2108–2135.
[57] X. Jiao, V. Akhlaghi, Yu Jiang, and R. K. Gupta. 2018. Energy-efficient neural networks using approximate computation reuse. *Proc. of the 2018 Design, Automation and Test in Europe Conference and Exhibition (DATE'18)*. 1223–1228.
[58] Norman Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. *SIGARCH Comput. Archit. News* 45, 2 (jun 2017), 1–12.
[59] Sangil Jung et al. 2019. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Conf. on Computer Vision and Pattern Recognition (CVPR'19)*. 4345–4354.
[60] Faiq Khalid et al. 2019. QuSecNets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks. In *Int. Symp. on On-Line Testing and Robust System Design (IOLTS'19)*. 182–187.
[61] Minkyu Kim and Jae Sun Seo. 2021. An energy-efficient deep convolutional neural network accelerator featuring conditional computing and low external memory access. *IEEE Journal of Solid-State Circuits* 56, 3 (2021), 803–813.
[62] Min Soo Kim, Alberto A. Del Barrio, Leonardo Tavares Oliveira, Roman Hermida, and Nader Bagherzadeh. 2019. Efficient Mitchell's approximate log multipliers for convolutional neural networks. *IEEE Trans. Comput.* 68, 5 (2019), 660–675.
[63] Skanda Koppula et al. 2019. EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM. In *Proc. of the 52nd Annual IEEE/ACM Int. Symp. on Microarchitecture (MICRO'19)*. 166–181. https://doi.org/10.1145/3352460.3358280
[64] Skanda Koppula et al. 2019. EDEN: Enabling energy-efficient, high-performance deep neural network inference using approximate DRAM. In *Proc. of the 52nd Annual IEEE/ACM Int. Symp. on Microarchitecture*. 166–181.
[65] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv* (6 2018). http://arxiv.org/abs/1806.08342.
[66] Alex Krizhevsky. 2012. Learning multiple layers of features from tiny images. University of Toronto.
[67] Cecilia De la Parra. 2020. Knowledge distillation and gradient estimation for active error compensation in approximate neural networks. (2020).
[68] Y. LeCun et al. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1, 4 (1989), 541–551.
[69] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[70] Chi Sing Leung, Hong Jiang Wang, and John Sum. 2010. On the selection of weight decay parameter for faulty networks. *IEEE Transactions on Neural Networks* 21, 8 (2010), 1232–1244.
[71] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi. 2021. Pruning and quantization for deep neural network acceleration: A survey. *ArXiv* (2021). https://arxiv.org/abs/2101.09671.
[72] Heng Liao, Jiajin Tu, Jing Xia, and Xiping Zhou. 2019. DaVinci: A scalable architecture for neural network computing. In *2019 IEEE Hot Chips 31 Symposium (HCS'19)*. 1–44. https://doi.org/10.1109/HOTCHIPS.2019.8875654
[73] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. 2020. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 2 (2020), 318–327.
[74] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2016. Neural networks with few multiplications. *ArXiv* (2016). https://arxiv.org/abs/1510.03009.
[75] John N. Mitchell. 1962. Computer multiplication and division using binary logarithms. *IRE Transactions on Electronic Computers* EC-11 (1962), 512–517.
[76] Luca Mocerino, Valerio Tenace, and Andrea Calimera. 2019. Energy-efficient convolutional neural networks via recurrent data reuse. In *Design, Automation Test in Europe Conference Exhibition (DATE'19)*. 848–853. https://doi.org/10.23919/DATE.2019.8714880

[77] Heba Mohsen, El-Sayed A. El-Dahshan, El-Sayed M. El-Horbaty, and Abdel-Badeeh M. Salem. 2018. Classification using deep learning neural networks for brain tumors. *Future Computing and Informatics Journal* 3, 1 (2018), 68–71.
[78] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. 2017. EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation and Test in Europe (DATE'17)*, 258–261.
[79] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and K. Roy. 2016. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *Int. Conf. on Computer-Aided Design (ICCAD'16)*.
[80] Vojtech Mrazek, Lukas Sekanina, and Zdenek Vasicek. 2020. Libraries of approximate circuits: Automated design and application in CNN accelerators. *IEEE J. Emerg. Sel. Topics Circuits Syst.* 10 (2020), 406–418.
[81] Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina, Muhammad Abdullah Hanif, and Muhammad Shafique. 2019. ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining. In *Int. Conference on Computer-Aided Design (ICCAD'19)*.
[82] Luis Muñoz-González and Emil C. Lupu. 2019. *The Security of Machine Learning Systems*. 47–79.
[83] Naresh Nagabushan, Nishank Satish, and S. Raghuram. 2016. Effect of injected noise in deep neural networks. In *2016 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC'16)*. 1–5. https://doi.org/10.1109/ICCIC.2016.7919546
[84] Nina Narodytska and Shiva Kasiviswanathan. 2017. Simple black-box adversarial attacks on deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'17)*. 1310–1318.
[85] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Ng. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.
[86] NVIDIA. 2020. *A100 Tensor Core GPU Architecture*. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/nvidia-ampere-architecture-whitepaper.pdf.
[87] S. Pagani, P. D. S. Manoj, A. Jantsch, and J. Henkel. 2020. Machine learning for power, energy, and thermal management on multicore processors: A survey. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 39, 1 (2020), 101–116. https://doi.org/10.1109/TCAD.2018.2878168
[88] Rasmus Berg Palm. 2012. Prediction as a candidate for learning deep hierarchical models of data. Technical University of Denmark, DTU Informatics.
[89] Priyadarshini Panda, Indranil Chakraborty, and Kaushik Roy. 2019. Discretization based solutions for secure machine learning against adversarial attacks. *IEEE Access* 7 (2019), 70157–70168.
[90] Angshuman Parashar et al. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 27–40.
[91] Jun-Seok Park et al. 2021. 9.5 A 6K-MAC feature-map-sparsity-aware neural processing unit in 5nm flagship Mobile SoC. In *IEEE International Solid- State Circuits Conference (ISSCC'21)*, Vol. 64. 152–154. https://doi.org/10.1109/ISSCC42613.2021.9365928
[92] Yashrajsinh Parmar and K. Sridharan. 2020. A resource-efficient multiplierless systolic array architecture for convolutions in deep networks. *IEEE Trans. Circuits Syst., II, Exp. Briefs* 67, 2 (Feb. 2020), 370–374.
[93] Ratko Pilipović, Patricio Bulić, and Uroš Lotrič. 2021. A two-stage operand trimming approximate logarithmic multiplier. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 6 (2021), 1–11.
[94] Duvindu Piyasena, Rukshan Wickramasinghe, Debdeep Paul, Siew Kei Lam, and Meiqing Wu. 2019. Reducing dynamic power in streaming CNN hardware accelerators by exploiting computational redundancies. In *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications (FPL'19)*. 354–359.
[95] Jiantao Qiu et al. 2016. Going deeper with embedded FPGA platform for convolutional neural network. *International Symposium on Field-Programmable Gate Arrays (FPGA'16)*. 26–35.
[96] Martin Rapp et al. 2021. MLCAD: A survey of research in machine learning for CAD keynote paper. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. https://doi.org/10.1109/TCAD.2021.3124762
[97] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Computer Vision (ECCV'16)*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 525–542.
[98] Sherief Reda and Muhammad Shafique. 2018. *Approximate Circuits: Methodologies and CAD* (1st ed.). Springer Publishing Company, Incorporated.
[99] Pengzhen Ren et al. 2021. A comprehensive survey of neural architecture search: Challenges and solutions. *ArXiv* (2021). https://arxiv.org/abs/2006.02903.
[100] Mohsin Riaz et al. 2020. CAxCNN: Towards the use of canonic sign digit based approximation for hardware-friendly convolutional neural networks. *IEEE Access* 8 (2020), 127014–127021.
[101] Marc Riera, Jose-Maria Arnau, and Antonio Gonzalez. 2018. Computation reuse in DNNs by exploiting input similarity. In *Annual International Symposium on Computer Architecture (ISCA'18)*. 57–68.

[102] S. Ryu, H. Kim, W. Yi, and J. Kim. 2019. BitBlade: Area and energy-efficient precision-scalable neural network accelerator with bitwise summation. In *2019 56th ACM/IEEE Design Automation Conference (DAC'19)*. 1–6.
[103] H. Saadat, H. Bokhari, and S. Parameswaran. 2018. Minimally biased multipliers for approximate integer and floating-point multiplication. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37 (2018), 2623–2635.
[104] Sami Salamin, Georgios Zervakis, Ourania Spantidi, Iraklis Anagnostopoulos, Jörg Henkel, and H. Amrouch. 2021. Reliability-aware quantization for anti-aging NPUs. In *Design, Automation Test in Europe Conf. Exhibition (DATE'21)*.
[105] Syed Shakib Sarwar, Swagath Venkataramani, Aayush Ankit, Anand Raghunathan, and Kaushik Roy. 2018. Energy-efficient neural computing with approximate multipliers. *J. Emerg. Technol. Comput. Syst.* 14, 2 (2018), 23.
[106] Oleg Semery. 2021. *pytorchcv · PyPI*. https://pypi.org/project/pytorchcv/.
[107] A. Shafiee et al. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Annual Int. Symposium on Computer Architecture (ISCA'16)*. 14–26. https://doi.org/10.1109/ISCA.2016.12
[108] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. 2015. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC'15)*. 1–6. https://doi.org/10.1145/2744769.2744778
[109] M. Shafique et al. 2020. Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead. *IEEE Design Test* 37, 2 (2020), 30–57.
[110] Yang Shao, Gregory N. Taff, and Stephen J. Walsh. 2011. Comparison of early stopping criteria for neural-network-based subpixel classification. *IEEE Geoscience and Remote Sensing Letters* 8, 1 (1 2011), 113–117.
[111] S. Sharify, A. D. Lascorz, K. Siu, P. Judd, and A. Moshovos. 2018. Loom: Exploiting weight and activation precisions to accelerate convolutional neural networks. In *Design Automation Conference (DAC'18)*. 1–6.
[112] H. Sharma et al. 2018. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network.In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA'18)*. 764–775.
[113] David R. So, Chen Liang, and Quoc V. Le. 2019. The Evolved Transformer. arXiv:1901.11117
[114] Taha Soliman, Cecilia De La Parra, Andre Guntoro, and Norbert Wehn. 2021. Adaptable approximation based on bit decomposition for deep neural network accelerators. In *Int. Conf. Artificial Intelligence Circuits and Systems*. 1–4.
[115] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[116] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105 (2017), 2295–2329.
[117] Florian Tambon et al. 2021. How to Certify Machine Learning Based Safety-critical Systems? A Systematic Literature Review. arXiv:2107.12045 [cs.LG]
[118] Zois Gerasimos Tasoulas, Georgios Zervakis, Iraklis Anagnostopoulos, Hussam Amrouch, and Jorg Henkel. 2020. Weight-oriented approximation for energy-efficient neural network inference accelerators. *IEEE Trans. Circuits Syst. I, Reg. Papers* 67 (2020), 4670–4683.
[119] T. Ujiie, M. Hiromoto, and T. Sato. 2016. Approximated prediction strategy for reducing power consumption of convolutional neural network processor. In *Conf. on Comp. Vision and Pattern Recog. Workshops (CVPRW'16)*. 870–876.
[120] Y. Umuroglu, L. Rasnayake, and M. Själander. 2018. BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing. In *2018 28th Int. Conf. on Field Programmable Logic and Applications (FPL'18)*. 307–3077.
[121] Zdenek Vasicek, Vojtech Mrazek, and Lukas Sekanina. 2019. Automated circuit approximation method driven by data distribution. In*Design, Automation and Test in Europe Conference Exhibition (DATE'19)*. 96–101.
[122] Z. Vasicek and L. Sekanina. 2015. Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation* 19, 3 (2015), 432–444.
[123] F. Vaverka, V. Mrazek, Z. Vasicek, and L. Sekanina. 2020. TFApprox: Towards a Fast Emulation of DNN Approximate Hardware Accelerators on GPU. 294–297.
[124] Swagath Venkataramani, Ashish Ranjan, K. Roy, and A. Raghunathan. 2014. AxNN: Energy-efficient neuromorphic systems using approximate computing. In *Int. Symp. on Low Power Electronics and Design (ISLPED'14)*. 27–32.
[125] Swagath Venkataramani et al. 2020. Efficient AI system design with cross-layer approximate computing. *Proc. IEEE* 108, 12 (2020), 2232–2250.
[126] Sebastian Vogel, Mengyu Liang, Andre Guntoro, Walter Stechele, and Gerd Ascheid. 2018. Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers (ICCAD'18)*. 1–8.
[127] Sebastian Vogel, Jannik Springer, Andre Guntoro, and Gerd Ascheid. 2019. Self-supervised quantization of pre-trained neural networks for multiplierless acceleration. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1094–1099.
[128] Naigang Wang, Jungwook Choi, D. Brand, Chia-Yu Chen, and K. Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. In *NeurIPS*.

[129] Ofri Wechsler, Michael Behar, and Bharat Daga. 2019. Spring hill (NNP-I 1000) Intel's data center inference chip. In *2019 IEEE Hot Chips 31 Symposium*. 1–12. https://doi.org/10.1109/HOTCHIPS.2019.8875671

[130] Tien-Ju Yang, Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2017. A method to estimate the energy consumption of deep neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*. 1916–1920. https://doi.org/10.1109/ACSSC.2017.8335698

[131] Georgios Zervakis, Hussam Amrouch, and Jörg Henkel. 2020. Design automation of approximate circuits with runtime reconfigurable accuracy. *IEEE Access* 8 (2020), 53522–53538. https://doi.org/10.1109/ACCESS.2020.2981395

[132] G. Zervakis, H. Saadat, H. Amrouch, A. Gerstlauer, S. Parameswaran, and J. Henkel. 2021. Approximate computing for ML: State-of-the-art, challenges and visions. In *Asia and South Pacific Design Automation Conference*. 189–196.

[133] Georgios Zervakis, Ourania Spantidi, Iraklis Anagnostopoulos, Hussam Amrouch, and Jörg Henkel. 2021. Control variate approximation for DNN accelerators. In *58th ACM/IEEE Design Automation Conference (DAC'21)*. 481–486. https://doi.org/10.1109/DAC18074.2021.9586092

[134] Georgios Zervakis, Kostas Tsoumanis, Sotirios Xydis, Dimitrios Soudris, and Kiamal Pekmestzi. 2016. Design-efficient approximate multiplication circuits through partial product perforation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 10 (2016), 3105–3117. https://doi.org/10.1109/TVLSI.2016.2535398

[135] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. 2018. LQ-nets: Learned quantization for highly accurate and compact deep neural networks. In *Computer Vision (ECCV'18)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.). 373–390.

[136] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. 2015. ApproxANN: An approximate computing framework for artificial neural network. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE'15)*. 701–706.

[137] Aojun Zhou, Anbang Yao, Yiwen Guo, L. Xu, and Y. Chen. 2017. Incremental network quantization: Towards lossless CNNs with low-precision weights. *ArXiv* (2017). https://arxiv.org/abs/1702.03044.

[138] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2018. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. arXiv:1606.06160 [cs.NE]

[139] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. 2017. Trained ternary quantization. In *5th International Conference on Learning Representations (ICLR'17)*.

Received 17 June 2021; revised 8 March 2022; accepted 14 March 2022