On-Device Deep Learning: Survey on Techniques Improving Energy Efficiency of DNNs

Anais Boumendil, Walid Bechkit, and Karima Benatchba

***Abstract*— Providing high-quality predictions is no longer the sole goal for neural networks. As we live in an increasingly interconnected world, these models need to match the constraints of resource-limited devices powering the Internet of Things (IoT) and embedded systems. Moreover, in the era of climate change, reducing the carbon footprint of neural networks is a critical step for green artificial intelligence, which is no longer an aspiration but a major need. Enhancing the energy efficiency of neural networks, in both training and inference phases, became a predominant research topic in the field. Training optimization has grown in interest recently but remains challenging, as it involves changes in the learning procedure that can impact the prediction quality significantly. This article presents a study on the most popular techniques aiming to reduce the energy consumption of neural networks' training. We first propose a classification of the methods before discussing and comparing the different categories. In addition, we outline some energy measurement techniques. We discuss the limitations identified during our study as well as some interesting directions, such as neuromorphic and reservoir computing (RC).**

***Index Terms*— Deep learning, efficiency, energy, neural networks, resource consumption.**

I. Introduction

In recent years, deep neural networks (DNNs) have gained huge popularity and become one of the major algorithms in artificial intelligence. DNNs allow remarkable improvements in many disciplines. For instance, convolutional neural networks (CNNs) provide impressive results in computer vision, especially through the series of models proposed for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [1], [2], [3]. These improvements are useful for many vision-related domains such as medical image analysis [4], autonomous vehicles [5], [6], and much more. DNNs also have a broad impact on natural language processing (NLP) [7], [8], [9], well illustrated by the impressive performance of ChatGPT [10], [11].

The improvements in DNNs' performances come with their growing complexity. If we take a look at computer vision models, the AlexNet [1] architecture contains around 60 million parameters, while VGG-16 [12] has a total of 138 million. Language models went further with 213 million parameters for the Transformer big model [7], 175 billion for GPT-3 [9], and one trillion for GPT-4 [10]. Nevertheless, increasing the number of parameters to capture the slightest improvements in prediction quality comes with an important growth in training costs and energy consumption.

There is an obvious mismatch between the energy requirements of neural networks and the available resources in the Internet of Things (IoT) and mobile devices. The cloud is a suitable solution to use DNNs in resource-constrained domains. It allows running the model on powerful hardware while sending the prediction results to low-power devices. However, direct integration of neural networks on resource-constrained equipment may be better for some applications with latency or privacy constraints [13], [14]. Moreover, optimizing energy consumption remains necessary on powerful hardware as well, especially to limit the carbon footprint of DNNs and their worrying environmental impact. Indeed, the study of Strubell et al. [15] shows that training the Transformer big model [7] with the neural architecture search (NAS) technique emits a higher quantity of CO_2 than five cars in their entire lifetime.

DNNs' efficiency became a major line of research that targeted both inference and training phases. Optimizing the training might be critical, as any change in the learning procedure affects the prediction performances of the model. Moreover, the choice of which changes to apply to improve the efficiency requires prior knowledge generally brought by training itself. Despite these difficulties, many studies provided encouraging results and paved the way for more efficient training. In this article, we review some of these works. We still include some approaches proposed for inference optimization when they can be adapted for training.

Our review shows that it is possible to optimize DNNs' processing at different levels. Indeed, some methods apply changes to the architecture to make it less expensive. Other techniques work on the hardware layer to make DNN training or inference faster and more energy-efficient. It is also possible to optimize the training by reducing the dataset size through

Manuscript received 7 June 2023; revised 7 April 2024; accepted 10 July 2024. This work was supported in part by French Embassy, Algeria, through the Project FSPI N°2021-016 LISEN 2020-02; and in part by the INSA Lyon-SPIE ICS Chair on Artificial Intelligence for Behavioral Flow Analysis in Digital Infrastructures. *(Corresponding author: Anais Boumendil.)*

Anais Boumendil is with INSA Lyon, Inria, CITI, UR3720, 69621 Villeurbanne, France, and also with École nationale Supérieure d'Informatique, Laboratoire de Méthodes de conception des Systèmes, Algiers 16059, Algeria (e-mail: anais.boumendil@insa-lyon.fr).

Walid Bechkit is with INSA Lyon, Inria, CITI, UR3720, 69621 Villeurbanne, France (e-mail: walid.bechkit@insa-lyon.fr).

Karima Benatchba is with the École nationale Supérieure d'Informatique, Laboratoire de Méthodes de conception des Systèmes, Algiers 16059, Algeria (e-mail: k_benatchba@esi.dz).

Digital Object Identifier 10.1109/TNNLS.2024.3430028

data selection. The diversity of the existing methods motivated us to propose a new classification.

This article is structured as follows. We first introduce the classification we propose, and we compare it to previous work. Second, we describe every category and review some representative methods of each class. Then, we discuss how energy efficiency is evaluated in the literature. Finally, we provide a summary of our study and discuss some limitations as well as some emerging and promising paradigms.

II. Classification of Methods

In this section, we first review the main representative surveys on efficient DNNs before presenting our own classification. We note that the different approaches that we mention here will be explained and discussed later in more detail.

A. Existing Classifications

Multiple studies targeted the resource efficiency of DNNs. The classification proposed in [16] gathers, on one side, the optimized hardware platforms used for DNNs' processing, and on the other side, algorithmic approaches including precision reduction, pruning, knowledge distillation, and compact architecture design. Roth et al. [17] describe three research directions to optimize the resource consumption of neural networks, namely, quantization, pruning, and structural efficiency (through knowledge distillation, manual design of efficient architectures, and automatic architecture search). Menghani [18] provides a five-category classification: compression methods (pruning and quantization), methods modifying the learning procedure (such as knowledge distillation), manual design of efficient architectures, automation mainly through NAS, and infrastructure (hardware and software) optimization.

Berthelier et al. [19] propose a classification of two classes. The first category includes compression techniques, while the second is named architecture optimization. The latter is divided into two subcategories: "architecture overview" that includes some optimized handcrafted architectures and "NAS" that is described in great detail.

Cai et al. [20] address compression approaches, AutoML frameworks for model compression, and efficient on-device training. They focus on mobile devices and specific tasks as point cloud, video, and NLP. Liu et al. [21] discuss useful techniques for edge intelligence systems, including model compression, hardware-aware NAS, handcrafted models, as well as adaptive models where different channels or layers are activated for each example during inference [22], [23]. Indeed, different parts of the DNN are needed from an input to another to obtain accurate predictions [24]. Lee et al. [25] classify the approaches into model level when the model size is compressed, arithmetic level when the precision is reduced, and implementation level when a hardware optimization is applied. Their survey focuses on CNNs. Mehlin et al. [26] provide an overview on how energy-efficient techniques are applied within the deep learning life cycle.

The survey of Bartoldson et al. [27] targets the training phase and the algorithmic approaches. They propose a classification by asking four questions.

1) Where to apply changes? function (architecture and model parameters), data (training data and derived data), and optimization (algorithm, objective).
2) What changes to make? The authors refer to the possible changes as the 5Rs of speedup: remove, restrict, reorder, replace, and retrofit (add a component with the aim of reducing the number of training iterations).
3) When to make changes? static (at initialization) or dynamic (during training).
4) How to make changes? according to prior knowledge (manual changes) or according to learned knowledge (by considering current or past trainings).

B. Proposed Classification

In comparison with the previous work, our survey is mainly directed toward training but does not exclude techniques related to the inference phase, as far as they can be adapted to training. We focus on energy consumption, but we mention techniques that can optimize other resources. Our classification is, therefore, general and inclusive, yet simple. We summarize it in Fig. 1.

In the first class, we gather compression methods: 1) pruning, which removes unimportant weights from the model; 2) quantization, which acts by reducing the precision; 3) reducing the number of multiplications; and 4) knowledge distillation, which refers to the use of a small student model guided by a more complex teacher network. Compression methods generally start from architectures with a good prediction quality and try to compress them to reduce their resource consumption while keeping the best possible tradeoff with accuracy.

The second class acts during the design phase and builds efficient architectures in terms of resource consumption and prediction quality. We divide it into two subgroups: the manual design group mainly gathering optimized handcrafted architectures and the automatic design group gathering NAS and growing neural network approaches.

The third class gathers data selection approaches. Neural networks are known for being data-intensive, as large datasets are needed for their training. Often, they are composed of redundant examples and data of varying difficulty. Therefore, if there is a way to find the most relevant data, the model can be trained with fewer examples. If the selected subset changes throughout training, the data selection strategy is adaptive. In contrast, if the selected subset remains the same during the entire training, the data selection strategy is nonadaptive. Decreasing the number of training examples considerably speeds up training and reduces energy consumption. This class, unlike the other ones, is exclusively oriented toward the training phase. Moreover, it is rarely addressed in previous surveys.

The last class concerns infrastructure optimization. For this class, we discuss the efforts made on the hardware platform design, including temporal and spatial architectures. We also discuss the adaptation of software/frameworks to enable DNN computing on resource-constrained devices.

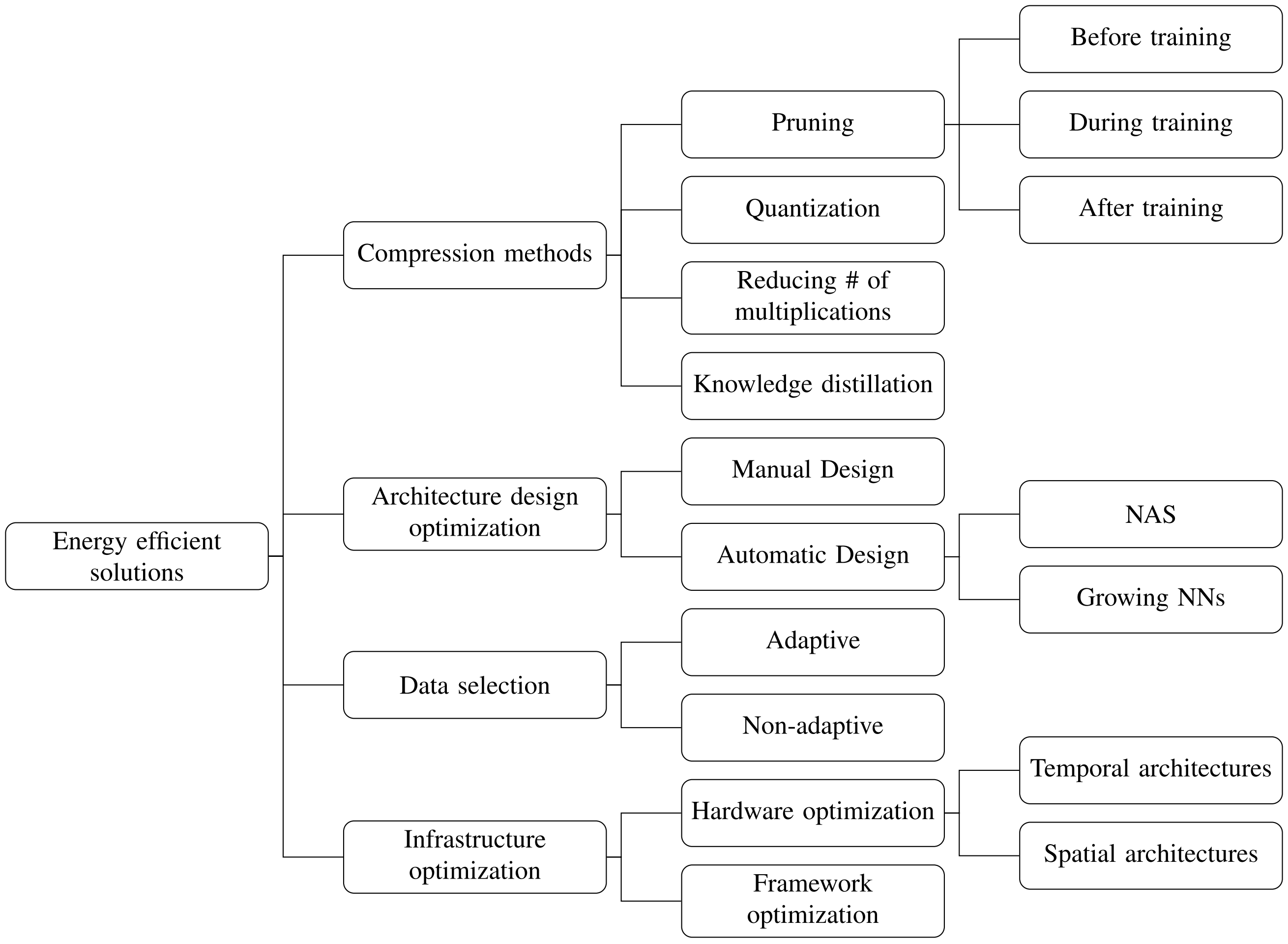


Fig. 1. Proposed classification.

III. Compression Methods

They consist in reducing the size or the computations of an existing model. Such approaches are often applied to overparameterized architectures. The compression methods aim to reduce their memory and energy consumption while keeping a good prediction level. In the following, we will present four techniques: pruning, quantization, reducing the number of multiplications, and knowledge distillation.

A. Pruning

Pruning is the process of removing parameters from a neural network. It is usually based on a score reflecting the importance of every parameter. The parameters with the lowest scores are removed according to a given pruning rate, which defines the percentage of weights to keep. The latter has to be fixed carefully to limit accuracy drops.

When individual weights are removed, pruning is unstructured. This type is implemented through masking weight matrices, which results in sparse structures. The pruned weights are not really removed but are rather replaced with zeros. Therefore, unstructured pruning requires custom hardware or specialized libraries to take advantage of the sparsity and offer energy gains.

When groups of weights as channels or filters are removed simultaneously, pruning is structured. This type allows dimension shrinking and results in a smaller model with fewer parameters. It can, therefore, offer energy gains on any hardware platform without further optimization.

Pruning can be applied at different stages of the model's life cycle. We first discuss some approaches that apply it after or during training. In this case, pruning mainly improves the energy efficiency of inference. We will then describe approaches that prune the network early during training or even at initialization to reduce training costs.

LeCun et al. [28] proposed one of the earliest pruning approaches, where they train a network until a quality solution is found. Then, they alternate between pruning individual weights and retraining the network. The pruning scores are computed with the Hessian matrix of the objective function with respect to each weight. This iterative train-prune-train framework is still used in recent approaches.

Yang et al. [29] guide the pruning by the energy consumption, as they first define a methodology to estimate it according to memory accesses and computations. Then, they repeat the following steps until the accuracy decreases below a given threshold.

1) *Order the Layers According to Their Energy Consumption:* The goal is to prune the weights of the most energy-hungry layers first. Indeed, the more weights are removed, the more difficult it will be to prune others. Imposing this order associates a priority with pruning.

2) *Prune Weights With Small Magnitude:* This strategy is the most common way to prune weights because of its speed and simplicity. However, it may increase the error due to the correlation between weights. The following step is a solution to this problem.
3) *Restore the Value of Some Weights:* In this step, the authors define an optimization problem whose solution allows restoring the weights that will minimize an error term.
4) *Locally Fine-Tune Weights:* By performing least-squares optimization on filters.
5) *Globally Fine-Tune:* By applying backpropagation.

The tests were performed on the 2014 ImageNet dataset [30] using an accelerator [31]. The proposed method reduces the inference energy consumption by 3.7× for the AlexNet architecture [1] and by 1.6× for the GoogLeNet model [2], while the accuracy degradation is lower than 1%.

Li et al. [32] prune the filters with the lowest L_1 norm, layer-wise. The approach is applied on a well-trained model that is fine-tuned after pruning. L_1-norm pruning can be seen as the generalization of magnitude-weight pruning to filters. This simple yet effective strategy allows reducing inference costs by 34% for the VGG-16 [12] model and by 38% [3] for the ResNet-110 model while keeping the accuracy close to the original model on the CIFAR-10 dataset [33]. Wen et al. [34] propose a method to learn a compressed structure of a given model during training by adjusting the structure of filters, channels, as well as the filter shape and the network depth. The approach proposed in [35] encourages filter sparsity by adding a regularization term (L_1/L_2 pseudonorm) to the loss function. With this approach, no fine-tuning is needed after pruning. Other structured pruning methods were proposed in [36], [37], [38], and [39].

1) Pruning at Initialization: Most of the previous approaches prune a well-trained model and then fine-tune it, which is only beneficial for inference efficiency. To reduce training costs, it is necessary to prune the network in early epochs or even at initialization, as in [40], [41], and [42]. Single-shot network pruning (SNIP) [40] aims to identify important connections independently of the value of the weights, by associating each connection to a variable c_i, which takes the value 1 when the connection i is active and the value 0 when it is pruned. The connection sensitivity is defined as the derivative of the error with respect to c_i and is used to compute pruning scores. The least important connections are pruned at initialization, and the network is then trained in the usual way.

Tanaka et al. [42] design SynFlow, an iterative and data-free pruning at initialization algorithm. SynFlow is designed to avoid layer collapse, one of the limits of pruning at initialization techniques. Layer collapse occurs when an algorithm prunes an entire layer, while prunable parameters remain elsewhere in the network [42], which makes the model untrainable and leads to a sudden accuracy drop.

If the techniques described above require multiple computations to measure the importance of weights, a recent work [43] shows that random pruning at initialization can be effective for deep models when choosing proper layer-wise sparsity ratios. For instance, pruning more parameters in the largest layers is a simple yet effective strategy [43].

Despite being a promising perspective to enhance training efficiency, pruning at initialization techniques does not outperform magnitude pruning after training [44]. To explain this gap in performances, Frankle et al. [44] studied the behavior of pruning methods at initialization (SNIP [40], GradSP [41], and SynFlow [42]). The results show that the three techniques are not sensitive to reinitialization, which is a critical point for other pruning algorithms. Moreover, these strategies are not sensitive to the random shuffling of pruning masks, which shows that the information used for pruning is not related to the weights individually but is rather layer-wise. Interestingly, GradSP [41] keeps good performances when inverting the pruning scores and thus pruning the supposed most important connections. This result raises many questions about the behavior of this technique. Finally, pruning with the three techniques later during training improves their results. This shows that their lower performances are not intrinsic to the methods but are rather due to applying them at initialization.

2) Lottery Ticket Hypothesis: The lottery ticket hypothesis reveals the existence of trainable subnetworks, called winning tickets, able to reach the performances of the full model while being smaller and less costly [45]. It is considered as an important finding since the common belief was that pruned networks are hard to train from scratch and pain to reach the accuracy of the original model [45].

In order to find the winning ticket, the original work [45] prunes the small magnitude weights of a trained network before restoring the starting initialization, which is a crucial point for hypothesis. The search procedure for the winning ticket requires training the original model, which is costly and slow. Indeed, revealing the existence of trainable subnetworks is important, but two questions have to be answered in order to prove the usefulness of the hypothesis: 1) are winning tickets reusable through datasets and optimizers? and 2) is it possible to find winning tickets early during training?

The first question addresses the generality of winning tickets through training configurations (datasets and optimizers). This property is important as it insures that a winning ticket can be reused and avoids to search for a new one for every change in the training settings. Morcos et al. [46] evaluate the performance of a ticket found on a source configuration (dataset and optimizer) on a target configuration (different dataset or optimizer). The results show that winning tickets can be transferred between optimizers and datasets of the same domain. Moreover, the tickets generated on larger datasets or datasets with a higher number of classes perform better when transferred. This brings encouraging results on the generality of winning tickets.

You et al. [47] provide a positive answer to the second question as they show that good quality tickets can be identified by the 20th epoch for a 160-epoch training (on the CIFAR-10/100 datasets [33] with the VGG-16 [12] and PreResNet101 [48] architectures). The proposed search procedure is summarized in Algorithm 1. It consists in pruning channels [36] and keeping binary masks describing the pruning scheme. When the Hamming distance between two consecutive masks is below a threshold, the winning ticket is found since the structure of the pruned network becomes stable.

Algorithm 1 Search for Winning Ticket [47]

```
Entries: $nbEpochs$, $threshold$
for epoch in $nbEpochs$ do
    update model parameters
    prune according to [36]
    compute pruning mask
    compute Hamming distance $d$ between masks
    if $d < threshold$ then
        winning ticket found
```

To verify that their method allows training in a resource-limited environment, You et al. [47] perform their experiments on the *Nvidia Jetson TX2* platform. The tests on CIFAR-10 [33] show that the method achieves energy gains of 8.6× for a pruning ratio of 50% and gains of 10.2× for a pruning ratio of 70% while maintaining accuracies above 92%. Thus, the proposed approach allows compressing the neural network, to reduce energy consumption while keeping very good prediction performances.

To sum up, pruning is a very popular compression method allowing to reduce the cost of DNN computing. There is a wide literature on pruning methods, we discussed some representative works, and other references can be found in [49], [50], [51], [52], [53], [54], [55], [56], [57], and [58].

B. Quantization

Typically, the weights and activations of a neural network are represented on 32 bits in the floating-point format. Quantization reduces precision by shrinking the number of representation bits.

Courbariaux et al. [59] propose two different ways to train a network with binary weights (−1 or 1). The first method consists in binarizing the weights in a deterministic way according to the following formula:

$$w_b = \begin{cases} +1, & \text{if, } w \geq 0 \\ -1, & \text{otherwise.} \end{cases} \tag{1}$$

The second method consists of a stochastic transformation according to the following formula:

$$w_b = \begin{cases} +1 \text{ with probability,} & p = \sigma(w) \\ -1 \text{ with probability,} & 1 - p \end{cases} \tag{2}$$

where $\sigma(x) = \max(0, \min(1, (x+1)/2))$. This method has regularization effects, as training a CNN on CIFAR-10 with binary weights gives an error of 9.90% for the deterministic transformation, 8.27% for the stochastic transformation, and 10.64% when no regularizer is applied [59].

Hubara et al. [60] propose an extension to the previous method, as they apply binary values for activations as well, using the same formulas [see 1 and 2]. Both methods use binary values for forward propagation and backward propagation, but a copy of the full-precision weights is kept for the update. Switching from 32 bits to binary precision results in a 32× smaller model with 32× less memory accesses, which can lead to a large reduction in the energy consumption [60].

Li et al. [61] use ternary weights (−1, 0, 1) during the forward and backward propagation phases. As for the binary methods, a copy of the full-precision weights is kept for the update. Tests on MNIST [62] and CIFAR-10 [33] show that ternary weights give close performances to the full-precision parameters. On CIFAR-10 for example, the ternary weights provide a test accuracy of 92.56% against 92.88% for full-precision weights. Ternary parameters perform worse on a bigger dataset as ImageNet [30] but are still better than binary weights.

Zhou et al. [63] study the impact of weights, activations, and gradients quantization by testing different values for the precision on the Street View House Numbers (SVHN) dataset [64]. They find that quantization reduces resource consumption but affects the accuracy. To limit this impact, they suggest representing the gradients on more than 4 bits. If we note the number of bits used for weights, activations, and gradients, respectively, W, A, and G then the authors advise to take: $W < A < G$, a rule defined according to the sensitivity of each element.

Fu et al. [65] explain that a low precision implies a high quantization noise, which favors the exploration of the space, while a high precision allows more relevant updates. To allow a tradeoff between the two, it is interesting to use a dynamic precision that can take a cyclic form, as in [65].

Quantization is considered as one of the most important and effective techniques to reduce energy consumption [65]. It is, therefore, an extensively studied approach. We review some works within this section, and other references can be found in [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], and [76].

C. Reducing the Number of Multiplications

Multiplication is the key and the dominant operation of DNNs. It is also the most costly [77]. To offer energy gains, some works propose to reduce the number of multiplications and to replace them by a more efficient operation. This approach can be seen as a subclass of quantization, but we present it in a separate category since it does not reduce the parameter's precision as quantization methods would do. Instead, this approach acts on the operations composing the DNNs and applies less costly variants. We included the reduction of multiplication's number in the compression methods since, just like the previous strategies of this category, it acts on existing architectures with a good prediction quality and reduces their costs.

Multiplications can be replaced by additions [78] or shifting [79]. In the AdderNet model [78], the convolution is replaced by the L_1 norm. Using the L_1 norm is more efficient as it only involves additions. The experiments on CIFAR-10/100 with the VGG, ResNet20, and ResNet32 architectures show that AdderNet achieves a very close accuracy to a multiplication-based CNN (less than 1% loss). The ShiftAddNet model [77] is inspired by the practices used in hardware platforms, where multiplication is implemented with additions and shifts of bits. The structure of ShiftAddNet consists, first, in applying a shift layer followed by an add layer that applies the L_1 norm as in the previous paper [78]. The method is compared to AdderNet [78] and a multiplication-based CNN [45]. ShiftAddNet reduces the energy consumption during training by 33.7% and 44.6% compared, respectively, to AdderNet and a multiplication-based CNN.

D. Knowledge Distillation

In general, complex and large models lead to better performances, while smaller architectures are more resource-efficient but limited in terms of prediction quality. The goal of knowledge distillation is to combine the advantages of the two cases. To do so, it guides the training of the small model (called student) by a complex network (called teacher) by integrating information from the teacher in the student's learning.

It is important to differentiate the concept of knowledge distillation from the one of transfer learning [80], as they have two separate goals and act differently. On the one hand, knowledge distillation aims to reduce the performance gap between a small model and a larger one on the same task by transferring information on the large network generalization. On the other hand, transfer learning aims to reduce the training time of a network solving a given task by initializing its parameters with the ones of a pretrained model on another similar task [81].

The key point of knowledge distillation is how the teacher's knowledge is transferred to the student. Hinton et al. [82] propose to modify the objective function of the student model. The latter will thus learn from the classical labels of the dataset and from the softmax distribution constructed by the teacher model during its training, which is commonly referred to as the soft labels. This scheme is commonly used as in [83], where it is combined with quantization. Other strategies were proposed for the knowledge transfer. Yim et al. [84] define the distilled knowledge in terms of the flow between two layers and compute it using the inner product between features from two layers. Lopes et al. [85] propose a data-free knowledge distillation method based on activation summaries to avoid accessing the original dataset. Since similar data have similar activations in a trained model, the knowledge can be transferred by penalizing the student when its activations differ from the teacher ones [86]. The work in [87] uses the adversarial samples found on the teacher to train the student, while the one in [88] relies on weighted soft labels in order to accomplish a bias–variance tradeoff.

Yang et al. [89] apply the teacher–student framework in M generations, where the student of generation i becomes the teacher of generation $i + 1$. The students are trained with a modified loss function to integrate the teacher's soft labels. To improve student performances, they also modify the loss function of the teacher to allow the emergence of secondary classes. This makes the teacher "more-tolerant" and leads to better students.

The knowledge distillation approach imposes to pretrain the teacher, which might be costly. Self-distillation [90], [91], [92] emerged as a solution to this drawback. This approach omits the teacher network and makes the student supervise itself. Among the possible ways to achieve this, Zhang et al. [90] propose to divide a CNN into shallow sections. For instance, the sections of a ResNet model [3] would be its different ResBlocks. A classifier, a bottleneck, and a fully connected layer are then added to every ResBlock. Each shallow section is considered as a submodel during training. We can see this strategy as simulating student models through the same network. The deepest section acts as the teacher and transfers knowledge to the previous ones.

The work studied in the context of knowledge distillation did not carry out energy measurements. Nevertheless, we consider that this method reduces energy consumption thanks to its compression power. Indeed, knowledge distillation allows a compressed and therefore less expensive model to reach performances that it would not obtain without being guided by the teacher model. The benefits of this approach are obvious during the inference phase since we deploy a compressed student model. The approach can bring benefits to training if the teacher is pretrained, which is always the case in offline knowledge distillation. In contrast, the online knowledge distillation trains both the teacher and the student simultaneously [93]. We did not include approaches from this type in our survey, as they only bring benefits to inference. For more details on the online variant, interested readers can refer to [93].

IV. Architecture Design Optimization

Instead of optimizing a model only in terms of accuracy and then compressing it as the approaches of the first class usually do, the architecture design optimization category aims to build an architecture that makes a tradeoff between prediction performance and resource consumption. It, therefore, acts during the design phase and is composed of two groups of methods that we detail in the following.

A. Manual Design

This class gathers handcrafted architectures designed for mobile and resource-limited platforms. Such architectures are built with optimized blocks. They can follow design principles aiming to reduce resource consumption. We discuss these principles and some well-known architectures in the following.

The crossing point between the last convolution layer and the first fully connected layer of a CNN has a high memory consumption. Lin et al. [94] propose two solutions to reduce it. The first is to use global average pooling that averages the features per channel to reduce their number. The second uses a convolution with a 1×1 filter, as it allows to control and specifically reduce the number of channels. This second principle is particularly used in the models we discuss in the rest of this section.

The MobileNet architecture [95] is one of the most well-known optimized models. It is based on the depth-wise separable convolution, which is an optimized variant of the standard convolution. It first applies a $k \times k$ convolution on each channel independently of the others, before gathering the results with a 1×1 convolution.

The depth-wise separable convolution is also a key operation in the MobileNetV2 architecture [96]. MobileNetV2 contains skip connections as in the ResNet models, but it uses inverted residuals. Instead of reducing the number of channels, performing a 3×3 convolution and then restoring the original number of channels as in a usual residual block, in MobileNetV2, the number of channels is increased before applying a 3×3 depth-wise separable convolution and restoring the number of channels. Indeed, the depth-wise convolution already reduces the number of parameters, which allows the usage of an opposite approach to the usual residuals. MobileNetV2 also introduces the concept of linear bottlenecks, in which the last convolution of the residual block has a linear output instead of using the rectified linear unit (ReLU) nonlinearity. It improves performances as it avoids the information loss caused by nonlinearities [96].

The SqueezeNet architecture [97] is based on fire modules that contain a squeeze layer followed by an expand layer. The squeeze layer contains only 1×1 convolutions and is used to reduce the number of feature maps before feeding the inputs into the expand layer, which contains 3×3 and 1×1 convolutions. SqueezeNet maximizes the usage of 1×1 filters since it has $9\times$ fewer parameters than 3×3 ones [97].

B. Automatic Design

The manual design of an efficient architecture can be a challenging task. It is, therefore, interesting to automate this process through approaches such as NAS or growing neural network techniques that we describe in the following.

1) Neural Architecture Search: The NAS automates the design of a neural network as well as hyperparameters' tuning. NAS can be broken down into three parts [18].

1) Search space defines the set of allowed operations in the architecture to be built (e.g., the allowed size of the filters).
2) Search algorithm used to explore the search space.
3) Evaluation strategy defines how to evaluate the designed architecture. It usually includes metrics such as the model's accuracy and error.

Zoph and Le [98] proposed one of the most famous NAS frameworks. The latter consists of a controller [a recurrent neural network, (RNN)] that defines the architecture by setting some parameters. This controller is trained using reinforcement learning. While NAS was originally intended to automate the construction of architectures, many recent works use it to build models that meet resource constraints by integrating efficiency metrics in the evaluation strategy [99], [100].

Gong et al. [101] target energy efficiency by combining NAS and quantization. This method can be seen as a controller that interacts with the task environment and the hardware environment [101]. The controller generates combinations of architectures that minimize error while not exceeding an energy threshold. The resulting architecture is a stack of MobileNetV2 [96] blocks. To integrate quantization, the search process includes weight precision choices (2, 4, 6, or 8 bits). This method reduces the inference energy consumption by 63% when compared to 8 bits—MobileNetV2 while maintaining comparable accuracy [101]. Instead of quantization, NAS can be combined with pruning [102].

Dai et al. [103] use a genetic algorithm to adapt a base model to a scenario imposing latency or energy constraints. The genetic algorithm provides candidate models that fit the scenario better and better. To measure the fitness of these candidate models, it is necessary to estimate their energy consumption, latency, and accuracy. The candidates with the best fitness will be chosen to undergo crossover and mutation in order to build the next generation. As estimating the previous metrics through training or by performing hardware measurements would be too slow, predictors were used to evaluate the fitness. The accuracy and energy predictors are built using a Gaussian process regression, while the latency predictor is built with a lookup table.

MobileNetV3 [104] was designed using NAS for block-wise search and the NetAdapt algorithm [105] for the layer-wise search. NetAdapt is used as a complementary approach to NAS in order to fine-tune layers individually [104]. MobileNetV3 also includes other improvements such as a redesign of costly layers at the beginning and the end of the network, as well as a new and faster nonlinear function [104].

ShiftAddNas [106] is a NAS algorithm with a hybrid search space. It builds architectures, including multiplication-based operations (powerful but costly) and multiplication-free operations (less powerful but hardware-efficient) as additions and shifting. This hybrid search space would thus allow the accuracy/efficiency tradeoff.

Because of its costly search algorithm, the NAS technique is more suited for inference optimization. It can, however, lead to architectures that are more efficient to train than handcrafted ones, as they contain fewer parameters and require fewer computations [107].

2) Growing Neural Networks: The usual training procedure consists in fixing an architecture and searching for the best parameters. The growing technique [108], [109], [110], [111], [112] proposes a different paradigm, as it aims to jointly optimize the architecture and its weights during training [111]. It starts with a small (seed) model and increases its size progressively during training by adding neurons. This avoids the usage of a large model, from the start, as pruning or NAS methods do. A growing algorithm answers three main questions [111]: when to add neurons? where to add neurons? (in existing layers or by creating new ones), and how to initialize new neurons?

The new neurons can be added randomly or according to a more complex strategy. Wu et al. [108] alternate between usual gradient descent updates and growing the model by splitting neurons. Yuan et al. [110] adjust a seed architecture by combining growing and pruning, which allows to reduce the training and inference FLOPS. The GradMAX algorithm [111] keeps the output of the network unchanged when adding new neurons, which is a desirable property for growing algorithms. To do so, it sets the incoming weights to zero. The outgoing weights of the new neurons are set as the top-K left singular vectors since it allows maximizing the gradient norm. Indeed, the larger the gradient norm, the larger the loss decrease [111]. The work in [112] provides a method to add new neurons by locating expressivity bottlenecks. It takes neuron redundancy into account, unlike some previous works.

V. Data Selection

The growing size of the datasets is one of the main reasons behind the high resource consumption of DNNs. Reducing the number of training examples through data selection is thus a way to enhance training efficiency. Data selection has been quite studied in the literature. Indeed, the dataset contains examples with varying difficulty and importance. Some examples are thus critical for learning, while others can be removed with little impact on accuracy. Data selection is mainly used in order to accelerate the training process, and this results in a lower energy consumption. In the following, we explore adaptive, nonadaptive, and assistant-based data selection. We also discuss the choice of the percentage of data to keep (selectivity ratio) as well as random selection, a simple yet effective technique.

A. Nonadaptive Data Selection

This type of data selection uses a fixed subset of examples for training. It can be considered as a preprocessing step where we select the subset before training. In order to perform the selection, it is necessary to evaluate the importance of examples. Paul et al. [113] propose two scores to quantify this importance. The first score is computed as the expected loss gradient norm and can be used to select examples at initialization, while the second is computed as the norm of the error vector and is a good indicator of the examples' importance after a few epochs. Another criterion is the forgetting statistics proposed in [114]. Forgetting an example means that the model classifies it correctly at a given time and then misclassifies it later during training. Removing least forgotten examples does not hurt performance, which proves that such samples are not essential for learning [114]. This selection criterion might be effective, but it requires training on the entire dataset to collect the forgetting statistics. Indeed, this approach in particular and nonadaptive data selection in general requires prior knowledge of examples' importance. To obtain this information, it is sometimes necessary to train the model on the entire dataset, for some or all epochs, which imposes an important selection cost. This is an important drawback of nonadaptive data selection. Once the subset is chosen, it is no longer updated, and hence, there is no additional cost during the training phase, but the preprocessing step (when selection is performed) can be very costly, especially for large models or large datasets. One solution to this problem is to use a smaller model, named proxy, to select the training subset of a larger model (that we want to train) as proposed in [115]. This work shows that models of different sizes select similar examples. However, the training of the proxy itself can remain a costly procedure.

B. Adaptive Data Selection

This type of data selection renews the subset of selected examples periodically during training. The main advantage is that it allows to adjust the chosen examples to the current prediction level of the model. The most obvious metric to measure this level is the loss. For instance, the "KAKURENBO" paper [116] proposes to select the fraction F of examples with the highest losses. In addition, they include in the selected subset, the misclassified examples, as well as the ones with a low prediction confidence. Jiang et al. [117] propose a selection strategy based on losses' history. They aim to reduce the number of backward phases by only applying them on examples with a high error. Instead of the loss, some works [118], [119], [120] aim to select the subset that matches the full gradient (sum of the gradients of all the training set). Yang et al. [120] further reduce the selection costs and improve the efficiency by removing the "already learned" examples from the training set. Such examples maintain a very small loss and a gradient close to zero during multiple iterations.

Yao et al. [121] explain that using one metric to select data is not enough. Therefore, they design a mixture of metrics to adapt the selection process to the model evolution all along training. At the beginning of the process, diversity is needed, and therefore, examples should be selected randomly. By the middle of training, the model needs to converge quickly, and thus, the metrics to consider should be the gradient (to focus on examples generating a large update) and prediction (correct or wrong prediction) ones. By the end of the training, the model goes into a stable phase and should focus on hard examples. Therefore, the selection should rely on loss and entropy criteria.

C. Automated Adaptive Data Selection

The techniques that we described earlier perform adaptive data selection according to a given criterion, generally meant to measure the importance of every example. A line of work proposes a more sophisticated approach that relies on the usage of an assistant/proxy model that will select the examples of the base model (the model to originally train). In this context, Zhang et al. [122] propose AutoAssist, a framework where a lightweight assistant model is used as a batch generator for the base model training. The assistant aims to filter the examples by computing a utility function that estimates the loss of the base model. The assistant is a shallower model (example: a regression) trained jointly with the base model to provide a better approximation of its losses, as training progresses. Fan et al. [123] propose an approach where two agents interact. The first agent is the student, the base model to be trained, while the second is called the teacher, which will determine the appropriate training examples. The teacher is trained using reinforcement learning where the state is defined by the mini-batch of data and the student model, and the action is to whether or not select an example. A trained teacher can be reused with similar students and configurations. Some approaches [124], [125] use such assistant models with curriculum learning [126], which consists in providing the examples in a specific order according to their difficulty. Although curriculum learning is a different approach, it has an important common point with data selection: quantifying the difficulty of examples. Hence, techniques proposed in the context of curriculum learning can be adapted to data selection.

D. How Much Data Should Be Kept?

An important aspect of the data selection process is to define how many examples to keep. Indeed, when the number of examples to keep decreases, the energy gains increase, but the impact on accuracy is heavier [127]. Therefore, it is necessary to find the appropriate percentage of examples to keep in order to obtain the best tradeoff. Our previous study [127] shows that this percentage is related to the difficulty of the dataset. For an easy task as CIFAR-10, keeping only 20% of data is enough to obtain high accuracy, while on CIFAR-100, it is better to keep 50% of the examples with the considered selection strategies [127]. The same study shows the benefits of using a decreasing percentage. Indeed, using larger subsets at the beginning of training and smaller ones later allows to reach a high accuracy with significant energy gains and accelerates convergence [127]. Yao et al. [121] provide similar conclusions. Parallel works on curriculum learning also highlight the benefits of a dynamic (increasing, decreasing, and even cyclical) training set size, even with a random order of examples [128], [129].

E. Random Data Selection: Is It That Naive?

Most approaches rely on specific importance criteria to perform data selection, while the random strategy is seen as a lower bound. However, a line of work has shown that random selection can be quite effective when applied in the correct setting. Indeed, Wang et al. [130] show that a 50% random batch skipping reduces the training costs by half without a significant accuracy loss. As mentioned earlier, Yao et al. [121] use the random metric at the beginning of training, as the model needs diversity in this phase. Our study [127] shows that the random selection provides a close and sometimes a better test accuracy when compared with a loss-based technique [117]. This work explains that the effectiveness of the random approach is due to the frequent updates of the selected subset. Indeed, the model needs to go through all examples multiple times during the training process, but it is not necessary to consider all the examples at every epoch. Using the random criterion in a nonadaptive scheme leads to poor results [127]. Okanovic et al. [131] compare the random sampling of a subset at each epoch to state-of-the-art data selection techniques and find that it leads to a better time-to-accuracy. Therefore, the usage of random adaptive data selection should be considered as a serious option, especially since it has a very low selection overhead when other criteria require further computations and might limit the gains of adaptive data selection.

VI. Infrastructure Optimization

This class gathers the efforts made on the hardware platform design and the software in order to optimize the computation of the DNNs. Indeed, it is necessary to provide an infrastructure to run the algorithms of the previous classes.

A. Hardware Optimization

The hardware platforms used for DNNs can be divided into temporal and spatial architectures [16]. The temporal architectures include processors (CPUs) and graphics processing units (GPUs). The growing interest toward DNNs made the hardware constructors integrate new features in CPUs/GPUs in order to optimize neural network computing. For instance, constructors provide libraries to optimize computation as the Intel Math Kernel Library (MKL) [132] or Nvidia cuDNN [133]. Nvidia also designed the tensor cores, specialized cores, that support mixed precision in order to improve training and inference speed [18].

Despite the efforts made to adapt or design platforms to optimize DNN computing, the temporal architectures do not necessarily provide energy gains. The spatial architectures are more suited for this goal as they consist of accelerators with many levels of memories, with different sizes. Larger memories are the most energy consuming. Accelerators help to optimize energy consumption since they can introduce specific data flows that maximize the reuse of data when it is in the least consuming memory [16]. The designing techniques of such platforms are outside the scope of this article, but examples of the latter can be found in [31], [134], [135], [136], [137], [138], [139], [140], [141], [142], and [143]. Interested readers may also refer to the following survey [144].

B. Framework Optimization

Usual deep learning frameworks, such as Pytorch [145] and TensorFlow [146], have a high memory usage and are not suited for resource-limited devices. Thus, lighter versions, namely, Pytorch Mobile [147] and TensorFlow lite [148], have been designed but remain more suitable for inference.

Some works propose compute engines built from scratch. Lin et al. [149] provide a training engine along with quantization and sparse update algorithms. Tang et al. [14] propose an inference engine for the SqueezeNet architecture [97]. They use the Advanced RISC Machines (ARM) library ACL to implement the basic operations of CNNs. They compare their engine built from scratch to the migration of TensorFlow to an ARM SoC. This migration can be complex because of the TensorFlow dependencies. Their engine avoids this problem and reduces the time to process an image from 420 to 320 ms. Building custom engines remains a complex procedure, only practical when dealing with simple network architectures.

Patil et al. [150] perform a system-level optimization of memory and energy consumption by designing a compiler that integrates two techniques: rematerialization and paging. Rematerialization consists in deleting an activation tensor as soon as it is no longer needed. Paging consists in transferring a tensor from primary to secondary memory when it is not used. Rematerialization needs to recompute the deleted data when it is needed again, and therefore, it has to be associated with low-cost layers. In contrast, paging is an interesting alternative when the data is used in complex calculations, to avoid recomputing it. The compiler built in [150] considers these characteristics and also imposes energy constraints to avoid any overhead that might come from the two techniques.

VII. Energy Consumption Evaluation

In order to assess the energy efficiency of DNNs, it is crucial to evaluate their power consumption. Before delving into the used energy assessment approaches, it is worth noting the lack of standardized benchmark tasks and datasets for this evaluation. Most works addressing efficiency consider computer vision tasks as image classification and object detection for their experiments. Benchmark datasets (such as CIFAR-10/100 [33], ImageNet [30], and COCO [151]) and architectures (such as ResNet [3] and VGG [12]) are commonly used. Other tasks, such as speech recognition and text classification, are considered in some works [115].

Regarding energy efficiency assessment, various techniques are used in the literature. Some rely on energy measurements, while others consider proxy metrics to estimate power consumption.

Among the energy measurement techniques, the most intuitive one is to use a power meter. This equipment is external and not integrated into the computing node [152]. The works in [47] and [130] used such a device for their energy measurements.

Many hardware constructors provide technologies to measure the power consumption of their devices [152]. For instance, Intel proposes the running average power limit (RAPL) technology that estimates power and energy consumption through a software power model [153]. Nvidia provides the NVidia Management Library (NVML), a C-based API to monitor different metrics, including power consumption of Nvidia GPUs [154]. For the Tegra-based devices that do not support the NVML technology, Nvidia provides a similar tool through Tegrastats [155]. Many Python libraries were developed to interact with the previous technologies and facilitate their usage. For instance, PyRAPL [156] allows getting the measurements obtained through the RAPL technology by only adding some decorators in a python code. PyJoules [157] follows the same concept but uses both RAPL and NVML and thus allows to monitor the energy consumption of both the CPU and the GPU. PyJoules is used in the work of [119]. Energy usage [158] and codecarbon [159] use the previous technologies to measure the energy consumption but also provide the estimated CO_2 emission based on the location.

Using a software solution is the easiest and most user-friendly option. However, all these tools rely on specific technologies and, thus, only work on a limited number of hardware platforms, generally restricted to Intel CPUs and Nvidia GPUs. Some tools, such as the codecarbon library, propose to use a constant thermal design power (TDP) to estimate power when the RAPL and NVML technologies are not available, but this solution is not very accurate. This leads some authors [29] to design their own measurement strategy.

Instead of the techniques described above, some works rely on proxy metrics to assess energy efficiency. For instance, many studies consider the number of floating operations (FLOPs) or sometimes the number of multiply-and-accumulates (MACs). FLOPs and MACs are not always a good indicator of energy consumption, as they fail to capture some factors as memory accesses [27].

To sum up, according to our literature review, there is no unified procedure for energy evaluation, as the considered metrics and the assessment tools vary from one work to another. We still believe that designing a unified evaluation procedure, based on energy measurements, is a key step toward more energy-efficient deep learning.

VIII. Summary and Discussion

In this section, we will summarize the different approaches that we described before discussing specific aspects such as the impact of the task on the effectiveness of the techniques, some additional research directions such as neuromorphic and reservoir computing (RC), the algorithmic and hardware co-design, and finally some limitations of the existing work.

A. Classification Summary

We recall that we proposed a classification of four categories. The first one gathers compression methods that reduce the size or the computations of an existing model to lower its costs. These techniques are more suited for overparameterized architectures, for which they have a limited impact on accuracy. The second class follows a somewhat opposite direction to the one of the compression methods. Indeed, it seeks to build optimized architectures at the design stage. This second class thus includes optimized handcrafted architectures as well as automated design techniques such as NAS or growing. The third class targets the data level and aims to lower the energy consumption by training the model on a subset of the dataset. Its challenge is to find the right strategy to select data. The last class gathers the efforts made on the hardware and software levels to optimize DNNs' computations.

We compare the different techniques in Table I in terms of the impact phase (training/inference), when the approach is applied and the level it acts on. Indeed, it is possible to apply changes to the architecture, the objective function, the data, or the hardware platform. The moment where the approach is applied impacts which phase is optimized. If the approach is applied before or during training, the benefits can be seen in both phases, whereas applying the approach after training only brings benefits to inference. This observation does not hold for data selection, which is exclusively oriented toward training optimization.

B. Dependence on the Task and Hyperparameters

The effectiveness of every technique depends on its own hyperparameters. Indeed, keeping a high number of examples when applying data selection prevents an important accuracy decrease but leads to limited energy gains, as shown in our previous work [127]. For the growing techniques, the size of the resulting architecture and how fast we get to this size during training can lower energy savings [160].

The task to perform also has an important impact. Data selection works better on dataset with many redundancies, as this allows to use less examples to maximize energy gains, without significantly impacting prediction quality [127]. Our study [160] shows that data selection, pruning, and growing lead to a higher accuracy decrease on a harder

TABLE I
GLOBAL COMPARISON BETWEEN CLASSES

Criterion	Compression methods				Architecture design optimization			Data selection	
	Pruning	Quantization	Reducing multiplication number	Knowledge distillation	Manuel design	NAS	Growing NN	Adaptive	Non-adaptive
Inference/ training benefits	Both	Both	Both	Both	Both	Inference	Training	Training	Training
When is it applied	Before/ During/After Training	During/After Training	During/After Training	During Training	Before Training	During Training	During Training	During Training	Before Training
Action Level	Model	Calculation	Calculation/ Operations	Objective function	Model	Model	Model	Data	Data

dataset as CIFAR-100 when compared to CIFAR-10. The architecture also impacts the effectiveness of some techniques. For instance, structured pruning works better on larger models, while its impact on accuracy is significant on smaller architectures [160].

C. More Efficient = More Brain-Inspired?

It is well known that neural networks draw inspiration from the human brain. However, they are characterized by single, static, and continuous activations [161], which is different from brain functioning. Spiking neural networks (SNNs), the third generation of neural networks, are considered as closer variants to the biological neurons.

A SNN architecture is a set of neurons interconnected by synapses [162] where the information is encoded in a spike form and is propagated from presynaptic to postsynaptic neurons [162]. In biological neurons, the postsynaptic node generates a spike when its membrane potential exceeds a threshold. This membrane potential is modified according to the synaptic current sent by the presynaptic neuron [161], [162]. These dynamics can be modeled using different strategies, the most popular being the leaky integrate-and-fire (LIF) [162]. The integrate mechanism consists for the neuron to sum its inputs and integrate them over time. When the integrated signal exceeds a threshold, the neuron fires and generates a spike [163].

The activation function of LIF neurons is nondifferentiable [162]. This makes the learning of SNNs challenging as the most common algorithm, the backpropagation, cannot be directly applied. Kim and Panda [164] enumerate three ways to train SNNs. The first is the spike-timing-dependent plasticity (STDP) algorithm [165]. It is based on the correlation between presynaptic and postsynaptic spikes, but its lake of a global optimization rule makes it only suitable for small networks [164]. The second way to enable SNN learning is to convert a conventional neural network into a spiking model [166], [167]. Despite being nondifferentiable, training SNNs using backpropagation is still studied in some works [168], [169].

Despite the challenges faced in SNN training, the interest toward these models has grown because of their high energy efficiency. Indeed, SNNs are event-driven, which implies that energy is only consumed when a spike occurs. SNNs' efficiency is also highly related to their implementation on neuromorphic hardware [164]. The latter are brain-inspired hardware platforms, known for being energy-efficient as they avoid the von Neumann bottleneck [163]. This bottleneck comes from the intensive and power-hungry exchange of data between the memory and the computing unit, as they are separated in von Neumann computers [163]. A solution to this problem is the usage of in-memory computing, where the computation is done within memory [170]. Neuromorphic computing also includes the usage of specific and novel materials to improve the efficiency of traditional CMOS solutions [163]. For instance, the memory resistor or memristor is commonly used, especially since it can change its conductance according to electrical pulses [163].

Some neuromorphic platforms are specifically built for SNNs because of their increasing popularity. For instance, Intel provides the Loihi platform, a 60-mm^2 chip [171], which includes 128 neuromorphic cores, 3 $\times$ 86 cores, and off-chip communication interfaces [171]. It uses an asynchronous network-on-chip (NoC) to transport communications, notably the spike messages required by SNNs [171]. SpiNNaker [172] is an ARM-based system for SNNs. It can model a billion neurons and up to a trillion synapses [172]. Other chips, such as [173] and [174], were designed for neuroscience-based models as SNNs.

SNNs and neuromorphic computing are not the central topic of this article that focuses on the optimization of conventional neural networks. They remain promising and important paradigms to enhance the energy efficiency of artificial intelligence techniques. Interested readers may refer to [162], [163], and [175] for additional details.

D. Reservoir Computing

RC is a bioinspired RNN. It is notable for its simplicity and suitability for time-series prediction or speech and text processing tasks [176], [177].

In RC, the network contains two elements. The first is the reservoir, a set of neurons initialized randomly and then fixed and not trained [176]. The second is the readout that can be assimilated to a trainable output layer [176]. RC relies on two main implementations: echo-state networks (ESNs) [178] and liquid-state machines (LSMs) [179].

Using fixed weights for the reservoir can sometimes limit performances, especially when processing different timescales [176], [180]. One solution is to use multiple reservoirs in parallel or to stack them [177], [181].

RC is usually considered as more energy-efficient than traditional recurrent networks, especially as its reservoir part is fixed, and only the readout is trained. Also, RC can be implemented on field-programmable gate array (FPGA) platforms [182], [183] or on neuromorphic circuits [184], [185], [186], which can further improve the efficiency.

E. Hardware and Algorithmic Co-Design

The algorithmic approaches outlined in this study necessitate varying degrees of support from the hardware layer to ensure an efficient execution. Specifically, unstructured pruning requires specific accelerators to take advantage of the sparsity it generates, for instance, by skipping multiplications with zeros [20]. Quantization also needs hardware support for low-precision computations, provided, for example, by Nvidia GPUs (through the tensor cores) and mobile ARM CPUs [20]. Another example that highlights the importance of combining hardware and software solutions is the self-organized maps (SOMs). SOMs are an unsupervised model that realizes a nonlinear mapping from the high-dimensional space of the inputs to a low-dimensional map [187]. SOMs can perform tasks such as clustering and dimensionality reduction and can be useful for the IoT applications [187], [188], [189]. Hardware implementations of SOMs are more efficient, especially because of the high parallelism of the algorithm. They are, therefore, often implemented on FPGAs and application-specific integrated circuits (ASICs) [187].

However, this perspective remains a combination of solutions and separates the hardware design from the algorithmic design and leaves a whole joint-design area unexplored. As a solution, a body of work [190], [191], [192] presents co-design approaches where the algorithm and hardware are jointly optimized.

Zhou et al. [191] propose neural architecture and accelerator search (NaaS), a framework for both NAS and hardware search. They target edge tensor processing unit (TPU) platforms known for being highly parameterized accelerators. Shi et al. [190] use a Bayesian optimization framework for the co-design of the hardware accelerator and the compiler that maps the model into the hardware. Abdelfattah et al. [192] use NAS to find the best model–accelerator pair for the energy-efficiency tradeoff and study three reinforcement learning search techniques. Some works [193], [194] focus on the co-design and co-exploration of network architectures and in-memory computing hardware, particularly known for its efficiency.

F. What Is Missing?

There are many encouraging results on enhancing DNNs' efficiency in general and training efficiency in particular. We, however, want to highlight an important limit of the existing works. Many papers assume or mention gains in energy consumption without properly measuring it. Alternative metrics, such as the training time, the size of the model, and the number of FLOPs, are used instead. However, as mentioned in Section VII, such proxies do not always reflect the energy consumption. Because quantifying a goal is the first step toward achieving it, we believe that it is necessary to measure or at least estimate the energy consumption of any approach proposed to improve the efficiency. Moreover, is it necessary to design a unified procedure to allow a fair comparison between different works.

Also, we want to highlight the fact that some approaches are much more studied than others. For instance, the compression methods are very popular, while growing is less explored in the context of energy efficiency. It would be interesting to direct research toward new approaches, especially since techniques, such as unstructured pruning, are very popular, while it does not provide direct energy savings without additional optimization or specific hardware for sparsity.

IX. Conclusion

In this article, we reviewed the important methods used to improve DNNs' energy efficiency. We proposed a classification and described various techniques while focusing on training optimization. Despite the difficulty of this challenge, the current results are quite promising. We notice that some approaches, such as compression ones, are widely studied, while other techniques are less investigated. For instance, the growing methods can be promising, but their impact on energy remains unexplored. Also, data selection is rarely applied to save energy directly, while it can be competitive to reach this goal. Moreover, it remains necessary to build or agree on a unified methodology for the energy measurements in order to really quantify the impact of the different techniques on energy and to allow a fair comparison between approaches.

Alongside with conventional neural networks, different paradigms, such as RC and SNNs, are emerging as promising solutions. These techniques are not always competitive with conventional neural networks on some tasks but are still getting more and more attention. This is because of their more concrete brain inspiration and suitability to neuromorphic computing, which is in turn an important alternative to overcome current hardware bottlenecks and move toward more energy-efficient solutions.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1097–1105.

[2] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Oct. 2015, pp. 1–9.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[4] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, "Medical image analysis using convolutional neural networks: A review," *J. Med. Syst.*, vol. 42, no. 11, pp. 1–13, Nov. 2018.

[5] B. T. Nugraha and S.-F. Su, "Towards self-driving car using convolutional neural network and road lane detector," in *Proc. 2nd Int. Conf. Automat. Cogn. Sci. Opt. Micro Electro-Mech. Syst. Inf. Technol. (ICACOMIT)*, 2017, pp. 65–69.

[6] A. K. Jain, "Working model of self-driving car using convolutional neural network, raspberry pi and Arduino," in *Proc. 2nd Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, Mar. 2018, pp. 1630–1635.

[7] A. Vaswani, "Attention is all you need," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 6000–6010.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[9] T. B. Brown et al., "Language models are few-shot learners," in *Proc. NIPS*, 2020, pp. 1877–1901.

[10] *GPT-4 Technical Report*, OpenAI, San Francisco, CA, USA, 2023.

[11] Y. Liu et al., "Summary of ChatGPT-related research and perspective towards the future of large language models," 2023, *arXiv:2304.01852*.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[13] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan. 2018.

[14] J. Tang, D. Sun, S. Liu, and J.-L. Gaudiot, "Enabling deep learning on IoT devices," *Computer*, vol. 50, no. 10, pp. 92–96, Oct. 2017.

[15] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," 2019, *arXiv:1906.02243*.

[16] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[17] W. Roth et al., "Resource-efficient neural networks for embedded systems," 2020, *arXiv:2001.03048*.

[18] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Comput. Surveys*, vol. 55, no. 12, pp. 1–37, 2023.

[19] A. Berthelier, T. Chateau, S. Duffner, C. Garcia, and C. Blanc, "Deep model compression and architecture optimization for embedded systems: A survey," *J. Signal Process. Syst.*, vol. 93, pp. 863–878, 2021.

[20] H. Cai et al., "Enable deep learning on mobile devices: Methods, systems, and applications," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 3, pp. 1–50, May 2022.

[21] D. Liu, H. Kong, X. Luo, W. Liu, and R. Subramaniam, "Bringing AI to edge: From deep learning's perspective," *Neurocomputing*, vol. 485, pp. 297–320, May 2022.

[22] Z. Jie, P. Sun, X. Li, J. Feng, and W. Liu, "Anytime recognition with routing convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 6, pp. 1875–1886, Jun. 2021.

[23] X. Li et al., "Predictive exit: Prediction of fine-grained early exits for computation-and energy-efficient inference," in *Proc. AAAI Conf. Artif. Intell.*, 2023, vol. 37, no. 7, pp. 8657–8665.

[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[25] J. Lee et al., "Resource-efficient convolutional networks: A survey on model-, arithmetic-, and implementation-level techniques," *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–36, Jul. 2023.

[26] V. Mehlin, S. Schacht, and C. Lanquillon, "Towards energy-efficient deep learning: An overview of energy-efficient approaches along the deep learning lifecycle," 2023, *arXiv:2303.01980*.

[27] B. R. Bartoldson, B. Kailkhura, and D. Blalock, "Compute-efficient deep learning: Algorithmic trends and opportunities," *J. Mach. Learn. Res.*, vol. 24, no. 122, pp. 1–77, 2023.

[28] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 1990, pp. 598–605.

[29] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 5687–5695.

[30] O. Russakovsky et al., "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[31] Y.-H. Chen, T. Krishina, J.-S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Nov. 2016.

[32] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[33] A. Krizhevsky, V. Nair, and G. Hinton. (2017). *The CIFAR-10 Dataset (2014)*. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[34] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 29, 2016, pp. 2082–2090.

[35] A. Berthelier, Y. Yan, T. Chateau, C. Blanc, S. Duffner, and C. Garcia, "Learning sparse filters in deep convolutional neural networks with a L_1/L_2 pseudo-norm," in *Proc. Int. Conf. Pattern Recognit.*, 2021, pp. 662–676.

[36] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2736–2744.

[37] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, Oct. 2017, pp. 1389–1397.

[38] B. Mussay, D. Feldman, S. Zhou, V. Braverman, and M. Osadchy, "Data-independent structured pruning of neural networks via coresets," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 7829–7841, Dec. 2022.

[39] S. Lin, R. Ji, Y. Li, C. Deng, and X. Li, "Toward compact ConvNets via structure-sparsity regularized filter pruning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 2, pp. 574–588, Feb. 2020.

[40] N. Lee, T. Ajanthan, and P. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[41] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020. [Online]. Available: https://openreview.net/forum?id=SkgsACVKPH

[42] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 6377–6389.

[43] S. Liu et al., "The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[44] J. Frankle, G. K. Dziugaite, D. Roy, and M. Carbin, "Pruning neural networks at initialization: Why are we missing the mark?" in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[45] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[46] A. Morcos, H. Yu, M. Paganini, and Y. Tian, "One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 4932–4942.

[47] H. You et al., "Drawing early-bird tickets: Toward more efficient training of deep networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 630–645.

[49] S. Hayou, J.-F. Ton, A. Doucet, and Y. W. Teh, "Robust pruning at initialization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[50] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[51] C. M. J. Tan and M. Motani, "DropNet: Reducing neural network complexity via iterative pruning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9356–9366.

[52] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[53] D. N. Hoang, S. Liu, R. Marculescu, and Z. Wang, "Revisiting pruning at initialization through the lens of Ramanujan graph," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[54] T. Narshana, C. Murti, and C. Bhattacharyya, "DFPC: Data flow driven pruning of coupled channels without data," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[55] C. Murti, T. Narshana, and C. Bhattacharyya, "TVSPrune—Pruning non-discriminative filters via total variation separability of intermediate representations without fine tuning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[56] E. Diao, G. Wang, J. Zhan, Y. Yang, J. Ding, and V. Tarokh, "Pruning deep neural networks from a sparsity perspective," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[57] A. H. Gadhikar, S. Mukherjee, and R. Burkholz, "Why random pruning is all we need to start sparse," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 10542–10570.

[58] M. Lu, X. Luo, T. Chen, W. Chen, D. Liu, and Z. Wang, "Learning pruning-friendly networks via frank-wolfe: One-shot, any-sparsity, and no retraining," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[59] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, 2015, pp. 3123–3131.

[60] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2016, pp. 4114–4122.

[61] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," 2016, *arXiv:1605.04711*.

[62] Y. LeCun. (1998). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/

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

[64] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, p. 4.

[65] Y. Fu, "CPT: Efficient deep neural network training via cyclic precision," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[66] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," in *Proc. Int. Conf. Learn. Represent. (ICLR) workshop*, 2015.

[67] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[68] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 525–542.

[69] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2017, pp. 1474–1479.

[70] T. Huang, L. Tao, and J. T. Zhou, "Adaptive precision training for resource constrained devices," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2020, pp. 1403–1408.

[71] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 7686–7695.

[72] H. Peng, J. Wu, Z. Zhang, S. Chen, and H.-T. Zhang, "Deep network quantization via error compensation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 9, pp. 4960–4970, Sep. 2022.

[73] B. Chmiel, R. Banner, E. Hoffer, H. Ben-Yaacov, and D. Soudry, "Accurate neural training with 4-bit matrix multiplications at standard formats," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023.

[74] S. Lee, J. Park, and D. Jeon, "Toward efficient low-precision training: Data format optimization and hysteresis quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[75] Q. Jin, "F8Net: Fixed-point 8-bit only multiplication for network quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[76] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[77] H. You et al., "ShiftAddNet: A hardware-inspired deep network," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 2771–2783.

[78] H. Chen et al., "AdderNet: Do we really need multiplications in deep learning?" in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1465–1474.

[79] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "DeepShift: Towards multiplication-less neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2021, pp. 2359–2368.

[80] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[81] A. Alkhulaifi, F. Alsahli, and I. Ahmad, "Knowledge distillation in deep learning and its applications," *PeerJ Comput. Sci.*, vol. 7, p. e474, Apr. 2021.

[82] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[83] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[84] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Sep. 2017, pp. 4133–4141.

[85] R. G. Lopes, S. Fenu, and T. Starner, "Data-free knowledge distillation for deep neural networks," 2017, *arXiv:1710.07535*.

[86] F. Tung and G. Mori, "Similarity-preserving knowledge distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1365–1374.

[87] B. Heo, M. Lee, S. Yun, and J. Y. Choi, "Knowledge distillation with adversarial samples supporting decision boundary," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, pp. 3771–3778, Jul. 2019.

[88] H. Zhou et al., "Rethinking soft labels for knowledge distillation: A bias-variance tradeoff perspective," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[89] C. Yang, L. Xie, S. Qiao, and A. L. Yuille, "Training deep neural networks in generations: A more tolerant teacher educates better students," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 5628–5635.

[90] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3713–3722.

[91] M. Ji, S. Shin, S. Hwang, G. Park, and I.-C. Moon, "Refine myself by teaching myself: Feature refinement via self-knowledge distillation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 10664–10673.

[92] T.-B. Xu and C.-L. Liu, "Deep neural network self-distillation exploiting data representation invariance," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 1, pp. 257–269, Jan. 2022.

[93] J. P. Gou, B. S. Yu, S. J. Maybank, and D. C. Tao, "Knowledge distillation: A survey," *Int. J. Comput. Vis.*, vol. 129, no. 31, pp. 1789–1819, Jul. 2021.

[94] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.

[95] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[96] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[97] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017.

[98] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2017, *arXiv:1611.01578*.

[99] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Sep. 2019, pp. 2820–2828.

[100] B. Wu et al., "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10734–10742.

[101] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, D. Z. Pan, Ed., Westminster, CO, USA, Nov. 2019, pp. 1–7.

[102] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 4977–4989.

[103] X. Dai et al., "ChamNet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Sep. 2019, pp. 11398–11407.

[104] A. Howard et al., "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[105] T.-J. Yang et al., "NetAdapt: Platform-aware neural network adaptation for mobile applications," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 285–300.

[106] H. You, B. Li, S. Huihong, Y. Fu, and Y. Lin, "ShiftAddNAS: Hardware-inspired search for more accurate and efficient neural networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 25566–25580.

[107] S. Dhar, J. Guo, J. Liu, S. Tripathi, U. Kurup, and M. Shah, "On-device machine learning: An algorithms and learning theory perspective," 2019, *arXiv:1911.00623*.

[108] L. Wu, D. Wang, and Q. Liu, "Splitting steepest descent for growing neural architectures," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 10656–10666.

[109] L. Wu, B. Liu, P. Stone, and Q. Liu, "Firefly neural architecture descent: A general approach for growing neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 22373–22383.

[110] X. Yuan, P. H. P. Savarese, and M. Maire, "Growing efficient deep networks by structured continuous sparsification," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[111] U. Evci, B. van Merrienboer, T. Unterthiner, F. Pedregosa, and M. Vladymyrov, "Gradmax: Growing neural networks using gradient information," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022.

[112] M. Verbockhaven and G. Charpiat. (2023). *Spotting Expressivity Bottlenecks and Fixing Them Optimally*. [Online]. Available: https://openreview.net/forum?id=xBeGd7sAND

[113] M. Paul, S. Ganguli, and G. K. Dziugaite, "Deep learning on a data diet: Finding important examples early in training," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, vol. 34, 2021, pp. 20596–20607.

[114] M. Toneva, A. Sordoni, R. T. d. Combes, A. Trischler, Y. Bengio, and G. J. Gordon, "An empirical study of example forgetting during deep neural network learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[115] C. Coleman, "Selection via proxy: Efficient data selection for deep learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020.

[116] T. N. Truong, B. Gerofi, E. J. Martinez-Noriega, F. Trahay, and M. Wahib, "KAKURENBO: Adaptively hiding samples in deep neural network training," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2023, pp. 37900–37922.

[117] A. H. Jiang et al., "Accelerating deep learning by focusing on the biggest losers," 2019, *arXiv:1910.00762*.

[118] B. Mirzasoleiman, J. Bilmes, and J. Leskovec, "Coresets for data-efficient training of machine learning models," in *Proc. 37th Int. Conf. Mach. Learn.*, 2020, pp. 6950–6960.

[119] K. Killamsetty, D. Sivasubramanian, B. Mirzasoleiman, G. Ramakrishnan, A. De, and R. Iyer, "Grad-match: A gradient matching based data subset selection for efficient learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021, pp. 5464–5474.

[120] Y. Yang, H. Kang, and B. Mirzasoleiman, "Towards sustainable learning: Coresets for data-efficient deep learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 39314–38330.

[121] P. Yao et al., "ASP: Automatic selection of proxy dataset for efficient AutoML," 2023, *arXiv:2310.11478*.

[122] J. Zhang, H.-F. Yu, and I. S. Dhillon, "Autoassist: A framework to accelerate training of deep neural networks," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 5998–6008.

[123] Y. Fan, F. Tian, T. Qin, X.-Y. Li, and T.-Y. Liu, "Learning to teach," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[124] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *Proc. ICML*, 2018, pp. 2304–2313.

[125] T.-H. Kim and J. Choi, "ScreenerNet: Learning self-paced curriculum for deep neural networks," 2018, *arXiv:1801.00904*.

[126] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. ICML*, 2009, pp. 41–48.

[127] A. Boumendil, W. Bechkit, and K. Benatchba, "On data selection for the energy efficiency of neural networks: Towards a new solution based on a dynamic selectivity ratio," in *Proc. IEEE 35th Int. Conf. Tools With Artif. Intell. (ICTAI)*, Nov. 2023, pp. 325–332.

[128] H. T. Kesgin and M. F. Amasyali, "Cyclical curriculum learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Apr. 17, 2023, doi: 10.1109/TNNLS.2023.3265331.

[129] X. Wu, E. Dyer, and B. Neyshabur, "When do curricula work?" in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2021.

[130] Y. Wang et al., "E2-train: Training state-of-the-art CNNs with over 80% energy savings," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 5138–5150.

[131] P. Okanovic et al., "Repeated random sampling for minimizing the Time-to-Accuracy of learning," 2023, *arXiv:2305.18424*.

[132] *Intel Math Kernel Library*. Accessed: May 31, 2023. [Online]. Available: https://www.intel.com/content/www/us/ en/docs/onemkl/get-started-guide/2023-0/overview.html

[133] *NVIDIA cuDNN*. Accessed: May 31, 2023. [Online]. Available: https://developer.nvidia.com/cudnn

[134] M. Sankaradas et al., "A massively parallel coprocessor for convolutional neural networks," in *Proc. 20th IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, Jul. 2009, pp. 53–60.

[135] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk, "Towards an embedded biologically-inspired machine vision processor," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2010, pp. 273–278.

[136] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. 37th Annu. Int. Symp. Comput. Archit.*, Jun. 2010, pp. 247–257.

[137] L. Cavigelli, D. Gschwend, C. Mayer, S. Willi, B. Muheim, and L. Benini, "Origami: A convolutional network accelerator," in *Proc. 25th, Ed., Great Lakes Symp. (VLSI)*, May 2015, pp. 199–204.

[138] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[139] Z. Du, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. Int. Symp. Comput. Archit.*, vols. 13–17. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers, Jun. 2015, pp. 92–104.

[140] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 13–19.

[141] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2015, pp. 161–170.

[142] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 269–284, 2014.

[143] J. Lu, C. Ni, and Z. Wang, "ETA: An efficient training accelerator for DNNs based on hardware-algorithm co-optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 7660–7674, Feb. 2022.

[144] K. S. Zaman, M. B. I. Reaz, S. H. M. Ali, A. A. A. Bakar, and M. E. H. Chowdhury, "Custom hardware architectures for deep learning on portable devices: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6068–6088, Nov. 2022.

[145] *Pytorch*. Accessed: Dec. 23, 2022. [Online]. Available: https://pytorch.org/

[146] *TensorFlow*. Accessed: Dec. 23, 2022. [Online]. Available: https://www.tensorflow.org/?hl=fr

[147] *Pytorch Mobile*. Accessed: Dec. 23, 2022. [Online]. Available: https://pytorch.org/mobile/home/

[148] *TensorFlow Lite*. Accessed: Dec. 23, 2022. [Online]. Available: https://www.tensorflow.org/lite

[149] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256KB memory," 2022, *arXiv:2206.15472*.

[150] S. G. Patil, P. Jain, P. Dutta, I. Stoica, and J. Gonzalez, "POET: Training neural networks on tiny devices with integrated rematerialization and paging," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 17573–17583.

[151] T. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

[152] M. Jay, V. Ostapenco, L. Lefevre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: Focus on CPU and GPU," in *Proc. IEEE/ACM 23rd Int. Symp. Cluster, Cloud Internet Comput. (CCGrid)*, May 2023, pp. 106–118.

[153] *Running Average Power Limit—RAPL*. Accessed: May 4, 2023. [Online]. Available: https://01.org/blogs/2014/running-average-power-limit-%E2%80%93-rapl

[154] *NVIDIA Management Library (NVML)*. Accessed: May 4, 2023. [Online]. Available: https://developer.nvidia.com/nvidia-management-library-nvml

[155] *Tegrastats Utility*. Accessed: May 4, 2023. [Online]. Available: https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/index.html

[156] *PyRAPL's Documentation*. Accessed: May 4, 2023. [Online]. Available: https://pyrapl.readthedocs.io/en/latest/

[157] *PyJoules's Documentation*. Accessed: May 4, 2023. [Online]. Available: https://pyjoules.readthedocs.io/en/latest/

[158] *Energyusage*. Accessed: May 4, 2023. [Online]. Available: https://pypi.org/project/energyusage/

[159] V. Schmidt et al., "CodeCarbon: Estimate and track carbon emissions from machine learning computing," MILA, BCG GAMMA, HAVERFORD College, Comet, 2021. [Online]. Available: https://codecarbon.io/

[160] A. Boumendil, W. Bechkit, P.-E. Portier, F. L. Mouël, and M. Egan, "Grow, prune or select data: Which technique allows the most energy-efficient neural network training?" in *Proc. IEEE 35th Int. Conf. Tools With Artif. Intell. (ICTAI)*, Nov. 2023, pp. 303–308.

[161] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural Netw.*, vol. 111, pp. 47–63, Mar. 2019.

[162] J. D. Nunes, M. Carvalho, D. Carneiro, and J. S. Cardoso, "Spiking neural networks: A survey," *IEEE Access*, vol. 10, pp. 60738–60764, 2022.

[163] D. V. Christensen et al., "2022 roadmap on neuromorphic computing and engineering," *Neuromorphic Comput. Eng.*, vol. 2, no. 2, 2022, Art. no. 022501.

[164] Y. Kim and P. Panda, "Visual explanations from spiking neural networks using inter-spike intervals," *Sci. Rep.*, vol. 11, no. 1, p. 19037, Sep. 2021.

[165] G.-Q. Bi and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type," *J. Neurosci.*, vol. 18, no. 24, pp. 10464–10472, Dec. 1998.

[166] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2015, pp. 1–8.

[167] B. Han, G. Srinivasan, and K. Roy, "RMP-SNN: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Sep. 2020, pp. 13558–13567.

[168] Y. Kim and P. Panda, "Revisiting batch normalization for training low-latency deep spiking neural networks from scratch," *Frontiers Neurosci.*, vol. 15, Dec. 2021, Art. no. 773954.

[169] C. Lee, S. S. Sarwar, P. Panda, G. Srinivasan, and K. Roy, "Enabling spike-based backpropagation for training deep neural network architectures," *Front. Neurosci.*, vol. 14, p. 119, Feb. 2020.

[170] D. Ielmini and H.-S.-P. Wong, "In-memory computing with resistive switching devices," *Nature Electron.*, vol. 1, no. 6, pp. 333–343, Jun. 2018.

[171] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[172] E. Painkras et al., "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation," *IEEE J. Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, Aug. 2013.

[173] F. Akopyan et al., "TrueNorth: Design and tool flow of a 65 mW 1 million neuron programmable neurosynaptic chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 10, pp. 1537–1557, Oct. 2015.

[174] B. V. Benjamin et al., "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proc. IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[175] A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu, and Q. Qiu, "A survey on neuromorphic computing: Models and hardware," *IEEE Circuits Syst. Mag.*, vol. 22, no. 2, pp. 6–35, 2nd Quart., 2022.

[176] H. Zhang and D. V. Vargas, "A survey on reservoir computing and its interdisciplinary applications beyond traditional machine learning," *IEEE Access*, vol. 11, pp. 81033–81070, 2023.

[177] C. Gallicchio, A. Micheli, and L. Pedrelli, "Deep reservoir computing: A critical experimental analysis," *Neurocomputing*, vol. 268, pp. 87–99, Dec. 2017.

[178] H. Jaeger, "The 'echo state' approach to analysing and training recurrent neural networks-with an erratum note," *German Nat. Res. Cntr. Inf. Technol., GMD Rep.*, vol. 148, no. 34, p. 13, 2001.

[179] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, Nov. 2002.

[180] M. A. Chitsazan, M. Sami Fadali, and A. M. Trzynadlowski, "Wind speed and wind direction forecasting using echo state network with nonlinear functions," *Renew. Energy*, vol. 131, pp. 879–889, Feb. 2019.

[181] M. L. Alomar et al., "Efficient parallel implementation of reservoir computing systems," *Neural Comput. Appl.*, vol. 32, no. 7, pp. 2299–2313, Apr. 2020.

[182] D. Verstraeten, B. Schrauwen, and D. Stroobandt, "Reservoir computing with stochastic bitstream neurons," in *Proc. 16th Annu. Prorisc Workshop*, 2005, pp. 454–459.

[183] M. L. Alomar, V. Canals, V. Martínez-Moll, and J. L. Rosselló, "Low-cost hardware implementation of reservoir computers," in *Proc. 24th Int. Workshop Power Timing Modeling, Optim. Simulation (PATMOS)*, Sep. 2014, pp. 1–5.

[184] X. Yang, W. Chen, and F. Z. Wang, "Investigations of the staircase memristor model and applications of memristor-based local connections," *Anal. Integr. Circuits Signal Process.*, vol. 87, no. 2, pp. 263–273, May 2016.

[185] A. M. Hassan, H. H. Li, and Y. Chen, "Hardware implementation of echo state networks using memristor double crossbar arrays," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2171–2177.

[186] N. Soures, L. Hays, and D. Kudithipudi, "Robustness of a memristor based Liquid State Machine," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 2414–2420.

[187] S. Jovanovic and H. Hikawa, "A survey of hardware self-organizing maps," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 11, pp. 8154–8173, Nov. 2023.

[188] M. Nair, S. Dang, and M. A. Beach, "IoT device authentication using self-organizing feature map data sets," *IEEE Commun. Mag.*, vol. 61, no. 9, pp. 162–168, Sep. 2023.

[189] A. Russo, F. Verdier, and B. Miramond, "Energy saving in a wireless sensor network by data prediction by using self-organized maps," *Proc. Comput. Sci.*, vol. 130, pp. 1090–1095, Jan. 2018.

[190] Z. Shi, C. Sakhuja, M. Hashemi, K. Swersky, and C. Lin, "Learned Hardware/Software co-design of neural accelerators," 2020, *arXiv:2010.02075*.

[191] Y. Zhou et al., "Towards the co-design of neural networks and accelerators," *Proc. Mach. Learn. Syst.*, vol. 4, pp. 141–152, Jan. 2022.

[192] M. S. Abdelfattah, Ł. Dudziak, T. Chau, R. Lee, H. Kim, and N. D. Lane, "Best of both worlds: AutoML codesign of a CNN and its hardware accelerator," in *Proc. ACM/EDAC/IEEE Design Autom. Conf.*, Sep. 2020, pp. 1–6.

[193] W. Jiang et al., "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 595–605, Apr. 2021.

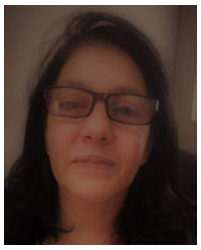
[194] S. Negi, I. Chakraborty, A. Ankit, and K. Roy, "NAX: Co-designing neural network and hardware architecture for memristive xbar based computing systems," 2021, *arXiv:2106.12125*.

Anais Boumendil received the master's and Engineering degrees in computer science from the École nationale Supérieure d'Informatique (ESI), Algiers, Algeria, in 2022. She is currently pursuing the Ph.D. degree with the CITI Laboratory, Inria, INSA Lyon, Villeurbanne, France.

Her research topic is around reducing the energy consumption of neural networks to allow their integration in resource-constrained environments.

Walid Bechkit received the Engineering degree in computer science from the École nationale Supérieure d'Informatique, Algiers, Algeria, in 2009, and the Ph.D. degree in system and information technology from the University of Technology of Compiègne (UTC), Compiègne, France, in 2012.

He is currently an Associate Professor within INSA Lyon, Villeurbanne, France, where he is also a member of the INRIA Agora Team, CITI Laboratory. His main research interests include wireless sensor networks, the Internet of Things (IoT) applications, data analysis, machine and deep learning, low power wide area network (LPWAN) technologies, and security.



Karima Benatchba is currently a Professor at the École nationale Supérieure d'Informatique, Algiers, Algeria, where she is also the Head of the Laboratoire des Méthodes de Conception des Systèmes (LMCS) and a member of the Optimisation Team. Her research focuses on the synergy between combinatorial optimization and machine learning.