IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 6, 2021, accepted November 5, 2021, date of publication November 8, 2021, date of current version November 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3126685

Neural Architecture Search and Hardware Accelerator Co-Search: A Survey

LUKAS SEKANINA, (Senior Member, IEEE)
Faculty of Information Technology, Brno University of Technology, 61266 Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz

This work was supported by the Czech Science Foundation Project under Grant 21-13001S.

ABSTRACT Deep neural networks (DNN) are now dominating in the most challenging applications of machine learning. As DNNs can have complex architectures with millions of trainable parameters (the so-called weights), their design and training are difficult even for highly qualified experts. In order to reduce human effort, neural architecture search (NAS) methods have been developed to automate the entire design process. The NAS methods typically combine searching in the space of candidate architectures and optimizing (learning) the weights using a gradient method. In this paper, we survey the key elements of NAS methods that – to various extents – consider hardware implementation of the resulting DNNs. We classified these methods into three major classes: single-objective NAS (no hardware is considered), hardware-aware NAS (DNN is optimized for a particular hardware platform), and NAS with hardware co-optimization (hardware is directly co-optimized with DNN as a part of NAS). Compared to previous surveys, we emphasize the multi-objective design approach that must be adopted in NAS and focus on co-design algorithms developed for concurrent optimization of DNN architectures and hardware platforms. As most research in this area deals with NAS for image classification using convolutional neural networks, we follow this trajectory in our paper. After reading the paper, the reader should understand why and how NAS and hardware co-optimization are currently used to build cutting-edge implementations of DNNs.

INDEX TERMS Automated design, classification, co-design, deep neural network, hardware accelerator, neural architecture search, optimization.

I. INTRODUCTION

Machine learning (ML) technology is now routinely applied in cutting-edge applications such as image, speech, and natural language recognition, data mining, autonomous car driving, and automated system design in which humans had no competitors a few years ago. The core ML algorithms utilize *deep neural networks* (DNNs) — complex computational models that must be designed and then trained using suitable data from a given application domain [1]. For example, DNN called ViT-H/14 that shows state-of-the-art classification accuracy on one of the most significant image classification benchmarks (ImageNet) consists of 632 million trainable parameters [2]. As designing such complex DNNs is very time-consuming and requires skilled experts, much effort has been invested in recent years to automate this work.

The associate editor coordinating the review of this manuscript and approving it for publication was Liangxiu Han.

Neural architecture search (NAS) [3]–[5] is a method capable of automated design of complex neural networks (NN) such as DNNs. In its single-objective setup, it creates neural networks that are optimized according to one objective (typically, the quality of output, expressed in terms of accuracy or other similar metrics). NAS has to solve two problems concurrently – designing the NN's architecture (including its size, structure, and types of components) and optimizing its trainable parameters (the so-called weights). Each of these problems is difficult in itself, and its solving requires considerable computing resources. The NAS methods typically combine searching in the space of candidate NNs (see Sect. IV-B) and optimizing (learning) the weights using a gradient method. However, the most significant benefit of the NAS methods is if they are used in a multi-objective setup and optimize not only the NN quality but also other parameters such as size or latency. If more objectives are considered, NAS methods currently provide state-of-the-art DNN implementations [6]–[8].

Only a fully trained NN is needed in most ML applications while its training is performed on a computer cluster before its deployment. However, even such a fully trained NN must execute billions of elementary arithmetic operations to process a single input, i.e., to perform the so-called *inference*. Note that each of the trainable parameters typically undergoes at least one multiplication during inference.

In order to reach desired latency or energy efficiency, hardware accelerators for NN inference were developed in recent years [9], [10]. In this direction, hardware-aware NAS methods were proposed to design NN architecture (and weights) optimally for a given hardware platform. Compared to single-objective methods, multi-objective hardware-aware NAS delivered similar accuracy but reduced latency, size, and/or power consumption, which was demonstrated across different hardware platforms [7], [11], [12]. However, these methods only optimize the NN architecture. They do not extend the search space to co-optimize NN architecture with parameters of hardware platforms (such as amount and type of resources, dataflow strategies, buffer sizes, and compiler options). Thus, they neglect the hardware design freedom provided by many platforms (e.g., in FPGAs) [13]. Hence, the newest NAS methods co-optimize NN architectures and hardware configuration to further improve latency and other parameters that are important in many applications such as IoT or mobile phones. These methods work in three search spaces (weights, NN architectures, and hardware configurations) and must innovatively orchestrate several search algorithms to produce the best trade-offs between the accuracy and various hardware-relevant metrics [8], [14], [15].

Modern NAS methods thus enabled to increase the design productivity by releasing human experts' capacity and improve the quality, performance, and energy efficiency of resulting neural accelerators by adopting a data-driven automated search-based design approach.

A. PREVIOUS SURVEYS

The body of work dealing with NAS has significantly increased since 2015, as illustrated in Fig. 1. This rapid development is captured by detailed survey papers being published since 2018 [4], [5], [16]–[18]. Some specialized surveys focused on particular search methods such as metaheuristics [19], neuroevolution [20], and reinforcement learning-based NAS [21]. The most recent survey [22] provides a unifying view on NAS in terms of representation of DNNs, variation operators, multi-objective search, constraint handling, and performance estimation. However, these surveys only briefly mention hardware-aware NAS, if at all. The hardware design community only very recently started to survey relevant papers on hardware-aware NAS [23]–[25]. A detailed survey is currently provided only by unpublished paper [25]. As their work was finished in 2020, it does not fully cover the most recent approaches in which neural architecture is co-designed with a hardware accelerator.

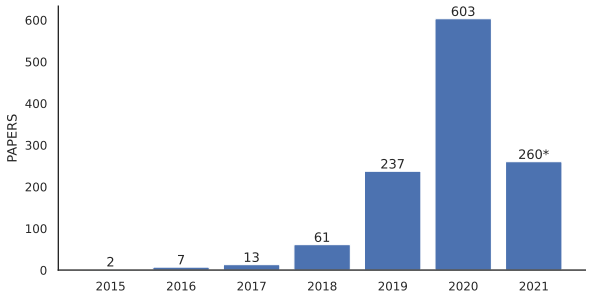

FIGURE 1. The number of papers on NAS according to automl.org database (*June 30, 2021).

B. PLAN OF THE SURVEY

In this paper, we focus on this new category: NAS with hardware co-design. Rather than providing a unified view which is hard to establish in this rapidly expanding field, our survey tries to follow the development from single-objective NAS methods, via hardware-aware NAS (in which DNN is optimized for a particular hardware platform) to the NAS with hardware co-search, in which the hardware design space has to be explored in addition to the space of network architectures. Relevant NAS methods are classified according to carefully selected criteria and arranged in a tabular form. For the last category, we propose a new classification approach to understand how multiple search algorithms interact to construct an efficient DNN and its hardware accelerator. We emphasize the importance of fair comparison and benchmarking methodology for NAS methods and show examples of such comparisons. The entire topic is challenging to present because it requires that the reader is familiar with different fields: DNNs, methods for multi-objective optimization, and hardware accelerator design. Hence, the survey starts with a brief introduction to DNNs and explains the principles of the accelerator design. Most research in this field deals with NAS for convolutional neural networks (CNN) applied to the image classification task. Our survey is primarily focused on this area. Table 1 provides a list of abbreviations used throughout this paper. As a core database of relevant papers, we selected *automl.org* [26], which contains not only papers from standard databases of IEEE or ACM but also unpublished preprints.

The rest of the paper is organized as follows. Section II summarizes relevant principles of neural networks, emphasizing CNNs, and their optimization and benchmarking. Section III is devoted to hardware accelerators developed for CNN inference. NAS methods that optimize only the CNN accuracy are surveyed in Section IV. The hardware-aware NAS is introduced in Section V, in which we also describe common multi-objective optimization approaches. The NAS with hardware co-design is discussed in Section VI. Specialized techniques and hardware platforms in NAS are treated separately in Section VII. Section VIII deals with evaluation and benchmarking of NAS methods. Concluding remarks are given in Section IX.

TABLE 1. List of abbreviations. Note that full titles of NAS methods, human-created CNN models, and benchmark sets are not included.

ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Networks
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EA	Evolutionary Algorithm
FLOPS	Floating-Point Operations per Second
FP	Floating-Point
FPGA	Field Programmable Gate Array
GP	Gaussian Process
GPU	Graphics Processing Units
IoT	Internet of Things
IP	Intellectual Property
LSTM	Long Short-Term Memory
LUT	Look-Up Table
MAC	Multiply-and-Accumulate
MCU	Microcontroller
MCTS	Monte Carlo Tree Search
M-H	Metropolis-Hasting algorithm
ML	Machine Learning
MLP	Multi-Layer Perceptron
NAS	Neural Architecture Search
NN	Neural Network
NoC	Network on Chip
NSGA	Non-Dominated Sorting Genetic Algorithm
OPS	Operations per Second
PE	Processing Element
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SIMD	Single-Instruction Multiple-Data
SIMT	Single-Instruction Multiple-Threads
SMBO	Sequential Model Based Optimization
TPU	Tensor Processing Units

II. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks are computational models inspired by biological brains. They are used in machine learning tasks such as classification, prediction, control, and function approximation. NN is defined by its *architecture* and *weights* [1]. The number and type of layers, neurons, and other parameters that define the architecture of the NN are called *hyperparameters*. Once the architecture and hyperparameters of NN are specified, NN can undergo a training procedure whose goal is to optimize trainable parameters (the so-called weights) to minimize a given *loss function*. In classical supervised gradient learning, the training algorithm works in iterations (epochs). For each data input (i.e., an input vector or a subset from input vectors called a *batch*), it computes the output vector, which is compared with the desired vector to determine the error. The error is propagated back along with the network, and by utilizing the gradient of the loss function, the weights are appropriately updated. While training is performed with a *training data set*, the final quality score of the NN (such as the classification accuracy) is determined for a *test data set*. The *accuracy* gives the proportions of correct classifications over a given data set. In this paper, the accuracy always refers to the accuracy on the test data. The top-n accuracy is the proportion of testing data for which any of the n highest-probability predictions is considered as a correct result.

In this paper, we will primarily deal with *convolutional neural networks* (CNNs) whose architecture is defined as a sequence of layers with no feedback that are composed of artificial neurons and other elements, and at least one of the layers is the so-called convolutional layer [27]. We will only briefly mention *recurrent neural networks* (RNNs) that have been developed for time-dependent problems. They support both feedback and feed-forward connections and can store intermediate results internally in the NN. *Long short-term memory* networks (LSTMs) are the most popular variant of RNNs capable of capturing long-term time dependencies [1].

After introducing selected basic types of NNs, this section will focus on well-known CNN models developed by human experts and various techniques improving the performance of CNNs. As the main focus of this paper is the automated design of hardware-aware CNNs, we will not further deal with other types of DNNs, training algorithms, and application domains.

A. NEURONS AND MULTI-LAYER NETWORKS

A basic *artificial neuron* is an elementary building block of complex NNs. A neuron has n inputs $(x_1, x_2, \ldots, x_n)$ and returns a scalar output y (Fig. 2). The inputs are multiplied with the weights $(w_1, w_2, \ldots, w_n)$ and summed together with a bias term b. A non-linear *activation function* $\sigma(z)$ is then applied to calculate the output of the neuron, i.e.

$$y = \sigma(z), \quad \text{where } z = \sum_{i=1}^{n} x_i w_i + b. \tag{1}$$

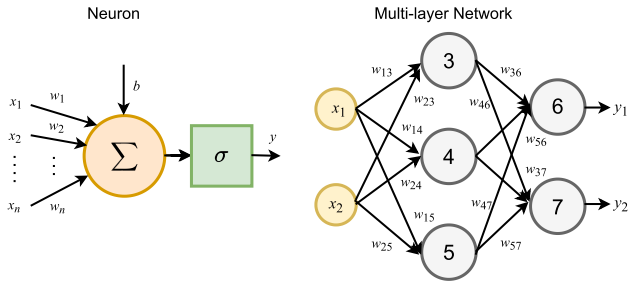

FIGURE 2. A single neuron (left) and an example of a multi-layer fully-connected feed-forward neural network (right).

Common activation functions are Rectified Linear Unit (ReLU), sigmoid or hyperbolic tangent [1]. When used as a classifier, one neuron can, after training, only classify the input vectors to linearly separable classes.

To approximate general functions, a *multi-layer* NN is used, which consists of multiple layers of neurons in which a suitable number of neurons that work in parallel constitute one layer (Fig. 2). *Fully connected* (FC) layers are usually organized in such a way that if some layer contains n neurons, then each neuron is connected to all m neurons of the previous layer, which requires to set up $m \times n$ weights.

If the NN has to approximate a function $F(x_1, x_2, \ldots, x_n)$ then the last (output) layer contains just one neuron. If the NN works as a classifier into k classes, then the output layer has k neurons. The i-th neuron gives the probability of i-th class, which is often obtained using a *softmax* function:

$$p(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}. \quad (2)$$

The number of layers and the number of neurons in each layer are hyperparameters of a multi-layer fully connected NN. When the trained NN is deployed, it only performs feed-forward computations to obtain the result for every input; this is called the *inference*.

B. CONVOLUTIONAL NEURAL NETWORKS

In order to reduce the number of weights and automatically extract essential features from raw data, CNNs have been introduced in the image processing domain. CNNs contain between five and hundreds of layers. Each convolutional layer generates, by applying one or several convolutional *kernels* (filters), a successively higher level of abstraction of the input data, called a *feature map*. The core computational procedure of a convolutional layer is a high-dimensional convolution (Fig. 3). The convolutional layers take input activation maps, arranged in three dimensions (i.e., height H_{in}, width W_{in} and channel C_{in}), and generate output activation maps, arranged in three dimensions (i.e., height H_{out}, width W_{out} and channel C_{out}). Mathematically, it is the convolution between the input activation maps and a set of C_{out} 3D filters. More precisely, every single 2D $H_{out} \times W_{out}$ plane of the output activation maps is a result of the convolution between the 3D input activation maps with a set of 3D filters. The final step is adding a 1D bias. Formally, the convolutional processes with the input activation maps, the output activation maps, the filters and the bias matrices denoted as X, Y, W, and B, respectively, can be expressed as

$$Y(z,t,q) = B(q) + \sum_{k=1}^{C_{in}} \sum_{j=1}^{H_f} \sum_{i=1}^{W_f} X(zS+j, tS+i, k) \times W(j,i,k,q), \quad (3)$$

$$H_{out} = (H_{in} - H_f + S)/S, \quad (4)$$

$$W_{out} = (W_{in} - W_f + S)/S, \quad (5)$$

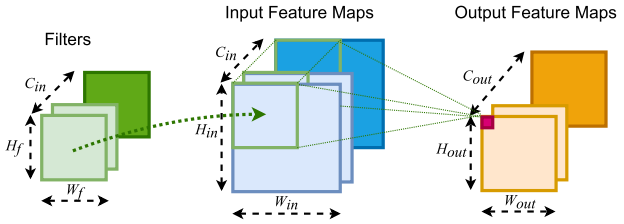

FIGURE 3. High dimensional convolution in CNNs.

where $1 \leq z \leq H_{out}$, $1 \leq t \leq W_{out}$ and $1 \leq q \leq C_{out}$. The *stride* S is the number of pixels of which the filter is shifted after each convolution. The parameters used in the convolutional process are summarized in Table 2.

TABLE 2. Parameters of convolutional layers.

Parameter	Description
H_{in}/W_{in}	Height/Width of the input activation maps
H_{out}/W_{out}	Height/Width of the output activation maps
H_f/W_f	Height/Width of the filter plane
C_{in}/C_{out}	No. of channels of the input/output activation maps
S	Stride size

CNNs are organized to learn the non-linear mapping between the features and resulting classes, layer by layer, where higher-level features are extracted from lower-level features obtained in previous layers. A non-linear activation function typically follows each convolutional layer. CNNs also contain pooling layers, normalization layers, and specialized blocks of layers (e.g., residual and inception blocks will be discussed in Section II-C). *Pooling layers* combine, by applying the averaging or maximum operators, a set of input values into a small number of output values to reduce the dimension of feature maps. *Normalization layers* enable to control the input distribution across layers which can help to speed up the training process and improve accuracy.

C. HUMAN-CREATED CNN MODELS AND BENCHMARKS

The use of a CNN called AlexNet [27] (Fig. 4a) trained on GPUs led to a breakthrough result in the ImageNet 2012 challenge focused on image classification. AlexNet achieved a top-5 error of 15.3%, i.e., 10.8% lower than other competitors utilizing conventional classifiers. The ImageNet

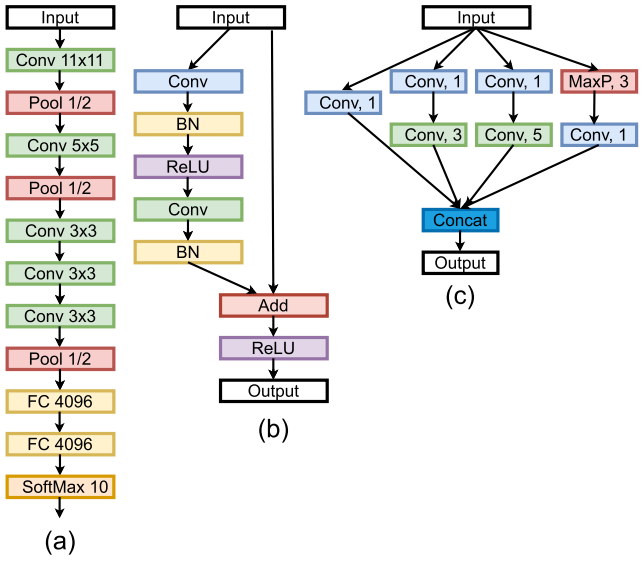

FIGURE 4. Examples of hand-crafted CNNs and their blocks: (a) AlexNet for CIFAR-10 classification; (b) Residual block from ResNet [31]; (c) Inception block from GoogleNet [30]. Abbreviations of layers: Conv – convolution; Pool – pooling; FC – fully connected; BN – batch normalization; Add – addition; MaxP – max. pooling; Concat – concatenation.

benchmark data set contains 1.2 million training images, with roughly 1000 images in each of 1000 categories [28].

Since that time, various innovations have been proposed to improve CNNs, leading to CNNs showing around 90% top-1 accuracy on ImageNet. In addition to maximizing the classification accuracy, many efforts have been invested in minimizing the size and latency of CNNs to deploy them in resource-constrained devices such as mobile phones.

Major innovations are presented by CNN models given in Table 3; a detailed benchmarking analysis is in [29]. Please note that some of the following networks exist in several versions (e.g., ResNet-8, ResNet-14 etc.), which differ in the number of layers, the structure of building blocks, and some other parameters. The top-1 accuracy depends on the training algorithm setup and time and resources available for training. We will briefly introduce some CNN models important for this paper.

TABLE 3. Parameters and performance of selected human-created CNN models on ImageNet. MobileNetV3 models were developed with help of NAS.

Model	Ref.	Top-1 Acc. [%]	Top-5 Acc. [%]	Parameters ($\cdot 10^6$)	MAC ($\cdot 10^9$)
AlexNet	[27]	63.3	83.6	61.0	0.724
VGG-16	[34]	76.3	93.2	138.0	15.500
GoogLeNetV1	[30]	-	93.3	7.0	1.430
ResNet-50	[31]	79.3	94.7	25.5	3.900
ShuffleNet (1.5)	[35]	71.5	-	3.4	0.292
ShuffleNet (x2)	[35]	73.7	-	5.4	0.524
MobileNetV1	[32]	70.6	-	4.2	0.575
MobileNetV2	[32]	72.0	-	3.4	0.300
MobileNetV3-Large	[33]	75.2	-	5.4	0.219
MobileNetV3-Small	[33]	67.4	-	2.9	0.066
ViT-H/14	[2]	88.6	-	632.0	-

GoogLeNet [30] is the first complex CNN formed by stacking with *inception modules* (Fig. 4c) in which various convolutional operations with different sizes are performed in parallel and their results are aggregated by concatenation. ResNet [31] introduced the so-called *residual blocks* (Fig. 4b) containing a shortcut connection that enables to eliminate the gradient vanishing problem without degeneration in CNNs since the gradient is directly passed through shortcut connections. A three-layer residual block is also called the *bottleneck module* because the two ends of the block are narrower than the middle. MobileNetV1 used depthwise separable convolutions as an efficient replacement for traditional convolution layers. MobileNetV2 proposed the linear bottleneck and inverted residual structure [32]. MobileNetV3 [33] directly employed a platform-aware NAS to find the structure of the network and the key hyperparameters. Inspired by the scaling of the so-called Transformer language models, Vision Transformer (ViT) models [2] were introduced. For example, a ViT-H/14 model with 632 million parameters shows state-of-the-art top-1 accuracy of 88.55% on ImageNet.

There are many software platforms (such as TensorFlow [36] and PyTorch [37]) enabling us to use an existing CNN model or create a new one and evaluate it on pre-prepared data sets. CNNs are evaluated on benchmark problems (or data sets), where image classification is the most popular. ImageNet is considered as a difficult and highly important benchmark, for which the ML community carefully monitors the progress in the Top-1 accuracy. Some smaller CNNs (developed for, e.g., low-power devices) are only evaluated on less complex image sets such as CIFAR-10 (10 image classes) [38], CIFAR-100 (100 classes) [38], MNIST (10 classes) [39], Fashion-MNIST (F-MNIST, 10 classes), SVHN (10 classes) [40], and NORB [41].

Some other data sets are utilized in the papers that will be surveyed in the next sections. Table 4 gives their abbreviation and a brief description. Further details are available in particular papers referenced in Section IV, V, and VI.

TABLE 4. Additional benchmark problems.

Abbreviation	Description
Chars74k	character recognition in natural images
Cityscapes	semantic understanding of street scenes
COCO	Microsoft large-scale object detection, segmentation
CUReT	Columbia-Utrecht texture classification
Flower-17	flower image classification
GTSRB	traffic sign recognition
IDS2012	network traffic classification
ISCX VPN	network traffic classification
KWS	audio, keyword spotting
MIMII	anomaly detection in sound
Nuclei	image segmentation
Pascal VOC	object detection, semantic segmentation, classification
PTB	The English Penn Treebank corpus for text labeling
Sport8	action recognition in images
STL-10	unsupervised feature learning for images
VWW	Visual Wake Words – person detection
WikiText-2	language modeling (text)

D. CNN OPTIMIZATION

CNNs are typically used for error-resilient applications in which a minor error introduced by inexact computing is often invisible to the end-user. Hence, CNNs can be simplified to reduce hardware resources, power consumption, or latency.

By *pruning*, some connections, neurons, filters, and channels can be removed [42]. By *quantization*, the most suitable number of bits and data format is assigned to selected weights, activations, and other intermediate results in the network instead of using the common 32-bit floating-point (FP) data type [42]–[44]. Recent studies have shown that with novel quantization methodologies, namely PACT and SAWB, and specialized number formats, DLFloat16 (16 bit) and Hybrid-FP8 (8 bit) for training, and INT4 for inference, no loss in accuracy can be reached for 4-bit inference for common CNN models on ImageNet [45]. *Model compression* tries to reduce the number of different weight values to minimize the CNN memory footprint [42], [46].

NAS algorithms that will be discussed in Sections V and VI implicitly perform pruning. Searching for the optimal bit widths is directly performed by, e.g., [47]–[51].

III. HARDWARE IMPLEMENTATION OF NEURAL NETWORKS

Accelerating the DNN training as well as inference in specialized hardware has been a vital research topic since 2014. While training is typically performed on (clusters of) graphics processing units (GPUs) or Tensor Processing Units (TPUs), the accelerated inference is carried out on a variety of computing platforms ranging from low-power processors to high-performance specialized multi-chip systems. As the paper deals with hardware-aware NAS, we will primarily focus on accelerators devoted to inference. This topic is well-covered in the literature, see, e.g., a recent book [52] or detailed fresh surveys from the year 2020 [10], [45], [46], [53]–[55].

The CNN accelerator design is motivated by the fact that most computations are carried out in convolutional layers whose computation is suitable for parallelization. Moreover, the parameters (weights) associated with convolutional filters are reused many times. For example, while 666 million multiply-and-accumulate (MAC) operations are performed in convolutional layers, only 58.6 million MACs are in fully connected layers of AlexNet. In the case of ResNet-50, the ratio of MACs conducted in convolutional to fully connected layers is 1930× [9]. Hence, by a smart organization of the convolutional operations, which involves supplying the relevant data on time, introducing a suitable data reusing strategy, and bit-width setting, a significant improvement in latency and power consumption can be obtained using specialized parallel hardware. Chen *et al.* suggested to directly optimize the data reuse, which is the number of MACs that use the same piece of data, i.e., MACs/data, to maximize the energy efficiency [56]. For example, if all data reuse is exploited, DRAM accesses in AlexNet can be reduced from 2 896 to 61 million [52].

A. HARDWARE PLATFORMS FOR DNNs

In addition to general-purpose multicore CPUs and GPUs, various specialized accelerators have been developed to implement DNNs. These accelerators are implemented either as application-specific integrated circuits (ASICs), in field-programmable gate arrays (FPGAs), or by extending common processors. The ASIC-based accelerators are typically created by leading companies, e.g., Google's series of TPUs [57], Intel's Movidius [58], IBM's TrueNorth [59], NVIDIA's NVDLA [60] and at universities, e.g., DaDianNao [61], Origami [62], Eyeriss [56], and ZASCA [55].

Google's TPU accelerators exist in several versions [57]. TPUv1, introduced in 2016, provided a systolic array of 256 × 256.8-bit FX multipliers allowing to significantly accelerate matrix multiplications for CNN inference (with the peak performance 92 TOPS at 75 W). TPUv2 and TPUv3 provide increased performance and support FP operations which makes them usable for DNN training. EdgeTPU is a version developed for edge computing and smartphones.

A detailed survey of FPGA-based acceleration techniques for DNNs was recently published in [10], [63]. On the other hand, DNN accelerators based on processors and microcontrollers are an attractive solution not only because of their low power operation useful in the IoT domain but also because of easier programmability and flexibility compared to specialized hardware accelerators. Thanks to the CMSIS-NN library [64] proposed by ARM for 16-bit and 8-bit quantized DNNs on Cortex-M microcontrollers; as well as PULP-NN, an opensource library targeting RISC-V processors, and supporting heavily quantized DNNs working on 8-bit, 4-bit, 2-bit, or even 1- bit data [65], these processors can implement complex CNNs such as MobileNetV1. To enable energy-efficient inference of quantized DNNs (with up to 550 GOPS/W) for IoT applications, the instruction set was extended for RISC-V by implementing low-bit width SIMD arithmetic instructions [66].

It is important to emphasize that the acceleration approaches mentioned above show very different trade-offs between two critical evaluation criteria – *performance* (inferences/s) and *energy efficiency* (inferences/s/W). On the AlexNet example, Table 5 indicates that GPUs provide the highest performance while specialized ASICs lead to the most energy-efficient implementations. Detailed evaluation methodology for DNN accelerators was published in [67].

TABLE 5. Performance and energy-efficiency of AlexNet on various platforms (according to [56], [68]).

Platform	Chip	Freq. [MHz]	Precision	Perform. [infer./s]	Power [W]	Efficiency [infer./s/W]
ASIC	Eyeriss	200	FX16	34.7	0.3	124.8
FPGA	Kintex KU115	235	FX8	2252	22.9	98.3
FPGA	Kintex KU115	235	FX16	1126	22.9	49.2
FPGA	Zynq XC7Z045	200	FX8	340	7.2	47.2
FPGA	Zynq XC7Z045	200	FX16	170	7.2	23.6
GPU	Jetson TX2	1 300	FP16	250	10.7	23.3
GPU	Titan X	1 417	FP32	5120	227.0	22.6
CPU	Core-i7	3 500	FP32	162	73.0	2.2

B. TEMPORAL ARCHITECTURES

The *temporal architecture* (typical for CPUs and GPUs) employs a set of ALUs with a fixed connection pattern and a hierarchical memory subsystem. Because this architecture is primarily intended for general-purpose computing, it is in principle less energy efficient than specialized architectures with a dedicated data flow organization. When CPU and GPU implement DNNs, their performance can be increased by suitable algorithmic techniques and compiler-level optimizations whose goal is to reduce the number of expensive arithmetic operations, maximize the degree of parallel processing, and optimize the memory access pattern. Libraries such as MKL [69] and cuDNN [70] provide optimized algorithms for efficient computing of matrix multiplication, multi-dimensional convolution, Fast Fourier Transform, and other valuable operations for DNNs. GPUs, as the most popular platforms for DNNs, range from small devices (e.g., NVIDIA Jetson Nano with 472 GFLOPS and 5-10 W) to high-performance nodes of supercomputers (e.g., NVIDIA V100 with 100 TFLOPS and 300 W).

C. SPATIAL ARCHITECTURES

Spatial DNN accelerators that are usually implemented in ASICs or FPGAs consist of an array of *Processing Elements* (PE), on-chip buffers, a controller, and external DRAM memory (Fig. 5). Each PE contains a MAC circuit to multiply the input data with weight and add the product to a partial sum. A small local memory (register file or buffer) implemented in each PE can store local data such as weights, activations, and partial sums. Another memory, a global buffer, is used to prefetch from DRAM the activations and weights associated with a part of DNN that will be processed in the next step.

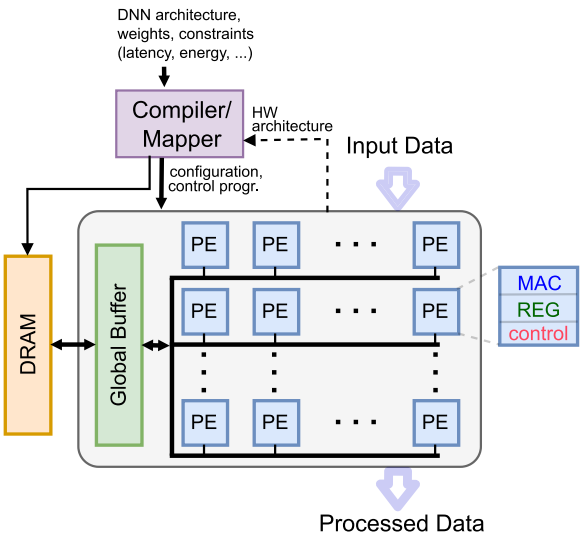

FIGURE 5. Typical organization of a DNN accelerator and its programming.

Dataflow is a general term covering the computation order, parallelization strategy, and tiling strategy applied in the accelerator. Tiles are chunks of data that fit the resources that are available to process them. For example, tiling of input feature map means that instead of loading an entire feature map, only a few rows and columns of that feature map are loaded and processed by the PE array. Dataflow is closely related to the data reuse strategy. The data elements (either weights, inputs, or partial sums) that have to be reused are mapped into local memory in PEs and kept there (stationary) until all relevant computations are performed with them. The common dataflow strategies are

- Weight Stationary, in which the weights are stored in PEs, e.g., used in [57].
- Output Stationary, in which the partial sums are stored in PEs. e.g., [62].
- Row Stationary, in which the weights are stored in PEs, and the operations of a row of the convolution are mapped to the same PE, e.g., [56].
- No Local Reuse, in which only the global buffer is used, e.g., [61].

The critical parameters of any DNN accelerator (i.e., the runtime, throughput, and energy efficiency) depend on a given DNN architecture and its mapping (via a suitable dataflow organization) to available resources. The data can be reused across time (via buffers) and space (over wires). The tile sizes are bound by buffer sizes within the accelerator. The total number of tiles depends on the DNN model size and the dataflow strategy. The total number of PEs in the accelerator determines the peak throughput [9].

Using cost models (such as MAESTRO [71]), the run time, resources utilization, delay, and power can be estimated for a given accelerator, DNN, and data set. A DNN is executed either layer-by-layer, or the entire DNN is pipelined across the accelerator, e.g., by means of systolic array principles. While the former approach is easier to schedule, it usually leads to less efficient utilization of resources. The latter approach requires a suitable dynamic partitioning of resources, which is challenging to manage for some DNN models, but the accelerator can be used more effectively. Some accelerators can effectively exploit data sparsity (many zeroes in the weights and activations) and configure an optimal bit width for arithmetic operations to improve performance [45], [56].

Depending on a given DNN, accelerator (specified in terms of architecture, available resources, and dataflow options), and constraints (such as the maximum latency), a compiler generates the control sequence for the accelerator. Many authors have addressed the optimal mapping and execution of the DNN on a given accelerator, see [55], [62], [71], [72]. The more challenging problem is to implement multiple accelerators on a single chip or to deploy multiple accelerators on multiple chips [13], [73]. Note that the search for an optimized implementation of a DNN for a given accelerator is one of the problems addressed by hardware-aware NAS (see Section VI).

D. APPROXIMATE IMPLEMENTATIONS OF CNNs

One of the most prominent approaches developed to reduce the power consumption of computer systems is *approximate computing* [74]. According to [75], the approximations were introduced into CNNs at the level of data type quantization, microarchitecture (e.g., pruning, weight sharing, and dataflow organization), MAC circuits, and memory (utilizing approximate memory cells, architecture, and weight compression). The RAPID AI accelerator was built from the ground up to investigate the impact of approximation techniques in CNNs. The authors revealed that the most significant gains are obtained when the cross-layer approximation approach is adopted, involving software, architecture, and hardware, breaking thus conventional methods focused on optimizing each layer of abstraction independently [45]. The NAS combined with hardware accelerator co-search has a great potential to perfectly solve this problem.

IV. SINGLE-OBJECTIVE NAS

Developing a high-quality DNN model for a given (previously unseen) data set and creating its implementation optimized for a target platform is a very time-consuming

task because it is inherently based on performing many experiments. The difficulty usually increases when the data set size grows, and challenging constraints (such as maximum-allowed latency or power consumption) are introduced. NAS methods were invented to automate this design process.

The whole NAS field started with the approaches in which only one objective, the *accuracy*, is optimized [3], see Fig. 6. Other objectives such as the number of parameters or FLOPS were not explicitly considered, but they have been often reported for resulting networks. Another goal, not explicitly formulated within the NAS methods, was to minimize the NAS execution time (or consumed energy or CO_2 emissions [76]). As we primarily deal with CNNs, the next paragraphs will mainly discuss CNN-oriented single-objective NAS.

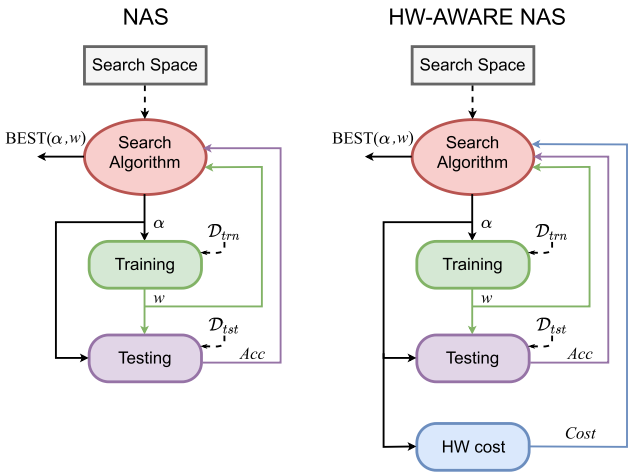

FIGURE 6. Single-objective NAS (left) and hardware-aware NAS (right). From the search space, the search algorithm samples a candidate DNN architecture α, which is trained to get the weights w, and tested to get the test accuracy *Acc*. The implementation cost is evaluated only for the hardware-aware NAS.

For given training and test data sets, $\mathcal{D} = \{\mathcal{D}_{trn}, \mathcal{D}_{tst}\}$, and loss function $\mathcal{L}$, the NAS problem can be formalized as a bilevel optimization problem [77]:

$$\begin{aligned} \alpha^* &= \text{argmin}\ \mathcal{L}_{D_{tst}}(\alpha, w^*(\alpha)) \\ &\text{s.t. } w^*(\alpha) = \text{argmin}\ \mathcal{L}_{D_{trn}}(\alpha, w), \\ &\alpha \in \Omega_\alpha, w \in \Omega_w, \end{aligned} \tag{6}$$

where the upper-level variable α defines a candidate CNN architecture, and the lower level variable $w(\alpha)$ defines the associated weights. Ω_α and Ω_w denote the space of CNN architectures and the space of CNN weights, respectively. $\mathcal{L}_\mathcal{D}(\alpha, w)$ is the cross-entropy loss on the data set $\mathcal{D}$ for architecture α and weights w.

The difficulty of the NAS problem lies in its complexity; the search space is enormous, and its dimension is variable. The search algorithm is often constructed as multi-objective (mandatory for hardware-aware NAS) and tries to balance the exploration and exploitation aspects. Moreover, because candidate networks are complex objects, their evaluation, which typically involves training, is very computationally expensive. The following sections survey the key principles of NAS algorithms.

A. SEARCH SPACE

A common practice is to model a candidate CNN using a directed acyclic graph encoded as a variable-length string. All possible strings describing valid CNNs constitute the search space. Three strategies for building the search spaces have dominated in recent years: (i) a macro search space which describes the entire CNN, (ii) a micro search space which defines the architecture of a subgraph (or several subgraphs) which is then repeatedly reused in the CNN, and (iii) a hierarchical search space.

In the case of the *macro search space* (Fig. 7a), the search space is determined by a set of possible operations for each node, hyperparameters of the network architecture, and a network template. The template can be a simple linear sequence of N nodes, or it can support branches and skip connections. Independent branches are at some point merged using a suitable operator such as concatenation or sum [3], [78]. Another option is to parameterize a well-known CNN (such as ResNet [31] or MobileNetV3 [33]) and use it as a template (e.g., in [7], [51], [79]–[81]) for building and constraining the search space. Some parts of CNN can be fixed by an engineer, and their implementation is then not subject to optimization. For example, the last fully connected layer is always present in a CNN-based classifier, and, hence, it usually makes no sense to search for its hyperparameters.

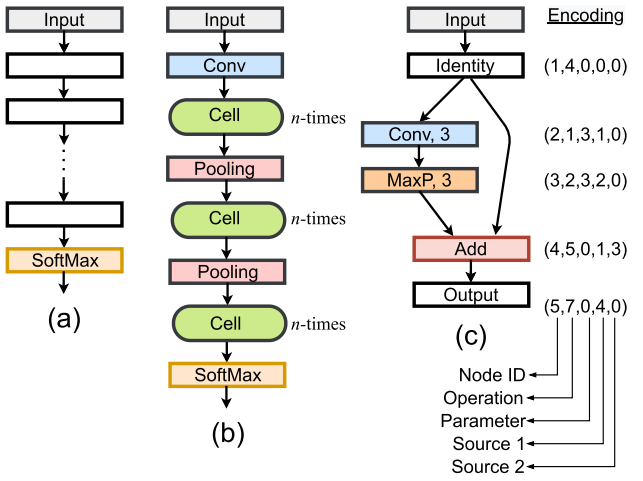

FIGURE 7. Construction of the NAS search space. (a) Macro: all layers (white boxes) are searched; (b) Micro: a cell is searched for a pre-defined CNN; (c) Example of cell's encoding according to [87] in which each node is defined by a five-tuple (Node ID; Operation; Parameter; Source ID 1; Source ID 2). The Operation is either (1) convolution, (2) max. pooling, (3) average pooling, (4) identity, (5) add, (6) concatenation, or (7) terminal node.

In the case of the *micro search space* (Fig. 7b), the architecture of a subgraph (also denoted as a cell, block, or segment) or several subgraphs is sought by NAS. This technique

effectively reduces the search space concerning the macro search. Each subgraph consists of several layers whose hyperparameters and connections have to be determined by NAS. The resulting subgraph(s) is/are then reused in the target CNN. For example, NASNet [82] proposes two types of cells: normal cell, which is used to extract advanced features, and reduction cell, whose task is to reduce the spatial resolution. Fig. 7c shows an example of a cell's encoding using a string of integers.

In the case of the *hierarchical search space* [83]–[86], a small set of primitives, including elementary operations like convolution, pooling, and identity is specified. Small sub-graphs (the so-called motifs) that consist of these elementary operations are then recursively used to establish the entire network. MnasNet [12] organizes target CNN into multiple sequentially connected segments, each having its separate repeating structure.

CNN can also be encoded *indirectly*, using a procedure (a generator) which generates it in a number of construction steps, e.g. [20], [88]. While indirect encoding can significantly reduce the search space size and produce complex networks it has not become popular in the current NAS. The reason is that it is tricky to devise a suitable unbiased generator for CNNs.

It is an open research problem of search space engineering to devise unbiased search spaces that can effectively be explored by search algorithms and, at the same time, enable the discovery of novel and competitive CNNs.

B. SEARCH ALGORITHM

Reinforcement learning (RL) and evolutionary algorithms (EA) dominate NAS methods. We will survey their principles; other relevant algorithms will be briefly mentioned. A special section is devoted to the differentiable NAS.

1) RL-BASED METHODS

In the pioneering work, Zoph and Le [3] used a recurrent network, the so-called *controller*, to sequentially generate vectors representing (hyperparameters of) candidate CNNs. The controller sequentially produces hyperparameters such as filter height and width, stride height and stride width, and the number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step of the RNN as input. After generating the entire CNN description, the candidate CNN is assembled, trained, and its validation accuracy serves as the reward signal to update the controller's parameters (the RNN's weights). Reinforcement learning thus tries to maximize the reward (validation accuracy) from the actions performed (decisions enabling to construct a candidate CNN) by the controller. The controller's parameters are iteratively updated by a policy gradient method such as REINFORCE [89]. Fig. 8 shows how a candidate CNN (including its architecture) can be generated. This approach or its various extensions were used in many NAS methods; see 'RL' in the following tables.

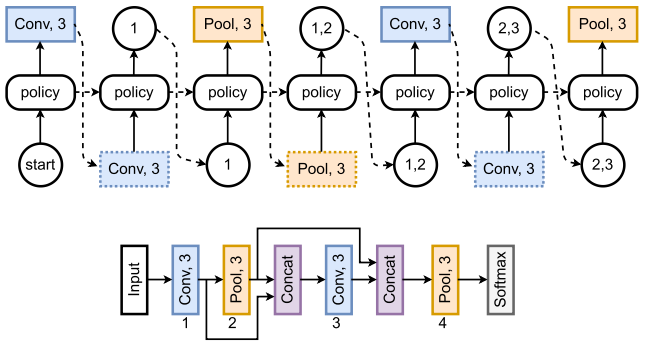

FIGURE 8. Top: Illustration of generating a CNN using the RL controller. The action selection consists of two stages: (i) the indexes of inputs and (ii) the operator over these inputs. Down: The resulting CNN.

2) EVOLUTIONARY SEARCH

Evolutionary algorithms were used for neural network design and optimization since the 1980s. Surveys [90], [91] provide an overview of the early methods developed not only for the architecture design but also for the optimization of the weights by EAs as a complement to gradient methods. The most exciting method developed in the pre-DNN era is the neuroevolution of augmenting topologies (NEAT) [92] which was quite competitive on small networks. NEAT was extended to CoDeepNEAT to produce DNNs through a co-evolutionary approach, with good results on CIFAR-10 and the Omniglot multitask learning domain problem [84]. A survey of recent neuroevolutionary methods was published in Nature [20].

The first study dealing with the evolution of undoubtedly complex CNNs was presented by Real *et al.* [93] who evolved a competitive solution to the CIFAR-10 problem. In their method, a candidate CNN is encoded as a graph whose nodes are rank-3 tensors or activations and edges are convolutions or identity connections. The initial population consists of 1000 single-layer networks with no convolutions. After their training and evaluation, parents are selected using a tournament selection, and offspring networks are then generated by mutation. The mutation operator is either adding or removing a layer, altering the hyperparameters of a layer, adding skip connections, or altering training hyperparameters. Whenever possible, learned weights and parameters are inherited from the parents to their offspring, which is called the *weight sharing*. These steps are repeated until a pre-defined number of generations is not exhausted. Real's EA works in the macro search space. While the EA is responsible for delivering the architecture of CNN, candidate CNNs are trained using a standard gradient descent algorithm.

Figure 9a summarizes the main steps of the evolutionary NAS method. The initial population is seeded either randomly or with existing models (to reduce the search time). Selection, recombination (mutation and crossover), and replacement are standard steps of a typical EA. As training is the most time-consuming step, various methods were developed to simplify it, see Section V-B.

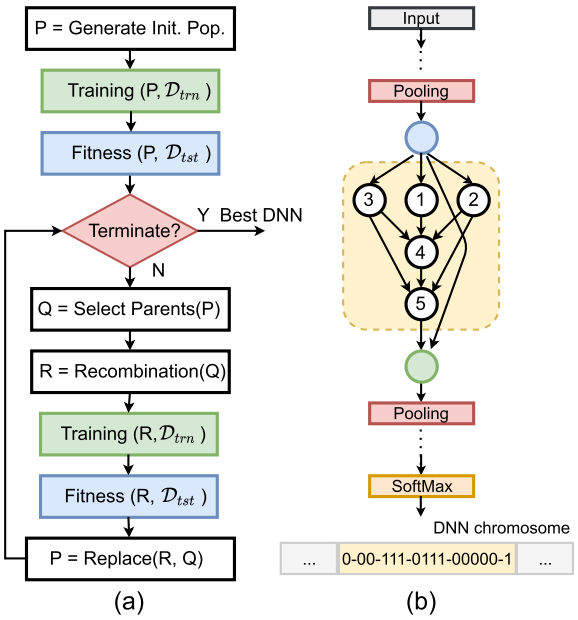

FIGURE 9. (a) Basic steps of the evolutionary design of DNNs. (b) Binary encoding of one stage in a pre-defined structure of CNN.

Candidate CNNs are typically encoded using strings of integers as illustrated in Fig. 7c. A very efficient binary encoding was proposed in GeneticNET [78]. A candidate network is composed of N stages, and each stage (the yellow box in Fig. 9b) contains up to K nodes. Fig. 9b shows that $j-1$ bits are devoted for encoding of the j-th node ($K=6$ in our example). Each of these $j-1$ bits determine if there is (1) or is not (0) a connection between the j-th node and nodes $1,\ldots,j-1$. The last bit informs if the skip connection is active. Each node can represent one of the pre-defined layers (convolution, poling, etc.) whose selection is encoded using one integer. The complete CNN encoding then consists of N parts, each of them devoted to one stage. The main advantage of this encoding is its compactness and the possibility of using a crossover binary operator, which proved to generate good offspring. This encoding was later reused in NSGANet methods [6], [94]. Note that genetic operators must be tuned for a particular encoding to produce good offspring. For example, [93], [95] only employ mutation operators; NSGANet methods [6], [94] utilize the aforementioned crossover.

AmoebaNet [96] provided the first large-scale comparison of EA and RL methods. Their simple EA searched over the same space as NASNet [82] and led to faster convergence to an accurate network when compared to RL and random search. Similar to the basic RL-based NAS methods, the main limitation of the EA-based NAS methods is their high computational overhead.

3) OTHER SEARCH STRATEGIES

Sequential Model Based Optimization (SMBO) is similar to the mutation-based EA. After generating and evaluating several smaller CNN models, it uses mutation to create more complex models. Their quality is predicted using a surrogate function, typically based on RNN. The surrogate function is updated using data collected from already evaluated networks. Having a large pool of candidate models, a selection strategy is needed to navigate in the search space. In PNAS, SMBO selects top-performing models based on predicted accuracy [97]. Note that in the *Monte Carlo Tree Search* (MCTS), a random selection is taken to choose which branch to expand for each node in the search tree [98].

Bayesian optimization methods (e.g., [85], [99]–[101]) employ a combination of a probabilistic surrogate model and an acquisition function to obtain suitable candidates. The acquisition function measures the utility by accounting for both the predicted response and the uncertainty in the prediction. The surrogate is constructed using the Gaussian process (GP), random forest, or similar methods. The idea is to limit the evaluation of the objective function by spending more time in choosing the most suitable candidates for the next step.

Training free approaches utilize results of theoretical analysis of DNNs. For example, TE-NAS [102] sorts candidate architectures according to a score obtained by analyzing the spectrum of the neural tangent kernel and the number of linear regions in the input space, which can be computed without training. The authors observed that these characteristics strongly correlate with the network's test accuracy.

Other algorithms, such as stochastic coordinate descent (SCD) [14], Metropolis-Hastings (M-H) [103], or Multivariate Information Geometric Optimization (MIGO) are employed less frequently.

C. SUPERNET AND ONE-SHOT METHODS

The authors of ENAS [104] observed that each candidate CNN could be seen as a subnetwork of a larger network. Hence, they constructed a generic network so that it is overparameterized, i.e., it contains all possible CNN realizations. It is also known as a *supernet*. Its nodes represent local computations, and the edges represent the flow of information. The nodes have their parameters, but they are used only when a particular node is activated. These parameters are *shared* among all subgraphs that can be sampled from the supernet. This idea is illustrated in Fig. 10. While it is time-consuming to train the supernet, obtaining a trained CNN (i.e., a subnetwork) from the supernet is computationally significantly less expensive as it requires to call a simple sampling algorithm. Sampled subnetworks require no training, or they are fine-tuned to improve their accuracy further.

The methods which train the network just once are also known as *one-shot* NAS methods. They can be applied for the micro as well as macro search space and combined with other optimization methods. Various extensions of this idea have been proposed, including hardware-aware NAS methods [7], [11], [86], [105], [106].

SinglePathNAS [107] considers all candidate convolutional operations as subsets of a single "superkernel".

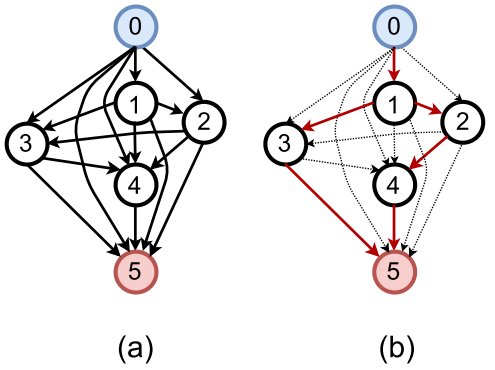

FIGURE 10. The principle of supernet. (a) Over-parameterized network whose nodes are local computations and the edges represent the flow of information. (b) One subnetwork sampled from the supernet which shares the weights (shown as red lines) with the supernet.

Rather than choosing among different paths/operations in the supernet, the NAS problem is solved by finding which subset of kernel weights should be used in each convolutional layer. By sharing the convolutional kernel weights, all candidate NAS operations are encoded into a single ''superkernel'', i.e., with a single path, for each layer of the one-shot NAS supernet. This approach allowed to reduce further the number of trainable parameters and the search time.

D. CONTINUOUS SEARCH SPACE AND GRADIENT SEARCH

Previously introduced methods operate in discrete search spaces and can be seen as black-box optimizers. They need a huge computational effort to discover an interesting CNN. In order to reduce computational requirements of NAS, DARTS [108] introduces a simple continuous relaxation scheme for the micro search space, which leads to a differentiable learning objective. The architecture and its weights can then be jointly optimized by a *gradient method* which is less computationally demanding than a black box optimizer.

A subnetwork (cell) is modeled as a directed acyclic graph consisting of N nodes with two input nodes and one output node. Each node $x^{(i)}$ is a potential feature map and each directed edge $(i; j)$ represents operation $o^{(i,j)}$ that transforms $x^{(i)}$. The output of the cell is calculated by a suitable reduction operation. Still considering a discrete space, each intermediate node is expressed as:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i,j)}). \tag{7}$$

To make the search space continuous, the categorical choice of a particular operation is relaxed using a softmax function:

$$\overline{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp(\gamma_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\gamma_{o'}^{(i,j)})} o(x), \tag{8}$$

where $\mathcal{O} = \{o_1, o_2, \ldots, o_k\}$ is a set of candidate operations (e.g., convolution, max pooling, zero) and $\gamma_o^{(i,j)}$ represents the weight of operation o on the edge connecting nodes i and j.

The architecture search was transformed to an optimization process for a set of continuous variables $\gamma_o^{(i,j)}$ representing encoding of the architecture. At the end of optimization, a discrete architecture of the cell is obtained by replacing each mixed operation $\overline{o}^{(i,j)}$ with the most likely operation $o^{(i,j)} = \text{argmax}_{o\in\mathcal{O}}\gamma_o^{(i,j)}$. The idea of DARTS is illustrated in Fig 11.

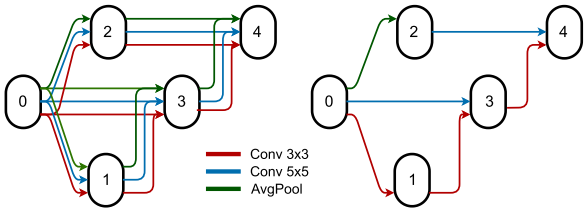

FIGURE 11. The principle of DARTS: continuous relaxation of the search space by placing a mixture of candidate operations on each edge (left), and selecting the resulting architecture after the joint optimization (right). Candidate operations are given in red, blue and green.

The parameters $\gamma = \{\gamma_o^{(i,j)}\}$ of the network architecture and network weights w are thus jointly optimized, i.e.

$$\min_{\gamma,w} \mathcal{L}_{\mathcal{D}_{tst}}(\gamma, w(\gamma)), \tag{9}$$

or form a bilevel optimization problem:

$$\begin{aligned} &\min_{\gamma} \mathcal{L}_{\mathcal{D}_{tst}}(\gamma, w^*(\gamma)) \\ &\text{s.t. } w^*(\gamma) = \text{argmin } \mathcal{L}_{\mathcal{D}_{trn}}(\gamma, w). \end{aligned} \tag{10}$$

Finally, parameters γ have to be discretized to obtain the final network architecture. DARTS enabled to reduce the search time to 4 GPU hours while delivering an accuracy comparable with other methods at that time.

Differentiable NAS has been quite often combined with the supernet approach. However, inconsistency in the performance of the parent network and the derived network was observed by many practitioners. One of the reasons is that DARTS jointly optimizes network weights and architectural parameters. At the same time, the subnetwork needs to optimize only a subset of weights for a few selected operations. Various approaches have been proposed to eliminate this behavior [109], [110]. P-DARTS employed the so-called progressive search to gradually increase the depth of the network during the search phase to avoid another problem associated with this approach – only shallow architectures are typically derived from the parent network [111].

E. SELECTED METHODS

Table 6 summarizes key properties of single-objective NAS methods. NAS methods are sorted according to the year of publication (of a regular paper even if a pre-print was published earlier) and then alphabetically. Every NAS method is characterized in terms of the *Search Algorithm*, *Search Space*, and the utilization of a *SuperNet*. Instead of using 'micro' to identify a micro search space, we use terminology taken from particular papers, i.e., 'cell', 'block', or 'stage', to characterize searched subnetworks. For ImageNet (ImgNet) and

TABLE 6. Single-objective NAS methods. The top-1 accuracy (Acc.) and parameters (Param.) are given for a CNN created by a particular NAS method and showed the highest accuracy in the corresponding paper. Symbol '–' denotes a non-reported value.

Method	Ref.	Year	NAS: Search		Super	ImgNet		C-10		Other
			Algorithm	Space	Net	Top-1 Acc.	Param. [$\cdot 10^6$]	Top-1 Acc.	Param. [$\cdot 10^6$]	datasets
NAS	[3]	2016	RL	macro		–	–	96.35	37.4	PTB
CoDeepNEAT	[84]	2017	EA	hierarchical				92.70	–	PTB, COCO
GeNET	[78]	2017	EA	stage		72.1	156.0	94.61	–	CIFAR-100, SVHN
MetaQNN	[112]	2017	RL	macro		–	–	93.08	11.2	MNIST, SVHN
ENAS	[104]	2018	RL	cell/macro		–	–	97.11	4.6	PTB
NASNet	[82]	2018	RL	cell		82.7	88.9	97.81	27.6	COCO
PNAS	[97]	2018	SMBO	block		82.9	86.1	96.59	3.2	
AMOEBA	[96]	2019	EA, RL	cell		83.9	469.0	96.66	3.2	
CGP-CNN19	[95]	2019	EA	global		–	–	95.10	2.7	CIFAR-100
DARTS	[108]	2019	gradient	cell		73.3	4.7	97.24	3.3	PTB, WikiText-2
MixNet	[113]	2019	RL	kernel		78.9	7.3	–	–	COCO
RENAS	[114]	2019	RL, EA	cell		75.7	5.4	97.12	3.5	COCO, PASCAL VOC
ShuffleNAS	[115]	2019	RL	cell		–	–	96.43	3.1	CIFAR-100
CNN-GA	[116]	2020	EA	macro		–	–	96.78	2.9	CIFAR-100
MS-RANAS	[117]	2020	gradient	cell		–	–	95.00	–	CIFAR-100
AS-NAS	[118]	2021	RL, EA	macro		–	–	97.77	16.6	NORB
DNAL	[119]	2021	gradient	channel	Y	75.0	3.6	94.30	1.2	
LaNAS	[98]	2021	MCTS, SMBO	cell	Y	80.8	8.2	99.01	44.1	
P-DARTS	[111]	2021	gradient	cell		75.9	5.4	97.75	10.5	CIFAR-100, COCO, 3 other
PC-DARTS	[110]	2021	gradient	cell	Y	75.9	5.1	97.45	3.2	
TE_NAS	[102]	2021	training free	cell	Y	75.5	5.4	97.37	3.8	
VINNAS	[120]	2021	gradient	cell	Y	–	–	96.06	1.8	MNIST, FMNIST

CIFAR-10 (C-10) data sets, we provide the best top-1 accuracy (and the number of parameters) presented in a particular paper, i.e., independently of the NAS method setup or used resources. When compared with human-created CNNs given in Table 3, NAS methods are quite competitive in terms of accuracy; however, only if the last generation of human-created CNNs (such as Vision Transformers) is not considered. We also list other data sets (denoted according to Table 4) that were employed to evaluate a particular NAS method. No performance indicators are reported for them to keep the table easily readable.

Please note that the objective of this survey is not to perform a detailed quantitative comparison of NAS methods. Despite some efforts towards a correct comparison methodology (see a discussion in Section IX), any comprehensive benchmarking of NAS methods (especially the multi-objective ones) has not been reported in the literature.

V. HARDWARE-AWARE NAS METHODS

Hardware-aware NAS methods were introduced to optimize neural networks not only for accuracy but also with respect to the target hardware platform, where the trained network is implemented. It was later shown by a detailed design space exploration [121] that optimal CNN architectures for different devices are not the same.

Typical objectives to be optimized for a given hardware platform are latency, throughput, energy efficiency, and memory usage. Hence, in specific steps of the NAS algorithm, all relevant objectives have to be evaluated, either by direct measurement on real hardware or estimated using software models (see Fig. 6). As the NAS algorithms are usually very time demanding, many techniques have been proposed for their acceleration, particularly for shortening the candidate network evaluation time.

The execution time of NAS is often seen as an additional objective to be optimized (minimized). For example, NAG [79] is a Pareto frontier-aware neural architecture generator that takes an arbitrary budget as input and produces the Pareto optimal architecture for the target budget.

In this section, we first introduce the principles of multi-objective optimization methods (Section V-A). Then, in Section V-B, we briefly survey the techniques developed to reduce the time needed to evaluate candidate designs. In Section V-C, we propose our classification of hardware-aware NAS methods.

A. MULTI-OBJECTIVE OPTIMIZATION

By extending the single-objective NAS formulation from Section IV, the NAS problem can be seen as a *multi-objective optimization problem*, i.e. an optimization problem that involves *multiple objective functions* f_i, $i = 1 \ldots m$ (all to be minimized, without loss of generality):

$$\begin{aligned} &\text{min. } f_1(\alpha; w^*(\alpha)), f_2(\alpha; w^*(\alpha)), \ldots, f_m(\alpha; w^*(\alpha)) \\ &\text{s.t. } w^*(\alpha) = \text{argmin } \mathcal{L}_{\mathcal{D}_{trn}}(\alpha, w), \end{aligned} \tag{11}$$

where the upper-level variable α defines a candidate neural network architecture, and the lower level variable $w(\alpha)$ defines the associated weights. One of the objective functions is typically loss on the test data.

In the multi-objective optimization, there does not usually exist one solution that minimizes all objective functions simultaneously because the design objectives are conflicting. Hence, rather than one (optimal) solution, the optimization results in a set of solutions, i.e. the solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives. Formally, a solution a is said to *(Pareto) dominate* another solution b, if $f_i(a) \leq f_i(b)$ for all $i \in \{1, 2, \ldots, m\}$ and $f_j(a) < f_j(b)$ for at least one

index $j \in \{1, 2, \ldots, m\}$, and all f_i have to be minimized. A solution a^+ is called a *non-dominated solution*, if there does not exist another solution that dominates it. The set of non-dominated solutions is called the *Pareto front*. We say that non-dominated solutions are *Pareto optimal solutions* if all possible candidate solutions are considered during the optimization, and there are no provably better non-dominated solutions in the search space. In practice, we are almost always faced with a situation in which a given method produces suboptimal solutions, i.e., the Pareto front contains the best non-dominated solutions obtained during the experiments conducted with the method. As it is not known ''how far'' the obtained solutions are from the truly Pareto optimal solutions, a common practice is to introduce a quality metric capable of measuring the distance between two sets of solutions obtained with two multi-objective optimization methods (see, for example, [122]) and compare them under this metric. For example, NSGANetV1 [94] employs the hypervolume performance metric, which calculates the dominated area (hypervolume, in the general case) from the set of solutions to a reference point which is usually an estimate of the nadir point – a vector concatenating worst objective values of the Pareto front.

A common approach to solve the multi-objective NAS problem adopted by the NAS community is either (i) to transform it into a single-objective one (using suitable constraints, prioritization, or aggregation techniques) and solve it with a common single-objective method or (ii) to employ a truly multi-objective approach (the so-called aposteriori methods) [123].

In the *constraints* utilizing method, only one of the objective functions is optimized while the remaining ones are expressed as constraints $h_i(a) \leq c_i$. A penalty function λ is then introduced to punish any violation of constraints c_i, i.e.

$$\min_a f(a) \cdot \prod_i \lambda(h_i(a), c_i). \tag{12}$$

For example, in order to constrain the latency (h_i), Tan *et al.* [12] defined the penalty function as

$$\lambda(h_i(a), c_i) = \left[\frac{h_i(a)}{c_i}\right]^p, \tag{13}$$

where p is treated as a hyperparameter controlling the desired tradeoff. However, this method does not guarantee that some hard constrains are not violated.

The *prioritization* means that the most crucial objective is optimized first. When a suitable solution is obtained, the second most crucial objective is optimized but ensuring that the first one is not worsened. This is repeated for all the objective functions according to their priority. The prioritization is also taken into account when multiple objectives are evaluated for a candidate solution a. First, the easiest-to-quantify objective is determined. If its value is not satisfactory, the remaining objectives are not evaluated, and a is discarded. Otherwise, the next objective is evaluated. For example, Smithson [103] first evaluated the number of MAC operations as it is easy to determine it, and if the candidate passes a certain limit, then its accuracy (whose obtaining requires time-consuming training) is assessed.

The *aggregation methods* introduce a suitable aggregation function (such as the weighted sum, weighted exponential sum, or weighted product) for the objective functions and optimize the composition. In the case of the linear weighted sum, the new objective function is

$$f^A(a) = \sum_{i=1}^{m} v_i \cdot f_i(a) \tag{14}$$

where v_i is the weight of the i-th objective and $\sum v_i = 1$. This approach suffers from several problems. First, it is not easy to find suitable values of v_i. Second, the linear weighted sum only works for problems with convex Pareto fronts, i.e., solutions on non-convex segments are unreachable. Third, similar to the constrained optimization, the method has to be executed several times with different weight settings to approximate the Pareto optimal front.

Truly *multi-objective optimization* methods iteratively build the Pareto front in the course of optimization by comparing candidate solutions using the non-dominance relation, promoting reasonable solutions, and trying to cover the expected Pareto front. This approach has significantly been developed within the evolutionary computation community. One of the most popular methods is NSGA-II [124]. It is based on sorting individuals in a population according to the dominance relation into multiple fronts. The first front contains all non-dominated solutions. Each subsequent front is constructed by removing all the preceding fronts from the population and finding a new Pareto front in the remaining individuals. The solutions within the individual fronts are then sorted according to the crowding distance metric. This metric helps to preserve the diversity of the population along the fronts. Best individuals then serve as parents for the new population. NSGA-II thus always produces a set of non-dominated solutions (i.e., a Pareto front) when it is terminated.

B. SHORTENING THE EVALUATION TIME

This section deals with techniques developed to reduce the evaluation time of candidate neural networks.

1) ACCURACY ESTIMATION

The quality of a candidate CNN architecture is typically obtained by training the CNN using a training data set and then measuring the final accuracy on the test set. Common strategies introduced to reduce the training time are decreasing the number of epochs and employing a proxy training data set (e.g., in [105]). The learning curve can also be extrapolated to estimate the performance of training. The extrapolation is based either on the number of iterations (or training time), or the size of the available data set for training. MetaQNN [112] compares the performance of the candidate CNN after the first training epoch with the

performance of a random predictor to check if it is helpful to decrease the learning rate and restart training. Large-scale Evolution [93] enables the CNN obtained from a mutation to inherit the parent's weights whenever possible.

Although the authors of ProxylessNAS [11] argue that architectures optimized on proxy tasks are not guaranteed to be optimal on the target task, many successful NAS methods developed after ProxylessNAS used surrogate models. The accuracy is predicted using neural networks, classification trees, regression trees, or Gaussian Process [6], [103], [125]–[127]. In general, surrogate models replace expensive objectives with pre-trained models that provide desired approximation.

2) LATENCY AND OTHER HARDWARE PARAMETERS

First multi-objective hardware-aware NAS methods have considered the number of parameters, FLOPS, and MACs as the additional objectives to optimize because they are not expensive to quantify for each candidate CNN. Later, when CNNs were adopted for hardware accelerators, and it was necessary to consider real latency and area overhead that are expensive to quantify exactly, various proxy measures were proposed to reduce the computational effort. We will deal with latency in the next paragraph (as most papers on NAS do), but other parameters can be estimated similarly. Latency can be estimated using:

- a surrogate model, e.g., [6], [126], [128]–[131];
- a suitable hardware simulator executing a candidate CNN, e.g. [51], [100], [132]–[135];
- a formula or model derived after analyzing the search space of possible CNN architectures, e.g., [14], [47], [49], [101], [136]–[138];
- a LUT-based model [105], [139], [140].

In LUT-based models, latency (and other parameters of interest) is stored to a LUT for each possible operation (e.g., a neuron, a convolution layer) from the space of all CNN architectures. These LUT values are either obtained by measurement on real hardware or estimated according to the data sheets for a given implementation technology. The total latency is then estimated using LUT values for the operations on the longest path from the input to the output of the CNN model.

C. NAS FOR PARTICULAR HARDWARE

An obvious approach to optimizing the CNN architecture for given hardware is employing only hardware-friendly hyperparameters and operations (suitable convolution types, arithmetic operator implementations, quantization schemes, or memory access mechanisms). For example, based on benchmarking 32 different operators, Hurricane [131] uses different subsets of operator choices for three types of hardware platforms. This way, the search space is narrowed towards CNN architectures suitable for a given hardware platform. An essential feature of the hardware-aware NAS methods is that they do not directly optimize the configuration of the hardware platform, i.e., *there is no additional search space* that can be explored to improve the implementation further.

1) PROPOSED CLASSIFICATION

Table 7 surveys key properties of hardware-aware NAS methods and classify them according to several criteria. If a method is not presented under any abbreviation in the source paper, we identify it in the table according to the first author. Some of the criteria are identical with respect to our classification introduced for the single-objective methods in Table 6 (i.e., *Search Space*, *Search Algorithm*, *SuperNet*). It can be seen that the newest methods frequently employ the concept of supernet, allowing them to reduce their execution time.

The search algorithms optimize the design objectives that are listed in the *Objectives* column, together with the accuracy, which is not mentioned as it is always involved. The *Estimation Method* column tells us if at all and how particular hardware parameters are estimated (the methods are abbreviated according to Section V-B). We observe that latency (Lat) and Energy are often estimated rather than measured. According to [11], [67], the number of parameters (weights) or FLOPS is not a good proxy for latency for complex CNNs. However, when smaller CNNs are developed for tiny MCUs, the number of parameters or FLOPS are often used for this purpose. The reason is that CNNs are executed on a processor with limited options for parallel processing and, hence, the correlation between the CNN complexity and execution time is high [99], [141], [142]. If the accuracy (Acc) is estimated, then a NN-based predictor (surrogate) is almost always utilized for this purpose [6], [103], [125], [126]. However, for example, ChamNet [127] utilizes a Gaussian Process-based surrogate.

While the single-objective NAS methods have been almost exclusively oriented to GPUs, the hardware-aware NAS methods target on all major hardware platforms, including:

- GPU, e.g. [6], [140], [143];
- FPGA, e.g. [101], [144], [145];
- ASIC, e.g. [126], [134], [135], [146];
- TPU, e.g. [133], [147];
- MCU, e.g. [99], [141], [142], [148];
- DSP, e.g. [131], [149];
- Mobile phones, e.g. [11], [12], [33], [105].

Moreover, about one-third of the NAS methods were evaluated on two or more platforms.

In order to provide basic information about the performance, we again present only the best top-1 accuracy reported in a particular paper for ImageNet (ImgNet) and CIFAR-10 (CF-10) and list other data sets used for evaluation. Compared to the single-objective NAS methods, it is even more challenging to present a fair quantitative evaluation. Section VIII will provide a comparison for Pixel 1 phone as the target platform. If accuracy is provided for ImageNet (i.e., the ImgNet column is not empty), the NAS method aims at solving complex problems; otherwise, it is focused on less

TABLE 7. Hardware-aware NAS methods.

Method	Ref.	Year	NAS: Search Algorithm	NAS: Search Space	Super Net	Objectives	Estimation Method	Target device	Data set (top-1 accuracy [%]) ImgNet	C-10	Other
DSE	[103]	2016	M-H	macro		Normalized Cost	Acc: NN; Cost: MAC, Mem	GPU		86.00	MNIST
Large-Scale	[93]	2017	EA	macro		FLOPS	None	GPU		94.60	CIFAR-100
DNAS	[81]	2018	gradient	block	Y	Cost	Cost: Param×BitWidth	ASIC	74.6	95.07	
DPP-Net	[125]	2018	SMBO	cell		Lat, Mem, Params	Acc: NN	CPU, GPU, Mobile	75.8	95.64	
HyperPower	[150]	2018	Bayes, RS	hyperp		Power	Power: linear model	GPU		78.19	MNIST
JASQNet	[151]	2018	EA	cell		params	None	GPU	72.8	97.10	
MONAS	[143]	2018	RL	macro		Energy, Power	NVIDIA profiling tool	GPU		95.50	
PPP-Net	[152]	2018	SMBO	block		Lat, Params, FLOPS	Acc: NN	GPU		95.64	
RENA	[153]	2018	RL	macro, block		Params, FLOPS	None	GPU		97.05	KWS
ECAD	[144]	2019	EA	hyperp		Lat, Energy	simulator	FPGA			MNIST
FBNet	[105]	2019	gradient	macro	Y	Lat	Lat: LUT	GPU, Mobile	74.9		
Gong et al.	[154]	2019	gradient	block		Energy	simulator	GPU, ASIC	71.8		CIFAR-100
ChamNet	[127]	2019	EA	hyperp		Lat, Energy	Acc, Energy: GP predictor	GPU, DSP, Mobile	75.4		
LEMONADE	[155]	2019	EA	macro, cell		Params	None	GPU		97.42	ImageNet(64x64)
MNASNet	[12]	2019	RL	block		Lat	None	mobile	76.7		COCO
MobileNetV3	[33]	2019	RL	block		Lat	None	mobile	75.2		COCO, Cityscapes
NSGANetV1	[94]	2019	EA, Bayes	block		FLOPS	None	GPU		96.15	CIFAR-100
ProxylessNAS	[11]	2019	gradient	macro	Y	Lat	Lat: LUT	GPU, CPU, Mobile	75.1	97.92	
SNAS	[109]	2019	gradient	cell		Params	None	GPU	72.7	97.15	
SpArSe	[99]	2019	Bayes	macro		Params, Mem	formula	MCU		73.84	MNIST, CUReT, Chars74k
TEA-DNN	[100]	2019	Bayes	cell		Lat, Energy	None	GPU, Movidius		93.10	
APNAS	[146]	2020	RL	block		Lat	Lat: cycle count	ASIC		93.75	CIFAR-100
APQ	[126]	2020	EA	block	Y	Lat, Energy	Acc: NN; Lat: LUT	ASIC	75.1		
Cassimon et al.	[156]	2020	RL	cell/macro		Lat, Mem, Cache	Lat: LUT	Raspberry Pi			non-standard
DeepMaker	[157]	2020	EA	hyperp		Size	None	CPU, GPU, FPGA		93.10	MNIST, CIFAR-100
Gupta et al.	[133]	2020	RL	block		Lat	Lat: simulator/formula	TPU, Mobile	75.6		
HNAS	[83]	2020	EA	macro		Lat	Lat: LUT	GPU, Mobile	76.1		
HTAS	[158]	2020	gradient	macro		Lat	Lat: model	CPU, GPU		94.33	CIFAR-100
Hurricane	[131]	2020	EA	macro	Y	Lat	Lat: Bayes. Regression	DSP, CPU, Movidius	76.7		
MCUNet	[148]	2020	EA	macro	Y	Lat, Mem, Flash	None	MCU	70.7		WWV, KWS
MemNAS	[159]	2020	RL	block		Mem	formula	Mobile	75.4	95.70	
MNASFPN	[160]	2020	RL	block		Lat	Lat: LUT	Mobile			COCO
MoArr	[161]	2020	RL	block		Params	None	GPU	76.0	97.39	
NAGO	[85]	2020	Bayes	hierarchical		Mem	Acc: NN	GPU	76.8	96.60	CIFAR-100, SPORT8 +2
NasCaps	[135]	2020	EA	hyperp		Lat, Mem, Energy	Lat: cycles; Energy: model	ASIC		85.99	MNIST, FMNIST, SVHN
NSGANetV2	[6]	2020	EA	block	Y	Lat, MAC, Params	Acc: ML-surrogate	GPU	80.4	98.40	CIFAR-100, 6 other sets
OFA	[7]	2020	gradient	block	Y	Lat	Acc, Lat: NN	GPU, FPGA, Mobile	80.0		
PONAS	[106]	2020	EA	macro	Y	Params, FLOPS	LUT	GPU	75.2		
RNAS	[138]	2020	gradient	macro	Y	Lat	Lat: anal. model	FPGA		94.18	
S3NAS	[147]	2020	gradient	block	Y	Lat	Lat: cycle-level simulator	TPUv3	82.7		
Schorn et al.	[162]	2020	EA	hyperp		Lat, Energy, FT	formula	GPU		93.48	GTSRB
SPOS	[163]	2020	EA	blocks	Y	Lat, FLOPS	None	GPU	75.3		
SinglePathNAS	[107]	2020	gradient	macro	Y	Lat	Lat: per-layer model	GPU, Mobile	74.9		
TF-NAS	[140]	2020	gradient	macro	Y	Lat	Lat: LUT	GPU	76.9		
TuNAS	[80]	2020	RL	macro	Y	Lat	None	Mobile	75.4		COCO
μNAS	[142]	2021	EA	macro		Lat, Mem, MAC	Lat: MAC	MCU		86.49	MNIST, Chars74K, etc.
E-DNAS	[149]	2021	gradient	block		Lat	Lat: LUT	DSP	76.9		PascalVOC2007
HardCoRe-NAS	[86]	2021	gradient	hierarchical	Y	Lat	Lat: formula	GPU, CPU	78.0		
HSCoNAS	[137]	2021	EA	block	Y	Lat	Lat: formula	GPU, CPU	77.6		
Lyu et al.	[164]	2021	RL	block	Y	Lat	None	GPU	73.7		
MDARTS	[134]	2021	gradient	cell	Y	Lat	Lat: simulator	ASIC		97.65	CIFAR-100
MicroNets	[141]	2021	gradient	block	Y	Lat, Energy	Lat, Energy: OP count	MCU			VWW, KWS, MIMII
MIGO-NAS	[165]	2021	MIGO	macro		Params	None	GPU	78.3	97.35	
NAG	[79]	2021	RL	block	Y	Lat	Acc: NN	GPU, CPU, Mobile	80.5		
NAS_Edge	[101]	2021	Bayes	cell		Lat	Lat: formula	FPGA	65.2	95.37	MNIST
NetAdaptV2	[166]	2021	RL	block	Y	Lat, MAC	None	GPU, Mobile	78.5		
Prabakaran et al.	[167]	2021	EA	macro		Mem	None	GPU, CPU			anomaly in ECG signals
Rmobile-Net	[168]	2021	gradient	hierarchical	Y	Lat, Cost	Lat: LUT	Mobile		96.56	
Wang et al.	[169]	2021	EA	block		FLOPS	None	GPU			IDS2012, ISCX VPN
Yang&Sun	[145]	2021	RL	macro		Lat	Lat: formula	FPGA		95.55	

challenging problems such as CIFAR-10, or even MNIST only.

2) SELECTED METHODS

The NAS methods surveyed in Table 7 show series of innovative approaches which, since the year 2016, have enabled the improvement of state-of-the-art results continuously. The improvements are in: (1) providing a better trade-off between the accuracy and latency (or other hardware parameters) on various hardware platforms and (2) reducing the design time and resources needed to achieve innovative solutions. Because of the space limitation, we briefly survey only some of the hardware-aware NAS methods in the following paragraphs.

The first genuinely multi-objective hardware-aware NAS is DSE [103] in which a candidate DNN is optimized using an adapted Metropolis-Hastings (M-H) algorithm. Its accuracy is predicted by an MLP which reads the DNN's hyperparameters. The second objective is a normalized cost evaluating the number of MAC operations and memory accesses carried out during inference. The MLP predictor is very accurate because its mean error is only 0.35%. A limitation of the method is that DSE's maximum accuracy on CIFAR-10 (86%) is very low compared to current approaches.

DNAS [81] is a framework based on a differentiable NAS capable of selecting the most suitable number of bits for FX operations conducted in each block of the CNN. DNAS creates a supernet whose layers contain several parallel edges representing convolution operators with quantized weights and activations with different precisions. As all layers in one block use the same precision, the search is conducted at the block level. If the precision is 0, the block is skipped, which changes the size of CNNs. It is assumed that the underlying hardware supports this type of quantization.

MNASNet [12] uses a factorized hierarchical search space that provides CNN architectures suitable for hardware accelerators. The layers are grouped into blocks based on their input resolutions and filter sizes. The RL-based search is constrained by maximum latency, which is directly measured on a mobile phone. However, the method requires 40 thousand GPU hours to produce a CNN, which is not competitive today.

In FBNet [105], the NAS algorithm first trains a stochastic supernet using SGD to optimize the architecture distribution. Using the LUT-based approach, a differentiable loss function is created for latency. This innovation allows one to use gradient-based optimization to solve the NAS problem together with optimizing latency. State-of-the-art trade-offs between the accuracy and latency were reported on ImageNet for several mobile phones. A limitation is that FBNet searches on a "proxy" dataset (i.e., a subset of the ImageNet dataset) and the entire supernet must be maintained in memory during the search.

ProxylessNAS [11] is another differentiable NAS in which an over-parameterized network that contains all candidate paths is trained. To guarantee its fidelity, no proxy such as reduced training data sets or shorter training periods are allowed. Specialized architecture parameters are introduced to learn which paths of the net are redundant. These parameters effectively switch off redundant parts of the network. To handle non-differentiable objectives such as latency during learning, network latency is modeled as a continuous function and optimized as regularization loss.

NSGANetV2 [6] extends the NSGANet [94]. Both methods work in the search space proposed by GeneticCNN [78] but employ a truly multi-objective evolutionary algorithm NSGA-II. Instead of gradient-based relaxations used in FBNet and ProxylessNAS, it builds surrogate models to predict the accuracy of candidate CNNs. It also uses a supernet trained with a progressive shrinking algorithm and weight sharing to reduce the training time. One nine data sets, including ImageNet, NSGANetV2 improved the state-of-the-art results.

In order to effectively develop CNNs for different accelerators with different latency constraints, OFA [7] proposes to train a once-for-all (OFA) network that supports diverse architectural settings. Its training is expensive (1 200 GPU hours on V100 GPUs) but is amortized. As different sub-networks are interfering with each other, the training process of the whole OFA network is inefficient. Hence, instead of directly optimizing the OFA from scratch, it is proposed to first train the largest CNN with maximum depth, width, and kernel size; and then progressively fine-tune the OFA network to support smaller sub-networks that share weights with the larger ones. Specialized sub-networks for diverse hardware platforms (from the cloud to the edge) and various constraints were derived from OFA using a pre-trained predictor in constant time.

APQ performs a joint search for architecture, pruning, and quantization policy using an evolutionary algorithm [126] starting with the MobileNetV2 network. The accuracy is predicted using a quantization-aware predictor implemented as a three-layer feed-forward NN. The input to the predictor is the encoding of the network architecture, the pruning strategy, and the quantization policy. The predictor is first trained without quantization, then transfers its weights to train the quantization-aware predictor, which largely reduces the data collection time. The latency and energy of each layer are pre-computed and stored in LUTs.

HTAS [158] first expands the global search space. The reason is that more efficient architectures can be wider than the original network structure in some layers, and it would be impossible to find them in the limited search space. Then, based on the latency measurements over the channel numbers, the hardware-friendly channel choices are selected to construct the hardware-aware search space. A differentiable NAS with the latency regularizer is then employed to seek the most suitable CNN for the target CPU or GPU, including the optimal selection of its hyperparameters.

MicroNets [141] are devoted to resource-constrained microcontrollers. They exploit a specialized software called TinyML, allowing ML tasks to be implemented on IoT devices. MicroNets are optimized for MCU inference performance using differentiable NAS with constraints on latency and memory (both SRAM and eFlash sizes are considered). The authors observed that that the number of operations is a viable proxy for both latency and energy. For inference, MCUs predominantly use 8-bit FX operations. However, MicroNets support sub-byte quantization on 4 bits. A similar approach, but targeting even smaller microcontrollers, was presented in μNAS [142].

VI. NAS WITH HARDWARE CO-DESIGN

When the hardware-aware NAS is connected with a hardware co-design algorithm, the CNN accelerator can be co-optimized with the CNN architecture. The authors of [13] observed that

"*the hardware-aware NAS has a much narrower search space than the proposed co-exploration approach. Basically, hardware-aware NAS will prune the architectures with high accuracy but fail to meet hardware specifications on fixed hardware design. However, by opening the hardware design space, it is possible to find a tailor-made hardware design for the pruned architectures to make them meet the hardware specifications. Therefore, compared with the HW-aware NAS, the co-exploration approach enlarges the search space.*

TABLE 8. NAS methods with hardware co-design.

Method	Ref.	Year	NAS: Search Algor.	NAS: Search Space	Super Net	Objectives	Multi-objective strategy	Accelerator Co-design Search Alg.	Accelerator Co-design Search Space (key parameters)	Quant	Estimation Method	Target device	Data set (top-1 accuracy [%]) ImgNet	C-10	Other
Common Platforms															
AutoDNN	[14]	2019	SCD	hyperp		Lat, Resources	Constr	in NAS	parallelization factor, quantization	Y	formula	FPGA			DAC'2018 comp.
Lu et al.	[47]	2019	RL	macro		Lat, LUT	co-search	DP	tiling, partition of layers	Y	formula	FPGA		82.98	
QNAS	[51]	2019	EA	block	Y	EDP	co-search	EA	#PE, mem. params	Y	simulator	ASIC	77.3	91.00	
CodesignNAS	[139]	2020	RL	cell		Lat, Area	Agg., Constr	in NAS	mem. params, parallelism degree		LUT	FPGA			CIFAR-100
DNA	[15]	2020	gradient	macro	Y	FPS,Lat,EDP	co-search	gradient	#PE, mem. params, DataFlow, tiling		Lat: simulator	FPGA, ASIC	75.7	96.50	CIFAR-100
EDD	[48]	2020	gradient	block		Lat,Resources	Agg.	in NAS	parallelization factor	Y	Lat: formula	GPU, FPGA	74.7		
HotNAS	[49]	2020	RL	macro		Lat	Agg., Constr	in NAS	mem. params, #PE, tiling, bandwidth	Y	Lat: formula	FPGA	73.4	97.13	
YOSO	[130]	2020	RL	cell	Y	Lat, Energy	Agg.	in NAS	#PE, mem. params, Dataflow		GP	ASIC		96.95	
DANCE	[170]	2021	RL	cell		EDA	Agg.	in NAS	#PE, mem. params, Dataflow		Lat: simulator	ASIC	70.4	95.00	
Liang et al.	[136]	2021	gradient	cell	Y	Lat, LUT, DSP	Agg.,Constr	in NAS	parallelization factor, buffering factor		Lat: formula	FPGA	73.3		
NAAS	[8]	2021	gradient	block	Y	EDP	co-search	EA	#PE, mem. params., compiler mapping		simulator	TPU, ASIC	79.0	93.20	
NAHAS	[129]	2021	RL	block	Y	Lat, Area	Agg.,Constr	in NAS	# PE, #SIMD units, mem. param.		Lat: NN	TPU	79.5		Cityscapes, Segm.
Pinos et al.	[171]	2021	EA	macro		Energy, Params	Pareto	in NAS	approximate multiplier type		formula	ASIC		83.98	
Multiple Models/Accelerators															
FNAS	[172]	2019	RL	hyperp		Lat	Agg., Constr	none	none		Lat: formula	FPGA		96.11	MNIST
ASICNAS	[73]	2020	RL	macro		Lat, Energy, Area	Agg., Constr	ILP, in NAS	accelerator type, #PE, bandwidth		simulator	ASIC		93.23	STL-10, Nuclei
HWSW-CoExp	[13]	2020	RL	macro		Lat	Agg.	in NAS	partitioning, assignment		Lat: model	FPGA	70,2	85.19	
Multi-HW	[128]	2021	RL	block	Y	Lat	Agg.	none	none		lin. model	TPU, GPU, DSP	75.8		
Unconventional platforms															
PABO	[173]	2019	Bayes	hyperp		Energy	Pareto	none	none		simulator	Memristive		86.00	Flower-17
NACIM	[50]	2021	RL	hyperp		Lat, Energy, Area	Agg.	in NAS	device type, circuit topology	Y	simulator	In-Memory		73.90	
NAS4RRAM	[132]	2021	EA	cells		Energy	Agg.,Constr	none	none		simulator	RRAM		84.40	CIFAR-100

As a result, it can make better trade-offs between accuracy and hardware efficiency."

Table 8 surveys major NAS methods utilizing hardware co-design. In addition to the columns used in Table 6 and 7, we added some new columns that characterize these methods. Their meaning will be defined in the next paragraphs.

Hardware platforms can be configured in multiple dimensions, including the PE array size, MAC circuit configuration, dataflow organization, tiling strategy, memory subsystem size and organization, and preferences for high-level synthesis software. In addition to the architecture search space and parameter search space (weights), there is an additional search space, called the *hardware search space*, containing all possible hardware configurations. The *Accelerator Co-design: Search Space* column shows the major hardware parameters optimized by a given NAS method.

The space of hardware configurations can be searched together with the space of DNN architectures using the same search algorithm, such as in [14], [48], [49], [129], [130], [139], [170]; see also 'in NAS' in the *Accelerator Co-design: Search Alg.* column of Table 8. However, another option for optimizing the hardware configurations is to use an independent search algorithm such as dynamic programming (DP) in [47], EA in [8], [51], gradient search in [15], or integer linear programming (ILP) in [73] as seen in the *Accelerator Co-design: Search Alg.* column of Table 8. The *Quant* column indicates that the method is also searching for a suitable quantization scheme.

In the *Multi-objective: strategy* column, the 'co-search' means that there are two independent search algorithms, i.e., a co-search is conducted; one search algorithm operates in the network architecture space and the other in the hardware search space. The *Multi-objective strategy* is based either on a Pareto front construction method ('Pareto'), aggregation method ('Agg'), or applying some constraints ('Constr').

Compared to previous tables, new objectives are defined in the *Objectives* column: Energy-Delay Product (EDP), Frame-Per-Second (FPS), and Energy-Delay-Area product (EDA). The meaning of *Data set* column is the same as in Table 7.

A. CO-SEARCH ORCHESTRATION

A straightforward approach to organizing the co-search is to generate a CNN-accelerator pair, which is evaluated by training the network to obtain its accuracy and measuring the hardware parameters (Fig. 12). Based on this evaluation, the next candidate pairs are generated until the desired solution is not obtained. However, this general approach leads to a time-consuming search process due to the prohibitively huge joint space composed of the coupled yet different network and accelerator spaces with extremely sparse optima. To reduce the design time, a supernet is often constructed (e.g., in [129], [130]) before starting the co-search and then sampled to quickly obtain a candidate CNN and its accuracy.

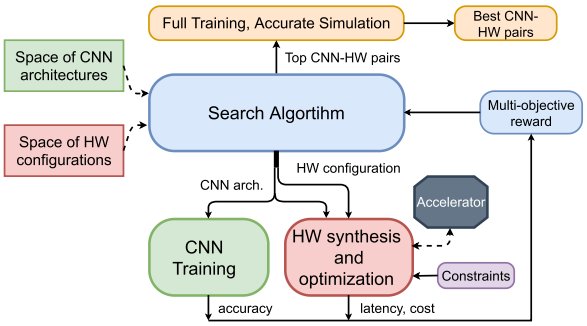

FIGURE 12. NAS and hardware co-design utilizing a single search algorithm operating in the architecture space and hardware configuration space.

In another co-search strategy used by, e.g., in [47], [51], the hardware optimization algorithm receives a CNN as the input and optimizes the hardware accelerator concerning desired objectives (Fig. 13). Suppose the accelerator optimizer produces a valid hardware configuration (i.e., all constraints are satisfied). In that case, the original CNN can undergo full training, and its accuracy, together with the hardware cost, is sent back to the NAS algorithm to generate another candidate CNN. Otherwise, the original CNN is discarded and a new one is generated. This strategy exploits the fact that the

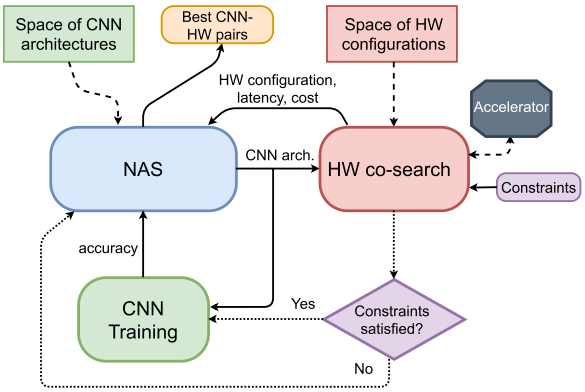

FIGURE 13. NAS and hardware co-design utilizing two search algorithms. A candidate CNN is fully trained only if the hardware co-search algorithm is able to find an accelerator configuration satisfying all constraints.

NAS performing network training is more computationally expensive than hardware synthesis and optimization.

Following the idea of differentiable NAS in DARTS [108] and hardware-aware differentiable NAS in FBNet [105] and ProxylessNAS [11], differentiable network-accelerator co-search framework was proposed in EDD [48], and later in DNA [15]. Let us use DNA to explain the method.

DNA enables co-searching for the CNN architecture together with the accelerators' configuration (e.g., the PE array size, the local and global buffer sizes, dataflow) and the mapping method (e.g., loop tiling strategy and loop size/order). DNA consists of two search algorithms: (1) the Differentiable Accelerator Search (DAS) in a generic accelerator design space, and (2) the Differentiable Network Search (DNS) based on FBNet [105]. In each iteration, the global co-search algorithm samples M networks from the current network distribution $NET(\alpha)$ and obtains the optimal accelerator for each of them using DAS. In order to continue the search in the CNN architecture space by DNS, the hardware cost loss is needed. It is obtained as an average hardware cost for each operator on the M optimized accelerators generated from the previous step. The optimization tasks performed by DNA can be formalized according to [15] as follows:

$$\begin{aligned} &\min_{\alpha} \ \mathcal{L}_{\mathcal{D}_{tst}}(\omega^*, NET(\alpha)) + \lambda \mathcal{L}_{hw}(NET(\alpha), HW(\gamma^*)) \\ &\text{s.t. } \omega^* = \arg\min_{\omega} \mathcal{L}_{\mathcal{D}_{trn}}(\omega, NET(\alpha)), \\ &\text{s.t. } \gamma^* = \arg\min_{\gamma} \mathcal{L}_{hw}(NET(\alpha), HW(\gamma)), \end{aligned} \tag{15}$$

where ω, α, and γ are the supernet weights, DNN architecture parameters, and the accelerator parameters, respectively; $NET(\alpha)$ and $HW(\gamma)$ denote the network and the accelerator space parameterized by α and γ, respectively. $\mathcal{L}_{hw}$ is the hardware-cost loss determined by both the network and its accelerator. The accelerator is characterized by its parameters $\gamma^S (s = 1, \ldots, S)$, which is a normalized vector representing the s-th accelerator parameter with each element of γ^S defining the probability of the corresponding choice of its represented accelerator parameter.

B. SELECTED METHODS

In order to demonstrate how NAS and hardware co-search can operate together, we briefly present the most interesting methods of this category.

Lu *et al.* [47] introduce a joint exploration of the space of neural architectures, FPGA implementations, and layer-wise quantization. The RL controller samples parameters of a candidate CNN architecture and its possible quantization. For the sampled network, the hardware builder searches the hardware space to find a suitable hardware model. Each candidate hardware model is validated against the specification (latency constraint) during the search, and the result is sent back to the controller. If there is a valid FPGA model, the sampled quantized CNN is trained, and its accuracy also serves as feedback to the controller. The hardware search space is determined by tiling parameters and partitioning the layers of CNN into clusters of the tile-based FPGA accelerator that are sought by a dynamic programming method (minimizing the number of LUTs and latency).

QNAS [51] focuses on optimizing the parameters of a mixed-precision systolic-array-like architecture (the array size, buffer input/weight/output size) while searching the quantized neural architecture. It includes an EA-based hardware architecture search and a one-shot supernet-based quantized neural architecture search. First, a suite of neural architectures is sampled as a benchmark to find the hardware architecture that achieves the best performance on the benchmark. The hardware architecture is fixed, and the quantized neural architecture search (QNAS) is then performed to determine the neural architecture and quantization policy.

In YOSO [130], each candidate solution in the search space concatenates the DNN architecture and the ASIC accelerator configuration. For experiments with CIFAR-10, there are 40 hyperparameters for CNN and four accelerator parameters that the RL controller generates. Hardware parameters are predicted using the Gaussian Process model to eliminate an ordinary time-consuming simulation.

In AutoDNN [14], each candidate CNN consists of several hardware-aware parameterizable cells called Bundles. By means of these cells, specialized software can map any candidate CNN generated by NAS to an FPGA accelerator based on a fine-grained tile-based pipeline architecture whose components are pre-designed and stored in a component library. Latency and resources are estimated and used back in the NAS algorithm. The application is an object detection task targeting a PYNQ-Z1 embedded FPGA. Two design problems are solved simultaneously: the bottom-up CNN model exploration, and the top-down FPGA accelerator generation.

In Codesign-NAS [139], RL controller selects a CNN architecture from a CNN search space and a hardware architecture from an accelerator design space. Both are sent to the evaluator that implements the CNN on the proposed accelerator to find accuracy and efficiency metrics, such as

latency, area, and power (based on pre-computed models). The authors enumerated 4 billion model-accelerator pairs to study the Pareto-front in a representative co-design search space. They proposed three different search strategies to navigate the co-design search space under one or two constraints. The CNN search space is based on NASBench [174] and the accelerator design space utilizes CHaiDNN — a library for the acceleration of CNNs on System-on chip FPGAs.

EDD [48] is the first co-exploration approach utilizing differentiable problem formulation inspired by DARTS. DNN hyperparameters and parameters of a simplified hardware platform (i.e., the parallel and tiling factor of an FPGA accelerator template) are integrated into one solution space so that gradient descent algorithm can be applied to find accurate and hardware friendly CNN implementations. Parallel factors describe parallelism, indicating how many multiplications can be done concurrently. A limitation of this method is a simplified evaluation of hardware parameters.

Another framework based on differentiable neural architecture search, DANCE [170], was developed with Eyeriss as the backbone hardware platform. An RL controller generates parameters describing the CNN architecture (based on ProxylessNAS) as well as the hardware parameters such as the number of PEs, buffer size, and dataflow pattern. At the heart of DANCE is the modeling of the accelerator evaluation software using a neural network (the evaluator) that can be used as a differentiable loss function. DANCE thus introduces a novel differentiable evaluator, which takes the architecture parameters from the RL controller, searches for the optimal hardware accelerator design, and evaluates its cost metrics. The evaluator is a pre-trained neural network frozen during search and used only to connect the architecture to the hardware cost metrics.

HotNAS [49] works in two steps: (1) it uses Monte Carlo test to select several backbone architectures from a model ZOO of pre-trained models (the so-called hot models) that meet a latency constraint; (2) an RNN-based reinforcement learning optimizer tunes hyperparameters of neural architecture and hardware design simultaneously. After setting the parameters of the FPGA accelerator, latency can be estimated using a simple latency model. HotNAS combines three compression techniques (pattern pruning, channel pruning, and quantization) with the neural architecture search (filter expansion) and hardware optimization.

Liang *et al.* [136] deals with FPGA accelerators of CNNs in which irregular connections in the sparse convolutional layers have to effectively be handled. To this end, a weight-oriented dataflow is proposed that exploits element-matrix multiplication as the critical operation. The corresponding accelerator features a tile lookup table and a channel multiplexer. To connect this accelerator with NAS, an analytical model is developed to estimate the latency and resources in the FPGA. NAS searches hardware design parameters (parallelization and buffering factors) and possible CNN models under resource constraints. The search space is developed using MobileNet's inverted residual block as the basic building block of the supernet.

Focusing on Google's EdgeTPU, NAHAS [129] performs a joint search in the space of CNN architectures and hardware accelerator configurations, where hardware resources and latency are constraints. The architecture search space is based on a new fused inverted bottleneck layer with tunable parameters. The accelerator search space is defined by seven parameters (e.g., the PE array size, the number of SIMD units, register file capacity). An in-house simulator is used to estimate latency and other hardware-related parameters.

NAAS [8] holistically searches the neural network architecture, accelerator architecture, and unlike other methods (e.g., [13], [73]), compiler mapping. The accelerator search space is defined by the number of processing elements, local memory size, global buffer size, memory bandwidth, and connectivity parameters. NAAS employs EA to optimize these parameters as well as the compiler mapping (the execution order and the tiling size). It introduces a special encoding, called importance-based encoding, for the accelerator space and the compiler mapping strategies to avoid enumerating all possible situations and representing them by indexes. First, NAAS generates a pool of accelerator candidates. For each accelerator candidate, a network architecture is sampled from a pre-trained OFA network [7] that satisfies the pre-defined accuracy requirement. Since each subnet of OFA network is well trained, the accuracy evaluation is fast. Finally, the compiler mapping strategy is sought for the network candidate on the corresponding accelerator candidate. A comparison of CNNs generated by NAAS and QNAS for an ASIC accelerator will be presented in Section VIII.

VII. OTHER APPROACHES IN NAS AND HARDWARE CO-OPTIMIZATION

This section is devoted to NAS working with specialized hardware, which includes multiple accelerators and unconventional accelerators. Relevant NAS methods are given as a part of Table 8.

A. MULTIPLE NETWORKS AND MULTIPLE ACCELERATORS

In this category, three application scenarios are considered: (i) a single CNN is executed on several pipelined accelerators to maximize the performance, (ii) a single CNN is executed on one accelerator, but the CNN is optimized for a group of accelerators to ensure good portability, and (iii) m CNNs are optimized for m tasks executed on k sub-accelerators available on a hybrid accelerator.

(i) Since the timing performance on a single FPGA is limited by its constrained resources, multiple FPGAs are often organized in a pipelined fashion to provide high throughput for image and video processing applications. This problem was addressed by FNAS [172], and later HWSW-CoExp [13] which supports multiple FPGAs to implement one CNN. The design problem is formulated as follows. Given a dataset, a pool of FPGAs, and a throughput specification, the objective of to find a CNN (parameters of all layers, the partition

of the layer set and assignment of pipelined stages to the set of FPGAs) such that the accuracy of the resulting network is maximized, the pipeline FPGA system can meet the required throughput, and the average utilization of all FPGAs is maximized. The co-exploration algorithm iteratively performs two actions: fast and slow exploration. In the fast exploration, a CNN architecture is predicted for which the design space is explored to generate a pipelined FPGA system meeting the throughput requirement. There is the same number of RNN controllers as the number of pipeline stages. This level explores the hardware design space without training child networks. In the slow exploration, the child network obtained from the previous step is trained. After that, a reward based on both the yielded accuracy and pipeline efficiency is generated, which is used to update the RNN controller.

(ii) Multi-HW [128] proposes another scenario – multi-hardware models, where a single CNN is optimized for multiple hardware but executed on one of them. The challenge is that hardware platforms differ in many aspects, including their data flows, supported operations, and latencies. A multi-hardware search space is defined as a set of CNN architectures that belongs to the intersection of supported architectures of hardware platforms. The search space is constructed with modified MobileNetV3 architecture as the template, and the search method is based on TuNAS [80]. To accelerate the evaluation step, latency is obtained from pre-trained linear cost models for target hardware. The reward function for the RL controller considers the worst and average latency across all the target hardware platforms. It was reported that multi-hardware models could provide state-of-the-art performance across multiple hardware in both average and worse case scenarios.

(iii) In the context of multitask workflow, which is typical for some ASIC accelerators on edge, the goal is to optimize m CNNs, each of them executing a different task, using k sub-accelerators. These sub-accelerators differ in the dataflow style, and they are connected using NoC. In ASICNAS [73], the RL controller simultaneously generates hyperparameters of CNNs (m segments) together with the parameters of hardware resource allocation for different accelerator templates (k segments). CNNs are specified by CNN type and hyperparameters, while the accelerators are specified by the accelerator type, the number of PEs, and bandwidth. The goal is to map network layers to a pool of available sub-accelerators and determine their execution orders on each sub-accelerator. Mapping and scheduling are solved by ILP combined with a heuristic approach. Circuit parameters are estimated using MAESTRO and serve as feedback to the RL controller.

B. UNCONVENTIONAL HARDWARE

Emerging technologies are investigated to find solutions that can improve the critical parameters of computer systems, in particular, performance, storage capacity, and energy efficiency. Some parts of CNN accelerators were implemented using such technologies. Recently, these accelerators were connected with NAS algorithms to find best-performing CNN-accelerator pairs. A typical feature of these methods is that they employ very specific simulators of the underlying hardware.

For example, PABO [173] uses NAS connected with a memristive crossbar-based CNN accelerator, where the CNN is mapped across the on-chip crossbar storage spatially. Note that the memristive devices have a high storage density, but the write cost is relatively high. PABO was later extended, and its efficiency was demonstrated on more benchmarks in [175]. NAS4RRAM is a NAS method for optimizing CNNs and Resistive Random Access Memory (RRAM)-based accelerators [132]. NACIM [50] jointly explores device, circuit, and architecture design space and also takes device variation into account to find the most robust neural architectures, coupled with the most efficient hardware design for an in-memory computing ASIC. The joint search space involves decision parameters determining the neural architecture, quantization, data flow, circuit design strategy, and low-level device selection (ReRAM, FeFET, STTMRAM).

VIII. EVALUATION OF NAS METHODS

NAS methods are evaluated according to the quality of produced DNNs and the resources needed to generate them. Note that the NAS methods are multi-objective and have to be compared under all relevant objectives. Hence, fair benchmarking of an extensive collection of NAS methods (particularly the hardware-aware NAS methods) remains an open research problem. The difficulty is that too many aspects have to be considered during the comparison, and their deep cross-analysis is expensive to perform. The most relevant factors that have to be considered are:

- stochastic nature of search algorithms whose performance depends on many parameters and algorithmic settings;
- (multiple) objectives to be optimized;
- constraints handling;
- quantization options in DNNs;
- the computing time devoted to the search, training, and post-optimization;
- available computational resources;
- architecture and parameters of target hardware in which the resulting CNNs have to be deployed;
- the quality of estimation methods used for accuracy, latency, energy, and other objectives;
- hardware synthesis and optimization algorithms;
- re-using of pre-designed networks and other knowledge;
- implementation aspects involving the quality of software and hardware libraries used, compilers, and parallelization strategies.

Using the data available in the literature, we can only compare those NAS methods whose evaluation was conducted under comparable conditions and for the same target hardware. Fig. 14 compares CNNs obtained using NAS methods that consider the so-called *mobile setting*

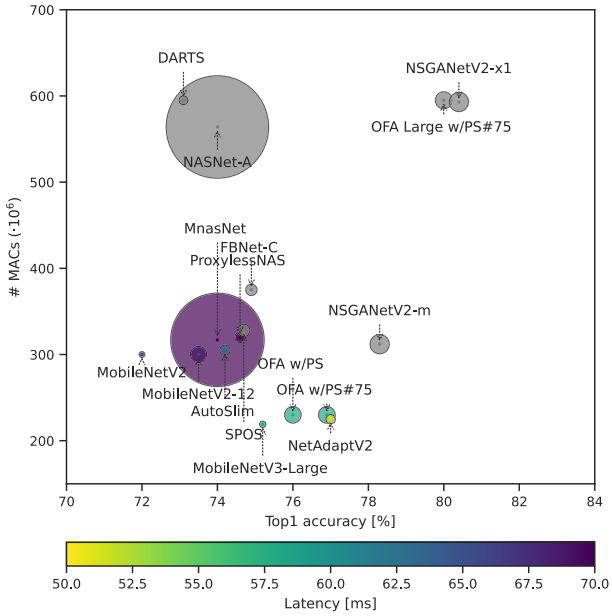

FIGURE 14. The top-1 accuracy, the number of MACs, and latency on Pixel 1 phone for CNNs obtained using selected NAS methods. An unknown latency is depicted using a grey color. The circle's area is proportional to the total design time (on a scale from 150 to 40 000 GPU hours).

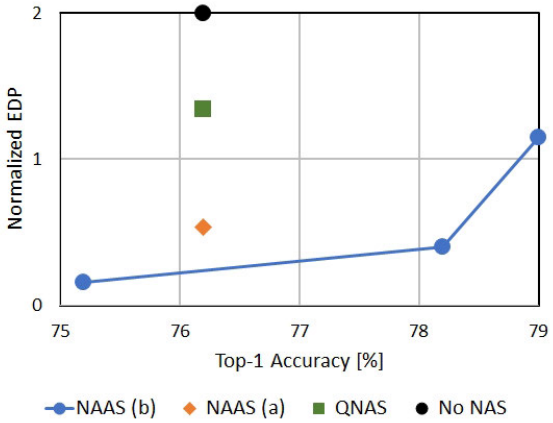

FIGURE 15. Normalized EDP and top-1 accuracy (on ImageNet) for CNNs running on an ASIC that were obtained by NAS methods (according to [8]): NAAS co-optimizing HW, compiler mapping and NN architecture (blue); NAAS co-optimizing HW and compiler mapping (orange); QNAS (green); No NAS conducted (black).

(up to $600{\cdot}10^6$ MACs per CNN inference). The criteria are the top-1 accuracy on ImageNet, the number of MACs, latency on Pixel 1 phone, and the total design time of the NAS method. As these data are available only for a few methods (OFA [7], ProxylessNAS [11], MNASNet [12], MobileNetV2 [32], MobileNetV3 [33], AutoSlim [176], and NetAdaptV2 [166]), we also included NSGANetV2 [6], DARTS [108], SPOS [163], FBNet [105], and NASNet [82] for which the latency on Pixel1 is not reported, but the design time is measured using the same methodology. Note that NASNet and DARTS do not optimize for latency. The total design time (in GPU hours) covers the search, training, and supernet training (for one deployment). It rangers from 150 (MobileNetV2) to 48 000 (NASNet) GPU hours. For other deployments (e.g., for another hardware), the design cost can partly be amortized. With a latency of 51 ms, the CNN created by NetAdaptV2 is the fastest network on Pixel 1 phone; its top-1 accuracy is high (77%), and the design time is less than 400 GPU hours. Under the mobile setting, NAS methods such as OFA and NSGANetV2 provide better trade-offs than human-crafted CNNs reported in Table 3. Fig. 14 also illustrates that if more resources are available, these methods can provide CNNs showing higher accuracy.

Another example deals with NAS utilizing hardware co-design. Fig. 15 shows the impact of using various NAS approaches in optimizing the accuracy and EDP of ImageNet classifiers based on ResNet-50 and implemented with resources similar to Eyeriss. The original implementation (black point) of ResNet-50 (no NAS employed) is improved by QNAS [51] (green point), which searches the network architecture and the accelerator sizing. Additional improvement is provided by NAAS performing the hardware and compiler mapping co-search (orange point). The best trade-offs are reported for NAAS utilizing the hardware, compiler mapping, and CNN architecture co-search (blue points). These results (adopted from [8]) demonstrate that exploiting more design spaces can lead to better CNN implementations.

In order to introduce a setup enabling designers to compare NAS methods and obtain reproducible results, almost half a million fully-trained CNNs sampled from a well defined compact search space were included in public data sets such as NAS-Bench-101 [174] and NAS-Bench-201 [177]. They can be used to compare new search algorithms intended for NAS methods that utilize the same search space. Instead of training each candidate CNN, the accuracy can be obtained by querying the pre-computed data set.

Considering the hardware-aware NAS methods, HW-NAS-Bench was introduced in 2021 [121]. In addition to the accuracy, it contains estimated hardware performance (e.g., energy cost and latency) of all the networks in the search spaces of both NAS-Bench-201 and FBNet, for six hardware platforms, including commercial edge devices FPGAs, and ASICs. For example, the HW-NAS-Bench provides the test accuracy vs. hardware cost of all the architectures in NAS-Bench-201 considering the ImageNet16-120 data set. It thus enabled the identification of the optimal CNN architecture-hardware pairs under a clearly defined setup. We expect other detailed studies to appear shortly, dealing with large-scale comparisons of multi-objective NAS methods. They should reveal the most suitable search algorithms and provide hints for building even better NAS methods. They should also focus on analyzing the relation between the quality or resulting CNNs/accelerators and the design time (or CO_2 emissions) needed for obtaining them [76].

IX. CONCLUDING REMARKS

We surveyed the key elements of recent NAS methods that – to various extents – consider hardware implementation of the resulting CNN. We classified these NAS methods into three major classes: single-objective NAS (no hardware is considered), hardware-aware NAS, and NAS with hardware co-optimization. Within each class we further categorized each method according several criteria as shown in Table 6, 7, and 8. We also provided additional details about selected methods that are essential in each class.

We showed that NAS methods improve design productivity and enable the designer to automatically obtain competitive CNNs for various hardware platforms and data sets. The original NAS approach [3] was significantly accelerated by using pre-trained supernets, adopting surrogate models, and incorporating the differentiable architecture search. Introducing the hardware search space has led to more efficient implementations of CNNs on particular hardware platforms. However, several search algorithms working in the space of weights, neural architectures, and hardware configurations have to be coordinated, making the entire method complicated. We proposed the first classification of NAS methods utilizing the hardware co-design.

As (hardware-aware) NAS methods are multi-objective, their fair assessment consists of evaluating multiple parameters of resulting implementations of NNs and the design cost (time). It thus leads to an expensive construction and comparison of multidimensional Pareto fronts (one Pareto front for one NAS method), which is often hard to perform because of incomplete information about some NAS methods. To support a fair benchmarking methodology and accelerate the development of new NAS methods, the open-source data sets containing many pre-trained and evaluated CNNs from a well-defined search space were introduced in the literature.

We can conclude that NAS methods have significantly been improved (in terms of search performance and the quality of resulting DNNs) since their first utilization by the ML community in 2016. Because hardware acceleration of DNNs is crucial for many application domains and DNNs are frequently applied in entirely new contexts, the importance of fully automated hardware-aware NAS as well as the NAS utilizing hardware co-design will grow in the following years. A barrier potentially slowing down their further expansion is the ever-increasing pressure to reduce the enormous energy requirements (and CO_2 emissions) of ML methods [76].

We pointed out in Section VII-B that DNNs are now implemented using emerging technologies. In the future, they can be deployed on more exotic platforms (such as nanoparticle networks configured using evolutionary algorithms [178]) that could provide richer and deeper interaction of machine learning and configurable physical *materio* and lead to more compact and energy-efficient solutions.

ACKNOWLEDGMENT

The author would like to thank Dr. Vojtěch Mrázek for his help with creating some of the figures.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," in *Proc. Int. Conf. Learn. Represent.*, 2021, pp. 1–22. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy
[3] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
[4] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, pp. 55:1-55:21, Mar. 2019.
[5] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," 2019, *arXiv:1905.01392*.
[6] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "NSGANetV2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *Computer Vision*. Cham, Switzerland: Springer, 2020, pp. 35–51.
[7] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–15.
[8] Y. Lin, M. Yang, and S. Han, "NAAS: Neural accelerator architecture search," in *Proc. 58th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 1–7.
[9] 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.
[10] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," *Neural Comput. Appl.*, vol. 32, no. 4, pp. 1109–1139, Feb. 2020.
[11] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–13.
[12] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2815–2823.
[13] W. Jiang, L. Yang, E. Sha, Q. Zhuge, S. Gu, Y. Shi, and J. Hu, "Hardware/software co-exploration of neural architectures," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4805–4815, Dec. 2020.
[14] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-M. Hwu, and D. Chen, "FPGA/DNN Co-Design: An efficient design methodology for 1ot intelligence on the edge," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.
[15] Y. Zhang, Y. Fu, W. Jiang, C. Li, H. You, M. Li, V. Chandra, and Y. Lin, "DNA: Differentiable network-accelerator co-search," 2020, *arXiv:2010.14778*.
[16] Y.-Q. Hu and Y. Yu, "A technical view on neural architecture search," *Int. J. Mach. Learn. Cybern.*, vol. 11, no. 4, pp. 795–811, Apr. 2020.
[17] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–34, May 2021.
[18] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.
[19] M. A. Elaziz, A. Dahou, L. Abualigah, L. Yu, M. Alshinwan, A. M. Khasawneh, and S. Lu, "Advanced metaheuristic optimization techniques in applications of deep neural networks: A review," *Neural Comput. Appl.*, vol. 2021, pp. 1–21, Apr. 2021.
[20] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, pp. 24–35, Jan. 2019.
[21] Y. Jaafra, J. L. Laurent, A. Deruyver, and M. S. Naceur, "Reinforcement learning for neural architecture search: A review," *Image Vis. Comput.*, vol. 89, pp. 57–66, Sep. 2019.
[22] E.-G. Talbi, "Automated design of deep neural networks: A survey and unified taxonomy," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–37, Apr. 2021.
[23] X. Zhang, W. Jiang, Y. Shi, and J. Hu, "When neural architecture search meets hardware implementation: From hardware awareness to co-design," in *Proc. IEEE Comput. SoC. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 25–30.

[24] W. J. Gross, B. H. Meyer, and A. Ardakani, "Hardware-aware design for edge intelligence," *IEEE Open J. Circuits Syst.*, vol. 2, pp. 113–127, 2021.
[25] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "A comprehensive survey on hardware-aware neural architecture search," 2021, *arXiv:2101.09336*.
[26] D. Deng and M. Lindauer. (2021). *Literature on Neural Architecture Search*. [Online]. Available: https://www.automl.org/
[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, 2012, pp. 1097–1105.
[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[29] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 2018.
[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014, *arXiv:1409.4842*.
[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.
[32] 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.
[33] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.
[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[35] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6848–6856.
[36] M. Abadi. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: https://www.tensorflow.org/
[37] A. Paszke, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. NeurIPS*, 2019, pp. 8024–8035.
[38] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.
[39] Y. LeCun, C. Cortes, and C. Burges, *MNIST Handwritten Digit Database*. Atlanta, GA, USA: ATT Labs, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist
[40] 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. Adv. Neural Inf. Process. Syst.*, 2011, pp. 1–9.
[41] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Comput. SoC. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2, Jun. 2004, pp. 97–104.
[42] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding," in *Proc. 4th Int. Conf. Learn. Represent.*, 2016, pp. 1–14.
[43] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," 2016, *arXiv:1602.02830*.
[44] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," 2016, *arXiv:1612.01064*.
[45] X. Sun, N. Wang, C.-Y. Chen, J. Choi, M. Kang, and A. Agarwal, "Efficient AI system design with cross-layer approximate computing," *Proc. IEEE*, vol. 108, no. 12, pp. 2232–2250, Dec. 2020.
[46] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.
[47] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," 2019, *arXiv:1911.00105*.
[48] Y. Li, C. Hao, X. Zhang, X. Liu, Y. Chen, J. Xiong, W.-M. Hwu, and D. Chen, "EDD: Efficient differentiable DNN architecture and implementation co-search for embedded AI solutions," in *Proc. 57th ACM/EDAC/IEEE Design Autom. Conf.*, Jul. 2020, pp. 1–6.
[49] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 4154–4165, Nov. 2020.
[50] W. Jiang, Q. Lou, Z. Yan, L. Yang, J. Hu, X. S. Hu, and Y. Shi, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 595–605, Apr. 2021.
[51] Y. Lin, D. Hafdi, K. Wang, Z. Liu, and S. Han, "Neural-hardware architecture search," in *Proc. 33rd Conf. Neural Inf. Process. Syst.*, 2019, pp. 1–5.
[52] V. Sze, Y. Chen, T. Yang, and J. S. Emer, *Efficient Processing of Deep Neural Networks*, (Synthesis Lectures on Computer Architecture). San Rafael, CA, USA: Morgan & Claypool, 2020.
[53] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, p. 113, Jul. 2020.
[54] M. Shafique, M. Naseer, T. Theocharides, C. Kyrkou, O. Mutlu, L. Orosa, and J. Choi, "Robust machine learning systems: Challenges, current trends, perspectives, and the road ahead," *IEEE Des. Test.*, vol. 37, no. 2, pp. 30–57, Apr. 2020.
[55] A. Ardakani, C. Condo, and W. J. Gross, "Fast and efficient convolutional accelerator for edge computing," *IEEE Trans. Comput.*, vol. 69, no. 1, pp. 138–152, Jan. 2020.
[56] Y. H. Chen, T. Krishna, 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, Jan. 2017.
[57] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Commun. ACM*, vol. 61, no. 9, pp. 50–59, Aug. 2018.
[58] (2021). *Intel Movidius Vision Processing Units (VPUs)*. [Online]. Available: https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu.html
[59] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, and Y. Nakamura, "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.
[60] (2021). *NVIDIA Deep Learning Accelerator (NVDLA)*. [Online]. Available: http://nvdla.org
[61] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A machine-learning supercomputer," in *Proc. 47th Annu. IEEE/ACM Int. Symp. Microarchitecture*, Dec. 2014, pp. 609–622.
[62] L. Cavigelli and L. Benini, "Origami: A 803-GOp/s/W convolutional network accelerator," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 11, pp. 2461–2475, Nov. 2017.
[63] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. K. Cheung, and A. andG Constantinides, "Deep neural network approximation for custom hardware: Where we've been, where we're going," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–39, 2019.
[64] L. Lai, N. Suda, and V. Chandra, "CMSIS-NN: Efficient neural network kernels for arm Cortex-M CPUs," 2018, *arXiv:1801.06601*.
[65] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "PULP-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Phil. Trans. R. SoC. A.*, vol. 378, no. 2164, 2019, Art. no. 20190155.
[66] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "XpulpNN: Accelerating quantized neural networks on RISC-V processors through ISA extensions," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 186–191.
[67] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "How to evaluate deep neural network processors: TOPS/W (Alone) considered harmful," *IEEE Solid StateCircuits Mag.*, vol. 12, no. 3, pp. 28–41, Jan. 2020.
[68] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W.-M. Hwu, and D. Chen, "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[69] (2021). *Intel-Optimized Math Library for Numerical Computing*. [Online]. Available: https://software.intel.com/en-us/mkl
[70] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "CuDNN: Efficient primitives for deep learning," 2014, *arXiv:1410.0759*.
[71] (2021). *MAESTRO: An Open-Source Infrastructure for Modeling Dataflows Within Deep Learning Accelerators*. [Online]. Available: http://maestro.ece.gatech.edu/
[72] S.-C. Kao, G. Jeong, and T. Krishna, "ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2020, pp. 622–636.
[73] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous ASIC accelerator designs targeting multiple tasks," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[74] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, 2016.
[75] S. S. Sarwar, G. Srinivasan, B. Han, P. Wijesinghe, A. Jaiswal, P. Panda, A. Raghunathan, and K. Roy, "Energy efficient neural computing: A study of cross-layer approximations," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 796–809, Dec. 2018.
[76] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," in *Proc. 10th AAAI Symp. Educ. Adv. Artif. Intell.*, 2020, pp. 13693–13696.
[77] J. Bracken and J. T. Mcgill, "Mathematical programs with optimization problems in the constraints," *Oper. Res.*, vol. 21, no. 1, pp. 37–44, 1973.
[78] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1388–1397.
[79] Y. Guo, Y. Chen, Y. Zheng, Q. Chen, P. Zhao, J. Chen, J. Huang, and M. Tan, "Pareto-Frontier-aware neural architecture generation for diverse budgets," 2021, *arXiv:2103.00219*.
[80] G. Bender, H. Liu, B. Chen, G. Chu, S. Cheng, P.-J. Kindermans, and Q. V. Le, "Can weight sharing outperform random architecture search an investigation with tunas," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 14311–14320.
[81] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed precision quantization of convnets via differentiable neural architecture search," 2018, *arXiv:1812.00090*.
[82] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Dec. 2018, pp. 8697–8710.
[83] X. Xia and W. Ding, "HNAS: Hierarchical neural architecture search on mobile devices," 2020, *arXiv:2005.07564*.
[84] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," 2017, *arXiv:1703.00548*.
[85] R. Ru, P. Esperança, and F. M. Carlucci, "Neural architecture generator optimization," in *Advances in Neural Information Processing Systems*, vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Red Hook, NY, USA: Curran Associates, 2020, pp. 12057–12069.
[86] N. Nayman, Y. Aflalo, A. Noy, and L. Zelnik-Manor, "HardCoRe-NAS: Hard constrained diffeRentiable neural architecture search," 2021, *arXiv:2102.11646*.
[87] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2018, pp. 2423–2432.
[88] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009.
[89] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.
[90] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.
[91] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, Mar. 2008.
[92] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[93] E. Real, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Sydney, NSW, Australia, vol. 70, Aug. 2017, pp. 2902–2911.
[94] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genetic Evol. Comput. Conf.*, Jul. 2019, pp. 419–427.
[95] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using Cartesian genetic programming," *Evol. Comput.*, vol. 28, no. 1, pp. 141–163, Mar. 2020.
[96] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, Jul. 2019, pp. 4780–4789.
[97] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Computer Vision*. Cham, Switzerland: Springer, 2018, pp. 19–35.
[98] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian, "Sample-efficient neural architecture search by learning actions for Monte Carlo tree search," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Apr. 7, 2021, doi: 10.1109/TPAMI.2021.3071343.
[99] I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, "SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers," in *Advances in Neural Information Processing Systems*, vol. 32. Red Hook, NY, USA: Curran Associates, 2019, pp. 1–26.
[100] L. Cai, A.-M. Barneche, A. Herbout, C. S. Foo, J. Lin, V. R. Chandrasekhar, and M. andM Sabry Aly, "TEA-DNN: The quest for time-energy-accuracy co-optimized deep neural networks," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Dec. 2019, pp. 1–6.
[101] Z. Yang, S. Zhang, R. Li, C. Li, M. Wang, D. Wang, and M. Zhang, "Efficient resource-aware convolutional neural architecture search for edge computing with Pareto-Bayesian optimization," *Sensors*, vol. 21, no. 2, p. 444, Jan. 2021.
[102] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on ImageNet in four GPU hours: A theoretically inspired perspective," 2021, *arXiv:2102.11535*.
[103] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: Multi-objective hyper-parameter optimization," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Dec. 2016, pp. 1–8.
[104] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 4092–4101.
[105] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "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.
[106] S.-Y. Huang and W.-T. Chu, "PONAS: Progressive one-shot neural architecture search for very efficient deployment," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–9.
[107] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path NAS: Designing hardware-efficient ConvNets in less than 4 hours," in *Machine Learning and Knowledge Discovery in Databases*. Cham, Switzerland: Springer, 2020, pp. 481–497.
[108] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. 7th Int. Conf. Learn. Represent.*, New Orleans, LA, USA, May 2019, pp. 1–13.
[109] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.
[110] Y. Xu, L. Xie, W. Dai, X. Zhang, X. Chen, G.-J. Qi, H. Xiong, and Q. Tian, "Partially-connected neural architecture search for reduced computational redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2953–2970, Sep. 2021.
[111] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive DARTS: Bridging the optimization gap for NAS in the wild," *Int. J. Comput. Vis.*, vol. 129, no. 3, pp. 638–655, Mar. 2021.
[112] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. 5th Int. Conf. Learn. Represent.*, 2017, pp. 1–18.
[113] M. Tan and Q. V. Le, "MixConv: Mixed depthwise convolutional kernels," in *Proc. 30th Brit. Mach. Vis. Conf.*, 2019, p. 74.
[114] Y. Chen, G. Meng, Q. Zhang, S. Xiang, C. Huang, L. Mu, and X. Wang, "RENAS: Reinforced evolutionary neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4782–4791.

[115] K. A. Laube and A. Zell, "ShuffleNASNets: Efficient CNN models through modified efficient neural architecture search," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–6.

[116] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[117] C. Cioflan and R. Timofte, "MS-RANAS: Multi-scale resource-aware neural architecture search," 2020, *arXiv:2009.13940*.

[118] T. Zhang, C. Lei, Z. Zhang, X.-B. Meng, and C. L. P. Chen, "AS-NAS: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning," *IEEE Trans. Evol. Comput.*, vol. 25, no. 5, pp. 830–841, Oct. 2021.

[119] Q. Guo, X.-J. Wu, J. Kittler, and Z. Feng, "Differentiable neural architecture learning for efficient neural network design," 2021, *arXiv:2103.02126*.

[120] M. Ferianc, H. Fan, and M. Rodrigues, "VINNAS: Variational inference-based neural network architecture search," 2020, *arXiv:2007.06103*.

[121] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin, "HW-NAS-Bench: Hardware-aware neural architecture search benchmark," in *Proc. 9th Int. Conf. Learn. Represent.*, 2021, pp. 1–18.

[122] C. A. C. Coello, S. G. Brambila, J. F. Gamboa, M. G. C. Tapia, and R. H. Gomez, "Evolutionary multiobjective optimization: Open research areas and some challenges lying ahead," *Complex Intell. Syst.*, vol. 2020, pp. 1–16, Jul. 2020.

[123] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Hoboken, NJ, USA: Wiley, 2009.

[124] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[125] J. Dong, A. Cheng, D. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware progressive search for pareto-optimal neural architectures," in *Proc. 15th Eur. Conf.*, vol. 11215. Cham, Switzerland: Springer, 2018, pp. 540–555.

[126] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "APQ: Joint search for network architecture, pruning and quantization policy," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2075–2084.

[127] X. Dai, Y. Jia, P. Vajda, M. Uyttendaele, N. K. Jha, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, and Y. Wu, "ChamNet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11390–11399.

[128] G. Chu, O. Arikan, G. Bender, W. Wang, A. Brighton, P.-J. Kindermans, H. Liu, B. Akin, S. Gupta, and A. Howard, "Discovering multi-hardware mobile models via architecture search," in *Proc. CVPR Workshop*, 2021, pp. 3022–3031.

[129] Y. Zhou, X. Dong, B. Akin, M. Tan, D. Peng, T. Meng, A. Yazdanbakhsh, D. Huang, R. Narayanaswami, and J. Laudon, "Rethinking co-design of neural architectures and hardware accelerators," 2021, *arXiv:2102.08619*.

[130] W. Chen, Y. Wang, S. Yang, C. Liu, and L. Zhang, "You only search once: A fast automation framework for single-stage DNN/Accelerator co-design," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1283–1286.

[131] L. L. Zhang, Y. Yang, Y. Jiang, W. Zhu, and Y. Liu, "Fast hardware-aware neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 2959–2967.

[132] Z. Yuan, J. Liu, X. Li, L. Yan, H. Chen, B. Wu, Y. Yang, and G. Sun, "NAS4RRAM: Neural network architecture search for inference on RRAM-based accelerators," *Sci. China Inf. Sci.*, vol. 64, no. 6, Jun. 2021, Art. no. 160407.

[133] S. Gupta and B. Akin, "Accelerator-aware neural network design using AutoML," in *Proc. Intell. Workshop Conf.*, 2020, pp. 1–5.

[134] S. Kim, H. Kwon, E. Kwon, Y. Choi, T.-H. Oh, and S. Kang, "MDARTS: Multi-objective differentiable neural architecture search," in *Proc. DATE*, 2021, pp. 1–6.

[135] A. Marchisio, A. Massa, V. Mrazek, B. Bussolino, M. Martina, and M. Shafique, "NASCaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks," in *Proc. 39th Int. Conf. Computer-Aided Design*, Nov. 2020, pp. 1–9.

[136] Y. Liang, L. Lu, Y. Jin, J. Xie, R. Huang, J. Zhang, and W. Lin, "An efficient hardware design for accelerating sparse CNNs with NAS-based models," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Mar. 17, 2021, doi: 10.1109/TCAD.2021.3066563.

[137] X. Luo, D. Liu, S. Huai, and W. Liu, "HSCoNAS: Hardware-software co-design of efficient DNNs via neural architecture search," in *Proc. DATE*, 2021, pp. 1–4.

[138] H. Fan, M. Ferianc, S. Liu, Z. Que, X. Niu, and W. Luk, "Optimizing FPGA-based CNN accelerator using differentiable neural architecture search," in *Proc. IEEE 38th Int. Conf. Comput. Design (ICCD)*, Oct. 2020, pp. 465–468.

[139] M. S. Abdelfattah, L. 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. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

[140] Y. Hu, X. Wu, and R. He, "TF-NAS: Rethinking three search freedoms of latency-constrained differentiable neural architecture search," in *Computer Vision*. Cham, Switzerland: Springer, 2020, pp. 123–139.

[141] C. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "MicroNets: Neural network architectures for deploying TinyML applications on commodity microcontrollers," in *Proc. Mach. Learn. Syst.*, 2021, pp. 1–16.

[142] E. Liberis, U. Dudziak, and N. D. Lane, "μNAS: Constrained neural architecture search for microcontrollers," in *Proc. EuroMLSys*, 2021, pp. 70–79.

[143] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "MONAS: Multi-objective neural architecture search using reinforcement learning," 2018, *arXiv:1806.10332*.

[144] P. Colangelo, O. Segal, A. Speicher, and M. Margala, "Artificial neural network and accelerator co-design using evolutionary algorithms," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–8.

[145] Z. Yang and Q. Sun, "Efficient resource-aware neural architecture search with dynamic adaptive network sampling," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[146] P. Achararit, M. A. Hanif, R. V. W. Putra, M. Shafique, and Y. Hara-Azumi, "APNAS: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators," *IEEE Access*, vol. 8, pp. 165319–165334, 2020.

[147] J. Lee, D. Kang, and S. Ha, "S3NAS: Fast NPU-aware neural architecture search methodology," 2020, *arXiv:2009.02009*.

[148] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. 34th Conf. Neural Inf. Process. Syst.*, 2020, pp. 1–12.

[149] J. G. López, A. Agudo, and F. Moreno-Noguer, "E-DNAS: Differentiable neural architecture search for embedded systems," in *Proc. 25h Int. Conf. Pattern Recognit. (ICPR)*, 2021, pp. 4704–4711.

[150] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "HyperPower: Power and memory-constrained hyper-parameter optimization for neural networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 19–24.

[151] Y. Chen, G. Meng, Q. Zhang, X. Zhang, L. Song, S. Xiang, and C. Pan, "Joint neural architecture search and quantization," 2018, *arXiv:1811.09426*.

[152] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "PPP-Net: Platform-aware progressive search for Pareto-optimal neural architectures," in *Int. Conf. Learn. Represent.*, 2018, pp. 1–4.

[153] Y. Zhou, S. Ebrahimi, S. Ö. Arák, H. Yu, H. Liu, and G. Diamos, "Resource-efficient neural architect," 2018, *arXiv:1806.07912*.

[154] 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. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, Nov. 2019, pp. 1–7.

[155] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via Lamarckian evolution," in *Proc. 7th Int. Conf. Learn. Represent.*, 2019, pp. 1–23.

[156] T. Cassimon, S. Vanneste, S. Bosmans, S. Mercelis, and P. Hellinckx, "Designing resource-constrained neural networks using neural architecture search targeting embedded devices," *Internet Things*, vol. 12, Dec. 2020, Art. no. 100234.

[157] M. Loni, S. Sinaei, A. Zoljodi, M. Daneshtalab, and M. Sjödin, "DeepMaker: A multi-objective optimization framework for deep neural networks in embedded systems," *Microprocessors Microsyst.*, vol. 73, Mar. 2020, Art. no. 102989.
[158] Y. Jiang, X. Wang, and W. Zhu, "Hardware-aware transformable architecture search with efficient search space," in *Proc. IEEE Int. Conf. Multimedia Expo. (ICME)*, Jul. 2020, pp. 1–6.
[159] P. Liu, B. Wu, H. Ma, and M. Seok, "MemNAS: Memory-efficient neural architecture search with grow-trim learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2105–2113.
[160] B. Chen, G. Ghiasi, H. Liu, T.-Y. Lin, D. Kalenichenko, H. Adam, and Q. V. Le, "MnasFPN: Learning latency-aware pyramid architecture for object detection on mobile devices," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, p. 13.
[161] C. Wang, H. Wang, G. Feng, and F. Geng, "Multi-objective neural architecture search based on diverse structures and adaptive recommendation," 2020, *arXiv:2007.02749*.
[162] C. Schorn, T. Elsken, S. Vogel, A. Runge, A. Guntoro, and G. Ascheid, "Automated design of error-resilient and hardware-efficient deep neural networks," *Neural Comput. Appl.*, vol. 32, no. 24, pp 18327–18345, 2020.
[163] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Proc. ECCV*, 2020, pp. 544–560.
[164] B. Lyu, H. Yuan, L. Lu, and Y. Zhang, "Resource-constrained neural architecture search on edge devices," *IEEE Trans. Netw. Sci. Eng.*, early access, Jan. 26, 2021, doi: 10.1109/TNSE.2021.3054583.
[165] X. Zheng, R. Ji, Y. Chen, Q. Wang, B. Zhang, Q. Ye, J. Chen, F. Huang, and Y. Tian, "MIGO-NAS: Towards fast and generalizable neural architecture search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2936–2952, Sep. 2021.
[166] T.-J. Yang, Y.-L. Liao, and V. Sze, "NetAdaptV2: Efficient neural architecture search with fast super-network training and architecture optimization," in *Proc. CVPR*, 2021, pp. 2402–2411.
[167] B. S. Prabakaran, A. Akhtar, S. Rehman, O. Hasan, and M. Shafique, "BioNetExplorer: Architecture-space exploration of bio-signal processing deep neural networks for wearables," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 13251–13265, Sep. 2021.
[168] K. Wang, P. Xu, C.-M. Chen, S. Kumari, M. Shojafar, and M. Alazab, "Neural architecture search for robust networks in 6G-enabled massive IoT domain," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5332–5339, Apr. 2021.
[169] X. Wang, X. Wang, L. Jin, R. Lv, B. Dai, M. He, and T. Lv, "Evolutionary algorithm-based and network architecture search-enabled multiobjective traffic classification," *IEEE Access*, vol. 9, pp 52310–52325, 2021.
[170] K. Choi, D. Hong, H. Yoon, J. Yu, Y. Kim, and J. Lee, "DANCE: Differentiable accelerator/network co-exploration," in *Proc. DAC*, 2021, pp. 1–7.
[171] M. Pinos, V. Mrazek, and L. Sekanina, "Evolutionary neural architecture search supporting approximate multipliers," in *Proc. Eur. Conf. Genetic Program.*, vol. 12691. Cham, Switzerland: Springer, 2021, pp. 82–97.
[172] W. Jiang, X. Zhang, E. H.-M. Sha, L. Yang, Q. Zhuge, Y. Shi, and J. Hu, "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2019, pp. 1–6.
[173] M. Parsa, A. Ankit, A. Ziabari, and K. Roy, "PABO: Pseudo agent-based multi-objective Bayesian hyperparameter optimization for efficient neural accelerator design," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–8.
[174] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds., 2019, pp. 7105–7114.
[175] M. Parsa, J. P. Mitchell, C. D. Schuman, R. M. Patton, T. E. Potok, and K. Roy, "Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design," *Frontiers Neurosci.*, vol. 14, p. 667, Jul. 2020.
[176] J. Yu and T. Huang, "AutoSlim: Towards one-shot architecture search for channel numbers," 2019, *arXiv:1903.11728*.
[177] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–16.
[178] S. K. Bose, C. P. Lawrence, Z. Liu, K. S. Makarenko, R. M. J. Van Damme, H. J. Broersma, and W. G. Van Der Wiel, "Evolution of a designless nanoparticle network into reconfigurable Boolean logic," *Nature Nanotechnol.*, vol. 10, no. 12, pp. 1048–1052, 2015.

LUKAS SEKANINA (Senior Member, IEEE) received the Ing. and Ph.D. degrees from the Brno University of Technology, Brno, Czech Republic, in 1999 and 2002, respectively. He was a Visiting Professor with Pennsylvania State University, Erie, PA, USA, in 2001. He received the Fulbright Scholarship to work with the NASA Jet Propulsion Laboratory, Caltech, in 2004. He is currently a Full Professor and the Head of the Department of Computer Systems, Faculty of Information Technology, Brno University of Technology. He has coauthored over 200 papers, mainly on evolvable hardware, evolutionary computation, and approximate computing, and one patent. He served as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, from 2011 to 2014, the *Genetic Programming and Evolvable Machines* Journal, and the *International Journal of Innovative Computing and Applications*.